Equitable Access to Justice: Logical LLMs Show Promise

Manuj Kant*
kantmanuj@gmail.com

Marzieh Nabi*
mnabi@paxai.co

Manav Kant*
mkant@caltech.edu

Preston Carlson
pjames27@stanford.edu

Megan Ma
meganma@law.stanford.edu

Abstract

The costs and complexity of the American judicial system limit access to legal solutions for many Americans. Large language models (LLMs) hold great potential to improve access to justice. However, a major challenge in applying AI and LLMs in legal contexts, where consistency and reliability are crucial, is the need for System 2 reasoning. In this paper, we explore the integration of LLMs with logic programming to enhance their ability to reason, bringing their strategic capabilities closer to that of a skilled lawyer. Our objective is to translate laws and contracts into logic programs that can be applied to specific legal cases, with a focus on insurance contracts. We demonstrate that while GPT-4o fails to encode a simple health insurance contract into logical code, the recently released OpenAI o1-preview model succeeds, exemplifying how LLMs with advanced System 2 reasoning capabilities can expand access to justice.

1 Introduction

Access to legal solutions has become increasingly limited across the low, middle, and upper-middle classes, with all facing significant barriers. More than 75% of litigants represent themselves [1], with California alone reporting over 4.3 million self-represented litigants [2]. This trend is largely attributed to the high cost of legal services and widespread distrust of attorneys, as indicated by surveys conducted by the California judicial system [3]. Addressing these challenges requires the development of reliable and transparent technological solutions to bridge the considerable gaps in the legal system for consumers.

Recently, legal applications have garnered significant attention as a promising use case for LLMs. Several scientific studies and business initiatives have highlighted both the potential and limitations of LLMs in the legal domain [4]. Considerable progress is still required before these technologies can deliver consistent and transparent solutions. While human lawyers can articulate the reasoning behind their decisions and strategies, LLMs currently lack this capability to a sufficient degree. [5, 6, 7, 8, 9, 10].

Before discussing which AI solutions might be appropriate for legal applications, it is crucial to first consider whether law is inherently deterministic. While laws provide a structured framework that can seem deterministic, the human element, interpretation, judgment, and discretion introduce a degree of uncertainty, making the law not entirely deterministic in practice. In summary, the deterministic aspects of law includes a) legal rules and statutes; and b) case precedents. The non-deterministic aspects are a) judicial interpretation; b) human judgment; and c) equity and fairness.

Given the multifaceted nature of legal practices, we propose that a combination of probabilistic and deterministic AI solutions is required to effectively address legal planning and reasoning. This raises the next logical question: which AI algorithms and relational frameworks are most suitable for developing reliable legal assistance? In the following sections of this article, we outline our current

*Equal contribution.

38th Conference on Neural Information Processing Systems (NeurIPS 2024).

approach to addressing these challenges, supported by an experiment that illustrate our findings in the context of contracts. We will then discuss our broader strategy and the future directions we intend to explore.

2 Our Current Approach

LLMs are great probabilistic solutions with rapid improvements in their capabilities. However, given their inherent probabilistic nature, there is always a chance of hallucination and inconsistent answers. On the other hand, we have logic programs with highly consistent responses and explainable answers. But their main limitation lies in their inherent lack of flexibility and scalability for handling certain complex tasks, especially in real-world applications as they struggle to model uncertainty, probabilistic reasoning, or temporal dynamics.

To leverage the strengths of both LLMs and logic programming, we explore various hybrid approaches that combine these two methodologies. In one such approach, LLMs are employed to automatically generate logical representations of legal statutes or rules. Once these representations are constructed, the specific details of a given case can be applied to this logic-based framework. This allows for a structured reasoning process, where the law's application to individual cases is derived through formal logic, thereby enhancing the interpretability and precision of legal decision-making.

The integration of LLMs with logic programming (neuro-symbolic AI) is becoming increasingly popular. AlphaGeometry [11] is a great example of the new horizons achievable by leveraging the strengths of each method. In [12], the authors proposed a neuro-symbolic approach, leveraging LLMs to generate logical representations of problems, with Prolog handling the deductive reasoning.

