There and Back Again: Combining Non-monotonic Logical Reasoning and Deep Learning on an Assistive Robot*

Mohan Sridharan[1,*], Chloé Benz[2], Arthur Findelair[3] and Kévin Gloaguen[4]

[1]Intelligent Robotics Lab, School of Computer Science, University of Birmingham, UK
[2]Illinois Institute of Technology, USA
[3]Illinois Institute of Technology, USA
[4]École Nationale Supérieure de Mécanique et d'Aérotechnique, France

Abstract

This paper describes the development of an architecture that combines non-monotonic logical reasoning and deep learning in virtual (simulated) and real (physical) environments for an assistive robot. As an illustrative example, we consider a robot assisting in a simulated restaurant environment. For any given goal, the architecture uses Answer Set Prolog to represent and reason with incomplete commonsense domain knowledge, providing a sequence of actions for the robot to execute. At the same time, reasoning directs the robot's learning of deep neural network models for human face and hand gestures made in the real world. These learned models are used to recognize and translate human gestures to scenarios that mimic real-world situations in the simulated environment, and to goals that need to be achieved by the robot in the simulated environment. We report the challenges faced in the development of such an integrated architecture, as well as the insights learned from the design, implementation, and evaluation of this architecture by a distributed team of researchers during the ongoing pandemic.

Keywords

Non-monotonic logical reasoning, Probabilistic reasoning, Interactive learning, Robotics

1. Motivation

Consider the motivating example of a mobile robot (*Pepper*) waiter in a simulated restaurant, as shown in Figure 1. The robot has to perform tasks such as seating customers at suitable tables, taking and delivering food orders, and collecting payment. To perform these tasks, the robot extracts and reasons with the information from different sensors (e.g., camera, range finder) and incomplete commonsense domain knowledge. This knowledge includes relational descriptions of the domain objects and their attributes (e.g., size, number, and relative positions of tables, chairs, and people). It also includes axioms governing actions and change in the domain (e.g., the preconditions and effects of seating a group of people at a particular table), including default statements that hold in all but a few exceptional circumstances (e.g., "customers typically need some time to look at the menu before they place an order"). Since the domain description is incomplete and can change over time, the robot also reasons with its knowledge and sensor observations to revise its

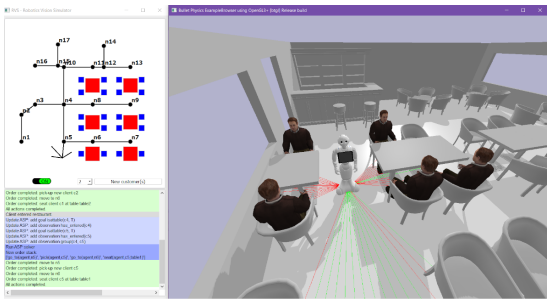

Figure 1: Illustrative snapshot of an assistive robot operating as a waiter in a simulated restaurant scenario.

knowledge (e.g., revise the number of people seated at different tables, learn the effects of different gestures). Furthermore, to promote better interaction with humans in the restaurant, the robot provides on-demand relational descriptions of its decisions and the evolution of beliefs.

Realizing the motivating scenario described above poses fundamental challenges in knowledge representation, reasoning, and learning. State of the art robot architectures often seek to address these challenges by using logics and probabilistic methods to represent and reason with domain knowledge and observations, and by using data-driven (deep) learning methods to extract knowledge from large, labeled datasets (e.g., of noisy sensor observations). However, practical domains make it

20th International Workshop on Non-Monotonic Reasoning
**Corresponding author.
m.sridharan@bham.ac.uk (M. Sridharan); chloe.c.benz@gmail.com (C. Benz); arthfind@gmail.com (A. Findelair); k.gloaguen1303@gmail.com (K. Gloaguen)
https://www.cs.bham.ac.uk/~sridharm/ (M. Sridharan)
0000-0001-9922-8969 (M. Sridharan)

difficult to provide a comprehensive encoding of domain knowledge, or the computational resources and examples needed to augment or revise the robot's knowledge. Furthermore, circumstances such as the ongoing pandemic make it rather challenging for a distributed team of researchers to design and evaluate such architectures for integrated robot systems.

This paper makes a two-fold contribution towards addressing the above-mentioned challenges. First, it uses the motivating example to describe the development of an architecture that adapts knowledge representation (KR) tools to achieve transparent, reliable, and efficient knowledge-based reasoning and data-driven learning on an assistive robot. Second, it highlights the advantages of using KR tools, and of formally coupling representation, reasoning and learning, to design such an architecture. More specifically, our architecture:

- Represents and performs non-monotonic logical reasoning with incomplete commonsense domain knowledge using Answer Set Prolog (ASP) to obtain a plan of abstract actions for any given goal;
- Executes each abstract action as a sequence of concrete actions by automatically identifying and reasoning probabilistically about the relevant domain knowledge at a finer granularity;
- Reasons with domain knowledge to allow humans making hand gestures in the physical world to interact with the simulated robot in a manner that mimics interaction in the physical world; and
- Reasons with domain knowledge to guide the learning of models for new hand gestures and the corresponding axioms, and for providing on-demand relational descriptions as *explanations* of the robot's decisions and beliefs.

The interactive interface between the virtual and physical world helped the three undergraduate student authors design, implement, and evaluate the architecture remotely over different time intervals during the pandemic. It also helped us explore the interplay between reasoning and learning. The "there and back again" in the title thus refers to the architecture's on-demand ability to traverse different points in space and time, and to transition between the physical and virtual world for human-robot collaboration. We demonstrate the capabilities of our architecture through experimental results and execution traces of use cases in our motivating restaurant domain.

The remainder of this paper is organized as follows. We begin by discussing related work in Section 2. Next, we describe our architecture and its components in Section 3. The execution traces and results of evaluating our architecture's components are described in Section 4, and the conclusions are described in Section 5.

2. Related Work

There is a well-established history of the use of logics in different AI and robotics applications. The non-monotonic logical reasoning paradigm used in this paper, ASP, has been used by an international community of researchers for many applications in robotics [1] and other fields [2]. There has also been a lot of work over multiple decades on integrating logical and probabilistic reasoning [3, 4, 5], and on using different logics for guiding probabilistic sequential decision making [6]. Our focus here is on building on this work to support transparent knowledge-based reasoning and data-driven learning in integrated robot systems.

