Noname manuscript No.
(will be inserted by the editor)

Interactive Relational Reinforcement Learning of Concept Semantics

Matthias Nickles · **Achim Rettinger**

the date of receipt and acceptance should be inserted later

Abstract We present a framework for the machine learning of denotational concept semantics using a simple form of symbolic interaction of machines with human users. The capability of software agents and robots to learn how to communicate verbally with human users would obviously be highly useful in several real-world applications, and our framework is meant to provide a further step towards this goal. Whereas the large majority of existing approaches to the machine learning of word sense and other language aspects focuses on learning using text corpora, our framework allows for the interactive learning of concepts in a dialog of human and agent, using an approach in the area of Relational Reinforcement Learning. Such an approach has a wide range of possible applications, e.g., the interactive acquisition of semantic categories for the Semantic Web, Human-Computer Interaction, (interactive) Information Retrieval, and Natural Language Processing.

Keywords: *Reinforcement Learning, Concept Learning, Symbol Grounding, Statistical Relational Learning, Interactive Learning, Meaning Disambiguation*

1 Introduction

The large majority of existing approaches to the machine learning of aspects of language and communication focuses on how to learn the semantics of words and other language constructs in a textual context, typically from large text corpora. These approaches can neither take into account the dynamic behavioral context of the word use nor do they learn in interaction with the human user who conceptualizes the respective word. Arguably, an

M. Nickles
Department of Computer Science
Technical University of Munich
E-mail: nickles@cs.tum.edu
Tel: +49-3212-1187769, Fax: +49-89-28917207

A. Rettinger
Institute AIFB
Karlsruhe Institute of Technology
E-mail: rettinger@kit.edu

important reason for this lack is that relevant approaches to machine learning (in particular Reinforcement Learning (RL)) typically still do not cope well with the complexity of high-level, symbolic interaction. Our approach aims at this issue by seamlessly combining approaches to Reinforcement Learning with rich yet computationally effective formal (logical) representation and reasoning capabilities. Being a form of Relational Reinforcement Learning (RRL) [1, 2, 27], our approach provides a more natural representation format and even logical reasoning capabilities in complex environments with a rich relational structure, such as realistic communication domains (compared to traditional Reinforcement Learning). While our implementation currently focuses "only" on the interactive learning of the denotational formal semantics of uninflected concept names (i.e., using machine learning in order to make the connection between a symbol and its initially unknown formal meaning), our approach already allows for the computationally efficient integration of logically represented knowledge into the learning process. Furthermore, our approach aims at the learning of context-sensitive policies for the incremental *querying* of concepts from human users, i.e., learning of sequences of questions and answer suggestions with which the learning agent tries to determine the formal semantics of a word (vs. some informal concept) in a dialog with a human user or another agent. Thus, our work has a strong pragmatic focus. Empirical results from extensive experimentation indicate the applicability of our approach and directions of future research.
While this work introduces our general framework, we envisage a wide area of potential application fields for our approach, such as the interactive acquisition of category semantics for the (Semantic) Web and semantic technologies in general, communication grounding and disambiguation in Human-Computer Interaction, interactive Information Retrieval, and Natural Language Processing.

This article significantly extends [19], which briefly outlines the basic framework and provides results from a single preliminary experiment. In contrast to [19], this article presents a wider range of experiments with improved setups and detailed result analyses. Furthermore, the description of the learning framework is vastly extended, including full algorithms and formal definitions.

The remainder of this paper is organized as follows: The next section presents related works. Section 3 describes the general learning task, the formal calculus being used, and other preliminaries. Section 4 and 5 introduce the formal learning framework, in particular the learning algorithms, with a general approach (Section 4) instantiated for concept learning (Section 5). Section 6 presents and discusses empirical results from the implementation of our framework, and Section 7 concludes with a discussion of our approach and outlines directions of future research.

2 Related Works

We are not aware of any other approaches to interactive Relational RL of concept semantics. However, various approaches to supervised and unsupervised disambiguation of word sense exist (see, e.g., [20, 25]), including approaches in the area of Statistical Relational Learning (e.g., [31]). Conceptually related to our approach are also approaches to grounded language acquisition, where the meaning of words is grounded in observable environment states or events (e.g., [24, 21]). Most of these do not combine grounding with a dialogical learning setup as we do, although some (e.g., [24]) enhance the learning process with the capacity fo

perform ask/tell interaction (but do not learn to act in such dialogues). The challenging task of learning to label words in natural language sentences with existing concepts (that is, their identifiers) is tackled in [9]. In contrast to all of these approaches, our work puts a strong emphasize on learning an interaction policy from reinforcements, in order to determine concept semantics in an interactive decision process.
Pioneering work in the area of computational emergence and evolution of language semantics in interactive settings was done by Luc Steels [22]. Although technically not very closely related, his work had a significant influence on ours and many other contemporary approaches to semantics learning.

Not many approaches to interactive learning in a MDP or POMDP setting exist (note that this is not the same as multi-agent learning, as it will become clear later in this paper), and to our best knowledge none of these aims at concept learning. [16] and various others propose approaches to Interactive RL where some "human teacher" provides rewards for a learning robot (which is a special case of our approach). [7] takes a more general look at Interactive RL and investigates the precise meaning of this term. [11] presents an approach to decision function learning from instructions provided in natural language, which contrasts with most other approaches (including ours) where machine learning uses labelled examples or rewards. However, there is some overlapping with our framework in that what our learning agent interacts with could also be seen as a sort of (semi-)supervisor which serves as a human teacher in support of the actual reinforcement learning task. Learning the ground meaning of human concepts or human instructions can also take the form of demonstrations (tackled typically using Inverse Reinforcement Learning), as, e.g., in [13]. An example for an early statistical approach to dialogue learning in a MDP setting is [4]. More recent results and an extensive survey regarding the use of (Hierarchical) RL for spoken dialogue optimization are provided in [28], mainly addressing core Natural Language Processing (NLP) problems such as uncertainty in speech recognition (which we do not aim at). [17] proposes an interesting approach to Hierarchical Reinforcement Learning of communication, with an emphasize on the optimization of communication needs for coordination, given the communication cost. [5] uses RRL in an agent communication scenario, but in contrast to us they aim at learning with whom to communicate. Apart from the latter, none of the aforementioned approaches uses formal logic, and none deals with concept semantics learning.
Although the framework presented in this paper does not aim at general RRL but rather at an application of RRL to interactive concept learning, an interesting detail of our approach is its use of an ASP solver as a formal reasoner for MDP state updates and for reward and action set computations, whereas other approaches to RRL typically either do not compute states, rewards and action alternatives using formal reasoning (but focus on other aspects of RRL instead) or use Prolog for these computations, such as [18].
We do not focus on classification (in the traditional sense) with this work, but in a certain sense, the example problems presented in Section 6 could be seen as sequential classification tasks, if we view the process of establishing a certain possible world as the grounding of a certain concept as classification. A related approach to sequential classification is proposed in [8], where classification is supported by a decision process where rewards are used to determine the features to be incorporated for each data-point. Similarly to our approach, this approach uses two kinds of actions: actions which affect the object domain and meta-level interactions (for feature determination), which appear to correspond to actions which affect the so-called *interaction state*, as described in Section 4. However, apart from these aspects, the underlying techniques appear to be quite different and the precise relation to our model is still open.

3 Preliminaries

3.1 Learning task and overall technical approach

Before we introduce the details of our learning framework, we want to outline our general learning task and the overall technical approach to this task.

The machine learning task addressed by this paper can be best described as reinforcement learning of interaction policies for the interactive learning of concepts (more precisely: the formal semantics of concepts) by querying human users. In other words, the task is to let a *learning agent* learn optimal (or near-optimal) dialogue act sequences consisting of questions (and answer suggestions) with which the agent tries to determine the unknown formal semantics of one or more concepts in dialogue with a human interaction partner (or multiple partners). The interaction partner knows the concept meaning, but cannot communicate it to the agent easily for some reason, e.g., because of a lack of knowledge of an adequate formal language, whereas the agent lacks knowledge of natural language.
So, the term "learning" has a twofold meaning here: learning concepts from users and learning how to do that.

The focus of our work is clearly more on the pragmatic aspects of learning semantics than on classic NLP aspects (such as data mining in large corpora). We also do not focus on the learning of complex concept structures (such as ontologies), syntax, or speech recognition, but on the learning of interaction trajectories usable for the learning of formal concept semantics.

In our framework, each concept consists of a number of instances. We define the formal semantics of a single instance to be a formal representation of a certain *possible world* (configuration) of the object domain (the domain the interaction partners talk about), or a consistent fragment thereof. In the framework, this is always a so-called *answer set* (also called *stable model*), as explained later in detail. The formal semantics of the concept (sometimes simply denoted as "the concept") is then the set of the formal semantics of its instances. As one can see, our learning framework abstracts away from properties of concepts (if we would want to define a concept intentionally) and also does intentionally not say anything about the object domain, since a concretization of these aspects is not required in the learning process.
Learning concept semantics using our approach thus actually means to learn which object domain constellation(-s) the interaction partner has in mind when using a word which names the respective concept. At this, potential problems such as *homonymy* and inconsistent concept instance semantics are dealt with to some degree (using the machine learning concept of non-stationarity), as explained later.

The learning agent and its interaction partner(s) initially share only a very limited set of simple words with mutually known semantics. The agent uses these words in querying the interaction partner in order to learn the semantics of new words (concept names). After a new concept has been learned, it can also be used in questions to the user.
The agent's task is to find out the semantics of concepts by more or less intelligently searching the space of all possible configurations of the object domain. The virtual or possibly even physical movements of the learning agent from one state to another are modeled directly as a Markov Decision Process (MDP). The agent can ask the human user simple questions ac-

cording to schemas such as "Does concept c_1 resemble the known concept c_2?" or "Is the wanted semantic related to the current state by the relative concept c?" (relative concepts being comparatives such as "bigger" or "lower"). Depending on the answer provided by the human, the agent proceeds to a new candidate state of the object domain, until the human user, who is able to observe the current state, identifies the state as the correct (partial or full) semantics of the concept being learned.
Only a simple interaction protocol is imposed (e.g., demanding that questions have to be answered) - everything else, in particular, when to use which question, needs to be learned by the agent from episodes of experience with its human interaction partner(s). Additional challenges here which we will deal with are humans who give inconsistent answers and the case that there might be multiple different interaction partners who possibly assign different meanings to the same word. However, we do not deal with noise and other errors on the level of technical signal transmission (such as dropped messages or speech recognition). The names of the concepts (i.e., the actual words) also do not matter much, only the actual concept behind the word is important. Concept identifiers might be provided by the human user or be automatically generated according to some scheme.

As a motivating example, consider the following learning task:

Let's suppose the learning agent lives in a so-called *blocks world*, consisting of a number of movable blocks and a table. The user wants the agent to reconfigure the blocks world. He has some idea (informal concept) of what "nice" blocks world configurations would look like, but the agent does not have a clue yet. A blocks world configuration is a specific arrangements of blocks and acts as a ground *instance* of the respective concept (see the four example instances of the concept "high" depicted in Figure 1). Unfortunately, the human user cannot communicate his idea to the agent in a single, precise message, since the blocks world is large and complex, and the concept might consist of several possibilities (various different configurations). Also, the user might not be able to formalize concepts anyway. Furthermore, the agent, being an artificial agent, understands only pretty simple symbolic commands and questions, but no natural language sentences. For all these reasons, the human needs to teach the agent iteratively in a simple step-by-step dialogue how a desired blocks world configuration looks like, by means of giving rewards for improvements of the blocks world and by answering questions received from the agent. Fortunately, user and agent can share a set of concepts whose semantics (formal groundings) are already known to the agent. Even more, after having learned a new concept, the agent memorizes its semantics and can use this concept subsequently within questions directed to the user. E.g., the agent might already know what "lower" means (a blocks world configuration is lower than another in terms of the height of towers of stacked blocks) and can thus ask the user whether the desired configuration is lower or not, compared to the current configuration. So, the agent acts by asking questions (using already learned or predefined shared concepts), but also by presenting the user at each step a new, improved (or sometime worse) blocks world configuration, until the user is happy with it.
The learning task is actually twofold: firstly, the agent learns what a concrete "nice" configuration is, and secondly and more importantly, the agent learns (over multiple such episodes) to improve its action policy (how to accumulate reward, which questions to ask, how to build parts of the configuration...). Each blocks world configuration has a precise formal representation which is taken as the actual semantics of the concept "nice blocks world" as soon as the user is satisfied with it (or as an instance of the semantics, in case there is more than one

"nice" blocks world configuration).

