LARA – Human-guided collaborative problem solver: Effective integration of learning, reasoning and communication

Harsha Kokel[1] HKOKEL@UTDALLAS.EDU
University of Texas at Dallas, TX, USA

Mayukh Das[1,2] MAYUKHDAS@MICROSOFT.COM
Microsoft Research, Bangalore, India

Rakibul Islam[3] MDRAKIBUL.ISLAM@WSU.EDU
Washington State University Pullman, WA, USA

Julia Bonn[4] JULIA.BONN@COLORADO.EDU
Jon Cai[4] JON.Z.CAI@COLORADO.EDU
University of Colorado at Boulder, CO, USA

Soham Dan[5] SOHAMDAN@SEAS.UPENN.EDU
University of Pennsylvania, PA, USA

Anjali Narayan-Chen[6] NRYNCHN2@ILLINOIS.EDU
Prashant Jayannavar[6] PAJ3@ILLINOIS.EDU
University of Illinois at Urbana-Champaign, IL, USA

Janardhan Rao Doppa[3] JANA.DOPPA@WSU.EDU
Washington State University Pullman, WA, USA

Julia Hockenmaier[6] JULIAHMR@ILLINOIS.EDU
University of Illinois at Urbana-Champaign, IL, USA

Sriraam Natarajan[1] SRIRAAM.NATARAJAN@UTDALLAS.EDU
University of Texas at Dallas, TX, USA

Martha Palmer[4] MARTHA.PALMER@COLORADO.EDU
University of Colorado at Boulder, CO, USA

Dan Roth[5] DANROTH@SEAS.UPENN.EDU
University of Pennsylvania, PA, USA

Abstract

We consider the problem of human-machine collaboration in the context of collaborative building in Minecraft. To this effect, we present an integrated system (LARA) that builds on advancements in several related fields - NLP, knowledge representation, inductive logic programming, planning and statistical relational AI. Specifically, LARA consists of a language parser and generator for effective communication, a rich representation based on first-order logic that allows for generalization, a concept learner that is capable of generalizing from a small number of instances by effectively exploiting human guidance and a planner capable of exploiting domain knowledge effectively. The resulting integrated system has been demonstrated as part of DARPA's Communicating with Computers (CwC) program and is presented in detail here.

1. Introduction

It is well known that human-machine collaborative planning and problem solving is quite challenging as it requires shared perception of the world, sophisticated language understanding, fluent execution, bi-directional communication, and contextual understanding. Specifically, we consider the task of collaborative building in Minecraft (Kokel et al., 2021) and develop *an integrated system* that builds on several different areas – hierarchical planning (Bercher et al., 2019; Erol et al., 1994; Nau et al., 1999), knowledge representation and reasoning (Brachman & Levesque, 2004), inductive logic programming (Cropper & Dumancic, 2022; Muggleton & Raedt, 1994; Raedt & Kersting, 2008), knowledge-based learning (Towell & Shavlik, 1994; Kokel et al., 2020), natural language processing (Banarescu et al., 2013; Bahdanau et al., 2015; Sutskever et al., 2014) and generation (Gatt & Krahmer, 2018) and statistical relational AI (Raedt et al., 2016).

It is natural to focus on the modality of communication such as gestures, or natural language when building human-AI collaborative systems. However, it is also essential to establish a common vocabulary for communication. This is especially important in a complex domain such as Minecraft where given the basic definitions of elementary shapes and sizes, higher-order concepts should be built. The key requirement is that the common vocabulary of concepts keeps growing as more tasks are solved and the interactions increase. The set of concepts should be easily learnable (with a small number of examples) and generalizable. To this effect, we assume the existence of a basic vocabulary and build upon an *inductive logic programming based concept learner* (Das et al., 2020). This concept learner learns a set of hierarchical concepts based on a very small (possibly one) number of examples using domain knowledge as an inductive bias.

While learning these generalized concepts, there is a necessity for the system to continue interacting in the environment and modifying its interactions based on the induced concepts and the feedback both from the environment and the human. The resulting plan must be both generalizable to a different number of objects, and compositional to employ effectively the hierarchies of concepts that are induced by the concept learner. To this effect, the induced concepts are used as preconditions to guide a hierarchical task planner (Das et al., 2018) in our framework. The planner uses domain knowledge and actively seeks human guidance in the form of knowledge constraints that are then used for both efficient and effective planning.

Finally, for effective communication, both the modality and the representation need to be established. For modality, we employ the use of NLP parsers (both rule-based and neural-based parsers) to translate the commands from the humans in natural language to a formal internal representation. And for the reverse communication from the internal representation to natural language, we restrict ourselves to specific forms and templates. Extending this to allow for richer neural language generators is our envisioned future work. For the internal representation, we employ the use of abstract meaning representations (Banarescu et al., 2013) that allow for capturing generalized knowledge.

We make the following key contributions: (1) we present an end-to-end system called LARA, Planning and learning via communication, that obtains instructions and knowledge in rich natural language, reasons with the observations and knowledge, and executes the plan automatically. (2) The system is capable of obtaining human "advice" as constraints both to learn the hierarchical concepts and to perform planning. (3) Most importantly, the system is capable of soliciting this advice in an active manner, thus reducing the effort needed from the human in collaborative planning and execution.

Unlike a traditional research paper, the integrated system paper is organized as follows: we introduce the necessary background of the components in the system, then present the overall system design with example scenarios and provide images of the demonstration (along with the link to the full video). Finally, we conclude the paper by discussing the salient features of the system and outlining areas of future research.

2. Background

2.1 Hierarchical planning

Hierarchical planning (Erol et al., 1994; Nau et al., 1999) consists of two types of tasks: *primitive* and *compound*. A *primitive task* is defined using preconditions and effects (equivalent to operators in classical planning). A primitive task is a single-step action that can only be executed in a state which satisfies the task preconditions. Upon executing a primitive task the propositions of the state change as dictated by the task effects. A *compound task* is a temporally extended action that requires one or more primitive tasks to be executed in a partially ordered manner.