There are many methods for learning logic-based representations of domain knowledge. This includes the incremental revision of action operators in first-order logic [7], the inductive learning of domain knowledge encoded as an Answer Set Prolog program [8], and the work on coupling non-monotonic logical reasoning with inductive learning or relational reinforcement learning to learn axioms [9, 10]. Our approach in this architecture is inspired by work in interactive task learning [11]; unlike methods that learn from many training examples, our approach seeks to identify and learn from a limited number of relevant training examples.

Given the use of deep networks in different applications, there is much interest in understanding their operation in terms of the features influencing network outputs [12, 13]. There is also work on neuro-symbolic systems that reason with learned symbolic structure or a scene graph in conjunction with deep networks to answer questions about images [14, 15]. Work in the broader areas of *explainable AI* and *explainable planning* can be categorized into two groups. Methods in one group modify or map learned models or reasoning systems to make their decisions more interpretable [16] or easier for humans to understand [17]. Methods in the other group provide descriptions that make a reasoning system's decisions more transparent [18], help humans understand plans [19], and help justify solutions obtained by non-monotonic logical reasoning [20]. Recent survey papers indicate that existing methods: (i) do not fully integrate reasoning and learning to inform and guide each other; (ii) do not fully exploit the available commonsense domain knowledge for reliable, efficient, and transparent reasoning and learning; and (iii) are often agnostic to how an explanation is structured or assumes comprehensive domain knowledge [21, 22]

Our work focuses on transparent, reliable, and efficient reasoning and learning in integrated robot systems that combine reasoning with incomplete commonsense domain knowledge and data-driven learning from limited examples. We seek to demonstrate that this objective can be achieved by building on KR tools. To do so, we build on some of the prior work of the lead author with others.

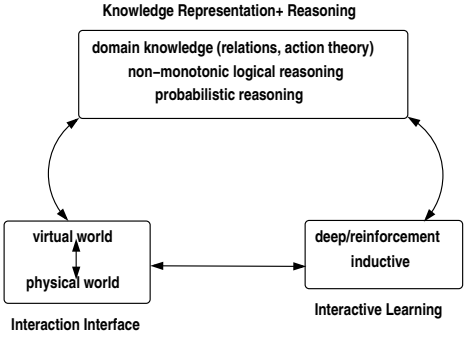

Figure 2: Overview of our architecture combining non-monotonic logical reasoning, probabilistic reasoning, and deep learning for reliable, efficient, and transparent reasoning and learning.

In particular, we build on work on: (i) a refinement-based architecture for representation and reasoning [23]; (ii) explainable agency and theory of explanations [24, 25]; and (iii) combining non-monotonic logical reasoning and deep learning for axiom learning and scene understanding [9, 26]. The novelty is in bringing these different strands together in an architecture, and in facilitating the interactive interface between the virtual and physical worlds for design and evaluation.

3. Architecture Description

Figure 2 presents an overview of the main components of our architecture. As stated earlier, the architecture uses ASP to represent and reason with commonsense domain knowledge, e.g., to reason about object and robot attributes to compute a plan to achieve a given goal. For more complex domains, this reasoning can take place using transition diagrams at two different resolutions, with the fine-resolution diagram defined as a refinement of the coarse-resolution diagram. Execution of the actions by a robot can then involve probabilistic reasoning with a relevant part of the fine-resolution transition diagram. Reasoning informs and guides both the interactive learning of previously unknown domain knowledge (which is used for subsequent reasoning), and the interface for interaction between a human in the physical world and the robot in the virtual world. Reasoning is also used to identify relevant literals and axioms to provide an on-demand description of the robot's decisions and beliefs. The individual components are described below using the following example domain.

Example Domain 1. *[Robot Waiter (RW) Domain]*
A Pepper robot operates as a waiter in a restaurant. Its tasks include: (i) greeting and seating customers; (ii) taking food orders and delivering food to specific tables; (iii) providing a bill and collecting payment; and (iv) responding to requests from the customer(s) and the designer. The robot uses probabilistic algorithms to model and account for the uncertainty experienced during perception and actuation. Interactions of the robot with a human supervisor are handled through the interface that interprets hand gestures made by a human in the physical world. The robot has incomplete (and potentially imprecise) domain knowledge, which includes number, size, and location of tables and chairs; spatial relations between objects; and some axioms governing domain dynamics such as:

- If the robot allocates a group of customers to a table, all members of the group are considered to be seated at that table.
- The robot cannot seat customers at a table that is not empty, i.e., is occupied.
- Any customer cannot be allocated to more than one table at a time.

This knowledge, e.g., the axioms describing dynamic changes and the values of some attributes of the domain or robot, may need to be revised over time.

3.1. Representation and Reasoning

To represent and reason with domain knowledge, we use CR-Prolog, an extension of Answer Set Prolog (ASP) that introduces *consistency restoring* (CR) rules [27]. ASP is based on stable model semantics, and supports *default negation* and *epistemic disjunction*, e.g., unlike "$\neg a$" that implies *a is believed to be false*, "$not\ a$" only implies *a is not believed to be true*, and unlike "$p \vee \neg p$" in propositional logic, "p *or* $\neg p$" is not tautologous. ASP can represent recursive definitions and constructs that are difficult to express in classical logic formalisms, and it supports non-monotonic logical reasoning, i.e., the ability to revise previously held conclusions based on new evidence. We use the terms "CR-Prolog" and "ASP" interchangeably in this paper.

Knowledge representation. A domain's description in ASP comprises a *system description* $\mathcal{D}$ and a *history* $\mathcal{H}$. $\mathcal{D}$ comprises a *sorted signature* Σ and axioms encoding the domain's dynamics. Σ comprises *basic sorts*, *statics*, i.e., domain attributes that do not change over time, *fluents*, i.e., domain attributes whose values can be changed, and *actions*; note that statics, fluents, and actions are described in terms of the sorts of their arguments. In the RW domain, the robot needs to reason about spatial relations between objects, and to plan and execute actions that change the domain. Such a dynamic domain is modeled in our architecture by first describing Σ and the domain's transition diagram in action language $\mathcal{AL}_d$ [28]; this description is then

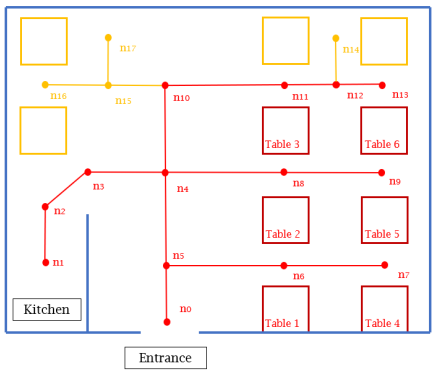

Figure 3: Example layout of the RW domain, which organizes the available space into nodes representing regions with specific tables.

translated to ASP statements. The basic sorts of the RW domain include $table$, $robot$, $customer$, $employee$, $waiter$, $furniture$, $gesture$, $gesture\_category$, and $step$ for temporal reasoning. The sorts may be organized hierarchically, e.g., *chair* and *table* are subsorts of the sort *furniture*, and the sort *employee* includes *robot* and *supervisor* as subsorts.

