Grants4Companies: Applying Declarative Methods for Recommending and Reasoning About Business Grants in the Austrian Public Administration (System Description)

Björn Lellmann[1(✉)], Philipp Marek[2], and Markus Triska[1]

[1] Bundesministerium für Finanzen, Vienna, Austria
{bjoern.lellmann,markus.triska}@bmf.gv.at
[2] Bundesrechenzentrum GmbH, Vienna, Austria
philipp.marek@brz.gv.at

Abstract. We describe the methods and technologies underlying the application *Grants4Companies*. The application uses a logic-based expert system to display a list of business grants suitable for the logged-in business. To evaluate suitability of the grants, formal representations of their conditions are evaluated against properties of the business, taken from the registers of the Austrian public administration. The logical language for the representations of the grant conditions is based on S-expressions. We further describe a Proof of Concept implementation of reasoning over the formalised grant conditions. The proof of concept is implemented in Common Lisp and interfaces with a reasoning engine implemented in Scryer Prolog. The application has recently gone live and is provided as part of the *Business Service Portal* by the Austrian Federal Ministry of Finance.

Keywords: Applications · Expert systems · S-Expressions · Common Lisp · Scryer Prolog

1 Introduction

Business grants are an important tool for steering and supporting the economy. In addition, they can be used to quickly react to and counter crises. However, the search for suitable business grants can be a challenge for companies and businesses in Austria. This is due to the large number of available business grants from a multitude of different providers. While there are dedicated search engines, companies and businesses often are simply not aware of the existence of grants on a specific topic, and hence cannot use these engines in a targeted search.

As an additional tool for providing targeted information about potentially interesting business grants to businesses the application *Grants4Companies* was

J. Gibbons and D. Miller (Eds.): FLOPS 2024, LNCS 14659, pp. 151–164, 2024.
https://doi.org/10.1007/978-981-97-2300-3_9

introduced. The application is part of the Austrian *Unternehmensserviceportal* (USP)[1] and is productive since November 2022 with around 50 visits per month on average. The application uses data about available grants from the Austrian *Transparenzportal*[2] to formalise formal grant conditions, e.g., on the type of business or the location of the head office. Data sources within the public administration are queried and used to evaluate these formalised criteria, to display a list of grants ordered according to the feasibility of applying - i.e. whether the business fulfils the criteria or doesn't fulfil the criteria; a third category contains grants for which the available information is not sufficient to decide.

While the application in the USP is written in Java[3], we have also implemented a Proof of Concept (PoC) for testing out new features, which we describe in more detail in this article. In particular, this PoC contains a reasoning engine for reasoning about the formalised grant conditions themselves. The main features of the PoC are implemented in Common Lisp while the reasoning engine is implemented in Scryer Prolog[4], following the Lean Methodology [1] for implementing proof search in logical calculi using Prolog's backtracking mechanism. The PoC is of interest for two reasons: First, it combines implementations in Common Lisp and Scryer Prolog to leverage the strengths of each programming language. Second, it provides an example and showcase for the use of declarative programming languages in public administration. To the best of our knowledge, such examples are currently rather rare.

The source code for the reasoning engine complete with examples of business grants with their conditions is available under https://github.com/blellmann/g4c-reasoner. While there is no openly accessible web interface, the reasoning engine can be loaded into the *Scryer Playground*[5], the freely accessible web interface for Scryer Prolog, and used for running evaluations.

In the remainder of the article we first give a brief overview of the development history (Sect. 2), followed by a description of the productive implementation of *Grants4Companies* (Sect. 3) and the technical details underlying the representation of the grants as well as their evaluation (Sect. 4). We then provide details about the PoC implementation (Sect. 5) including the implementation of the reasoning engine and the interface between the Common Lisp implementation and the Prolog reasoner, before concluding with an outlook (Sect. 6). We do not include any benchmark results or comparisons regarding efficiency of the reasoning engine here, since the focus of the implementation is on correctness instead of maximal efficiency, and it is part of a PoC implementation. Since the examples of grants are taken from the official productive data set, we chose to keep the original formulation of the examples and several concepts of the

1 The official Austrian portal for interaction between businesses and public administration. See https://www.usp.gv.at/en/ueber-das-usp/index.html.

2 The official Austrian data base containing (amongst other information) data about the available grants. See https://transparenzportal.gv.at.

3 Due to interoperability concerns with existing libraries.

4 See https://www.scryer.pl.

5 See https://play.scryer.pl.

representation language in German, providing additional explanations in English. The technical terms from Austrian legislation can of course be adapted to other languages.

