Grants4Companies: The Common Lisp PoC

Philipp Marek
Bundesrechenzentrum GmbH
Vienna, Austria
philipp.marek@brz.gv.at

Björn Lellmann
Bundesministerium für Finanzen
Vienna, Austria
bjoern.lellmann@bmf.gv.at

Markus Triska
Bundesministerium für Finanzen
Vienna, Austria
markus.triska@bmf.gv.at

ABSTRACT

The application *Grants4Companies* was recently introduced in the Austrian Federal Business Portal (*Unternehmensserviceportal*, *USP*). The productive application displays a list of business grants which apply to a business depending on the data available about this business in the systems of Austrian public administration. In this article we describe the underlying Proof of Concept implementation, used to experiment with and test new features. This PoC implementation is written in Common Lisp, interfaces with a Prolog-reasoner, and makes use of formalised grant descriptions based on S-expressions.

CCS CONCEPTS

• **Theory of computation** → *Automated reasoning*; • **Computing methodologies** → **Knowledge representation and reasoning**; • **Software and its engineering** → **Context specific languages**.

KEYWORDS

S-Expressions, Expert System, Applications

ACM Reference Format:
Philipp Marek, Björn Lellmann, and Markus Triska. 2024. Grants4Companies: The Common Lisp PoC. In *Proceedings of the 17th European Lisp Symposium (ELS'24)*. ACM, New York, NY, USA, 6 pages. https://doi.org/10.5281/zenodo.10992449

1 INTRODUCTION

Business grants offer vital funding opportunities for businesses and companies, and at the same time provide an important tool for supporting and steering economy. The efficiency of this tool, however, depends on whether it is easily possibly for businesses to find the suitable grants. In order to support businesses in this search in Austria we recently introduced the application *Grants4Companies* in the Austrian *Unternehmensserviceportal*[1] (*USP*, the *Business Service Portal*). The USP is the main portal for contact between businesses and public administration in Austria, currently serving over 120 integrated applications and more than 600.000 registered businesses.

The application *Grants4Companies* was introduced in November 2022 and offers registered businesses the possibility to evaluate a list of business grants based on the data available for the business from sources in Austrian public administration. For this purpose, formal eligibility criteria of a number of business grants are formalised in a logical language. The descriptions and eligibility criteria are based on the *Transparenzportal*[2], the official Austrian portal containing a wealth of data about Austrian funding possibilities for businesses as well as natural persons. Following the explicit consent for using the data available for the registered business, the formalised eligibility criteria are evaluated based on data from Austrian registers. The registers queried are currently the *Unternehmensregister für die Zwecke der Verwaltung*[3] (*Administrative business register*, a register collecting data about businesses from several sources in Austrian public administration) and the *Firmenbuch*[4] (Austrian company register). An extension to further registers is planned. The available grants are then presented to the business based on the evaluation of their criteria as "criteria are satisfied", "criteria are not satisfied" or "information is missing for sufficiently evaluating the criteria".

To quickly evaluate different strategies, ideas, and concepts, we implemented a Proof-of-Concept (POC) in Common Lisp in December 2019; over time we extended this POC to experiment with and test new features. In this article we describe the POC implementation and the underlying design choices. Differences between the POC and the productive software are noted as well. The POC also includes an interface to a logical reasoning engine for proving properties of the grants (or combinations thereof) which are independent of the particular businesses. This reasoning engine is implemented Prolog. Here we focus on the Lisp-specific parts of the POC and refer the reader to the companion paper [5] for the detailed description of the interface to the Prolog reasoning engine.

2 THE INITIAL PROOF-OF-CONCEPT

The POC is written in Common Lisp. Apart from personal preferences the reasons consisted of easier symbolic manipulation, quicker iteration cycles, and better performance. Based on that data point, choosing S-expressions as grant definition format (see Sec. 3) certainly looks like an easy and logical choice, and indeed was selected after careful deliberation. Apart from reading grant definitions, type-checking them, allowing evaluation against (fake) company data (the POC has no connection to the production register data bases), and converting the grant code back to natural language, the POC also acquired (limited) symbolic capabilities: *given a company, which data points (e.g., HQ location) need changing to match additional grants*? Other symbolic computations were made available by transpiling the grant code to Prolog and providing a Prolog REPL in the web interface; this allows queries like *Which grant totally includes another grant?*. For further discussion of this topic please see the already mentioned companion paper [5].

