Towards Protecting Vital Healthcare Programs by Extracting Actionable Knowledge from Policy

Vanessa Lopez[*a], Nagesh Yadav[b], Gabriele Picco[a], Inge Vejsbjerg[a], Eoin Carroll[b], Seamus Brady[b], Marco Luca Sbodio[a], Lam Thanh Hoang[a], Miao Wei[b], John Segrave[*b]

[a] IBM Research Europe, Dublin, Ireland
[b] IBM Watson Health, Dublin, Ireland
*Equal author contribution

Abstract

In challenging economic times, obtaining value for money by ensuring financial integrity and fairer distribution of services are among the top priorities for social and health-care systems globally. However, healthcare billing policies are complex and identifying non-compliance is often narrow-scope, manual and expensive. Maintaining 'integrity' is a challenge - ensuring that scarce resources get to those in need and are not lost to fraud and waste. Our approach fuses recent advances in dependency parsing with a policy ontology to convert the content of regulatory healthcare policy into human-friendly policy rules, that are amenable to machine-execution, with human oversight. We describe the ontology-guided transformation of textual patterns into a semantically-meaningful knowledge graph of rules, outline our experiments and evaluate results against policy rules obtained from professional investigators. The aim is to make a policy-compliance 'landscape' visible to healthcare programs - helping them identify Fraud, Waste or Abuse.

1 Introduction

The WHO (World Health Organization, 2010) lists fairness, financial integrity ["Program Integrity"] and access in healthcare among the top global healthcare priorities. In the U.S., an estimated annual amount of USD$20-30B is lost to **Fraud, Waste and abuse (FWA)** (Shrank et al., 2019). These vital funds never make it to the vulnerable citizens that they were intended to serve.

To combat this, countries with insurance-based healthcare programs (e.g. Medicaid, Medicare), employ claim investigators to validate the integrity of reimbursement claims submitted by providers. Investigators verify these claims against the program's policies, with the goal of reducing wasteful practices, identifying fraud or abuse and closing policy gaps. This is a labor-intensive task – claim volumes are high, policies are complex and investigative resources are limited.

More broadly, governments regulate a wide range of sectors, with extensive rules and policies. These policies drive significant spending by the regulated organizations. e.g., a European Union Commission study (2019) found that the annual cost of complying with EU financial regulations is around EUR€11.3B. In Australia, Deloitte (2014) estimated the costs of administering and complying with public sector rules at AUD$94B.

In this paper we propose a methodology for automatically extracting knowledge from healthcare policy documents, in the form of **Benefit Rules (BRs)** that are both human-understandable and machine consumable. These BRs can be applied automatically to flag discrepancies in claims, with limited human effort. This paper focuses on the extraction of these rules, and not their execution.

Policy rules have a major impact both on the health of the citizens they serve, and the financial integrity of the Programs that pay the Service Providers. Human-understanding, oversight and control are first-class AI-design concerns for this domain. Model 'explainability' is not enough. Transparency is needed anywhere that provider or citizen coverage is at stake. Users need to see and influence which rules are being applied, and on what policy basis decisions are being made.

To achieve this, we anchor our methodology in an ontology[1], that both guides and constrains AI extraction tasks. We build on recent NLP advances

[1] The ontology and Benefit Rules benchmark are released as open source at: https://github.com/IBM/rules_extraction_from_healthcare_policy

Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021, pages 3509–3521

to identify patterns in dependency paths that connect relevant entities.

Finally, we transform the dependent entities into knowledge-graph fragments, which are assembled into graphs that represent actionable Benefit Rules. Users can curate these rules as they are well-structured and expressed in familiar terms.

Automated extraction of these rules from high-volume policies (e.g., Medicaid), will enable the emergence of a new generation of tools for safeguarding Program Integrity, e.g. execution of these rules against claims data enables an overview of the 'policy-compliance landscape' that does not exist today.

Section 2 presents Rules as Code and related work in knowledge extraction for social good. Section 3 describes domain requirements. Section 4 presents the architecture for our 'Claim Audit' extraction pipeline. Section 5 describes how dependency parsing and the ontology are used to extract semantically-rich rule fragments. In Section 6 we evaluate our results with professional policy investigators using dental policies, and in Section 7, we present future work.