2.1 Limitations of Our Current Approach

In our current approach, we leverage LLMs to generate logical representations from legal texts. LLMs offer a significant advantage in developing these representations at scale, enabling the efficient processing of vast and complex legal corpora. However, the accuracy and quality of the logic produced by LLMs remain a critical concern, as these models can misinterpret legal terms, omit critical details, generate logical inconsistencies, or overgeneralize legal principles. Additionally, LLMs may struggle with nuances, ambiguities, and the conditional or temporal relationships inherent in legal texts, leading to potential errors. Moreover, potential biases in their original training data can further compromise the validity of the generated logic. Therefore, it is essential to implement robust mechanisms to prevent these types of errors and mitigate the potential negative impact of LLMs. Ensuring the integrity of the generated logic is crucial for maintaining the reliability and trustworthiness of our proposed approach.

Encouragingly, however, we have found that the quality of the logical representations generated by LLMs is significantly improving as these models become more powerful and sophisticated. In the Experiment section, we demonstrate the quality differences between two of the most recent OpenAI models, GPT-4o and OpenAI o1-preview, specifically in generating Prolog representations of certain legal contracts. Another important mechanism to ensure the accuracy of these logical representations is incorporating human feedback. To achieve this, we propose having expert attorneys, familiar with the specific legal domains, review the generated logic to validate and further enhance its quality.

3 Experiment - Hospital Cash Benefit Policy

In our experiments, we focus on legal contracts, particularly the challenges consumers face in understanding health insurance coverage. A June 2024 Stanford survey revealed that 83% of participants used traditional methods to check their insurance policy, with 82% finding the process frustrating. Computational law experts highlight the importance of "computable contracts" [14] to reduce confusion and help identify coverage gaps.

Computable Contracts: Ideally, insurance contracts would be represented as interpretable computer programs (computable contracts), which could be easily audited by regulatory bodies and legal/domain experts. These programs would allow consumers to check their coverage through a simple command rather than poring over complex documents. Logic programming languages like Prolog, which exhibit logical reasoning, offer both interpretability and automation.

However, manually encoding contracts into logic programs is time-consuming and not scalable. We demonstrate how LLMs can assist in scaling this encoding process. Specifically, the recent OpenAI o1-preview model, with advanced System 2 reasoning capabilities, significantly outperforms its predecessor, GPT-4o, in encoding policies into logic programs. In essence, we show that advanced System 2 reasoning in LLMs enables the creation of logic programs with similarly advanced reasoning abilities.

Approach - We prompted both GPT-4o and OpenAI o1-preview (see Appendix A.2 for the prompt) to translate a simplified version of the Chubb Hospital Cash Benefit insurance policy [15] (Appendix A.1) into logical rules in Prolog. The prompt provided no hints or guidelines regarding code structure, but provided a couple of baseline assumptions and clarified that the code should answer questions about whether a given claim is covered under the policy. Note that all prompts to *both* GPT-4o and OpenAI o1-preview in this paper were made with the default parameters for o1-preview (top_p = 1, temperature = 1, n = 1, presence_penalty = 0, frequency_penalty = 0).

Graphical representations of the code output generated by each model are shown in Figure 1 and Figure 2. Both models structured the code with a single root node, claim_covered, determining whether a claim is covered.

The comparison between the two encodings focuses on the condition in Section 1.3, which requires that *no later than the 7th month anniversary of the policy's effective date, written confirmation from the medical provider regarding a wellness visit within 6 months must be supplied.* Additionally, for the policy to remain in effect, *the condition in Section 1.3 must either be pending or satisfied in a timely manner.*

This means that, within 7 months of the policy start date, the insuree must provide confirmation of a wellness visit made within the first 6 months. However, if 7 months have not yet passed, the condition is considered "pending," and the policy can remain active without confirmation of the wellness visit.

GPT-4o Policy Encoding Analysis - A cursory look at the leaves of GPT-4o's encoding reveals its lack of understanding of this condition, as there is simply no node checking for whether some action is taken within 7 months (although there are conditions checking for each of 6 and 12 months).

