

Project Proposal: An embedded DSL for querying OpenStreetMap

Matthew Occleshaw (mjo52) — Pembroke College

1 Introduction

OpenStreetMap (OSM) holds a rich set of geographic data on the world we live in, right down to the locations of bins, or whether a pedestrian crossing has tactile paving [3]. Ergonomic access to such detailed information could provide a benefit to many, especially those with access needs, for example by allowing routes to be planned that only go along streets with tactile aids for blind or visually impaired people or finding public toilets nearby. OSM stores its data in key-value tags, with a huge variety of information stored to allow you to answer questions like “find the accessible entrances to all Cambridge colleges”.

However, OSM data is entirely crowdsourced and the project enforces no strict schema on what people submit. There is an extensive wiki on which the community have defined some *de facto* conventions for many common types of entity, but there is great variation in the extent to which they are followed. The term *folksonomy* is often used to describe OSM’s key-value tag-based taxonomy and other similar organic tag-based systems maintained by communities [2].

Some tools exist for querying OSM, most significantly Overpass for high-level queries via an API [4] and osmium for fast lower-level queries over local data [5]. However, they tend not to address the incomplete nature of the folksonomy satisfactorily. For example, if we were looking to write a query to get to a station travelling only along lit roads, in Overpass we might use a filter like `[lit=yes]`, but this would silently drop every road lacking the `lit` tag. While we can select for the absence of a tag (using `[!lit]`), absence is still treated as false by default, and reconciling these ideas manually greatly complicates queries.

If we make incompleteness part of the semantics of the query DSL, with, for example, filters returning certain and possible sets, then we can address this problem. By embedding the DSL within OCaml, we can make use of its type system to distinguish certain results from possible ones, and use typed tag accessors (lenses and prisms) that separate present, absent, and malformed values [1]. Tag values that cannot be interpreted under the community’s conventions can then be treated as unknown in the semantics, in the same way as missing tags, so that both sources of incompleteness are handled uniformly. Furthermore, the gap between the certain and the possible is useful, as it tells OSM contributors where they are missing data that would be needed to resolve the query definitively.

2 Starting point

Only background research and exploration has been conducted. I spent some time looking into OpenStreetMap data, specifically the format it is provided in and the structure (or lack thereof) it provides. I will therefore need to spend some time further understanding the full range of semi-structured data held and the conventions governing it.

My knowledge of OCaml is only that of the Part IA *Foundations of Computer Science* course, so I will need to spend some time developing that, especially in relation to the more advanced parts of the OCaml type system/runtime like runtime types, GADTs, etc. My knowledge of semantics is largely limited to that of the Part IB *Semantics of Programming Languages* course, so this will also need some development.

3 Substance, structure, and success criteria

The *core goals* are the minimum expected goals that must be completed to consider this project a success. They correspond to a working eDSL that can be used to answer meaningful questions about the world based on OSM data.

The *extension goals* are planned improvements over the system arising from the core goals, but are relatively more difficult and not required to be completed for the project to be considered a success.

The success criteria each goal will be judged by in the evaluation section are included inline. There is also an extended section on evaluation at the end. All the goals will also be supported by unit testing to help ensure the software's real behaviour matches what we intended.

We will start by defining some terms:

Core Keys The fixed set of OSM tags we expect to support in the DSL, probably around ten

Core Query Suite Around five concrete questions drawn from the identified incomplete-data and spatial problems, covering filters (selecting elements by their tag), way-to-node joins (ways are lists of nodes that OSM uses to track where roads go; we need to retrieve the nodes before we can know where the roads go), counts (incompleteness leads to intervals between the certainly known answer and the possible maximum), and containment (does something lie within a given area)

Extended Query Suite A further set of problems to evaluate against, including constrained routing (a path through the road graph where every edge satisfies a condition) and probably dealing with multipolygons (the way that OSM represents more complex nested areas like a park with a lake in it)

Core Extract and Extended Extract Cambridgeshire and Great Britain respectively

Correctness criterion The results agree with osmium wherever the answer doesn't depend on the incompleteness properties of our system

Soundness criterion For any way of filling in unknown values, $certain \subseteq true \subseteq possible$

3.1 Core goals

The query language An embedded DSL making use of GADTs and its interpreter answer the Core Query Suite on the Core Extract and meet the correctness criterion. To do this, it must be able to read the data, handle joins, output meaningful GeoJSON, and likely will use some sort of optics to handle the different sorts of values the Core Keys can take.

Incompleteness semantics Queries over Core Keys return certain and possible sets and interval counts. The soundness criterion is met (no formal proof, just extensive testing and informal reasoning).

