A Appendix

A.1 Example from PRONTOQA-OOD

```
Q: Everything that is a lorpus, a brimpus, or a jompus is a shumpus.  Every
wumpus is a vumpus and a sterpus and a brimpus.  Everything that is a
vumpus, a grimpus, or a brimpus is a lempus.  Everything that is a lempus
or a jompus or a lorpus is a dumpus.  Vumpuses are rompuses.  Every sterpus
is a gorpus.  Everything that is a vumpus, a grimpus, or a brimpus is a
dumpus.  Wumpuses are shumpuses.  Polly is a rompus.  Polly is a wumpus.
Prove:  Polly is a lempus or an impus or a yumpus.

A: Polly is a wumpus.  Every wumpus is a vumpus and a sterpus and a brimpus.
Polly is a vumpus and a sterpus and a brimpus.  Polly is a brimpus.  Polly
is a vumpus, a grimpus, or a brimpus.  Everything that is a vumpus, a
grimpus, or a brimpus is a lempus.  Polly is a lempus.  Polly is a lempus,
an impus, or a yumpus.
```

FIGURE 11: An example from PRONTOQA-OOD. This is a compositional example with a min depth of 4 and 3 rule types. The given answer is the expected chain-of-thought. The question is shown in `blue`, the query in `red`, and the chain-of-thought/answer in `green`.

A.2 Generative process details

In this section, we describe the process to generate examples of each deduction rule.

Implication elimination (i.e. *modus ponens*) Given $f(c)$ and $\forall x(f(x) \to g(x))$, prove $g(c)$. These are the examples in the original PRONTOQA. We follow the same process here:

1. Generate an ontology. For simplicity, we generate linear ontologies, consisting of a collection of concepts, as well as subtype-supertype relations between those concepts (i.e. concept f is a subtype of the supertype g if every instance of f is an instance of g). For simplicity, we limit each type to have at most one supertype.
2. Perform a random walk of length k from a randomly selected start vertex, where k is the desired proof depth.
3. Traverse the edges of the ontology and convert each into a sentence of the question.
4. Convert each step of the random walk into a sentence of the gold chain-of-thought.

Note that this process allows us to generate proofs of any depth, but the width is fixed to 1.

Conjunction introduction Given A and B, prove $A \land B$. The generative process is a modified version of that for implication elimination. Instead of generating rules of the form $\forall x(f(x) \to g(x))$, we generate rules of the form $\forall x(f_1(x) \land \ldots \land f_n(x) \to g(x))$, where n is the proof width. Given, $f_1(c)$, ..., and $f_n(c)$, the model must first prove $f_1(c) \land \ldots \land f_n(c)$ before applying implication elimination to prove $g(c)$. To increase the depth of the proof, $g(c)$ itself can be part of a conjunct in the antecedent of another rule.

Conjunction elimination Given $A \land B$, prove A. These examples are identical to those in conjunction introduction, except the conjunction appears in the consequent of each rule, rather than in the antecedent: $\forall x(f(x) \to g_1(x) \land \ldots \land g_n(x))$ where n is the proof width.

Disjunction introduction Given A, prove $A \lor B$. These examples are identical to those in conjunction introduction, except the conjunction is replaced with disjunction: $\forall x(f_1(x) \lor \ldots \lor f_n(x) \to g(x))$ where n is the proof width. But note that grounded axioms are not necessary for every disjunct: To apply the rule $\forall x(f_1(x) \lor \ldots \lor f_n(x) \to g(x))$, knowing $f_n(c)$ is sufficient, and we do not need to generate grounded axioms for the other disjuncts $f_i(c)$ for $i < n$.

Disjunction elimination (i.e. *proof by cases*) Given $A_1 \lor \ldots \lor A_n$, and $A_i \vdash C$ for all i, prove C. Here, n is the proof width. While it is possible to construct proofs containing multiple nested applications of disjunction elimination, such proofs are quite complex, even for humans to understand, and so we fix the depth of these examples to 1. To generate an example, we first generate the disjunction: $f_1(c) \lor \ldots \lor f_n(c)$. Next, generate the rules for each case: $\forall x(f_i(x) \to g(x))$ for all i. The goal is to prove $g(c)$.

