Logical English 2: A Reimplementation

Miguel Calejo[1], Jacinto A. Dávila Quintero[1,2,†]

[1]*logicalcontracts.com*
[3]*Universidad de Los Andes, CESIMO, Mérida, Venezuela*

Abstract

Logical English (LE) is a controlled natural language for expressing legal and business rules in a form that is at once human-readable and executable. We compare two implementations of the language: the original system (LE1), developed at Imperial College and partner organisations and oriented toward research breadth across multiple reasoning back-ends and several natural-language front-ends; and a full reimplementation (LE2), undertaken by the same working group with the goal of consolidating the stable core of LE into a self-contained development environment. We summarise the language, sketch the design and scope of LE1, and describe the language-level and tooling-level changes introduced by LE2 — in particular first-class scenario testing, structured diagnostics, an integrated browser IDE with template-aware editing and interactive justification trees, an LLM assistant for authoring, a graph view of the knowledge base, and for the Model Context Protocol .

Keywords

Logical English, Controlled Natural Language, Logic Programming, Legal Tech, Prolog, Computable Contracts

1. Introduction

Logical English (LE) is a controlled natural language designed so that domain experts can write executable rules that read like ordinary English yet have a precise logical reading [1, 2]. The motivation is to reduce the distance between informal legal or business prose and the formal artefacts that machines reason with: rather than encoding a regulation in a programming language and then verifying that the encoding is faithful to the text, one would prefer the regulation itself to serve as the program. LE has been applied to tax law, contracts, citizenship rules, financial agreements, and, more recently, to a domain-specific dialect for insurance contracts [3]; the same line of work has been used to discuss the limits and opportunities of LLM-assisted authoring of financial contracts [5], and has also been proposed as a vehicle for teaching logic and computational thinking to school-age learners [2].

A program in LE consists of *templates* — natural-language patterns with typed argument slots, e.g. `*a person* is the father of *another person*` — used to express *facts*, *rules* (`Conclusion if Conditions`), *queries* and *scenarios* of test facts. Reasoning produces both answers and *justification trees* phrased in the same controlled English, allowing users to inspect how a conclusion was reached. The language has remained close to a stable core since [1]: templates with `*. . . *` slots, conjunction, disjunction, `unless`, weak and strong negation, iterative conditions (`for all cases in which . . . it is the case that . . .`), aggregates (`sum`, `count`, `min`, `max`, `average`), an ontology section for `is_a` taxonomies, and the use of `that` to embed sentences as meta-arguments.

This note compares two implementations of that language. The first, referred to here as **LE1**, is the original implementation associated with [1, 2] and developed at Imperial College and partner organisations. The second, **LE2**, is a clean reimplementation undertaken by the same working group, with the goal of making LE easier to use, easier to deploy, and easier to embed in modern toolchains. The two systems interpret essentially the same language — programs written for one will typically port to the other with small surface adjustments — but they differ substantially in scope, tooling, and intended audience.

IV Prolog Education Workshop 2026 (PEG 2026)
mc@logicalcontracts.com (M. Calejo); jacinto@ula.ve (J. A. Dávila Quintero)
https://logicalcontracts.com (M. Calejo)
0000-0003-1053-8763 (J. A. Dávila Quintero)
© 2026 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR Workshop Proceedings ceur-ws.org ISSN 1613-0073
published 2026-11-03

CEUR-WS.org/Vol-4311/PEG_regular_13.pdf

2. Logical English (LE1)

LE1 was conceived as a research artefact whose primary value was to demonstrate that a controlled natural language, as syntactic sugar for logic programming, can faithfully represent legal and contractual reasoning. Its design emphasises **breadth**: a wide range of language constructs, several reasoning back-ends, and integration with an existing notebook environment (SWISH from SWI-Prolog) for logic programming.