[1]https://www.usp.gv.at/en/index.html

[2]https://transparenzportal.gv.at/tdb/tp/startpage

[3]https://www.statistik.at/en/databases/business-register/administrative-business-register-abr

[4]https://www.justiz.gv.at/service/datenbanken/firmenbuch.36f.de.html

3 GRANT LANGUAGE AND SEMANTICS

There are a few thousand programming languages, even without counting the one-offs that are used by less than 10 people worldwide. Some share a bit of syntax, others are completely different. As basis for the formal language for specifying the grant conditions we needed something

- easy to read,
- unambiguous,
- future-proof (ie. backwards-compatible even for vastly changed situations),
- and easy to parse.

The first point means no XML; infix operators with their precedences (see the presentation) are worse off for the second. One format stood out as especially long lasting: S-Expressions. Participants of this ELS will already know, but the quick overview is:

- Evaluation from inner to outer, left to right within one form[5].
- No other precedence rules [6].
- Tokens are either atoms (numbers, strings, symbols), or
  - an opening parenthesis,
  - a list of (zero or more) tokens,
  - and a closing parenthesis.
- Comments are introduced with one or more semicolons (compatible with Common Lisp), but are not discarded but associated with the next form resp. the surrounding form. This allows to have human-readable explanations collected and used for explaining the evaluation of a grant.
- For ease of use (and compatibility with Common Lisp), symbols are defined to be case-insensitive; mixed case is not used. In strings and comments the case is kept, though.

S-Expressions have been cited for these features for a long time[1, 102], even in completely different fields (e.g. music [3, p.171]).

3.1 Concepts of the Formalisation Language

One point that we pondered for quite some time was the actual language used for the concepts of the formalisation language. We finally decided on a German/English mix:

- The specific data-query-functions, i.e., atomic concepts, that are derived from laws written in German were kept in German (`BETRIEBSSTANDORT-IN`, `ÖNACE-IN`, …); the higher similarity to the original law proved to be helpful in translating to computer language.
- Typical programming "keywords" like `AND`, `OR`, `NOT`, used for constructing complex expressions, were taken from English - the higher familiarity with these (compared to `"UND"`, `"ODER"`, `"NICHT"`, which just remind us of Winword macros!) makes them a better match.

3.2 Packages and Local Definitions

Grant definitions are stored in a GIT repository (for Version Control, Historical, Reproducability, and Data Sharing reasons). A directory tree definition ensures that each funding agency has their own workspace (sub)directory; a scoping rule that reads all grants within a directory into the same package allows funding agencies to define their own higher-level functions (see the presentation for an example).

The package that gets created for each directory imports functions from an API package automatically; so the most-often used symbols can be referenced directly. In the future, some kind of marker (eg. a specific filename like `package.lisp`) might switch to another behaviour: defaulting to another, improved API package, or manually specifying a `DEFPACKAGE` form. This separation into multiple packages also allows to divide up some responsibilities: by having high-level constructs in an extra package, it becomes much easier to maintain that in some separate organization.

An issue that came up right from the beginning is having one concept in multiple different implementations. A clause "*Der Antragsteller muss ein KMU*[6] *sein*" is used in many grants; sadly there are three different definitions for this term, one from the federal government in Austria, one from the EU, and one from the FFG[7].

Making (optional) packages available solves this in a neat way - there are simply three functions, `GV.AT:IS-KMU`, `FFG:IS-KMU`, and `EU:IS-KMU`. This enables the use of different interpretations of the same natural language term depending on the source of the regulation. Of course, the person digitizing the grant needs to know which one to use, which is a separate can of worms.

3.3 Security

Of course, nothing is ever quite so simple. As the funding agency is the (only) one who knows exactly what they want, they're the logical choice for capturing the intent in computer code, and maintaining it later on - including putting a cut-off date on it, or invalidating it some other way. Formalisation of the grants currently still happens via a few people and not the funding agencies, as would ideally be the case. But that means that the "code" (which is "data" here as well) is then run on a different computer system: primarily the central evaluation platform in the USP, but also decentralized on some funding agency's machine (when testing a grant), or potentially at the *Statistics Austria* (when estimating the number of businesses that potentially match some newly-defined grant). So all kinds of concerns regarding security come up! To address these, the specification says that only exported symbols from defined API packages may be used - so it's not allowed (respectively possible) to write `(CL:WITH-OPEN-FILE (s "/etc/shadow") ...)` in a grant to hack the grant evaluation platform.

