

When No Paths Lead to Rome: Benchmarking Systematic Neural Relational Reasoning

Supplementary Materials

Anonymous Author(s)

Affiliation

Address

email

We discovered an issue with the results table and Figure 4c in the main paper. All our main findings remain valid. Please see the table below and Figure 1c for the updated results.

Table 1: Results of state-of-the-art models for systematic reasoning on the NoRA test sets.

		Trained with ambiguity					Trained without ambiguity			
		In dist.	D	W	BL	OPEC	In dist.	D-na	BL-na	OPEC-na
Exact-match Accuracy	ET (single-edge)	0.885	0.741	0.703	0.245	0.060	0.800	0.822	0.104	0.110
	ET (multi-edge)	0.900	0.493	0.790	0.785	0.037	0.800	0.494	0.056	0.077
	RAT (single-edge)	0.721	0.494	0.615	0.234	0.042	0.800	0.493	0.092	0.094
	RAT (multi-edge)	0.900	0.676	0.668	0.540	0.028	0.827	0.768	0.023	0.017
	EpiGNN-min (margin)	0.334	0.491	0.176	0.000	0.000	0.208	0.485	0.000	0.000
	EpiGNN-min (BCE)	0.451	0.665	0.456	0.154	0.005	0.475	0.488	0.008	0.025
	EpiGNN-mu1 (BCE)	0.520	0.604	0.491	0.156	0.009	0.539	0.716	0.027	0.045
	NBFNet (margin)	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
	NBFNet (BCE)	0.576	0.531	0.460	0.153	0.009	0.679	0.764	0.012	0.043
	R-GCN	0.347	0.672	0.283	0.051	0.032	0.579	0.740	0.018	0.012
Weighted F1	ET (single-edge)	-	0.740	0.814	0.432	0.413	-	0.816	0.233	0.410
	ET (multi-edge)	-	0.335	0.860	0.888	0.504	-	0.336	0.120	0.394
	RAT (single-edge)	-	0.329	0.744	0.437	0.399	-	0.333	0.188	0.383
	RAT (multi-edge)	-	0.677	0.766	0.747	0.457	-	0.759	0.067	0.294
	EpiGNN-min (margin)	-	0.326	0.296	0.082	0.116	-	0.326	0.112	0.206
	EpiGNN-min (BCE)	-	0.625	0.633	0.316	0.180	-	0.319	0.049	0.218
	EpiGNN-mu1 (BCE)	-	0.554	0.667	0.320	0.185	-	0.717	0.076	0.249
	NBFNet (margin)	-	0.000	0.000	0.000	0.000	-	0.000	0.000	0.000
	NBFNet (BCE)	-	0.646	0.665	0.347	0.225	-	0.775	0.083	0.261
	R-GCN	-	0.691	0.455	0.189	0.286	-	0.704	0.122	0.195

1 Updated experimental results

1.1 Main results

The updated results are shown in Table 1, in terms of accuracy (i.e. we measure how often the multi-hot prediction of the relation set \mathcal{R} exactly matches the ground truth). Models trained on *Train-a* are evaluated on the four test splits with ambiguity, while models trained on *Train-na* are evaluated on the three remaining splits. We now also report on an in-distribution test split, containing problem instances with the same constraints on the difficulty level as the training examples.

ETs emerge as the overall best-performing model. The single-edge variant generally performs better, although it performs poorly on BL.

Surprisingly, we often find that the results with ambiguity are higher than the corresponding results without ambiguity. To some extent, this can be explained by differences in the distribution of the

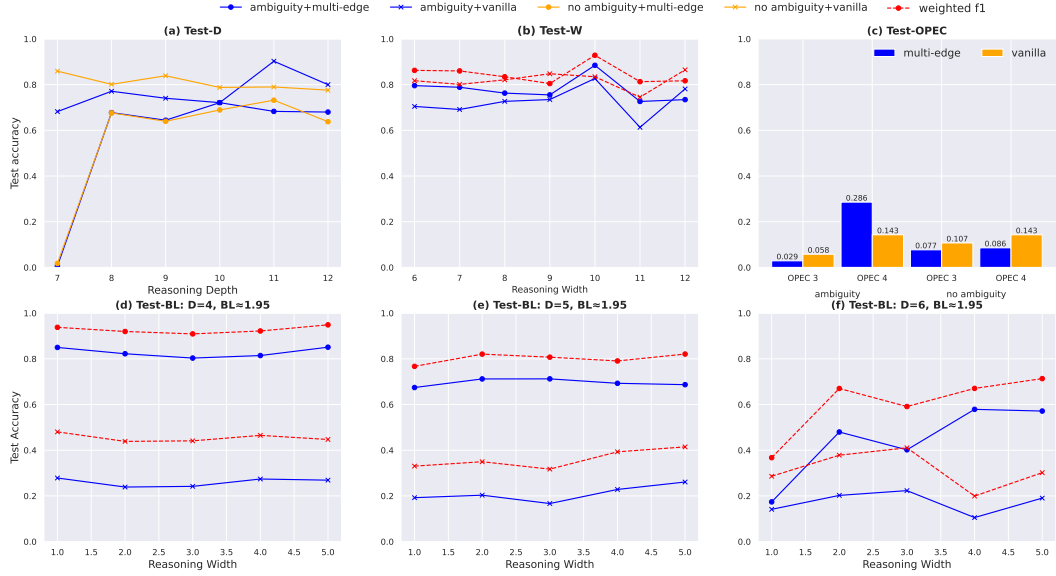


Figure 1: Depth-first analysis of the performance of ETs on various splits of the dataset. Weighted F1 scores per class are computed to avoid class imbalances affecting the metric score for the various test splits.

problem instances, where the test sets without ambiguity sometimes have a higher proportion of difficult problems. As we show in the analysis below, when controlling for problem difficulty, ambiguous instances can be harder than non-ambiguous instances. However, ambiguities may also introduce the possibility for reasoning shortcuts, which some models are able to exploit. The GNN methods all perform very poorly on the BL and OPEC test sets (both with and without ambiguity), which can be explained by their strong alignment with path-based reasoning. ETs also perform poorly on OPEC, but they clearly outperform the GNNs on BL.

Among the GNN models, EpiGNNs with BCE loss and multiplication-based aggregation perform the best on the out-of-distribution test splits with ambiguity. NBFNet with BCE loss performs better on the in-distribution test splits and on D-na. Finally, the results also confirm that the margin-based loss, where the model outputs are target edge vectors that are contrastively trained, is not suitable for this multi-label setting.

The weighted F1-score metric is calculated as the F1-score for each label, aggregated using a weighted mean (based on their frequency in the dataset). Exact-match accuracy requires models to predict all labels correctly when multiple labels are true. The weighted F1 metric still provides positive contribution when at least some labels are predicted correctly. Consequently, this metric can often yield higher scores. This is evident in the test-OPEC dataset, where multiple true relationships exist (e.g., “aunt” and “maternal aunt”). Identifying “maternal aunt” requires using off-path information.

1.2 Depth-first analyses for other baselines

We provide the depth first analyses for a GNN (EpiGNN) in Figure 2, RAT in Figure 3 and ET in Figure 1. Broadly, the trends observed in the main text hold for other models with respect to length and width generalization. For RAT, similar to ET, the single-edge or vanilla model has a higher OPEC performance than it multi-edge counterpart in Figure 3(c). Also, the multi-edge RAT is better at width generalization in figures 3(d)-(f). We also show a weighted F1 that overcomes class imbalances for some figures which highlights a similar trend to the accuracy curves. For the EpiGNN, the mul aggregation function does notably better than min for OPEC in figures 2(c) and also on the Test-D split in figures 2(a).

Note that OPEC 4 is easier than OPEC 3 for all models including ET in Figure 1 after the results update that fixed a minor typo in the code that affected the order with which the test data were collected. All other findings in the main text concerning ET are unchanged.

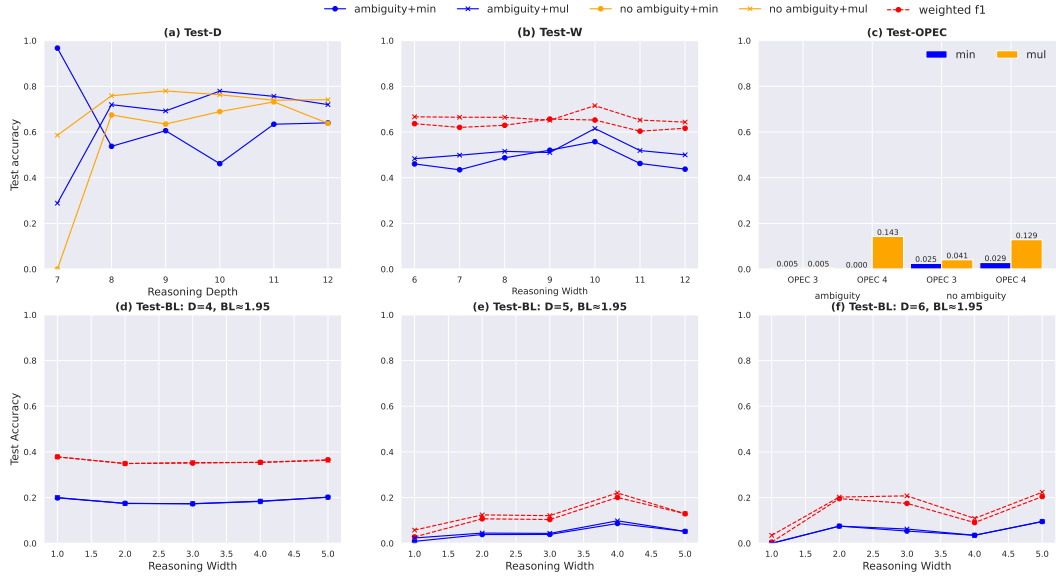


Figure 2: Depth-first analysis of the performance of EpiGNN on various splits of the dataset. Weighted F1 scores per class are computed to avoid class imbalances affecting the metric score for the various test splits.