Form of delivery. LE1 is distributed as a SWI-Prolog package intended to be loaded into SWISH, the web-based Prolog notebook server. A LE document is typically embedded as a string inside a Prolog source file and translated on demand by predicates such as `answer/1` or `answer X with scenario Y`. Users interact with the system either through a SWISH notebook or through an external Visual Studio Code extension that communicates with a SWISH server.

Reasoning back-ends. LE1 can target three back-ends from the same LE source: pure Prolog, *TaxLog* (a Prolog dialect with explanation facilities, originally developed for tax-law applications), and *s(CASP)*, a constraint-based ASP system that supports negation, abducibles and HTML-rendered justifications. The choice of back-end is determined by a `the target language is:` declaration. This plurality made LE1 a useful platform for investigating which form of non-monotonic reasoning is best suited to a given legal artefact.

Language scope. Beyond the stable core listed in Section 1, LE1 supports several constructs that reflect its experimental character: multilingual front-ends for English, French, Italian and Spanish; *event* and *fluent* sections for temporal reasoning in the style of the Event Calculus; the form `*X* is a collection of ... where ...` for set-comprehension; multiple combined knowledge bases with simple inheritance; rule names; and prototype mechanisms for deontic operators and rule priorities. The handbook [7] and the `le_syntax.md` [8] documents describe these features in detail; several of them are flagged as experimental or "currently being tested".

Application portfolio. The repository ships an example library of contract and statute fragments encoded in LE — small business restructure rollover, net-asset-value test for capital gains, ISDA Master Agreement clauses, the OECD model tax convention, criminal-justice rules, an escrow blockchain agreement, Italian citizenship law — most of which appear, in whole or in part, in the published literature [1, 2]. **InsurLE** [3] is a more recent extension developed in the same family: a domain-specific controlled natural language for insurance contracts that adds an insurance ontology, restrictive relative clauses, and a rule-and-exception structure that mirrors the form in which insurers currently write policy documents. The aim is for the contract text and the machine-readable representation to coincide, eliminating the verification gap that arises when a separate codification is maintained alongside the prose.

Style. LE1 grew incrementally over several years. Its strength is that almost every language idea explored in the LE research line has been prototyped at least once; the corresponding cost is a sizeable code base whose features have accumulated organically and whose deployment story (SWISH plus external clients) is oriented toward researchers and pilot users rather than general developers. Some underlying problems with the file management system on SWISH, that have been hard to track and fix, may have also contributed to the cost.

3. Logical English 2

LE2 is a redesign and full reimplementation of the same language family. Its goal is not to extend LE — the language is, if anything, slightly narrower — but to consolidate the parts of LE that are robustly

useful into a single coherent system that can be installed, deployed, and extended outside a SWISH environment.

3.1. Language

LE2 carries forward the stable LE core: templates and facts, `if`-rules, `and`/`or`/`unless`/negation, iterative conditions, aggregates, ontologies, scenarios, queries, the `that` meta-argument convention, and fluents/events for temporal modelling. Three smaller language changes are worth noting.

First, LE2 treats *testing* as a first-class concept. A scenario may declare expected answers (`<query> expects answers [“. . . ”]`); these are parsed as part of the document and exercised by a built-in test runner. A LE file is, in effect, both program and test suite — a property intended to support continuous verification as the rules evolve.

Second, LE2 exposes a small **introspection API**, a set of internal structures to represent each LE document, accessible as named, Prolog system predicates: the active knowledge base, the source-level identifier of the current rule, the type taxonomy, hierarchical designators for numbered rules of the form `1.1.a`, and structured records for parsing issues and expected answers. This makes LE programs themselves describable from within and underneath LE, which can be useful for explanation, for legal cross-referencing, and for tools built on top of the language.