Proof by contradiction Given $A \vdash B$ and $\neg B$, prove $\neg A$. Note that this is a rule composed of two natural deduction rules: negation elimination and introduction. But since those individual rules do not lend themselves to a natural text representation, we choose to study their composition. Similar to disjunction elimination, it is possible to construct proofs containing multiple nested applications of proof by contradiction, but such proofs are unnaturally complex. So we fix the depth to 1. To generate an example, we first generate an axiom $\neg g(c)$. Next, for each subproof, we generate a rule $\forall x(f_1(x) \lor \ldots \lor f_n(x) \to g(x))$, where n is the proof width. The goal is to prove $\neg f_1(c) \land \ldots \land \neg f_n(c)$. Note that in addition to proof by contradiction, this proof requires disjunction introduction, implication elimination, and conjunction introduction.

Note that the above list constitutes a complete set of deduction rules from propositional natural deduction, save for one rule: implication introduction. However, it is unclear how to construct an example with this deduction rule where its difficulty can be controlled by increasing the width or depth of the proof (e.g. how can a statement of the form $A_1 \to A_2 \to \ldots \to A_n$ be expressed in natural language?).

A.3 Generating compositional proofs

We use a simple recursive procedure to generate compositional proofs: (1) select a deduction rule uniformly at random, (2) select the premises for the selected rule, (3) recursively generate a subproof for each premise. A consistency checking step is required to make sure we avoid generating contradictory axioms.[6] In addition, we avoid generating an elimination rule directly following an introduction rule (or vice versa).[7] See Algorithm 1 in the appendix for pseudocode of this procedure. To test compositional proofs of various sizes, we implement a parameter that controls the minimum depth of the proof tree, and another parameter that controls the number of distinct rule types in each proof.

A.4 Further details on evaluation of CoT

We aim to test whether LLMs are able to use deduction rules OOD, where the rules do not appear in the in-context examples, and we take care not to be overly strict. For example, we wish to avoid penalizing the model for formatting differences, so long as the reasoning is correct. To this end, in determining whether a logical form follows from previous logical forms, we consider any deduction rule listed in Table 2. We also allow for two additional rules: (1) given $\forall x(f(x) \to g(x))$ and $\forall x(g(x) \to h(x))$ conclude $\forall x(f(x) \to h(x))$, and (2) given $\forall x(f(x) \to g(x))$ and $\neg g(c)$ conclude $\neg f(c)$ (i.e. modus tollens).[8] Additionally, we are flexible with respect to the ordering of conjuncts and disjuncts. For example, given the previous steps $f(a) \land g(a)$ and $\forall x(g(x) \land f(x) \to u(x) \lor v(x))$, we consider $v(a) \lor u(a)$ to be valid.

A.5 Generating distractors

Implication elimination For any rule $\forall x(f(x) \to g(x))$ in the gold proof, we generate a distractor rule $\forall x(f(x) \to h(x))$ where the concept h is a distractor and is not helpful in completing the proof. In addition, for any ground logical form in the gold proof $f(c)$, we generate a distractor logical form $h(c)$ as well as a rule $\forall x(h(x) \to h'(x))$. Note that the original PRONTOQA only adds a single distractor, whereas we add multiple, one for each hop in the proof.

Conjunction introduction Similar to those in implication elimination. For any rule $\forall x(f_1(x) \land \ldots \land f_n(x) \to g(x))$, we generate a rule of the form $\forall x(h_1(x) \land \ldots \land h_{n-1}(x) \land f_n(x) \to g(x))$ where

[6] An example is: Suppose we select conjunction introduction as the first rule; next, we recursively generate the proof of each conjunct; suppose for each of these, we choose to generate the axioms `cat(alex)` and $\neg$`cat(alex)`.

[7] In the following example, a conjunction introduction step immediately follows a conjunction elimination step: "Jay is a cat and orange. Jay is a cat. Jay is orange. Jay is a cat and orange."

