User Interactions and Negative Examples to Improve the Learning of Semantic Rules in a Cognitive Exercise Scenario*

Alejandro Suárez-Hernández[1], Antonio Andriella[2], Carme Torras[1] and Guillem Alenyà[1]

***Abstract*— Enabling a robot to perform new tasks is a complex endeavor, usually beyond the reach of non-technical users. For this reason, research efforts that aim at empowering end-users to teach robots new abilities using intuitive modes of interaction are valuable. In this article, we present *INtuitive PROgramming 2* (INPRO2), a learning framework that allows inferring planning actions from demonstrations given by a human teacher. INPRO2 operates in an assistive scenario, in which the robot may learn from a healthcare professional (a therapist or caregiver) new cognitive exercises that can be later administered to patients with cognitive impairment. INPRO2 features significant improvements over previous work, namely: (1) exploitation of negative examples; (2) proactive interaction with the teacher to ask questions about the legality of certain movements; and (3) learning goals in addition to legal actions. Through simulations, we show the performance of different proactive strategies for gathering negative examples. Real-world experiments with human teachers and a TIAGo robot are also presented to qualitatively illustrate INPRO2.**

I. Introduction

Socially Assistive Robots (SARs) are intended to help humans via social interactions. Among the different fields in which these robots have been employed, cognitive training therapy is one area of interest in which SARs have proven their potential for enhancing therapists' effectiveness and lessening their workload [1]. A recent study, by Andriella *et al.* [2], describes a novel personalised framework embedded in a social robot to provide tailored assistive behaviour to Persons with Dementia (PwDs) while playing cognitive training exercises as part of their daily therapy.

One limitation of this approach is that the initial set of cognitive exercises may be limited in terms of type and complexity. Indeed, the ability to tailor the exercise to the patient is key to making the therapy effective and keeping them motivated. The most straightforward approach for introducing new exercises relies on hand-crafting a description of the rules of the exercise in a formal language. However, this approach would require HRI experts to manually code new exercises, since healthcare professionals do not have the technical skills to extend the robot's repertoire of exercises by themselves. Therefore, novel ways to intuitively program new exercises seem necessary to make robots more accessible to non-experts [3].

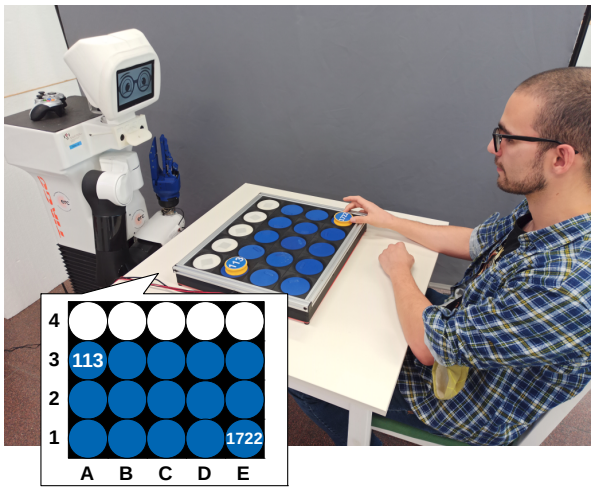

Fig. 1. Teaching an exercise consisting in moving the odd numbers horizontally to the rightmost column, and the even numbers vertically to the topmost row. Each step, a number must move to an adjacent cell.

To address this research gap, we propose a novel framework that allows therapists to teach the robot new exercises via demonstrations (Fig. 1). In previous approaches, this robot-caregiver interaction was addressed through the *INtuitive PROgramming* (INPRO) framework [4], [5] for learning STRIPS [6] action schemata that encode the rules of the exercise in their preconditions.

In the present article, we propose INPRO2, a learning algorithm that features significant improvements over INPRO. Unlike INPRO, INPRO2 relies on *Online Action Recognition through Unification* (OARU) [7], an algorithm for learning and recognizing STRIPS action schemata from a stream of symbolic states given by the teacher (the caregiver for our use case).

Our contributions are: **(1)** the use of OARU in INPRO2 allows the robot to learn complex exercises without a priori knowledge of the number of schemata nor their parameters; **(2)** the ability to learn from forbidden state transitions (negative examples) to fix overly permissive preconditions; **(3)** learning goals, in addition to legal moves; and **(4)** proactive interventions in critical moments of the teaching process, asking for the legality of certain actions to potentially dis-

*This work was partially funded by the EU H2020 Programme under grant agreement ERC–2016–ADG–741930 (CLOTHILDE); by MCIN/ AEI /10.13039/501100011033 under the project CHLOE-GRAPH (PID2020-119244GB-I00); by MCIN/ AEI /10.13039/501100011033 and by the "European Union NextGenerationEU/PRTR under the project COHERENT (PCI2020-120718-2); AA acknowledges the EU H2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 801342 (Tecniospring INDUSTRY); ASH acknowledges the financial support of the Apadrina la Ciencia association and of the Ford Spanish branch.

[1]Authors are with Institut de Robòtica i Informàtica Industrial, CSIC-UPC, Parc Tecnològic de Barcelona. C/ Llorens i Artigas 4-6, 08028, Barcelona, Spain {asuarez, torras, galenya}@iri.upc.edu

[2] A. Andriella is with Pal Robotics, C/ de Pujades, 77, 08005 Barcelona, Spain. antonio.andriella@pal-robotics.com

cover negative examples and learn from them.

While INPRO2 can work with unembodied devices, several studies indicate that robots, because of their embodiment, favor a better social response [8], [9], [10]. With INPRO2, we aim at empowering therapists with the ability to teach new personalised exercises to the robot, making existing applications [1] more suitable for long-term therapies.