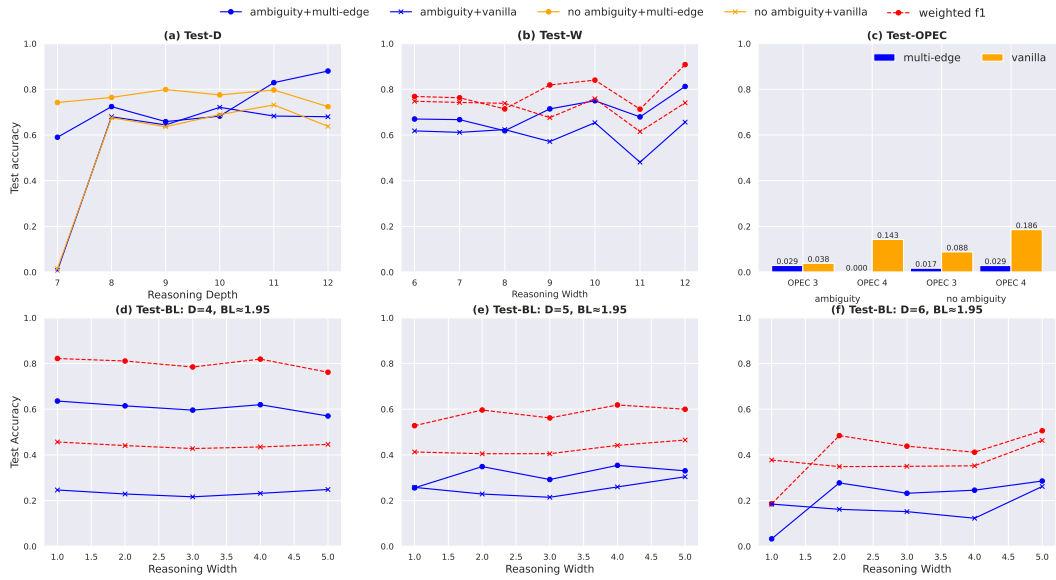


Figure 3: Depth-first analysis of the performance of RAT on various splits of the dataset. Weighted F1 scores per class are computed to avoid class imbalances affecting the metric score for the various test splits.

2 Notation And task: Intuitive walkthrough

Here we give an intuitive overview using examples instead of formal definitions for the notations introduced formally in the main paper. We focus on stories without ambiguity, ambiguity is discussed at length with examples in Section 10. We use **Answer Set Programming (ASP)** as the underlying language to encode problem instances in NoRA.

The dataset is composed of three parts: **world rules**, **stories**, and **entailed atoms**. The *world rules* define the underlying regularity of relationships in a given universe. These rules are not exposed to the user. The goal of learning models is to infer these hidden rules through example instances and apply them to reasoning tasks. We define two sets of world rules (i.e. two worlds) for the benchmark. **NoRA-mini**: A simplified world used for illustrative purposes. **NoRA-full**: A richer and more fine-grained world with a broader set of rules, used to generate the full benchmark.

World rules Figure 4(a) shows an example of the world rules in NoRA-mini. These rules fall into three categories: definite rules, constraints and facts.

A **definite rule** consists of a **body** and a **head**. The body is a conjunction of one or more atoms; the head is a single atom. In the absence of constraints, we can think of these rules in terms of standard implication: if all atoms in the body are true, then the head must also be true. For example, consider the following rule from Figure 4(a):

```
living_in_same_place(X,Z) :-living_in_same_place(X,Y),  
                             living_in_same_place(Y,Z).
```

This rule states: for any entities X, Y, and Z, if X lives in the same place as Y, and Y lives in the same place as Z, then X also lives in the same place as Z.

Constraints are rules without a head. They specify sets of atoms that are *not* allowed to be simultaneously true. For example:

```
:- belongs_to(X, underage), parent_of(X, Y).
```

This constraint expresses that an underage person cannot be a parent. Note, in our notation, `rel(X, Y)` means X is rel of Y. So, `parent_of(X, Y)` means X is the parent of Y.

Facts are atoms that are always true. They are rules without a body and are often used to declare properties of constants. For example:

```
is_agegroup(underage).
```

Stories Each story consists of a set of *story facts*, which are grounded atoms, i.e., they contain no variables. For example, in Figure 4(b), the fact:

```
school_mates_with(ram, irfan).
```

states that ram and irfan are schoolmates. Combining the story facts with the fixed world rules one obtains a logic program (abusing terminology we sometimes call this logic program the story).

Entailed atoms via stable models Stable models/answer sets are the solution of a logic program. Intuitively, they are a minimal set of atoms/facts that follow from the logic program (see Section 11 for formal definitions). A stable model includes both the explicitly stated story facts and additional possible atoms that follow logically. These additional atoms are called **entailed atoms**. Figure 4(c) shows the stable model of the story from Figure 4(b). The entailed atoms are highlighted. If an entailed atom has a binary predicate (relationship), its first argument is called the source entity and its second argument the target entity.

Example format and reasoning task While the world rules are kept fixed, multiple logic programs are generated by randomly sampling many sets of story facts. For each such program, the corresponding entailed atoms are computed.

An individual **example** in the dataset consists of:

- The story facts (input), encoded as a graph.
- The target and source entities of an entailed atom, which define the query.

Let a and b be the atoms defined in the query. The task is to predict all relations r such that $r(a, b)$ can be entailed from the story facts. For the example in Figure 4(d), the entailed atom is `living_in_same_place(irfan, lola)`. A model attempting to solve NoRA will be shown the story-facts, the source entity `irfan`, and the target entity `lola`, and it must infer the missing predicate—`living_in_same_place` in this case. In NoRA-full, multiple relationships/predicates might be true.

Reasoning depth The difficulty of deriving an entailed atom is influenced by the number of reasoning steps required to reach it, excluding the direct use of story-facts. For example, Figure 4(e) shows that in NoRA-mini, deriving `living_in_same_place(irfan, lola)` requires six inference steps. Since derivations may not be unique, we use derivations that are minimal (in a sense) to calculate the metric called **reasoning depth**.

3 Data generation and sampling

Data generation process We generate approximately 500,000 example instances by repeatedly sampling random story facts, as detailed below. The same set of world rules is used for all stories (see Table 2). A single logic program may have multiple entailed atoms, and hence gives rise to multiple example instances in the final dataset. Each story contains two types of entities: *persons* and *places*. First, all entities are generated and assigned a type (person or place). This assignment is governed by a parameter called `person_percent`, which determines the probability that an entity is a person. Higher values of `person_percent` result in more persons, while lower values yield more places. The value of `person_percent` for each story is recorded and included in the dataset. When a person entity is introduced, its gender is either assigned (male or female) or left unspecified. This is controlled by a per-story parameter called `no_gender_assign`, which captures the proportion of person entities with unspecified gender.

Relationships are sampled from the list of binary predicates defined in the world rules (see Table 2). For each story fact, a predicate is sampled and applied to a randomly chosen pair of entities, resulting in a fact of the form `rel(e1, e2)` being added to the story. After each fact is added, the resulting logic program is solved to ensure that at least one answer set exists—i.e., that the story remains consistent. If the added fact introduces a contradiction, it is discarded and a new one is sampled instead. The number of entities per story is sampled uniformly between 20 and 50, and the total number of story facts per instance ranges from 30 to 75. Details of how ambiguous facts are introduced into the stories are provided in Section 10.

While there are many possible relationships that can hold between two people, we only consider one relationship between people and places, namely `living_in`. We thus need to make sure that queries where the source entity is a person and the target entity is a place are not trivial to answer, i.e. that models cannot rely on the shortcut that in such cases the answer is always the singleton `{living_in}`. To this end, we have introduced an additional predicate `not_living_in`, which is inferred by the following rule, encoding the fact that a person can only live in one place:

$$\text{not_living_in}(X, Z) \text{ :- } \text{living_in}(X, Y), Y \neq Z$$

Dataset construction From this pool of example instances, we construct training and testing datasets under the constraint that *all target query relationships in the test sets must appear in the training data*. To balance the distribution of problem difficulty in the training set, we use *inverse transform sampling*. A general discussion of the nuances of re-sampling techniques can be found in [Levina and Priesemann, 2017, Das et al., 2022]. We use rejections sampling, enabling stratified sampling via quantile functions to obtain the training set. See discussion below:

Difficulty stratification Examples are binned by four metrics:

- **Reasoning Depth** (3 bins uniformly covering the range),
- **Reasoning Width** (3 bins uniformly covering the range),

(a) World Rules

Definite Rules

```
living_in_same_place(Y, X) :-  
  school_mates_with(Y, X).  
living_in_same_place(Y, X) :-  
  belongs_to(X, underage), parent_of(Y,  
  X).  
living_in_same_place(Y, X) :-  
  living_in_same_place(X, Y).  
living_in(Y, Z) :-  
  living_in_same_place(X, Y), living_in(X,  
  Z).  
belongs_to(X, underage) :-  
  school_mates_with(X, U).  
living_in_same_place(X, Z) :-  
  living_in_same_place(X, Y),  
  living_in_same_place(Y, Z).
```

Constraint

```
:- belongs_to(X, underage), parent_of(X,  
  Y).
```

Facts

```
is_agegroup(underage).
```

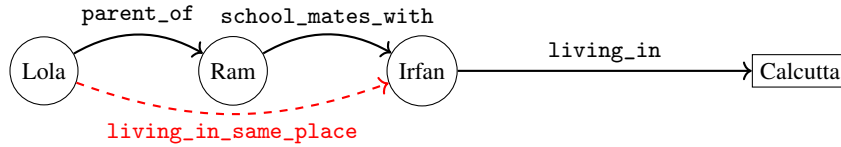
(b) Story Facts

```
school_mates_with(ram, irfan).  
parent_of(lola, ram).  
living_in(irfan, calcutta).
```

(c) Stable Model

```
living_in_same_place(ram, irfan),  
living_in_same_place(lola, ram),  
living_in_same_place(ram, lola),  
living_in_same_place(irfan, ram),  
living_in_same_place(lola, irfan),  
living_in_same_place(irfan, lola),  
living_in_same_place(ram, ram),  
living_in_same_place(lola, lola),  
living_in_same_place(irfan, irfan),  
school_mates_with(ram, irfan),  
parent_of(lola, ram),  
belongs_to(ram, underage),  
living_in(irfan, calcutta),  
living_in(ram, calcutta),  
living_in(lola, calcutta),  
is_agegroup(underage).
```

(d) Visualizing the Reasoning Task



(e) Derivation for `living_in_same_place(irfan, lola)`

```
Fact: school_mates_with(ram, irfan)  
1. living_in_same_place(ram, irfan) :- school_mates_with(ram, irfan).  
2. living_in_same_place(irfan, ram) :- living_in_same_place(ram, irfan).  
3. belongs_to(ram, underage) :- school_mates_with(ram, irfan).  
Fact: parent_of(lola, ram)  
4. living_in_same_place(lola, ram) :- belongs_to(ram, underage),  
   parent_of(lola, ram).  
5. living_in_same_place(ram, lola) :- living_in_same_place(lola, ram).  
6. living_in_same_place(irfan, lola) :- living_in_same_place(irfan, ram),  
   living_in_same_place(ram, lola).
```

Figure 4: Illustration of (a) world rules, (b) story facts, (c) stable model with entailed atoms in **red**, and (d) visual reasoning task: *What is the relationship between Lola and Irfan?* Correct answer: `living_in_same_place`. (e) Reasoning depth for the entailed atom in d.