Third, LE2 deliberately drops or postpones some of LE1’s experimental constructs. The current implementation targets only Prolog. Other targets, such as s(CASP) and TaxLog, and also other sources, like the multilingual front-ends (Español Lógico and Italiano Logico were also prototyped on the same codebase), are not present; the `is a collection of . . . where . . .` form and some meta-reasoning, temporal and constrained reasoning prototypes are likewise absent from this iteration. These are roadmap items rather than rejected ideas, and the reimplementation includes explicit extension points (multifile hooks at the parsing, second-pass and reasoning levels) so that they — or proprietary additions — can be re-introduced without forking the core.

3.2. Tooling

A salient difference between the two systems is the **integrated development environment** included with LE2. Rather than relying on SWISH or an external editor, LE2 ships a browser-based IDE with template-aware syntax highlighting (so user-defined template words are not confused with language keywords), live error reporting against a structured diagnostic model, text hiding/unhiding by section and rule, completion of templates, hover inspection, and an integrated query and explanation panel.

The IDE is supplemented by an in-document chat-style **LE Assistant** that can drive an LLM (OpenAI, Anthropic or Google models, configured locally by the user) to explain, refactor, or generate LE code. The assistant can both produce new code from a natural-language description (Figure 1) and apply targeted edits to the document under user supervision (Figure 2). This kind of LLM-assisted authoring of legal and financial rules — and its limits — has been the subject of independent discussion in the financial-contracts literature [5, 6].

LE2 also speaks two protocols that did not have a usable form during LE1’s main development. A **Debug Adapter Protocol (DAP)** server is under development, with the intention that any DAP-compliant client (VS Code, JetBrains IDEs and others) will be able to step through an LE proof, set breakpoints, and inspect the chain of ancestor goals; the underlying tracer is operational, while the client-facing surface is still being stabilised. LE2 also implements the **Model Context Protocol (MCP) [9]**, an emerging standard for tool-using LLMs, which allows agents to load LE documents, run queries and verify rules through a typed interface. Together these protocols are intended to make LE invokable by external software — IDEs and tool-using language models alike — without bespoke glue.

In addition to the textual editor, LE2 generates a **graph view** of the loaded knowledge base (Figure 3): predicates appear as nodes and rule dependencies as edges, providing a structural overview of larger documents and a navigational complement to the source view. This is a mere pedagogical aid at this