2 Related Work

2.1 Rules As Code (RaC)

Organizations need novel approaches to help with regulatory compliance, and **Rules as Code (RaC)** is an initiative that envisages *"an official version of rules (e.g., laws and regulations) in a machine-consumable form, which allows rules to be understood and actioned by computer systems in a consistent way"* (Mohun et al., 2020). It forms part of a broad movement towards digital government and has garnered broad public-sector interest.

Approaches to achieving machine-executable RaC rules for published legislation run from manual coding by multi-disciplinary teams to automatic code-generation from natural-language legislation (Mohun et al., 2020). The former approach brings legislative drafters, policy analysts and software developers together to co-produce human and machine-consumable versions of rules. Examples include the New Zealand Better Rules Discovery initiative (Digital Government NZ, 2018) and OpenFisca (OpenFisca.org, nd) in France. The latter approach uses NLP technology to assist policy experts in converting policy texts to machine-consumable forms, helping scale the RaC process, and is being explored by the AustLII's DataLex Project (Greenleaf et al., 2020) and CSIRO Data61's Regulation as a Platform project (Data61, 2019). Our work involves taking this latter approach for government healthcare insurance policy.

2.2 Knowledge Extraction for Social Good

Related to our work, Kiyavitskaya et al. (2018) extracts right, obligation, exception and constraints from legal documents by annotating entities with a domain-specific ontology consisting of entity vocabulary and normative phrasal templates built manually by domain experts. Dragoni et al. (2016) parses sentences into grammar trees using Stanford NLP, and annotates legal concepts with a manually-built ontology. The annotation is turned into rules using a set of hand-crafted rules.

Our work is similar, but we extract tuples from dependency trees and reason over the ontology to produce a Benefit Rule knowledge graph.

Recent efforts to build graphs for social good include (Assom, 2020) - constructing knowledge graphs to extract food-trading activities for sustainable food trading and security. Puri et al. (2020) discusses challenges in extracting knowledge graphs from UN datasets for sustainable development goals. Khetan et al. (2020) describes the use of NLP tools like Spacy, CoreNLP, ClausIE and OpenIE to extract information from unstructured text provided by the UN. Kejriwal et al. (2017) constructs a knowledge graph that supports a semantic search engine for investigators in human-trafficking.

AI for social good is a broad research topic as described in Shi et al. (2020). For public health, Nordon et al. (2019) uses biomedical knowledge graph for drug discovery. Finally, Percha and Altman (2018) connect entity-pair dependency paths to extract relations between chemicals, genes and diseases.

Our focus is on AI and knowledge-extraction to help combat FWA in healthcare programs. Today, FWA detection generally relies on two approaches. Firstly, traditional data mining to identify outliers and anomalous billing patterns in claims (Joudaki et al., 2015). While valuable, this approach presents challenges when building legal cases, as it is not innately grounded in policy. Secondly, hand-coded algorithms written by analysts to find claims that are not compliant with policy. These are labor-intensive to develop and maintain in the face of ever-evolving policy. Worse, they cover only a

small fraction of the policy 'landscape', resulting in a prioritization catch-22 - algorithms are needed in order to know which areas most need algorithms.

3 Background and Requirements

Despite the variety of ways that Benefit Rules (BRs) are expressed across healthcare policies, experts know the common entities, relationships and logical constraints that underpin them. To extract BRs that are correct, human-friendly and executable, we need to account for these semantics. Popular generic language models do not capture such implicit knowledge and expert-labeled datasets are expensive to develop (and small).

We apply these semantics via a BR ontology co-created with domain experts as described in (Lopez et al., 2019). The ontology guides the extraction of well-structured, consistent knowledge graphs from the policy. It links relevant entities together with their context in sentences, e.g., take the policy paragraph and its ontology subset shown in Figure 1. To extract the two distinct Benefit Rules shown in Figure 2, the two distinct roles of the service '*full-mouth debridement*' must be recognized. Widely-adopted medical terminology standards (e.g. UMLS, CPT, HCPCS) can be attached to ontology concepts for consistent representation and mapping of billing codes/values in a BR.