- 134 • **Branching Length (BL)** (2 bins uniformly covering the range),
135 • **OPEC** (2 bins: 0 vs. 1–2).

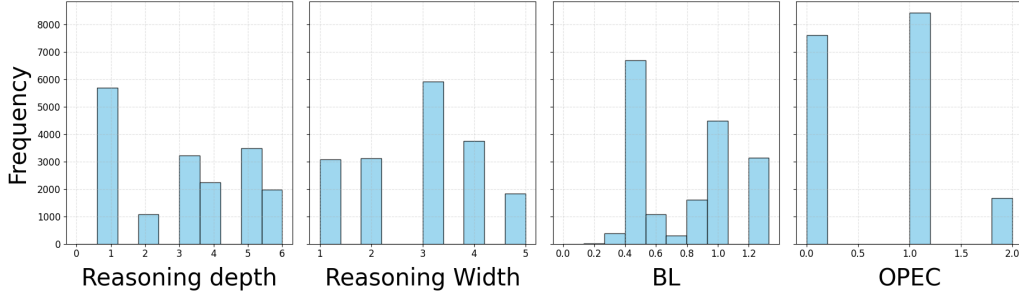
136 The sampling process follows a rejection-based strategy, beginning with a large pool of candidate
137 examples and iteratively removing samples to achieve a balanced marginal distribution over difficulty
138 metrics. We aim to balance the dataset along several predefined difficulty axes, denoted as S_{diff} . Each
139 axis in S_{diff} corresponds to a difficulty metric—such as reasoning depth, branching length (BL), or
140 OPEC—and is associated with a specific number of target bins.

141 Sampling proceeds in multiple passes (up to a maximum of `max_p`), terminating early if a satisfactory
142 balance is achieved. In each pass, the following steps are performed:

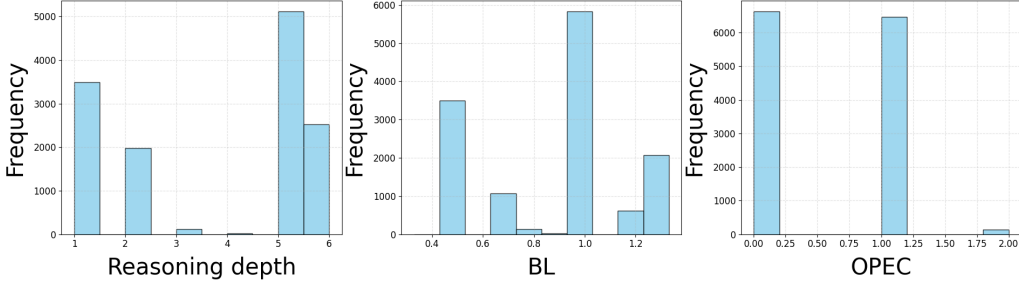
- 143 1. **Score Initialization:** Initialize a removal score of zero for all examples.

- 144 2. **Metric-wise Imbalance Scoring:** For each difficulty metric $p_{dm} \in S_{diff}$, bin the dataset into
145 $num_bins[p_{dm}]$ quantile-based bins. Identify the over-represented bins (i.e., bins whose
146 sample count exceeds the target). For every example in an over-represented bin, increment
147 its score by one.
- 148 3. **Overrepresentation Removal:** After processing all metrics, make a second pass over S_{diff} .
149 For each over-represented bin, identify examples with the highest accumulated scores and
150 remove them first.

151 **Training Data distributions** This two-pass process, repeated across sampling rounds, ensures
152 that examples contributing disproportionately to skewed distributions are pruned while maintaining
153 as much diversity and coverage as possible. Figures 5 show the difficulty metric distributions for
154 Train-A (ambiguous) and Train-NA (non-ambiguous) sets.



(a) Train-A (with ambiguity)



(b) Train-NA (no ambiguity)

Figure 5: Distributions of difficulty metrics across training sets.

155 **Held out test data** As mentioned in the main text, we evaluate on various held-out test datasets,
156 where each test dataset is designed to be hard according to one difficulty metric while remaining
157 in-distribution (compared to the training set) in terms of other difficulty metrics. For the test datasets
158 with ambiguity:

- 159 • For Test-D, we ensure that a positive refinement (refinements that have an answer set) has a
160 reasoning depth greater than 6. This is to ensure the problem is actually difficult, as models
161 often take shortcuts by ignoring the derivation of the contradiction in other refinements.
- 162 • Likewise, for Test-BL and Test-OPEC, we make sure a positive refinement has $BL > 1.5$
163 and $OPEC \geq 3$, respectively.

164 4 NoRA rules reflect real-world intuitions

165 We contend that the world rules from NoRA are *realistic*, in that human beings are able to intuitively
166 accept them to be true (or at least plausible). We believe this is a useful feature of our dataset, as it
167 makes it easier to compare neural reasoning models, such as the ones we discuss in this paper, with

LLM based approaches. To test our hypothesis that the rules are realistic, we used an LLM, namely o4-mini, to complete the 284 rules with zero-shot prompting in an open-ended question answering format. Specifically, given the body of a rule, we asked the model to predict the head.

Here are the results for the three types of NoRA rules:

• Rule type: constraint	89.0/90.0 (98.9%) correct
• Rule type: definite_rule	184.0/194.0 (94.8%) correct
• Overall accuracy: 96.1%	

The prompt we used first defines all predicates:

```

Here are some Predicate Definitions:
- "child_of(X,Y)": "X is a child of Y. Order matters: the first argument is
  the child, the second is the parent"
...
[all predicates are likewise described]
```

The next part of the prompt differs for rules and constraints. For definite rules, where the head is a binary predicate, we use the following prompt:

```

Given that all of the following atoms are true:
grandparent_of(X,Y), belongs_to_group(X, male)

What is the relationship between X, Y?
Provide only the predicate with variables in exactly this format:
rel(X,Y)
What is the predicate name that should replace 'rel'? Your response should
be rel(X,Y), where rel is your guess. If you think multiple predicates
could work, you must choose the most specific one. For example:
- If both brother and sibling are suitable, choose brother as it's more
specific.
```

For a constraint, we instead use the following prompt:

```

Given that all of the following atoms are true:
has_property(Y, no_daughters), daughter_of(X,Y)

Can this combination of facts logically exist?
Answer exactly one of:
[Possible] [Impossible] [Inevitable]
```

As a sanity check, we also tested the LRM (o4-mini) with 90 random constraints not in the NoRA rules, but which use the same predicates. Each of these non-world constraints is a slight modification of NoRA constraints. For example, while “:- aunt_or_uncle_of(Y,X), grandchild_of(Y,X).” is a NoRA constraint stating someone’s aunt cannot also be their grandchild, we modified it to the non-world constraint “:- aunt_or_uncle_of(Y,X), grandchild_of(U,V).” This should be possible as U,V and X,Y can be different pairs of people, and thus in the real world there is no obstruction for this to be true. We note the o4-model’s response with the same constraint prompts as above for these non-world constraints: the model **always responded with “[Possible]”**.

5 Experiments with trainable relational reasoning models

5.1 Loss functions

Margin loss Let us write \mathbf{x}_i to denote the prediction that is obtained by the model for training example i , and let \mathbf{r}_i denote the embedding of relation r_i . We write \mathbf{r}'_i to denote some negative example, i.e. \mathbf{r}'_i for some $r'_i \in \mathcal{R} \setminus \{r_i\}$, the set of all possible relations in NoRA. In the case of

multiple target relation vectors, \mathbf{r}_i , we take the average, $\bar{\mathbf{r}}_i$. The overall loss function is:

$$\mathcal{L}_{\text{margin}} = \sum_{i \in \mathcal{D}} \max \left(0, \text{CE}(\mathbf{x}_i, \bar{\mathbf{r}}_i) - \text{CE}(\mathbf{x}_i, \mathbf{r}'_i) + \Delta \right) \quad (1)$$

where CE is the cross entropy function and Δ is the margin value that is set to 1.0 after hyperparameter tuning. The margin loss over multiple models involves an additional sum over the cross entropy differences predicted target relation per model inside the max. At inference time, the target relation is predicted using the negative cross entropy as a score function, with respect to every relation vector in \mathcal{R} .

Multilabel binary cross entropy We use a multilabel version of the Binary Cross Entropy (BCE) loss for the multilabel multiclass classification setting for all NoRA problems. The logits for each class are transformed using a sigmoid function and then the problem is treated as a binary classification problem with a multi-hot target binary vector.

$$\mathcal{L}_{\text{BCE}} = \sum_{i \in \mathcal{D}, j \in \mathcal{R}} \text{CE}(\sigma(x_{ij}), y_{ij}) \quad (2)$$

where i is the sample index, j is the relation index, x_{ij} is the predicted logit and the y_{ij} is the one-hot target class label.