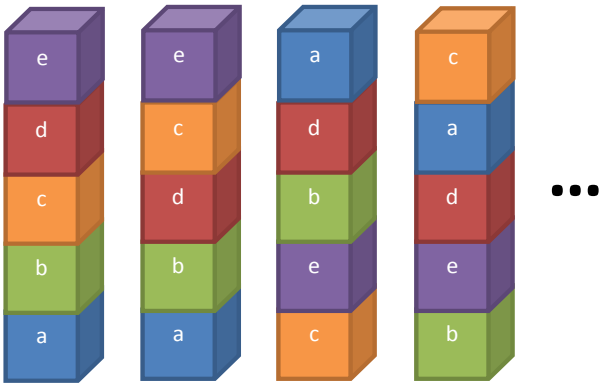

Fig. 1 Four example instances (blocks world configurations) of the concept "high (building)" in a blocks world which consists of five blocks altogether (a,b,c,d,e)

In our experiments, all learning takes place in an interactive setting - however, our framework allows for the integration of any kind of formal background knowledge and data from other information sources, as long as it is provided in form of first-order logic, and provided the respective object domain is finite (a restriction shared with most other approaches in the field of Statistical Relational Learning, such as Markov Logic Networks [6]).

We believe that this scenario is basically realistic - in many real-world software applications, e.g., in the HCI area or in dialogical information retrieval, users need to communicate with machines using a simple yet flexible vocabulary, in case the user is not able to provide commands and other communications using some complex formal language, whereas software is typically not easily equipped with sufficient or any NLP capabilities. Having an artificially intelligent facility for learning (or disambiguating) the meaning of words from the user is supposed to be highly useful here.

A core technical theme of our paper is the efficient utilization of formal knowledge during the statistical learning process. A combination of logical knowledge representation and statistical learning is not only a necessity for the space efficient representation and processing of large state/action spaces, but is also required in order to utilize large or complex amounts of knowledge for the learning process, such as given interaction protocol specifications, guidance for the human interaction partner(s), information from external sources such as databases, and formal rules which constrain agent behavior. Furthermore, our framework needs to deal with formal knowledge acquired during the interaction process (in particular memorizing the semantics of concepts already learned). We address these requirements using a form of Relational Reinforcement Learning (RRL) supported by Answer Set Programming (ASP). ASP also neatly supports automated planning [14], which comes handy

later in our framework (Section 5).

Being part of the increasingly popular area of Statistical Relational Learning (SRL) [35], Relational Reinforcement Learning [1, 2] uses relational representations of Markov states and actions (e.g., formal-logically). This allows for a rich formal characterization of complex domains whose complex structural properties would otherwise be inaccessible to RL. Various approaches to Relational RL exist, including [2, 32]; for an extensive survey, please see [27].
While we do not aim at improving RRL as such with this work (but rather aim at applying it for interactive concept semantics learning), our framework differs from traditional Relational RL approaches in its human-agent interaction component and by the use of a highly efficient approach to the employment of formal rules as background knowledge, namely an Answer-Set Programming (ASP) implementation of the Event Calculus (EC) [39]. The basic learning algorithm is a variant of Relational Q-learning [10]. Besides the general benefit of Relational RL (i.e., the ability to represent and learn in structurally very complex domains), the main advantage of this hybrid approach is that it seamlessly integrates logical reasoning and RL. Concretely, the use of the Event Calculus (EC) significantly simplifies the modeling of logical conditions, contexts and effects for/of agent and human actions, and the use of ASP facilitates a computationally efficient computation of reasoning tasks in the EC, including planning (a discussion of efficiency issues will be presented in Section 5).

3.2 The Event Calculus

The Event Calculus [38, 37] and its cousin, the Situation Calculus (SC) [26], are popular and easily implementable formal calculi for reasoning about actions and other events and their effects in dynamic systems. EC and SC are close relatives: both can be implemented in a logic programming environment, both provide efficient means for dealing with the frame problem (the old AI problem of formalizing domain dynamics without having to make explicit all non-effects of actions), and syntactical schemas for translating one into the other exist. The reason for using the EC instead of the SC (or other related formalisms, such as temporal action logics) in this work is thus to some degree a matter of taste. However, technically, the EC's provision of a discrete time structure simplifies the specification of dynamic systems such as MDPs, whereas the main strength of the SC over the EC (its better suitability for hypothetical reasoning) seems not so important for our use case. A detailed comparison of EC and SC can be found in [37]
There are several variants of the EC. We make use of the so-called Circumscriptive Event Calculus [38], which is arguably one of the most popular forms of the EC (circumscription is a specific way of solving the frame problem).
The EC defines a first-order language with reified fluents (conditions which can change over time, such as "the green block lies on the table") from which only the knowledge of the following predicates is required for understanding this paper. In our framework, the background knowledge base of the learning agent consists entirely of EC formulas.

$HoldsAt(f,t)$ denotes that fluent f is true at time point t.
$Happens(e,t)$ denotes that event e occurs at time point t. For us the most important events are the actions performed by the learning agent.
$Initiates(e,f,t)$ specifies that fluent f becomes true at time $t+1$ in case event e occurs at time t.

$Terminates(e,f,t)$ specifies that fluent f becomes false at time $t+1$ if event e occurs at time t.
$Clipped(t_1,f,t_2)$ denotes that fluent f is terminated between times t_1 and t_2.
$Declipped(t_1,f,t_2)$ means that fluent f is initiated between times t_1 and t_2.

However, for understanding most of this paper, the knowledge of the following predicates should suffice: *HoldsAt*(f,t), *Happens*(e,t), and *Initiates*(e,f,t).

Time points are discrete, linearly ordered, and can refer to the past, the present, and the future.
Although fluents look much like predicates, they are in fact reified. As such, they can be quantified over.

3.2.1 Blocks world domain formulated using the EC

As a running example for an object domain, we use the well-known *blocks world* (BW), which is arguably the most frequently used benchmark domain in the context of RRL. Although straightforwardly representable using the Event Calculus and easy to explain, the BW is highly scalable and far from trivial from a computational point of view. BWs are ideal for use with RRL, because unless the BW is very small, it can be compactly represented only using a relational language with variables.
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 either a *block* (named as *blockA*, *blockB*, ...), or the *table*, or nothing. The fact that some block x is on top of some other block y is expressed with $on(x,y)$. $on(x,table)$ means that block x is directly on the table. $clear(x)$ denotes that there is currently no block on top of block x. Fluent *clear* seems redundant (it can always be expressed in terms of *on*, provided the BW is finite and its size is fixed), but *clear* is helpful as a short-cut for an equivalent configuration of fluents with predicate *on*.
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 which moves block x on top of block y is expressed with $stack(x,y)$. If the action succeeded, subsequently $on(x,y)$ holds.
Our BW is finite and fully observable both for the learning agent and the human interaction partners.

Using the EC, the truth of statements about which block is where is time-dependant and dynamic, that is, these statements are fluents. E.g., $HoldsAt(on(blockC,blockB),25)$ denotes that *blockC* lies on *blockB* at time 25. We can also quantify over blocks and time steps:
$\exists block.HoldsAt(on(block,table),12)$
denotes that there is at least one block on the table at time step 12.
As another example,
$Happens(stack(blockA,blockB),T+1) : -Happens(stack(blockB,table),T)$
specifies that if *blockB* has been put on the table at time T (a universally quantified variable), *blockA* needs to be stacked on top of *blockB* at the following time step.
Events are not restricted to action events, but in our framework, the only types of events are actions performed by the learning agent in the object domain (such as $stack(blockB,table)$), speech acts addressed to its interaction partner(s), and speech acts performed by interaction partners.

Actions in the object domains are precisely those events which transform the current Markov state (informally, a set of fluents from the object domain, such as $\{on(blockA,table), on(blockB,blockA)\}$) into another state during the search for the semantics of some concept. Speech acts also cause state transitions, but we will later see how to separate "object domain" states (e.g., blocks world configurations) from "interaction domain" states.

3.2.2 Reasoning in the EC

Reasoning in this calculus is first order reasoning based on a set of certain axioms. For a detailed description and the reasons behind these axioms, please see [38].

$$Clipped(t_1,f,t_2) \leftrightarrow \exists a,t.Happens(a,t) \wedge t_1 \leq t < t_2 \wedge Terminates(a,f,t) \tag{1}$$

$$Declipped(t_1,f,t_2) \leftrightarrow \exists a,t.Happens(a,t) \wedge t_1 \leq t < t_2 \wedge Initiates(a,f,t) \tag{2}$$

$$HoldsAt(f,t_2) \leftarrow Happens(a,t_1) \wedge Initiates(a,f,t_1) \wedge t_1 < t_2 \wedge \neg Clipped(t_1,f,t_2) \tag{3}$$

$$\neg HoldsAt(f,t_2) \leftarrow Happens(a,t_1) \wedge Terminates(a,f,t_1) \wedge t_1 < t_2 \wedge \neg Declipped(t_1,f,t_2) \tag{4}$$

$$HoldsAt(f,t_2) \leftarrow HoldsAt(f,t_1) \wedge t_1 < t_2 \wedge \neg Clipped(t_1,f,t_2) \tag{5}$$

$$\neg HoldsAt(f,t_2) \leftarrow \neg HoldsAt(f,t_1) \wedge t_1 < t_2 \wedge \neg Declipped(t_1,f,t_2) \tag{6}$$

EC reasoning can take many concrete forms, including the use of Prolog and the use of Answer Set Programming [39]. The ASP approach currently provides the fastest form of reasoning in the Event Calculus for standard benchmark problems, and for finite domains, the ASP-based implementation of the EC is fully equivalent to the Circumscriptive EC [39]. We utilize an interplay of EC and ASP, because they add up to an ideal combination of reasoning about dynamic systems and the efficient practical realization of such reasoning. A shortcoming of this solution is that it restricts us to finite domains (a restriction shared with most current approaches to Statistical Relational Learning).
The ASP-solver computes a finite set of so-called *answer sets* or *stable models* of the agent's EC knowledge base (a form of satisfying logical models in form of sets of propositional formulas). From now on, if not indicated otherwise, we will use the term "model" as in ASP-terminology, i.e., to denote a stable model. Readers not familiar with ASP may conceive "model" quite safely as ground "possible world" or as "Herbrand interpretation" within the scope of this work. Related to the EC, a model is a set of fluents which hold at a certain time step.

4 Learning framework

In the following, we will specify our learning framework, in two steps: in this section, a number of interactive and non-interactive RRL algorithms are presented, and in Section 5, the utilization of those algorithms for concept semantics learning is introduced. The latter part is achieved almost entirely in form of logical programming (facts and rules formalized using the EC) and is in no way fixed - virtually any sort of logical specifications could be "plugged into" the learning algorithms in order to provide different learning settings or prior knowledge.

4.1 Basic learning algorithms

As already pointed out, in our scenario the learning agent learns how to act optimally in its environment. As a part of that, it learns to communicate with one (human) interaction partner at a time.
Apart from what is needed for the learning process, no assumptions or requirements are imposed on the agent's cognitive, reasoning or sensing capabilities.

Before we introduce our interactive learning algorithms, which are based on general RRL algorithms introduced in [1, 2, 10, 15], let us briefly recapitulate general Relational Reinforcement Learning (RRL): Reinforcement Learning (RL) aims at learning an optimal behavior policy, that is, which action is preferred in any given state of the learning agent's environment in order to maximize future rewards. The difference between standard RL and RRL [1] is that the latter uses a relational representation format for states and actions (or even a formal logic) and makes use of this format in order to improve learning, for example by employing some relational regression mechanism or by enabling a compact relational representation of the approximated value function.