This article is structured as follows: Section II analyses related literature; Section III presents concepts and notations required to understand the problem formulation and methodology, presented respectively in Section IV and Section V; Section VI contains a quantitative evaluation of our framework, as well as the report of some real-world qualitative experiments with a TIAGo robot; and, finally, Section VII shows our conclusions and future work.

II. Related Work

Martínez *et al.* use *Reinforcement Learning* (RL) to learn actions with stochastic effects [11] with the help of teacher demonstrations. Unlike INPRO2, this approach requires action signatures (i.e. the number of available actions and their parameters) to be given. *MuZero* [12] has been used to learn games like chess and go. Notwithstanding its ability to learn both the rules of the game and play skillfully, it is not feasible for a caregiver to provide the amount of required data.

Inverse RL (IRL) [13], [14] makes use of the teacher demonstrations to learn the reward of the different actions under different circumstances. This can be used in principle to distinguish between legal and illegal moves (actions with high and low reward, respectively). While this method is not as data intensive as MuZero, it requires a non-empty action set. Another disadvantage is that it does not provide precise preconditions (as the STRIPS formalism does), and thus lack INPRO and INPRO2's explainability.

Algorithms for inductive learning of high-level action models [15], [16], [17] observe interactions with the environment, and refine planning operators to match the observations. System-centric algorithms like ARMS [18] generate deterministic planning operators with weighted MAX-SAT solvers. SLAF [19] learns from partial observations. FAMA [20] computes STRIPS operators from minimal observations using classical planning. UDAM [21] uses a similar approach to FAMA, but removes the limitation of requiring action signatures. INPRO [5] was inspired by UDAM. However, INPRO2 uses OARU [7], which is based on a notably different principle of clustering actions more suitable for online learning and recognition.

Senft *et al.* [22] and Efthymiou *et al.* [23]'s motivation is reminiscent of ours. In particular, they empower adult educators to design the behavior of a robotic tutor for children. Winkle *et al.* [24] seek to allow the robot's behavior to be tailored to the user needs in a SAR context for a coaching use case. These methods offer a different perspective of the learning challenge: they are adequate for learning how to interact, but not to learn precise logical descriptions, like INPRO2 for board exercises. It is also worth mentioning the work of Causo *et al.* [25]. Like in our cognitive exercise use case, teacher robots could benefit from intuitive ways of being programmed for giving new lessons.

III. Background

Let us start introducing some concepts from **First-Order Logic** (FOL), **STRIPS** and the **OARU algorithm**.

A. First Order Logic

We denote as $\mathcal{D}$ the *domain of discourse*, a set of world objects. In Fig. 1, for instance, board locations *a1* and *b2*, and tokens *113* and *1722* are objects within $\mathcal{D}$. A *predicate* consists of a predicate symbol p and an *arity* n. A *predicate variable* is an n-ary predicate parameterized with a tuple of n objects (e.g. $p(x_1, \ldots, x_n)$, with $\{x_1, \ldots, x_n\} \subseteq \mathcal{D}$), and evaluates to either *true* or *false*. An interpretation of an n-ary predicate is a set of n-tuples from $\mathcal{D}^n$ for which a predicate evaluates to *true*. In a slight abuse of notation, from now on we will refer to predicate variables simply as predicates. In FOL, we use *formulas* to express statements over the objects in $\mathcal{D}$. In this paper, we consider restricted formulas that consist of either single predicates or conjunctions ($\wedge$) of predicates[1]. Consider, for instance, the predicates $at(x, y)$ and $empty(z)$, whose semantics are, respectively, "token x is at cell y" and "cell z is empty". In Fig. 1, the following formulas evaluate to *true*: (1) $at(113, a3)$, and (2) $at(1722, e1) \wedge empty(e4)$. On the other hand, the following formulas evaluate to *false*: (3) $empty(a3)$, and (4) $at(113, a3) \wedge at(1722, a1)$.

We use the notation $set(X)$, where $X = x_1 \wedge x_2 \wedge \ldots \wedge x_n$ is a restricted FOL formula, to denotes the set $\{x_1, x_2, \ldots, x_n\}$.

A *state* s consists of a collection of assignments to a set of binary variables. Alternatively, s can be viewed as an interpretation over a set of predicates[2], each one representing a fact about the world. A state can be compactly represented as the set of *active predicates* (i.e. predicates that evaluate to *true*). The following is an excerpt of the state from Fig. 1:

$$s = \{at(113, a3), at(1722, e1), empty(a4), empty(b4), \ldots\}$$

In the rest of the paper, any manipulation of world states expressed with set operators is based on this compact form.

B. STRIPS Actions

STRIPS is a formalism to specify actions. A STRIPS action schema is a tuple $a = \langle head_a, pre_a, add_a, del_a \rangle$, where:

- $head_a$ is the *action signature*. It consists in a duad $\langle N_a, P_a \rangle$, where N_a is a human-readable name for the action, and P_a is a list of free variables which constitute the action parameters that take values over $\mathcal{D}$.
- pre_a is the *precondition*, a restricted FOL formula that must evaluate to true in the current state in order for the action to be applicable.

[1]The full specification of FOL includes negations, disjunctions and quantifiers, but these are not used for our method.

[2]Since predicates are statically evaluated, in dynamic settings the concept of *fluent* (variables that vary over time) is often used instead. Despite this technicality, for simplicity we will adhere to the use of the term predicate.