[8] Analogous to the *broadly-valid* steps in PRONTOQA.

Algorithm 1: Pseudocode for generating examples of compositional proofs in PRONTOQA-OOD. In this algorithm, Ω denotes the set of all logical forms. The function `generate_compositional_proof` is initially called with parameters Ω, $\varnothing$, d, e, and `false`, where d is the requested depth and e is a randomly selected entity name (e.g. `alex`, `fae`, etc). `sample` is a helper function that, given an input set of logical forms S and an entity e, returns `sample_uniform`($\{$set of logical forms in S with minimal depth where all atoms are of the form $t(e)$ where t is a predicate$\}$).

```
function generate_compositional_proof(set of possible conclusions (logical forms) C,
                                      disallowed deduction rules R,
                                      requested depth d,
                                      ground entity e,
                                      is proof hypothetical h)
  initialize A as the set of all deduction rules excluding those in R
  /* filter deduction rules such that: (1) an element of C can be a conclusion
     of the rule, (2) for which we have sufficient depth, and (3) we don't create
     overly complex logical forms */
  if C does not contain a conjunction
    set A = A \ {conjunction_introduction}
  if C does not contain a disjunction
    set A = A \ {disjunction_introduction}
  if h = true or d = 1 or C does not contain a negation
    set A = A \ {proof_by_contradiction}
  if h = true or d = 1 or C contains only conjunctions or only disjunctions
    set A = A \ {disjunction_elimination}
  if C contains only conjunctions or only disjunctions
    set A = A \ {conjunction_elimination}
  if C contains only conjunctions or only disjunctions and any operand is negated
    set A = A \ {implication_elimination}
  if d = 0 or C contains a singleton logical form or A = ∅
    return axiom step with conclusion given by sample(C, e)
  r = sample_uniform(A)
  if r = implication_elimination
    do
      for any c ∈ C, a and c share any operands or negations of operands
    while a = generate_compositional_proof(Ω, ∅, d − 1, e, h)
    do
      a and s do not share any operands or negations of operands
    while s = sample(C, e)
    return implication_elimination with premises a and ∀x(a[e → x] → s[e → x])
  else if r = conjunction_introduction
    initialize P as an empty list, and i = 0
    L = |C| if C contains only conjunctions, else L = 3
    do
      let C_i = i^th operand of C if C contains only conjunctions, else C_i = Ω
      a = generate_compositional_proof(C_i, {conjunction_elimination}, d − 1, e, h)
      if a is atomic and a is not any other operand of C
        append a to P
        i = i + 1
    while i < L
    return conjunction_introduction with premises P
  else if r = conjunction_elimination
    let C′ be the set of conjunctions of length 3, i = sample_uniform({1, 2, 3})
    C′ = {c ∈ C′ : the i^th operand of c′ is in C}
    do
      a = generate_compositional_proof(C′, {conjunction_introduction}, d − 1, e, h)
    while a has no duplicate operands, and each operand of a is not itself a conjunction or disjunction
    return conjunction_elimination with premise a and conclusion given by the i^th operand of a
```

h_i are distractor concepts. Grounded distractor conjuncts are also generated as axioms $h_i(c)$, so that, given $f_n(c)$, both the gold rule and distractor rule are valid proof steps.

Conjunction elimination Distractors are generated similarly to the conjunction introduction case.

Algorithm 1: (continued from previous page)