The basic algorithm (Algorithm 1) which we build upon is plain Q-learning, but demanding a relational (e.g., logical) format for states and actions, which will be utilized in concrete instances of the algorithm later in this paper.
We assume full observability of the environment, which allows us to represent the underlying state progression as a Markov Decision Process (MDP).

Definition 1 (Value-based Relational RL) Let

- S be a set of possible domain states, represented in a relational format,
- A be the set of all agent actions, represented in a relational format,
- $A_s : S \to 2^A$ result in a set of agent actions possible in a certain state, represented in a relational format,
- $PS : S \times A \times S \to [0;1]$ be a state transition probability function (typically unknown), and
- $R : S \times A \to \mathbb{R}$ be a real-valued reward function.

The agent's goal is to learn an optimal action selecting policy $\pi^* : S \to A$ which maximizes the *discounted return* $R_t^{\pi^*} = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$ from any time step t. $r_t = R(s_t, a_t)$ is

the *reward* received after action a_t has been performed in state s_t at time t. In other words, the return is the cumulative reward obtained in the future, starting in state s_t. Future rewards are reduced by some discount factor $\gamma^k \in [0;1]$. The optimal *Q-function* $Q^*_{Rel} : S \times A \rightarrow \mathbb{R}$ yields the value (*Q-value*) of taking a certain action in a certain state, and is estimated using a form of Relational Q-learning [10] in our framework. The current estimate is the Q-function (or Q-table) Q_{Rel}. The subindex *Rel* should indicate that we will later use a relational means in order to improve the estimation of Q-values (see Section 4.3.1). Also, state updates and various other operations will make use of the relational format of states and actions in later instances of the basic algorithms.
The optimal policy follows directly from Q^*_{Rel} (that is, choose in each state the action with the highest Q-value according to Q^*_{Rel}).

Algorithm 1 (Relational Q-Learning)

loop
  Initialize Q_{Rel}
  Specify start state s
  repeat
    $a \leftarrow \pi(s)$, where π is some policy based on Q_{Rel} (e.g., ε-greedy)
    Perform action a, observe new state $\bar{s}$
    Observe reward $r \leftarrow R(\bar{s},a)$
    if $\bar{s}$ is not a goal state **then**
      $Q_{Rel}(s,a) \leftarrow r + \gamma max_{\bar{a} \in As(\bar{s})} Q_{Rel}(\bar{s},\bar{a})$ (learn: correct the Q-value by updating the old value using new information (value iteration update))
    else
      $Q_{Rel}(s,a) \leftarrow r$ (learn)
    end if
    $s \leftarrow \bar{s}$
  until s is a goal state
end loop

Definition 2 (Interactive Relational RL)
From now on, reward will be given directly by the interaction partner (or more precisely: they are determined from the state of the interaction), the partner's actions will be made explicit in the algorithms (as opposed to being just visible as ordinary observations in the environment), and the partner's actions can constrained by the agent. The following algorithm (Algorithm 2) for Interactive Relational RL (IRRL) is nevertheless not an extension or modification of (Relational) Q-learning, but just a notational variant where states are split into domain and interaction states. However, externalization of reward-giving to some human user and letting the state depend on human-generated events have important consequences wrt. stationarity of the environment and the reward function, as elaborated below.
Furthermore, we now allow for *guidance*: the agent computes the set of possible actions allowed to be performed by the human interaction partner and lets the human perform only an action from this set, i.e., the agent constraints its environment. (Note that letting the interaction partner provide reward is also a form of guidance, however, we use the term guidance only in the aforementioned sense in this paper.)
Let

– *OS* be a set of object domain states, each represented as a set of object domain fluents (represented in a relational format). In our running example domain, these are the pos-

sible constellations of the blocks world.

– IS be a set of *interaction states*, each represented as a set of *interaction fluents* (in a relational format). Informally, an interaction state is a "state of communication affairs", as opposed to states of the object domain (e.g., "The interaction partner has demanded that..." or "The interaction partner has replied positively"). In other words, interaction states are meta-states referring to social interaction and speaking *about* the lower-level object domain. In particular, this comprises all relevant information the learner has about the interaction with its human partner, determined from the course of speech acts which have been performed so far.
Formally, a fluent is an interaction fluent in our framework simply if it is not a condition of the object domain.

  Later, we will simply combine OS and IS into a single notion of state, but for now we keep them separated in order to emphasize the different content levels of OS and IS.

– A be a set of all possible actions of the learning agent, represented in a relational format,

– $E_s : OS \times IS \rightarrow 2^A$ give a set of agent actions possible in a certain combined domain/interaction state, represented in a relational format,

– IA be a set of all possible actions of the interaction partner, each represented in a relational format,

– $EH_s : OS \times IS \rightarrow 2^{IA}$ give a set interaction partner action events possible in a certain combined domain/interaction state, represented in a relational format,

– $\theta : \mathbb{N} \times OS \times IS \times A \times IA \rightarrow IS$ be the interaction state determination function (see below),

– $R_{IS} : IS \rightarrow \mathbb{R}$ be a real-valued reward function which determines the rewards from interaction state using formal reasoning,

– $\pi : OS \times IS \times 2^A \rightarrow A$ be the policy function of the learning agent. There are several possibilities to define this function (see, e.g., [2]). In Definition 3, we will propose a concrete instance.

– $Q_{Rel} : OS \times IS \times A \rightarrow \mathbb{R}$ simply corresponds to $Q_{Rel}(os \cup is, a)$ in Definition 1.

(Other symbols as in regular Relational Q-learning, see Definition 1)

Rewards depends here on some external, possibly unreliable source (human, robot or software agent). That is, the reward function is not necessarily stationary and thus convergence of the Q-learning value estimation process to an optimal policy is no longer guaranteed; this issue is also a known obstacle in Multiagent Learning (where one or more of the interaction partners also explicitly learn, and not just a single agent as in our scenario), which we do not consider here.

Even more, we possibly have a non-stationary model of the environment where the state transitions (as for the IS) depend on time and history. This also means that learnability might

be impossible or severely restricted, depending on the "stationarity" of the interaction partner. In particular, observe that the result of the interaction state transition function θ depends also on a time point. We use $\theta_t(os, is, a, ah)$ to stress that the other agent or human reacts on the recent action a of the learning agent, but also that this is not the only determining factor.

Later, we will tackle this issue by means of so-called *profiles* and by a form of generalization.

The idea of having "interactive states" is borrowed from [12] (similar ideas can be found in the area of Multiagent Learning), but with the difference that we do not aim at modeling the interaction partner's cognitive state (in terms of nested beliefs or the like), but only those aspects of the human-agent interaction which do not directly relate to the object domain. Note that both types of state (both *IS* and *OS*) are assumed to be fully observable for the learning agent.

Algorithm 2 (Interactive Relational Q-Learning)

loop
  Initialize Q_{Rel} (the Q-function, cf. Def. 1)
  Specify start state os, is, start time t
  repeat
    $a \leftarrow \pi(os, is, A)$
    Perform action a
    Observe new object domain state $\overline{os}$
    Provide guidance: Let interaction partner choose her action ah solely from set $EH_s(os, is)$
    Let interaction partner perform ah
    Determine new interaction state: $\overline{is} \leftarrow \theta_t(os, is, a, ah)$
    $t \leftarrow t + 1$
    Obtain reward: $r \leftarrow R_{IS}(\overline{is})$
    if $\overline{os}$ is not a goal state **then**
      $Q_{Rel}(os, is, a) \leftarrow r + \gamma \, max_{\bar{a} \in E_s(\overline{os}, \overline{is})} Q_{Rel}(\overline{os}, \overline{is}, \bar{a})$ (learn)
    else
      $Q_{Rel}(os, is, a) \leftarrow r$ (learn)
    end if
    $os \leftarrow \overline{os}$
    $is \leftarrow \overline{is}$
  until os is a goal state
end loop

4.2 ASP-supported Relational RL

We now present the (non-interactive) ASP-supported variant of RRL, an instance of RRL which makes use of Answer Set Programming for all state updates and action set and reward computations, based on an approach introduced in [15].

Before we show the actual algorithm (Algorithm 3), we need to explain how it deals with formal knowledge: syntactically, our learning agent's knowledge base (KB) is a disjunctive logic program, and all reasoning is done using an Answer Set Programming (ASP) reasoner

("solver"). The learning algorithm performs certain monotonic extensions of the KB determined by the agents interaction with its environment, and possibly further revisions of its KB depending on the agent's cognitive architecture (which we do not specify).
The ASP-solver is invoked in each learning step in order to compute a finite set of so-called *stable models* (*answer sets*) of the learning agent's KB. These are types of satisfying logical models in form of sets of ground atoms. Fluents and actions are present within stable models as ground atoms of the form *HoldsAt(fluent, time)* and *Happens(event, time)*. The function of the ASP-solver at each learning step is manifold: most important, it computes the new state in the MDP, given the agent's recent action and observation. It does so with help of static background knowledge, such as post-/preconditions of actions formulated in the EC, and optionally further rules which are domain-specific, such as the physics of the blocks world.
Furthermore, the solver computes the set of actions which are logically possible at the next time step and from which the learning agent chooses its next action.

ASP is arguable a more convenient way for computing the set of alternative actions and the updated state, compared to traditional logic programming used in RRL, since the stable models obtained from the ASP-solver are already complete sets of ground atoms which cover the entire requested information set (and ASP-solvers are, like SAT-solvers, optimized for computing such sets). No further computation needs to be performed for this, just a set of formal rules is required. Furthermore, ASP currently provides the fastest form of reasoning in the EC for standard benchmark problems, and translations of FOL EC into ASP (logic programs using stable model semantics) are fully equivalent to the Circumscriptive EC [39] for finite domains like ours.

Definition 3 (ASP-supported RRL)
Let

- S the set of all possible states,
- A be the (global) set of all possible actions of the learning agent, each represented in a relational format,
- $Q_{Rel} : S \times A \to \mathbb{R}$ be the Q-function (as in Definition 1),
- KB be the set of all possible knowledge bases,
- kb be the knowledge base of the learning agent. This comprises all kinds of rules and facts the agents happens to know or comes to believe (including information about actions and observed events).
- $models : KB \to 2^M$ be a function which computes all stable models of a knowledge base, with M being the set of all Herbrand-interpretations of KB[1].
- $E_s : 2^M \times \mathbb{N} \to 2^A$ computes the set of all actions of the learning agent which are possible according to at least one given model. M as defined above.

[1] In order to avoid a lengthy formal introduction of the term stable model here, which would be out of scope.

If the current state corresponds to time step t, an action is considered possible if it might happen at time t. Formally:
$a \in E_s(M,t) \leftrightarrow \exists m \in M : Happens(a,t) \in m$.

– *fluents* : $2^M \times \mathbb{N} \rightarrow S$ be a function which computes from stable models the set of all fluents which hold at a certain time t. This subset of the entire stable model constitutes a state. It does not contain any "time stamps", i.e., if a fluent appears as ground atom *HoldsAt(fluent, time)* within the stable model, the state contains just *fluent*.
If the argument of *fluents* is a non-empty set (nondeterminism), one model is picked ("observed"), according to some probability distribution. M as defined above.

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

– $\pi : S \times 2^A \rightarrow A$ be the policy function with

$$\pi(s,A') = argmax_{a \in A'} Q_{Rel}(s,a)$$

(A' being a subset of A)
Instead of *argmax*, Boltzmann-*softmax* or ε-greedy action selection can be used in the body of π, in order to foster exploration. In our experiments, we used ε-greedy with an ε of 0.2, meaning that the action with the highest value is picked with probability $1-0.2$, and another action is randomly selected with probability 0.2. Note that this is not the only source of randomness (and therefore exploration) in action selection: the list of stable models (and legal actions) retrieved from the ASP-solver in each learning step is in unspecified order and the initial state in each episode is chosen randomly.