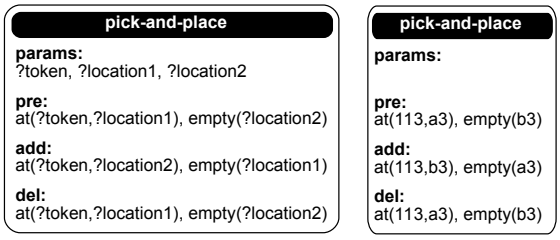

Fig. 2. Action schema with a possible grounding

- add_a is the *add list*, the list of predicates that will be set to *true* (or added to the compact form of the state) after the execution of the action.
- del_a is the *delete list*, the list of predicates that will be set to *false* (or deleted from the compact form of the state) after the execution of the action.

Predicates in pre_a, add_a, and del_a may be parameterized with objects and free variables from $D \cup P_a$. A schema with no parameters is said to be *grounded*. Any schema a can be grounded through a substitution $g : P_a \rightarrow \mathcal{D}$, resulting in action a_g. a_g is constructed by replacing each occurrence of the parameters P_a in pre_a, add_a, and del_a according to g. We always assume that $set(add_{a_g}) \cap set(del_{a_g}) = \emptyset$ (otherwise, the grounding is not possible). A grounded action a_g causes a state transition $s \rightarrow s'$ when $set(pre_{a_g}) \subseteq s$ and $s' = s \cup set(add_{a_g}) \setminus set(del_{a_g})$. Equivalently, we may say that a_g explains the transition $s \rightarrow s'$. Fig. 2 shows an example of a STRIPS action schema that allows the movement of any token to any empty position of the board, together with a possible grounding. As a convention in this paper, names starting with a question mark (*?*) are action parameters.

C. OARU

OARU is an algorithm for recognizing the STRIPS action schema and the grounding behind each transition in a stream of states. More importantly for our purposes, OARU learns a new STRIPS schema when the existing ones in its internal action library cannot explain a transition.

OARU maintains and edits an action library $\mathcal{A}$, initially empty. OARU receives as input a stream of states $S = s_1 \rightarrow s_2 \rightarrow \ldots s_{|S|}$. A pair $o_i = \langle s_i, s_{i+1} \rangle$ represents a state transition, or *observation*. For each observation, OARU may output: (1) a grounding of one of a schema $a \in \mathcal{A}$ that explains the observation, possibly editing a to introduce new parameters and relax the precondition; or (2) add an entirely new schema with no parameters with a stringent precondition (e.g. the entire state s_i as a precondition, or s_i filtered according to some criterion) that we call *Trivially Grounded Action* (TGA), with the intention of relaxing it when future similar transitions are observed.

OARU's procedure to introduce new parameters and relaxing preconditions is called *Action Unification* (AU), and works by merging two action schemata, producing a new one. To merge two actions, AU solves an optimization problem, encoded as a *Weighted Partial MaxSat* (WPMS) problem. To solve this problem, we resort to the Z3 SMT solving software. The merging procedure incurs in introducing new parameters and relaxing preconditions. Therefore, AU can be seen as a tool for generalizing actions. OARU uses AU as a subroutine to perform hierarchical clustering over the learnt actions. Please, refer to the original paper for a more in-depth explanation of AU and OARU [7].

We highlight that OARU was conceived to operate in open-world settings to allow partial observability (i.e. predicates with *unknown* evaluations are allowed). However, in this paper, we consider only fully observable settings.

IV. PROBLEM FORMULATION

We address the challenge of learning board exercises from human-given demonstrations. We assume that the teacher always provides correct demonstrations and that the learner has access to the entire state (full observability). We formalize this problem in terms of **input** and **expected output**.

A. Input

The demonstrations given by the teacher are structured into a series of independent executions of the exercise called *runs*: $R = \{S_1, S_2, \ldots, S_{|R|}\}$ ($|R| \geq 1$). A run $S_i = s_{i,1} \rightarrow s_{i,2} \rightarrow \ldots \rightarrow s_{i,l_i}$ is a stream of l_i states and consists of: (1) an initial state $s_{i,1}$, given by the teacher, that represents an arbitrary starting configuration of the board; (2) several intermediate states $\{s_{i,2}, \ldots, s_{i,l_i-1}\}$, each one being consequence of a move made by the teacher on the previous state; and (3) a final state $s_{i,l_i} = s_{i,l_i-1} \cup \{goal\text{-}achieved()\}$, where the special predicate *goal-achieved*() serves to assert that the goal of the exercise has been accomplished.

In addition, we allow to enrich the input with a set of *negative examples*: $N = \{n_1, \ldots, n_{|N|}\}$. Each n_i is an observation (i.e. a pair of states) that showcases an impossible transition, according to the rules of the exercise taught by the teacher.

We assume that there is a routine that analyzes each world state, extracts the relevant features for describing the board and the interaction among tokens, and expresses them into predicates. This is typical in robotic applications, in which there exists a scene understanding module that transforms sensor raw data into a workable format.

B. Expected Output

INPRO2 is expected to output a library of STRIPS schemata, $\mathcal{A}$ such that: (1) the entire set of observations among all the runs can be explained by the grounding of some action in $\mathcal{A}$; and (2) none of the negative examples in N are possible under any grounding of the actions in $\mathcal{A}$.

Later in Section V-B we will see how the definition of the problem's inputs and outputs, as defined in this section, naturally allows learning the goal of the exercises.

V. METHODOLOGY

INPRO2 uses OARU at its core to learn and maintain an action library $\mathcal{A}$ that encodes the rules and goal of the exercise, without requiring action signatures. In this section, we explain how we have extended OARU to **learn from negative examples** and **INPRO2's flow of execution**.

