Playing by the Book: An Interactive Game Approach for Action Graph Extraction from Text

Ronen Tamari*
The Hebrew University of Jerusalem
ronent@cs.huji.ac.il

Hiroyuki Shindo
NAIST / RIKEN-AIP
shindo@is.naist.jp

Dafna Shahaf
The Hebrew University of Jerusalem
dshahaf@cs.huji.ac.il

Yuji Matsumoto
NAIST / RIKEN-AIP
matsu@is.naist.jp

Abstract

Understanding procedural text requires tracking entities, actions and effects as the narrative unfolds. We focus on the challenging real-world problem of action-graph extraction from *materials science* papers, where language is highly specialized and data annotation is expensive and scarce. We propose a novel approach, TEXT2QUEST, where procedural text is interpreted as instructions for an *interactive game*. A learning agent completes the game by executing the procedure correctly in a text-based simulated lab environment. The framework can complement existing approaches and enables richer forms of learning compared to static texts. We discuss potential limitations and advantages of the approach, and release a prototype proof-of-concept, hoping to encourage research in this direction.

1 Introduction

Materials science literature includes a vast amount of synthesis procedures described in natural language. The ability to automatically parse these texts into a structured form could allow for data-driven synthesis planning, a key enabler in the design and discovery of novel materials (Kim et al., 2018; Mysore et al., 2017). A particularly useful parsing is **action graph extraction**, which maps a passage describing a procedure to a symbolic action-graph representation of the core entities, operations and their accompanying arguments, as they unfold throughout the text (Fig. 1).

Procedural text understanding is a highly challenging task for today's learning algorithms (Lucy and Gauthier, 2017; Levy et al., 2017). Synthesis procedures are especially challenging, as they are written in difficult and highly technical language assuming prior knowledge. Some texts are long, many follow a non-linear narrative, or include logical quantifiers ("all synthesis steps were performed in an argon atmosphere..."). Furthermore, annotated data is scarce and expensive to obtain.

Two related research areas are **grounded semantic parsing** and **state-tracking reading-comprehension**. Grounded (or executable) semantic parsers map natural language to a symbolic representation which can also be thought of as a sequence of instructions in some pre-defined programming language. Such "neural-programing" architectures offer strong symbolic reasoning capabilities, compositionality modelling, and strong generalization (Reed and de Freitas, 2015), but are typically applied to simple texts due to prohibitive annotation costs (Liang et al., 2016). **State-tracking** models (Bosselut et al., 2018; Das et al., 2018; Bansal et al., 2017) can model complex relations between entities as they unfold, with easier training but less symbolic reasoning abilities. Their applicability to longer texts is hindered as well by the lack of fine-grained annotated data.

In this work we describe an approach, TEXT2QUEST, that attempts to combine the strengths of both methods. Instead of trying to learn from static text, we propose to treat procedural text as **instructions for an interactive game** (or "quest"). The learning agent interacts with entities defined in the text by executing symbolic actions (Fig. 2). A text-based symbolic interpreter handles execution and tracking of the agent's state and actions. The game is completed by "simulating" the instructions correctly; i.e., mapping instructions to a sequence of actions. Correct simulation thus directly yields the desired action graph.

While there is some engineering overhead required for the simulator, we demonstrate that it is relatively straightforward to convert an annotation schema to a text-based game. We believe that the benefits make it worth pursuing: the game for-

*Work was begun while author was an intern at RIKEN and continued at the Hebrew University.

Proceedings of the Workshop on Extracting Structured Knowledge from Scientific Publications, pages 62–71
Minneapolis, USA, June 6, 2019.

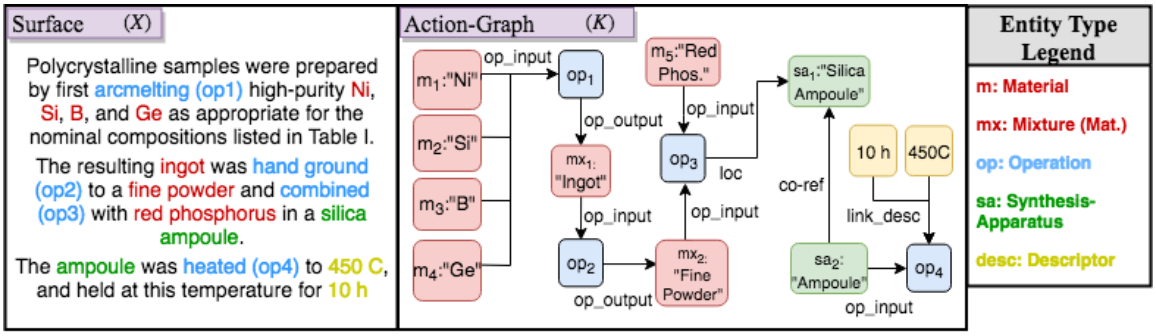


Figure 1: Sample surface text (left) and possible corresponding action-graph (right) for typical partial material synthesis procedure. Operation numbers in parentheses are added for clarity. Nodes are entities, edges are relations linking them, equivalent to actions in the text-based game.