2 Development History

To assess the basic feasibility of the approach, we started with a pilot project, using Common Lisp for rapid prototyping. Grants were expressed as Lisp forms, a natural representation when working with Lisp. The pilot was successful, and also served as an illustration and internal tool for communicating the approach we planned. Already in this phase of the project, particular care was taken to explain in the UX that company data would only be processed with explicit user consent, and no data would be stored permanently by the planned service. In order to demonstrate the key concepts without any legal concerns, the pilot did not use any real company data, but only a fixed set of imaginary test companies.

For the production version of our service, we replaced the Common Lisp engine with a Java-based implementation to align the engine with architectural principles of surrounding IT services, and we retained the representation of grants as Lisp forms. As a result, the Lisp-based pilot can still be seamlessly used on the production data of the formalised grants to quickly prototype and assess additional features, while the Java-based Lisp parser and evaluation engine can also be used in other IT-services that require a Java implementation for architectural or other reasons. Only the production version of the service has access to real company data, and explicit consent of the company is required.

An additional component is the Prolog-based reasoner described in Sect. 5.1. This component can be used independently of the production environment to reason about grants, and is freely provided in a public repository. This component can reason with the productive formalised grants. Since the reasoning concerns only logical relations between the grants themselves, no company data is used by the reasoner.

3 Grants4Companies Overview

While the main focus of this article is the presentation of the PoC implementation of extended features for *Grants4Companies*, for context we briefly describe the productive application. *Grants4Companies* is an application in the Austrian *Unternehmensserviceportal* (USP)[6]. The USP is Austria's main digital portal for the interaction between public administration and businesses with currently more than 600.000 registered businesses and more than 120 integrated applications. It also acts as identity provider for the businesses.

After logging into the USP and starting the application *Grants4Companies*, businesses consent to the use of their data from registers of public administration in line with the GDPR [10]. Following this consent, the application fetches

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

available data about the companies from registers of public administration. Currently the data sources are the *Unternehmensregister* and the *Firmenbuch*, the data used concerns, e.g., information about the geographic location of the business, its legal type, or the area of business following the Austrian version of the NACE-classification[7]. The extension to further registers is planned. Companies are then presented with a list of grants, ordered according to whether the formal grant criteria are satisfied by the company, not satisfied, or cannot be sufficiently evaluated based on the available data. The latter option caters for potential unavailability of necessary data from the registers, due to lack of coverage or also maintenance downtime of the registers. The results can be filtered and sorted according to the evaluation result, categories of the grants, or application date. A screenshot of the productive version is shown in Fig. 11.

The architecture of *Grants4Companies* follows that of classical knowledge based systems, with a clear separation between the knowledge base, i.e., grant definitions including the formalised grant criteria, and evaluation engine. The evaluation engine of the productive version of *Grants4Companies* is implemented in Java. The knowledge base contains currently 45 grants which were formalised manually. The details of the formal language used for representing the grants will be considered in Sect. 4.1. The knowledge base is stored in a GIT repository to keep track of historical data, and enable version control, reproducibility and data sharing. This knowledge base is shared with the PoC implementation.

4 Representation and Evaluation of the Grant Conditions

The knowledge base containing the grants with their formalised grant conditions is based on data about Austrian grants contained in the Austrian *Transparenzportal*[8], a portal provided by the Austrian Ministry of Finance, where funding agencies are to enter grants and the granted funding. For the PoC and the initial productive version of *Grants4Companies*, a number of grants were formalised manually by us, the current knowledge base contain 45 grants. In the future this might be extended following a rules as code approach [6], e.g., using tools like *POTATO* [5,8] for automatically suggesting formalised grant conditions based on the natural language descriptions provided by the funding agencies.

4.1 Representation of the Grants

The grant conditions are formalised as quantifier-free logical formulae. The language contains predicates for expressing properties of the businesses related to location, legal form, classification of business activity, etc. Examples of *atomic formulae* with their intended semantics are given in Fig. 1. For ease of use by Austrian funding agencies, these predicates are formulated in German and

[7] See https://ec.europa.eu/eurostat/statistics-explained/index.php?title=Glossary:Statistical_classification_of_economic_activities_in_the_European_Community_(NACE).
[8] See https://transparenzportal.gv.at/tdb/tp/startpage (in German).

