EECS 595 Final Project Report

Ben VanDerPloeg

Abstract

As noted by the authors of Storks et al. (2021), the ubiquitous language model has dominated the field of NLP for the past few years, leveraging huge corpora to gain rich contextual understandings of natural language at the expense of interpretability. Their new physical commonsense dataset, Tiered Reasoning for Intuitive Physics (TRIP), presents tasks which require a model to not only give the correct answer, but to explain its reasoning through identifying relevant portions of text and physical state changes. In this spirit, we herein present a hyper-interpretable system [1] for the TRIP tiered reasoning tasks which explicitly tracks distributions of object states throughout a story to reach an explainable conclusion.

1 Introduction

Tiered Reasoning for Intuitive Physics (TRIP) is a new dataset published by Storks et al. (2021) which explores physical commonsense reasoning with an emphasis on explainability and the validity of models' underlying reasoning. Consisting of pairs of brief stories – one of which is plausible, and the other not – a model's primary task is to decide which story is plausible. However, the additional tasks proposed by the authors also challenge the model's ability to explain which sentences in the story conflict, as well as what *physical state changes* contributed to the conflict. Hence, the reasoning tasks are *tiered*, each requiring a deeper understanding of the story than the last.

In this report, we detail our implementation of a lightweight system for the TRIP tasks which attempts to take interpretability a step further by leveraging explicit distributions of object states over time to make inferences about plausibility. Each stage in the pipeline is almost entirely "transparent", and the sequence of intermediate inferences made on a story are readily understandable.

[1] https://github.com/lildutchie99/EECS595FinalProject

We begin by estimating a distribution of the physical state changes (preconditions and effects) undergone by each object in each sentence, where states consist of a handful of binary attributes such as "conscious" and "wet". We then iterate through the story, tracking the state of the objects and noting when a precondition is likely to have not been satisfied. From these conflicts, we can look back at previous modifications to object attributes to determine the source of the conflict as well as the physical state changes responsible, thereby solving the tiered tasks.

2 Related Work

As the authors noted, the tasks proposed in this paper are largely novel (as is the dataset) so there is no prior work on this exact problem. Some benchmarks and approaches for related physical common sense reasoning tasks are given in the original paper. As for our approach, there is much work in past decades on applying logic programming to NLP tasks, perhaps the most relevant to this approach being Lima et al. (2019). Similar to our approach for state change prediction, they model semantic relationships as dependency graph paths and infer transfer rules (or in our case distributions) from data. This work, however, is on the Relation Extraction (RE) task rather than sequential physical reasoning, so the pipeline ends with outputting predicate tuples, rather than reasoning about physical state changes as we would like to do here.

3 Approaches

The components of the tiered reasoning system are explicitly modeled as submodules in our implementation. The following present information on each submodule in detail.

3.1 State Change Prediction

The first task in the pipeline is a supervised learning problem: to predict the physical state changes