Statics of the RW domain include relations *edge(node, node)* and *linked(node, furniture)*; the former is a graph-based encoding of regions, e.g., see Figure 3, and the latter associates particular tables to particular nodes. Fluents include relations such as *location(robot, node)*, *iswaiting(customer)*, *attable(customer, table)*, *occupancy(table, num)*, and *haspaid(customer)*. Actions of the RW domain include *move(robot, node)*, which causes the robot to move to a particular node; *seat(robot, customer, table)*, which causes the robot to seat particular customer(s) at a particular table; and *givebill(robot, table)*, which causes the robot to give the bill to a customer at a particular table. In addition, relation *holds(fluent, step)* implies that a particular fluent holds true at a particular timestep, and *occurs(action, step)* implies the occurrence of a particular action at a particular timestep of the plan.

Given the signature Σ, axioms describing a domain's dynamics consist of causal laws, state constraints, and executability conditions. For the RA domain, these are translated to statements in ASP such as:

$$holds(loc(R,N),I+1) \leftarrow \quad (1a)$$
$$occurs(move(R,N),I)$$
$$holds(attable(C,T),I+1) \leftarrow \quad (1b)$$
$$occurs(seat(R,C,T),I)$$
$$\neg holds(attable(C,T2),I) \leftarrow \quad (1c)$$
$$holds(attable(C,T1),I),\ \ T1 \neq T2$$
$$\neg holds(occupancy(T,X2),I) \leftarrow \quad (1d)$$
$$holds(occupancy(T,X1),I),\ \ X1 \neq X2$$
$$\neg occurs(move(R,N),I) \leftarrow \quad (1e)$$
$$holds(loc(R,M),I),\ \neg edge(M,N)$$
$$\neg occurs(givebill(R,T),I) \leftarrow \quad (1f)$$
$$\neg holds(wantsbill(T),I)$$

which encode two causal laws, two state constraints, and two executability conditions respectively. For example, Statement 1(a) is a causal law that implies that executing the *move* action causes the robot's location to be the desired node in the next time step, Statement 1(c) is a constraint stating that a customer can only be at one table at a time, and Statement 1(e) is an executability condition that implies that a move to a target location is not possible if it is not connected to the robot's current location. The axioms also encode some *default statements* that hold in all but a few exceptional situations. For example, in the RW domain, we may want to encode that "clean plates are usually in the kitchen" unless stated otherwise:

$$holds(loc(P,kitchen),I) \leftarrow\ holds(clean(P),I),$$
$$plate(P),\ not\ \neg holds(loc(P,kitchen),I) \quad (2)$$

where "not" denotes default negation. One potential exception to this axiom is that some clean plates may also be placed near the buffet table; these exceptions can also be encoded. In addition to axioms, information extracted from the sensor inputs (e.g., different hand gestures) are also converted to ASP statements at that time step. Each gesture is also associated with the corresponding axioms; more specific details are provided in Section 3.3.

A dynamic domain's history $\mathcal{H}$ typically comprises records of: (a) fluents observed to be true or false at a particular time step; and (b) the actual execution of particular actions at particular time steps:

$$obs(fluent, boolean, step)$$
$$hpd(action, step)$$

Prior work demonstrated that this notion of history can be expanded to include defaults describing the values of fluents in the initial state, along with exceptions [23].

Reasoning. Given the representation of domain knowledge described above, the robot still needs to reason with this knowledge and observations perform tasks such as inference, planning, and diagnostics. In our architecture, we automatically construct the CR-Prolog program $\Pi(\mathcal{D},\mathcal{H})$, which includes Σ and axioms of $\mathcal{D}$, inertia axioms, reality check axioms, closed world assumptions for actions, and observations, actions, and defaults from $\mathcal{H}$; a basic version of this program can be viewed online [29]. For planning and diagnostics, this program also includes helper axioms that define a goal, and require the robot to search until a consistent model of the world is constructed and a plan is computed to achieve the goal. Planning,

diagnostics, and inference are then reduced to computing *answer sets* of Π; we use the SPARC system [30] to compute answer set(s). Each answer set represents the robot's beliefs in a possible world; the literals of fluents and statics at a time step represent the domain's *state* at that time step. As stated earlier, our architecture's non-monotonic reasoning ability supports recovery from incorrect inferences due to incomplete knowledge or noisy sensor inputs.

Prior work by the lead author and others resulted in an architecture for reasoning with transition diagrams at two resolutions, with the fine-resolution diagram formally defined as a *refinement* of the coarse-resolution diagram [23]. This definition differs from recent work on refinement and abstraction of ASP programs and other logics [31, 32] in how the transition diagrams are coupled formally to satisfy the requirements in the challenging context of integrated robot systems. This relation guarantees the existence of a path in the fine-resolution transition diagram implementing each coarse-resolution transition. The robot can then use non-monotonic logical reasoning to compute a sequence of abstract actions for any given goal, implementing each abstract action as a sequence of fine-resolution actions by automatically zooming to and reasoning probabilistically with the part of the fine-resolution diagram relevant to the coarse-resolution transition. We build on that notion of *relevance* to automatically: (a) constrain the robot's attention to the nodes and regions relevant to any given transition or plan that the robot has to execute—this supports selective grounding; (b) limit recognition of hand gestures to the subset relevant to the task at hand, e.g., gestures for placing an order once customers are seated, and limit learning to previously unknown hand gestures and related axioms—see Section 3.3; and (c) provide relational descriptions of decisions by tracing the evolution of relevant beliefs and application of relevant axioms—see Section 3.3. For ease of understanding, we define the notion of relevance for a given transition; similar definitions can be provided for a given goal or literal.