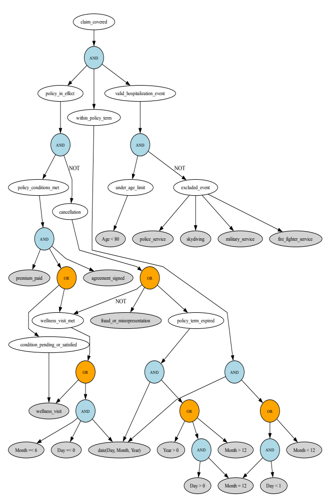

Figure 1: Prolog translation of simplified Chubb policy generated by GPT-4o. Corresponding code in Appendix A.3.1

Taking a deeper look at the mechanisms by which GPT-4o attempts to encode the wellness visit condition, we see (tracing the graph) that one of condition_pending_or_satisfied and wellness_visit_met are required for policy_in_effect. Then, condition_pending_or_satisfied is satisfied exactly when wellness_visit is. This seems to indicate that a wellness visit is required for the condition to be satisfied OR pending, which is odd since if the condition is still pending, then the wellness visit should not be required. Furthermore, wellness_visit_met can also be satisfied by wellness_visit OR if date (which presumably refers to the date of the wellness visit) is at most 6 months.

One should feel no shame in finding the above confusing, as the delineated logic is not only incorrect in several ways, but also muddled and disorganized. The "pending" part of the condition is omitted from the encoding in all but name, the wellness_visit node is redundantly used by both condition_pending_or_satisfied and wellness_visit_met, and the semantic meanings of some variables and nodes are murky at best. Even on this simplified version of a relatively simple contract, GPT-4o's encoding is a far cry from the interpretable and automated ideal of computable contracts.

OpenAI o1-preview Policy Encoding Analysis - Even at first glance, the policy encoding generated by OpenAI o1-preview looks much more organized than the one generated by GPT-4o.

Focusing again on how the model encodes the condition defined in Section 1.3, we first note that there are three relevant temporal variables in this encoding: Time, VisitTime, and ConfirmTime. These correspond with the time of hospitalization (i.e. when the claim is made), the time of the required wellness visit, and the time at which confirmation of that visit was given to the insurer, respectively. Then, tracing the graph, we see that Time $<$ 213 or satisfied_condition_1_3 for policy_in_effect, where this encoding seems to be measuring time in days.

Thus, either the time of hospitalization is within the 7 month period in which the condition defined in Section 1.3 is considered pending, or that condition must be satisfied. For the condition to be satisfied, it is required that $0 \leq$ VisitTime ≤ 183 and VisitTime $\leq$ ConfirmTime ≤ 213. Thus, the wellness visit must have occurred within 6 months and the confirmation given within 7.

The simple and organized manner in which OpenAI o1-preview encodes the above condition is a testament to its advanced reasoning capabilities.

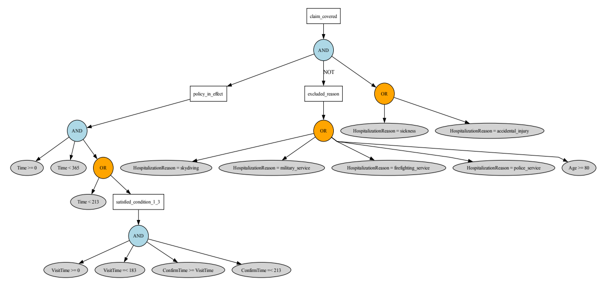

Figure 2: Prolog translation of simplified Chubb policy generated by OpenAI o1-preview. Corresponding code in Appendix A.3.2

Empirical Comparison of Policy Encoding Correctness - While our graphical representations indicate that o1-preview's encoding of the contract is more interpretable than 4o's, we also conducted an empirical measurement to compare the models' respective accuracies. We performed ten trials comparing o1-preview's and 4o's encodings of the simplified Chubb insurance policy. In each trial, both models generated their own policy encoding (Appendix A.3) from the prompt in Appendix A.2. Then, both models were prompted (Appendix A.5) to translate nine natural language yes/no questions (Appendix A.4) into Prolog queries (Appendix A.6) on their respective policy encodings. Using SWISH [16], the policy code from each model was run on the query encodings generated for that policy code, and the number of correct answers was recorded.