mat allows applying powerful neural programming methods, with a significantly richer training environment, including advances such as curriculum learning, common-sense and domain-specific constraints, and full state tracking. Such "friendly" environments that assist the learning agent have been shown to be valuable (Liang et al., 2016) and enable learning of patterns that are often hard to learn from surface annotations alone, such as implicit effects of operations (i.e., filtering a mixture splits it into two entities).

Interestingly, understanding by simulation aligns well with models of human cognition; mental simulation, the ability to construct and manipulate an internal world model, is a cornerstone of human intelligence involved in many unique behaviors, including language comprehension (Marblestone et al., 2016; Hamrick, 2019). In this work we take first steps towards this idea. Our contributions are:

- We propose a **novel formulation** of the problem of procedural text understanding as a text-based game, enabling the use of neural programming and text-based reinforcement learning (RL) methods.
- We present and release TEXTLABS[1], an instance of TEXT2QUEST designed for interaction with synthesis procedure texts. We focus on the material-science setting, but the approach is intended to be more generally applicable.
- We propose to address the problem of obtaining full-graph annotations at scale by coupling the simulator with **controllable natural language generation** (NLG) to generate synthetic data, also enabling curriculum learning.

[1]Code and experiments available at https://github.com/ronentk/TextLabs

```
The starting materials are nickel and
zinc. First, grind the materials. After
that, mix, heat and compact them.
You are in the Lab.
Available operations are: a grind_op, a
heat_op and a mix_op.
Available materials are: nickel and zinc.
> input_assign nickel to grind_op
Input assigned!
> input_assign zinc to grind_op
Input assigned!
> run grind_op
Ground materials!
> Examine grind_op's output
The grind_op's output is a mixture. It
contains nickel and zinc. It is powder.
```

Figure 2: Excerpt from an actual "material synthesis quest" generated by our system with example input/outputs.

While this work is preliminary in nature, neural programming and text-based reinforcement learning approaches are attracting significant and growing interest, and we expect advances in these areas to directly benefit future versions of the system.

2 Related Work

Procedure understanding: Many recent works have focused on tracking entities and relations in long texts, such as cooking recipes and scientific processes (Bosselut et al., 2018; Das et al., 2018). However, these methods do not directly extract a full action graph. For action graph extraction, earlier works use sequence tagging methods (Mysore et al., 2017). Feng et al. (2018) have applied deep-RL to the problem of extracting action sequences, but assume explicit procedural instruction texts. In Johnson (2017), a graph is constructed from simple generated stories, using state tracking at each time step as supervision.

Semantic parsing & Neural Programming: Research to-date has focused mainly on shorter

and simpler texts which may require complex symbolic reasoning, such as mapping natural language to queries over knowledge graphs (Liang et al., 2016). In the case of narrative parsing, the text itself may be complex while the programs are relatively simple (creating and linking between entities present in the text). Recent work (Lu et al., 2018) frames narrative understanding as neural-programming, the learner converts a document into a structured form, using a predefined set of data-structures. This approach is similar to ours, though with simpler texts and without a simulated environment. In our approach, the learning architecture is decoupled from the symbolic interpreter environment, enabling greater architectural flexibility.

Text-RL: Text-based games are used to study language grounding and understanding and RL for combinatorical action spaces (Zahavy et al., 2018; Narasimhan, 2017) but have not yet been applied to real world problems. TextWorld (Côté et al., 2018) is a recently released reinforcement learning sandbox environment for creation of custom text-based games, upon which we base TEXTLABS.

3 Problem Formulation

Entities, Relations & Rules ($\mathcal{E}, \mathcal{R}, \Lambda$): Assume two vocabularies defining types of *entities* $\mathcal{E} = \{e_1, ..., e_N\}$ and *relations* $\mathcal{R} = \{r_1, ..., r_K\}$. A *fact* f is a grounded predicate of the form $f = r\,(h, t)\,,\ h, t \in \mathcal{E},\ r \in \mathcal{R}$ (single or double argument predicate relations are allowed). We define the set of valid world-states S, where a state $s \in S$ is a set of facts, and validity is decided by a world-model Λ defined using linear logic. Λ is comprised of production rules (or transition rules) over entities and relations governing which new facts can be produced from a given state. Following the schema used in the Synthesis Project[2] (see for example MSP), entity types include materials, operations, and relevant descriptors (like operation conditions, etc.). Relations link between entities (like *input(material,operation)* or denote single predicate relations (entity properties such as *solid(material)*). We currently use a simplified version of the schema to ease the learning problem. See appendix A.1 for a mapping of relations and entities. Production rules correspond to the actions available to the learner, in our domain these include for example *link-descriptor(descriptor,entity)*, *input-assign(material, operation)*. While not currently included, actions such as co-reference linking and generation of entities can also be incorporated.