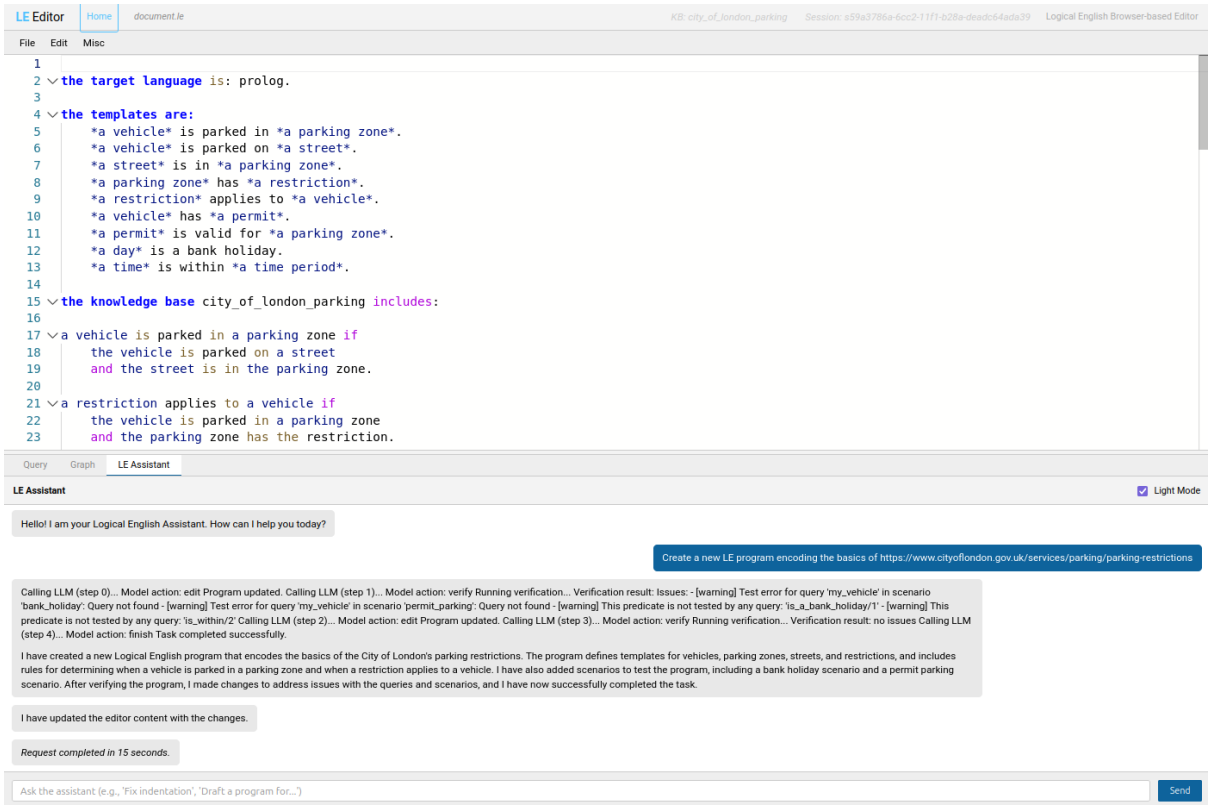


Figure 1: The LE Assistant generating a new program from a natural-language brief: Create a new LE program encoding the basics of https://www.cityoflondon.gov.uk/services/parking/parking-restrictions

point. We look forward to explore tools to extract the graph's elements and relations from the underlying texts such as [10] and [11].

3.3. Reasoning and verification

LE2's reasoner is a fresh meta-interpreter rather than a layer over an existing logic-programming substrate. The most user-visible consequence is that explanations are pervasive: every successful proof yields a natural-language justification tree, and every *failed* query yields a failure tree explaining why each candidate path did not succeed. Justification nodes carry source-range information, so the IDE can render explanations as a navigable structure: clicking a step in the tree selects the corresponding rule or clause in the editor (Figure 4). Each loaded document is associated with an isolated reasoning session, so concurrent users — and concurrent scenarios within a session — do not interfere with one another's facts.

A separate **verifier** runs at every load and produces a structured list of issues: missing templates, undefined predicates, untested predicates, rules with no variables, redefined system templates, and failed expectations. Each issue carries a description, a suggested fix and a source range, so the editor can render diagnostics inline and offer quick-fixes. LE1 reports diagnostics as text; LE2 records them as structured data, which is exposed to clients over the API.

3.4. Deployment

LE1 is run as a SWISH plug-in or via small example servers bundled with the repository. LE2 is a self-contained HTTP service: a single command brings up a JSON-over-HTTP API together with the editor as a web application, and a Docker image packages both. A hosted demonstration runs at https://le2.logicalcontracts.com. The intent is that an organisation adopting LE for an actual workflow should not need to learn SWISH first.