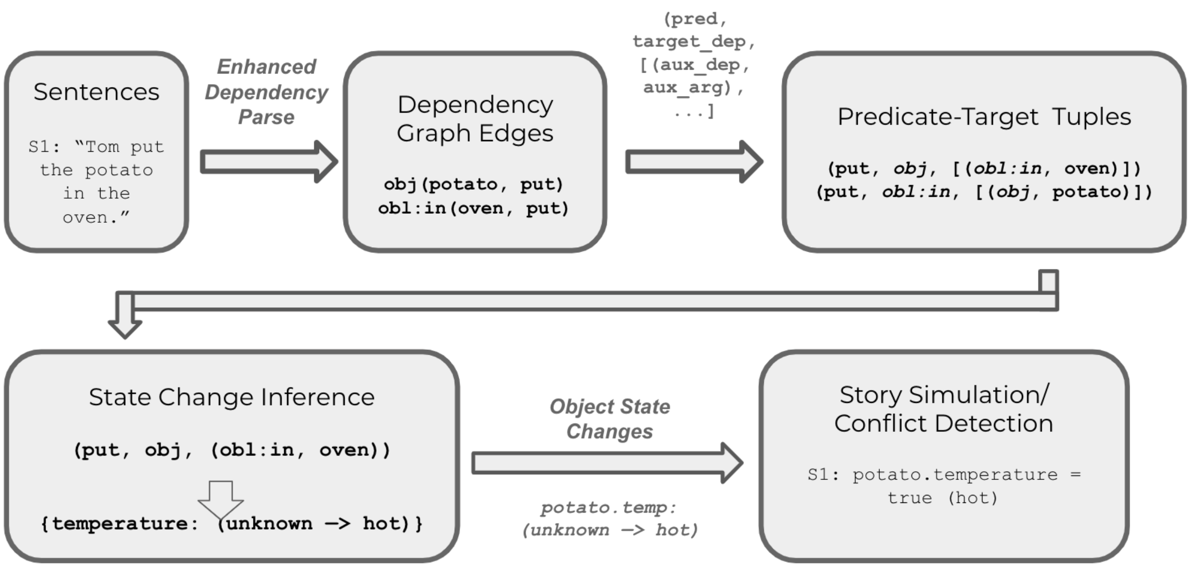
Sentences
S1: "Tom put the potato in the oven."
Enhanced Dependency Parse
Dependency Graph Edges
obj(potato, put)
obl:in(oven, put)
(pred, target_dep, [(aux_dep, aux_arg), ...]
Predicate-Target Tuples
(put, obj, [(obl:in, oven)])
(put, obl:in, [(obj, potato)])
State Change Inference
(put, obj, (obl:in, oven))
{temperature: (unknown -> hot)}
Object State Changes
potato.temp: (unknown -> hot)
Story Simulation/ Conflict Detection
S1: potato.temperature = true (hot)

Figure 1: Processing the sentence *Tom put the potato in the oven.*

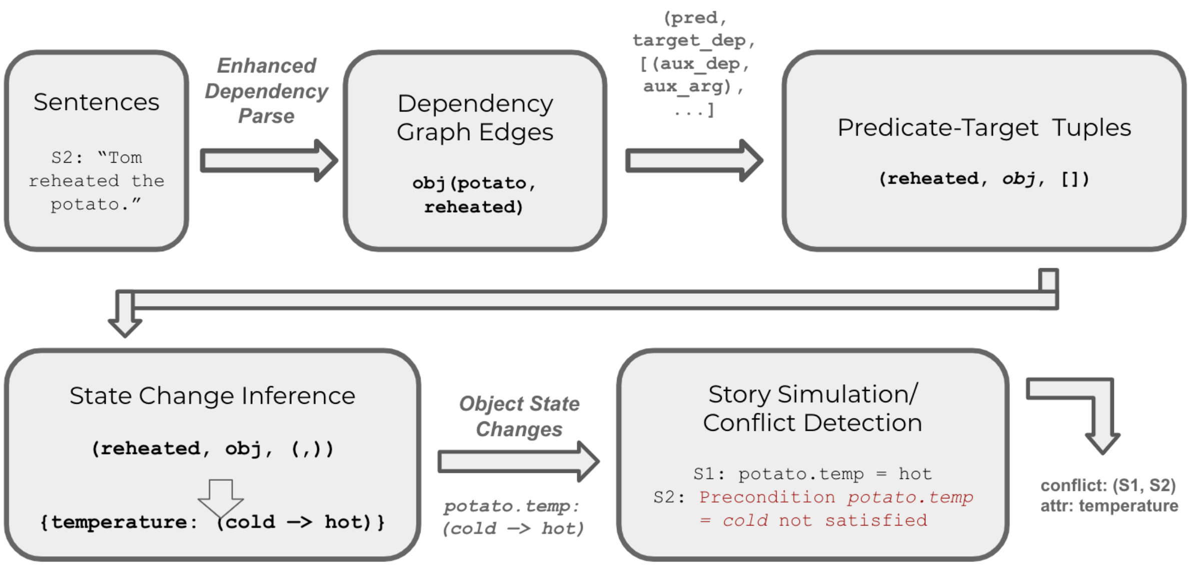
Sentences
S2: "Tom reheated the potato."
Enhanced Dependency Parse
Dependency Graph Edges
obj(potato, reheated)
(pred, target_dep, [(aux_dep, aux_arg), ...]
Predicate-Target Tuples
(reheated, obj, [])
State Change Inference
(reheated, obj, (,))
{temperature: (cold -> hot)}
Object State Changes
potato.temp: (cold -> hot)
Story Simulation/ Conflict Detection
S1: potato.temp = hot
S2: Precondition potato.temp = cold not satisfied
conflict: (S1, S2)
attr: temperature