Action-Graph (K): An action sequence is defined to be a sequence of valid actions (or production rules) rooted at some initial state s_0: $K = (s_0, \lambda_0, \lambda_1, ..., \lambda_n)$ (applying λ_i to s_i results in s_{i+1}, intermediate states are left out for brevity). Note that actions may apply to implicit entities not present in the surface text (for example, the result of an operation). Construction of an action graph corresponding to K is straightforward (entities as nodes, actions connecting them as edges), and henceforth we use K to denote either the sequence or the graph. Note that there can be multiple possible action sequences resulting in the same action graph, equivalent w.r.t the topological ordering of operations induced by their dependencies.

Surface (X): A *surface* is simply a text in natural language describing a process.

Learning Task: Our objective is to learn a mapping $\Psi : X \rightarrow K$. As this mapping may be highly complex, we convert the problem to a structured prediction setting. As an intermediate step **we map an input X to an enriched text-based-game G representation** (details below), where the solution of G is the required action graph K. The game is modelled as a partially observable Markov Decision Process (POMDP) $G = (S, A, T, \Omega, O, R, \gamma)$.

We refer the reader to Côté et al. (2018) for a detailed exposition, and focus here on mapping the game-setting to our approach: S are states, A are actions, T are conditional state transition probabilities, where all are constant per domain and defined by $\mathcal{E}, \mathcal{R}, \Lambda$. Ω are observations, and O are conditional observations probabilities. $R : S \times A \rightarrow \mathbb{R}$ is the reward function, $\gamma \in [0, 1]$ is the discount factor. As γ, Ω, O are also preset (with actual observations dependent on agent actions), mapping a surface X to game G boils down to providing a list of entities for initializing s_0. For training and evaluation, a reward function must also be provided (not necessary for applying a trained model on un-annotated text "in the wild").

If a fully annotated action graph is available (whether synthetic or real), this mapping is simple: the initial game state s_0 is a room where the agent is placed alongside all entities. Each edge corresponds to an action in the game. Given an action sequence K, a reward function R can be automatically computed, giving intermediate rewards and penalizing wrong actions. A quest in TextWorld can be defined via a final goal state, thus allowing

[2] https://www.synthesisproject.org/

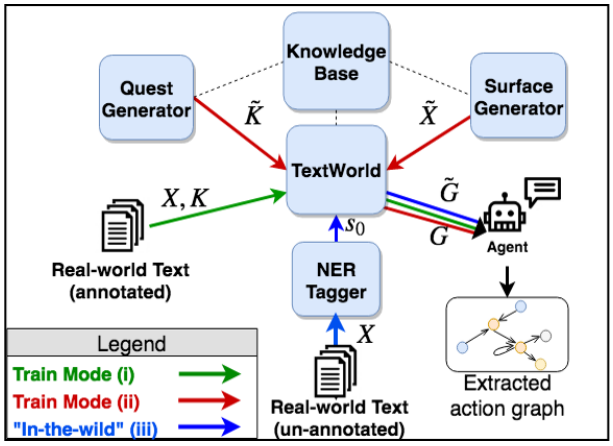


Figure 3: Proposed solution architecture of TEXT2QUEST. (i) Flow for training agent on games from real annotated data. (ii) Flow for training agent on synthetic games. (iii) Extracting action graph from un-annotated real data.

multiple possible winning action sequences. See appendices A.2, A.3 for examples.

For data "in the wild", entities can be identified using named entity recognition (NER) as preprocessing. Future directions include end-to-end learning to reduce cascading initialization errors.

By default, the TextWorld environment is partially observable. The agent observes the surface X at time $t = 0$ and other textual descriptions upon executing an "examine" action. Unlike classic text-based games where partial observability is part of the challenge, in our case we can adopt the "friendly-environment" perspective and assist the learner with information such as state-tracking or action pruning (Liang et al., 2016; Johnson, 2017).