5.2 Initialization and compute

All trainable parameters for the models are uniformly initialized. All baseline results that were obtained by us were hyperparameter-tuned using grid search, as detailed below. Some baseline results were obtained from their corresponding papers and reported verbatim (as indicated in the results tables). All experiments were conducted using RTX 4090 GPUs. A single experiment using the trainable models can be conducted within a few minutes to 1 hour on a single GPU. This includes training and testing a single model on any test split of NoRA. A single hyperparameter set evaluation is done on about 20% of the total epochs and training data compared to a full experiment and would take a commensurate amount of time.

5.3 Hyperparameter settings

We use the Adam optimizer [Kingma and Ba, 2017]. All the models were hyperparameter tuned using an economical grid search over key parameters. For ET and RAT, a grid search was performed over the number of attention heads, hidden dimension size, the number of message passing rounds, and dropout rate. For the GNNs, we grid searched over the hidden dimension size, the number of message passing rounds. In addition for EpiGNN, we also tuned the number of facets. All the optimal hyperparameters are available in the companion code with the manuscript.

6 Experiments with large reasoning models

Rule recovery task In addition to the results presented in Section 5 of the main paper on the performance of Large Reasoning Models (LRMs), we evaluate LRM models on a second diagnostic task. Since all NoRA world rules are provided to the model, we additionally task the LRM with outputting the complete set of world rules it used to solve the given query completion task. We call this the *Rule Recovery Task*. Successful completion of the query completion task *without* correct rule recovery indicates that the model may be taking shortcuts to arrive at the correct answer without following the intended reasoning steps.

Figure 6 presents our results side-by-side with the query completion results (a copy of Fig. 3(b) from the main paper), for easy comparison. This parallel presentation is particularly informative as both tasks are evaluated on identical example instances. The results reveal that while models may have good precision on the rule recovery task, recalling all applicable rules proves substantially more difficult, especially in cases requiring reasoning with significant off-path complexity. The models are evaluated on examples such as those in Figure 3 of the main paper.

For Figure 3(d) in the main paper, the mean success rate and its 95% confidence interval are estimated using bootstrapping. For Figure 3(b), the performance of the o3 variant is assessed across

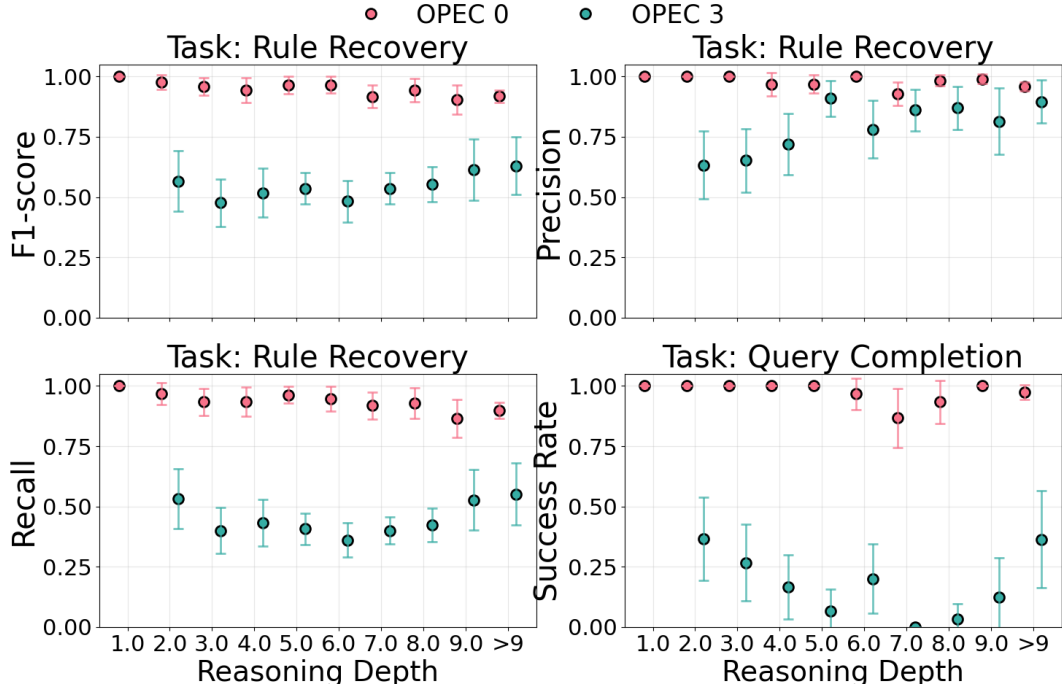


Figure 6: Performance of OpenAI’s o3 model on Query Completion and Rule Recovery Tasks. Results separated according to OPEC and the reasoning depth of examples.

different reasoning depths. Mean success rates are computed as sample averages, with confidence intervals derived via normal approximation using standard deviation estimates from a Binomial parameterization.

Prompt format for query completion and rule recovery tasks

The large reasoning model (LRM) is prompted with the following structure for both the query completion and rule recovery tasks:

Section 1: Predicate definitions *Here are some Predicate Definitions:*

- `grandparent_of(X,Y)`: X is a grandparent of Y. Order matters: the first argument is the grandparent, the second is the grandchild.
- ... [Additional predicate definitions follow in the actual prompt]

Section 2: World rules There are three types of rules:

- **A. Definite Rule:** Has a head and a body. It means if all atoms in the body are true, then the head is true.
- **B. Constraint:** Has only a body. It states that the atoms in the body cannot all be true at the same time.
- **C. Fact:** Has only a head. This atom is always true.

Variables are capitalized and rules with variables hold universally for all substitutions.

Here are the NoRA world rules. Rules are indexed and follow the format:

Head :- Body.

Example:

278 1: grandparent_of(Y,X) :- grandchild_of(X,Y).

279 [All world rules are then enumerated by index.]

280 **Section 3: Two exemplars TASK:** You will be given a story made up of predicates describing
281 relationships between entities ...

282 **Example 1:**

283 0 is a is_person.
284 0 is a is_female.
285 1 is a is_place.
286 2 is a is_person.
287 3 is a is_person.
288 ... [more story facts]

289 *Query:* What is the relation between 11 and 23? What are the indexes of the world rules you will
290 need to derive this?

291 *Response:*

292 • **query_label:** niece_of
293 • **rules_used:** {192, 64, 194, 46, 23}
294 • **reasoning:**
295 From story fact 23, we know that individual 23 is the maternal aunt of individual 11.
296 Applying world rule 192, we deduce that 23 is the maternal aunt or uncle of 11. Rule
297 194 generalizes this to aunt_or_uncle_of. Rule 23 inverts this relation to yield that 11
298 is a nibbling of 23. The story also indicates that 37 is the parent of 11 and has no sons.
299 Applying rule 64, we infer that 11 is female. Rule 46 finally allows us to conclude that 11 is
300 the niece of 23.

301 **Example 2:**

302 [Another guided exemplar with similar format]

303 **Section 4: Actual problem instance STORY:**

304 0 is a is_person.
305 1 is a is_person.
306 1 is a is_male.
307 2 is a is_person.
308 ... [More story facts]

309 **QUERY:**

310 What is the predicate between 35 and 6? If a relationship between 35 and 6 is
311 explicitly given in the story facts, and there is some other relationship that is also
312 true, you need to uncover the unstated predicate. If multiple predicates capture the
313 relationship between 35 and 6, choose the most specific one.
314 What are the indexes of the world rules you will need to derive this?

315 **Expected Output:**

316 • **query_label:** ...
317 • **rules_used:** { ... }
318 • **reasoning:** ...

319 **7 Large reasoning models use shortcuts**

320 In the NoRA world rules, the knowledge that “a sibling of my sibling is also my sibling” is *not*
321 explicitly encoded as a definite rule. To prove it, one has to chain through the parent–child relations,
322 repeatedly applying the following three world rules:

323 (W1) `child_of(Y,X) :- child_of(Z_1,X), sibling_of(Y,Z_1).`
 324 (W2) `parent_of(Y,X) :- sibling_of(Z_1,X), parent_of(Y,Z_1).`
 325 (W3) `sibling_of(Y,X) :- parent_of(Z_1,X), child_of(Y,Z_1), Y \neq X.`

326 Even before normalising gendered relations, establishing sibling transitivity therefore demands at
 327 least four inference steps. In contrast, LRMs have internalized the following direct rule:

328 (S) `sibling_of(X,Z) :- sibling_of(X,Y), sibling_of(Y,Z).`

329 Rule (S) is not a NoRA world rule, yet LRMs (like o3) can apply it, collapsing a multi-hop proof
 330 into a single step. Consequently, tasks that appear to require deep reasoning are effectively much
 331 shallower for such models. Every test instance that o3 solved at a reasoning depth > 9 contained
 332 sibling transitivity as a sub-problem, so the model’s actual reasoning depth was far lower than our
 333 theoretical estimate. An example instance with OPEC > 9 that the o3 model predicts correctly is
 334 shown in 7.

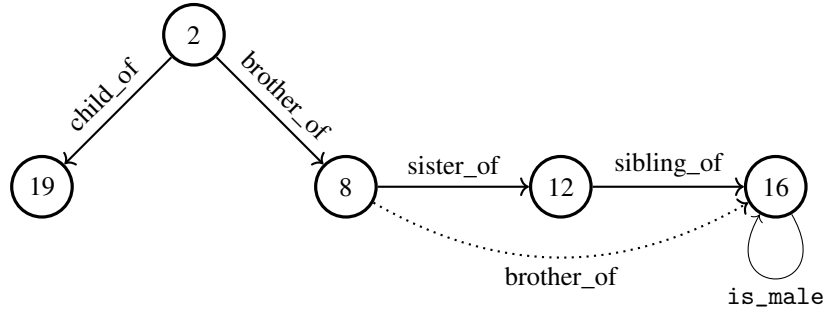


Figure 7: Illustrative fragment of the NoRA graph. Solid edges follow world rules (W1–W3); the dotted edge shows the shortcut (S) inferred by the LRM.