Definition 1. *[Relevant object constants]*
Let $T = \langle \sigma_1, a_{tg}, \sigma_2 \rangle$ be the transition of interest. Let $relCon(T)$ be the set of object constants of signature Σ of $\mathcal{D}$ identified using the following rules:

- Object constants from a_{tg} are in $relCon(T)$;
- If $f(x_1, \ldots, x_n, y)$ is a literal formed of a domain attribute, and the literal belongs to σ_1 or σ_2, but not both, then $x_1, \ldots, x_n, y$ are in $relCon(T)$;
- If body B of an axiom of a_{tg} contains $f(x_1, \ldots, x_n, Y)$, a term whose domain is ground, and $f(x_1, \ldots, x_n, y) \in \sigma_1$, then $x_1, \ldots, x_n, y$ are in $relCon(T)$.

Object constants from $relCon(T)$ are said to be *relevant* to T. For example, consider an initial state σ_1 with $loc(rob_1, n_1)$ and $loc(waiter, kitchen)$, and action $a_{tg} = move(rob_1, n_2)$. The object constants relevant to this transition then include rob_1, n_1, n_2, and $kitchen$.

Definition 2. *[Relevant system description]*
The system description relevant to a transition $T = \langle \sigma_1, a_{tg}, \sigma_2 \rangle$, i.e., $\mathcal{D}(T)$, is defined by signature $\Sigma(T)$ and axioms. $\Sigma(T)$ is constructed to comprise:

- Basic sorts of Σ that produce a non-empty intersection with $relCon(T)$.
- All object constants of basic sorts of $\Sigma(T)$ that form the range of a static attribute.
- The object constants of basic sorts of $\Sigma(T)$ that form the range of a fluent, or the domain of a fluent or a static, and are in $relCon(T)$.
- Domain attributes restricted to $\Sigma(T)$'s basic sorts.

Axioms of $\mathcal{D}(T)$ are those of $\mathcal{D}$ restricted to $\Sigma(T)$. It can be shown that for each transition in the transition diagram of $\mathcal{D}$, there is a transition in the transition diagram of $\mathcal{D}(T)$. States of $\mathcal{D}(T)$, i.e., literals comprising fluents and statics in the answer set of the ASP program, and ground actions of $\mathcal{D}(T)$, are candidates for further exploration. Continuing with the example in Definition 1, for $a_{tg} = move(rob_1, n_2)$, $\mathcal{D}(T)$ will not include axioms corresponding to other actions, e.g., for seating customers at a table or giving the bill to a customer. If the robot has to perform fine-resolution probabilistic reasoning for action execution, only the refinement of the relevant system description will be considered.

A robot waiter equipped with the representation and reasoning module described above, still needs to interact with humans. To support design and evaluation when in-person interaction with the robot is not possible, we incorporated the interactive simulation module, as described below.

3.2. Interactive Simulation and Hand Gestures

We developed a simulation environment and interface for the design and evaluation of our architecture. We used PyBullet [33], a Python-based module for simulating games and domains for machine learning and robotics. It enables us to quickly load different articulated bodies and provides built-in support for forward and inverse kinematics, collision detection, and simulation of domain dynamics.

In our architecture, PyBullet is used to automatically generate a restaurant layout, e.g., see Figure 4, based on the domain information encoded in the ASP program, e.g., Figure 3. Using the built-in blender of PyBullet, we are able to populate the simulated restaurant with a Pepper robot, tables, chairs, and the desired number of customers. We are also able to make on-demand revisions to the domain, e.g., to match changes in the domain knowledge. In addition, our simulator supports the movement of the

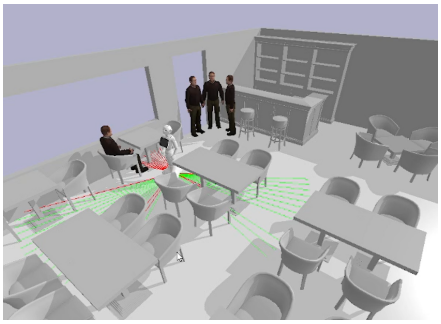

Figure 4: Simulated restaurant layout in PyBullet with robot waiter and customers.

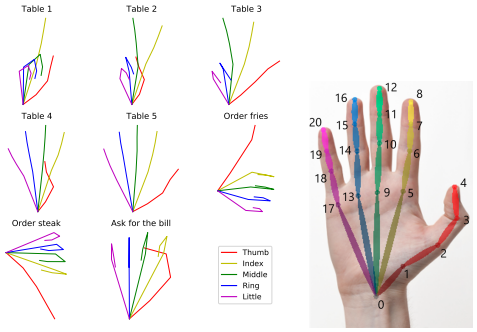

Figure 5: (Left) Subset of hand gestures providing directions to robot; (Right) The 21 keypoints used to model each hand gesture.

robot in the restaurant based on the axioms encoded in the ASP program. Furthermore, it is also possible to introduce new objects in the simulator (e.g., using hand gestures, see below) and automatically add this information to the ASP program for further reasoning

Recall that communication of human instructions to the robot waiter is based on hand gestures made in the physical world. To support such interaction, we first enabled our architecture to recognize a base set of hand gestures; a subset of these gestures are shown in Figure 5(left). To model and recognize hand gestures, we integrate the OpenPose system [34] that characterizes gestures using 21 keypoints, as shown in Figure 5(right). After the integration, the simulator allows us to capture images of the hand gestures made in the physical world to quickly train deep network models that can accurately recognize these gestures in new videos (i.e., image sequences). We used an existing Python library for training these deep network models with experimentally determined loss functions—Figure 6. Note that the modularity of the architecture makes it easy to quickly explore the different deep network models without changing other parts of the architecture. The known hand gestures with trained models are then grouped in different categories based on whether they are related to seating customers, handling food orders, or executing terminal transactions (e.g., provide bill).

3.3. Interactive Learning and Transparency

The architecture described so far reasons with incomplete domain knowledge, which may lead the robot to make incorrect decisions or cause the robot's performance to suffer, e.g., the robot may compute incorrect or unnecessarily long plans for any given goal. Also, the encoded knowledge and models may need to change over time. We address this requirement by introducing a module for interactive learning and generation of relational descriptions as "explanations" of the robot's decisions and beliefs.