4 Proposed Solution Architecture

Our system consists of 6 core modules (Fig. 3): a Knowledge Base defines entity, relation and action vocabularies. This is used by the Surface Generator and Quest Generator modules to generate pairs $(\tilde{X}, \tilde{K})$ of synthetic surfaces and their corresponding action graphs for training. For un-annotated text, a pre-trained domain specific NER tagger[3] is used to extract an initial game state s_0 by identifying the mentioned entities. A learning agent extracts K from a generated game.

The TEXT2QUEST architecture supports three central modes of operation: (i) Enrich existing real world annotated pairs (X, K) by converting them to game instances for training the game-solving agent. (ii) Produce synthetic training pairs $(\tilde{X}, \tilde{K})$. (iii) Convert un-annotated texts to game instances for action graph extraction "in the wild".

The current version of TEXTLABS supports mode (ii). We implemented simple prototypes of the domain-specific Knowledge Base, plus Quest and Surface Generators. See Sec. A.1 for details about converting the entity and relation annotation schema into TextWorld. TextWorld is easily extensible and can support a variety of interaction semantics. Aside from adding a domain specific entity type-tree and actions, most of the underlying logic engine and interface is handled automatically. For the game environment, we use Inform7, a programming language and interpreter for text-based games. For quest generation, we currently use simple forward chaining and heuristic search strategies to create plausible quests (for example, all start materials must be incorporated into the synthesis route). Combining these with a simple rule-based Surface Generator already allows for creating simple synthetic training game instances (Fig. 2).

5 Preliminary Evaluation

As a very preliminary sanity check for the TEXTLABS environment, we train a simple text-based RL agent on synthetic games in increasingly difficult environments. Difficulty is measured by maximum quest length, and the number of entities in the target action graph. See Sec. A.2 for representative examples. We use the basic LSTM-DQN agent of Narasimhan (2017) adapted to the TEXTLABS setting. The action space is $A = \{W_v \times W_{o_1} \times W_{o_2}\}$, where W_v consists of 8 action-verbs corresponding to the entity relations tracked and additional native TextWorld actions like *take* (see Sec. A.1 for details). W_{o_1}, W_{o_2} are (identical) sets of potential arguments corresponding to the active entities which can be interacted with in the game (single and double argument actions allowed). As this basic agent is not conditioned on previous actions, we further concatenate the last four commands taken to the current observation. For the same reason, we also append the full quest instructions at every timestep's observation. All illegal actions are pruned at each state to reduce search space size.

We train the agent on 100 games per level and test on 10 games. Evaluation is measured by avg. normalized reward per game: $\frac{1}{|K|}\sum_{t=1}^{T} r_i$, where K is the true action sequence, T is the episode

[3]For the materials synthesis domain we use the tagger available at https://github.com/olivettigroup/materials-synthesis-generative-models

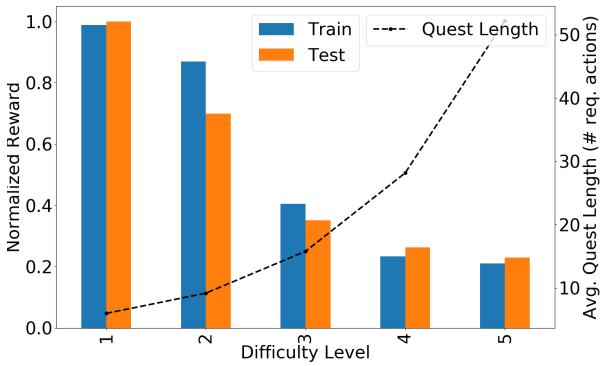


Figure 4: Preliminary evaluation results for a basic LSTM-DQN text-RL agent on synthetic quests. Dotted line shows average generated quest lengths.

length (set to 50) and $r_i = 1$ for each action in K and -1 for otherwise (and 0 for neutral actions like *examine*). A normalized score of 1 means the agent performed the required actions exactly.

As can be seen in Fig. 4, the agent learns to successfully perform the required actions only for the easiest levels. Examining longer games the agent did not complete, we note that the lack of conditioning on previous states is a serious limitation. Equipping agents with better sequence encoding (e.g., attention), recurrent memory, and utilizing state information is expected to significantly improve performance. Furthermore, due to technical limitations of the current implementation, some actions cannot be reversed. This adds to the difficulty of the task, and will be addressed in future versions. Finally, learning good initial policies for semantic parsers is known to be a hard problem with RL alone, and related approaches commonly use hybrid RL/supervised training methods (Liang et al., 2016; Jiang et al., 2012).

6 Discussion

Our approach faces tough challenges. However, we are encouraged by the significant recent advances towards these challenges in related areas, and plan to leverage this progress for our framework.