In hierarchical planning, one or more methods are defined for each compound task. A method is described as a three tuple, consisting of compound tasks, preconditions, and a sequence of partially ordered tasks (primitive or compound). When a state satisfies the method precondition, the compound task can be decomposed in that state to the sequence of partially ordered tasks. The problem of achieving a goal condition, starting from a given initial state, is also posed as a compound task (or set of compound tasks). Each compound task is recursively decomposed using methods, till a satisfying sequence of primitive tasks is identified. A sequence of primitive tasks is considered satisfying if executing that sequence starting at the initial state results in a state that satisfies the goal condition.

While classical planners use a common input language called *Planning Domain Description Language* (PDDL) (Fox & Long, 2003), PDDL does not support compound tasks and methods. Recently, Höller et al. (2020) proposed a *Hierarchical Domain Description Language* (HDDL) as an extension of the STRIPS fragment of PDDL2.1 defined in Fox & Long (2003). Both PDDL and HDDL are first-order languages and use first-order formulas over predicates for goal description.

2.2 Concept learning

Inductive logic programming (ILP) learns rules or relations inductively from the given set of background knowledge and positive as well as negative examples (Muggleton & Raedt, 1994). The rules or relations in ILP are learned as declarative logic programs, often as horn clauses. *Concept learning* in ILP, essentially, reduces to learning a clausal theory (a first-order logic program) that covers as many positive examples of the target concept as possible and as few negative examples as possible (Raedt, 1997). ILP employs background knowledge as search bias to constrain the space of the hypothesis. Golem (Muggleton & Feng, 1990), FOIL (Quinlan, 1990), Progol (Muggleton, 1995), TILDE (Blockeel & Raedt, 1998), Alpeh (Srinivasan, 1999), FOCL (Pazzani et al., 1991) are some examples of ILP systems that learn the target concept by induction. One important aspect of a cogent concept learning framework is that it should represent concept hierarchies, allowing us to induce more complex concepts given previously learned ones, as opposed to learning traditional logic programs with one level of abstraction (Fu & Buchanan, 1985)

2.3 Neural parsers

Abstract Meaning Representation (AMR) is a semantic representation of natural language (Banarescu et al., 2013), where concepts are represented as nodes and relations between concepts are represented with edges. AMR can be systematically translated to first-order logic (Bos, 2016) and, hence, is a suitable representation for integration with ILP systems. With the tremendous success of neural networks to solve various sequence-to-sequence problems (Bahdanau et al., 2015; Sutskever et al., 2014), multiple neural models are proposed for parsing text to AMRs (Xu et al., 2020; Konstas et al., 2017; Zhang et al., 2019; Peng et al., 2017).

2.4 Minecraft

Minecraft is a popular computer-based game developed by Mojang Studios and released in 2011[1]. Minecraft exposes a virtual 3D world where player avatars can explore the world, travel on adventures, build structures, hunt for food, harvest raw materials, craft tools, and kill zombies. It poses many challenging problems which have intrigued researchers to use it as a test bed for various open problems in AI (Kanervisto et al., 2021; Salge et al., 2020; Shah et al., 2021). Project Malmo (Johnson et al., 2016), built on top of Minecraft, exposes an API for flexible AI experiments.

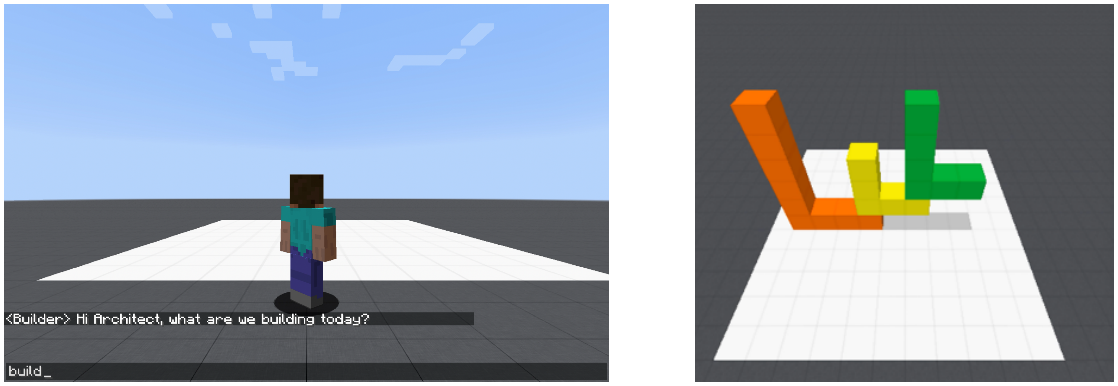
3. LARA - Planning and learning via communication

We now present the details of our collaborative problem solving system and discuss each of the components in detail.

3.1 Problem definition

This work considers the problem of human-machine collaboration in a Minecraft environment. A *collaborative building task* (Jayannavar et al., 2020; Kokel et al., 2021; Narayan-Chen et al., 2019;

1. https://minecraft.net



(a) (b)

Figure 1: **(a)** Minecraft builder screen showing the 3D build region and the chat interface. **(b)** Example of a target structure in the oracle screen. The architect can see both screens.

Narayan-Chen, 2020) is defined in the context of Minecraft where a human and a machine have to collaborate to build a target structure by block placements. Blocks are restricted to be one of six colors: red, blue, green, purple, orange, and yellow. A fixed 3D grid of size $11 \times 9 \times 11$ is defined as a build region, as shown in Figure 1a. The avatar can only move and place blocks in this build region. The task is to build a target structure in this stipulated build region. The target structure candidate is sampled from a preset list of complex shapes and displayed in a separate oracle window. An example of a target structure is shown in Figure 1b. Two players, an architect and a builder, collaborate and communicate using natural language via the chat interface. The architect and the builder take turns on the chat window.