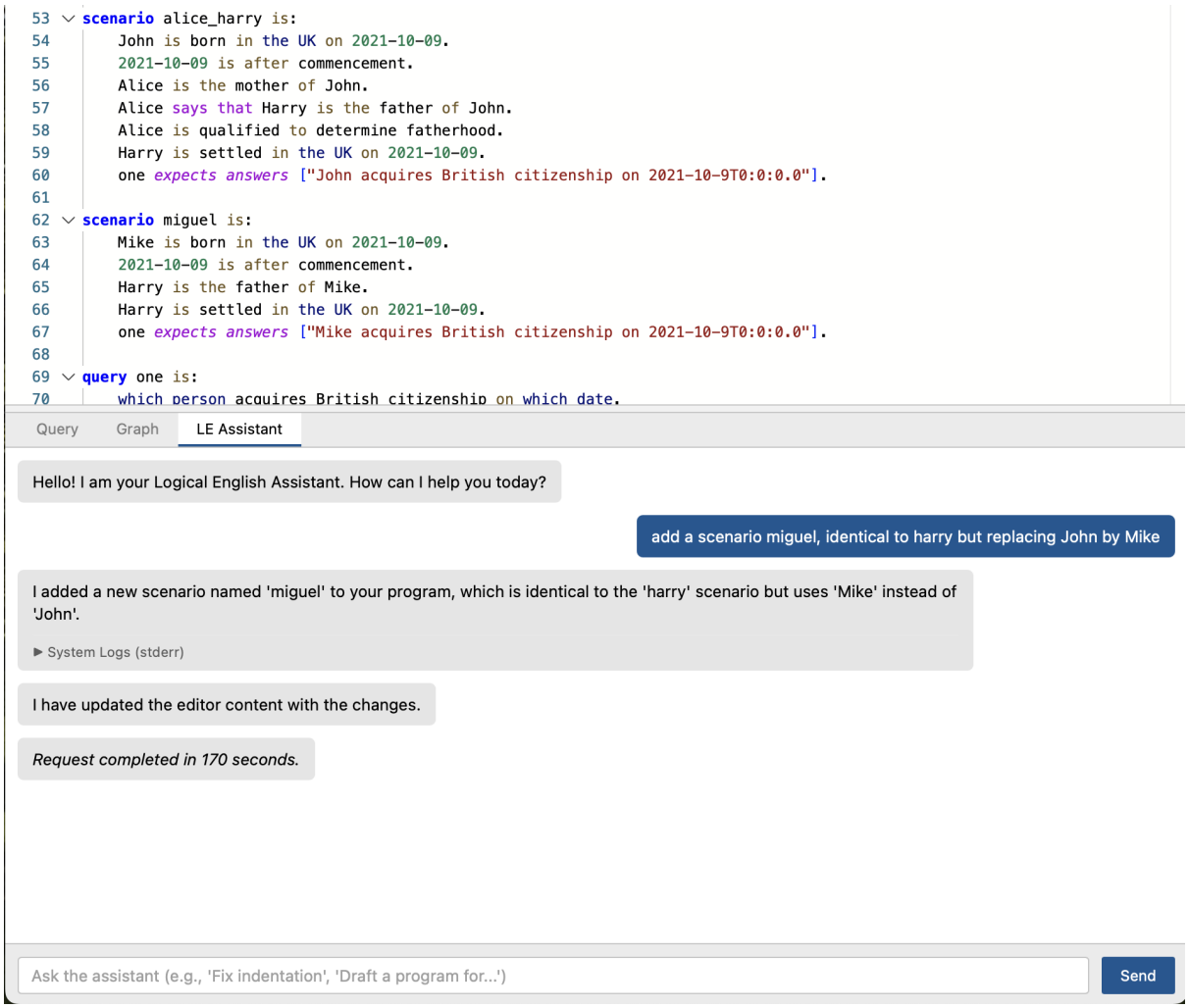


Figure 2: The LE Assistant applying targeted edits to an existing document.

3.5. What LE2 inherits and what it changes, in summary

LE2 keeps:

- the language and semantics of LE's stable core, sufficient to express the bulk of the LE1 example portfolio;
- the principle that the source text and the executable artefact should coincide;
- the use of justification trees rendered in the same controlled English as the rules.

LE2 changes:

- **Surface**: a dedicated browser IDE replaces SWISH and external extensions; explanations and diagnostics are presented as interactive structures rather than as text output.
- **Engineering**: a smaller, more modular code base built around a source-tracking pipeline that retains character offsets from token to proof tree.
- **Integration**: web, MCP and DAP endpoints make LE usable from IDEs and from LLM-based agents without bespoke glue.
- **Process**: tests are part of the language, the verifier is part of the loop, and extension points are part of the architecture.