Programming semantics and rewards for instruction-following agents is known to be notoriously difficult (Winograd, 1972) as language and environments grow increasingly complex. Research on **learned instruction-conditional reward models** (Bahdanau et al., 2018) is a promising approach towards reducing the amount of "environment engineering" required.

Another critical open question in our framework is whether the surface generator will be able to generate surfaces representative enough to allow for generalization to real examples. Current NLG systems are increasingly capable of structured text generation (Marcheggiani and Perez, 2018), and though they produce relatively short surfaces, we believe that coupling them with the generated action graphs is a promising approach to scaling up to longer sequences while maintaining coherence. Such systems can use sentence-level semantic parses as training data, meaning they can leverage existing weakly-supervised shallow parsing techniques. Encouraging for our modelling paradigm, recent work (Peng et al., 2018) extending the Dyna-Q (DQ) framework (Sutton, 1990) demonstrates a real-world application of structured NLG with a simulated RL training environment.

Given sufficient text generation capabilities, one may question the added utility of the game environment (as opposed to learning a direct mapping $X \rightarrow K$). Recent research suggests that for stronger generalization, data alone may not be enough, and symbolic reasoning capabilities are necessary (Khashabi et al., 2018; Yi et al., 2018). Given the compositional complexity and difficulty of the language involved, we believe they will prove necessary in our setting as well.

7 Conclusions

There is a growing need for combining neuro-symbolic reasoning with advanced language representation methods. In the case of procedural text understanding, key obstacles are suitable training environments, as well as the lack of fully annotated action graphs. Motivated by this, we proposed TEXT2QUEST, an approach intended to enhance learning by turning raw text inputs into a structured *text-based game* environment, as well as augmenting data with synthetic fully annotated action graphs. To encourage further research in this direction, we publicly release TEXTLABS, an instance of TEXT2QUEST for the materials synthesis task. We implemented prototype modules for basic game generation and solving. Future work will focus on designing learning agents to solve the games, as well as improving text generation capabilities. We hope that the proposed approach will lead to developing useful systems for action graph extraction as well as other language understanding tasks.

References

The materials science procedural text corpus. https://github.com/olivettigroup/annotated-materials-syntheses. Accessed: 5/4/2019.

Dzmitry Bahdanau, Felix Hill, Jan Leike, Edward Hughes, Pushmeet Kohli, and Edward Grefenstette. 2018. Learning to Understand Goal Specifications by Modelling Reward. pages 1–19.

Trapit Bansal, Arvind Neelakantan, and Andrew McCallum. 2017. RelNet: End-to-end Modeling of Entities & Relations. pages 1–6.

Antoine Bosselut, Omer Levy, Ari Holtzman, Corin Ennis, Dieter Fox, and Yejin Choi. 2018. Simulating action dynamics with neural process networks. *Proceedings of the 6th International Conference for Learning Representations*.

Marc-Alexandre Côté, Ákos Kádár, Xingdi Yuan, Ben Kybartas, Tavian Barnes, Emery Fine, James Moore, Matthew Hausknecht, Layla El Asri, Mahmoud Adada, Wendy Tay, and Adam Trischler. 2018. Textworld: A learning environment for text-based games. *CoRR*, abs/1806.11532.

Rajarshi Das, Tsendsuren Munkhdalai, Xingdi Yuan, Adam Trischler, and Andrew McCallum. 2018. Building Dynamic Knowledge Graphs from Text using Machine Reading Comprehension. pages 1–12.

Wenfeng Feng, Hankz Hankui Zhuo, and Subbarao Kambhampati. 2018. Extracting Action Sequences from Texts Based on Deep Reinforcement Learning.

Jessica B Hamrick. 2019. Analogues of mental simulation and imagination in deep learning. *Current Opinion in Behavioral Sciences*.

Jiarong Jiang, Adam R. Teichert, Hal Daumé, and Jason Eisner. 2012. Learned prioritization for trading off accuracy and speed. In *NIPS*.

Daniel D. Johnson. 2017. Learning graphical state transitions. In *ICLR 2017*.

Daniel Khashabi, Tushar Khot, Ashutosh Sabharwal, and Dan Roth. 2018. Question answering as global reasoning over semantic abstractions. In *AAAI*.

Edward Kim, Zach Jensen, Alexander van Grootel, Kevin Huang, Matthew Staib, Sheshera Mysore, Haw-Shiuan Chang, Emma Strubell, Andrew McCallum, Stefanie Jegelka, and Elsa Olivetti. 2018. Inorganic Materials Synthesis Planning with Literature-Trained Neural Networks.

Omer Levy, Minjoon Seo, Eunsol Choi, and Luke Zettlemoyer. 2017. Zero-Shot Relation Extraction via Reading Comprehension.