Figure 2: Processing the subsequent sentence *Tom reheated the potato.*

(preconditions and effects) of each story object at each timestep, given the dataset annotations. For this task, we employ a dependency graph-based approach which maps predicate-entity pairs to distributions over physical state changes.

The obvious first step is to obtain a dependency graph. For this, we use CoreNLP's Enhanced++ dependency parser, which provides a graph augmentation of a standard dependency tree by applying transform rules to add extra edges. While these transforms don't add knowledge, they enrich the graph and make more shallow semantic relations readily observable. For example, in the sentence *Ann used the blender to make a drink out of cucumber and milk*, a standard dependency parse only relates the verb *make* and the noun *cucumber* by a single vague edge of type `obl`. However, in an Enhanced++ graph, there are additional edges of type `obl:out_of` from *make* to both *cucumber* and *milk*; these edges concisely distill the prepositional relation into an edge type, and also resolve the conjunctive noun phrase to infer that both "cucumber" and "milk" are prepositional objects. This makes far more semantic information available to the parsing system.

The system then proceeds by learning a set of "rule classes", each of which is a tuple consisting of a predicate (verb) and a typed dependency. A particular rule class represents the effect of the predicate on entities which are related to it via this dependency. For example, in the sentence *Tom put the potato in the oven*, the rule class (*put*, `obj`) encodes the effect of the predicate *put* on the entity *potato* (as there is a dependency graph edge of type `obj` from *put* to *potato*, i.e. *potato* is the direct object of *put*). Similarly, the rule class (*put*, `obl:in`) encodes the effect of the predicate *put* on the entity *oven* (`obl:in` is an enhanced dependency edge corresponding to a prepositional phrase with the preposition "in"). Note how the predicate's different effects on each of these entities is informed by the dependency relation: as the object which is being *put* somewhere, the potato undergoes effects related to the predicate and surrounding context (in this case, one such effect is that the potato becomes hot). Conversely, as the object into which something is being *put*, the microwave undergoes a different set of state changes (for example, it now contains something). While a rule class often does not contain enough information to reason about all physical state changes of an object (knowing that the *potato* was put somewhere is not enough to infer that it becomes hot), these rule classes provide a foundation on which further inferences about state changes can be made.

To encode this clearly relevant information about the sentence's context, we can define a set of *auxiliary dependencies* (ADs) for a particular instance of a rule class in a sentence, which encode the relation of other objects, phrases, and clauses to the predicate. Continuing the previous example, for the rule class `(put, obj)` instantiated on *potato*, the one AD present is (`obl:in`, *oven*). Notice that this information along with the rule class infers a lot more information about the state changes of the potato: it is the general case that if you put something in the oven, that this something becomes hot. These are the type of mappings which the system will attempt to induce from the training data and apply at inference time to determine state changes.

To learn a state change classifier, for each rule class, the system estimates a distribution over state changes for each attribute conditioned on the presence of each AD tuple (or lack thereof). When evaluating an instance of a rule class in a sentence, we can then calculate the state change distribution of each attribute conditioned on each AD, and combine the distributions conditioned on each AD to get the result distribution (more on this later). Formally, the system is trying to approximate the following function:

$$f_r(c, a, x) = \Pr[C_{r,a} = c \mid x \in \vec{X_{r,a}}]$$

where

- $r = (\texttt{predicate}, \texttt{dep})$ is the rule class
- c and a are respectively the state change and attribute whose probability is being evaluated
- $x = (auxdep, auxentity)$ is the auxiliary dependency tuple being evaluated
- $C_{r,a}$ is a random variable representing the state change of attribute a over all instances of r in all sentences, and
- $X_{r,a}$ is a random variable similarly representing the set of auxiliary dependency tuples over all instances of r in all sentences.