3.2 Extension goals

Extended Query Suite answered Features added to the query capabilities of the DSL to support the answering of these questions. This would require the introduction of routing capabilities into the query system.

Scaling Capable of answering queries over the whole Extended Extract with roughly flat memory usage, as well as using an R-tree for efficient spatial queries.

Gap explanation Finds the smallest set of missing tags that would make a possible answer certain as well as showing a deterministic diff between the certain route and the possible route.

Formal semantics A formal semantics is defined for the query core, likely with a proof of soundness.

Runtime types Types for tags beyond the Core Keys are discovered from the data at runtime, so that queries can be written over keys whose values were not fixed in advance.

3.3 Evaluation

Correctness Answers to the Core Query Suite will be compared to reference answers from osmium.

Soundness We can take well-tagged data and corrupt a selection of tags, then check that the original answer always lies between the certain and possible results.

Benefit over existing tools The same queries can be run under our open-world semantic model and under the existing closed-world ones (e.g. Overpass) to compare how many hidden mistakes are fixed.

Performance Profiling can be carried out against osmium on the Core Extract and on the Extended Extract if the scaling extension is reached.

4 Plan of work

9 – 23 October 2026

Background reading on the OSM data model, existing tools such as Overpass and osmium, and relevant theory on optics, GADTs and querying incomplete data. Set up the OCaml project and choose the Core Keys and Core Query Suite.

Milestone 1: implementation plan confirmed with supervisors.

24 October – 6 November 2026

Implement reading of OSM data from the Core Extract, and access to the Core Keys.

Milestone 2: Core Extract can be loaded and queried at a basic level.

7 – 20 November 2026

Implement the query core and its interpreter, covering the features needed for the Core Query Suite.

Milestone 3: the query language goal is complete.

21 November – 4 December 2026

Will have a Part II module exam at the end of this period, so most time will go on revision. Lighter project work on designing the incompleteness semantics.

Milestone 4: design of the incompleteness semantics confirmed with supervisors.

5 – 18 December 2026

On holiday, so little project work planned (unless needed to catch up to this schedule).

19 December 2026 – 15 January 2027

Implement the incompleteness semantics and test against the soundness criterion.

Milestone 5: the incompleteness semantics goal is complete. All core goals met.

16 – 29 January 2027

Absorb any overrun on the core goals, then begin work on the Extended Query Suite. Begin the progress report.

Milestone 6: extension plan confirmed with supervisors.

30 January – 12 February 2027

Write the progress report and presentation. Continue work on the Extended Query Suite.

Milestone 7: progress report submitted.

13 – 26 February 2027

Further extensions as time allows, prioritising scaling, then runtime types, gap explanation or formal semantics.

Milestone 8: implementation complete.

27 February – 12 March 2027

Carry out the evaluation and draft the evaluation chapter.

Milestone 9: evaluation results documented.

13 – 26 March 2027

Draft the remaining dissertation chapters.

Milestone 10: full dissertation draft sent to supervisors for feedback.

27 March – 23 April 2027

Act on feedback. Exam revision.

Milestone 11: second draft sent to supervisors for feedback.

24 April – 7 May 2027

Final revisions and proofreading. Exam revision.

Milestone 12: dissertation submitted.

5 Resource declaration

I have free access to a laptop with reasonable specifications (Intel Core i7-1185G7, 16GB RAM), which should generally be suitable for the development of the eDSL. In the event it fails, both my write-up and the code for my project will be backed up in remote Git repositories, hosted on GitHub, and I will be able to borrow a laptop from my college to continue working while a replacement is sourced. I accept full responsibility for this machine and I have made contingency plans to protect myself against hardware and/or software failure.

I may require access to a machine with more storage/memory to test the project on larger amounts of data for performance profiling. Prof. Anil Madhavapeddy has agreed to facilitate this if required.

The OCaml libraries and tools I plan to use are all open-source and freely available. OpenStreetMap data is available under a free and open data licence.

References

- [1] Patrick Ferris. Ocaml optics. <https://github.com/geocaml/ocaml-optics>, 2026. Accessed 8 October 2026.
- [2] Franz-Benjamin Mocnik, Alexander Zipf, and Martin Raifer. The OpenStreetMap folksonomy and its evolution. *Geo-spatial Information Science*, 20(3):219–230, September 2017.
- [3] OpenStreetMap contributors. OpenStreetMap. <https://www.openstreetmap.org>, 2026. Accessed 8 October 2026.
- [4] OpenStreetMap Wiki contributors. Overpass API. https://wiki.openstreetmap.org/wiki/Overpass_API, 2026. Accessed 8 October 2026.
- [5] Jochen Topf. Osmium tool. <https://osmcode.org/osmium-tool/>, 2026. Accessed 8 October 2026.