Chen Liang, Jonathan Berant, Quoc Le, Kenneth D. Forbus, and Ni Lao. 2016. Neural Symbolic Machines: Learning Semantic Parsers on Freebase with Weak Supervision (Short Version).

Zhengdong Lu, Haotian Cui, Xianggen Liu, Yukun Yan, and Daqi Zheng. 2018. Object-oriented neural programming (oonp) for document understanding. In *ACL*.

Li Lucy and Jon Gauthier. 2017. Are distributional representations ready for the real world? Evaluating word vectors for grounded perceptual meaning.

Adam H. Marblestone, Gregory Wayne, and Konrad P. Körding. 2016. Toward an integration of deep learning and neuroscience. In *Front. Comput. Neurosci.*

Diego Marcheggiani and Laura Perez. 2018. Deep Graph Convolutional Encoders for Structured Data to Text Generation.

Sheshera Mysore, Edward Kim, Emma Strubell, Ao Liu, Haw-Shiuan Chang, Srikrishna Kompella, Kevin Huang, Andrew McCallum, and Elsa Olivetti. 2017. Automatically Extracting Action Graphs from Materials Science Synthesis Procedures.

Karthik Narasimhan. 2017. *Grounding natural language with autonomous interaction*. Ph.D. thesis, Massachusetts Institute of Technology, Cambridge, USA.

Baolin Peng, Xiujun Li, Jianfeng Gao, Jingjing Liu, Kam-Fai Wong, and Shang-Yu Su. 2018. Deep Dyna-Q: Integrating Planning for Task-Completion Dialogue Policy Learning.

Scott Reed and Nando de Freitas. 2015. Neural Programmer-Interpreters. pages 1–13.

Richard S. Sutton. 1990. Integrated Architectures for Learning, Planning, and Reacting Based on Approximating Dynamic Programming. In *Machine Learning Proceedings 1990*.

Terry Winograd. 1972. Understanding natural language. *Cognitive Psychology*, 3(1):1 – 191.

Kexin Yi, Jiajun Wu, Chuang Gan, Antonio Torralba, Pushmeet Kohli, and Joshua B. Tenenbaum. 2018. Neural-Symbolic VQA: Disentangling Reasoning from Vision and Language Understanding. (NeurIPS).

Tom Zahavy, Matan Haroush, Nadav Merlis, Daniel Jaymin Mankowitz, and Shie Mannor. 2018. Learn what not to learn: Action elimination with deep reinforcement learning. In *NeurIPS*.

A Appendices

A.1 Entity & Relation Types

We have claimed that converting an annotation schema to a game for TEXTLABS was relatively straightforward. In this section, we provide details of the mapping between the Synthesis Project annotation schema of (denoted with "SP" in the tables) to the TEXTLABS implementation (denoted "TL"). A mapping between the central entity types is presented in Figure 5, as well as the TEXTLABS actions and representative corresponding relations in the schema. All current TEXTLABS entities and actions are shown here, though not all of the original entities and relations are listed. For the full mapping, refer to the project source repository.

A.2 Synthetic Action-Graphs

Figure 6 displays sample representative generated quests for the various difficulty levels evaluated in Sec. 5, demonstrating the controllable complexity. As can be seen by comparison with the real text in Fig. 7 (which is only one sentence), these graphs correspond to short real-world surfaces, where even the hardest could by covered by a 2-3 sentence-long procedure.

A.3 Action-Graphs from Real Annotated Graphs

We now provide further details on how the original Synthesis Project (SP) annotated graphs can be converted to a TEXTLABS action graph K. There are some minor differences between the formats, primarily in the handling of the SP "next-operation" relation. Rather than use a "next-operation" relation, we currently opt to explicitly model inputs/outputs to operations, as can be seen in Fig. 7. This is a natural abstraction away from the surface text enabled by the grounded environment, and helps in tracking which materials participated in each operation, which is useful information for later analysis. Also, as noted, we currently use a simplified mapping (for example, many descriptor annotations such "Amount-Unit", "Property-Misc", etc. are chunked together as generic descriptors). In Fig. 7 we show K both in action graph and action sequence form to demonstrate the equivalence. Also, we note that the "next-operation" annotations in MSP are currently just placeholders and not the true labels. For the purpose of demonstration, in Fig. 7 we manually add the correct annotation to our example (center and bottom).