Note that no joint distribution over auxiliary dependencies is calculated: it was observed early on that ADs rarely interact with each other (and for many sentences there is only one to begin with), so an independence assumption is appropriate. We therefore only condition on one at a time.

In addition to the normal auxiliary dependencies, two types of “wildcard" ADs are also included in the calculated distributions. The first of these has a wildcard for the entity field, e.g. `(obl:in, *)`. This matches any AD with the relation `obl:in` regardless of entity, so the distribution for this AD will be marginalized over auxiliary entity. This allows us to resolve out-of-vocabulary queries at evaluation time: when evaluating the distribution of a particular AD that doesn’t appear in the training set for a particular rule class but whose relation does, we can default to the relation wildcard, and use this marginalized distribution as an informed guess of the actual distribution for the AD. The second wildcard marginalizes over *all* instances of the rule class regardless of ADs, and therefore can be used when an unknown relation in an AD appears.

To combine the independent distributions of each AD for a particular application of a rule class, a simple arithmetic mean is used. This helps immunize the system against noise in the individual distributions – even if a particular state change has a low/zero probability in one distribution due to never being observed, it can end up being significant if it has high probability in the distributions for other ADs.

3.2 Story State Tracking and Conflict Detection

To detect conflicts in a story, an object state “simulation" of the story is performed, i.e. the system iterates through each sentence, tracking the states of objects based on predicted preconditions and effects. The conflict likelihood for a each object-attribute-timestep tuple is calculated as the probability that the precondition is violated. Note that due to its non-binary nature and lack of explicit preconditions and effects, the `location` attribute is *not* used or tracked.

To start, the probabilities of all attributes for all objects are initialized in accordance with the defaults in the original TRIP paper/codebase. Per these defaults, the attributes `conscious`, `exist`, `functional`, and `moveable` are all initiailized to *true* ($q = 1$) and all other attributes are initialized to *unknown* ($q = 0.5$). Additionally, for each attribute of each object, an array $\vec{h} = \langle 1.0, 0.0, \cdots, 0.0\rangle$ with a length one greater than the number of sentences in the story is initialized. This is known as the *affectors* distribution: it tracks the likelihood that a particular sentence was the last to update an object’s attribute at a specific timestep (sentence). The first element is the probability that it was never affected (i.e. by the initial conditions), while the subsequent elements correspond to each sentence in order.

Following initialization, for each sentence in the story (zero-indexed by j), the following steps are performed:

1. The distributions of state changes for each attribute of each object (entity) are predicted. Objects not appearing in the sentence or that don’t match a rule class are unaffected.

2. Precondition and effect probabilities for each attribute of each object are calculated based on the state change distribution. The result is:

   (a) P_{POS}, P_{NEG}, and P_{NONE} for the probability of a positive, negative, and null (nonexistent) precondition respectively, and

   (b) E_{POS}, E_{NEG}, E_{UNK}, and E_{NONE} for the positive, negative, unknown, and null effects, respectively. Notice that “unknown" and “none" have a subtle but important distinction: the former indicates uncertainty in the attribute (higher E_{UNK} pushes state probability towards 0.5), while the latter indicates no effect at all (the state as of the previous sentence is unmodified).

3. Calculate and store the conflict likelihood for each attribute of each object. Given the state probability q of some attribute being true for some object:

   $$P_{confl} = q \cdot P_{NEG} + (1 - q) \cdot P_{POS}$$

   which is equal to the probability that the precondition is violated.