The architect can view the target structure in the oracle window and can also see the current state of the build region. The builder can not see the oracle window. It can move in the build region to place and remove blocks. The role of the architect is played by humans and the role of the builder is played by LARA. The game begins with a simple target structure in the oracle window and a greeting from the builder LARA to the architect (as seen in Figure 1a).

For a successful target structure construction and most importantly, generalization of the learned concepts, the architect must decompose the target structure into smaller structures and instruct LARA to achieve those subtasks. LARA must parse the instructions, seek clarifications as appropriate, and execute the subtask(s). A sample interaction between an architect and the builder LARA is shown in Figure 2, where the target structure is a red *L*. The challenges posed by the Minecraft-based blocks world task are

1. the communication between the architect and the builder is inherently bi-directional (see for example Figure 2).
2. the builder should be able to seek clarifications as required.

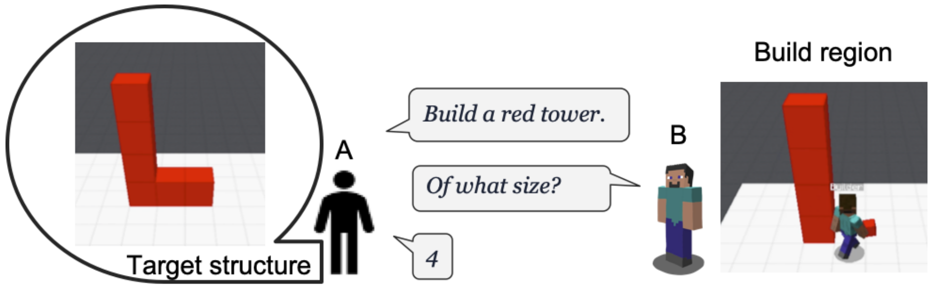


Figure 2: Target structure on the left is visible only to the architect (A). The architect instructs the builder (B) to build a red tower. B seeks clarification about the size and then proceeds to build the tower in the build region.

3. both players must share some initial structures in the vocabulary, expand the vocabulary with experience, agree upon the changes and reuse the learned higher-level concepts as appropriate. This requires an effective reasoning process over the learned concepts.

These problems of the proposed task highlight the key challenges of the collaborative planning problem: *bi-directional communication, contextual understanding, composable vocabulary and a powerful concept learner that can induce new, rich concepts based on limited interaction and experience*.

Our **key contribution in this work is the demonstration of our collaborative planning and problem-solving agent** that addresses these key challenges. Our system has the capability to understand, quantify and measure “what-it-doesn’t-know" (dearth of relevant information) and leverage that understanding to elicit “advice/knowledge/constraints" at the most appropriate decision points from the human and potentially learn better plans for increasingly complex structures. Some recent works (for e.g. Narayan-Chen et al. 2019 and Köhn et al. 2020) introduced a similar Minecraft environment, but focused on the dialogue generation and instruction giving; instead of the dialogue understanding, concept induction, and planning challenges we focus on here.

3.2 System setup

For the collaborative building task in Minecraft, a few essential pieces of prior knowledge (domain information) are assumed to exist in both the builder and the architect. These essentials include directions, primitive structures, block indicators, and six colors. For simplicity, we only use the directions w.r.t the architect’s viewpoint. Eight primitive structures include a block, tower, row, column, cube, cuboid, square, and rectangle. Five of them are shown in Figure 3a. Three primitive shapes not shown here include a single block, a square lying on the floor, and a rectangle lying on the floor. Block indicators include guides for pointing to a single block that is a component of the structure, a few examples are shown in Figure 3b–3d. Additionally, the terms height, width, and length are used to represent the size of the structure from top-end to bottom-end, left-end to right-end, and front-end to back-end, respectively. So, the tower size is its height, the row size is its width, and the column size is its length. These elements of initial knowledge are, in essence, primitives of the domain that are later expanded upon by the learner.

With this initial knowledge, the architect instructs the builder to build the target structure. In our system, the architect can either write a natural language text to instruct the builder or write “UNDO" to revoke the last instruction. Upon receiving the instruction, the builder can either execute the instruction, or seek additional information for clarification (see for e.g. Fig. 2), provide prompts so that the architect can provide instruction in a fashion that is comprehensible to the builder. Once the target structure is constructed, the architect states “done". This would instruct the builder that the task of building the target structure is accomplished in the build region. When the structure building is completed, the builder offers to remember the structure for reuse. The architect can then provide a name for the structure and describe its height, width, and length to the builder. The builder might ask some yes/no questions to induce a generalized concept of the structure (Das et al., 2020). If the builder is successful, the new structure is then added to the builder’s capabilities and can then be treated as a lower-level structure. The process continues so that more complex concepts are introduced as needed.