3.4 Example grant

An example of a grant definition is shown in Fig. 1.

4 EVALUATING GRANTS

As grant forms are *by definition* side-effect free, their evaluation is in principle straightforward: Evaluate the atomic concepts based on the available data, and recursively evaluate complex expressions according to the outermost operator. In the productive version the atomic concepts are evaluated based on the data available for the logged-in business from public administration, while the POC uses

[5]Common Lisp Hyperspec, 3.1.2.1.2.3 Function Forms

[6]"*KMU*" means "*Klein- und Mittelunternehmen*", ie. "*small and medium-sized companies*".

[7]"*FFG*" is the abbreviation for "*Österreichische Forschungsförderungsgesellschaft*" (in English "*Austrian Research Promotion Agency*"), see https://www.ffg.at/en.

```
(define-grant ("Umweltschutz- und Energieeffizienzförderung - Förderung sonstiger Energieeffizienzmaßnahmen Villach"
               (:href "https://transparenzportal.gv.at/tdb/tp/leistung/1052703.html")
               (:transparenzportal-ref-nr 1052703)
               (:Fördergebiet :Umwelt)
               (gültig-von "2019-01-01"))
  "Unter der Berücksichtigung der Verwendung erneuerbarer Energieträger sowie
   der Umsetzung der Intention der Umweltschutz- und Energieeffizienzrichtlinie im
   Bereich privater Haushalte fördert die Stadt Villach folgende Energieeffizienzmaßnahmen."
  ;; Voraussetzungen
  ;;
  ;; - Förderungswerber/innen können natürliche oder juristische Personen sein.
  ;;   Bei juristischen Personen hat die firmenmäßige bzw. statutenkonforme
  ;;   Unterfertigung des Antrages auf Gewährung einer Förderung durch den
  ;;   Vertretungsbefugten zu erfolgen.
  (AND
    (GV.AT:natürliche-oder-juristische-Person)
    ;; - Die Förderungswerber haben bei der Antragstellung zu erklären, dass
    ;;   für die beantragten Förderungen keine weiteren Förderungen von anderen Stellen
    ;;   beantragt wurden.
    ;; - Ein Förderungsansuchen muss spätestens innerhalb von 8 Monaten nach
    ;;   Umsetzung der Maßnahme/n bzw. Kaufdatum bei der Stadt Villach einlangen
    ;; - Die Förderung wird nur für die sach- und fachgerechten Umsetzung der
    ;;   Maßnahme (Einbau) im Stadtgebiet von Villach gewährt.
    (OR
      (Unternehmenssitz-in 20201)
      (Betriebsstandort-in 20201))))
```

Figure 1: Example grant, TPPNr#1052703

dummy data of made-up businesses instead. Apart from this, there are some finer points for taking into consideration.

4.1 Evaluation modes

Because there are two main use cases, the Proof-of-Concept in Common Lisp implements two evaluation modes.

Fully Recursive, Exhausting Evaluation. In this mode all the forms are evaluated and their intermediate results are kept; by reporting these values in the same tree structure as the grant, manual verification of the calculation can be performed (Fig. 2). This evaluation mode is used when displaying grant results for a single company.

Fast Evaluation using Shortcut Properties. As the grants forms are pure, we have a few degrees of freedom for manipulating them or otherwise speed up evaluation. We implemented a short-circuit evaluation which can quickly discard grant/company pairs, allowing for faster mass assessments: given a newly proposed grant, how many companies in Austria will be able to apply? For the future, further optimisation are possible: for commutative operations (like "AND", "OR", numerical addition via "+"), we can reorder the forms before compiling. In the typical case of a top-level AND we can look at the sub-forms, and move the one with the highest probability for a negative result to the front, benefitting the short-circuiting operation again. This reordering is not implemented yet, though[8].

4.2 Three-valued Logic

Of course not all data required to evaluate whether a company satisfies the formalised eligibility criteria of a grant is always available. While data like *location* of a company needs to be provided before it is officially recognised, e.g., the (Ö)NACE classification[9] is not complete. In particular, for a sizeable number of companies the ÖNACE-classification has not yet been assigned. In addition, a data source might be not available, eg. due to maintenance work.