335 8 Derivation step sensitivity

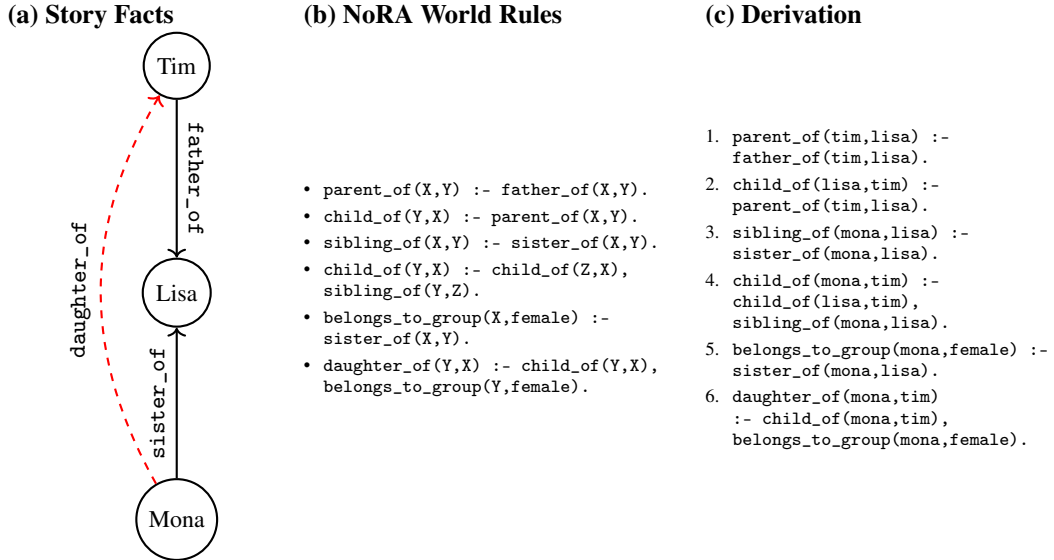


Figure 8: Example showing a derivation using a minimal number of rules.

336 The number of derivation steps is sensitive to the precise way in which world rules are framed.
 337 To illustrate this, consider our NoRA world rules, which are designed to be minimal and avoid

338 redundancy. These rules imply certain relationships (which are not explicit in the world rules). A
 339 model could also memorize these implied rules. This would result in shorter derivations but would
 340 necessitate memorizing a larger number of rules.

341 Consider the example in Figure 8. Using the NoRA rules, entailing that Mona is the daughter of
 342 Tim requires six derivation steps. However, a rule not explicitly stated in the NoRA world rules, but
 343 implied by them, is:

`daughter_of(Z,Y) :- father_of(X,Y), sister_of(Z,Y).`

344 If models were to learn such implied rules directly, the derivation for the same entailment would be
 345 reduced to a single step.

346 CLUTRR does not count inverse relationships, such as `parent_of(X,Y) :- child_of(Y,X)`, as
 347 derivation steps, whereas such steps are counted in NoRA. Since we have diverse types of rules in
 348 NoRA, making a judgment on what counts as a derivation step requires more consideration.

349 9 Comparing BL and OPEC as measures of non-path reasoning

350 The Backtrack Load (BL) is the ratio of the number of inference steps to the number of entities
 351 involved. As noted in Section 8, the number of derivation steps is dependent on the way the world
 352 rules are set up. Since we have avoided including redundant rules when specifying the world rules,
 353 many problems have a large number of derivation steps. BL is therefore susceptible to overestimating
 354 non-path difficulty. An important advantage of BL, however, is that it is capable of identifying
 355 non-path reasoning even in cases without non-path edges. On the other hand, OPEC can only identify
 356 non-path reasoning when there are off-path edges (i.e. edges which are not on any path between
 357 source and target), but it is not dependent on how the world rules are encoded.

358 The two metrics are still correlated, as demonstrated by the violin plot in Figure 9.

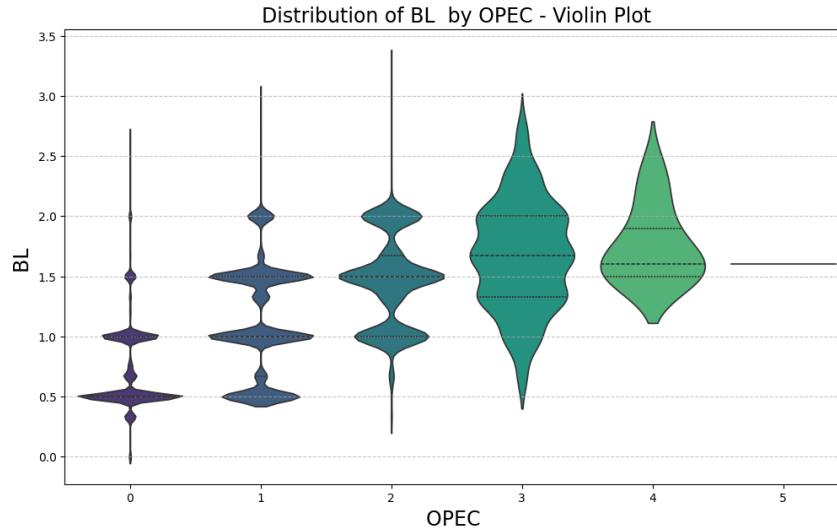


Figure 9: Comparison of BL and OPEC metrics, showing their correlation.

359 10 Ambiguous facts, story encodings and reasoning width

360 Real-world text is often ambiguous or incomplete. One motivation for including ambiguity in NoRA
 361 is that relation extraction pipelines based on coreference resolution can introduce noise or uncertainty.
 362 Additionally, narratives themselves may be under-specified. Consider the story fragment:

363 *Paul went to his grandmother Sheila's house... Sheila's son Dixon was not happy*
 364 *with her decisions.*

(a) Ambiguous Story Facts

```
belongs_to(ryan, underage).
school_mates_with(cole, will).
living_in_same_place(sheila, lalit).
living_in(lalit, kgp).
living_in(phil, kgp).
```

```
1{living_in(cole, east_rock);
   living_in(cole, dwight)}1.
1{child_of(ryan, brutus);
   child_of(ryan, cole)}1.
1{colleague_of(brutus, phil);
   colleague_of(brutus, sheila)}1.
```

(b) Refinements and Derivations

(i)

```
colleague_of(brutus, phil),
child_of(ryan, brutus),
living_in(cole, east_rock)
```

```
living_in(brutus, kgp) :-
colleague_of(brutus, phil),
living_in(phil, kgp).
living_in(ryan, kgp) :-
belongs_to(ryan, underage),
parent_of(brutus, ryan),
living_in(brutus, kgp).
```

(ii)

```
colleague_of(brutus, phil),
child_of(ryan, brutus),
living_in(cole, dwight)
```

Same derivation as (i).

(iii)

```
colleague_of(brutus, sheila),
child_of(ryan, brutus),
living_in(cole, east_rock)
```

```
living_in(sheila, kgp) :-
living_in_same_place(sheila, lalit),
living_in(lalit, kgp).
living_in(brutus, kgp) :-
colleague_of(brutus, sheila),
living_in(sheila, kgp).
living_in(ryan, kgp) :-
belongs_to(ryan, underage),
parent_of(brutus, ryan),
living_in(brutus, kgp).
```

(iv)

```
colleague_of(brutus, sheila),
child_of(ryan, brutus),
living_in(cole, dwight)
```

Same derivation as (iii).

(v)

```
child_of(ryan, cole), living_in(cole,
east_rock),
colleague_of(brutus, sheila)
```

```
belongs_to(cole, underage) :-
school_mates_with(cole, will).
```

```
:- belongs_to(cole, underage),
parent_of(cole, ryan).
Contradiction.
```

(vi)

```
child_of(ryan, cole), living_in(cole,
dwight),
colleague_of(brutus, phil)
```

Same contradiction as (v).

(vii)

```
child_of(ryan, cole), living_in(cole,
east_rock),
colleague_of(brutus, phil)
```

Same contradiction as (v).

(viii)

```
child_of(ryan, cole), living_in(cole,
dwight),
colleague_of(brutus, sheila)
```

Same contradiction as (v).

Figure 10: (a) An ambiguous story in NoRA, with three cardinality-based facts (highlighted in blue). (b) Each numbered box corresponds to a refinement. The top rectangle in each branch highlights the specific choices made for ambiguous facts, and the body shows the derivation of the entailed atom `living_in(ryan, kgp)` or the contradiction that arises.

365 From this, it is unclear whether Dixon is Paul's father or uncle. To reflect such real-world uncertainty,
 366 NoRA includes ambiguous story-facts encoded in ASP using *cardinality facts* of the form `1{atom1;`
 367 `atom2; ...; atomk}u`, which indicates that the number of true atoms in the set `{atom1, atom2,`
 368 `..., atomk}` lies between 1 and `u` (both inclusive).

369 Once such ambiguous facts are introduced into a story, the resulting logic program may admit multiple
 370 *stable models*. An **entailed atom** in this setting is defined as an atom that is part of *every* stable
 371 model but is not explicitly listed as a story fact. Figure 10(a) shows an ambiguous story in NoRA
 372 that contains three ambiguous facts. These yield $2^3 = 8$ possible refinements, of which four result

(a) Story Facts in ASP Syntax

```
1{sibling_of(tim,lisa); sibling_of(tim,aby); sibling_of(tim,fin)}3.
1{living_in(lisa,kgp); living_in(lisa,rome)}1.
living_in(fin,kgp).
belongs_to_group(tim,male).
```

(b) Graph Encoding of Story Facts

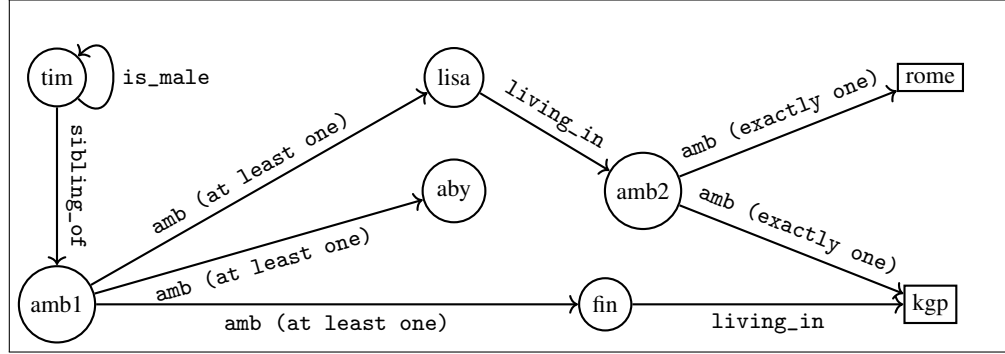


Figure 11: (a) An example story in ASP syntax with two ambiguous facts. (b) Corresponding graph encoding: ambiguous facts are handled via auxiliary nodes with labeled ambiguity constraints on edges.