Interactive learning. The interactive learning component of our architecture has two parts. Given the use of hand gestures for human-robot interaction, the first part seeks to detect new gestures and learn models for these gestures. A new hand gesture is detected when the observed gesture differs significantly from any of the known gestures. A significant difference is experimentally determined as a difference in 15% of the keypoints in a sequence of images. When a new gesture is recognized, the robot automatically gathers a sequence of image frames, extracts features from these images, stores them in a separate file and quickly updates the hand gesture recognition models to include this new gesture. A key feature of our architecture is that reasoning and learning inform and guide each other. For example, when the robot has to recognize and respond to gestures, it automatically limits itself to gestures relevant to its current category of tasks, e.g., a robot delivering food cannot respond to direction from a supervisor to seat new customers[1]. Also, any newly learned gesture is placed in the appropriate category of gestures (determined based on purpose of gesture) for subsequent reasoning. This use of reasoning to direct learning speeds up recognition and learning.

The second part of the learning component focuses on acquiring axioms corresponding to any new gesture, and merging the axioms with the existing ones. This is achieved by taking the label provided by human for the new gesture and checking if the corresponding instruction (e.g., seat two people) can be executed with the existing knowledge. If that is possible, no further learning is performed. If existing knowledge is insufficient to execute the new instruction, or if the human provides feedback, e.g., a textual or verbal description that is processed using existing tools, which includes an action, literals extracted from the feedback are used to construct an axiom that is merged with existing ones. Once again, reasoning helps

[1] Associating priority levels with tasks will enable the robot to interrupt its current task to execute a higher-priority task.

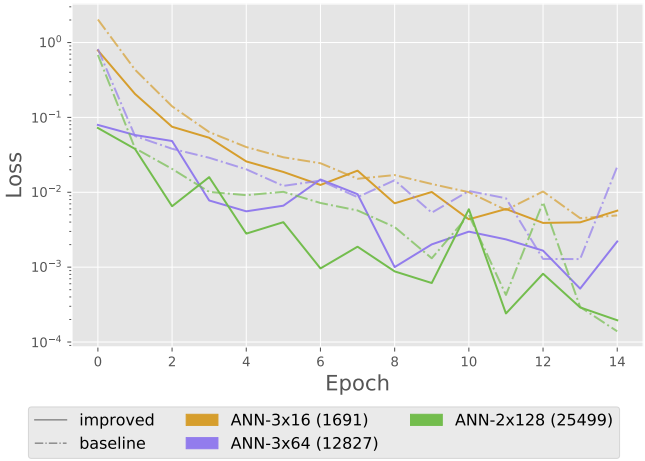

Figure 6: Learning curves for acquiring models for the hand gestures using different deep network structures; models with low loss are obtained over a few epochs when guided by reasoning.

direct this learning by limiting scope to the relevant object constants and description. For example, assume that the robot is shown a new gesture for seating a group of customers at a table. The robot will use human feedback about this new gesture, and only consider literals corresponding to: the location of these customers, its own location, and the occupancy of tables in the restaurant, to learn axioms for the new action.

Tracing explanations. Our architecture supports the ability to infer the sequence of axioms and beliefs that explains the evolution of any given belief or the non-selection of any given ground action at a given time. We build on the idea of *proof trees*, which have been used to explain observations in classical first-order logic [35], and adapt it to our architecture that is based on descriptions in non-monotonic logic. Our approach is based on the following sequence of steps:

1. Select axioms that have the target belief or action in the head.
2. Ground literals in each such axiom's body and check whether these ground literals are supported (i.e., satisfied) by the current answer set.
3. Create a new branch in the proof tree (that has the target belief or action as root) for each selected axiom supported by the current answer set, and store the axiom and the related supporting ground literals in suitable nodes.
4. Repeats Steps 1-3 with the supporting ground literals in Step 3 as target beliefs in Step 1, until all branches reach a leaf node without further supporting axioms.

Paths from the root to the leaves in these trees provide explanations. If multiple such paths exist, we currently select one of the shortest branches at random; other heuristics could be used to compare the explanations. For example, if the robot is asked why it seated a group of three customers at $Table_5$, it can trace the current belief about the group back to the initial state through the application of relevant axioms, and come up with an explanation such as: "The three customers came to the restaurant and wanted to be seated as a group. $Table_5$ at node n_7 was the table closest to the entrance that had the desired number of seats available. I seated the customers at $Table_5$".

In addition to tracing the evolution of a target belief and justifying the non-selection of a particular action, our architecture can also provide: (a) a description of any computed or executed plan in terms of literals in the plan; (b) justification for executing a particular action at a particular time step by examining the change in state caused by the action's execution and how this state change achieves the goal or facilitates the execution of the next action in the plan; and (c) inferred outcome(s) of the execution of hypothetical actions based on a mental simulation guided by the current domain knowledge. In all these cases, the identified literals are encapsulated in a prespecified answer template to provide the descriptions. For proof of concept examples in simplistic scene understanding scenarios, please see [9]; some specific examples in the RW domain are provided below (Section 4.1).

Control loop. Algorithm 1 is the overall control loop for the architecture. The baseline behavior (lines 3-8) is to plan and execute actions to achieve the given goal as long as a consistent model of history can be computed. If such a model cannot be constructed, it is attributed to

Algorithm 1: Our architecture's control loop.

Input: $\Pi(\mathcal{D}, \mathcal{H})$; goal description; initial state σ_1.
Output: Control signals for robot to execute.

```
1  planMode = true, learnExplainMode = false
2  while true do
3      Add observations to history.
4      ComputeAnswerSets(Π(D, H))
5      if planMode then
6          if existsGoal then
7              if explainedObs then
8                  ExecutePlanStep()
9              else
10                 planMode = false
11                 learnExplainMode = true
12             end
13         else
14             learnExplainMode = true
15         end
16     else
17         if interrupt then
18             planMode = true
19         else if learnExplainMode then
20             AcquireKnowledgeExplain()
21     end
22 end
```

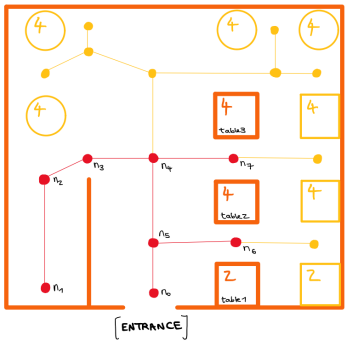

Figure 7: Example layout of the RW domain used in Execution Examples 1- 2.

an unexplained, unexpected observation, and the robot triggers interactive exploration (lines 9-12). Interactive exploration is also triggered if no active goal exists to be achieved (lines 13-15). Depending on the human input, the architecture either acquires the previously unknown gestures and axioms, or attempts to provide the desired description of a target decision or belief (lines 19-21). When in the learning mode, the robot can be interrupted if needed (lines 17-18), e.g., to pursue a new goal.