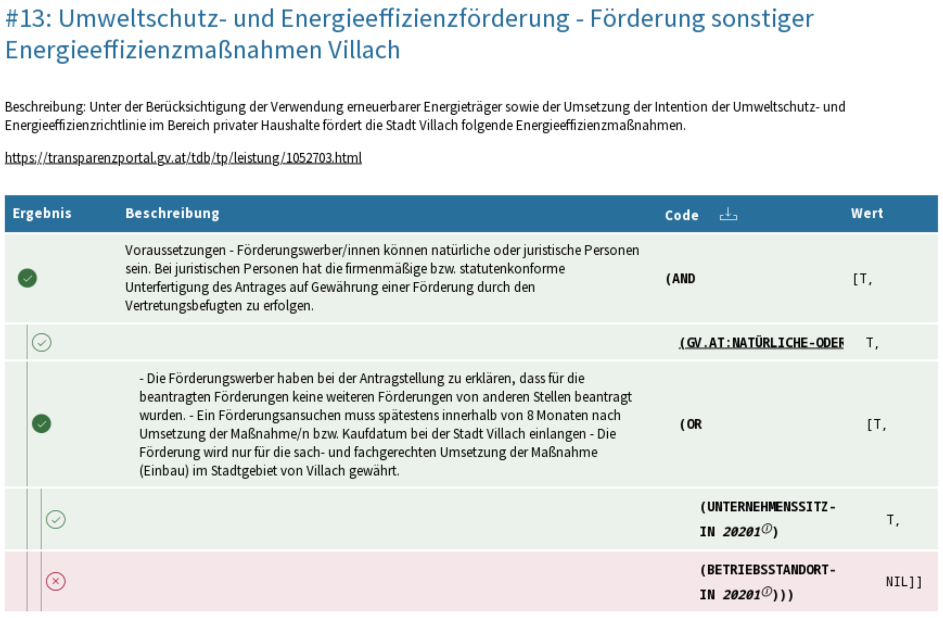
So the evaluation allows a value of *unknown* as well; many operations then need to propagate that *unknown* upwards. Easy cases are `OR` with a `true` value or `AND` with a `false` value. To be precise, we use (so far quantifier-free) *strong Kleene-Logic* K_3, considered e.g. in [4]. This ensures that grants which have been evaluated for a company to `true` or `false` while some of their atomic components are evaluated to `unknown`, get evaluated to the same result when additional data becomes available and these components no longer return `unknown`. Range-based reasoning for numeric operations would also be possible, but is not implemented yet.

4.3 Extension to probabilities

A major difference between the POC and the production software in the USP is that the POC already got extended to experiment with a 12-bits+1 probability space, with `false` being at one end, `true` at the other, and the unknown space spanning the values in between, with the canonical 0.5 unknown value in the exact center of the value range. This probability space gets evaluated Bayes-compatibly - so an `AND` over three `unknown`s means 0.5 to the third power, or a probability of 0.125. Of course this makes potentially problematic assumptions about the independence of the sub-expressions of a complex expression. The analysis on the suitability of this approach

[8]See chapter 4.9 below.

[9]https://www.statistik.at/en/databases/classification-database



#13: Umweltschutz- und Energieeffizienzförderung - Förderung sonstiger Energieeffizienzmaßnahmen Villach

Beschreibung: Unter der Berücksichtigung der Verwendung erneuerbarer Energieträger sowie der Umsetzung der Intention der Umweltschutz- und Energieeffizienzrichtlinie im Bereich privater Haushalte fördert die Stadt Villach folgende Energieeffizienzmaßnahmen.

https://transparenzportal.gv.at/tdb/tp/leistung/1052703.html

Ergebnis	Beschreibung	Code	Wert
✓	Voraussetzungen - Förderungswerber/innen können natürliche oder juristische Personen sein. Bei juristischen Personen hat die firmenmäßige bzw. statutenkonforme Unterfertigung des Antrages auf Gewährung einer Förderung durch den Vertretungsbefugten zu erfolgen.	(AND	[T,
✓		(GV.AT:NATÜRLICHE-ODEF	T,
✓	- Die Förderungswerber haben bei der Antragstellung zu erklären, dass für die beantragten Förderungen keine weiteren Förderungen von anderen Stellen beantragt wurden. - Ein Förderungsansuchen muss spätestens innerhalb von 8 Monaten nach Umsetzung der Maßnahme/n bzw. Kaufdatum bei der Stadt Villach einlangen - Die Förderung wird nur für die sach- und fachgerechten Umsetzung der Maßnahme (Einbau) im Stadtgebiet von Villach gewährt.	(OR	[T,
✓		(UNTERNEHMENSSITZ-IN *20201*)	T,
✗		(BETRIEBSSTANDORT-IN *20201*)))	NIL]]

Figure 2: Evaluated example grant with results.

and potential alternative ones is still ongoing, but its implementation in the POC means it is possible to experiment with the approach. By explicitly avoiding saturation[10], the strong Kleene-Logic still applies – but having a range of unknowns means that the output category where it matters most can be sensibly sorted!