Atomic Formula	Intended semantics
`Betriebsstandort-in(L)`	The business has a location in one of the areas/regions specified in the list `L`
`Rechtsform-in(L)`	The legal form of the business is one of those in the list `L`
`ÖNACE-in(L)`	The business activity classification falls under one of the areas in the list L

Fig. 1. Examples of atomic formulae and their intended semantics

often take a list as argument. *Complex formulae* are built from the atomic formulae as well as $\top, \bot$ as usual using the standard propositional connectives $\neg, \vee, \wedge, \rightarrow$. At the current state there was no need for quantifiers, these might be added in the future. Working in a quantifier-free language has the benefit of a greatly reduced complexity for the reasoning tasks, of course. For the sake of referring to commonly used concepts, the language also contains *defined concepts*. On the logical level, these are given as pairs $(\mathfrak{d}, D)$ consisting of the name $\mathfrak{d}$ of the concept, which can be used like an atomic formula, and its definition D, i.e., a formula not containing $\mathfrak{d}$. The definition might contain other defined concepts, absence of cycles is assumed to be ensured externally. E.g., the concept of a legal person is introduced as the an abbreviation with name `G4c/Grants_Gv.At:Ist-Juristische-Person` for the formula `Rechtsform-in(L)`, where $L = [\texttt{Genossenschaft}, \texttt{Verein}, \ldots]$ is a list of the legal forms which count as legal persons in Austria. Naming the definitions in the style of packages makes it possible to differentiate between concepts with the same name from different funding agencies, e.g., general funding conditions specific to the funding agencies.

On a technical level, the language used for representing the logical formulae is based on the Lisp-syntax of *S-expressions* [2, 102]. In particular, the logical formulae formalising the grant conditions are represented in prefix notation as lists, where the first element is the logical connective and the following elements are its arguments. E.g., a formula $\neg A \wedge (B \vee C)$ is represented as the S-expression `(and (neg A) (or B C))`. Predicates are represented by (Common Lisp) symbols. E.g., the predicate `Betriebsstandort-in` represents the fact that the business has a location in one of a list of certain areas given by their *Gemeindekennzahl*, the Austrian identification number for municipalities. To enable restriction also on a regional or county level, also prefixes of these identification numbers are covered. E.g., the atomic formula `(Betriebsstandort-in 2 617 60101)` represents the assertion that the business has a site in the county Carinthia, the region East Styria, or the municipality of the city of Graz.

The full representation of a grant also contains in addition to the formalised grant conditions also its name, metadata about application dates and links to the full description on the Transparenzportal, as well as the natural language description of the grant conditions. The latter are included as Lisp comments

```
(def-concept gv.at:natürliche-oder-juristische-Person
  (OR
    (Rechtsform-in :Einzelunternehmen)
    (gv.at:Ist-Juristische-Person)))
```

Fig. 2. The definition of the concept `gv.at:natürliche-oder-juristische-Person`. The formula captures the condition that the applicant is a natural person, i.e., the legal form of the company is that of a sole trader (`:Einzelunternehmen`), or a legal person (captured by the defined formula `gv.at:Ist-Juristische-Person`).

interspersed with the formalised conditions in the spirit of literate programming [4]. This allows to have human-readable explanations collected and used for explaining the evaluation of a grant. An example of a grant in this representation is given in Fig. 3. Defined concepts $(\mathfrak{d}, D)$ are represented as `(def-concept` $\mathfrak{d}$ D`)`. An issue that came up right from the beginning is having one concept in multiple different implementations. A clause specifying that the company has to be a small or medium enterprise (SME, in German "*Der Antragsteller muss ein KMU 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[9]. As mentioned, this ambiguity is solved via package names - 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. An example of a defined concept is given in Fig. 2.

4.2 Evaluation of the Grants

Evaluating whether the formal conditions of a grant apply for a specific business essentially corresponds to checking, whether the business is a model of the logical formula representing these conditions. Here the business is identified with its properties given by the data about the business available. The atomic formulae are chosen to directly correspond to data fields from specific registers and hence their evaluation is rather straightforward. Complex formulae are evaluated according to their main logical connectives. Names $\mathfrak{d}$ for defined concepts $(\mathfrak{d}, D)$ are unpacked into their definition D and then evaluated.

Of course not all the data required to evaluate whether a company satisfies the formalised eligibility criteria of a grant is necessarily always available. While data like *location* of a company needs to be provided before it is officially recognized, e.g., the (Ö)NACE classification[10] of the economic activities of Austrian businesses is not complete. In particular, for a sizeable number of companies the ÖNACE-classification has not yet been assigned. In addition, the connection to a specific register might drop out temporarily due to maintenance work.

[9] The *Österreichische Förderagentur für wirtschaftsnahe Forschung, Entwicklung und Innovation*, in English *Austrian Research Promotion Agency*.