	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5	Trial 6	Trial 7	Trial 8	Trial 9	Trial 10	Mean
o1-preview	8	7	8	8	8	8	6	8	6	8	7.5 ± 0.23
4o	6	0	5	2	0	0	5	5	0	1	2.4 ± 0.81

Table 1: Scores of o1-preview and GPT-4o across trials and their means (Error bars calculated via calculator represent 1-sigma sample standard error over trial-to-trial variability, assuming normal errors.).

On average, 4o answered 2.4 out of 9 queries correctly, while o1-preview averaged 7.5. In four of the trials, 4o scored 0 (due to bad syntax in the policy encoding). Even when excluding these runs, 4o averaged only 4 correct answers, 3.5 fewer than o1-preview. This demonstrates that o1-preview's queries on its policy encodings yielded much more accurate results than those of 4o.

4 Our Future Approaches

We are on the cusp of an exciting era where AI can make legal solutions more accessible by applying human-like thinking, including planning and reasoning. In addition to our approach in this paper–using LLMs to generate logical representations–we explore several other potential approaches.

One approach within the realm of LLMs and logic programming is to fine-tune language models using logic-based explanations. In [9], the authors demonstrated how the "Self-Taught Reasoner (STaR)" method enhances language model reasoning through rationale generation, which provides step-by-step explanations, and rationalization, which corrects incorrect answers. This iterative process improves reasoning capabilities without the need for large annotated datasets. Applying a similar approach with logic generation could enhance the legal reasoning capabilities of language models

Our second proposal focuses on generating knowledge graph representations of legal systems, similar to the mental models experienced attorneys develop. These models integrate local and federal laws, case precedents, and relevant facts, as well as more nuanced factors. We propose using LLMs to create local knowledge graphs that mirror these mental models. Each jurisdiction would have its own tailored graph, which experts would review and refine, enabling more accurate legal decision-making.

Our third proposal suggests a dynamic programming approach to legal planning using LLMs and logic-based representations of the law. Like experienced attorneys who continuously refine legal strategies based on new information, this approach involves simulating scenarios, assessing risks, and adjusting plans iteratively. We propose digitizing this process with LLMs, generating logic-based representations of laws, statutes, and precedents [13]. Each decision will build on prior steps, allowing for continuous refinement and optimization of legal strategies in a digital environment.

References

[1] Self-Represented Litigation Network. (2021) `https://www.srln.org/node/21/about-srln#:~:text=The%20Need,SRLs?%20(SRLN%202015).`

[2] Statewide Action Plan for Serving Self-Represented Litigants. Judicial Council of California, `https://www.courts.ca.gov/documents/selfreplitsrept.pdf`

[3] Judicial Council of California. (2019), Handling Cases Involving Self-Represented Litigants, *https://www.courts.ca.gov/partners/documents/benchguide_self_rep_litigants.pdf*.

[4] Dahl, M., Magesh, V., Suzgun, M., & Ho, D. (2024) Large Legal Fictions: Profiling Legal Hallucinations in Large Language Models. *https://arxiv.org/abs/2401.01301*.

[5] Li, Z., Liu, H., Zhou D., & Ma, T. (2023) Chain of Thought Empowers Transformers to Solve Inherently Serial Problems. *https://arxiv.org/abs/2402.12875*.

[6] Lyu, Q., Havaldar, S., Stein, A., Zhang, L., Rao, D., Wong, E., Apidianaki, M., & Callison-Burch, C. (2023) Faithful Chain-of-Thought Reasoning. *https://arxiv.org/abs/2301.13379*.

[7] Besta, M., Blach, N., Kubicek1, A., Gerstenberger, R., Podstawski, M., Gianinazzi, L., Gajda, J., Lehmann, T., Niewiadomski, H., Nyczyk, P., & Hoefler, T. (2024) Graph of Thoughts: Solving Elaborate Problems with Large Language Models. *The Thirty-Eighth AAAI Conference on Artificial Intelligence*.