373 in contradictions, leaving four consistent stable models. A common atom across all four models is
 374 `living_in(ryan, kgp)`, which is thus considered an entailed atom and may be used to construct a
 375 dataset example instance.

376 Ambiguity introduces a new notion of difficulty. For the entailed atom `living_in(ryan, kgp)`,
 377 Figure 10(b) shows eight refinements (i–viii), of which v–viii lead to contradictions and share the
 378 same structure. Among the positive refinements, refinement i and ii yield identical derivations, as do
 379 iii and iv. Intuitively **reasoning width** for a query is the sum of:

- 380 • the number of distinct derivations/proofs that yield the entailed atom across all stable models,
 381 and
- 382 • the number of distinct derivations/proofs that lead to contradiction in the remaining refine-
 383 ments.

384 For the example in Figure 10 (with story facts in 2a and the entailed atom `living_in(ryan, kgp)`),
 385 this number is 3. A formal definition of **reasoning width** is provided in the main text.

386 In the stories of NoRA, only specific types of ambiguous facts are used. These follow the ASP
 387 cardinality format:

388 $l\{\text{atom1}; \text{atom2}; \dots; \text{atomk}\}u$

389 where $k \in \{2, 3\}$, and either $u = l = 1$ (meaning *exactly one* atom is true), or $l = 1$ and $u = k$
 390 (meaning *at least one* atom is true). The atoms used in such ambiguous facts are of the following
 391 types:

- 392 1. `living_in(a, bi)`, where a is a person and b_i are different possible locations. The same
 393 a appears in all atoms of the ambiguous fact, i.e., the ambiguity is over which location a
 394 lives in.
- 395 2. `rel(a, bi)`, where a and b_i are persons, and `rel` is a binary predicate over people (e.g.,
 396 `grandparent_of`, `sibling_of`). The same a and the same `rel` are used in all atoms of
 397 the ambiguous fact, i.e., the ambiguity is over whom a stands in relation to.

398 **Graph encoding of story facts.** When story facts are provided to a GNN, they must be converted
 399 into directed graphs. For non-ambiguous facts of the form $\text{rel}(a, b)$, we follow the standard
 400 convention: draw a directed edge from a to b with edge label rel . Special entities like `male` in
 401 relationships such as `belongs_to_group(sam, male)` are encoded as self-loops (e.g., $\text{sam} \rightarrow \text{sam}$
 402 labeled `is_male`), since neural models using these graphs rely solely on edge labels and cannot learn
 403 from node labels.

404 For ambiguous facts, a dedicated **ambiguous node** is introduced to maintain the structure and support
 405 model interpretability. Two types of constructions are used:

- 406 • For ambiguous facts of the form $1\{\text{rel}(a, b_1); \text{rel}(a, b_2); \dots; \text{rel}(a, b_k)\}_k$, where
 407 *at least one* relation is true:

408 Add an edge from a to a newly created node p_amb_node with label rel , and
 409 edges from p_amb_node to each b_i labeled `amb`, *at least 1 is true*.

- 410 • For ambiguous facts of the form $1\{\text{rel}(a, b_1); \text{rel}(a, b_2); \dots; \text{rel}(a, b_k)\}_1$, where
 411 *exactly one* relation is true:

412 Add an edge from a to p_amb_node labeled rel , and edges from p_amb_node
 413 to each b_i labeled `amb`, *exactly 1 is true*.

414 Each ambiguous fact introduces exactly one such p_amb_node . This design allows GNN-based
 415 models to reason over ambiguous structures using only edge labels (see 11).

416 11 Stable models

417 Solving a logic program involves computing its **stable models**, which are also known as **answer sets**
 418 [Lifschitz, 2008]. First note that while we usually specify ASP programs using rules with variables,
 419 the semantics of answer sets is defined w.r.t. the grounding of such programs. A ground rule is
 420 obtained by replacing the variables in an ASP rule by constants that appear in the program. The
 421 grounding of an ASP program consists of all the possible ground rules that we can obtain from its
 422 rules. Let us now assume that P is a ground program (i.e. the grounding of an ASP program).

423 In the absence of rules without negation-as-failure, a stable model of P is a minimal set of atoms,
 424 such that:

- 425 1. If we assign `true` to every atom in the set, and `false` to all other possible atoms, then all
 426 rules in P are satisfied.
- 427 2. No strict subset of the model satisfies the above condition.

428 For rules with negation-as-failure, answer sets are defined in terms of the Gelfond-Lifschitz reduct.
 429 While we do not explicitly rely on negation-as-failure in our encoding, for ambiguous facts (see
 430 below), we use a language construct that under the hood is translated to such rules. Some of the rules
 431 then have conditions with negation-as-failure, of the form $\text{not } r(a, b)$. Such conditions are intuitively
 432 satisfied unless $r(a, b)$ can be inferred. The Gelfond-Lifschitz reduct of a logic program P w.r.t. the
 433 answer set A is the logic program P^A that we obtain as follows:

- 434 • Any rule with a condition of the form $\text{not } r(a, b)$ such that $r(a, b) \in A$ is removed from the
 435 program.
- 436 • Every condition of the form $\text{not } r(a, b)$ such that $r(a, b) \notin A$ is removed from the body of
 437 the rule in which it occurs.

438 Note that the reduct P^A no longer contains negation-as-failure. We then say that A is an answer set
 439 of P iff it is an answer set of the reduct P^A .

440 Intuitively, a stable model includes both the explicitly stated story facts and additional atoms that
 441 follow logically.

1. grandparent_of(Y,X) :- grandchild_of(X,Y).
2. grandchild_of(X,Y) :- grandparent_of(Y,X).
3. grandparent_of(X,Y) :- grandfather_of(X,Y).
4. grandparent_of(X,Y) :- grandmother_of(X,Y).
5. grandmother_of(X,Y) :- grandparent_of(X,Y), belongs_to_group(X, female).
6. grandfather_of(X,Y) :- grandparent_of(X,Y), belongs_to_group(X, male).
7. parent_of(Y,X) :- child_of(X,Y).
8. child_of(Y,X) :- parent_of(X,Y).
9. parent_of(X,Y) :- father_of(X,Y).
10. parent_of(X,Y) :- mother_of(X,Y).
11. mother_of(X,Y) :- parent_of(X,Y), belongs_to_group(X, female).
12. father_of(X,Y) :- parent_of(X,Y), belongs_to_group(X, male).
13. sibling_of(Y,X) :- sibling_of(X,Y), X != Y.
14. sibling_of(X,Y) :- brother_of(X,Y).
15. sibling_of(X,Y) :- sister_of(X,Y).
16. sister_of(X,Y) :- sibling_of(X,Y), belongs_to_group(X, female).
17. brother_of(X,Y) :- sibling_of(X,Y), belongs_to_group(X, male).
18. aunt_or_uncle_of(Y,X) :- nibling_of(X,Y).
19. aunt_or_uncle_of(X,Y) :- aunt_of(X,Y).
20. aunt_or_uncle_of(X,Y) :- uncle_of(X,Y).
21. uncle_of(X,Y) :- aunt_or_uncle_of(X,Y), belongs_to_group(X, male).
22. aunt_of(X,Y) :- aunt_or_uncle_of(X,Y), belongs_to_group(X, female).
23. nibling_of(Y,X) :- aunt_or_uncle_of(X,Y).
24. parent_in_law_of(X,Y) :- father_in_law_of(X,Y).
25. parent_in_law_of(X,Y) :- mother_in_law_of(X,Y).
26. parent_in_law_of(Y,X) :- child_in_law_of(X,Y).
27. child_in_law_of(Y,X) :- parent_in_law_of(X,Y).
28. father_in_law_of(X,Y) :- parent_in_law_of(X,Y), belongs_to_group(X, male).
29. mother_in_law_of(X,Y) :- parent_in_law_of(X,Y), belongs_to_group(X, female).
30. spouse_of(Y,X) :- spouse_of(X,Y), X != Y.
31. spouse_of(X,Y) :- husband_of(X,Y).
32. spouse_of(X,Y) :- wife_of(X,Y).
33. husband_of(X,Y) :- spouse_of(X,Y), belongs_to_group(X, male).
34. wife_of(X,Y) :- spouse_of(X,Y), belongs_to_group(X, female).
35. child_of(X,Y) :- son_of(X,Y).
36. child_of(X,Y) :- daughter_of(X,Y).
37. daughter_of(X,Y) :- child_of(X,Y), belongs_to_group(X, female).
38. son_of(X,Y) :- child_of(X,Y), belongs_to_group(X, male).
39. grandchild_of(X,Y) :- grandson_of(X,Y).
40. grandchild_of(X,Y) :- granddaughter_of(X,Y).
41. granddaughter_of(X,Y) :- grandchild_of(X,Y), belongs_to_group(X, female).
42. grandson_of(X,Y) :- grandchild_of(X,Y), belongs_to_group(X, male).
43. nibling_of(X,Y) :- nephew_of(X,Y).
44. nibling_of(X,Y) :- niece_of(X,Y).
45. nephew_of(X,Y) :- nibling_of(X,Y), belongs_to_group(X, male).
46. niece_of(X,Y) :- nibling_of(X,Y), belongs_to_group(X, female).
47. child_in_law_of(X,Y) :- son_in_law_of(X,Y).
48. child_in_law_of(X,Y) :- daughter_in_law_of(X,Y).
49. daughter_in_law_of(X,Y) :- child_in_law_of(X,Y), belongs_to_group(X, female).
50. son_in_law_of(X,Y) :- child_in_law_of(X,Y), belongs_to_group(X, male).
51. sibling_in_law_of(Y,X) :- sibling_in_law_of(X,Y), X != Y.
52. sibling_in_law_of(X,Y) :- brother_in_law_of(X,Y).
53. sibling_in_law_of(X,Y) :- sister_in_law_of(X,Y).
54. sister_in_law_of(X,Y) :- sibling_in_law_of(X,Y), belongs_to_group(X, female).
55. brother_in_law_of(X,Y) :- sibling_in_law_of(X,Y), belongs_to_group(X, male).
56. belongs_to_group(Y, female) :- sibling_of(X,Y), has_property(X, no_brothers).
57. belongs_to_group(Y, male) :- sibling_of(X,Y), has_property(X, no_sisters).
58. belongs_to_group(Y, female) :- parent_of(X,Y), has_property(X, no_sons).
59. belongs_to_group(Y, male) :- parent_of(X,Y), has_property(X, no_daughters).
60. maternal_aunt_of(X,Y) :- aunt_of(X,Y), paternal_grandparent_of(Z,Y), has_property(Z, no_daughters).
61. paternal_aunt_of(X,Y) :- aunt_of(X,Y), maternal_grandparent_of(Z,Y), has_property(Z, no_daughters).
62. maternal_uncle_of(X,Y) :- uncle_of(X,Y), paternal_grandparent_of(Z,Y), has_property(Z, no_sons).
63. paternal_uncle_of(X,Y) :- uncle_of(X,Y), maternal_grandparent_of(Z,Y), has_property(Z, no_sons).
64. maternal_aunt_or_uncle_of(X,Y) :- aunt_or_uncle_of(X,Y), has_property(X, no_brothers).
65. paternal_aunt_or_uncle_of(X,Y) :- aunt_or_uncle_of(X,Y), has_property(X, no_sisters).
66. paternal_grandparent_of(X,Y) :- grandparent_of(X,Y), has_property(X, no_daughters).
67. maternal_grandparent_of(X,Y) :- grandparent_of(X,Y), has_property(X, no_sons).
68. has_property(X, no_daughters) :- parent_of(X,Y), belongs_to_group(Y, male), has_property(Y, no_sisters).
69. has_property(X, no_sons) :- parent_of(X,Y), belongs_to_group(Y, female), has_property(Y, no_brothers).
70. has_property(Y, no_brothers) :- parent_of(X,Y), has_property(X, no_sons).
71. has_property(Y, no_sisters) :- parent_of(X,Y), has_property(X, no_daughters).
72. :- has_property(Y, no_brothers), brother_of(X,Y).
73. :- has_property(Y, no_sisters), sister_of(X,Y).