3.3 System Architecture

The architecture of our system LARA is illustrated in Figure 4 that integrates different research components to develop a human-machine collaborative system. It primarily consists of four key components: *Minecraft simulator*, *NLP engine*, *Planner*, and *Concept learner*. The Minecraft system and the interfacing APIs form the Minecraft simulator. The module processing natural language

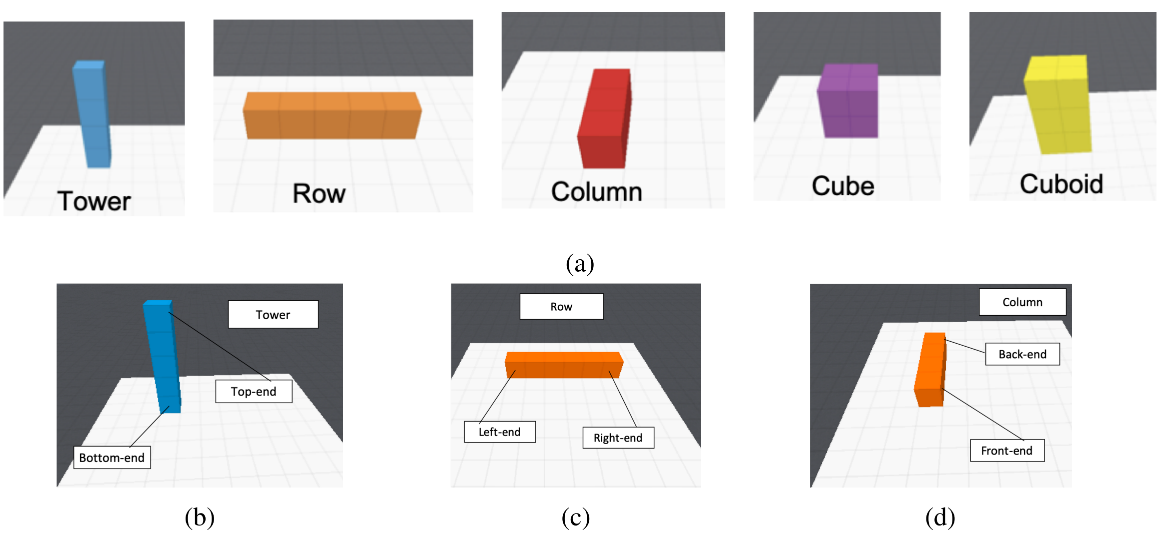


Figure 3: (a) Five of the eight primitive structures: tower, row, column, cube, and cuboid. Block indicators that point to a single block of the structure. (b) top-end and bottom-end, (c) left-end and right-end, (d) back-end and front-end. For complex structures, these indicators can be combined, for example, the front-bottom-left block of a cube.

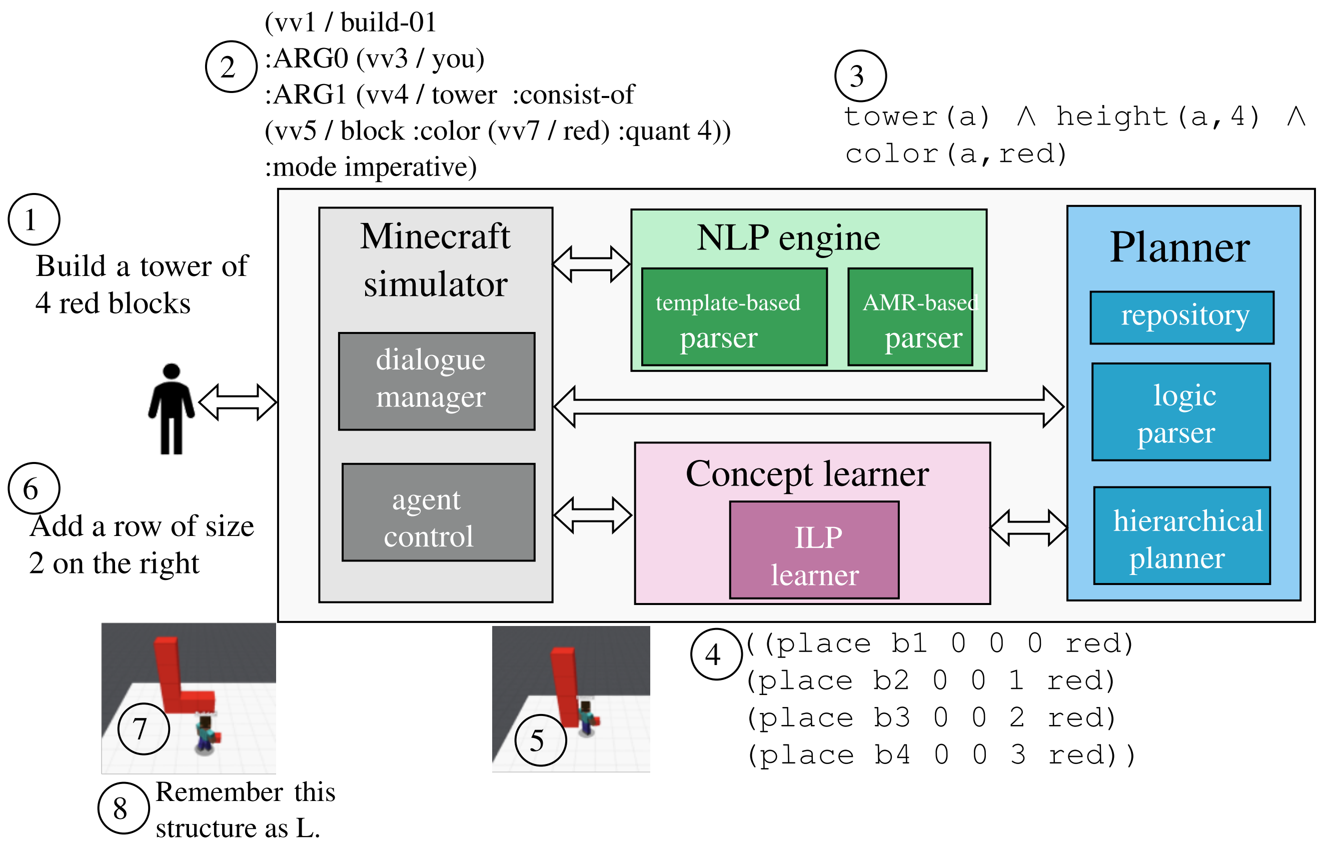


Figure 4: Architecture of LARA. It consists of Minecraft simulator, NLP engine, planner, and concept learner. An example flow of building the "L" shape is illustrated. ① Natural language instruction by the architect to build a red tower. ② AMR representation of the instruction parsed by AMR parser. ③ Logic representation of the instruction. ④ Block placement plan generated by the planner. ⑤ Agent control executes the plan in the build region. ⑥ Next instruction from the architect to add a row. ⑦ Agent control executes the next instruction in build region. ⑧ Structure saved as "L".

text forms the NLP engine. The module generating the action plan for the Minecraft avatar constitutes the planner. Finally, the module learning new structures forms the concept learner component. We now describe each of these components in greater detail.