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

```
(define-grant ("Umweltschutz- und Energieeffizienzförderung - Förderung
 sonstiger Energieeffizienzmanahmen 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 Energieeffizienzmanahmen."
  ;; Voraussetzungen
  ;;
  ;; - Förderungswerber/innen können natürliche oder juristische Personen
  ;;   sein. Bei juristischen Personen hat die firmenmäige 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 Manahme/n bzw. Kaufdatum bei der Stadt
    ;;   Villach einlangen
    ;; - Die Förderung wird nur für die sach- und fachgerechten Umsetzung
    ;;   der Manahme (Einbau) im Stadtgebiet von Villach gewährt.
    (OR
      (Unternehmenssitz-in 20201)
      (Betriebsstandort-in 20201))))
```

Fig. 3. Example grant, TP-Nr.1052703. The grant provides funding for increasing energy efficiency. It is applicable to natural and legal persons (`GV.AT:natürliche-oder-juristische-Person`) in the city of Villach (the `Unternehmenssitz` or a `Betriebsstandort` has to be in the municipal identification number 20201). Some other conditions cannot be checked automatically based on the data about the company available within public administration and hence are not formalised (e.g., that the request for funding has to be submitted at most 8 months after implementing the measures for increasing energy efficiency).

To cover these eventualities, the evaluation is done in a three-valued logic, which allows a third truth value of *unknown* next to *true* and *false*. The logical connectives then propagate the truth value *unknown* upwards, whenever no definite evaluation to *true* or *false* is possible. To be precise, we use (so far quantifier-free) *strong Kleene-Logic* K_3, considered, e.g., in [3]. The truth tables for the logical connectives are given in Fig. 5. 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* are evaluated with the same result

```
förderung("G4c/Grants_Umweltschutz- Und Energieeffizienzförderung
 - Förderung Sonstiger Energieeffizienzmanahmen Villach",
  förderkriterien(
    (
      df( "G4c/Grants_Gv.At:Natürliche-Oder-Juristische-Person" )
    and
      (
        at( unternehmenssitz_in( [ 20201 ] ) )
      or
        at( betriebsstandort_in( [ 20201 ] ) )
      )
    )
  )
).
```

Fig. 4. Example grant in Prolog syntax, TPPNr#1052703. For the original formulation of this particular grant, see Fig. 3

	$\neg$
$\bot$	$\top$
u	u
$\top$	$\bot$

$\wedge$	$\bot$	u	$\top$
$\bot$	$\bot$	$\bot$	$\bot$
u	$\bot$	u	u
$\top$	$\bot$	u	$\top$

$\vee$	$\bot$	u	$\top$
$\bot$	$\bot$	u	$\top$
u	u	u	$\top$
$\top$	$\top$	$\top$	$\top$

$\rightarrow$	$\bot$	u	$\top$
$\bot$	$\top$	$\top$	$\top$
u	u	u	$\top$
$\top$	$\bot$	u	$\top$

Fig. 5. The truth tables for 3-valued strong Kleene logic K_3. The truth values *false*, *unknown*, *true* are represented by $\bot$, u and $\top$, respectively.

when additional data becomes available and some of the atomic components are no longer evaluated as *unknown*. Range-based reasoning for numeric operations would also be possible, and is planned as future work.

As a further potential next step, the symbolic representation also allows for some easy optimizations – for commutative connectives/operations (like `AND`, `OR`, possibly in the future also numerical addition via `+`), we could reorder the arguments before evaluating. By moving the subformula with the highest probability for a negative result to the front, a short-cutting evaluation could quickly discard grant/company pairs, allowing for mass assessments: given a newly proposed grant, how many companies in Austria will (be able to) apply? This reordering is not implemented yet, though.

5 PoC: Extensions and Interfaces

The PoC also contains an implementation of the evaluation of grant conditions based on company data. However, for the purpose of this article we concentrate on the functionality which goes beyond that of the productive system. In particular, the representation of grant conditions as logical formulae opens the possibility to not just evaluate the conditions based on business data, but to also reason *about* the conditions themselves. Interesting questions here are in particular consistency, useful for discovering mistakes in the formalisation of grant conditions, and logical implication, useful for finding unintended overlap between multiple grants in the same area. To enable such reasoning, we implemented backwards proof search in a Gentzen-style sequent calculus (see, e.g., [9] for the proof-theoretic background). Following the Lean-methodology [1] we make use of Prolog's backtracking mechanism to perform the proof search.