4. Update the state probability of each attribute of each object for the next sentence:

   $$q' = \langle E_{POS},\ E_{NEG},\ E_{UNK},\ E_{NONE}\rangle \cdot \langle 1,\ 0,\ 0.5,\ q\rangle$$

   i.e., calculate the expected value of the state probability given the effects distribution.

5. Update the affectors distribution for each attribute of each object based on E_{NONE}:

$$\vec{h}' = \vec{h} \cdot E_{NONE}$$

$$\vec{h}'[j+1] \leftarrow (1 - E_{NONE})$$

i.e. the probability of all other sentences being the affector is scaled by the probability that this sentence is irrelevant, and the probability of this sentence being the affector is the probabilility that it is relevant (the complement of E_{NONE}).

With the results of this simulation, the system then proceeds to extract results for the plausibility, consistency, and verifiability tasks. We begin with plausibility. For each story, the argmax of the affector distribution for each object, attribute, and timestep is taken to determine the evidence. This reveals which sentence (if any) exerted the strongest influence on a particular object attribute at the time of the conflict (the breakpoint). If this indicates that the conflict was with the initial states (i.e. no sentence has significantly affected the state), we don't consider this conflict as we assume all conflicts must be between two sentences. Out of all remaining conflicts, we select the one with the highest probability. The story with the higher such conflict probability is selected as the implausible story.

With the implausible story decided, the system proceeds with the consistency task. Observe that the timestep (sentence index) of the selected conflict is the breakpoint, as this is the sentence at which the selected conflict arised. This is returned along with the evidence as calculated in the previous paragraph. For the verifiability task, we simply return the most likely precondition at the breakpoint sentence, and the most likely effect at the evidence sentence, for the object and attribute corresponding to the selected conflict. While the verifiability task doesn't require that the selected precondition and effect states be directly related to the conflict (so long as they are non-default), this policy will always output the states which were directly used to infer the conflict, providing an additional level of interpretability and clarity.

4 Evaluation

We will first introduce the results from the state change prediction task. In correctly predicting state changes, the proposed system achieves a precision of 81.6% and a recall of 51.4%. F-1 scores for the prediction of preconditions and effects separately compared with the best model from the TRIP paper is shown in Figure 3.

	Preconditions F1	Effects F1
Best from Paper	54.9%	57.3%
This System	65.4%	64.6%

Figure 3: Comparison of F1 between best model from TRIP paper and this system on test set.

Split	State Conflict Percentage
`train`	22.9%
`dev`	21.4%
`test`	23.4%

Figure 4: Portion of implausible stories with at least one conflict (violated precondition) between the evidence and breakpoint sentences.

Before introducing the results from the TRIP end tasks, we will present a statistic collected on the dataset itself, which will allow us to better understand the later results. Because our system uses explicit object state tracking to infer conflicts, an informative measure is the proportion of story pairs for which the evidence sentence (or at least one of them if multiple) has an effect which directly violates a precondition of the breakpoint sentence. If no such pair exists, the reasoning system presented herein would be unable to deduce its implausibility any better than random chance, even if all state change predictions are correct. The state conflict percentage for each split is presented in Figure 4.

With this in mind, we created a subset of the dataset containing only instances where the evidence and breakpoint have a state change conflict as described. We then evaluate the plausibility, consistency, and verifiability performance of our model on both the original test set and this refined version. In each of these cases, we also evaluate the system with the state change predictions replaced with the ground truth annotations from the dataset. This helps us understand the performance of the conflict detector in isolation, as well as how useful the state change predictor is in practice. The results are shown in Table 5.

For comparison, the best results on each task across all models evaluated in the original TRIP paper are included in Table 6.

	Entire `test` set	Refined `test` set
Predicted State Changes	54.7% / 9.5% / 6.4%	65.0% / 27.8% / 21.4%
Ground Truth State Changes	64.6% / 20.1% / 18.8%	91.9% / 76.7% / 72.8%

Figure 5: Performance on TRIP plausibility/consistency/verifiability tasks in various configurations

	Accuracy	Consistency	Verifiability
Best Model	78.3%	28.0%	10.6%
Random Baseline	47.8%	11.3%	0.0%

Figure 6: Results for best model and random baseline in the TRIP paper on each of the proposed tasks.