[8] Plaat, A., Wong, A., Verberne, S., Broekens, J., Stein, N., & Back, T. (2024) Reasoning with Large Language Models, a Survey. *https://arxiv.org/abs/2407.11511*.

[9] Eric Zelikman, Yuhuai Wu, Jesse Mu, &Goodman, N. (2022). STaR: Bootstrapping Reasoning With Reasoning. *https://arxiv.org/abs/2203.14465*.

[10] Yao, S., Yu, D., Zhao, J., Shafran, I., Griffiths, T., Cao, Y., & Narasimhan, K. (2023) Tree of Thoughts: Deliberate Problem Solving with Large Language Models. *37th Conference on Neural Information Processing Systems (NeurIPS 2023)*.

[11] Google Deep Mind. (2024) *https://deepmind.google/discover/blog/alphageometry-an-olympiad-level-ai-system-for-geometry/*

[12] Borazjanizadeh, N., & Piantadosi, S. T. (2024) Reliable Reasoning Beyond Natural Language. *https://arxiv.org/abs/2407.11373*.

[13] Dagan G., Keller, F., &Lascarides, A. (2023) Dynamic Planning with a LLM. *https://arxiv.org/abs/2308.06391*.

[14] Cummins, J., Clack, C. D. (2022). Transforming commercial contracts through computable contracting. *Journal of Strategic Contracting and Negotiation 6, 3 - 25. https://journals.sagepub.com/doi/10.1177/20555636211072560*

[15] CodeX Stanford Formation Demo. *https://codexstanford.github.io/formation-demo/*

[16] SWISH – SWI-Prolog for SHaring. *https://swish.swi-prolog.org/*

A Appendix / supplemental material

A.1 Simplified Chubb Hospital Cash Benefit Policy

Between:
CODEX INSURANCE LIMITED ("us")
and
________________ ("You")

This policy is provided on the following terms and conditions:

POLICY IN EFFECT AND CONDITIONS

1.1 The payment of any benefit under this policy is conditioned on the policy being in effect at the time of the hospitalization for sickness or accidental injury on which the claim for such benefit is premised. The policy will be in effect if:

(a) This agreement is signed,
(b) The applicable premium for the policy period has been paid, and
(c) The condition set out in Section 1.3 is still pending or has been satisfied in a timely fashion, and
(d) The policy has not been canceled.

1.2 Cancelation will be deemed to have occurred if there is fraud, or any misrepresentation or material withholding of any information provided by you to the Company in connection with any communication or information relating to this policy, or if the condition set out in Section 1.3 has not been satisfied in a timely fashion. It will also be automatically canceled at midnight, US Eastern time then in effect, on the last day of the policy term described in Section 5 below.

1.3 No later than the 7th month anniversary of the effective date of this policy, you will supply us with written confirmation from the medical provider in question of a wellness visit for yourself with a qualified medical provider occurring no later than the 6th month anniversary of the effective date of this policy.

GENERAL EXCLUSIONS

2.1 Your policy will not apply to, and no benefit will be paid with respect to, any event causing sickness or accidental injury arising directly or indirectly out of:

1. Skydiving; or
2. Service in the military; or
3. Service as a fire fighter; or
4. Service in the police; or
5. If your age at the time of the hospitalization is equal to or greater than 80 years of age.

GENERAL CONDITIONS

3.1 Where does Your Policy apply?

3.1.1 Your Policy insures You twenty-four (24) hours a day anywhere in the world.

3.2 Arbitration

3.2.1 If any dispute or disagreement arises regarding any matter pertaining to or concerning this Policy, the dispute or disagreement must be referred to arbitration in accordance with the provisions of the Arbitration Act (Cap. 10) and any statutory modification or re-enactment thereof then in force, such arbitration to be commenced within three (3) months from the day such parties are unable to settle the dispute or difference. If You fail to commence arbitration in accordance with this clause, it is agreed that any cause of action and any right to make a claim that You have or may have against Us shall be extinguished completely. Where there is a dispute or disagreement, the issuance of a valid arbitration award shall also be a condition precedent to our liability under this Policy. In no case shall

You seek to recover on this Policy before the expiration of sixty (60) days after written proof of claim has been submitted to Us in accordance with the provisions of this Policy.