We use Scryer Prolog due to its strong conformance to the Prolog ISO standard, which will ease future cooperations with other organizations and public administrations. In addition, the system is freely available and allows inspection of its entire source code, which works towards our aim of providing full transparency and explainability of all computed results.

5.1 Symbolic Reasoning over Grants

The Proof-of-Concept has the ability to connect one or more *Scryer Prolog*[11] sessions to the web frontend, providing a convenient REPL that is pre-loaded with some known facts and the transpiled grant forms (see Fig. 4). We included a prototypical implementation of logical reasoning over the formalised eligibility criteria in the form of a *sequent calculus*, specifically a `G3`-style calculus for classical (propositional) logic (see, e.g. [9]), extended to cover basic facts about atomic statements and the defined concepts. We use reasoning in classical logic and not the three-valued logic used for evaluating the grants, because reasoning about the logical properties of grant conditions is independent of the data available for particular businesses. A calculus for the three-valued logic used could be implemented, e.g., following [7]. However, this would be useful mainly for reasoning about which grants are shown to the business with which evaluation.

As usual, *sequents* are of the form $A_1, \ldots, A_n \Rightarrow B_1, \ldots, B_m$ with $n, m \geq 0$ and are interpreted as the logical formula $A_1 \wedge \cdots \wedge A_n \rightarrow B_1 \vee \cdots \vee B_m$. The standard logical rules are given in Fig. 6. Basic knowledge about implications between atomic statements is included in the form of (rather simple) *ground sequents*, and defined concepts are included in the form of separate left- and right rules for each defined concept. The ground sequents and rules for unpacking the defined concepts are given in Fig. 7. Cut-free completeness of the calculus follows from an extension of [9, Thm.4.6.1] to the calculus with defined formulae, noting that the set of ground sequents is closed under substitutions (because no variables occur), contraction and basic cuts. In order to avoid unnecessary repetitions, in the implementation the rules are given as facts about the term `rule(Name, Prem_List/PF)`, where `Name` is the name of the rule, `Prem_List` is the list of premisses and `PF` is the principal formula of the rule, i.e., a sequent with exactly one formula on the left or right hand side. The provability predicate is given by `prov2//2`, which is true if the first argument is a derivable sequent, and the second argument a term describing a corresponding derivation. Examples are given in Fig. 8. The auxiliary predicate `merge_sequent_list//3`

[11] See www.scryer.pl.

$$\frac{}{\Gamma, A \Rightarrow A, \Delta} \qquad \frac{}{\Gamma, \bot \Rightarrow \Delta}\ \bot_L \qquad \frac{}{\Gamma \Rightarrow \top, \Delta}\ \top_R$$

$$\frac{\Gamma \Rightarrow A, \Delta}{\Gamma, \neg A \Rightarrow \Delta}\ \neg_L \qquad \frac{\Gamma, A \Rightarrow \Delta}{\Gamma \Rightarrow \neg A, \Delta}\ \neg_R$$

$$\frac{\Gamma, A, B \Rightarrow \Delta}{\Gamma, A \wedge B \Rightarrow \Delta}\ \wedge_L \qquad \frac{\Gamma \Rightarrow A, \Delta \quad \Gamma \Rightarrow B, \Delta}{\Gamma \Rightarrow A \wedge B, \Delta}\ \wedge_R$$

$$\frac{\Gamma, A \Rightarrow \Delta \quad \Gamma, B \Rightarrow \Delta}{\Gamma, A \vee B \Rightarrow \Delta}\ \vee_L \qquad \frac{\Gamma \Rightarrow A, B \Delta}{\Gamma \Rightarrow A \vee B, \Delta}\ \vee_R$$

$$\frac{\Gamma, B \Rightarrow \Delta \quad \Gamma \Rightarrow A, \Delta}{\Gamma, A \to B \Rightarrow D}\ \to_L \qquad \frac{\Gamma, A \Rightarrow B, \Delta}{\Gamma \Rightarrow A \to B, \Delta}\ \to_R$$