3.3.1 Minecraft simulator

Project Malmo (Johnson et al., 2016) is extended for our collaborative building task by adding a dialogue manager and agent control module. The **dialogue manager** has two major responsibilities, 1. to triage the messages from the architect and 2. to generate natural language text for bi-directional communication. A message from the architect could either be a build instruction, a clarification, a concept explanation, or an UNDO operation. Dialogue Manager would accordingly pass forward the request to either the NLP engine, the planner, the concept learner, or agent-control module, respectively. For bi-directional communication, the dialogue manager maintains a fixed set of templates sentences with slots. These slots are then populated accordingly as required and sent via the chat interface. The set of template sentences maintained by the dialogue manager are provided in the Appendix[2]. The **agent control** module processes the plan generated by the planner and sends the action commands to MALMO API for execution in the Minecraft environment.

Predicates:

`{block/1, tower/1, row/1, cube/1, cuboid/1, square/1, rectangle/1, width/2 height/2, length/2, size/2, color/2, spatial_rel/3, top_end/2, bottom_end/2,` $\cdots$ `}`

Objects:

`{red, blue, green, purple, orange, yellow, east, west, north, south, top, bottom, a, b,` $\cdots$ `, z, aa, ab,` $\cdots$ `, zz, 0, 1, 2,` $\cdots$ `, 10}`

(a)

Build a tower of 4 red blocks.

`tower(a)` $\wedge$ `height(a,4)` $\wedge$ `color(a,red)`

Build a tower of 4 red blocks and add a row of size 2 on the right.

`tower(a)` $\wedge$ `color(a,red)` $\wedge$ `height(a,4)` $\wedge$ `row(b)` $\wedge$ `color(b,red)` $\wedge$ `width(b,2)` $\wedge$ `block(b1)` $\wedge$ `block(b2)` $\wedge$ `bottom_end(a,b1)` $\wedge$ `left_end(b,b2)` $\wedge$ `spatial_rel(west,b1,b2)`

(b)

Figure 5: FOL representation of collaborative building task. **(a)** FOL language (Complete list of predicates is deferred to Appendix) **(b)** Example FOL instructions.

3.3.2 NLP engine

The main job of the NLP engine is to parse the natural language instructions from the architect and provide a **generalized semantic representation** of such instructions for processing. We lever-

2. Appendix and demos available at https://bit.ly/3Rya01D

age first-order logic (FOL) to encode this semantic representation. We choose this for two specific practical reasons. First, FOL representation is compatible with the ILP-based concept learner. Second, Planning Domain Description Language (PDDL) is also a first-order predicate representation. Given the compatibility of FOL representation with the two components (the concept learner and the planner), it was a natural choice. While the above two reasons are from a pragmatic point of view, the use of FOL is necessary as there is a necessity to learn conceptual knowledge at multiple levels of abstraction – individual object level (ex., the red block), sets of objects (ex., the red blocks) or over all the objects. Learning and reasoning in such a rich space of abstractions is facilitated by the use of ILP and relational planners.

However, while it is possible to represent the complete semantics of the natural language instruction in FOL, it is also clear that for the purposes of this task, a restricted form of FOL would suffice. This restriction is necessary to maintain the tractability of learning and reasoning. The essential knowledge discussed earlier in Section 3.2 comprises the predicates of this language. Figure 5a presents the first-order language used to describe the collaborative building tasks. Figure 5b shows examples of FOL representations of natural language instructions.

To translate the natural language to FOL, we developed two independent NLP parsers: template-based and AMR-based. Our domain-specific **template-based parser** uses fixed set of templates consisting of slots. These slots are filled by the parser by going through the sentence and looking for matching short phrases. Once the slots are filled, the template is translated to the logic format. It is similar in spirit to *Template Matcher* parser by Jackson et al. (1991), with output in FOL. While our parser is quite fast, it has a few limitations. This parser maintains a *fixed* list of structures known to the builder and their dimensions, and thus has quite a limited vocabulary. It does not support all possible phrasing of an instruction and requires manual updates of templates to support new sentence formulations. To overcome these limitations, we built an AMR-based parser.

While the template-based parser uses predefined templates for translation of simpler sentences, the **AMR-based parser** supports free form sentences of varying complexity. As AMRs can be systematically translated to FOL (Bos, 2016), we use a neural parser to parse natural language text to AMRs. Figure 4 ② shows AMR representation of the sample natural language instruction presented earlier. AMR annotation had not been approached with spatial semantics in mind. Bonn et al. (2020) extends AMR with spatial addendum, which enables more expressive representation for spatial relationships required in the three-dimensional domain of Minecraft. Minecraft specific 3D structure building dialogues were collected between human architect and human builder and annotated with the new inventory of spatial rolesets (Narayan-Chen et al., 2019). A state-of-the-art neural AMR parser, STOG (Zhang et al., 2019), was trained on this Minecraft spatial AMR corpus (Bonn et al., 2020).