Algorithm 1: OARU's updates for negative examples

```
1  A := ∅                                  // Action library
2  N := ∅                  // Set of negative examples
3  Function Legal(a)
       Input: Action schema a
       Output: false if a allows any negative example,
               true otherwise
4      for n = ⟨s, s'⟩ ∈ N do
5          if ∃g s.t. a_g explains s → s' then
6              return false
7          end
8      end
9      return true
10 end
11 Function ProcessObservation(s, s')
       Input: Observation ⟨s, s'⟩
       Output: Grounded action a_g explaining s → s'
12     a_tga := BuildTGA(s, s')
13     a := ∅, a' := ∅, d_min := ∞
14     for β ∈ A do
15         ⟨a_u, d_{β,a_g}⟩ := MergeAU(β, a_tga)
16         if d_{β,a_tga} < d_min and Legal(a_u) then
17             a := a_u, a' := β, d_min := d_{β,a_tga}
18         end
19     end
20     if a' ≠ ∅ then
21         A := A ∪ {a} \ {a'}
22         a_g := grounding of a that explains s → s'
23     end
24     else
25         A := A ∪ {a_tga}  a_g := a_tga
26     end
27     return a_g
28 end
29 Function AddNegativeExample(s, s')
       Input: Observation ⟨s, s'⟩
       Output: None
30     N := N ∪ {⟨s, s'⟩}
31     while ∃a ∈ A s.t. not Legal(a) do
32         A := A∪{LeftParent(a), RightParent(a)}\{a}
33     end
34     while ∃a, a' ∈ A s.t. a ≠ a', ⟨a_u, d_{a,a'}⟩ =
        MergeAU(a,a') with d_{a,a'} < ∞, and
        Legal(a_u) do
35         A := A ∪ {a_u} \ {a, a'}
36     end
37 end
```

A. Learning from Negative Examples

One of the original limitations of OARU is that it always merges schemata with the same effect. This is undesirable when the only correct way of accurately reflecting the rules of an exercise using STRIPS is with two actions with identical effects but different preconditions. This situation arises in the exercise from Fig. 1: both odd and even numbers can be moved in a pick-and-place fashion, but odd numbers must move horizontally while even ones must move vertically. Without the modifications presented in this section, OARU would produce a single pick-and-place action for moving a token anywhere. We overcome this by enhancing OARU with the ability to undo previously merged actions via negative examples. The changes are shown in Algorithm 1. We note that the routine `MergeAU` (used in lines 15 and 34) merges two actions via Action Unification and returns the resulting action and the distance (real value) between the merged actions (∞ if the actions cannot be merged). Including negative examples forces two changes.

First, whenever a new observation is fed to OARU (i.e. a demonstration $s \rightarrow s'$ given by the teacher), the algorithm must guarantee that, if it edits any action in its internal action library, the resulting action is not able to produce any of the negative examples. Function `ProcessObservation`, defined in line 11, is almost identical to OARU's original algorithm [7], with the exception of the additional check in line 16 to verify that the newly constructed a_u is a legal action (using method `Legal` defined in line 3).

Second, when a negative example is added to OARU (via the `AddNegativeExample` method at line 29), every illegal action must be undone. Each illegal action a is decomposed back into its left parent and its right parent (the actions that were merged via `MergeAU` to form a). This refactoring process is depicted in lines 31-33. While this would be enough to guarantee that no illegal action is present in $\mathcal{A}$, it could happen that some of the ancestors of the actions that were undone can be (re)merged with different actions, producing legal actions. This remerging process is depicted in lines 34-36.

Fig. 3 depicts this mechanism with our running example of moving odd and even numbers. Since there are two kind of action with the same effect for odd and even numbers, OARU merges them together. However, after just one negative example, OARU is able to fix this mistake. This feature allows OARU to learn much more complex domain mechanics than before, because it allows to contextualize action based not just on their effect, but also on their precondition.

B. INPRO2

Our INPRO2 framework's top procedure coordinates demonstration gathering, negative example gathering, and calls to OARU during a teaching session. It follows the next flow of execution:

1) The teacher specifies which tokens are available for the exercise to establish the domain of discourse $\mathcal{D}$.
2) The initial position for a new run of the exercise is set.
3) INPRO2 gathers observations for the current run, registering the configuration of the board after each teacher's move. Each observation is fed to OARU.
4) After each observation, INPRO2 may ask for the legality of a move different from the teacher's (see Section V-C). If the move is illegal it is fed to OARU as a negative example.

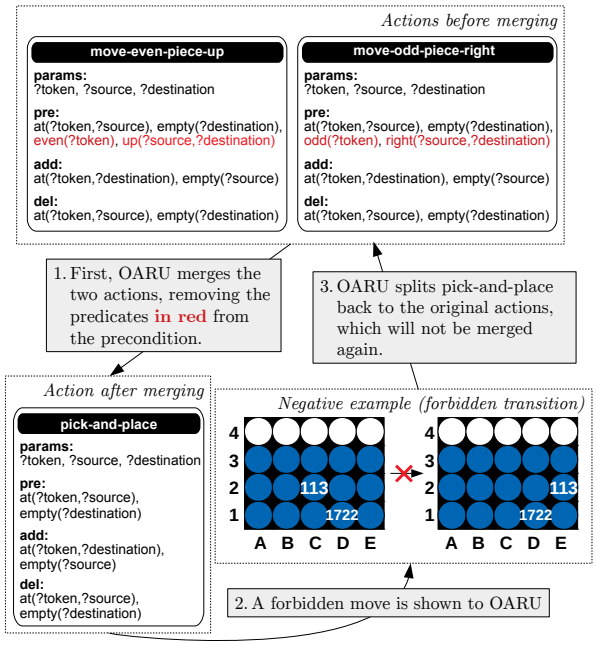

Fig. 3. Undoing an overly relaxed action thanks to a negative example. The human-readable names of actions and parameters have been chosen by us for the purpose of clarity in this paper, not by INPRO2.

