Integrating Relational Reinforcement Learning with Reasoning about Actions and Change[⋆]

Matthias Nickles

Department of Computer Science, Technical University of Munich
Boltzmannstr.3, D-85748 Garching, Germany, nickles@cs.tum.edu

1 Introduction and related works

We present an approach to the integration of Relational Reinforcement Learning (RRL) [1, 4, 10] with the Event Calculus (EC) [2], focussing on the combination of statistical learning and automated planning. Our framework allows for the formal specification of background knowledge, soft and hard policies, and rewards for the Reinforcement Learning (RL) task, and facilitates the constraining of the learning process by means of rich sub-policies and abductively generated plans. A new algorithm for relational instance-based regression is proposed which improves learning results in certain important use cases.
We are not aware of other approaches to the integration of RRL with the EC. However, several related approaches exist. In [15] a simulator of the environment is employed as a stochastic sample generator for RRL. However, this learning approach is goal-based and does not approximate a value function as in our case but learns policy representations. [5] proposes an approach where the performance of Relational Q-Learning is improved at runtime with plans generated using learned probabilistic models of the environment. [8, 13, 14] integrate RL with programs in the Golog action language (based on the Situation Calculus). [7, 6] provide logical formalism for the representation of MDPs. Various (rather remotely related) approaches combine planning with learning, e.g., [9]. [16] uses a temporal logic approach to temporally-extend rewards in RRL. [12] shows how EC programs can be learned from observations using Inductive Logic programming. In contrast, we do not learn EC programs but employ them for learning.

2 The Event Calculus

The EC and its cousin, the Situation Calculus, are popular, effective and easily implementable first-order calculi for reasoning about actions and their effects in dynamic systems. The EC defines a certain first-order language (with reified fluents) from which only the following is required to understand most of this paper.

$holdsAt(f, t)$ specifies that fluent f is true at time point t.
$happens(e, t)$ specifies that action (event) e occurs at time point t.

Time points are discrete and can refer to the past, the present, and the future.

A blocks world: As a running example, we make use of the well-known *blocks world* (BW), which is arguably the by far most frequently used domain in the context of RRL and other types of Statistical Relational Learning. The BW is also a classic planning domain. A BW is a relationally structured domain in which an agent observes and acts in a sort of grid with discrete positions. At each position there can be a block (named with a lower-case letter a, b, c, ...) or the $table$, or nothing. The fact that some block x is on top of some other block y at some point in time t is expressed with $holdsAt(on(x, y), t)$ in the EC. $holdsAt(on(x, table), t)$ means that block x is directly on the table at time step t. $holdsAt(clear(x), t)$ denotes that there is currently no block on top of block

⋆ **Preliminary version, extended abstract**

x. The agent acts in the BW by moving blocks using a "stacking" action, conditioned by certain pre- and post-conditions (e.g., both the moved block and the target of the move need to be clear in beforehand). A stacking action at time t which moves block x on top of block y is expressed with $happens(stack(x, y), t)$. If the action succeeded, subsequently $holdsAt(on(x, y), t + 1)$ holds.
For the purpose of this paper, the BW is considered to be fully observable for the learning agent. However, actions can have nondeterministic outcomes.

Reasoning with the EC: EC reasoning can take many concrete forms, including the use of plain Prolog. In this work, we assume an EC reasoner which computes a finite set of satisfying models of the agent's EC program (knowledge base). In our experiments, we have used Potassco (the Potsdam Answer Set Solving Collection)[1] for this purpose, i.e., a model is here in fact an answer set. For every time step $t_i, 0 \leq i \leq tmax$, each of the models consists of a number of ground facts, including the truth values of all fluents and which action happens or might happen in the future. The reasoner thus provides us with all or a subset of all alternative futures (possible worlds). In case of a deterministic domain, the models are contingent only wrt. the agent's actions. For nondeterministic domains, they may also comprise uncertain outcomes of actions.