Algorithm 3 (ASP-supported Relational Q-learning)

loop
Specify start state (possibly empty, but normally a random sample of all logically consistent states): $s = fluents(m)$, with $M = models(kb)$
Start time: $t \leftarrow 0$
repeat
$a \leftarrow \pi(s, E_s(M,t))$
Perform action a
Retrieve new state s:
$kb \leftarrow kb \cup Happens(a,t+1)$
$M \leftarrow models(kb)$
$s = fluents(M,t+1)$
if M is nondeterministic wrt. the next state[2] **then**
$kb \leftarrow kb \cup \bar{s}$ (fix the choice of the new state in the knowledge base)
end if
Get reward: $r \leftarrow R(\bar{s})$
if $\bar{s}$ not a goal state **then**
$Q_{Rel}(s,a) \leftarrow r + \gamma\, Q_{Rel}(\bar{s},\bar{a})$ (learn)
else
$Q_{Rel}(s,a) \leftarrow r$ (learn)

[2] Formally: M is called nondeterministic in our framework wrt. the state at some time t iff $\exists m,m' \in M$: $m \models holdsAt(f,t) \wedge m' \not\models holdsAt(f,t)$ for some fluent f.

end if
$s \leftarrow \overline{s}$
$t \leftarrow t+1$
until s is a goal state, or some maximum reward sum is exceeded, or some maximum number of time steps is reached
end loop

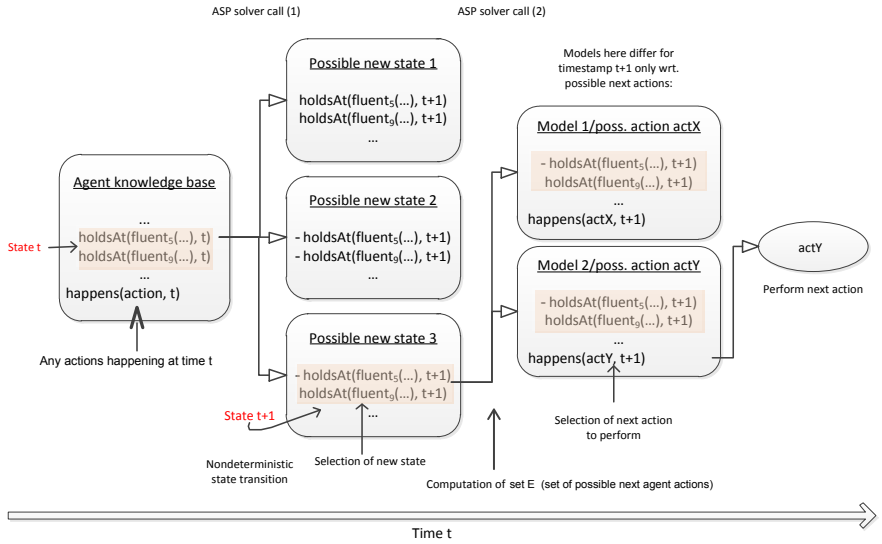

Fig. 2 State and action selection steps in the learning process (somewhat simplified, omitting profiles). Note that the formula prefix "-" means negation. Further note that actions happening at time t (left hand side) comprise agent as well as human actions.

The previous algorithm contains non-trivial model and action selection steps. After the agent has performed its most recent action, the ASP solver provides it with a set of models M from which the next state is selected. The transition of states is possibly nondeterministic, namely in the case when multiple candidates for the next state exist (e.g., think of rules which say (informally): "After tossing a coin, the outcome is either heads or tails. If it is heads, stack blockA on top of blockB. Otherwise, put blockA on the table." - such rules can be easily specified in the EC using disjunction and various other choice constructs). To make the process more intuitive, the major steps involved here are depicted graphically in Figure 2 (somewhat simplified compared to the full algorithms).

4.3 ASP-supported Interactive Relational RL

Merging the previous algorithms, we obtain our approach to ASP-supported Interactive RRL. As a further ingredient, we add the possibility of non-monotonic knowledge base

modifications by means of so-called *profiles*.

Profiles and states

Profiles address the problem of possible non-stationary behavior of interaction partners - these might change their ostensible beliefs and intentions during conversation (or more concretely, the intended meaning of a concept name), in which case the state of interaction becomes inconsistent and the Q-table might contain misleading values obtained from earlier learning phases. Also, the problem of *homonymy* (the same word spelling is used for different concepts) is dealt with using profiles.

Profiles work similarly to cookies from the viewpoint of web servers: they allow the learner to adapt its learning process to the respective partner and its current behavioral profile. A profile is basically a "tag" in form of adequate ground atoms and therefore profiles are part of the states of the MDP. But in contrast to ordinary formulas which are part of the state, profiles are automatically carried over to the next learning episode, by incorporating them at runtime into the agent's background knowledge base (until they are updated or removed). Furthermore, they are automatically removed from the knowledge base when they become logically inconsistent with the rest of the agent's knowledge (e.g., when the profile becomes negated in response to some agent action or the behavior of the interaction partner). Finally, profiles do not only structure the Q-table (as all components of the Markov state do), but they also limit the computation of similar state/action-pairs (using k-nearest neighbor in the instance-based learning mechanism, cf. Section 4.3.1) to those pairs whose states belong to the same profile-determined partition of the Q-table. Doing so, a profile can provide a simple, continually revisable model of the current communication partner (or any other aspect of the learner's environment) during all those learning episodes where the respective profile is set. This allows for the learning algorithm, e.g., to account for mutually-inconsistent human behavioral patterns, by "switching" between those non-stationary patterns. In contrast to techniques for dealing with semantic drift, this also accounts for recurring patterns (e.g., dealing with different, mutually-inconsistently acting partners who take turns in talking to the learning agent). However, profiles are not a mechanism for concept (meaning) evolution (as in Luc Steels' works [22]), since they do only affect the learning of the agent's action policy, not the learned concepts as such.

By incorporating profiles into Q-table entries, profiles can be used to "partition" the Q-table into subsections, each corresponding to a certain profile. If the learner changes the current profile whenever it has obtained relevant new information (e.g., detection of inconsistent ostensible beliefs of the interaction partner, as in an example shown in Section 6), it switches to the appropriate partition of the Q-table. Sufficiently fine grained and loaded with information from the observation history, profiles might even enable the learner to deal with simple POMDPs, like an approximation of continuous space belief-MDPs (but we haven't investigated this possibility).

Altogether, our learning agent deals at each step of a learning episode with the following information sets:

1. the learning agent's current knowledge base KB ,
2. the stable models of KB (sets of ground atoms deduced from KB, corresponding to possible worlds),

3. the (observable) object domain states OS and the interaction states IS, and
4. the current profile.

Together, these information pools form a hierarchy (with KB being at the root) from which the stable models are derived by the ASP solver. From the stable models, the domain state, the interaction state and the profile are obtained by selecting the appropriate fluents.
Each Markov state combines domain state, interaction state and the current profile. The first two are not conceptually different from the point of view of Reinforcement Learning.

Definition 4 (ASP-supported IRRL)
Let

- OS as in Definition 2,
- IS as in Definition 2,
- $S : OS \times IS$ (all possible states, combining object domain and interaction states),
- P the set of all those ground atoms which can possibly be used in profiles,
- A be the (global) set of all possible actions of the learning agent, each represented in a relational format,
- $Q_{Rel} : S \times P \times A \to \mathbb{R}$ be the Q-table. Note that the actual state is from $OS \times IS$, whereas the profiles provide a sectioning of the Q-table in sub-Q-tables, each for a different profile. Below we will propose a mechanism for a more elaborate representation of Q-values which makes use of prediction.
- IA be a set of all possible action events that can be performed by the interaction partner, each represented in a relational format,
- KB be the set of all possible knowledge bases,
- kb be the knowledge base of the learning agent. This comprises all kinds of rules and facts the agents happens to know or comes to believe (including information about actions and observed events).
- $models : KB \to 2^M$ be a function which computes all stable models of a knowledge base, with M being the set of all Herbrand-interpretations of KB[3].
- $E_s : 2^M \times \mathbb{N} \to 2^A$ computes the set of all actions of the learning agent which are possible according to at least one given model. M as defined above.

  If the current state corresponds to time step t, an action is considered possible if it might happen at time t. Formally:
  $a \in E_s(M,t) \leftrightarrow \exists m \in M : Happens(a,t) \in m$.

[3] In order to avoid a lengthy formal introduction of the term stable model here, which would be out of scope.

– $EH_s : 2^M \times \mathbb{N} \rightarrow 2^{IA}$ be conceptually the same as E_s, but for action event emitted by the interaction partner (as a sort of soft constraining or guidance for the partner). M as defined above.

– *fluents* : $2^M \times \mathbb{N} \rightarrow S$ be a function which computes from stable models the set of all object domain and interaction fluents which hold at a certain time t. This subset of the entire stable model constitutes a state. It does not contain any "time stamps", i.e., if a fluent appears as ground atom *HoldsAt(fluent, time)* within the stable model, the state contains just *fluent*.
If the argument of *fluents* is a non-empty set (nondeterminism), one model is picked ("observed"), according to some probability distribution. M as defined above.

– $R : S \rightarrow \mathbb{R}$ be a real-valued interactive reward function.
Obviously, the direct provision of reward by the interaction partner (as in conventional Interactive RL [7, 16]) is a special case of this.

– $\pi : S \times 2^A \times 2^M \rightarrow A$ be the policy function with

$$\pi^{profile}(s, A') = argmax_{a \in A'} Q_{Rel}^{profile}(s, a)$$

(A' being a subset of A)
Instead of *argmax*, Boltzmann-*softmax* or epsilon-greedy action selection can be used in the body of π, as described before.

Algorithm 4 (Interactive ASP-supported Relational Q-learning)

profile $\leftarrow \{\}$
loop
  Specify start state (possibly empty, but normally a random sample of all logically consistent states): $s = fluents(m)$,
  with $M = models(kb \cup profile)$
  Start time: $t \leftarrow 0$
  repeat
    $a \leftarrow \pi^{profile}(s, E_s(M,t))$
    Perform action a
    Retrieve new state s:
      Provide guidance: Let partner choose action ah solely from $EH_s(M,t)$
      Let interaction partner perform ah
      $kb \leftarrow kb \cup Happens(a,t+1) \cup Happens(ah,t+1)$
      $M \leftarrow models(kb \cup profile)$
      if M empty (i.e., $kb \cup profile$ unsatisfiable) **then**
        profile $\leftarrow \{\}$
        $M \leftarrow models(kb)$
      end if
      $s = fluents(M, t+1)$
      if M is nondeterministic wrt. the next state[4] **then**
        $kb \leftarrow kb \cup \bar{s}$ (fix the choice of the new state in the knowledge base)

[4] Formally: M is called nondeterministic in our framework wrt. the state at some time t iff $\exists m, m' \in M$: $m \models holdsAt(f,t) \wedge m' \not\models holdsAt(f,t)$ for some fluent f.

end if
Get reward: $r \leftarrow R(\bar{s})$
$profile \leftarrow \{atom \in model, model \in M\} \cap P$ (update profile from recent set of stable models [5])
if $\bar{s}$ not a goal state **then**
$Q_{Rel}^{profile}(s,a) \leftarrow r + \gamma\, Q_{Rel}^{profile}(\bar{s},\bar{a})$ (learn)
else
$Q_{Rel}^{profile}(s,a) \leftarrow r$ (learn)
end if
$s \leftarrow \bar{s}$
$t \leftarrow t+1$
until s is a goal state, or some maximum reward sum is exceeded, or some maximum number of time steps is reached
end loop

Remark: Any update of the interaction state is determined by the learning agent's KB, the learning agent's recent action, and of course the recent action of the interaction partner.

We also provide a SARSA-variant of the previous learning algorithm (Algorithm 5). It is based on the core (non-interactive) Relational SARSA algorithms presented in [10, 15]. The difference between this algorithm and Algorithm 4 is precisely the difference between regular Q-Learning and SARSA-learning. Algorithm 5 can be found in Appendix A.