else if $r =$ *disjunction_introduction*
  if $C = \Omega$ let C be the set of disjunctions of length 3
  $i =$ `sample_uniform`(number of disjuncts in C)
  do
    let $C_i = i^{th}$ operand of C
    $a =$ `generate_compositional_proof`$(C_i, \{$`disjunction_elimination`$\}, d-1, e, h)$
  while *a is atomic and a is not any other operand of C*
  replace i^{th} operand of C with a
  do
    $x =$ `sample`(C, e)
  while *i^{th} disjunct of x is distinct from all other disjuncts*
  return *disjunction_introduction with premise given by the i^{th} operand of x and conclusion x*
else if $r =$ *disjunction_elimination*
  initialize P as an empty list
  while $|P| < 2$ **do**
    $p =$ `generate_compositional_proof`$(C, \{$`disjunction_introduction`$\}, d-1, e,$ `true`$)$
    if *p is not a conjunction or disjunction and p has an axiom that is not an axiom of any $q \in P$*
      append p to P
    let A_i be the set of axioms of P_i that are not axioms of P_j for $i \neq j$
    let $a_i =$ `sample_uniform`(A_i) for all i
    let a' be a disjunction with disjuncts a_i
    $a =$ `generate_compositional_proof`$(\{a'\}, \{$`disjunction_introduction`$\}, d-1, e, h)$
    return *disjunction_introduction with premises a and P_i*
else if $r =$ *proof_by_contradiction*
  let N be the set of all negated logical forms
  $a =$ `generate_compositional_proof`$(N, \{$`proof_by_contradiction`$\}, d-1, h)$
  do
    let $a = \neg s$
    $b =$ `generate_compositional_proof`$(\{s\}, \{$`proof_by_contradiction`$\}, d-1, e,$ `true`$)$
  while *b has an atomic non-negated axiom that is not an axiom of a*
  $s' =$ `sample_uniform`$(\{$atomic non-negated axioms of b that are not axioms of $a\})$
  return *proof_by_contradiction with premises a and b and conclusion $\neg s'$*

Disjunction introduction Distractors are generated similarly to the conjunction introduction case.

Disjunction elimination Since this deduction step has many premises, multiple distractors are necessary to ensure the model doesn't resort to heuristics. For every rule of the form $\forall x(f_i(x) \rightarrow g(x))$, two distractor rules are generated: $\forall x(f_i(x) \rightarrow h'(x))$ and $\forall x(h_i(x) \rightarrow g(x))$. A distractor disjunction is also generated: $h''(c) \vee h_1(c) \vee \ldots \vee h_{n-1}(c)$.

Proof by contradiction As with disjunction elimination, multiple distractors are necessary here. We generate two distractor rules $\forall x(f_1(x) \vee \ldots \vee f_n(x) \rightarrow h(x))$ and $\forall x(h_1(x) \vee \ldots \vee h_n(x) \rightarrow g(x))$. We also generate the distractor axiom $\neg h'(c)$ so that the model is forced to choose between two axioms for the first step of the proof.

To avoid creating inconsistencies when generating a distractor rule, we avoid using existing predicates in the consequent of each rule.

Algorithm 2: Pseudocode for evaluating the output chain-of-thought. Here, the comparison operations between logical forms ignore the order of conjuncts if both operands are conjunctions; and similarly for disjunctions. In addition, when iterating over previous steps in the proof, we consider them in reverse order, so that more recent steps are prioritized. The helper function `negate` is defined, in order of precedence: `negate`$(\neg A) = A$, `negate`$(A \vee B) =$ `negate`$(A) \wedge$ `negate`(B), `negate`$(A \wedge B) =$ `negate`$(A) \vee$ `negate`(B), or `negate`$(A) = \neg A$.

function `evaluate_cot`(*context sentences* $Q_1, \ldots, Q_m$, *predicted chain-of-thought sentences* $C_1, \ldots, C_n$, *goal sentence* g)

  $L^g =$ `semantic_parse`(g) `/* parse the goal */`
  for $i \in 1, \ldots, m$ **do** `/* parse the context */`
    $L_i^Q =$ `semantic_parse`(Q_i)
  for $i \in 1, \ldots, m$ **do** `/* parse the predicted chain-of-thought */`
    $L_i^C =$ `semantic_parse`(C_i)
  initialize S as an empty set, and H as an empty map
  for $i \in 1, \ldots, n$ **do** `/* reconstruct the proof from the chain-of-thought */`
    if L_i^C *indicates 'this is a contradiction'*
      if *negate*$(L_{i+1}^C) \in H(L_{i-1}^C)$
        $(P, D, k) = (\{L_{i-1}^C,$ `negate`$(L_{i+1}^C)\}, \{$`negate`$(L_{i+1}^C)\}, 1)$
      else continue
    else
      $(P, D, k) =$ `is_provable`$(L_i^C, \{L_1^Q, \ldots, L_m^Q\}, S, H)$
    set $H(L_i^C) = \bigcup_{p \in P} H(p) \setminus D$
    if $k \geq 0$
      add L_i^C to S
  return $L^g \in S$ `/* the proof is correct if the final conclusion is provable */`