3.3.3 Planner

This component is responsible for providing the action sequence of placing blocks in the Minecraft environment. Wichlacz et al. (2019) show that a hierarchical planner is better suited than a classical planner for the Minecraft building task. Consequently, a **hierarchical planner**, JSHOP2 (Ilghami, 2006), is employed in this system to generate build plans. JSHOP2 uses a restricted version of HDDL, same as SHOP2 (Nau et al., 2003), and expects the goal description as a logical combination

(a)	(b)

tower(a) $\wedge$ **height(a,4)** $\wedge$ **color(a,red)**
`(tower 0 0 0 4 red)`
 	
tower(a) $\wedge$ **color(a,red)** $\wedge$ **height(a,4)** $\wedge$ **row(b)** $\wedge$ **color(b,red)** $\wedge$ **width(b,2)** $\wedge$ **block(b1)** $\wedge$ **block(b2)** $\wedge$ **bottom_end(a,b1)** $\wedge$ **left_end(b,b2)** $\wedge$ **spatial_rel(west,b1,b2)**
`(tower 0 0 0 4 red)`
`(row 0 1 0 2 red)`
	L(c) $\wedge$ **color(c,red)** $\wedge$ **height(c,4)** $\wedge$ **width(c,3)**
	
	Active advice queries to the architect (responses):
	1. Is the new height same as the height of the tower? (yes)
	2. Is the new width one more than the width of the row? (yes)
	3. Should the height of an L be one more than the width of the L? (no)

Figure 6: **(a)** Goal description of the FOL instruction. **(b)** FOL representation of new structure "L" and active advice queries to the architect.

of predicates defined in the planning domain. For a concise representation, we use the following format to define predicates for all the primitive structures,

```
(structure-name x-location y-location z-location
                    [height] [width] [length] color).
```

Following a LISP format, the predicate name is the first element of the tuple. The name of the structure is defined as a predicate name. The first three arguments are the X, Y, and Z coordinates of a pivotal block of the structure. The next three arguments represent the three dimensions of the structure. These dimensions, denoted within square brackets are not mandatory for all. Different structures have different dimensions, so the mandatory arguments change accordingly. For example, a tower only contains height. A tower predicate has 5 arguments. "`(tower 0 0 0 4 red)`" represents a tower with height 4. the color of the tower is red, and the pivotal block (i.e. lowest block) is at the $(0, 0, 0)$ location in the grid. A complete list of predicates in the planning domain are presented in the Appendix.

The **logic parser**, translates the FOL instruction to the goal description in the above format. Location coordinates are assigned to each structure in a way that respects the grid boundaries while achieving the spatial relations. Figure 6 presents example goal descriptions of two FOL instructions presented earlier. When any of the required dimension values are missing, the planner returns an INCOMPLETE GOAL DESCRIPTION error, with the missing dimension. Upon receiving this error, the dialogue manager generates a natural language question for that dimension and the goal description is completed from the response.

The **repository** contains the description of all the structures known to the builder. Initially, the repository only contains the mapping between the 8 primitive structures and dimensions; identifying the required argument for each primitive structure. Gradually, descriptions of new structures are also added to the repository. More details are provided in Section 3.3.4. When the goal description is complete, the goal description is provided to the hierarchical planner, along with the description of the current build region as the initial state. The hierarchical planner then provides a sequence of block placement or removal actions to transform the current build region to the goal description. An example plan is presented in Figure 4 ④.

3.3.4 Concept learner

While there have been various ILP systems proposed to learn concepts (Muggleton & Feng, 1990; Muggleton, 1995; Quinlan, 1990), they all require multiple (positive and negative) examples for each concept. Contrary to these approaches, our collaborative building task requires the agent to learn a new structure concept from just one positive example of the structure. To this effect, Das et al. (2020) developed a *Guided One-shot Concept Induction* (GOCI) framework. GOCI uses an ILP engine to learn candidate hypothesis for the concept and actively solicits advice from humans to prune the hypothesis space. These advice queries can be posed as simple yes/no questions to the architect. Consider learning a simple "L" structure from a single example of L shown in Figure 4. This example of "L" has red color, height 4, and width 3. It is composed of a tower and a row of size 4 and 2, respectively. The FOL representation of the new structure and the advice queried by GOCI are furnished in Figure 6b.

On receiving the responses from the architect, GOCI prunes the hypothesis space and finds the best suitable hypothesis to learn a generalized concept of L which is a composition of existing structures. In this current example, the concept L is learned as the following horn clause,

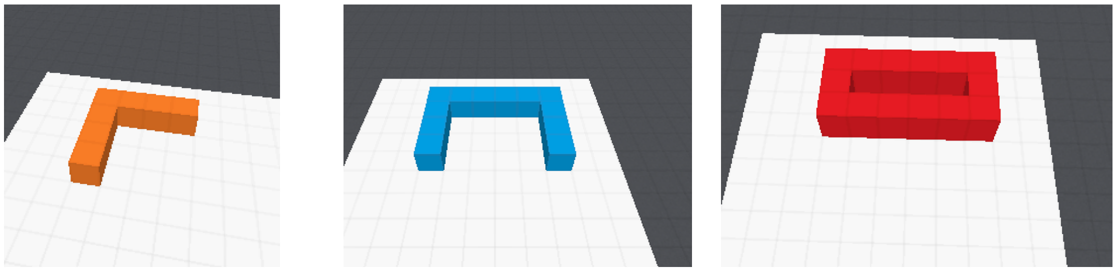
```
L(X) :- tower(A) ∧ color(A,C) ∧ height(A,H) ∧ row(B) ∧
    color(B,C) ∧ width(B,W) ∧ block(R) ∧ block(S) ∧
    bottom_end(A,R) ∧ left_end(B,S) ∧ spatial_rel(west,R,S) ∧
    color(X,C) ∧ height(X,H) ∧ width(X,V) ∧ one_more(V, W).
```

Note that the uppercase indicates variables, the only constant in this horn clause is `west`. Clausal representations of new structures are stored in the repository. On encountering such non-primitive structures in the instruction, the logical parser would first replace the non-primitive structure with the primitive structures from the clausal theory and then generate a goal description.