4.3.1 Adding regression

We provide the previous algorithms with an (optional) form of relational generalization over similar state/action-pairs, in order to facilitate learning even if the agent has seen only a small number of learning examples yet.
For this, we use an instance-based regression mechanism Q_{RIB}, which provides value predictions for learning examples. Like the Relational-Instance-Based learning (RIB) approach presented in [2], Q_{RIB} is for predicting the Q-values of examples (s,a), calculated using a k-nearest-neighbor (kNN) estimator as follows. $(\bar{s},\bar{a})$ denotes examples which are already in memory (and in the same partition as s) and which are used as a kind of "prototypes" in the estimation of Q-values for "similar" new examples, as follows:

$$Q_{Rel}^{p}(os,is,a) = Q_{RIB}^{p}(os \cup is,a) \tag{7}$$

$$Q_{RIB}^{p}(s,a) = \frac{\sum_{(\bar{s},\bar{a})\in p} \frac{Q_{RIB}^{p}(\bar{s},\bar{a})}{d((s,a),(\bar{s},\bar{a}))}}{\sum_{(\bar{s},\bar{a})\in p} \frac{1}{d((s,a),(\bar{s},\bar{a}))}} \tag{8}$$

At this, $d((s,a),(\bar{s},\bar{a}))$ is a distance or pseudo-distance between state/action pairs. In the literature, various such metrics have been proposed (e.g., [2, 10]). Most of them are in some way heuristically tailored to a concrete learning task in some concrete domain. A discussion of these various approaches is out of the scope of this paper. After various experiments, we found the following approach to work quite well (as indicated by our evaluation results), namely a weighted editing distance for Herbrand interpretations. This kind

[5] I.e., a profile is set simply by picking from the current models one or more atoms which hold in these models (and which must also be in the set P). Note that the previous profile became part of M in an earlier step.

of distance comes handy here, because we already need to deal with certain Herbrand interpretations, simply in form of the stable models delivered by the ASP solver. More precisely, we calculate the distance between state/action pairs as follows:

$$d((s,a),(\bar{s},\bar{a})) = \gamma(1 - \frac{|s \cap \bar{s}|}{max(|s|,|\bar{s}|)+\varepsilon} + \begin{cases} 0 & if\, a = \bar{a} \\ 1 & else \end{cases}) \quad (9)$$

At this, ε is some very small constant value in order to prevent division by zero. γ is a normalization factor which should be proportional to the size of the domain (in our experiments, we used $8.5n$ for a blocks world with n blocks).

Complexity issues with this regression mechanism will be discussed in Section 5 below.

5 Framework instantiation for concept learning

We will now take the previously introduced general algorithms and utilize them for interactive concept semantics learning. This instance algorithm is virtually entirely realized using formal rules in the EC. This is worth noticing, because it puts quite a lot of workload on the formal reasoner, that is, it motivates the use of the ASP solver. As mentioned earlier, in contrast to other types of reasoning (e.g., using Prolog), the focus is here on the computationally efficient computation of a potentially large number of satisfying models (interpretations) at each step of the learning process, which contain allowed agent and human actions at that time as well as possibly nondeterministic sets of candidate states after state updating actions. These models need to fulfil a relatively large number of constraints (formal rules).

More concretely, the formal context within which the previously presented core learning algorithms operate comprises 1) the specification of the concept domain (e.g., the blocks world), 2) the specification of rewards which the learning agent receives, 3) the specification of a communication protocol and any other constraints and rules regarding the behavior of the agent and the human interaction partner (including guidance), and 4) the formal semantics of already learned concepts, in order to utilize this knowledge in communication with the human interaction partner. Optionally, it is also possible to provide a simulation of human behavior in form of formal rules.
However, our framework do not impose any specific kind of formal context. Therefore, the following aspects can basically be varied without the need to adapt the fundamental framework.

As pointed out earlier, interactive concept semantics learning can be seen as an interactive, guided search task in the space of all possible concept domain instantiations - guided both on the side of the searching (learning) agent and on the side of the human user from whom the agent wants to know the semantics. The most recent search state is identified with the most recent Markov state, and (full) reward is given by the human in case the agent has successfully identified a state which corresponds to the (partial) semantics of the concept currently being learned.
At each step in the learning process, the agent either "breaks down" the search space or asks the human user a question (as simple as possible) about the unknown concept. This goes on until the agent has identified the correct denotation (a subset or a set of subsets of the domain). What is being learned are mainly 1) which questions to ask, and which sequence

of questions and other actions is optimal, 2) how to deal with different human interaction partners which use different meanings for the same word, or how to deal with a certain human who varies the meaning for the same word, and 3) how to utilize dynamically acquired knowledge about concepts (or single instances of concepts) which have already been learned for the learning of new concepts (or new instances of concepts). The latter point comprises both reinforcement learning and memorize and recall - the agent updates its KB with newly acquired denotations and learns by trial-and-reinforcement/error when to use already learned words in questions addressed to the human (e.g., "Is the wanted concept X similar to concept Y?"). Item 2) makes use of the *profiles* introduced earlier.

Concept instances do not need to exhaustively describe exactly a single certain domain state - instead, they can also denote sets of states (such as all blocks world configurations where the red block is on top of the green block) and, in case of predefined concepts, a meaning which is relative to the current state (comparatives such as "higher" or "lower") .

Note that the learning of a certain concept or concept instance might span over several episodes. At this, we implicitly assume that all instances of the same concept are somewhat similar, in order to make agent's questions useful if they refer to already learned concepts. Also, in case multiple different concepts are being learned in the same experiment (over multiple episodes), it is helpful for learning success if those concepts are also mutually similar to some degree, since the same Q-table is used for learning all these concepts (profiles do not help here). How such "similarity" can look like in practice will be explained in the next section (evaluation).

Further note that we do not name concepts but identify them by numbers, or more precisely, symbols of the form $C_{i,j}$, where i is the concept number and j is the number of the respective concept instance. While it would be trivial to enhance the agent-human dialogue with phrases such as "Learn the meaning of *word*!", within the scope of this work this would be a purely cosmetic "improvement" over the use of numerical concept and instance indices. However, if we were to learn the semantics of words which exist in real languages, it would of course make sense to expose those words to the learning agent, e.g., in order to retrieve further information about their meaning from other sources, such as the Internet.

Algorithm 6

$profile \leftarrow \{\}$
$i \leftarrow 0, j \leftarrow 0$ (i is the number of the currently learned concept, j will be the number of the next learned instance of concept i)
loop
  Specify start state (possibly empty or random): $s = fluent(m)$,
  with $M = models(kb \cup profile)$
  Start time: $t \leftarrow 0$
  repeat
    Perform action a and let interaction partner perform action ah, as described in Algorithm 4
    Update state s, kb and $profile$, and get reward, as described in Algorithm 4
    $t \leftarrow t + 1$
    if $ah = Happens(tell(happyI), t)$ **then**
      $kb \leftarrow kb \cup [C_{i,j} := os(s)]$ (add learned concept instance and its semantics (os) to agent's memory)

$j \leftarrow j+1$ (start learning a new instance of the currently learned concept)

else if $ah = Happens(tell(happyC),t)$ **then**
$kb \leftarrow kb \cup [C_{i,j} := os(s)]$ (add learned concept instance and its semantics (os) to agent's memory)
$i \leftarrow i+1$ (start learning a new concept)
end if
until $ah = Happens(tell(happy*),t)$ or maximum number of time steps is reached
end loop

The meaning of the symbols is as in Algorithm 4, except for the following:

- $os : S \rightarrow OS$ yields the non-interaction part of a state.

- PC is a number of helper concepts whose meaning is a priorly agreed among agent and human. They can be used to help directing the agent towards the wanted semantics (e.g., if the concept "lower" is in PC, the agent might ask "Is the wanted state lower than the current state?"), as explained above.

- $A = DA \cup \{Happens(ask(applies(p)),t), Happens(ask(akinTo(C_i)),t),$ $Happens(ask(akinTo(C_{i,j})),t) : p \in PC\}$. The agent can make use of *akinTo* questions to ask the human whether the concept is similar (or equal) to an already learned concept or concept instance. Sometimes, we will use $ask(p)$ as a shorthand notion for $ask(applies(p))$.
  At this, DA is the set of agent actions which directly affect the object domain (vs. the interaction state), such as stacking actions in the blocks world. In particular, we require that the agent knows how to perform special $goto(C_{i,j})$ and $ensure(p)$ actions. These actions are performed by the agent after a positive answer given by the human in response to questions of the form $akinTo(C_{i,j})$?, $akinTo(C_i)$?, or $applies(p)$?, respectively.
  With $goto(C_{i,j})$, the agent (that is, the concept search process) immediately jumps to (or "builds") a domain state which corresponds to concept instance $C_{i,j}$ (provided this concept instance has already been learned, of course).
  With $ensure(p)$, the agent transforms the state according to the predefined helper concept $p \in PC$. So what $ensure(p)$ does depends on p. As a simple example, a predefined "relative" blocks world concept $p =$"lower" could be implemented as a stacking action which puts some random block (which is currently not on the table) on the table, after the user gave the answer "yes" to the question $applies(lower)$?.

  However, the use of logical reasoning and in particular ASP allows for further reaching possibilities. In our evaluation, an $ensure(XonY)$ action (with X and Y being blocks) does not immediately stack block X on block Y, since this could be counter-productive in case later there also needs to be some block Z put underneath of Y. Instead, $ensure(XonY)$ ensures that *at the end* of the learning episode, X is on Y. This causes that all models computed by the ASP solver ensure a coherent sequence of actions leading from the current state to a state where block X is on block Y, including eventualities such as putting blocks below block Y. Technically, $ensure(XonY)$ adds a *planning goal* to the agent's knowledge base, and the reasoner automatically finds this plan, which is then executed by the learning agent, enforced by the normal constraining of agent actions using $E_s(M,t)$. Details on how this precisely works are out of the scope of this paper

and can be found in [14].

- $IA = \{< noAction >, Happens(tell(yes),t), Happens(tell(no),t),$
  $Happens(tell(dontKnow),t),$
  $Happens(tell(happyI),t), Happens(tell(happyC),t) : t \in [1..\text{max time}]\}$.
  $tell(happyI)$ is uttered by the human interaction partner after the agent has successfully identified one instance of the currently learned concept. The only difference between $happyI$ and $happyC$ is that in the latter case the human acknowledges that the entire concept (i.e., all its instances) has been learned by the agent. $< noAction >$ means the human does not say anything at the respective time step.

- $$E_s(M,t) = \begin{cases} \{goto(C_{i,j})\}, \text{ if } Happens(ask(akinTo(C_{i,j})),t-1) \in kb \\ \qquad \wedge Happens(tell(yes),t-1) \in kb \\ \{goto(C_{i,j})\}, \text{ if } Happens(ask(akinTo(C_i)),t-1) \in kb \\ \qquad \wedge Happens(tell(yes),t-1) \in kb \\ \qquad \text{with } j \text{ picked randomly from learned instances of } C_i \\ \{ensure(p)\}, \text{ if } Happens(ask(applies(p)),t-1) \in kb \\ \qquad \wedge Happens(tell(yes),t-1) \in kb \\ A \backslash \{Happens(ask(akinTo(C_i)),t) : \neg \exists j.[C_{i,j} := *] \in kb\} \\ \quad \backslash \{Happens(ask(akinTo(C_{i,j})),t) : [C_{i,j} := *] \notin kb\} \quad \text{otherwise} \end{cases}$$

  Remark: theoretically, rules such as these could also be learned (e.g., from examples). But since rule learning is out of the scope of this paper and covered by other approaches such as Inductive Logic Programming, we leave it out here.
- $EH_s(M,t) = \{< noAction >\}$ if the agent action at time t is from DA, and $IA \backslash \{< noAction >\}$ otherwise

- $R(\bar{s})$ should normally result in a large reward for interaction states reached with a $happyI$ or $happyC$ act emitted by the human interaction partner. In our experiments, we found it also useful to give a small positive intermediate reward for each $tell(yes)$ event and a small negative reward for $tell(dontKnow)$.

The SARSA-variant of this (Algorithm 7) is omitted in this paper - the difference compared to Algorithm 6 is completely analogous to the difference between Algorithms 4 and 5.