4.4 Numeric calculations in the POC

The POC includes a small set of date and numeric capabilities - like checking whether a date precedes another (used to find out how long a company exists), respectively mirroring the calculations in some of the natural language grant texts; as an example, during the pandemic the "*Härtefallfonds*" asked whether the income exceeds 80% of some social security limit. See Fig. 3 for an abbreviated example; for production use the part of the calculation that references a common concept (e.g., the "*sozialversicherungsrechtliche Höchstbeitragsgrundlage*", the *Social security maximum contribution base*) would be extracted in its own function.

4.5 Calculations for the Past

If a calculation must be run later on (to check its validity, an application coming in the next calendar year, etc.), some concepts need to know the *application date*. The previously mentioned "*sozialversicherungsrechtliche Höchstbeitragsgrundlage*", like many other law-mandated values like tax limits, changes over time - but the value that was valid at the date of application must be used (which could be a few years in the past), so the function that encapsulates that concept needs to take the application date into consideration.

[10]So that an AND over 12 or more unknowns will never becomes a `false`, etc.

4.6 Input/output type derivation and -checks

The POC implements the expected boolean operators (AND, OR, NOT) as well as the G4C-specific atoms (like fetching the company legal form, the place of the headquarter, etc.), and some numeric capabilities. That means that grant descriptions (forms) have different input and output types:

- AND, OR, NOT only accept boolean (resp. probability) values;
- the numeric operators expect numbers and return numbers;
- the output of data query functions (atomic propositions) depends on the specific atom.

By using a small set of hard-coded input and output types, the types of all forms in a grant can be fully derived and checked for consistency; also, the expected value type of questions (see below) can be automatically decided and the correct type of HTML input field[11] used in the questioning form. This is one area where having some extra support from the Common Lisp compiler[12] would be a great plus: a stable, documented function to get the compiler-derived types of (some) subforms and a list of type mismatches after compilation. Because some macros are being used[13], association to the source forms might become a challenge, though.

4.7 Interactively asking for Data

Not all data queried by grants is stored in government registers; other data (in particular personal information, like number of disabled employees) would skyrocket the costs if it was stored persistently; and some items cannot be known in advance (eg., clauses

[11]Like `<input type=text>`, `type=number`, `type=date`, or radiobuttons.
[12]not all of them, of course -just one (SBCL) would be enough!
[13]Most notably for AND and similar, so that intermediate results get stored and associated to the subform - a function would only receive input values!

```
;; Im letzten abgeschlossenen Wirtschaftsjahr darf das Einkommen
;; vor Steuern und Sozialversicherungsabgaben maximal
(<= (frage "Einkommen")
    (*
      ;; 80%
      0.8d0
     ;; der jährlichen sozialversicherungsrechtlichen Höchstbeitragsgrundlage
     ;; betragen (https://www.oesterreich.gv.at/lexicon/H/Seite.991498.html).
      (+ (* 12 5370)
         ;; Sonderzahlung
         10740)))
```

Figure 3: Numeric calculation.

that describe the application itself). To reduce the set of potentially applicable grants it makes sense to ask a (limited) number of questions regarding the most often used data items.

The POC includes a high-performance evaluation engine for the grants (see below for details); this allows to recalculate the applicable set of results for the (planned) set of about 3000 grants in the backend and send results back to the frontend, *interactively*. So when some question gets answered, a quick check with the back-end allows to shrink the useful set of questions immediately, reducing the cognitive load on the person using the interface.

As an example, see Fig. 4 for the form before a (too high) number is put into row 2 ("*Anzahl der Kinosäle*"); as soon as the number 100 was acknowledged (typically by pressing `Tab` to get to the next input field), the form data are sent to the POC, which recalculates all grants that contain this question and replies with an update regarding styles (colors) and availability of input boxes - see Fig. 5. As the given value is too high, the conditions of the grant the data is used in (here labelled "48") can not be fulfilled any more - so the label's background becomes red, and related inputs are immediately disabled, as there's no need to answer them any more; if there were more questions for other grants, the cursor would move to the next one (done automatically by the browser frontend).