The most important feature of the GOCI framework is that it is capable of learning concept hierarchies. For example, once an L structure is learned as a combination of a tower and a row, a U structure can be learned as a combination of L and a tower. We demonstrate the concept hierarchy in Section 4. If needed, the architect can also ask the system to forget a learned concept. This can be done by the following command in the chat interface, "Forget <structure-name>".

4. Demonstration

Demonstration videos of LARA are available from the following URL: `https://bit.ly/3Rya01D`. Here we present the capability of LARA to learn concept hierarchies using the GOCI framework.



(a) Gamma structure (Γ) (b) Cap structure ($\sqcap$) (c) Box structure ($\square$)

Build an orange row of size 4.
Place a orange column of size 3.
The first block of the column is in front of the left end of the row.
done

This is an orange Gamma of length 4 and width 4.

(d) Instructions for Gamma

Build a blue gamma of size 4 by 6.
Place a blue column of size 3 in front of gamma.
The backend of the column is in front of the left end of the gamma.
done

This is a Cap of length 4 and width 6 in blue color.

(e) Instructions for Cap

Build a red cap of size 2 by 6.
Place a red row of size 6 in front of the cap.
The rightend of the row is in front of the right end of the cap.
done

This is a box of length 4 and width 6 in blue color.

(f) Instructions for Square

Figure 7: Demonstration of the concept hierarchy. (a) Gamma structure (Γ) made from a row and column , (b) Cap structure ($\sqcap$) made from a gamma and a column and, (c) Box structure ($\square$) made from a cap and a row. Subfigures (d), (e), and (f) presents the natural language instructions.

Figure 7 presents three different concepts, at three different levels of the hierarchy. The first structure Gamma (Figure 7a) is built as a composition of a row and a column. The natural language instructions for this structure are shown in Figure 7d. The next structure, Cap (Figure 7b) is composed of a gamma and a column. This forms the second level of hierarchy. Further, Figure 7c shows third level of hierarchy where the concept Box is composed of a cap and a row.

5. Discussion

We have considered the problem of human-machine collabaration in the context of a Minecraft task. Our proposed system LARA, is an integration of several important areas of related research. Specifically, it uses NLP for communication with the human, knowledge representation for representing and reasoning with generalized knowledge, inductive logic programming for efficiently learning higher-order concepts, and hierarchical task planning for effective problem-solving. LARA has the following salient features:

- It allows for *rich natural language instructions* from a human architect by employing an NLP-based parser.
- It represents common knowledge in a rich *FOL based knowledge-base that allows for effective generalization while not sacrificing efficient reasoning*.
- It uses an *ILP based concept learner that efficiently learns hierarchical, generalized concepts* from a very small number of (potentially single) examples.
- It uses a *hierarchical planner that constructs plans in a stage-wise* manner to effectively exploit the concepts learned in earlier interactions.
- It computes its *uncertainty over its plans/concepts* and queries the human expert for additional knowledge.

In effect, the system knows-what-it-knows and solicits information about what it does not know. This additional information is then used to guide the concept learner and the planner for their tasks.

While successful, the system can potentially be improved in several directions. The richness of natural language interaction can be improved by adding more recent neural-based parsers and generators. Allowing for multiple modalities of communication including gestures is an important future direction. Extending the planner and concept learner to handle complex hybrid data by allowing for them to be differentiable is a necessary step. Finally, large-scale evaluation in more complex domains is an interesting future direction.

Acknowledgements

We gratefully acknowledge the support of CwC Program Contract W911NF-15-1-0461 with US Defense Advanced Research Projects Agency (DARPA) and Army Research Office (ARO). We do not reflect views of the DARPA, ARO or the US government.

References

Bahdanau, D., Cho, K., & Bengio, Y. (2015). Neural machine translation by jointly learning to align and translate. *ICLR*.

Banarescu, L., et al. (2013). Abstract meaning representation for sembanking. *Proceedings of the 7th Linguistic Annotation Workshop and Interoperability with Discourse, LAW-ID@ACL*.

Bercher, P., Alford, R., & Höller, D. (2019). A survey on hierarchical planning - one abstract idea, many concrete realizations. *IJCAI* (pp. 6267–6275). ijcai.org.

Blockeel, H., & Raedt, L. D. (1998). Top-down induction of first-order logical decision trees. *Artif. Intell.*, *101*, 285–297.

Bonn, J., Palmer, M., Cai, Z., & Wright-Bettner, K. (2020). Spatial AMR: Expanded spatial annotation in the context of a grounded Minecraft corpus. *LREC*.

Bos, J. (2016). Expressive Power of Abstract Meaning Representations. *Computational Linguistics*, *42*, 527–535.

Brachman, R. J., & Levesque, H. J. (2004). *Knowledge representation and reasoning*. Elsevier.

Cropper, A., & Dumancic, S. (2022). Inductive logic programming at 30: A new introduction. *J. Artif. Intell. Res.*, *74*, 765–850.

Das, M., Odom, P., Islam, M. R., Doppa, J. R., Roth, D., & Natarajan, S. (2018). Preference-guided planning: An active elicitation approach. *AAMAS* (pp. 1921–1923).

Das, M., Ramanan, N., Doppa, J. R., & Natarajan, S. (2020). Few-shot induction of generalized logical concepts via human guidance. *Frontiers in Robotics and AI*, *7*, 122.

Erol, K., Hendler, J. A., & Nau, D. S. (1994). HTN planning: Complexity and expressivity. *AAAI* (pp. 1123–1128). AAAI Press / The MIT Press.

Fox, M., & Long, D. (2003). PDDL2.1: an extension to PDDL for expressing temporal planning domains. *J. Artif. Intell. Res.*, *20*, 61–124.