5.1 Rule specification

The next section provides extensive examples which illustrate these algorithms.

All the various formal aspects are uniformly and flexibly given in form of EC rules and facts in the learning agents knowledge base KB. We cannot present the entire set of rules used for the prototypical implementation, but just a few examples. Note that these rules are not formalized in Prolog, but as ASP rules (AnsProlog), which are semantically quite different despite their similar syntax.

Action alternatives are specified using disjunctions, for example:

Happens(tell(dontKnow),T) | *...* | *Happens(tell(happyC),T)*
: − *Happens(ask(...), T).*

Happens(stack(a,table),T+1) | *...* | *Happens(stack(g,table),T+1))*
: − *Happens(ask(isLower), T)*, *Happens(tell(yes), T).*

Rewards (being mere fluents in our framework) are specified using ordinary rules too, e.g.:

HoldsAt(reward(-10),T) : − *Happens(tell(dontKnow),T).*

All event handling makes use of EC axioms, e.g.:

Initiates(stack(X,Y),on(X,Y),T)
: − *T* < *maxtime, not HoldsAt(on(X,Y),T).*

The informal meaning of the last rule is that the fluent "block X is on top of block Y" starts to hold after the stacking action *stack(X,Y)* has been performed at time T, provided block X is not yet on top of block Y at this time. X, Y and T are variables which are instantiated with any block or time step, respectively.

5.2 Efficiency issues

A note on the time complexity of the presented algorithms: as with reinforcement learning in general, reaching a goal state can take a number of agent actions (time steps) which is exponential in the number of states. Specific to RRL, as instantiated in our algorithms, is the complexity of ASP solver calls for state updates and action set computations, and the complexity of the optional regression mechanism (Section 4.3.1). As for solver calls, it is difficult to state their precise complexity, since it depends on the concrete ASP solver which is used (that is, the solving algorithm), its configuration, and various model search heuristics applied by this solver. Generally, the downside of all logic-programming approaches is that computing stable models (answer sets) is NP-hard and $\sum_2^P$ if there are disjunctions in rule heads [3], which is a typical case in our scenario in order to model nondeterminism. The upside is that ASP-solvers are optimized to deal with this high complexity and that ASP does not have Prolog's non-termination issues. Also, the high complexity corresponds to high expressiveness, e.g., representability of problems which cannot be polynomially reduced into SAT. As for the concrete base rule set used in our framework, there is currently no faster approach to typical benchmark problems for the (discrete) EC than ASP [39].
As for the instance-based regression mechanism described in Section 4.3.1, prediction of a single Q-value from all existing Q-values (using a k-nearest-neighbor approach) can become problematic for large state/action spaces. On the other hand, learning using this approach typically requires much less state/action examples and actually improves scalability, so this issue is mitigated (it wasn't any problem in our experiments). Otherwise, several means exist in order to reduce the inflow of state/action pairs [2] which could be easily added to our algorithms. All in all, instance-based regression usually performs comparably well as the much more complicated use of relational kernels and performs generally better than the classic approach to RRL (*TILDE*) [2]. In our own experiments (cf. next section), use of regression outperformed the non-generalizing variants of our algorithms.

6 Evaluation

To illustrate the potential of our learning framework in the domain of learning semantics, we have picked two concept learning tasks for empirical experiments. First, various dimensions of the learning of simple concept semantics are investigated (see Sec. 6.1), namely learning of different concepts (see Sec. 6.1.1), scalability issues (see Sec. 6.1.2), analysis of communication actions (see Sec. 6.1.3) and non-stationarity (see Sec. 6.1.4). Then, a scenario with more complex concepts is investigated (see Sec. 6.2).

At this, our intention is to give a proof-of-concept by showing how our approach can learn in a range of different problems using a uniform learning approach, rather than comparing its performance to other, more specialized and thus restricted algorithms.

6.1 Learning of simple concept semantics

As a first series of experiments, we present a relatively simple learning task situated in the blocks world domain (this is basically the same task as the motivating example presented in Section 3). The agent's goal is to find out the denotation of the term "nice building", as conceptualized by users. To achieve this, the agent can in each step either move one block, or ask a question and the user then answers these questions (although not necessarily correctly or in a consistent fashion). In our initial experiments, we simulate the behavior of the interaction partners to avoid manual training. Here, two users are simulated: The first user wants the agent to build a "nice building" which for this user means arranging a number of blocks next to each other on the table. We denote this task *lowBuild*. For the other user a "nice building" is achieved by building a tower with all blocks on top of each other. We denote this task *highBuild*.

Each reinforcement learning episode starts with blocks randomly placed on top of each other. Then, in each time step the agent can rearrange one block or in turn ask questions such as *ask(isHigher)* (to build a higher building), *ask(isXonY)* (to put block X on Y, using a subset of block combinations). Potential user responses can be *tell(yes)*, *tell(no)*, *tell(dontKnow)*, and *tell(happy)*. The simulated user truthfully answers "yes" and "no" when being asked "isHigher" or "isLower", and "dontKnow" otherwise, until the desired building is achieved by the agent (answer "happy"). In this series of experiments, we use Algorithms 6/7, but without the possibility to learn more than one concept and without memorization of previously learned concepts.

For a *tell(dontKnow)* the agent receives a small punishment (-0.1), for a *tell(yes)* a small reward (0.2) and a large reward (1.0) if the interaction partner responds that the desired blocks world constellation was reached ("happy"). The maximum number of steps per epoch is up to 20. The overall goal is on the one hand to learn to ask the right questions in the right sequence and on the other hand to learn the meaning of "nice (building)" and to "build" it. Note that at the end of a successful learning process, the final state contains a formal representation of the learned concept, i.e., it's formal meaning (e.g., a blocks world constellation where all blocks are on the table).

6.1.1 Learning high vs. low buildings

We ran Algorithm 7 (i.e., the SARSA variant of Algorithm 6) on this setup for 10 trials with 500 episodes each and monitored the sum of rewards and number of steps performed in each episode. Relational Instance Based Regression (RIB), a form of generalization (see Section

4.3.1) was activated and an ε-greedy action selection strategy has been used (with an ε of 0.2, as explained in Definition 3). Fig. 3 shows the average reward over all trials for the task *highBuild* and Fig. 4 the averaged steps required to reach a reward of 1 or larger. Similarly, Fig. 5 shows the average reward over all trials for the task *lowBuild* and Fig. 6 the averaged steps. In both cases the numbers of blocks are set to 7.

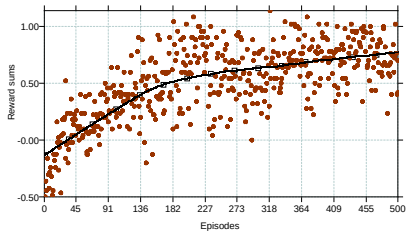

Fig. 3 Learning of task *highBuild* with 7 blocks: Rewards (y-axis) over epochs (x-axis); Dots show average over 10 trials. Polynomial curves are fitted to the dots.

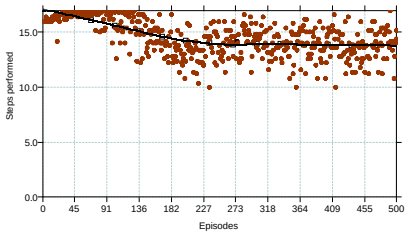

Fig. 4 Learning of task *highBuild* with 7 blocks: Steps (y-axis) over epochs (x-axis); Dots show average over 10 trials. Polynomial curves are fitted to the dots.

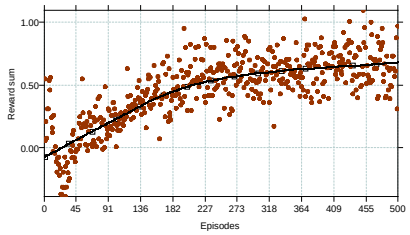

Fig. 5 Learning of task *lowBuild* with 7 blocks: Rewards (y-axis) over epochs (x-axis); Dots show average over 10 trials. Polynomial curves are fitted to the dots.

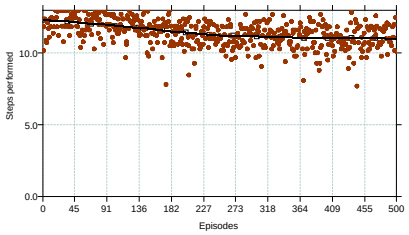

Fig. 6 Learning of task *lowBuild* with 7 blocks: Steps (y-axis) over epochs (x-axis); Dots show average over 10 trials. Polynomial curves are fitted to the dots.

The results clearly indicate that in both setups the agent learns to build a "nice building" after fewer steps with an increasing number of training epochs. The task *highBuild* seems to be harder to learn than *lowBuild* in terms of steps needed. We attribute this to the fact that finding *lowBuild* at random is more likely. More steps also means more chances to collect reward which explains the higher level of reward of *highBuild* in the end.

6.1.2 Learning 4 vs. 7 vs. 10 block buildings

To test how the performance reacts to the scale of the domain, we changed the number of blocks and repeated the experiments.

In our baseline setting, we tried to learn the task *lowBuild* where the size of the building consists only of 4 blocks. The number of rules for this problem was about 60, including

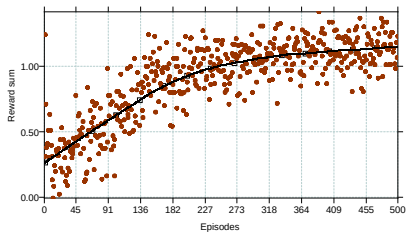

Fig. 7 Learning of task *lowBuild* with 4 blocks: Rewards (y-axis) over epochs (x-axis); Dots show average over 10 trials. Polynomial curves are fitted to the dots.

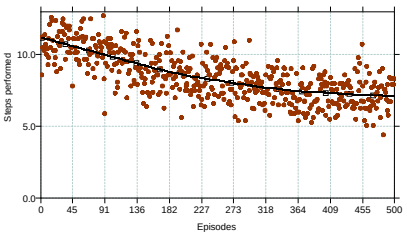

Fig. 8 Learning of task *lowBuild* with 4 blocks: Steps (y-axis) over epochs (x-axis); Dots show average over 10 trials. Polynomial curves are fitted to the dots.

12 Event Calculus axioms (please note: the sheer number does not tell much about the reasoning complexity. The number of blocks is much more important here).
As before, we ran this 4 blocks setup for 10 trials with 500 episodes each and monitored the sum of rewards and number of steps performed in each episode. Fig. 7 shows the average reward over all trials and Fig. 8 the averaged number of steps required. Again, the results clearly indicate that the agent learns to build a "nice building" after fewer steps with an increasing number of training epochs.

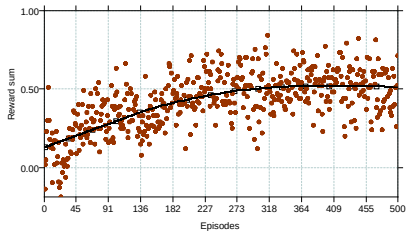

Fig. 9 Learning of task *lowBuild* with 10 blocks: Rewards (y-axis) over epochs (x-axis); Dots show average over 10 trials. Polynomial curves are fitted to the dots.

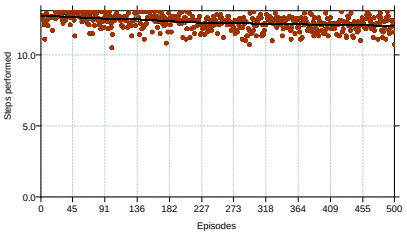

Fig. 10 Learning of task *lowBuild* with 10 blocks: Steps (y-axis) over epochs (x-axis); Dots show average over 10 trials. Polynomial curves are fitted to the dots.

In addition to the 7 and 4 blocks scenario we tried to scale our experiments to 10 blocks which increases the reasoning complexity considerably. A blocks world with 10 blocks has around 58 million states which means it is extremely hard to tackle with regular Q-learning. Again, we ran this setup for 10 trials with 500 episodes each and monitored the sum of rewards and number of steps performed in each episode. Fig. 9 shows the average reward over all trials and Fig. 10 the averaged number of steps required. Although considerably slower, the results show again that the agent learns to build a "low building" more often with an increasing number of training epochs.