Fig. 6. The sequent rules of the propositional part of calculus $\mathsf{G3}$

$$\frac{}{\Gamma, \mathsf{unternehmenssitz\_in}(L_1) \Rightarrow \mathsf{unternehmenssitz\_in}(L_2), \Delta}\ \begin{array}{c}\forall x \in L_1 \exists y \in L_2 : \\ y \text{ is prefix of } x\end{array}$$

$$\frac{}{\Gamma, \mathsf{betriebsstandort\_in}(L_1) \Rightarrow \mathsf{betriebsstandort\_in}(L_2), \Delta}\ \begin{array}{c}\forall x \in L_1 \exists y \in L_2 : \\ y \text{ is prefix of } x\end{array}$$

$$\frac{}{\Gamma, \mathsf{oenace\_in}(L_1) \Rightarrow \mathsf{oenace\_in}(L_2), \Delta}\ \begin{array}{c}\forall x \in L_1 \exists y \in L_2 : \\ y \text{ is prefix of } x\end{array}$$

$$\frac{}{\Gamma, \mathsf{rechtsform\_in}(L_1) \Rightarrow \mathsf{rechtsform\_in}(L_2), \Delta}\ L_1 \subseteq L_2$$

$$\frac{\Gamma, D \Rightarrow \Delta}{\Gamma, \mathfrak{d} \Rightarrow \Delta}\ (\mathfrak{d}, D)_L \qquad \frac{\Gamma \Rightarrow D, \Delta}{\Gamma \Rightarrow \mathfrak{d}, \Delta}\ (\mathfrak{d}, D)_R$$

Fig. 7. The ground sequents and definition rules used in the calculus. In the definition rules the pair $(\mathfrak{d}, D)$ is a defined concept.

is true if the first argument contains a list $\Gamma_1 \Rightarrow \Delta_1, \; ..., \Gamma_n \Rightarrow \Delta_n$ of premisses, i.e., sequents, the second argument contains a sequent $\Sigma \Rightarrow \Pi$, and the third argument contains the list of premisses merged with this sequent, i.e., $\Gamma_1, \Sigma \Rightarrow \Pi, \Delta_1, \; \ldots, \Gamma_n, \Sigma \Rightarrow \Pi, \Delta_n$. Since the rules of the calculus are invertible, we could introduce prolog cuts `!` after the goals `rule(Rule_name, ...)` to increase efficiency – this would not influence completeness wrt. derivability of sequents. However, since this *would* limit the number of derivations found, and to preserve monotonicity of the program, we refrain from doing so.

The result of querying for logical implication between the formalised eligibility conditions of two grants is shown in Fig. 9. The prolog variable `Tree` is instantiated with a term for the derivation of the result abbreviated here for the sake of better readability.

The terms representing derivations can also be converted into human-readable form in a formalised natural language using Prolog's Definite Clause Grammars. The result is a string containing html-code which can be displayed in a browser, see Fig. 10.

```
rule(andL, [[A,B]=>[]] / and(A,B)=>[]).
rule(andR, [ [] => [A], [] => [B] ] / [] => and(A,B)).

prov2(Gamma => Delta, der(Rule_name, Gamma => Delta, [Fml] => [],
New_prems_ders)) :-
   select(Fml,Gamma,Omega),
   rule(Rule_name,Prems / Fml => []),
   merge_sequent_list(Prems, Omega => Delta, New_Prems),
   maplist(prov2,New_Prems, New_prems_ders).

prov2(Gamma => Delta, der(Rule_name, Gamma => Delta, [] => [Fml],
New_prems_ders)) :-
   select(Fml, Delta, Pi),
   rule(Rule_name, Prems / [] => Fml),
   merge_sequent_list(Prems,Gamma => Pi, New_prems),
   maplist(prov2, New_prems, New_prems_ders).
```

Fig. 8. Examples of the Prolog code for rules of the sequent calculus.

5.2 Interface Between Lisp and Prolog

To enable the reasoning functionality from within the Lisp-part of the PoC, the prolog prover is called and its output on `std_out` interpreted. For this, the implementation of a basic interface was necessary.

Conversion from Grant-Code in Lisp-Syntax to Prolog. Since the formalised grant conditions are given in the syntax of S-Expressions, they need to be converted to Prolog terms. While there is an existing project that transpiles S-expressions to ISO Prolog[12], it didn't fit our usecase; this library only allows batch processing and not the desired interactive querying, the already-parsed internal grant structure isn't supported, and a few special-cases demand a non-verbatim translation. In our implementation, negation and the typical infix operators `AND` and `OR` get printed out with parenthesis, to ensure the right precedence – the Prolog side ignores superfluous parens anyway.

Parsing Prolog Output. Custom parsing of Prolog output provides a nice special case: At the beginning of an output block, one or more lines containing a string beginning with the sequence `<html>` are recognized and displayed verbatim; this way a human-readable version of the derivations can be created in Prolog by nesting `<div>`s as necessary. Some CSS provided by the POC is then used by the browser to provide a nice visual display. Regular prolog output is parsed via the `ESRAP` library.