3.3 Laws of New York

3.3.1 Your Policy is governed by the laws of New York.

3.4 US Currency

3.4.1 All payments by You to Us and by Us to You or someone else under your policy must be in United States currency.

3.5 Premium

3.5.1 The premium described in Section 5 below shall be paid in one lump sum at the signing of the policy.

3.6 Policy Term The term of this policy will begin on the date accepted by Us as signified by our signature of the policy (the effective date) and will last for a period of one year from that date, unless previously canceled pursuant to Section 1 above.

A.2 Prompt for Generating Policy Encodings

- Given the insurance contract below, translate the document into valid Prolog rules so that I can run a Prolog query on the code regarding whether or not some claim is covered under the policy and receive the correct answer to the question.

- Please fully define all predicates and DO NOT define any facts, only rules that can be used to answer queries on this insurance contract.

- Assume that all dates/times in any query to this code (apart from the claimant's age) will be given RELATIVE to the effective date of the policy (i.e. there will never be a need to calculate the time elapsed between two dates). Take dates RELATIVE TO the effective date into account when writing this encoding.

- Assume that the agreement has been signed and the premium has been paid (on time). There is no need to encode rules or facts for these conditions.

- Return only Prolog code in your reply. No explanation is necessary.

- Ensure that:

1. The legal text is appropriately translated into correct Prolog rules.
2. The output does not redefine, misuse, or conflict with any built-in Prolog predicates.
3. If dynamic predicates are necessary, they are declared and managed correctly.
4. All predicates used in the generated Prolog code, including those referenced in the query, are fully defined and error-free to prevent issues like "procedure does not exist."
5. Logical relationships, conditions, and dependencies in the text are faithfully represented in the Prolog rules to ensure accurate query results.

- Insurance contract: {text_content}

A.3 LLM Policy Encodings

A.3.1 GPT-4o Policy Encoding

```
% Dynamic predicates to allow for updates if necessary
:- dynamic policy_in_effect/0.
:- dynamic cancellation/0.
:- dynamic wellness_visit/0.

% Policy will be in effect if it has not been canceled and all
    conditions are met.
policy_in_effect :-
    \+ cancellation,
```

```
    policy_conditions_met.

% Conditions for the policy to be in effect
policy_conditions_met :-
    agreement_signed,
    premium_paid,
    (condition_pending_or_satisfied; wellness_visit_met).

% Policy cancellation conditions
cancellation :-
    fraud_or_misrepresentation,
    !.
cancellation :-
    \+ wellness_visit_met,
    !.
cancellation :-
    policy_term_expired.

% Condition 1.3
condition_pending_or_satisfied :- wellness_visit.

% Policy term expiration check
policy_term_expired :-
    date(Day, Month, Year),
    (Month > 12; (Month = 12, Day > 0) ; Year > 0).

% Placeholder predicates to simulate conditions
agreement_signed.
premium_paid.
fraud_or_misrepresentation :- fail. % No fraud by default

% Wellness visit check
wellness_visit_met :-
    date(Day, Month, _),
    Month =< 6,
    Day =< 0 ; % Assuming verification by certain day or prior month
        with Day <= 0
    wellness_visit.

% Check if a claim is covered
claim_covered(Accident, Hospitalization, Age) :-
    policy_in_effect,
    within_policy_term,
    valid_hospitalization_event(Accident, Age).

% Validity check for hospitalization event
valid_hospitalization_event(Accident, Age) :-
    \+ excluded_event(Accident),
    under_age_limit(Age).

% Exclusions based on events
excluded_event(skydiving).
excluded_event(military_service).
excluded_event(fire_fighter_service).
excluded_event(police_service).

% Age check
under_age_limit(Age) :-
    Age < 80.

% Check if within policy term
within_policy_term :-
    date(Day, Month, _),
    (Month < 12 ; (Month = 12, Day < 1)).
```

```
% Placeholder date predicate
date(0, 0, 0). % Assuming effective date being the 0th day of the 0th
    month

% Queries can be structured as:
% ?- claim_covered(Accident, Hospitalization, Age).
% where Accident is one of [skydiving, military_service,
    fire_fighter_service, police_service, other],
% and Age is a valid integer representing the claimant's age.
```