In order to illustrate the difference Fig. 11 shows the rewards over all setups and Fig. 12 the number of steps required. Compared to the 7 block setup the 4 blocks setup is considerably easier, which is as expected. More interesting is the observation that 10 blocks seems not to be that much harder compared to 7 blocks. Thus, a linear increase in blocks, which means an exponential increase in blocks world constellations, does result in a sub-linear

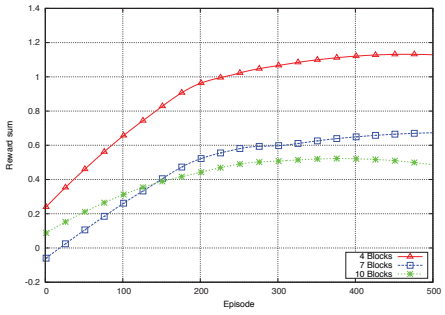

Fig. 11 Comparison of task *lowBuild* with 4 vs 7 vs 10 blocks: Rewards (y-axis) over epochs (x-axis).

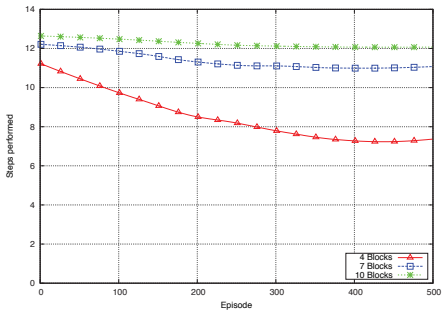

Fig. 12 Comparison of task *lowBuild* with 4 vs 7 vs 10 blocks: Steps (y-axis) over epochs (x-axis).

decrease in learning efficiency. This is an encouraging result as this indicates that communication and reasoning seems to be increasingly beneficial with increasing complexity.

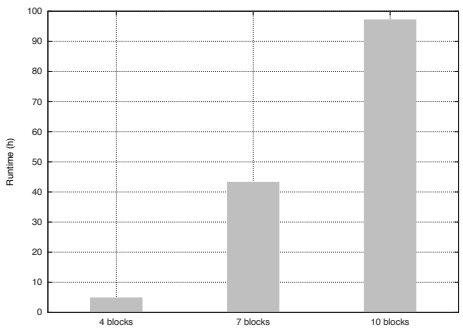

Fig. 13 Comparision of task *lowBuild* with 4 vs 7 vs 10 blocks: Runtime (y-axis) over blocks (x-axis), all trials.

Finally, one has to consider that an increase in blocks also increases the complexity of reasoning as the number of blocks world constellations grows exponentially. To investigate this, Fig. 13 shows the number of hours needed to run a full experiment with 500 episodes and 10 trials.

6.1.3 Learning to ask vs. to stack

So far we could prove that our approach can indeed learn the semantic of the concept “nice building” with different meaning. While the good scalability might indicate that our communication framework has a beneficial effect for large domains, those experiments do not show to which amount and effect the agent uses communication to ease the task. To investigate the extend to which communication helps, we compared our scenario with a setup where communication actions do not trigger any valuable response by the simulated communication partner (denoted *No_Interposed_Questions*). To avoid an advantage of the setup used so far with helpful responses, no intermediate rewards where given for answers like *tell(yes)* (denoted *No_Intermediate_Rewards*) in both setups. This way there is no bias in the reward-baseline and the results purely show the effect of mutual communication. In this

scenario we chose 5 blocks as this was a convenient compromise between complexity of the task and runtime of the experiments.

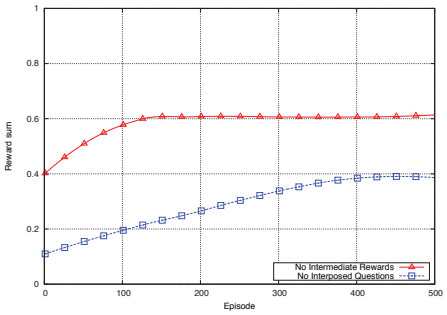

Fig. 14 Comparison of task *lowBuild* (5 blocks) with (*No_Interposed_Questions*) vs without (*No_Intermediate_Rewards*) communication: Rewards (y-axis) over epochs (x-axis). Numbers are first averaged over 10 trials and then fitted with polynomial curves.

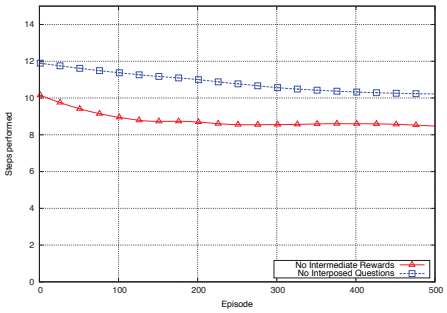

Fig. 15 Comparison of task *lowBuild* (5 blocks) with (*No_Interposed_Questions*) vs without (*No_Intermediate_Rewards*) communication: Steps (y-axis) over epochs (x-axis); Numbers are first averaged over 10 trials and then fitted with polynomial curves.

Fig. 14 shows the reward and Fig. 15 the number of steps required for both setups. Both *No_Intermediate_Rewards*-curves are clearly better than *No_Interposed_Questions* which proves the positive effect of communication. The agent not only learns quicker with communication, but also has an advantage right from the beginning. This is due to a reduction of the set of potential models that need to be explored, as soon as just one question has been asked (randomly) and answered. Thus, the chances of finding the right building are clearly higher with communication.

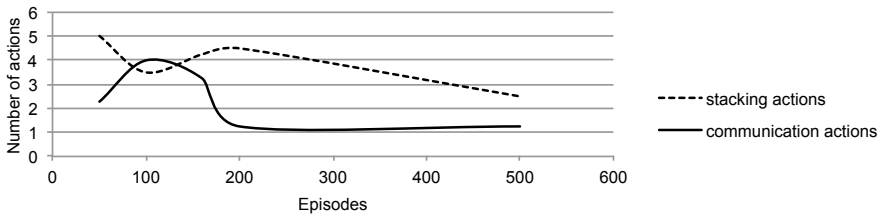

Fig. 16 Number of actions (y-axis) performed over 500 episodes (x-axis). Dashed curves shows number of stacking actions and solid curve communication action. Counted during learning of task *lowBuild* with 4 blocks.

In a final set of experiments we investigated the number of communication acts vs. the number of stacking actions performed by the agent while learning task *lowBuild* with 4 blocks (cmp. Fig. 7 and Fig. 8). With this evaluation, we want to test if communication is used differently over time. Fig. 16 shows the number of actions (y-axis) performed over 500 episodes. Apparently, the agent learns in the first 100 episodes to ask questions about the shape of the tower and then increasingly us this knowledge to learn the best strategy to build this tower. After about episode 200, communication is not needed anymore and most

actions are used to build the correct structure. This behavior appears to be an intuitively optimal strategy to solve this task.

6.1.4 Learning to deal with non-stationarity

As a second series of experiments on the baseline task, we tested the effect of *profiles* (cf. Section 4) as part of Markov states in order to improve non-stationarity handling (Fig. 17).

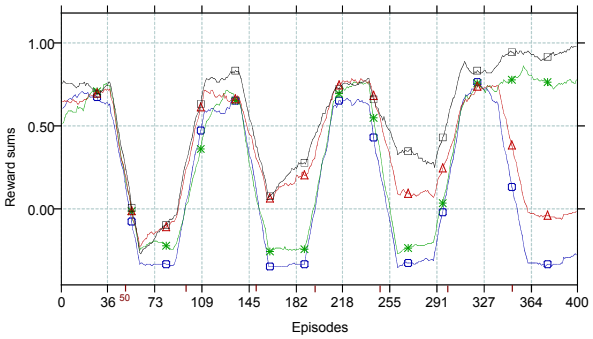

Fig. 17 Non-stationarity handling in the 'nice building' domain. Black/squares: using Algorithm 7. Red/triangles: Algorithm 7 with profiles deactivated. Green/stars: finite window, size 100. Blue/circles: finite window, size 50. (Color figure online)

We conducted four experiments: in these experiments, the human user does not have only a single preference, but switches back and forth between two different semantics of the word "nice" within the same trial. At the end of each episode it is determined from the user's communication whether (s)he has preferred a "high building" or a "low building" in this episode. Of course, instead of a single interaction partner, this fluctuating semantics could likewise be produced by several users with different, mutually-inconsistent conceptualizations for the word "nice".

In our simulation, a certain user preference is maintained for 50 episodes and changes then from "high building" to "low building" or vice versa.

The respective preference is added to the profile, which carries this information across episodes and sections the Q-table. We reduced the maximum number of time steps to 11, in order to make learning more difficult.

In the first experiment, again Algorithm 7 is used. In the second experiment, the same algorithm is used, but profiles are omitted from the Q-table. In the third experiment, a so-called *finite window* approach (a standard approach to non-stationarity handling in RL) is used: at each learning step, only the most recent 100 entries in the Q-table are maintained, older entries are deleted. Experiment four does the same but with a window size of 50. The results are depicted in Fig. 17. They show a significant performance gain by use of the profile-based approach over both finite window approaches. The finite window approach with size 100 eventually performs well too (with a extraordinary jump upwards near the end), but not during most of the episodes.

6.2 Learning of more complex concept semantics

To move towards a more realistic scenario, we generalized the "nice building" task (Section 6.1) to the "complex towers" task. This task again uses Algorithms 6/7 (and again the SARSA version was used to achieve the results), and includes the memorization and re-use of already learned concepts and concept instances, making it much more realistic than the "nice building" scenario.

In each trial, the learning agent is supposed to learn 10 unknown concepts. Each concept is represented by four instances, each instance being one concrete but random arrangement of blocks (with a total number of four blocks). Each instance is randomly generated at the beginning of each trial. E.g., an instance might be (informally) "blockC is on blockB, blockB is on blockA, blockA is on the table, blockD is also on the table".

All instances of the same concept are similar to each other (they need to be identical in at least three ground atoms (out of five) to one instance of the same concept). Furthermore, we require that each instance must be similar to at least one other concept (in 2.5 ground atoms on average over all instances of the other concept). This is required in order to make multiple concepts reasonably learnable using a single, moderate-length reinforcement learning task. On a side note, this also demonstrates that the learning approach has a certain transfer capability.

Concepts are learned in random order, but learning of a new concept only starts after all instances of the previous concept have been successfully learned. Also the instances representing each concept are learned in an arbitrary order. The learning of a certain concept instance can take several episodes which do not have to be one after another.

In its questions to the user, the agent can use the predefined comparatives "isLower?", "isHigher?", "isXonY?", but also questions which refer to just learned concepts and instances, namely $ask(akinToC_i)$ and $ask(akinToC_{i,j})$. On a positive reply to the latter two questions, the agent recalls and activates a previously learned concept instance. The user answers with *tell(yes)*, *tell(no)*, *tell(dontKnow)*, or with *tell(happyI)* or *tell(happyC)* (an instance or a concept has been recognized). Please refer to Section 5 for the detailed meaning of these utterances. It is particularly noteworthy that a positive answer to the *ask(isXonY)* questions lead to an automated planning process, described in the scope of Algorithm 6.

"happyI" or "happyC" carry a reward of 2.0. After all 10 concepts have been fully learned (i.e., for each of them, all its four instances have been learned), for the remaining episodes of the trial the maximum single reward (2.0) is given.

We run two experiments with this scenario: with and without the use of Relational Instance Based Regression (RIB). Fig. 18 shows the development of gained rewards per episode over 500 episodes and 10 trials, with each episode lasting maximally 13 time steps (or until a reward sum of at least 2 has been reached), using Algorithm 7 with RIB deactivated (a default Q-value of 0.1 was used for unknown state/action examples). Fig. 19 shows the number of required steps. Figs. 20 and 21 show rewards and step numbers for the same setup, but with RIB active.
While "complexTowers" are learned quite fast even without RIB, the gain from using generalization is very significant. The likely reason why generalization works so well here is that multiple *similar* concepts are being learned in the same trial. Even without experimentation, we can safely say that without any concept similarities, reinforcement learning across multiple concepts would be seriously hampered or impossible, since experience could not be

reused (but remember that two sorts of learning are happening with Algorithms 6/7: RRL plus learning by memorizing and recall of already learned concepts).