5) Once the teacher has finished demonstrating one run of the exercise, they either conclude the teaching session or start a new run (go back to step 2). At this point, INPRO2 uses the last state of the run to learn or refine the goal of the exercise.

In addition to legal actions, INPRO2 is able to learn goals of moderate complexity thanks to our problem formulation: appending a new state $s_{i,l_i} = s_{i,l_i-1} \cup \{\textit{goal-achieved}()\}$ at the end of each $S_i \in R$ leads OARU to learn one or more special schemata whose sole purpose is to assert the achievement of the goal. Since the final observation at the end of each run always corresponds to adding the *goal-achieved*() predicate, there is a non-empty set $B = \{b | b_i \in \mathcal{A}, add_b = \{\textit{goal-achieved}()\}$ of schemata whose preconditions precisely reflect the learnt goal of the exercise.

C. Proactive Gathering of Negative Examples

Our rationale for actively gathering negative examples is twofold: (1) it sets a two-way interaction with the robot, providing the teacher feedback about the learning progression; and (2) while it is natural for a human teacher to provide examples on how a task is performed, it may be not so obvious to give examples of wrong ways to complete the task (especially if the teacher does not have technical knowledge about the learner). We consider three different strategies for collecting negative examples.

Random: The robot asks for the validity of random moves at random times during the teaching process. The random move can be sampled from one of the groundings of the available actions in $\mathcal{A}$ (if there is any). After each demonstration, the learner can decide with a certain fixed probability p whether to ask for the legality of a move. While simple, this strategy does not use any information about the given demonstration, nor its currently learnt schemata, to choose the movement to ask about, nor the proper moment at which it should do it.

Planning: A more sophisticated strategy to elicit and exploit negative examples that takes advantage of the learnt goals. We observe that actions that are learnt incorrectly usually have overly relaxed preconditions. Therefore, INPRO2 may find plans towards the goal significantly shorter than the length of the demonstrated runs. When the user demonstrates a move that does not agree with the optimal one suggested by INPRO2, this indicates that one of the learnt actions is too lax. Then, the robot asks about the legality of the supposedly optimal move to potentially gather negative examples. If the demonstrated movement is optimal, this strategy prompts the user when the planned action is different from the demonstrated one with a certain probability p. This ensures that negative examples can still be collected when the overly relaxed actions do not allow for shorter plans.

VI. Evaluation

We have implemented and evaluated our framework[3]. A TIAGo robot has been used as the physical embodiment of the learning agent. We use 5×4 electronic board equipped with RFID sensors and RFID tokens. This setup allows us to keep the robot informed at all times of the current configuration of the board.

We start by showcasing INPRO2 with an **example teaching session**. We also provide the results of several simulations in order to **quantitatively evaluate** the different proactive strategies for gathering negative examples.

A. Example of a Teaching Session

The system has been tested with human teachers and a variety of exercises. Fig. 4 shows three teaching sessions with human participants. Next, we will analyze in detail the session depicted in the middle image, which features the exercise from Fig. 1.

Fig. 5 shows the states observed by the robot during the teaching session. Each column is a different state stream (i.e. run), while each row is a state in the corresponding stream. The following is a step-by-step log of the events registered by INPRO2 with the *planning* strategy and $p = 0.15$ (i.e. the robot has a 15% probability of prompting the user when the demonstrated movement is optimal, but different from the planned one):

```
0 : new_demonstration. Added TGA action-1.
1 : new_demonstration. Action action-1
    upgraded to action-3.
2 : new_demonstration. Action action-3
    upgraded to action-5. (...)
3 : new_demonstration. No updates, action
    action-5 already explains the
    demonstration. (...)
4 : new_demonstration. Added goal transition
    action-7. (...)
5 : new_demonstration. No updates, action
    action-3 already explains the
```

[3]Source code: https://github.com/sprkrd/sat_strips_learn/

Fig. 4. Three teachers demonstrating different exercises to the robot: a spelling exercise (left); and numerical ones (middle and right).

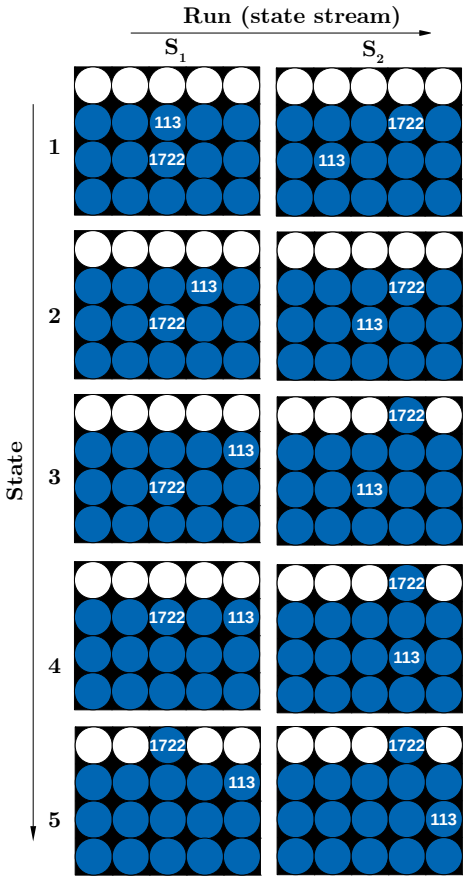

Fig. 5. Movements demonstrated during a teaching session.