Table 2: NoRA World Rules, Page 1 (Rules 1–73)

442 12 Complete NoRA world rules (total: 284 rules)

443 References

- 444 Anirban Das, Manfred Denker, Anna Levina, and Lucia Tabacu. A monte carlo algorithm for multiple
445 stochastic integrals of stable processes. *Stochastics & Dynamics*, 22(8), 2022.
- 446 Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. 2017.

- 447 Anna Levina and Viola Priesemann. Subsampling scaling. *Nature communications*, 8(1):15140,
448 2017.
- 449 Vladimir Lifschitz. Twelve definitions of a stable model. In *International Conference on Logic*
450 *Programming*, pages 37–51. Springer, 2008.

```

74 :- has_property(Y, no_sisters), sister_of(X,Y).
75 :- has_property(Y, no_daughters), daughter_of(X,Y).
76 :- has_property(Y, no_sons), son_of(X,Y).
77 has_property(Y, no_siblings) :- has_property(Y, no_brothers), has_property(Y, no_sisters).
78 has_property(Y, no_children) :- has_property(Y, no_daughters), has_property(Y, no_sons).
79 :- sibling_in_law_of(X,Y), has_property(Y, no_siblings), has_property(X, no_siblings).
80 :- aunt_or_uncle_of(X,Y), has_property(X, no_siblings).
81 :- parent_of(X,Y), has_property(X, no_children).
82 :- grandparent_of(X,Y), has_property(X, no_children).
83 :- parent_in_law_of(X,Y), has_property(X, no_children).
84 nibling_of(Y,X) :- sibling_of(Z1,X), child_of(Y,Z1).
85 nibling_of(Y,X) :- sibling_in_law_of(Z1,X), child_of(Y,Z1).
86 parent_of(Y,X) :- sibling_of(Z1,X), parent_of(Y,Z1).
87 child_of(Y,X) :- child_of(Z1,X), sibling_of(Y,Z1).
88 aunt_or_uncle_of(Y,X) :- parent_of(Z1,X), sibling_of(Y,Z1).
89 sibling_of(Y,X) :- parent_of(Z1,X), child_of(Y,Z1), Y != X.
90 granddaughter_of(Y,X) :- child_of(Z1,X), daughter_of(Y,Z1).
91 grandson_of(Y,X) :- child_of(Z1,X), son_of(Y,Z1).
92 grandchild_of(Y,X) :- child_of(Z1,X), child_of(Y,Z1).
93 child_in_law_of(Y,X) :- child_of(Z1,X), spouse_of(Y,Z1).
94 parent_in_law_of(Y,X) :- spouse_of(Z1,X), parent_of(Y,Z1).
95 parent_of(X,Y) :- spouse_of(X,Z), parent_of(Z,Y).
96 grandparent_of(Y,X) :- parent_of(Z1,X), parent_of(Y,Z1).
97 sibling_in_law_of(Y,X) :- sibling_of(Z1,X), spouse_of(Y,Z1).
98 belongs_to_group(X, female) :- mother_of(X,Y).
99 belongs_to_group(X, female) :- grandmother_of(X,Y).
100 belongs_to_group(X, female) :- sister_of(X,Y).
101 belongs_to_group(X, female) :- aunt_of(X,Y).
102 belongs_to_group(X, female) :- wife_of(X,Y).
103 belongs_to_group(X, female) :- daughter_of(X,Y).
104 belongs_to_group(X, female) :- granddaughter_of(X,Y).
105 belongs_to_group(X, female) :- niece_of(X,Y).
106 belongs_to_group(X, female) :- daughter_in_law_of(X,Y).
107 belongs_to_group(X, female) :- sister_in_law_of(X,Y).
108 belongs_to_group(X, female) :- mother_in_law_of(X,Y).
109 belongs_to_group(X, male) :- father_of(X,Y).
110 belongs_to_group(X, male) :- grandfather_of(X,Y).
111 belongs_to_group(X, male) :- brother_of(X,Y).
112 belongs_to_group(X, male) :- uncle_of(X,Y).
113 belongs_to_group(X, male) :- husband_of(X,Y).
114 belongs_to_group(X, male) :- son_of(X,Y).
115 belongs_to_group(X, male) :- grandson_of(X,Y).
116 belongs_to_group(X, male) :- nephew_of(X,Y).
117 belongs_to_group(X, male) :- son_in_law_of(X,Y).
118 belongs_to_group(X, male) :- brother_in_law_of(X,Y).
119 belongs_to_group(X, male) :- father_in_law_of(X,Y).
120 :- aunt_or_uncle_of(Y,X), child_in_law_of(Y,X).
121 :- aunt_or_uncle_of(Y,X), child_of(Y,X).
122 :- aunt_or_uncle_of(Y,X), grandchild_of(Y,X).
123 :- aunt_or_uncle_of(Y,X), grandparent_of(Y,X).
124 :- aunt_or_uncle_of(Y,X), nibling_of(Y,X).
125 :- aunt_or_uncle_of(Y,X), parent_in_law_of(Y,X).
126 :- aunt_or_uncle_of(Y,X), parent_of(Y,X).
127 :- aunt_or_uncle_of(Y,X), sibling_in_law_of(Y,X).
128 :- aunt_or_uncle_of(Y,X), sibling_of(Y,X).
129 :- aunt_or_uncle_of(Y,X), spouse_of(Y,X).
130 :- child_in_law_of(Y,X), child_of(Y,X).
131 :- child_in_law_of(Y,X), grandchild_of(Y,X).
132 :- child_in_law_of(Y,X), grandparent_of(Y,X).
133 :- child_in_law_of(Y,X), nibling_of(Y,X).
134 :- child_in_law_of(Y,X), parent_in_law_of(Y,X).
135 :- child_in_law_of(Y,X), parent_of(Y,X).
136 :- child_in_law_of(Y,X), sibling_in_law_of(Y,X).
137 :- child_in_law_of(Y,X), sibling_of(Y,X).
138 :- child_in_law_of(Y,X), spouse_of(Y,X).
139 :- child_of(Y,X), grandchild_of(Y,X).
140 :- child_of(Y,X), grandparent_of(Y,X).
141 :- child_of(Y,X), nibling_of(Y,X).
142 :- child_of(Y,X), parent_in_law_of(Y,X).
143 :- child_of(Y,X), parent_of(Y,X).
144 :- child_of(Y,X), sibling_in_law_of(Y,X).
145 :- child_of(Y,X), sibling_of(Y,X).
146 :- child_of(Y,X), spouse_of(Y,X).