Nondeterministic domains: We can use the EC to specify nondeterministic block worlds, which is of course particularly relevant in the context of RL. There are various approaches to nondeterminism using the EC. In the following example, we make the result of an agent's stacking action determined by the random outcome of a coin flip. In case of "heads", the stacking action fails and the block lands on the table. The coin flip is modeled using a so-called "determining fluent":

```
initiates(stack(X,Y),on(X,table),T) :- holdsAt(determFluent(heads), T).
initiates(stack(X,Y),on(X,Y),T) :- holdsAt(determFluent(tails), T).
```

Such nondeterministic rules indirectly induce a probability distribution over the truth values of fluents. E.g., given the rules above and the occurrence of action $stack(a, b)$ at time t, the probability that $holdsAt(on(a, b), t+1)$ is part of a satisfying model is 0.5.

Automated planning Using the EC, we can provide the learning agent with semantically rich first-order state descriptions and an exploration policy. While the specification of background knowledge and policies in the EC is straightforward, we need to say more about planning. Planning using the EC is typically achieved using abduction. The planning goal can be specified in the agent's knowledge base by means of a statement like $notnotholdsAt(goal, tmax)$ for some future time $tmax$. Also the initial state is specified, corresponding to time step 0. The EC reasoner computes a number of models which each comprises both the initial and the goal state, and a number of actions which lead from the former state to the latter state, i.e., a plan. A minimal plan can be found by an incremental decrease of $tmax$ until the knowledge base becomes unsatisfiable.

3 Relational Reinforcement Learning

RRL differs from ordinary RL in that it uses a relational representation for Markov states and actions, and thus allows for a natural representation of complex domains whose rich structural properties would otherwise be inaccessible to RL. Various approaches to RRL exist.

Definition 1 (Relational Reinforcement Learning) Let

- S be a set of states, represented in a relational format (e.g., logically),
- A be a set of actions, represented in a relational format (e.g., logically),
- $T : S \times A \times S \rightarrow [0; 1]$ be a stochastic state transition function,

[1] http://potassco.sourceforge.net/

– $R : S \times A \rightarrow \mathbb{R}$ be a real-valued reward function.

The agent's goal is to learn an action policy $\pi : S \rightarrow A \in [0; 1]$ that maximizes the discounted return $R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$ from any time step t. This return is the cumulative reward obtained in the future, starting from state s_t. Future rewards are weakened by some discount factor $\gamma^k \in [0; 1]$. R is approximated using a Q-value function $Q : S \times A \rightarrow R$ such that $Q(s, a) = \mathbb{E}(R_t | s_t = s, a_t = a)$.

4 RRL with a EC reasoner as a model generator

Markov states are associated now with time points in the EC-sense, and the state corresponding to a certain time point t comprises information about which fluents and actions are holding/happening at that time according to the agent's dynamically updated knowledge base KB. When the agent has performed an action or observed a state, KB is updated accordingly. In the following, the set of truth values of fluents at time t is denoted as $Fluents_t(KB)$, the set of possible actions at the subsequent time $t + 1$ is denoted as $dact(KB, t + 1)$. This approach provides the following key benefits:

Seamless integration with planning An expert can provide logically represented goals for planning and logically represented goals or rewards for learning within the same specification. The EC-reasoner can also be employed for the deductive or abductive computation of examples for instance-based learning.

Provision of possible "futures" The EC-reasoner computes the set of logically possible actions in the subsequent time step as well as deterministic and nondeterministic state transitions. If the maximum set of models provided by the reasoner is incomplete (and thus faster to compute), the EC reasoner approaches a stochastic state/action sample generator.

Convenient formal domain description and formal specification of sub-policies, agent decision making and rewards, including (non-Markovian) rewards spawning multiple time steps [16], as outlined below.