[12] https://github.com/cl-model-languages/cl-prolog2.

```
% förderung(F1, förderkriterien(K1)),
  förderung(F2, förderkriterien(K2)),
  dif(K1, K2),
  prov2([K1] => [K2], Tree),
  !.
F1 = "Per-Bundesland/Steiermark_Beratungskostenzuschuss-Für-Gastronomie-
      /Hotelleriebetriebe-In-Der-Steiermark",
K1 = ( df("Gv.At_Natürliche-Oder-Juristische-Person")
       or
       at(rechtsform_in(["Offene-Gesellschaft","Kommanditgesellschaft"]))
     )
     and
     ( at(önace_in(["55"]))
       or
       at(önace_in(["56"]))
     )
     and
     ( at(unternehmenssitz_in(["Land-Stmk"]))
       or
       at(betriebsstandort_in(["Land-Stmk"]))
     ),
F2 = "Per-Bundesland/Steiermark_Förderung-Zur-Wirtschaftsinitiative-
      Nachhaltigkeit-Steiermark",
K2 = at(unternehmenssitz_in(["Land-Stmk"]))
     or
     at(betriebsstandort_in(["Land-Stmk"])),
Tree = der(andL,...).
```

Fig. 9. Reasoning over grants. The query is shown at the top. The variables `F1` and `F2` are instantiated with the names of grants, such that the conditions `K1` of the first one imply the conditions `K2` of the second one. The variable `Tree` is instantiated with the derivation witnessing provability of the sequent `K1` $\Rightarrow$ `K2`, abbreviated here for the sake of space.

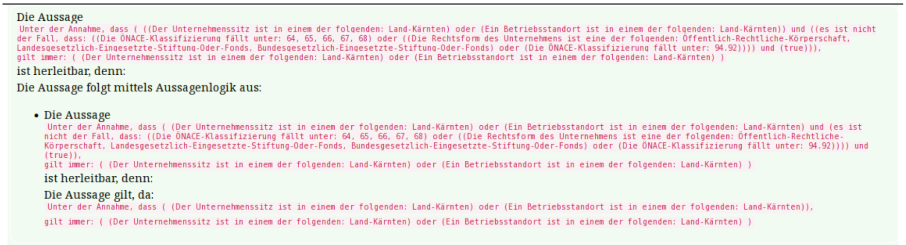

Fig. 10. HTML output of a Prolog reasoning.