Also, the list of questions is sorted by impact - the more grants' results a question influences, the sooner it is listed.

4.8 Limited Reasoning - "What If"?

Before a Prolog interface was implemented, the POC got (limited) exploration capabilities, giving simple "What If?" answers.

As an example, one computation checks whether a grant would fail to apply because of (parametrized) number of clauses in it (kind of deduplicated, in case they are used in multiple places in a grant, like three `AND` branches all concerned with the `ÖNACE` and some other stuff); this allows to check for things like "*What do I need to change to apply for other grants?*".

4.9 Performance

The Common Lisp POC, utilizing SBCL[14] on standard x86-64 hardware, compiles the grant forms to native code; for nested loops over multiple (test) companies and about 30 grants (including fairly complex ones, see the presentation) the evaluation time averages to 0.5µsec (about 1000 CPU cycles). For ~600000 companies in Austria a test cycle in a browser frontend therefore takes less than half a second, facilitating true interactive grant development.

[14]https://sbcl.org

5 DATA SOURCES

The current sources for data about the companies in the production environment are the "*Unternehmensregister für die Zwecke der Verwaltung*" and the "*Firmenbuch*", a public listing of companies. These two registers provide general data about the company, but in order to evaluate certain eligibility conditions other information about the company might be required. Querying additional registers providing this information is ongoing work.

To enable also the evaluation of conditions, for which no data is available from an official source, in the POC we already drafted the concept `FRAGE` (question), which asks data from the company to answer grant forms (deduplicating questions, and not asking for data that is irrelevant because it won't be used[15].

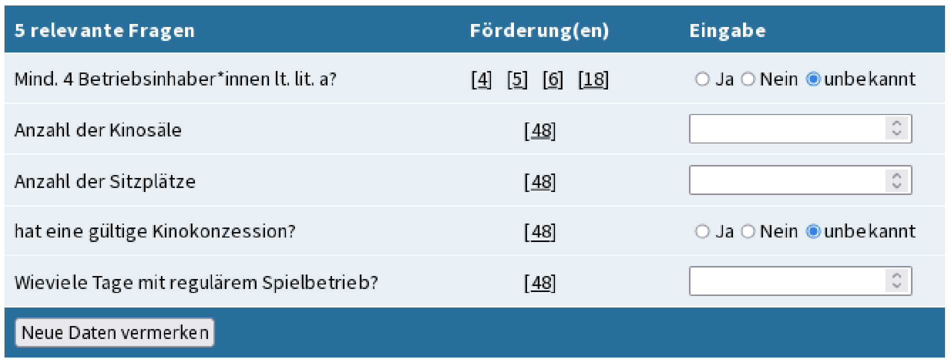
For the future, we're investigating to add other data sources as well; most of these will (for legal reasons) require an explicit consent from the company.

6 EXPERIENCE REPORT

Our experience matches documented history: Common Lisp is a viable programming language for rapid prototyping. Using it for production use still proves challenging - the strategic focus of most companies is still fixed on Java, changing the multi-man-year lore cannot be done in a day. While there are Prolog libraries for Common Lisp[16], and even one that allows converting Lisp data to Prolog syntax and forward the result to an implementation[17], none of them completely fulfilled the requirements:

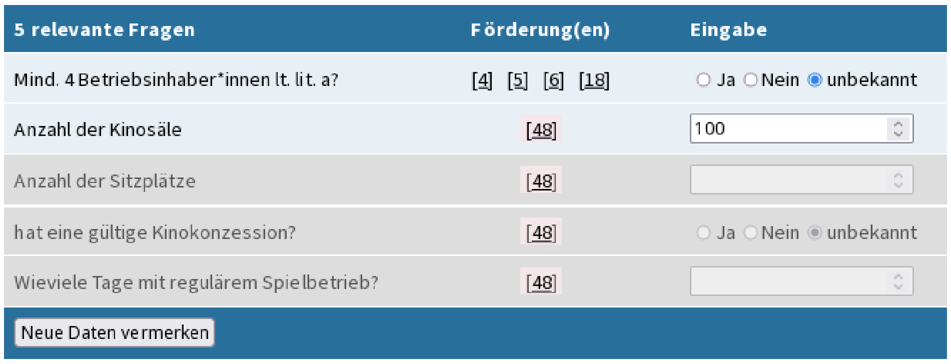
- bidirectional communication,
- parsing the Prolog output to provide a highlighted/clickable display in the web UI,
- multiple parallel, independent sessions to concurrently test different analyses,
- ability to export the Prolog input data for use in a separated (offline), ISO-conformant Prolog system.