Apart from the straightforward use for the domain specification (e.g., modeling the action pre- and post-conditions for the blocks word), another important use case for our framework is the combined specification of sub-policies, planning goals and rewards (contingent on the truth values of fluents). A few very simple examples (required rules for BW domain specification and further background knowledge are omitted for lack of space):

```
holdsAt(reward(1),T) :- holdsAt(on(a,b),T), holdsAt(on(a,table),T-2).
holdsAt(reward(1),T) :- holdsAt(on(a,b),T), holdsAt(on(b,table),T-2).
holdsAt(reward(1),T) :- holdsAt(on(a,b),T), holdsAt(on(c,table),T-2).
holdsAt(reward(0),T) :- not holdsAt(on(a,b),T).
holdsAt(reward(0),T) :- not holdsAt(on(a,table),T-2), not holdsAt(on(b,table),T-2),
not holdsAt(on(c,table),T-2).
happens(stack(b,a),7) | happens(stack(b,table),7) | happens(stack(b,table),8).
not not happens(stack(a,b),20). A planning goal
holdsAt(actionWeight(100), T) :- happens(stack(b,c),T). A hard action constraint, cf. below
holdsAt(actionWeight(0), T) :- not happens(stack(b,c),T).
```

Definition 2 (RRL-EC) Let

– A, T, R as in standard RRL,
– $n \in \mathbb{N}$ be a finite temporal reasoning horizon,
– $KB = \{KB_i : 1 \leq i \leq n\}$ be the set of knowledge bases (EC programs) at time points $\leq n$. After each agent action and observation, an update of KB_i to KB_{i+1} takes place in order to include this action and observation. Note that in the EC, each KB_i spawns all time steps, not just time i.

- S be the set of possible states, consisting of sets of those fluents which hold at time t (minus special-purpose fluents such as $reward$ and $actionWeight$). For the current time step t a state is observable and can be computed from KB_t using function $Fluents : KB \times \mathbb{N} \rightarrow S$.
- $\delta : KB \times A \rightarrow KB$ be a knowledge base update function which updates the learning agent's knowledge base KB_t to its successor KB_{t+1} after a new event (action) has occurred.
- $\pi : KB \times S \times \mathbb{N} \rightarrow A$ be a Q/KB-optimal policy with

  $$\pi(KB_t, s_t, t) = argmax_{a \in dact(KB_t, t+1)}(Q(s_t, a) actionWeight(t+1, a)) \quad (1)$$

  using a Q-value function as specified above and a constraining function $dact : KB \times t \rightarrow 2^A$ which results in the set of actions which are logically possible (according to KB_t) at time point $t+1$ (i.e., one step in the future).
  $actionWeight(t, action)$ denotes that action a has a certain *weight* at time t.
  $actionWeight$ can be used to provide soft policies (i.e., default policies which are overridable where values of the respective state/action-pairs become sufficiently large). It is obtained at runtime from an eponymous fluent (see Algorithm 1). Instead of $argmax$, a Boltzmann-*softmax* could optionally be used, in order to foster exploration.

The following algorithms can be seen as instances of the incremental Temporal Difference RRL algorithms introduced in [11] and SARSA, but with the provision that the set of possible states, the set of actions possible in each state and the rewards are obtained deductively/abductively from an incrementally updated EC knowledge base. The first algorithm uses regression (Relational Instance-Based learning) to predict the values of unseen examples:

Algorithm 1 (RRL-RIB-TD-EC)

```
Require: State/action space (fragmentary), knowledge base $KB_1$, policy function $\pi$, action constraining function $dact$, state determination function $Fluents$, regression system for $Q_{RIB}$ (see below), discount factor $\gamma$
Ensure: Approximation of $Q_{RIB}$
  loop
    specify start state $s_t = Fluents(KB_1, 1), t = 0$
    $a \leftarrow \pi(KB_t, s_t, t)$
    repeat
      Perform action $a$,
      $KB_{t+1} \leftarrow \delta(KB_t, a)$,
      $s_{t+1} = Fluents(KB_{t+1}, t+1)$,
      Get reward $r :\Leftrightarrow KB_{t+1} \models holdsAt(Reward(r), t+1)$
      Get weights of possible subsequent actions:
          $actionWeight(t+1, action) = w_j$
          $:\Leftrightarrow KB_{t+1} \wedge happens(action, t+1) \models holdsAt(actionWeight(w_j), t+1)$
      $\bar{a} \leftarrow \pi(KB_{t+1}, s_{t+1}, t+1)$
      if $t \leq n$ then
        $Q_{RIB}(s_t, a) \leftarrow r + \gamma Q_{RIB}(s_{t+1}, \bar{a})$   (learn)
      else
        $Q_{RIB}(s_t, a) \leftarrow r$   (learn)
      end if
      $t \leftarrow t+1, a \leftarrow \bar{a}$
    until $t = n$
  end loop
```