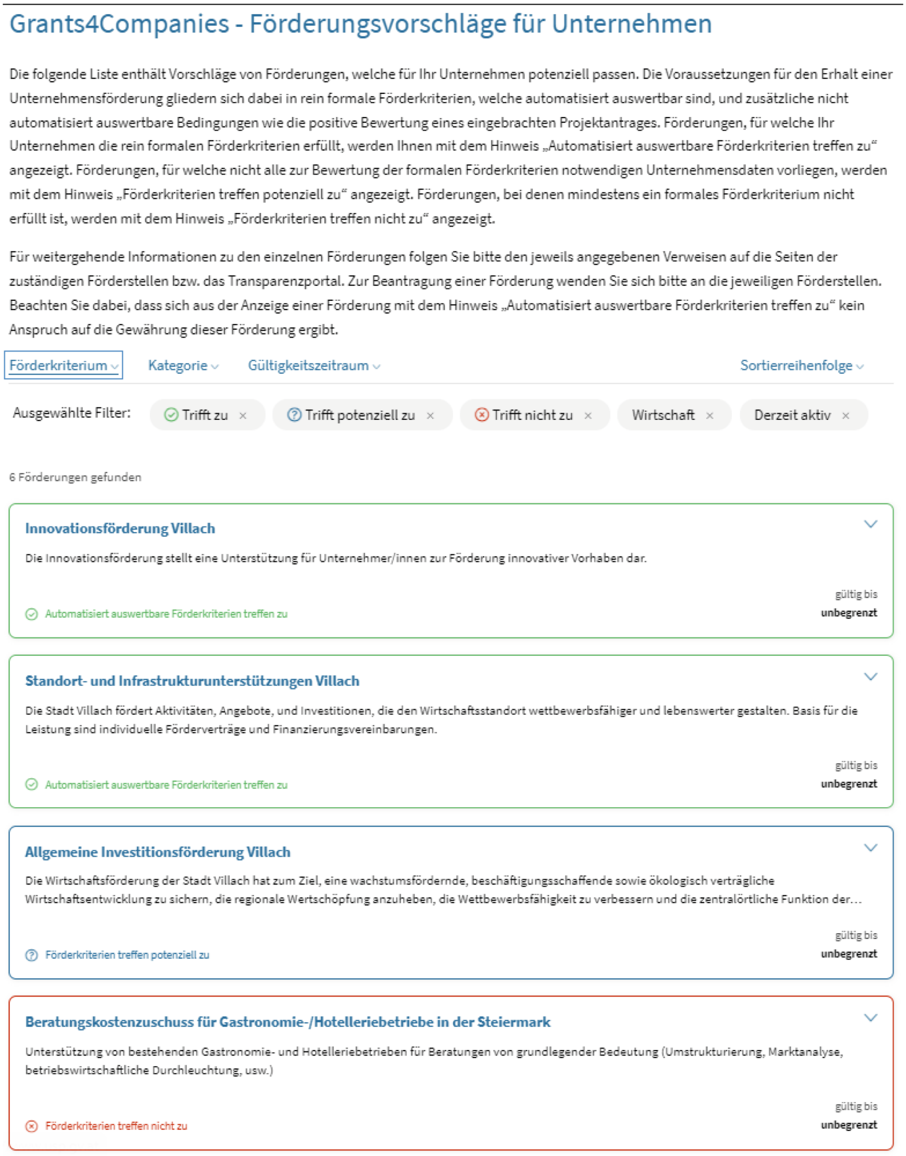

Fig. 11. The productive version of *Grants4Companies*. This figure shows the main page with a short description at the top and the list of grants. The grants are sorted with the applicable ones shown at the top of the list, the not applicable ones at the bottom, and the ones requiring further data for a conclusive evaluation in the middle. The list can be filtered, e.g., according to the topic of the grant, in this case "economy" ("Wirtschaft").

6 Conclusion and Outlook

We presented *Grants4Companies*, an application in the Austrian public administration, which uses declarative methods to recommend business grants based on the data available for the businesses from sources in the public administration. We also presented the Proof of Concept implementation of logical reasoning over the formalised grant conditions, implemented in Common Lisp and Scryer Prolog. A main interest here lies in the fact that the PoC implementation uses declarative and logical methods in the context of an application, which is already live in public administration.

In terms of future work we are steadily extending the list of covered grants, and are considering automatised rules extraction methods (e.g., [5,8]) for speeding up this process. Extending the coverage of the grants will necessitate the extension of the logical language and hence the reasoning mechanisms to further concepts and also towards (limited) reasoning with natural numbers. We envisage the resulting tool to become a possible basis for systematic analyses of the Austrian landscape of business grants by stakeholders in funding agencies and public administration.

References

1. Beckert, B., Posegga, J.: Logic programming as a basis for lean automated deduction. J. Log. Program. **28**(3), 231–236 (1996). https://doi.org/10.1016/0743-1066(96)00054-4. https://www.sciencedirect.com/science/article/pii/0743106696000544
2. Belzer, J., Holzman, A., Kent, A.: Encyclopedia of Computer Science and Technology: Volume 10 - Linear and Matrix Algebra to Microorganisms: Computer-Assisted Identification. Taylor & Francis (1978)
3. Kleene, S.C.: Introduction to Metamathematics. North-Holland, Amsterdam (1952)
4. Knuth, D.: Literate programming. Comput. J. **27**(2), 97–111 (1984)
5. Kovács, A., Gémes, K., Iklódi, E., Recski, G.: Potato: explainable information extraction framework. In: Proceedings of the 31st ACM International Conference on Information & Knowledge Management, CIKM 2022, pp. 4897-4901. Association for Computing Machinery, New York (2022). https://doi.org/10.1145/3511808.3557196
6. Mowbray, A., Chung, P., Greenleaf, G.: Representing legislative rules as code: reducing the problems of 'scaling up'. Comput. Law Secur. Rev. **48**, 105772 (2023)
7. Multlog: Analytic proof systems for strong Kleene logic K_3 (2022). pdf generated by MULTLOG, v.1.16a. https://logic.at/multlog. https://logic.at/multlog/kleene.pdf
8. Recski, G., Lellmann, B., Kovács, A., Hanbury, A.: Explainable rule extraction via semantic graphs. In: ASAIL 2021. CEUR Workshop Proceedings, pp. 24–35 (2021)
9. Troelstra, A.S., Schwichtenberg, H.: Basic Proof Theory. Cambridge Tracts In Theoretical Computer Science, vol. 43, 2nd edn. Cambridge University Press, Cambridge (2000)
10. European Union: Regulation (EU) 2016/679 of the European parliament and of the council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing directive 95/46/EC (general data protection regulation). OJL (2016)