Entity Type (SP)	**Entity Type (TL)**	**Notes**
Material	Material	
Number	Descriptor	
Operation	Operation	
Amount-Unit	Descriptor	
Condition-Unit	Operation-Descriptor	
Material-Descriptor	Material-Descriptor	
Condition-Misc	Operation-Descriptor	
Synthesis-Apparatus	Synthesis-Apparatus	
Nonrecipe-Material	Null	Currently ignored, not part of synthesis
Brand	Descriptor	
Apparatus-Descriptor	Synthesis-Apparatus-Descriptor	
-	Mixture	Internal entity, represents a mixture
Relation Type (SP)	**Action (TL)**	
Participant-Material	input-assign	
Apparatus-of	locate	
Recipe-Target	obtain	
Descriptor-of	link-descriptor	
-	run-op	Internal, used for simulating actions
-	take/drop/examine	Native TextWorld actions on entities

Figure 5: Central entity/relation types from the Synthesis Project schema ("SP"), and the corresponding TEXT-LABS version ("TL").

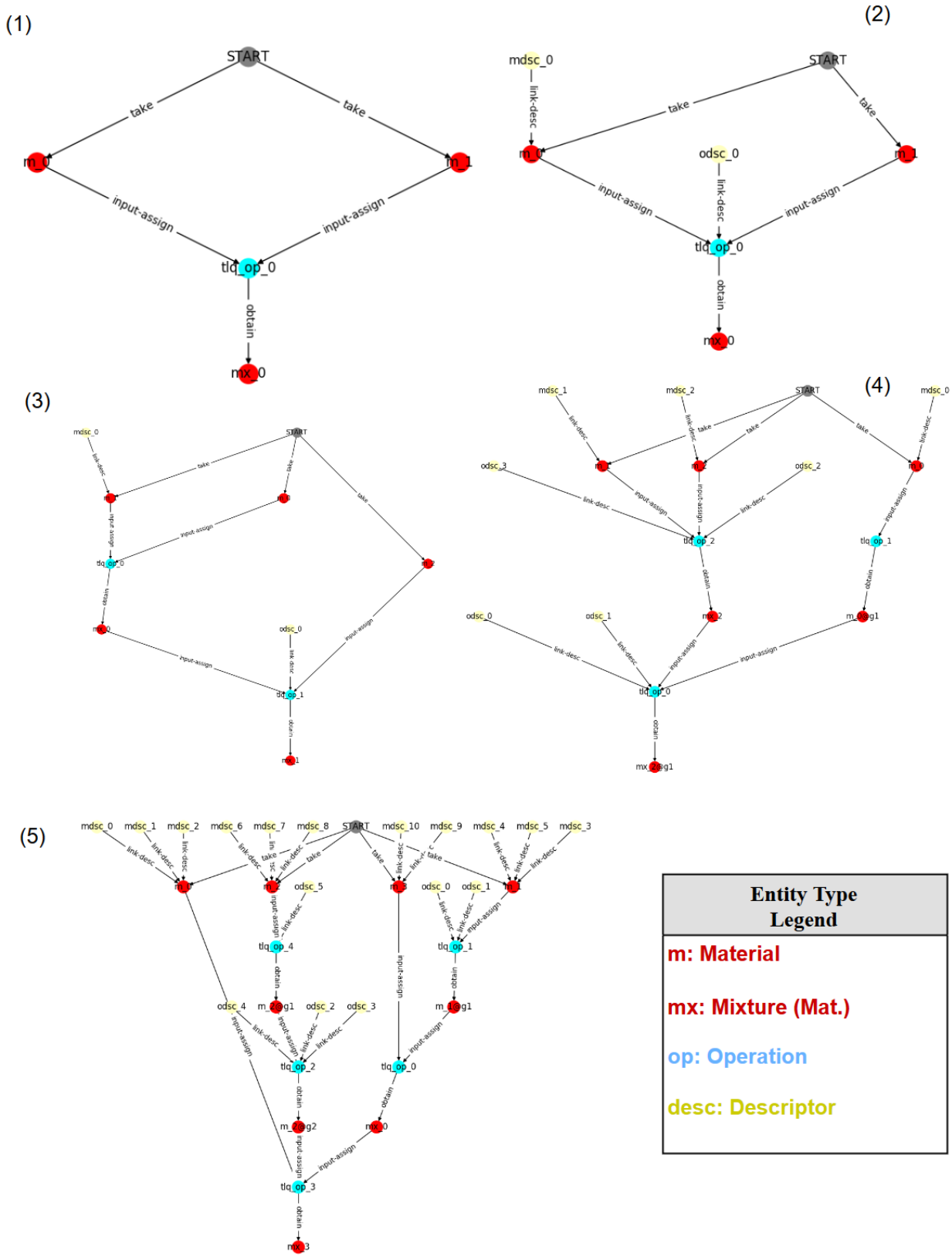
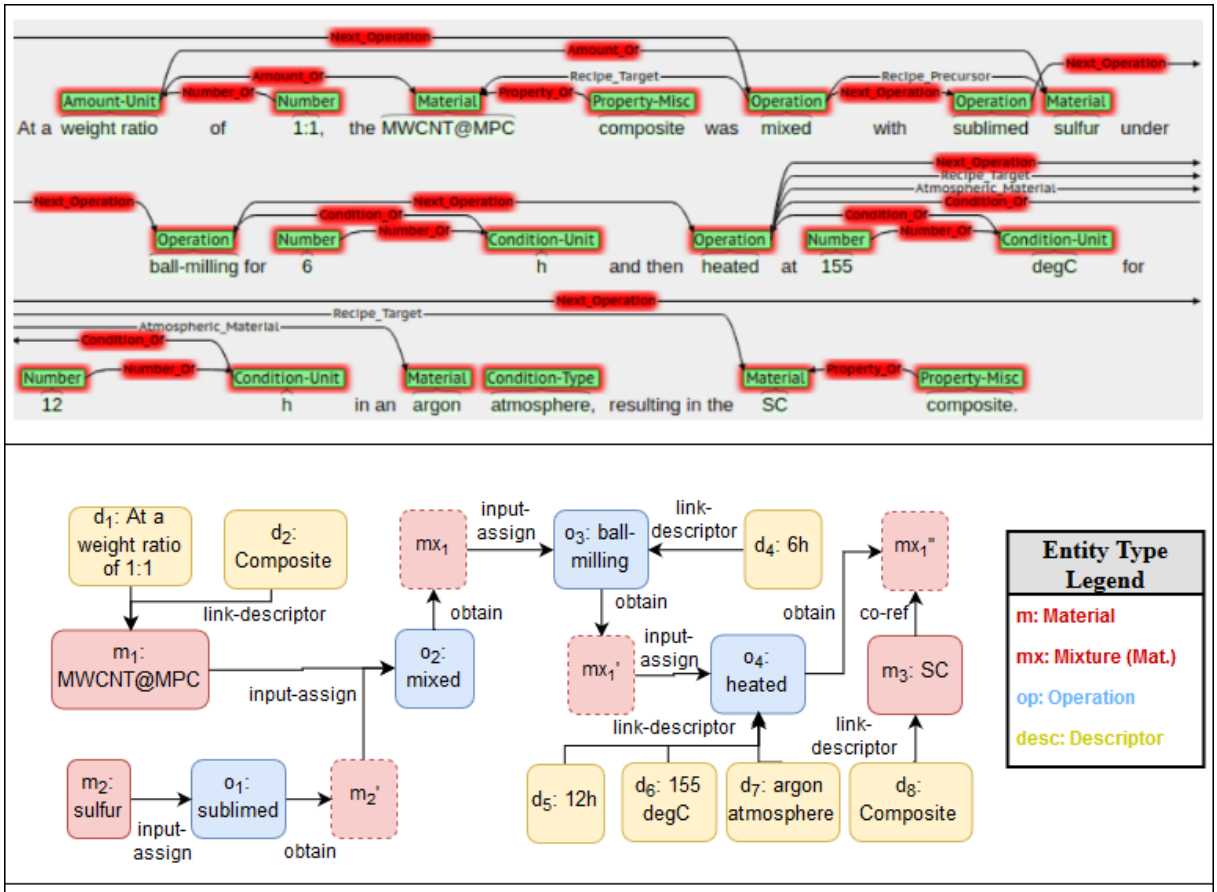


Figure 6: Sample representative generated quests for various difficulty levels (listed in parentheses by each graph). Each edge corresponds to an action in the text-based game.




```
s_0 = { at(m1, r), at(m2, r), ... }

1. link-descriptor(d1, m1)
2. link-descriptor(d2, m1)
3. input-assign(m1, o2)
4. input-assign(m2, o1)
5. run-op(o1)
6. obtain-output(o1)
7. input-assign(m2, o2)
8. run-op(o2)
9. obtain-output(o2)
10. input-assign(mx1, o3)
11. link-descriptor(d4, o3)
12. run-op(o3)
13. obtain-output(o3)
14. input-assign(mx1, o4)
15. link-descriptor(d5, o4)
16. link-descriptor(d6, o4)
17. link-descriptor(d7, o4)
18. run-op(o4)
19. obtain-output(o4)
20. co-ref(m3, mx1)
21. link-descriptor(d8, m3)
```

Figure 7: Comparisons of the equivalent action graph representations. **Top**: Action graph section from Synthesis Project (MSP). **Center**: TEXTLABS, showing same section with K in graph form. Dashed borders indicate operation result entities which may be implicit in the text. **Bottom**: TEXTLABS with same K as list of actions from initial state s_0.