LE2 narrows:

- to a single Prolog back-end and a single (English) front-end, with s(CASP), TaxLog, multilingual support and several experimental constructs deferred to future work.

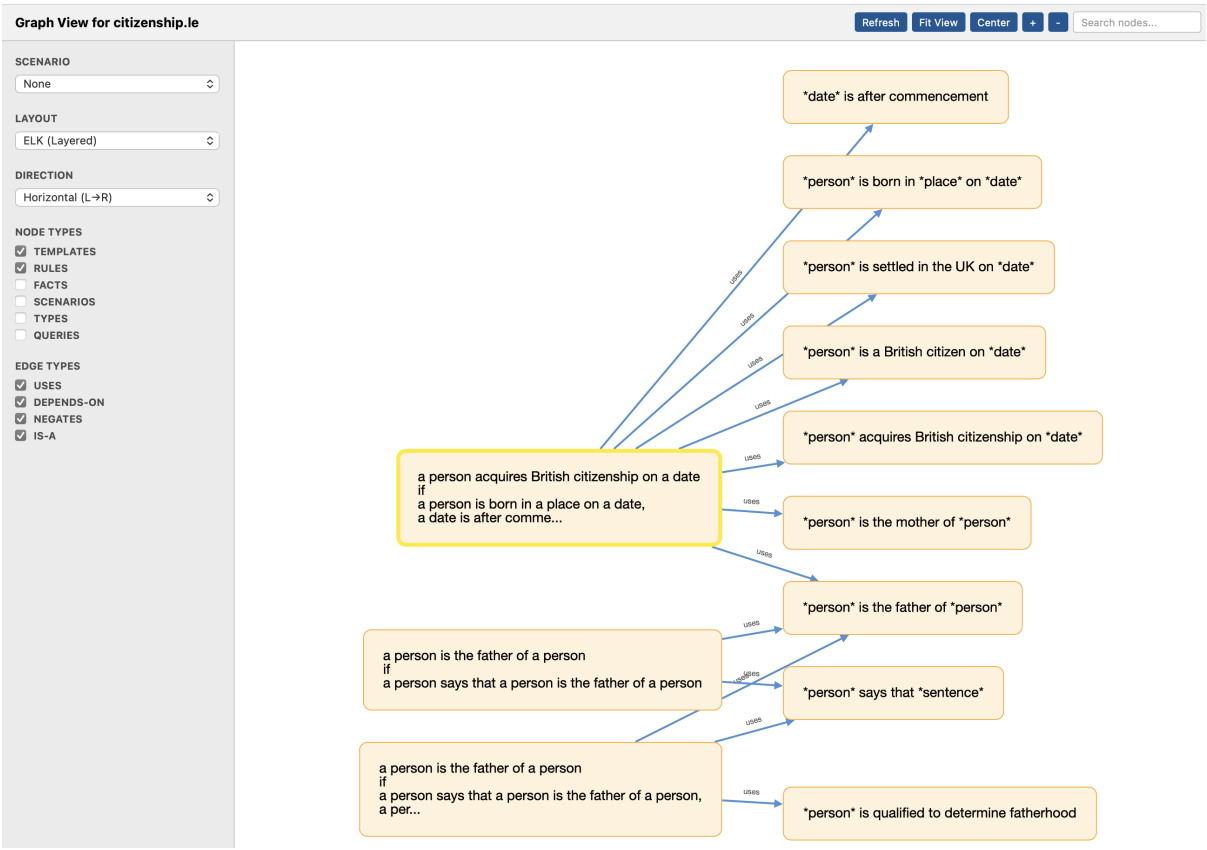


Figure 3: Graph view of a knowledge base, showing predicates and their rule-level dependencies.