Fu, L., & Buchanan, B. G. (1985). Learning intermediate concepts in constructing a hierarchical knowledge base. *IJCAI* (pp. 659–666). Citeseer.

Gatt, A., & Krahmer, E. (2018). Survey of the state of the art in natural language generation: Core tasks, applications and evaluation. *J. Artif. Intell. Res.*, *61*, 65–170.

Höller, D., Behnke, G., Bercher, P., Biundo, S., Fiorino, H., Pellier, D., & Alford, R. (2020). HDDL: an extension to PDDL for expressing hierarchical planning problems. *AAAI* (pp. 9883–9891).

Ilghami, O. (2006). Documentation for jshop2. *Tech Report*.

Jackson, E., Appelt, D. E., Bear, J., Moore, R. C., & Podlozny, A. (1991). A template matcher for robust NL interpretation. *HLT*. Morgan Kaufmann.

Jayannavar, P., Narayan-Chen, A., & Hockenmaier, J. (2020). Learning to execute instructions in a minecraft dialogue. *ACL* (pp. 2589–2602).

Johnson, M., Hofmann, K., Hutton, T., & Bignell, D. (2016). The malmo platform for ai experimentation. *IJCAI*.

Kanervisto, A., et al. (2021). Minerl diamond 2021 competition: Overview, results, and lessons learned. *NeurIPS (Competition and Demos)* (pp. 13–28). PMLR.

Köhn, A., Wichlacz, J., Schäfer, C., Torralba, Á., Hoffmann, J., & Koller, A. (2020). MC-saar-instruct: a platform for Minecraft instruction giving agents. *SIGDIAL* (pp. 53–56).

Kokel, H., Odom, P., Yang, S., & Natarajan, S. (2020). A unified framework for knowledge intensive gradient boosting: Leveraging human experts for noisy sparse domains. *AAAI*.

Kokel, H., et al. (2021). Human-guided collaborative problem solving: A natural language based framework. *ICAPS (Demo Track)*.

Konstas, I., Iyer, S., Yatskar, M., Choi, Y., & Zettlemoyer, L. (2017). Neural AMR: sequence-to-sequence models for parsing and generation. *ACL (1)* (pp. 146–157).

Muggleton, S., & Feng, C. (1990). Efficient induction of logic programs. *New Generation Computing*. Academic Press.

Muggleton, S. H. (1995). Inverse entailment and progol. *New Gener. Comput.*, *13*, 245–286.

Muggleton, S. H., & Raedt, L. D. (1994). Inductive logic programming: Theory and methods. *J. Log. Program.*, *19/20*, 629–679.

Narayan-Chen, A., Jayannavar, P., & Hockenmaier, J. (2019). Collaborative dialogue in minecraft. *ACL*.

Narayan-Chen, A. Y. (2020). *Towards collaborative dialogue in minecraft*. Doctoral dissertation, University of Illinois at Urbana-Champaign.

Nau, D., Cao, Y., Lotem, A., & Munoz-Avila, H. (1999). Shop: Simple hierarchical ordered planner. *IJCAI* (pp. 968–975).

Nau, D. S., Au, T., Ilghami, O., Kuter, U., Murdock, J. W., Wu, D., & Yaman, F. (2003). SHOP2: an HTN planning system. *J. Artif. Intell. Res.*, *20*, 379–404.

Pazzani, M. J., Brunk, C., & Silverstein, G. (1991). A knowledge-intensive approach to learning relational concepts. *ML* (pp. 432–436). Morgan Kaufmann.

Peng, X., Wang, C., Gildea, D., & Xue, N. (2017). Addressing the data sparsity issue in neural AMR parsing. *EACL (1)* (pp. 366–375).

Quinlan, J. R. (1990). Learning logical definitions from relations. *Mach. Learn.*, *5*, 239–266.

Raedt, L. D. (1997). Logical settings for concept-learning. *Artif. Intell.*, *95*, 187–201.

Raedt, L. D., & Kersting, K. (2008). Probabilistic inductive logic programming. In *Probabilistic inductive logic programming*, volume 4911 of *Lecture Notes in Computer Science*, 1–27.

Raedt, L. D., Kersting, K., Natarajan, S., & Poole, D. (2016). Statistical relational artificial intelligence: Logic, probability, and computation. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, *10*, 1–189.

Salge, C., Green, M. C., Canaan, R., Skwarski, F., Fritsch, R., Brightmoore, A., Ye, S., Cao, C., & Togelius, J. (2020). The AI settlement generation challenge in minecraft. *Künstliche Intell.*, *34*.

Shah, R., et al. (2021). Retrospective on the 2021 minerl BASALT competition on learning from human feedback. *NeurIPS (Competition and Demos)* (pp. 259–272). PMLR.

Srinivasan, A. (1999). The aleph manual. From `https://www.cs.ox.ac.uk/activities/programinduction/Aleph/aleph`.

Sutskever, I., Vinyals, O., & Le, Q. V. (2014). Sequence to sequence learning with neural networks. *NIPS* (pp. 3104–3112).

Towell, G. G., & Shavlik, J. W. (1994). Knowledge-based artificial neural networks. *Artificial intelligence*.

Wichlacz, J., Torralba, A., & Hoffmann, J. (2019). Construction-planning models in minecraft. *ICAPS workshop on HPlan*.

Xu, D., Li, J., Zhu, M., Zhang, M., & Zhou, G. (2020). Improving AMR parsing with sequence-to-sequence pre-training. *EMNLP (1)* (pp. 2501–2511). Association for Computational Linguistics.

Zhang, S., Ma, X., Duh, K., & Durme, B. V. (2019). AMR parsing as sequence-to-graph transduction. *ACL (1)* (pp. 80–94).