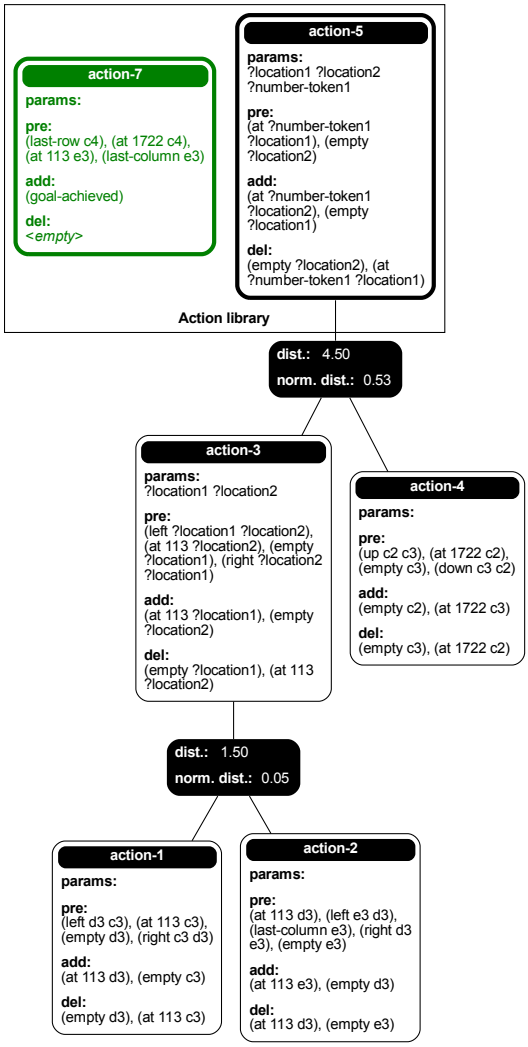

Fig. 6. Action library after S_1.

```
    demonstration). (...). Robot prompts the
    teacher: "In the previous position, could
    you have moved 113 from B2 to E3 and
    achieve the goal faster?".
    User answers: "No".
6 : new_negative_example. Added negative
    example (...). Added actions
    [action-8, action-3]. Removed actions:
    [action-5].
7 : new_demonstration. No updates (...)
8 : new_demonstration. No updates (...)
9 : new_demonstration. No updates (...)
10: new_demonstration. Action action-7 (goal)
    upgraded to action-14.
```

Events 0 to 4 correspond to states 1 to 5 from S_1 during the first run, S_1. Fig. 6 shows the status of the action library, and the history of merged actions, after the first run. Redundant merges (i.e. those that do not update the action library) have been omitted for the sake of brevity. The diagram shows which actions have been merged, and the distance between the merged actions according to the AU algorithm. The action shown in green is the last modification to the library (the goal transition, from event 4).

Event 5 is noteworthy: the user has demonstrated a movement of the token *113* from *b2* to *c2*. Using *action-5* from Fig. 6, however, INPRO2 can find a sequence of actions that leads to the learned goal much faster (directly moving *113* to

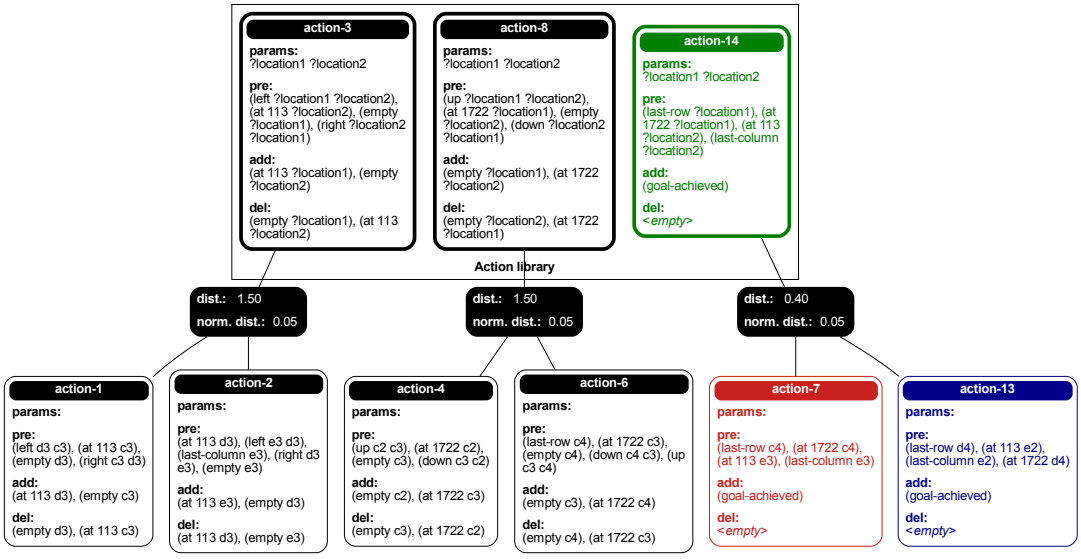

Fig. 7. Action library after S_2.

e3). The robot prompts the teacher about this move and finds out that it is not valid. This situation was already depicted in Fig. 3. The negative action helps INPRO2 notice that *action-5* is too lax and splits it back into *action-3* and *action-4*. Internally, OARU re-merges the transitions in a way that the negative example is not allowed by any action. The resulting action library after processing the remaining movements and the goal transition (events 6 to 10) is shown in Fig. 7. Notice that one action has been learned to move the odd number (*action-3*), another for the even number (*action-8*) and a last one for asserting the achievement of the goal (*action-14*). Thus, two runs and one negative example have been enough to learn this exercise.