4. Conclusion

LE1 and LE2 are best understood as two phases of the same project. LE1 established the language and the case for it [1, 2], explored its reach through experimental constructs and multiple back-ends, and demonstrated domain-specific viability through extensions such as InsurLE [3]. LE2 packages a consolidated core of that work as a self-contained development environment, with verification, debugging and LLM integration included. The two systems are complementary: LE1 remains the reference for the wider design space LE has explored, while LE2 is intended as a more accessible entry point for new users and new applications, including the LLM-assisted authoring scenarios discussed in [5].

Acknowledgments

The Logical English line of work has involved a wide group of contributors and supporting organisations across several jurisdictions. The copyright holders of the LE2 implementation, by country, are LodgeIT (AU), AORA Law and Axiome (UK), AINexus (USA), Bob Kowalski (UK), Miguel Calejo (PT) and Jacinto Dávila (VE). The authors are grateful to Robert Kowalski, Andrew Noble, John Cummins, Chris and Bruce Mennell, Galileo Sartor and Faramarz Farhoodi for discussions, feedback, and example material that have shaped both the language and its current implementation. The LE1 implementation, on which LE2 builds, is the work of a similarly broad group acknowledged in [1, 2].

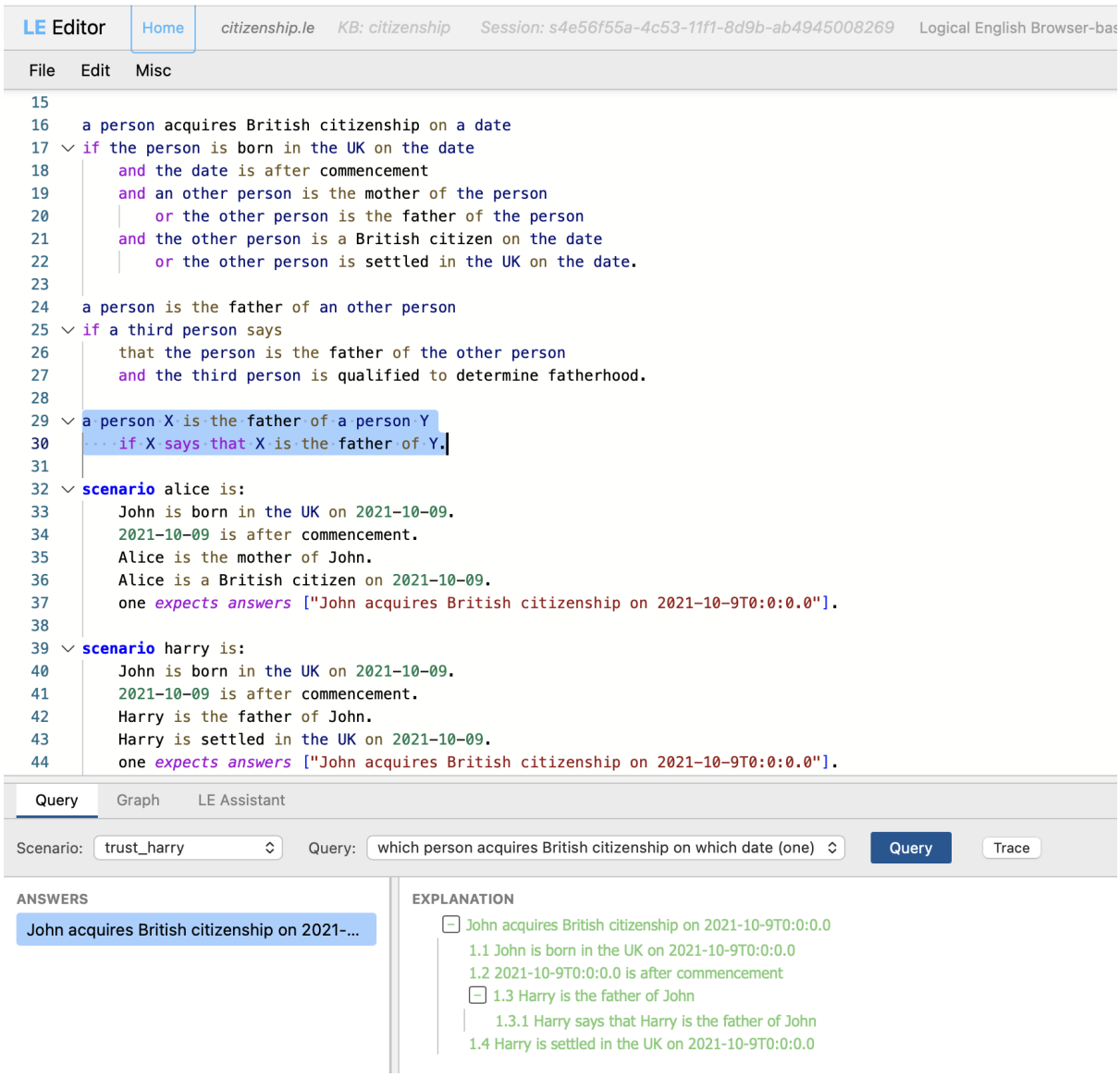


Figure 4: An interactive explanation tree in the LE2 IDE; clicking a node selects the corresponding source range.

Declaration on Generative AI

During the preparation of this work, the author(s) used Claude (Sonnet and Opus) and Google Gemini in order to draft and revise prose. After using these tools, the author(s) reviewed and edited the content as needed and take(s) full responsibility for the publication's content.

References

[1] R. Kowalski, J. Dávila, M. Calejo, *Logical English for Legal Applications*, 2021. http://www.doc.ic.ac.uk/~rak/papers/LE_for_LA.pdf.