A.3.2 OpenAI o1-preview Policy Encoding

```
policy_in_effect(Time, VisitTime, ConfirmTime) :-
     Time >= 0,
     Time < 365,
     (Time < 213 ; satisfied_condition_1_3(VisitTime, ConfirmTime)).

satisfied_condition_1_3(VisitTime, ConfirmTime) :-
     VisitTime >= 0,
     VisitTime =< 183,
     ConfirmTime >= VisitTime,
     ConfirmTime =< 213.

claim_covered(Time, VisitTime, ConfirmTime, HospitalizationReason, Age
    ) :-
     policy_in_effect(Time, VisitTime, ConfirmTime),
     (HospitalizationReason = sickness ; HospitalizationReason =
         accidental_injury),
     \+ excluded_reason(HospitalizationReason, Age).

excluded_reason(HospitalizationReason, Age) :-
     HospitalizationReason = skydiving ;
     HospitalizationReason = military_service ;
     HospitalizationReason = firefighting_service ;
     HospitalizationReason = police_service ;
     Age >= 80.
```

A.4 Queries and Answers for Empirical Evaluation

All queries are preceded by the disclaimer: "Assuming all other conditions are met and no other exclusions apply (where by 'other,' I mean anything not referenced in the query that follows),. . . "

Query: "will the claim be covered under the policy if I was hospitalized by burns suffered while doing my duty as a firefighter?" **Answer:** "No."

Query: "will the claim be covered under the policy if I am 78 years old at the time of hospitalization?" **Answer:** "Yes."

Query: "will the claim be covered under the policy if I was hospitalized for pneumonia 5 months after the policy's effective date, and my age at the time of hospitalization is 65?" **Answer:** "Yes."

Query: "will the claim be covered under the policy if I was hospitalized due to a fall while traveling abroad and I had given confirmation of my wellness visit 8 months after the policy's effective date?" **Answer:** "No."

Query: "will the claim be covered under the policy if I was hospitalized for punching my own face to show off for my friends and I did not commit fraud or misrepresentation?" **Answer:** "No."

Query: "will the claim be covered under the policy if I was hospitalized due to an injury sustained while skydiving, my age at the time of hospitalization was 79, and proof of my wellness visit was provided 6.5 months after the policy's effective date?" **Answer:** "No."

Query: "will the claim be covered under the policy if I was hospitalized for a heart attack, proof of the wellness visit was submitted 2 months after the policy's effective date, and my age at the time of hospitalization was 75?" **Answer:** "Yes."

Query: "will the claim be covered under the policy if I was hospitalized after being injured in a military training exercise, the hospitalization occurred within the policy term, and I did not commit fraud?" **Answer:** "No."

Query: "will the claim be covered under the policy if I was hospitalized due to my son biting me in the ankle, proof of my wellness visit was provided 6 months after the effective date, and I was serving as a police officer at the time of hospitalization?" **Answer:** "Yes."

A.5 Prompt for Generating Query Encodings

- I have given below:

1. A question about whether or not the policy defined in a given insurance contract applies in a particular situation
2. The text of the insurance contract
3. A Prolog encoding of the insurance contract

- Encode the question into a Prolog query such that it can be run on the Prolog encoding of the insurance contract, returning the correct answer to the question.

- Assume that the agreement has been signed and the premium has been paid (on time). There is no need to encode rules or facts for these conditions.

- Return only Prolog code in your reply. No explanation is necessary.