B. Quantitative Evaluation

We simulate a human teacher showing demonstrations to the robot until the exercise is learnt. We simulate 1000 teaching sessions for the following exercises:

1) **Spelling**: 10 letters are initially arranged at the bottom two rows of the board. The goal is to spell either the word MARIE or the word CURIE at the top, from left to right moving one token at a time.
2) **Ordering**: 10 numbers are initially arranged at the bottom two rows of the board. The goal is to pick 5 of those numbers, one at a time, and place them in the top row ordered.
3) **Odd-Even**: the running example through this article.
4) **Pacman**: a token (player character) is moved around capturing other tokens (pellets).

TABLE I

AVERAGE **D** AND **P** (ROUNDED TO CLOSEST INTEGER) FOR DIFFERENT EXERCISES (**E**) AND STRATEGIES.

E	p = .00		p = .15		p = .50		p = .85		p = 1.0		Plan	
	D	P	D	P	D	P	D	P	D	P	D	P
1	-	-	15	2	13	5	10	7	10	8	15	2
2	-	-	15	2	14	6	10	6	10	8	15	2
3	-	-	12	2	9	4	7	5	7	5	7	1
4	18	0	18	3	18	8	18	12	18	16	18	3

For each teaching session, we measure: **D**, the number of demonstrations given by the teacher until the exercise is learnt; and **P**, the number of times the robot has prompted the teacher for the legality of a move. We evaluate the mean of **D** and **P** across the 1000 teaching sessions for the **Random** and **Planning** strategies (Tab. I). The **Random** strategy is evaluated for $p \in \{0, 0.15, 0.5, 0.85, 1\}$. The **Planning** strategy is evaluated for $p = 0.15$ and $n = 3$. We note that when $p = 0$ there are no prompts, and with $p = 1$ the robot tries to prompt the teacher after each demonstration.

Our first highlight is that some of the exercises (**Spelling**, **ordering** and **Odd-Even**) cannot be learnt without negative examples ($p = 0$). **Spelling** and **Ordering** need negative examples because the tokens cannot be put in the top row in any order (there is a *left-to-right* constraint). For **Odd-Even**, the reason has already been exposed in Fig. 3.

Secondly, we observe that for the random strategies, $P \neq pD$ necessarily (let us remember that p is the probability of prompting the teacher for the legality of a random move after a demonstration). In particular, one might expect that, with $p = 1$, $P = D$ (one prompt after each teacher's move). This is not the case because when the robot has just started to learn, it is very unlikely that there is a grounding of an action in $\mathcal{A}$ that can be applied in the current state.

Notice that, for **Spelling** and **Ordering**, there is not much difference between **Random** with $p = 0.15$ and **Planning**. The reason for this is that these two exercises require negative examples to undo an overly permissive action that allows placing the tokens in any non *left-to-right* sequence. However, this action does not allow to complete the exercise faster. Thus, the **Planning** strategy falls back to **Random** with $p = 0.15$. On the other hand, if we observe **Random** with larger values of p, we can see that the number

of needed demonstrations decreases (negative examples are found sooner), and the number of prompts increases.

In **Odd-Even** the situation is different. This exercise leads OARU to learn an overly permissive action that allows it to move a token to its final destination, without performing the intermediate steps. This is quickly caught by the **Planning** strategy, which is able to reduce both the number of needed demonstrations and the number of prompts.

In **Pacman** we can see that, since negative examples do not help, the number of needed demonstrations is about the same for every strategy. This exercise requires a large number of demonstrations because it showcases two kinds of moves: displacements to adjacent cells, and captures. This means that the **Random** strategies with low values of p, and the **Planning** strategies, offer less overhead in terms of unnecessary prompts to the user. Still, it could be argued that some degree of feedback is desirable to offer some feeling of progression to the teacher.

These results seem to suggest that the **Planning** strategy can be very helpful in certain exercises, while not incurring excessive overhead in terms of the number of prompts, performing similarly to **Random** with small values of p.

VII. Conclusions and Future Work

In this paper we have presented INPRO2, a framework based on OARU's algorithm to enable a robotic agent to effectively learn cognitive exercises in terms of STRIPS action schemata. We have extended OARU with the ability to process negative examples, which is valuable to learn certain exercises. These negative examples are gathered proactively by the robot. We have proposed a simple random strategy and a planning-based one to actively engage the user and gather negative examples. In addition, our problem formulation allows us to learn exercise goals.

In this work, we have not focused on effective ways of explaining to the teacher the learnt exercises. This is indeed an interesting research direction since it can contribute to the explainability of the method. Future work also includes analyzing the impact of proactive strategies in humans in terms of cognitive load and subjective experience. This study is currently being conducted, and seeks to shed light on the compromise between the number of demonstrations and prompts preferred by users.

References

[1] A. Andriella, C. Torras, C. Abdelnour, and G. Alenyà, "Introducing CARESSER: A framework for in situ learning robot social assistance from expert knowledge and demonstrations," *User Modeling and User-Adapted Interaction*, vol. 33, no. 2, pp. 441–496, 3 2023.

[2] A. Andriella, C. Torras, and G. Alenyà, "Cognitive System Framework for Brain-Training Exercise Based on Human-Robot Interaction," *Cognitive Computation*, vol. 12, no. 4, pp. 793–810, 2020.