[2] R. Kowalski, J. Dávila, G. Sartor, M. Calejo, *Logical English for Law and Education*, in: Prolog: The Next 50 Years, Lecture Notes in Computer Science, vol. 13900, Springer, 2023. https://doi.org/10.1007/978-3-031-35254-6_24.

[3] R. Kowalski, J. Cummins, J. Dávila, D. Ovenden, *Computable Contracts for Insurance: Establishing an Insurance-Specific Controlled Natural Language — InsurLE*, British Insurance Law Association Journal, 2024. http://www.doc.ic.ac.uk/~rak/papers/InsurLE.pdf.

[4] *Logical English 2.0* — project repository, documentation and live demo. https://github.com/LogicalContracts/LogicalEnglish2, https://le2.logicalcontracts.com.

[5] M. Calejo, *Can we vibe code financial contracts?*, Butterworths Journal of International Banking and Financial Law, March 2026, pp. 162–164. https://www.jibfl.co.uk/articles/can-we-vibe-code-financial-contracts-.

[6] J. Dávila Quintero, *Decompiling Language Models into Logic Programs*, Joint Proceedings of the Workshops and Doctoral Consortium of the 41st International Conference on Logic Programming, September 9—13, 2025, Rende, Italy. https://ceur-ws.org/Vol-4117/PEG-Regular-6.pdf.

[7] *Introduction to Logical English (LE) on SWISH - Version 1.0* — HandBook https://github.com/LogicalContracts/LogicalEnglish/blob/main/le_handbook.pdf.

[8] *The Syntax of Logical English, LE* — Version 1.0 https://github.com/LogicalContracts/LogicalEnglish/blob/main/le_syntax.md.

[9] C. Kelly, *Model Context Protocol (MCP) Explained* https://humanloop.com/blog/mcp.

[10] P. Tarau, E. Blanco, *Interactive text graph mining with a Prolog-based dialog engine*, Theory and Practice of Logic Programming, 21(2), 244-263. 2021 https://arxiv.org/abs/2008.00956

[11] F. Li, H. Wang, K. Basu, E. Salazar, G. Gupta, *DiscASP: A Graph-based ASP System for Finding Relevant Consistent Concepts with Applications to Conversational Socialbots* ICLP Technical Communications 2021: 205-218, 2021 https://arxiv.org/abs/2109.08297