4. Execution Traces and Results

Meaningfully evaluating architectures for integrated robot systems is challenging. It is difficult to find a baseline that provides all the capabilities supported by our architecture, and it is also difficult to evaluate the capabilities of each component of the architecture in isolation. Also, given that reasoning and learning guide each other in our architecture to automatically identify and focus only on the relevant information, task complexity and scalability do not necessarily change substantially by increasing the number of tasks, and just reporting success in many scenarios is not very informative. In addition, it was difficult to use a physical robot to conduct the experimental trials during the pandemic. We thus focus on illustrating the capabilities of our architecture using a combination of execution traces (i.e., use cases) and some experiments that provide quantitative results. The key hypotheses to be evaluated are:

- **H1** : our architecture enables the robot to compute and execute plans to achieve desired goals;
- **H2** : having reasoning inform and guide learning improves computational efficiency of learning and recognition accuracy of the learned models; and
- **H3** : exploiting the links between reasoning and learning provides suitable relational descriptions as explanations of decisions and beliefs.

We explore hypotheses **H1** and **H3** in the execution traces (Section 4.1), and provide experimental results in support of **H2** (Section 4.2).

4.1. Execution traces

We provide two execution traces to illustrate the operation of our architecture in specific scenarios. *Videos corresponding to these traces can be viewed online* [29][2]. In all the scenarios, the human user (in the physical world) uses hand gestures to create different situations and also to mimic the gestures to be made by the customers or the supervisor in the restaurant environment. The layout used to generate these traces is shown in Figure 7; it is simplified version of Figure 3.

Execution Example 1. *[Plan, execute, explain]*
Consider a scenario in which there is one customer cu_1 seated at $table_1$ in the restaurant, and the robot waiter is in the region of node n_4. In this scenario, the restaurant is organized into regions corresponding to eight nodes: $n_0 - n_7$. The subsequent steps in this scenario are:

- Three new customers ($cu_2 - cu_4$) are introduced in the restaurant as a group by the human designer showing a suitable hand gesture. This information is also added to the ASP program automatically.

[2]https://www.cs.bham.ac.uk/~sridharm/KR22/

- The hand gesture also lets the robot waiter (rob_1) know that the new customers are to be seated at a table. The robot comes up with a plan based on the updated ASP program and the vacant table that is closest to it:

$$move(rob_1, n_5),\ move(rob_1, n_0),$$
$$pickup(rob_1, group_1),\ move(rob_1, n_5),$$
$$move(rob_1, n_6), seat(rob_1, group_1, table_2)$$

- Note that applying the *pickup* action to any customer in a group causes the same effect on all customers in the group. This plan is executed and the state is updated accordingly, e.g., $cu_2 - cu_4$ are seated at $table_2$ after the plan is executed.
- The robot can be asked about the executed plan.
  Human: "why did you seat all the customers at $table_2$?"
  Pepper: "Because all the customers wanted to sit together and $table_2$ was the closest available table."

- After some time, cu_1 has finished eating and would like to leave. The designer imitates the hand gesture that the customer would do in the restaurant to ask for the bill. This is translated into a goal in the ASP program: $haspaid(cu_1)$.
- The robot computes and executes a suitable plan to give the bill to cu_1, collect payment, and provide a receipt, after which cu_1 leaves the restaurant.

Figure 8 shows snapshots from the beginning, middle, and end of this scenario.

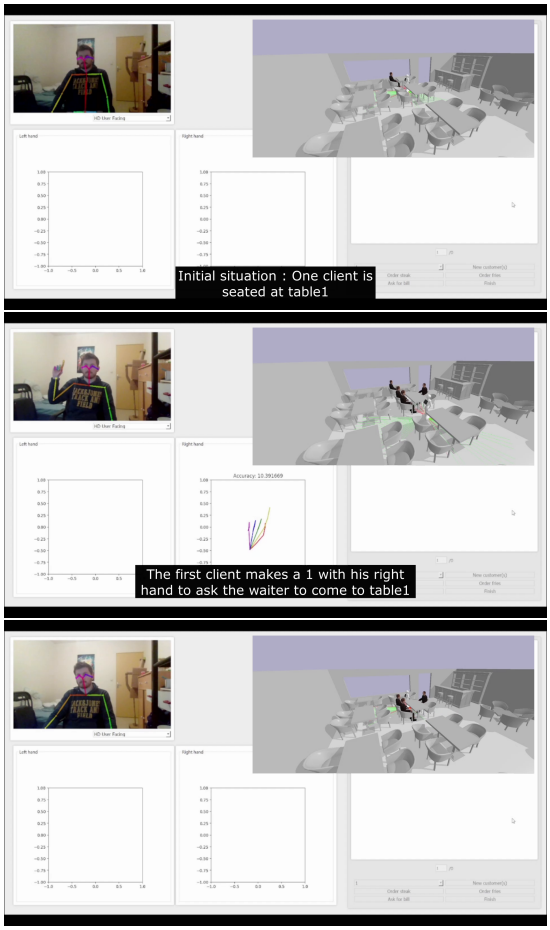

Figure 8: Snapshots from the beginning, middle, and end of scenario in Execution Example 1: (top) there is initially one customer cu_1 seated at $table_1$; (middle) the three new customers are at $table_2$ and cu_1 gets the robot waiter's attention to request the bill; and (bottom) cu_1 has left the restaurant after paying the bill.

Execution Example 2. *[Learn, plan, explain]*
Consider another scenario in which the restaurant initially has no customers. Robot waiter rob_1 is in the region of node n_1 and knows that $table_1$ and $table_2$ have capacity two and four respectively. Once again, the restaurant is organized into regions corresponding to eight nodes: $n_0 - n_7$. The subsequent steps in this scenario are:

- The human (in the physical world) makes a hand gesture that is unknown to the robot waiter. The robot responds by identifying this as a new gesture and conveys that this will be added to the database of hand gestures.
- Robot adds the new hand gesture and solicits feedback about the gesture. The human (designer) intentionally provides a complex instruction (textually) that this gesture corresponds to "serve steak to a group of three new customers, and then give them the bill".