function `is_provable`(*logical form* φ, *set of axioms* A, *previous conclusions* S, *hypothesis map* H)

  if $\varphi \in A$
    return $(\{\varphi\}, 1)$ `/* this is an axiom */`
  else if φ *is a conjunction or disjunction*
    initialize P' as an empty list, and $k' = 0$
    for φ_i *operand in* φ **do**
      $(P, k) =$ `is_provable`(φ_i, A, S, H)
      if φ *is a conjunction*
        if $k \geq 0$ *and the step immediately preceding* φ *in the proof is in* P
          append P to P'
          set $k' = k' + k$
        else break
      else if $k > 0$ *and* φ *is a disjunction*
        return $(P, \varnothing, k + 1)$ `/* provable by disjunction introduction */`
    if P' *has the same size as* φ *has operands*
      return $(\bigcup P', \varnothing, k')$ `/* provable by conjunction introduction */`
  for $a \in S \cup A$ **do**
    if a *is a conjunction and* $\varphi = a_i$ *for some* i
      return $(\{a\}, \varnothing, 1 + \mathbb{1}\{a \in A\})$ `/* provable by conjunction elimination */`
    else if a *has form* $\forall x(\psi \to \gamma)$ *where* $\gamma[x \mapsto c] = \varphi$
      $(P, k) =$ `is_provable`$(\psi[x \mapsto c], A, S, H)$
      if $k \geq 0$ *and the step immediately preceding* φ *in the proof is in* $P \cup \{a\}$
        return $(P \cup \{a\}, \varnothing, k + \mathbb{1}\{a \in A\})$ `/* provable by conjunction elimination */`
  for $s \in S$ *where* s *is a disjunction* **do**
    if *for all disjuncts* s_i*, there is a* $s_j \in S$ *such that* $s_j = \varphi$ *and* $s_i \in H(s_j)$
      return $(\{s_j\}, \{s_i\}, 1)$ `/* provable by disjunction elimination */`

Algorithm 2: (continued from previous page)

for $a \in S \cup A$ **do**
    if *a has form $\forall x(\psi \to \gamma)$ where $\gamma[x \mapsto c] = \varphi$*
        $(P, k) =$ `is_provable(`$\psi[x \mapsto c], A, S, H$`)`
        if *$k \geq 0$ and the step immediately preceding φ in the proof is in $P \cup \{a\}$*
            return $(P \cup \{a\}, \varnothing, k + \mathbb{1}\{a \in A\})$ `/* provable by implication elimination */`
    else if *a has form $\forall x(\psi \to \gamma)$ where* `negate(`$\psi[x \mapsto c]$`)` $= \varphi$
        $(P, k) =$ `is_provable(negate(`$\gamma[x \mapsto c]$`)`$, A, S, H$`)`
        if *$k \geq 0$ and the step immediately preceding φ in the proof is in $P \cup \{a\}$*
            `/* provable with additional deduction rules (modus tollens) */`
            return $(P \cup \{a\}, \varnothing, k + \mathbb{1}\{a \in A\})$

if $\varphi \in S$
    return $(\{\varphi\}, \varnothing, 0)$ `/* proved by previous step */`
else if *φ has form $\forall x(\psi \to \gamma)$*
    `/* note: we precompute this graph */`
    let G be the graph where for any axiom in A with form $\forall x(\alpha \to \beta)$, α and β are vertices and there is a directed edge from α to β
    if *there is a path in G from ψ to γ*
        `/* provable with additional deduction rules */`
        return (*axioms corresponding to path edges* , $\varnothing$, *length of path*)
return $(\varnothing, \varnothing, -1)$ `/* this step is not provable (i.e., invalid) */`