This algorithm requires an instance-based regression mechanism Q_{RIB} which provides value predictions for learning examples which are not in memory yet. This could be one of the previously introduced RIB systems [4, 11], or the following new approach using a planning (pseudo-)distance $Q_{RIB-Plan}$. Like the RIB approach presented in [4], $Q_{RIB-Plan}$ is for unseen examples (s, a) calculated using a "relational" k-nearest-neighbor estimation as follows. $(\bar{s}, \bar{a})$ denotes examples whose values are already in memory:

$$Q_{RIB-Plan}(s, a) = \frac{\sum_{\bar{s},\bar{a}} \frac{Q_{RIB-Plan}(\bar{s},\bar{a})}{d_P((s,a),(\bar{s},\bar{a}))}}{\sum_{\bar{s},\bar{a}} \frac{1}{d_P((s,a),(\bar{s},\bar{a}))}} \qquad (2)$$

with $d_P((s, a), (\bar{s}, \bar{a})) =$
$argmin_{tgoal-t}(\models s \wedge happens(a, t{+}1) \wedge happens(a_2, t{+}2) \wedge ... \wedge \bar{s} \wedge happens(\bar{a}, tgoal))$, i.e., the path length of the shortest possible plan which leads from s updated with action a to $\bar{s}$ updated with action $\bar{a}$.
Clearly, d_P is in general not a real distance, if because we cannot guarantee for arbitrary domains that any given plan could be executed backwards (i.e., d_P is not symmetric).
Benefits compared to those relational distance metrics which have been used with RRL before is that $Q_{RIB-Plan}$ does neither require goals nor syntactic action inspection, and that it can immediately be used with other domains than BW without adaptation. It is also much simpler than kernel-based relational distances. However, a shortcoming of $Q_{RIB-Plan}$ is the relatively long time required for each distance calculation due to frequent costly invocations of the reasoner. We nevertheless obtained favorable experimental results using a small modification of $Q_{RIB-Plan}$ (cf. Section 5).

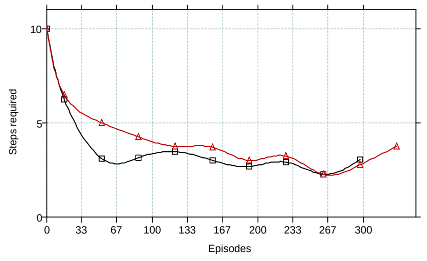
Algorithm 2 (RRL-TD-EC) is as RRL-RIB-TD-EC, but instead of regression, $Q_{RIB}(s_t, a)$ yields a constant default value (e.g., 0) in case of unseen learning examples.

Algorithms 3/4 (RIB/RRL-Q-EC) is as RRL-RIB-TD-EC respectively RRL-TD-EC, except that standard offline Q-learning with a variant of the Bellman equation which accounts for nondeterministic domains is being used. For lack of space, these algorithms are omitted here.

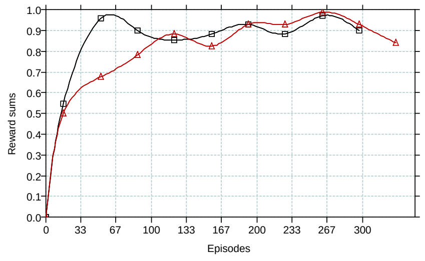
5 Empirical evaluation

Our learning framework is fully implemented. In the following, some experimental findings are reported. Further empirical results and a more detailed analysis can be found in the full version of this paper.