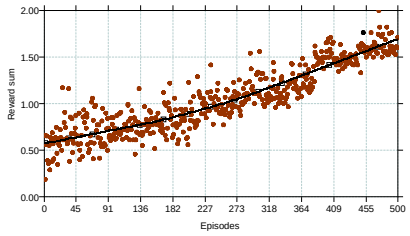

Fig. 18 Learning of task *complexTowers*: Rewards (y-axis) over epochs (x-axis); Dots show average over 10 trials. Polynomial curves are fitted to the dots.

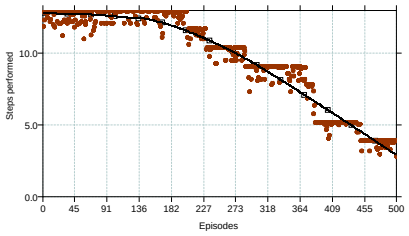

Fig. 19 Learning of task *complexTowers*: Steps (y-axis) over epochs (x-axis); Dots show average over 10 trials. Polynomial curves are fitted to the dots.

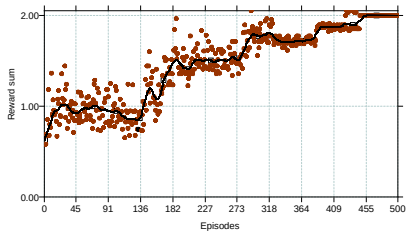

Fig. 20 Learning of task *complexTowers* with generalization: Rewards (y-axis) over epochs (x-axis); Dots show average over 10 trials. Polynomial curves are fitted to the dots.

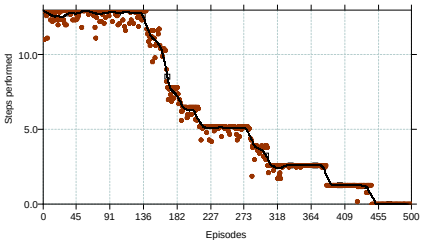

Fig. 21 Learning of task *complexTowers* with generalization: Steps (y-axis) over epochs (x-axis); Dots show average over 10 trials. Polynomial curves are fitted to the dots. Note that the number of steps approaches zero, because in this experiment, several concepts are learned over an entire trial - as soon as all concepts have been successfully learned, there remains nothing to do for the agent and maximum reward is obtained.

7 Conclusion

We have presented an approach to Relational Reinforcement Learning for the *interactive* learning of concept semantics in a dialogical setting, with a focus on the integration of Q-/SARSA-Learning on the one hand and EC/ASP-based formal reasoning about the state of interaction and the set of possible actions on the other. RRL is still a relatively young research direction, and we believe that such a hybrid approach of combining classic approaches to RL with rich yet computationally efficient relational representation formats and temporal reasoning capabilities is most promising in order to adequately model complex learning domains.

An extensive empirical evaluation has been conducted. Results are mostly positive and indicate the applicability and significance of the presented learning framework for realistic semantic learning tasks. We could show that a) the agent can learn different meanings of a concept, b) scale to different levels of complexity, including a very ambitious 10 blocks block world, c) use communication to ease the task, d) deal with non-stationarity and e) more complex scenarios.

However, our experiments leave room for more complex use cases and experimental setups. In particular, we plan to research next to what extent the concept semantics learning strategies facilitated by our framework can be applied in real-world scenarios, and which modifications need to be applied in order to facilitate such use, if any. Applications of interest are, e.g., interactive information retrieval on the Web and interactive concept and ontology learning from a large number of users on the Web. Furthermore, we are currently investigating the use of alternative approaches to RRL, such as relational policy gradient approaches, in our framework.

Acknowledgements. This work was partially supported by Deutsche Forschungsgemeinschaft (DFG).
We would like to thank the reviewers for their very helpful and detailed comments.

References

1. S. Dzeroski, L. De Raedt, H. Blockeel: Relational reinforcement learning. Procs. ICML'98. Morgan Kaufmann, 1998.
2. K. Driessens: Relational Reinforcement Learning. PhD thesis, Katholieke Universiteit Leuven, 2004.
3. T. Eiter, G. Gottlob: On the Computational Cost of Disjunctive Logic Programming: Propositional Case. In Annals of Mathematics and Artiicial Intelligence 15(3-4), p. 289-323, 1995.
4. E. Levin, R. Pieraccini: A stochastic model of computer-human interaction for learning dialog strategies. In Procs. of the European Conference on Speech Communication and Technology (Eurospeech), p. 1883-1886, 1997.
5. M. Ponsen, T. Croonenborghs, K. Tuyls, J. Ramon, K. Driessens: Learning with whom to communicate using relational reinforcement learning. Procs. AAMAS'09, 2009.
6. M. Richardson, P. Domingos: Markov Logic Networks. In Machine Learning, 62 (2006), pp 107-136.
7. T. Yamaguchi, T. Nishimura, K. Sato: How to Recommend Preferable Solutions of a User in Interactive Reinforcement Learning? Procs. SICE Annual Conference, 2008.
8. G. Arnold-Dulac, L. Denoyer, Ph. Preux, P. Gallinari: Datum-wise classification. A Sequential Approach to Sparsity in Machine Learning and Knowledge Discovery in Databases. Procs. European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD 2011). Springer, 2011.
9. A. Bordes, N. Usunier, R. Collobert, J. Weston: Towards Understanding Situated Natural Language. Procs. International Conference on Artificial Intelligence and Statistics (AISTATS), 2010.
10. Ch. Rodrigues, P. Gerard, C. Rouveirol: Relational TD Reinforcement Learning, Procs. EWRL'08, 2008.
11. D. Goldwasser, D. Roth: Learning from Natural Instructions. Proceedings of International Joint Conference of Artificial Intelligence, 2011.

12. J. Gmytrasiewicz, P. Doshi: Interactive POMDPs: Properties and Preliminary Results. Procs. AAMAS'04, 2004.
13. J. MacGlashan, M. Babes-Vroman, K. Winner, R. Gao, M. desJardins, M. Littman, S. Muresan: Learning to Interpret Natural Language Instructions. In Procs AAAI-2012 Workshop on Grounding Language for Physical Systems, 2012.
14. V. Lifschitz: Answer Set Planning. In D. D. Schreye (ed.), Procs. of the 16th International Conference on Logic Programming (ICLP?99), pp. 23?37. The MIT Press, 1999.
15. M. Nickles: Integrating Relational Reinforcement Learning with Reasoning about Actions and Change. Proceedings of the 21st International Conference on Inductive Logic Programming (ILP 2011). Springer LNAI, 2012.
16. W. B. Knox, P. Stone: Augmenting Reinforcement Learning with Human Feedback. Procs. ICML 2011 Workshop on New Developments in Imitation Learning, 2011.
17. M. Ghavamzadeh, S. Mahadevan: Learning to Communicate and Act Using Hierarchical Reinforcement Learning. Procs. AAMAS'04, 2004.
18. J. Méhat, T. Cazenave: Combining UCT and Nested Monte Carlo Search for Single-Player General Game Playing. In IEEE Transactions on Computational Intelligence and AI in Games 2(4): 271-277, 2010.
19. M. Nickles, A. Rettinger: Towards Interactive Relational Reinforcement Learning of Concepts. Abstract paper presented at the "Learning Semantics" Workshop at the 25th Annual Conference on Neural Information Processing Systems (NIPS 2011), 2011.
20. R. Navigli: Word Sense Disambiguation: A Survey. ACM Computing Surveys, 41(2), 2009.
21. D. L. Chen, R. J. Mooney: Panning for Gold: Finding Relevant Semantic Content for Grounded Language Learning. In Procs. MLSLP, 2011.
22. L. Steels: Grounding Symbols through Evolutionary Language Games. In A. Cangelosi, D. Parisi: Simulating the Evolution of Language. Springer, 2001.
23. W. Kerr, P. R. Cohen, Y.-H. Chang: Learning and Playing in Wubble World. Procs. AIIDE, 2008.
24. R. Mihalcea: Unsupervised Large Vocabulary Word. Sense Disambiguation with Graph-based. Algorithm for Sequence Data Labeling. Procs. of the Conference on Human Language Technology and Empirical Methods in Natural Language Processing (HLT'05), 2005.
25. R. Reiter: The frame problem in the situation calculus: a simple solution (sometimes) and a completeness result for goal regression. In Vladimir Lifshitz (ed.), Artificial intelligence and mathematical theory of computation: papers in honour of John McCarthy, San Diego, USA. Academic Press Professional, 1991.
26. M. Van Otterlo: The Logic of Adaptive Behavior. IOS Press, Amsterdam, 2009.
27. H. Cuayahuitl: Hierarchical Reinforcement Learning for Spoken Dialogue Systems. Ph.D. Thesis, School of Informatics, University of Edinburgh, 2009.
28. L. Specia et al: Word Sense Disambiguation Using Inductive Logic Programming. In Selected papers from the 16th International Conference on Inductive Logic Programming, Springer, 2007.
29. T. Croonenborghs, J. Ramon, M. Bruynooghe: Towards informed reinforcement learning. Procs. of the Workshop on Relational Reinforcement Learning at ICML'04, 2004.
30. L. Getoor, B. Taskar, eds. Introduction to Statistical Relational Learning. MIT Press, 2007.
31. K. Van Belleghem, M. Denecker, D. De Schreye: On the relation between situation calculus and event calculus. Journal of Logic Programming Vol. 31 (1-3), pp. 3-37. Elsevier, 1997.

32. M. Shanahan: A Circumscriptive Calculus of Events. Artificial Intelligence, 1995: 249-284, 1995.
33. T.-W. Kim, J. Lee, R. Palla: Circumscriptive event calculus as answer set programming. Procs. IJCAI'09, 2009.

A Algorithm 5 (Interactive ASP-supported Relational SARSA-learning)

loop
  Specify start state $s = fluents(m)$,
  with $M = models(kb \cup profile)$
  Start time: $t \leftarrow 0$
  Retrieve possible actions: $A' \leftarrow E_s(M, t+1)$
  $a \leftarrow \pi^{profile}(s, A')$
  Memorize the most recent action of the learner: $la \leftarrow a$
  repeat
    Perform action a
    Retrieve new state:
      Provide guidance: Let partner choose action ah solely from $EH_s(M,t)$
      Let interaction partner perform ah
      $kb \leftarrow kb \cup Happens(a,t+1) \cup Happens(ah,t+1)$
      $M \leftarrow models(kb \cup profile)$
      if M empty (i.e., $kb \cup profile$ unsatisfiable) **then**
        $profile \leftarrow \{\}$
        $M \leftarrow models(kb)$
      end if
      $\bar{s} = fluents(M, t+1)$
      if M is nondeterministic wrt. the next state **then**
        $kb \leftarrow kb \cup \bar{s}$ (fix the choice of the new state in the knowledge base)
      end if
    Get reward: $r \leftarrow R(\bar{s})$
    $profile \leftarrow \{atom \in model, model \in M\} \cap P$ (set new profile from recent set of stable models)
    $\bar{a} \leftarrow \pi^{profile}(s, E_s(M, t+1))$
    if $\bar{s}$ not a goal state **then**
      $Q^{profile}_{Rel}(s, la) \leftarrow r + \gamma Q^{profile}_{Rel}(\bar{s}, \bar{a})$ (learn)
    else
      $Q^{profile}_{Rel}(s, la) \leftarrow r$ (learn)
    end if
    $s \leftarrow \bar{s}$
    $a \leftarrow \bar{a}$
    if a was an action of the learning agent **then**
      $la \leftarrow a$
    end if
    $t \leftarrow t + 1$
  until s is a goal state, or some maximum reward sum is exceeded, or some maximum number of time steps is reached
end loop