```

Table 3: NoRA world rules (rules 74–146, continued).

```

147 :- grandchild_of(Y,X), grandparent_of(Y,X).
148 :- grandchild_of(Y,X), nibbling_of(Y,X).
149 :- grandchild_of(Y,X), parent_in_law_of(Y,X).
150 :- grandchild_of(Y,X), parent_of(Y,X).
151 :- grandchild_of(Y,X), sibling_in_law_of(Y,X).
152 :- grandchild_of(Y,X), sibling_of(Y,X).
153 :- grandchild_of(Y,X), spouse_of(Y,X).
154 :- grandparent_of(Y,X), nibbling_of(Y,X).
155 :- grandparent_of(Y,X), parent_in_law_of(Y,X).
156 :- grandparent_of(Y,X), parent_of(Y,X).
157 :- grandparent_of(Y,X), sibling_in_law_of(Y,X).
158 :- grandparent_of(Y,X), sibling_of(Y,X).
159 :- grandparent_of(Y,X), spouse_of(Y,X).
160 :- nibbling_of(Y,X), parent_in_law_of(Y,X).
161 :- nibbling_of(Y,X), parent_of(Y,X).
162 :- nibbling_of(Y,X), sibling_in_law_of(Y,X).
163 :- nibbling_of(Y,X), sibling_of(Y,X).
164 :- nibbling_of(Y,X), spouse_of(Y,X).
165 :- parent_in_law_of(Y,X), parent_of(Y,X).
166 :- parent_in_law_of(Y,X), sibling_in_law_of(Y,X).
167 :- parent_in_law_of(Y,X), sibling_of(Y,X).
168 :- parent_in_law_of(Y,X), spouse_of(Y,X).
169 :- parent_of(U,X), parent_of(V,X), parent_of(W,X), U != V, W != V, U != W.
170 :- parent_of(U,X), parent_of(V,X), belongs_to_group(U,male), belongs_to_group(V,male).
171 :- parent_of(U,X), parent_of(V,X), belongs_to_group(U,female), belongs_to_group(V,female).
172 :- grandparent_of(A,X), grandparent_of(B,X), grandparent_of(C,X), grandparent_of(D,X), grandparent_of(E,X), A != B, A != C, A != D, A != E, B != C,
173 :- spouse_of(X,U), spouse_of(X,V), U != V.
174 paternal_aunt_or_uncle_of(Y,X) :- father_of(Z1,X), sibling_of(Y,Z1).
175 maternal_aunt_or_uncle_of(Y,X) :- mother_of(Z1,X), sibling_of(Y,Z1).
176 paternal_grandparent_of(Y,X) :- father_of(Z1,X), parent_of(Y,Z1).
177 maternal_grandparent_of(Y,X) :- mother_of(Z1,X), parent_of(Y,Z1).
178 paternal_aunt_or_uncle_of(X,Y) :- paternal_uncle_of(X,Y).
179 paternal_aunt_or_uncle_of(X,Y) :- paternal_aunt_of(X,Y).
180 aunt_or_uncle_of(X,Y) :- paternal_aunt_or_uncle_of(X,Y).
181 maternal_aunt_or_uncle_of(X,Y) :- maternal_uncle_of(X,Y).
182 maternal_aunt_or_uncle_of(X,Y) :- maternal_aunt_of(X,Y).
183 aunt_or_uncle_of(X,Y) :- maternal_aunt_or_uncle_of(X,Y).
184 paternal_grandparent_of(X,Y) :- paternal_grandmother_of(X,Y).
185 maternal_grandparent_of(X,Y) :- maternal_grandmother_of(X,Y).
186 maternal_grandparent_of(X,Y) :- maternal_grandfather_of(X,Y).
187 paternal_grandparent_of(X,Y) :- paternal_grandfather_of(X,Y).
188 grandparent_of(X,Y) :- maternal_grandparent_of(X,Y).
189 grandparent_of(X,Y) :- paternal_grandparent_of(X,Y).
190 paternal_grandfather_of(X,Y) :- paternal_grandparent_of(X,Y), belongs_to_group(X, male).
191 paternal_grandmother_of(X,Y) :- paternal_grandparent_of(X,Y), belongs_to_group(X, female).
192 maternal_grandfather_of(X,Y) :- maternal_grandparent_of(X,Y), belongs_to_group(X, male).
193 maternal_grandmother_of(X,Y) :- maternal_grandparent_of(X,Y), belongs_to_group(X, female).
194 :- parent_of(Y,X), sibling_in_law_of(Y,X).
195 :- parent_of(Y,X), sibling_of(Y,X).
196 :- parent_of(Y,X), spouse_of(Y,X).
197 :- sibling_in_law_of(Y,X), sibling_of(Y,X).
198 :- sibling_in_law_of(Y,X), spouse_of(Y,X).
199 :- sibling_of(Y,X), spouse_of(Y,X).
200 maternal_aunt_of(X,Y) :- maternal_aunt_or_uncle_of(X,Y), belongs_to_group(X, female).
201 maternal_uncle_of(X,Y) :- maternal_aunt_or_uncle_of(X,Y), belongs_to_group(X, male).
202 paternal_aunt_of(X,Y) :- paternal_aunt_or_uncle_of(X,Y), belongs_to_group(X, female).
203 paternal_uncle_of(X,Y) :- paternal_aunt_or_uncle_of(X,Y), belongs_to_group(X, male).
204 parent_of(Y, X) :- mother_of(X,Z1), maternal_grandparent_of(Y, Z1).
205 parent_in_law_of(Y, X) :- mother_of(X,Z1), paternal_grandparent_of(Y, Z1).
206 parent_of(Y, X) :- father_of(X,Z1), paternal_grandparent_of(Y, Z1).
207 parent_in_law_of(Y, X) :- father_of(X,Z1), maternal_grandparent_of(Y, Z1).
208 sibling_of(X, Y) :- father_of(X,Z1), paternal_aunt_or_uncle_of(Y, Z1).
209 sibling_in_law_of(X, Y) :- mother_of(X,Z1), paternal_aunt_or_uncle_of(Y, Z1).

```

Table 4: NoRA world rules (rules 147–219, continued).

```

220 sibling_of(X, Y) :- mother_of(X, Z1) , maternal_aunt_or_uncle_of(Y, Z1).
221 sibling_in_law_of(X, Y) :- father_of(X, Z1) , maternal_aunt_or_uncle_of(Y, Z1).
222 belongs_to_group(X, female) :- paternal_grandmother_of(X, Y).
223 belongs_to_group(X, female) :- maternal_grandmother_of(X, Y).
224 belongs_to_group(X, male) :- maternal_grandfather_of(X, Y).
225 belongs_to_group(X, male) :- paternal_grandfather_of(X, Y).
226 belongs_to_group(X, female) :- maternal_aunt_of(X, Y).
227 belongs_to_group(X, female) :- paternal_aunt_of(X, Y).
228 belongs_to_group(X, male) :- maternal_uncle_of(X, Y).
229 belongs_to_group(X, male) :- paternal_uncle_of(X, Y).
230 :- maternal_grandparent_of(X, Y), paternal_grandparent_of(X, Y).
231 :- maternal_aunt_or_uncle_of(X, Y), paternal_aunt_or_uncle_of(X, Y).
232 living_in_same_place(Y, X) :- belongs_to(X, underage), parent_of(Y, X).
233 living_in(Y, Z) :- living_in_same_place(X, Y), living_in(X, Z).
234 living_in_same_place(X, Y) :- living_in_same_place(Y, X).
235 living_in_same_place(Y, X) :- school_mates_with(Y, X).
236 living_in_same_place(Y, X) :- colleague_of(Y, X).
237 colleague_of(X, Y) :- colleague_of(Y, X).
238 colleague_of(X, Y) :- colleague_of(X, Z), colleague_of(Z, Y).
239 school_mates_with(X, Y) :- school_mates_with(Y, X), Y != X.
240 school_mates_with(X, Y) :- school_mates_with(X, Z), school_mates_with(Z, Y), Y != X.
241 :- belongs_to(X, underage), spouse_of(X, Y).
242 :- belongs_to(X, underage), parent_of(X, Y).
243 :- belongs_to(X, underage), grandparent_of(X, Y).
244 :- colleague_of(X, Y), belongs_to(X, underage).
245 :- is_person(X), is_place(X).
246 :- is_person(X), is_gender(X).
247 :- is_person(X), is_agegroup(X).
248 :- is_person(X), is_property(X).
249 :- is_place(X), is_gender(X).
250 :- is_place(X), is_agegroup(X).
251 :- is_place(X), is_property(X).
252 :- is_gender(X), is_agegroup(X).
253 :- is_gender(X), is_property(X).
254 :- is_agegroup(X), is_property(X).
255 is_agegroup(Y) :- belongs_to(X, Y).
256 is_person(X) :- belongs_to(X, underage).
257 is_person(X) :- living_in_same_place(X, Y).
258 is_person(Y) :- living_in_same_place(X, Y).
259 is_person(X) :- school_mates_with(X, Y).
260 is_person(Y) :- school_mates_with(X, Y).
261 is_person(X) :- colleague_of(X, Y).
262 is_person(Y) :- colleague_of(X, Y).
263 is_gender(X) :- belongs_to_group(Y, X).
264 is_person(Y) :- belongs_to_group(Y, X).
265 :- belongs_to_group(X, female), belongs_to_group(X, male).
266 is_place(Z) :- living_in(X, Z).
267 is_person(X) :- living_in(X, Z).
268 :- living_in(X, V), living_in(X, U), U != V.
269 :- is_person(X), is_place(X).
270 :- is_person(X), is_gender(X).
271 :- is_place(X), is_gender(X).
272 is_person(X) :- aunt_or_uncle_of(X, Y).
273 is_person(Y) :- aunt_or_uncle_of(X, Y).
274 is_person(X) :- parent_of(X, Y).
275 is_person(Y) :- parent_of(X, Y).
276 is_person(X) :- grandparent_of(X, Y).
277 is_person(Y) :- grandparent_of(X, Y).
278 is_person(X) :- parent_in_law_of(X, Y).
279 is_person(Y) :- parent_in_law_of(X, Y).
280 is_person(X) :- sibling_of(X, Y).
281 is_person(Y) :- sibling_of(X, Y).
282 is_person(X) :- spouse_of(X, Y).
283 is_person(Y) :- spouse_of(X, Y).
284 Not_living_in(X, V) :- living_in(X, U), U != V.

```

Table 5: NoRA world rules (rules 220–284, final).