- Ensure that:

1. The output does not redefine, misuse, or conflict with any built-in Prolog predicates.
2. If dynamic predicates are necessary, they are declared and managed correctly.
3. All predicates used in the generated Prolog code, including those referenced in the query, are fully defined and error-free to prevent issues like "procedure does not exist."
4. Logical relationships, conditions, and dependencies in the text are faithfully represented in the Prolog rules to ensure accurate query results.
5. No absolute dates/times (apart from the claimant's age) are encoded in your query. Only include dates/times RELATIVE to the effective date of the policy (again, except for age).
6. Set any facts/rules/parameters in the code such that ALL conditions (for the policy to apply) which are UNRELATED to the above query are satisfied.
7. Set any facts/rules/parameters in the code such that NO exclusions (which would prevent the policy from applying) which are UNRELATED to the above query are satisfied.

- Question: {query}

- Insurance contract: {text_content}

- Insurance contract Prolog encoding: {policy_encoding}

A.6 LLM Query Encodings

A.6.1 GPT-4o Query Encodings

```
Query 1 Prolog:
?- claim_covered(fire_fighter_service, hospitalization, Age).

Query 2 Prolog:
?- claim_covered(other, hospitalization, 78).
```

```
Query 3 Prolog:
?- claim_covered(other, true, 65).

Query 4 Prolog:
% Updating the current conditions for the query
:- retractall(date(_, _, _)).
:- assert(date(0, 8, 0)). % 8 months after policy effective date

% Mark health visit as complete, despite late confirmation
:- retractall(wellness_visit).
:- assert(wellness_visit).

% Formatting the query based on the claim details
?- claim_covered(other, hospitalization, 40). % Fall during travel,
    hospitalized, age is 40

Query 5 Prolog:
% Define the incident with its characteristics
incident_details(punching_own_face, hospitalized, 25). % Example age,
    assuming it's 25

% Prolog query to determine if the claim is covered
?- incident_details(Accident, Hospitalization, Age),
   claim_covered(Accident, Hospitalization, Age).

Query 6 Prolog:
% Set up the situation to reflect the claim question
date(7, 0, 0). % Assume the date for checking purposes

% Query to check if the claim is covered
?- claim_covered(skydiving, true, 79).

Query 7 Prolog:
% Prolog query for the given scenario
?- assert(date(0, 2, 0)), % Setting date to 2 months after policy
    effective date
   claim_covered(other, heart_attack, 75).

Query 8 Prolog:
?- claim_covered(military_service, Hospitalization, Age).

Query 9 Prolog:
?- claim_covered(other, hospitalization, 30).
```

A.6.2 OpenAI o1-preview Query Encodings

```
Query 1 Prolog:
policy_in_effect(Time, VisitTime, ConfirmTime) :-
    Time >= 0,
    Time < 365,
    (Time < 213 ; satisfied_condition_1_3(VisitTime, ConfirmTime)).

satisfied_condition_1_3(VisitTime, ConfirmTime) :-
    VisitTime >= 0,
    VisitTime =< 183,
    ConfirmTime >= VisitTime,
    ConfirmTime =< 213.

excluded_reason(Cause, Age) :-
    Cause = skydiving ;
    Cause = military_service ;
    Cause = firefighting_service ;
    Cause = police_service ;
```

```
    Age >= 80.

claim_covered(Time, VisitTime, ConfirmTime, HospitalizationReason,
    Cause, Age) :-
    policy_in_effect(Time, VisitTime, ConfirmTime),
    (HospitalizationReason = sickness ; HospitalizationReason =
        accidental_injury),
    \+ excluded_reason(Cause, Age).

?- claim_covered(100, 150, 160, accidental_injury,
    firefighting_service, 40).

Query 2 Prolog:
claim_covered(200, 150, 160, sickness, 78).

Query 3 Prolog:
?- claim_covered(150, 0, 0, sickness, 65).

Query 4 Prolog:
claim_covered(240, 180, 244, accidental_injury, 30).

Query 5 Prolog:
?- claim_covered(100, 0, 0, intentional_injury, 30).

Query 6 Prolog:
?- claim_covered(200, 180, 198, skydiving, 79).

Query 7 Prolog:
?- claim_covered(100, 30, 60, sickness, 75).

Query 8 Prolog:
?- claim_covered(100, 0, 0, military_service, 40).

Query 9 Prolog:
?- claim_covered(200, 180, 183, accidental_injury, 40).
```