- Since rob_1 knows that serving a customer implies giving them the food item they want, it is able to parse this complex instruction into the component actions. When the human then makes the same hand gesture again and introduces three new customers ($cu_2 - cu_4$) near the restaurant's entrance, rob_1 computes a suitable plan (some steps omitted to promote understanding).

$$move(rob_1, n_2), \ldots, pickup(rob_1, cu_2), \ldots,$$
$$seat(rob_1, cu_2, table_2), \ldots,$$
$$serve(rob_1, steak, table_2), \ldots,$$
$$givebill(rob_1, table_2), \ldots,$$

- Plan is executed and the state is updated accordingly at different time steps, e.g., $cu_2 - cu_4$ are

Figure 9: Snapshots from the beginning, middle, and end of scenario in Execution Example 2: (top) there is initially no customer in the restaurant; (middle) the newly learned hand gesture is made to get the robot to serve steak to a group of customers; and (end) the robot provides a bill to the customers after they have completed their meal.

seated at $table_2$ after the $seat$ action is executed.

- The robot can be asked about specific plan steps.
  Human: "why did you not serve pasta to $table_2$?"
  Pepper: "Because all customers at $table_2$ wanted to eat steak."

  This explanation is based on the previously-described approach to trace beliefs and the application of relevant axioms.

Figure 9 shows snapshots from the beginning, middle, and end of this scenario.

We evaluated the architecture in many other scenarios grounded in the motivating (restaurant) domain; the robot was able to successfully compute and execute plans to achieve the assigned goals, identify and learn previously unknown knowledge, and provide on-demand explanations of decision and beliefs.

4.2. Experimental results

To further explore the effect of reasoning guiding learning, we conducted some quantitative studies. The first experiment examined the benefits of reasoning guiding the learning of deep network models for hand gestures. Deep learning methods typically need many labeled training examples and epochs to learn models for the target classification task. However, since learning in our architecture is constrained (by reasoning) to specific gestures or classes of gestures at a time, it took fewer samples and fewer epochs to acquire the desired models that provide high accuracy—see Figure 10.

The second experiment examined whether reasoning helped improve the recognition accuracy. In this experiment, we considered 30 hand gestures. One round of testing included 40 iterations of each hand gesture by a person who did not participate in training. We conducted multiple rounds of testing and ground truth information was provided by the designers (i.e., student authors). In the absence of the coupling between reasoning and learning, the learned models had (on average) an accuracy of 85% over the different hand gestures. However, with learning being directed to specific (classes of) gestures, the learned models resulted in better classification accuracy—$\approx 100\%$.

The third experiment examined the ability to provide explanatory descriptions in response to different types of queries in different situations. A description was considered to be correct if it had all the correct literals but no additional literals. Overall, the interplay between reasoning (with relevant knowledge) and learning (of previously unknown knowledge) led to the correct relational descriptions in 95% cases, with the "errors" being descriptions containing additional literals that were not essential to answer the query posed but were not necessarily wrong. In the absence of the learned knowledge, the accuracy (averaged over query types) was $65 - 80\%$.

5. Discussion and Conclusions

We conclude by highlighting the key capabilities of our architecture:

- Once the designer has provided the domain-specific information (e.g., arrangement of rooms, range of robot's sensors), planning, diagnostics, and plan execution can be automated. The coupling between reasoning and learning enables more complex theories (of cognition, action) to be encoded without increasing the computational effort substantially.

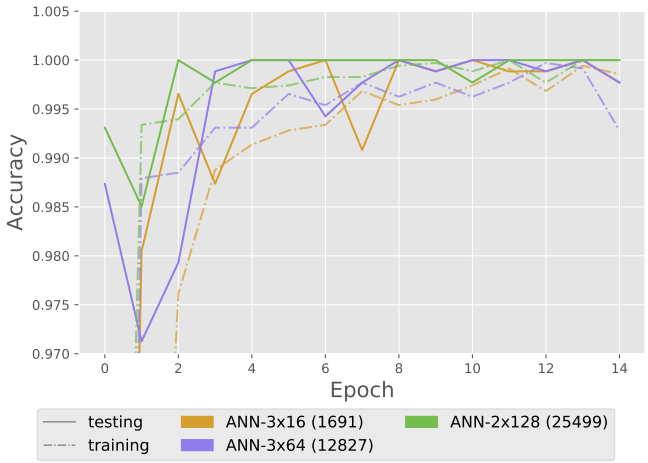

Figure 10: Deep network models provide high (recognition) accuracy for hand gestures within a few epochs when guided by reasoning.

- Second, exploiting the interplay between knowledge-based reasoning and data-driven learning provides a clear separation of concerns, and helps focus attention automatically to the relevant knowledge and observed anomalies, thus improving the reliability and efficiency of reasoning and learning.
- Third, it is easier to understand and modify the observed behavior than with architectures that consider all the available knowledge or only support data-driven learning. The robot is able to provide relational descriptions of its decisions and the evolution of its beliefs.
- Fourth, there is smooth transfer of control and relevant knowledge between components of the architecture, and increased confidence in the correctness of the robot's behavior. Also, the underlying methodology can be used with different robots and in different application domains.
- Fifth, using KR tools and the coupling between reasoning and learning as the foundation promotes modularity and simplifies the design and evaluation of architectures for integrated robot systems.

Future work will further explore the interplay between reasoning and learning for explaining decisions and beliefs while performing reasoning and learning in more complex robotics domains. We will also investigate the use of our architecture on a physical robot interacting with humans through noisy sensors and actuators. The longer-term objective is to support transparent reasoning and learning in integrated robot systems operating in complex domains.

References

[1] E. Erdem, V. Patoglu, Applications of ASP in Robotics, Kunstliche Intelligenz 32 (2018) 143–149.

[2] E. Erdem, M. Gelfond, N. Leone, Applications of Answer Set Programming, AI Magazine 37 (2016) 53–68.

[3] K. Kersting, L. D. Raedt, Bayesian Logic Programs, in: International Conference on Logic Programming, London, UK, 2000.

[4] L. D. Raedt, A. Kimmig, Probabilistic Logic Programming Concepts, Machine Learning 100 (2015) 5–47.

[5] M. Richardson, P. Domingos, Markov Logic Networks, Machine Learning 62 (2006) 107–136.