The black (square-decorated) curves in Figure 1 show the averaged performance of algorithm RRL-RIB-TD-EC and $Q_{RIB-Plan}$ as regression system for learning how to stack all blocks in a deterministic BW with five blocks on top of each other. Each episode starts with a new random state of the blocks world. Reward is given only for reaching the goal, which aborts the respective episode. In this example. the goal is only used to calculate rewards, not for regression or EC reasoning. While the matching/edit distance metrics used in [4] (and which is based on the general distance proposed in [3]) makes a compact and generalized value function possible, given a reduced inflow of learning examples, it is not reasonably applicable in our targeted setting (no goal use and no symbolic action examination in the regression computation, unlimited example inflow). We performed two trials with 300 episodes each. To minimize the number of costly invocations of the reasoner, plan distances are computed only in case the Q-value of the respective second learning example in memory ($(\bar{s}, \bar{a})$ in equation (2)) is outside of a certain insignificance interval (initially $[0.4, 0.8]$), which is enlarged by a small factor after each distance calculation. For this experiment, a reduced set of EC axioms has been used and the set of possible actions at each time step has been computed without the reasoner, just in order to reduce computation time. The red (triangle-decorated) curves show the results for the same setting but without regression. While from about episode 130 on both curves begin to converge, the regression mechanism shows a significance improvement of learning performance up to this point, due to effective provision of predictions in place of missing examples. We expect this effect to be in particular



(a) Number of time steps until reward



(b) Reward sums (moving average)

Fig. 1. RIB with planning distance

relevant for applications where it is important to see a learning success after a small number of episodes whereas the absolute time required for each episode is of minor importance (e.g., learning in human/computer interaction scenarios).

References

1. S. Dzeroski, L. De Raedt, H. Blockeel: Relational reinforcement learning. Procs ICML'98. Morgan Kaufmann, 1998.
2. R. Kowalski, M. Sergot: A Logic-Based Calculus of Events. In New Generation Computing 4: 6795, 1986.
3. J. Ramon, M. Bruynooghe: A polynomial time computable metric between point sets. Acta Informatica, 37:765780, 2001.
4. K. Driessens: Relational Reinforcement Learning. PhD thesis, Department of Computer Science, Katholieke Universiteit Leuven, 2004.
5. T. Croonenborghs, J. Ramon, M. Bruynooghe: Towards informed reinforcement learning. Procs. of the Workshop on Relational Reinforcement Learning at ICML'04, 2004.
6. K. Kersting, L. De Raedt: Logical Markov decision programs. Procs. IJCAI'03 Workshop on Learning Statistical Models of Relational Data, 2003.
7. C. Boutilier, R. Reiter, B. Price: Symbolic dynamic programming for First-order MDP's. Procs. IJCAI-01, Morgan Kaufmann Publishers, 2001.
8. I.A. Letia, D. Precup: Developing collaborative Golog agents by reinforcement learning. Procs. ICTAI'01. IEEE Computer Society, 2001.
9. D. Bryce: POND: The Partially-Observable and Non-Deterministic Planner. Notes on The 5th International Planning Competition at ICAPS'06, 2006.
10. M. Van Otterlo: A Survey of RL in Relational Domains, CTIT Technical Report Series, 2005.
11. C. Rodrigues, P. Gerard, C. Rouveirol: Relational TD Reinforcement Learning. Procs. EWRL'08, 2008
12. S. Moyle, S. Muggleton: Learning Programs in the Event Calculus. Lecture Notes in Computer Science, Springer 1997.
13. A. Finzi, T. Lukasiewicz: Adaptive multi-agent programming in GTGolog. Procs of the 29th Annual German Conference on Artificial Intelligence (KI 2006), 2006.
14. D. Beck, G. Lakemeyer: Reinforcement Learning for Golog Programs. Procs. Workshop on Relational Approaches to Knowledge Representation and Learning, 2009.
15. A. Fern, S. Yoon, R. Givan: Reinforcement Learning in Relational Domains: A Policy-Language Approach. In L. Getoor, B. Taskar, eds. Introduction to Statistical Relational Learning. MIT Press, 2007.
16. C. Gretton: Gradient-Based Relational Reinforcement-Learning of Temporally Extended Policies. International Conference on Automated Planning and Scheduling (ICAPS'07), 2007.