[3] A. Kubota, E. I. C. Peterson, V. Rajendren, H. Kress-Gazit, and L. D. Riek, "Jessie: Synthesizing social robot behaviors for personalized neurorehabilitation and beyond," in *Proceedings of the 2020 ACM/IEEE International Conference on Human-Robot Interaction*, ser. HRI '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 121–130.

[4] A. Andriella, A. Suárez-Hernández, J. Segovia-Aguas, C. Torras, and G. Alenyà, "Natural teaching of robot-assisted rearranging exercises for cognitive training,," in *International Conference on Social Robotics*. Springer International Publishing, 2019, pp. 611–621.

[5] A. Suárez-Hernández, A. Andriella, A. Taranović, J. Segovia-Aguas, C. Torras, and G. Alenyà, "Automatic learning of cognitive exercises for socially assistive robotics," in *2021 30th IEEE International Conference on Robot Human Interactive Communication (RO-MAN)*, no. 1, 2021, pp. 139–146.

[6] R. E. Fikes and N. J. Nilsson, "STRIPS: A new approach to the application of theorem proving to problem solving," *Artificial Intelligence*, vol. 2, no. 3-4, pp. 189–208, 1971.

[7] A. Suárez-Hernández, J. Segovia-Aguas, C. Torras, and G. Alenyà, "Online Action Recognition," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 13, pp. 11 981–11 989, 2021.

[8] S. Thellman, A. Silvervarg, A. Gulz, and T. Ziemke, "Physical vs. virtual agent embodiment and effects on social interaction," in *Intelligent Virtual Agents: 16th International Conference, IVA 2016, Los Angeles, CA, USA, September 20–23, 2016, Proceedings 16*. Springer, 2016, pp. 412–415.

[9] S. Schneider and F. Kummert, "Comparing the effects of social robots and virtual agents on exercising motivation," in *Social Robotics: 10th International Conference, ICSR 2018, Qingdao, China, November 28-30, 2018, Proceedings 10*. Springer, 2018, pp. 451–461.

[10] S. Rossi, M. Staffa, and A. Tamburro, "Socially assistive robot for providing recommendations: Comparing a humanoid robot with a mobile application," *International Journal of Social Robotics*, vol. 10, pp. 265–278, 2018.

[11] D. Martínez, G. Alenyà, and C. Torras, "Relational reinforcement learning with guided demonstrations," *Artificial Intelligence*, vol. 247, pp. 295–312, 6 2017.

[12] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel, T. Lillicrap, and D. Silver, "Mastering Atari, Go, chess and shogi by planning with a learned model," *Nature*, vol. 588, no. 7839, pp. 604–609, 12 2020.

[13] S. Arora and P. Doshi, "A survey of inverse reinforcement learning: Challenges, methods and progress," *Artificial Intelligence*, vol. 297, p. 103500, 8 2021.

[14] B. Okal and K. O. Arras, "Learning socially normative robot navigation behaviors with Bayesian inverse reinforcement learning," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, 2016, pp. 2889–2895.

[15] Y. Gil, "Learning by Experimentation: Incremental Refinement of Incomplete Planning Domains," in *Machine Learning Proceedings 1994*. Elsevier, 1994, pp. 87–95.

[16] S. Benson, "Inductive Learning of Reactive Action Models," in *Machine Learning Proceedings 1995*. Elsevier, 1995, pp. 47–54.

[17] X. Wang, "Learning by Observation and Practice: An Incremental Approach for Planning Operator Acquisition," in *Machine Learning Proceedings 1995*. Elsevier, 1995, pp. 549–557.

[18] Q. Yang, K. Wu, and Y. Jiang, "Learning action models from plan examples using weighted MAX-SAT," *Artificial Intelligence*, vol. 171, no. 2-3, pp. 107–143, 2007.

[19] E. Amir and A. Chang, "Learning Partially Observable Deterministic Action Models," *Journal of Artificial Intelligence Research*, vol. 33, pp. 349–402, 11 2008.

[20] D. Aineto, S. Jiménez, and E. Onaindia, "Learning STRIPS Action Models with Classical Planning," *International Conference on Automated Planning and Scheduling*, vol. 28, no. 1, pp. 399–407, 2018.

[21] A. Suárez-Hernández, J. Segovia-Aguas, C. Torras, and G. Alenyà, "STRIPS Action Discovery," in *AAAI 2020 workshop on Generalization in Planning*. arXiv:2001.11457, 2020.

[22] E. Senft, S. Lemaignan, M. Bartlett, P. Baxter, and T. Belpaeme, "Robots in the classroom: Learning to be a Good Tutor," in *Robots for Learning Workshop at HRI*, 2018.

[23] N. Efthymiou, P. Filntisis, G. Potamianos, and P. Maragos, "A robotic edutainment framework for designing child-robot interaction scenarios," in *The 14th PErvasive Technologies Related to Assistive Environments Conference*. New York, NY, USA: ACM, 6 2021, pp. 160–166.

[24] K. Winkle, S. Lemaignan, P. Caleb-Solly, P. Bremner, A. Turton, and U. Leonards, "In-Situ Learning from a Domain Expert for Real World Socially Assistive Robot Deployment," in *Robotics: Science and Systems XVI*, vol. 10. Robotics: Science and Systems Foundation, 7 2020.

[25] A. Causo, P. Z. Win, P. S. Guo, and I.-M. Chen, "Deploying social robots as teaching aid in pre-school K2 classes: A proof-of-concept study," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 4264–4269.