[15]For example, `(AND (`*<some clause that evaluates to false>*`) (FRAGE "..."))` doesn't need to be asked for this grant – though another grant might require the same data item and is not always rejected!
[16]See, e.g., [2], https://www.lispworks.com/documentation/lw445/KW-W/html/kwprolog-w-152.htm, https://github.com/nikodemus/screamer
[17]https://github.com/cl-model-languages/cl-prolog2



5 relevante Fragen	Förderung(en)	Eingabe
Mind. 4 Betriebsinhaber*innen lt. lit. a?	[4] [5] [6] [18]	Ja Nein [✓ unbekannt]
Anzahl der Kinosäle	[48]	
Anzahl der Sitzplätze	[48]	
hat eine gültige Kinokonzession?	[48]	Ja Nein [✓ unbekannt]
Wieviele Tage mit regulärem Spielbetrieb?	[48]	
Neue Daten vermerken		

Figure 4: Interactive queries, before answering.



5 relevante Fragen	Förderung(en)	Eingabe
Mind. 4 Betriebsinhaber*innen lt. lit. a?	[4] [5] [6] [18]	Ja Nein [✓ unbekannt]
Anzahl der Kinosäle	[48]	100
Anzahl der Sitzplätze	[48]	
hat eine gültige Kinokonzession?	[48]	Ja Nein [✓ unbekannt]
Wieviele Tage mit regulärem Spielbetrieb?	[48]	
Neue Daten vermerken		

Figure 5: Interactive queries, after input.

So we ended up with our own implementation, using *Scryer Prolog*[18] as backend. The technical iterations proved to be easy; organisational/legistic changes (eg., having computer code on an equivalent legal basis as the grant texts) are hard, and still being worked upon. The bottleneck for broad usage is the translation from natural language to computer code; work takes place to run first translations via an Artificial Intelligence[19]. Of course, to get some real legal weight, a legal spokesperson would need to sign off the translated computer code; designing processes (re-translating the code to natural language for easier comparison again, having the sign-off directly via a GIT commit, etc.) is another required major step forward. Work continues...

7 CONCLUSION

By using plain text files with a reasonably simple syntax it is possible to translate written law into computer-readable data that's at the same time usable as computer code. By using data sources that are defined to contain valid and up-to-date data, a quick pre-selection (ie. not showing grants that are known not to apply to a company) can be provided to company owners. In the future, the fact-checking that currently is done manually could possibly be avoided - either by just providing the available data items in some secure form (a digitally signed JSON-blob), or by simply signing a statement that the company matches the requirements, obviating any need for further checks.

[18]https://www.scryer.pl

[19]Efforts driven by the Ministry of Finance under the umbrella *Law as Code*, though interest is found on the EU level as well, see https://joinup.ec.europa.eu/collection/better-legislation-smoother-implementation/news/new-course-law-code.

REFERENCES

[1] J. Belzer, A.G. Holzman, and A. Kent. 1978. *Encyclopedia of Computer Science and Technology: Volume 10 - Linear and Matrix Algebra to Microorganisms: Computer-Assisted Identification.* Taylor & Francis.
[2] Giuseppe Cattaneo and Vincenzo Loia. 1988. A Common-LISP implementation of an extended Prolog system. *SIGPLAN Notices* 23, 4 (1988), 87–102.
[3] Lounette M. Dyer. 1986. MUSE: An Integrated Software Environment for Computer Music Applications. In *Proceedings of the 1986 International Computer Music Conference, ICMC 1986, Den Haag, The Netherlands, October 20-24, 1986.* Michigan Publishing, 167–172. https://hdl.handle.net/2027/spo.bbp2372.1986.033
[4] Stephen Cole Kleene. 1952. *Introduction to Metamathematics.* North-Holland, Amsterdam.
[5] Björn Lellmann, Philipp Marek, and Markus Triska. 2024. Grants4Companies: Applying declarative methods for recommending and reasoning about business grants in the Austrian public administration (System description). In *Proceedings of FLOPS2024 (accepted).*
[6] William G. Wong. 1983. LISP for CP/M. *Microsystems* 4, 8 (1983), 30–43.