5 Discussion

The results on the end task in the standard configuration are notably poor, with the accuracy of 54.7% being only slightly better than a random guess. Also discouraging is the fact that the consistency score is lower than that of the random baseline. Interestingly, however, the verifiability score is higher than the baseline's 0%, suggesting that the system is able to make a statistically significant number of verifiable predictions.

Reasons for the poor performance are illuminated by the other results. When ground truth state changes are used instead of those predicted by the dependency graph system, the accuracy makes a modest climb to 64.6%, but the consistency and verifiability double and triple, respectively. This makes sense given the poor recall of 51.4% for the state change predictor, which appears to be missing out on a lot of attributes relevant to conflicts. However, it is interesting to note that despite having far worse end task performance, this model does have a higher F1 score in preconditions and effects than does the best model from the paper. This would seem to suggest that the paper's system relies less on the physical state changes for reasoning, as it is still able to get relatively good end task performance without a complete understanding of physical states.

We also see that, with errors from the state change prediction state eliminated, this system is able to not only achieve a greater verifiability than the best LM-based model but also has a greater *fraction* of correct plausibility predictions which are verifiable (28.0% vs 14.4%). Obviously this is far from an apples-to-apples comparison as the LM models don't have access to ground truth state changes, but it nonetheless suggests the effectiveness of this system's method of reasoning if paired with a more successful state change predictor.

This becomes all the more clear when we look at results on the refined test set, which only contains stories whose evidence and breakpoint sentences have some state conflict/violated precondition. Clearly, any model which bases its decisions on state changes (such as ours) would be unable to make informed decisions on data with no such conflicts. Interestingly, we see that the dataset as a whole has a surprisingly low fraction of such stories, with all splits in the low-mid twenties. This suggests that, while the state change annotations may be useful in supervising the intermediate reasoning of models, they don't themselves contain enough information to make plausibility decisions on the majority of the dataset. There are plenty of reasonable explanations for this, most notably that the relatively simplified state space consisting of a handful of binary attributes will fail to capture the conflict in many stories even if well-annotated.

Filtering out such stories that lack an evidence-breakpoint conflict shows drastic improvements around the board. The combination of the refined test set with the ground truth state changes shows that in isolation, when only given stories with conflicts which are understandable via state changes, the story simulator/conflict detector performs very well, with the majority of correct predictions being consistent and verifiable. Even when using the predicted state changes the improvement is significant, particularly in consistency and verifiability. Together, these results show that this system is fit for performing inference on data which is thoroughly explained by state changes, but that the majority of TRIP is too diverse to be effectively reasoned over using state changes alone.

6 Conclusion

Despite the underwhelming performance of the system as a whole, the performance comparison provided valuable insights on the nature of both the TRIP dataset as well as the problem it represents. Notably, we found that over 75% of implausible sentences in TRIP don't have a breakpoint-evidence conflict in physical state annotations, which severely limits the performance of a system which relies on this intermediate representation as the core element of its reasoning. However, when these "impossible" stories were controlled for, we saw a sharp increase in performance, showing this system's effectiveness on data with a tight semantic linkage between sentences and state annotations. However, TRIP as a whole is evidently too broad and open-domain to be reasoned over using only these physical state changes, so this system isn't particularly suited for it. These results pose a reminder that for many real-world datasets, creating intermediate annotations that fully explain the data is difficult if not completely infeasible, so effective approaches will need to conduct inference on more information than just the annotations.

References

Rinaldo Lima, Bernard ESPINASSE, and Fred Freitas. 2019. A logic-based relational learning approach to relation extraction: The OntoILPER system. *Engineering Applications of Artificial Intelligence*, 78:142–157.

Shane Storks, Qiaozi Gao, Yichi Zhang, and Joyce Chai. 2021. Tiered reasoning for intuitive physics: Toward verifiable commonsense language understanding.