[6] S. Zhang, M. Sridharan, A Survey of Knowledge-based Sequential Decision Making under Uncertainty, Artificial Intelligene Magazine 43 (2022) 249–266.

[7] Y. Gil, Learning by Experimentation: Incremental Refinement of Incomplete Planning Domains, in: International Conference on Machine Learning, New Brunswick, USA, 1994, pp. 87–95.

[8] M. Law, A. Russo, K. Broda, The ILASP System for Inductive Learning of Answer Set Program, Association for Logic Programming Newsletter (2020).

[9] T. Mota, M. Sridharan, A. Leonardis, Integrated Commonsense Reasoning and Deep Learning for Transparent Decision Making in Robotics, Springer Nature CS 2 (2021) 1–18.

[10] M. Sridharan, B. Meadows, Knowledge Representation and Interactive Learning of Domain Knowledge for Human-Robot Collaboration, Advances in Cog-

nitive Systems 7 (2018) 77–96.
[11] J. E. Laird, K. Gluck, J. Anderson, K. D. Forbus, O. C. Jenkins, C. Lebiere, D. Salvucci, M. Scheutz, A. Thomaz, G. Trafton, R. E. Wray, S. Mohan, J. R. Kirk, Interactive Task Learning, IEEE Intelligent Systems 32 (2017) 6–21.
[12] R. Assaf, A. Schumann, Explainable Deep Neural Networks for Multivariate Time Series Predictions, in: International Joint Conference on Artificial Intelligence, Macao, China, 2019, pp. 6488–6490.
[13] Wojciech Samek and Thomas Wiegand and Klaus-Robert Muller, Explainable Artificial Intelligence: Understanding, Visualizing and Interpreting Deep Learning Models, ITU Journal: ICT Discoveries (Special Issue 1): The Impact of Artificial Intelligence (AI) on Communication Networks and Services 1 (2017) 1–10.
[14] W. Norcliffe-Brown, E. Vafeais, S. Parisot, Learning Conditioned Graph Structures for Interpretable Visual Question Answering, in: Neural Information Processing Systems, Montreal, Canada, 2018.
[15] K. Yi, J. Wu, C. Gan, A. Torralba, P. Kohli, J. B. Tenenbaum, Neural-Symbolic VQA: Disentangling Reasoning from Vision and Language Understanding, in: Neural Information Processing Systems, Montreal, Canada, 2018.
[16] M. Ribeiro, S. Singh, C. Guestrin, Why Should I Trust You? Explaining the Predictions of Any Classifier, in: ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2016, pp. 1135–1144.
[17] Y. Zhang, S. Sreedharan, A. Kulkarni, T. Chakraborti, H. H. Zhuo, S. Kambhampati, Plan explicability and predictability for robot task planning, in: International Conference on Robotics and Automation, 2017, pp. 1313–1320.
[18] R. Borgo, M. Cashmore, D. Magazzeni, Towards Providing Explanations for AI Planner Decisions, in: IJCAI Workshop on Explainable Artificial Intelligence, 2018, pp. 11–17.
[19] P. Bercher, S. Biundo, T. Geier, T. Hoernle, F. Nothdurft, F. Richter, B. Schattenberg, Plan, repair, execute, explain - how planning helps to assemble your home theater, in: Twenty-Fourth International Conference on Automated Planning and Scheduling, 2014.
[20] J. Fandinno, C. Schulz, Answering the "Why" in Answer Set Programming: A Survey of Explanation Approaches, Theory and Practice of Logic Programming 19 (2019) 114–203.
[21] S. Anjomshoae, A. Najjar, D. Calvaresi, K. Framling, Explainable agents and robots: Results from a systematic literature review, in: International Conference on Autonomous Agents and Multiagent Systems (AAMAS), Montreal, Canada, 2019.
[22] T. Miller, Explanations in Artificial Intelligence: Insights from the Social Sciences, Artificial Intelligence 267 (2019) 1–38.
[23] M. Sridharan, M. Gelfond, S. Zhang, J. Wyatt, REBA: A Refinement-Based Architecture for Knowledge Representation and Reasoning in Robotics, Journal of Artificial Intelligence Research 65 (2019) 87–180.
[24] P. Langley, B. Meadows, M. Sridharan, D. Choi, Explainable Agency for Intelligent Autonomous Systems, in: Innovative Applications of Artificial Intelligence, San Francisco, USA, 2017.
[25] M. Sridharan, B. Meadows, Towards a Theory of Explanations for Human-Robot Collaboration, Kunstliche Intelligenz 33 (2019) 331–342.
[26] T. Mota, M. Sridharan, Commonsense Reasoning and Knowledge Acquisition to Guide Deep Learning on Robots, in: Robotics Science and Systems, Freiburg, Germany, 2019.
[27] M. Balduccini, M. Gelfond, Logic Programs with Consistency-Restoring Rules, in: AAAI Spring Symposium on Logical Formalization of Commonsense Reasoning, 2003, pp. 9–18.
[28] M. Gelfond, D. Inclezan, Some Properties of System Descriptions of AL_d, Journal of Applied Non-Classical Logics, Special Issue on Equilibrium Logic and Answer Set Programming 23 (2013) 105–120.
[29] M. Sridharan, Supporting code and videos, 2022. https://www.cs.bham.ac.uk/~sridharm/KRFiles/.
[30] E. Balai, M. Gelfond, Y. Zhang, Towards Answer Set Programming with Sorts, in: International Conference on Logic Programming and Nonmonotonic Reasoning, Corunna, Spain, 2013.
[31] B. Banihashemi, G. D. Giacomo, Y. Lesperance, Abstraction of Agents Executing Online and their Abilities in Situation Calculus, in: International Joint Conference on Artificial Intelligence, Stockholm, Sweden, 2018.
[32] Z. Saribatur, T. Eiter, P. Schuller, Abstraction for Non-ground Answer Set Programs, Artificial Intelligence 300 (2021) 103563.
[33] E. Coumans, Y. Bai, PyBullet: A Python Module for Physics Simulation for Games, Robotics, and Machine Learning, Technical Report, http://pybullet.org, 2016-2022.
[34] Z. Cao, G. Hidalgo Martinez, T. Simon, S. Wei, Y. A. Sheikh, OpenPose: Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields, IEEE Transactions on Pattern Analysis and Machine Intelligence (2019).
[35] G. Ferrand, W. Lessaint, A. Tessier, Explanations and Proof Trees, Computing and Informatics 25 (2006) 1001–1021.