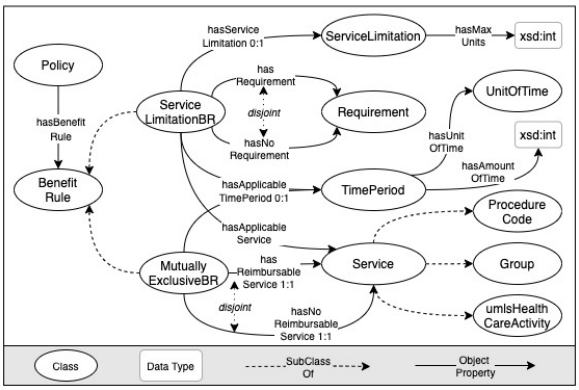

Figure 1: Ontology subset describing the policy: *Full-mouth debridement to enable comprehensive periodontal evaluation and diagnosis is a covered service and does not require prior approval. It is payable once in a 24-month period. Full-mouth debridement is not payable on the same date of services as other prophylactic or preventative procedures*

Human understanding and control are also key requirements. Policy can be challenging to interpret, and amenable to mis-interpretation or mis-application. Expert operators must be able to understand the policy provenance and correct any incorrectly-extracted BRs. The ontology powers this 'explainability' in two ways. Firstly, by enabling consistently-structured knowledge graphs to be extracted from inconsistent policy representations, removing ambiguities between the rules interpretation and policy intent. Secondly – by expressing them simply, using familiar user concepts. All our graphs can be presented as a set of simple, editable condition-value pairs, as in Figure 2. This is achieved by 'flattening' the graph - taking only the leaf properties and values. While our UI needs visual design, users report that these representations 'feel right' and that correcting extraction errors/omissions is straightforward.

Finally, extracted BRs must be amenable to execution - i.e., converted into a form that can automatically label claims as policy-compliant (or not). Here again, the ontology helps by mapping each condition to consistent constraints - this time for the selection, filtering and aggregation of claims data. While beyond the scope of this paper, our work to-date suggests that ontology-conforming BRs execute with similar accuracy to algorithms hand-written by claim investigators. (Of course, execution accuracy will still also depend on accuracy of automated extraction and human curation).

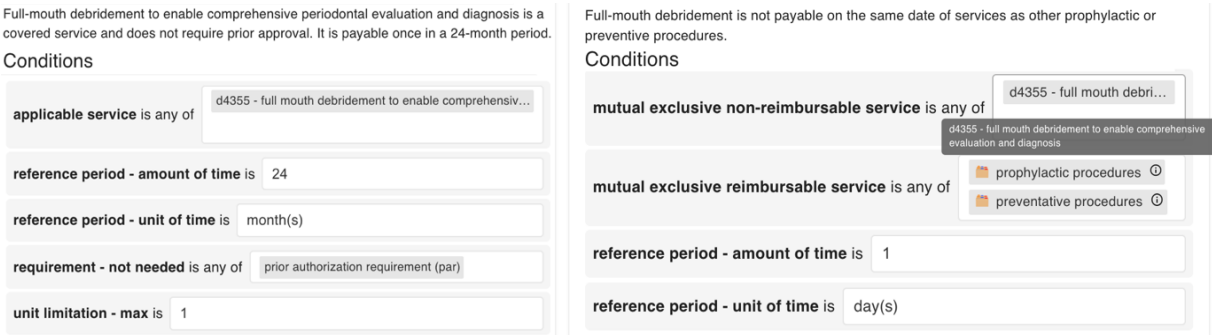

Figure 2: Benefit Rules extracted from a paragraph in a dental policy (DHS, 2013). On the left: a Service Limitation BR on the number of units a provider can bill for a service per patient over a period of time. On the right: a Mutually Exclusive BR on services that cannot be billed together in a given period

4 Proposed Extraction Pipeline

We propose a methodology to extract Benefit Rules from policy automatically. Figure 3 presents the main components, from policy enrichment to ontology-guided knowledge extraction and user validation. All steps in the pipeline are configurable - similar functional components can easily be replaced/added. We give an overview for each step, the functionality and data requirements:

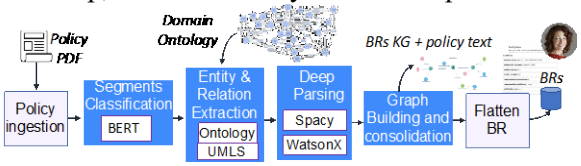

Figure 3. Pipeline from extraction of Benefit Rules

Data preparation and Policy ingestion: all domain information is captured in the ontology, so the technical components remain domain agnostic. While the ontology schema generalises across many policies, instance data is domain-specific and must be prepared ('lifted'). Some instances are common across policy areas and states, such as an eligible 'Place of Service' (e.g. *hospital)* (Centers for Medicare & Medicaid, n.d.). Other instance data is specific to the target domain, such as 'body parts' for tooth identifiers in dental claims. These can be automatically 'lifted' into the ontology from tabular data sources containing a main entity, a list of surface forms (to address vocabulary heterogeneity) and other attributes. They are added as individuals of a given entity type on application startup, and according to a user-configurable mapping. The lexicalizations are made available when processing policy text and identifying entity mentions. New instance data can be added as the need arises

Next, PDF policies are transformed into enriched HTML using an off-the-shelf conversion tool (compare and comply, n.d.), outlining headings, passages and paragraphs for later use at passage-level by annotators and extractors, as well as at paragraph-level by the classification and consolidation.

Segment Classification: a classifier optionally filters incoming paragraphs, deciding whether they are likely to contain BRs. This can significantly reduce compute-time and can also improve Precision, at the cost of some Recall. This is the only component in the pipeline that requires enough ground truth data on (validated) BRs and the associated paragraphs to fine-tune deep learning models. In section 6 we evaluate a fine-tuned, BERT-based classifier.

Entity & Relation Extraction: Here, candidate ontology entities/types are annotated in the text by two complementary annotators. The first is based on WatsonX (Kalyanpur et al., 2012) , a generic entity and UMLS-based clinical annotator. A Lucene search index is used to find approximate matches for any annotated entities (verbs, noun-phrases, etc.), in the ontology lexicalizations. It is also used to retrieve semantic types (e.g. diseases) relevant to benefit rules, from terminology services such as UMLS (UMLS semantic hierarchy, nd).

The second - SystemT (Chiticariu et al., 2018), extracts entity mentions from dictionaries and regular expressions. These are built automatically, from the instances 'lifted' into the ontology at initialization time, as described earlier. This enables entity mentions to be matched with complex labels, like '*Full-mouth debridement to enable comprehensive periodontal evaluation and diagnoses'*. All lexicalizations in the ontology are useful to address vocabulary heterogeneity.

Annotators label these textual spans with specific ontology labels (URIs) and other useful information (lemma, POS, UMLS type, etc). These annotations can later be used to simplify the dependency trees of sentences containing complex entities. Since there can be overlapping annotations and competing annotations for the same span, heuristics are used – e.g. 'longest span', or preferring exact matches to approximate ones. Disambiguation is otherwise performed later.

Deep Parsing and Graph Building: In this step, BR knowledge graphs are obtained by combining dependency trees together with the annotations, in an ontology-guided way. First, linguistic links are identified between annotated entities. Then, these are converted into semantic triples and linked together into knowledge graph fragments by reasoning over the ontology. Fragments from different sentences in a paragraph are **consolidated** together to produce a set of well-formed Benefit Rule knowledge graphs that respect the ontology semantics. Finally, to enable human oversight and control, all knowledge graphs are 'flattened' into a user-friendly, editable format. The ability to do this flattening is a key 'explainability' property of the ontology structure and thus, of knowledge graphs derived from it. The extracted and curated rules are then stored in a Knowledge Base.

User Validation: extracted BRs are shown to investigators in a prototype workbench. Here they can be reviewed against the corresponding policy text and corrected when necessary. Validated BRs form a shared store of high-quality, machine-readable rules, making the rule creation and consumption process more transparent. In a related work, we execute these rules on claims data to discover inappropriate payments. This shortens the investigator's workflow by giving them an immediate view of policy-relevant claims data (normally obtained through time-consuming data requests, spreadsheets and algorithm coding). It also grounds their work to specific policy clauses, helping them build a watertight case for recovery.

5 Knowledge Extraction Approaches

Dependency parsing has been frequently used to support relation extraction by capturing words that are close in context, even if far in sentence distance. Given a dependency tree, a set of subtree extraction rules identifies linguistically-connected terms, in the form of Predicate Argument Structure (**PAS**) tuples (Section 5.1). PAS tuples represent dependencies between textual entities, such as binary or ternary relations. They provide an easy intermediate representation to match text sentences to triples (subject, predicate, object) constituting a knowledge graph. The ontology can then be used to check if the linguistic tuples make sense semantically. Transformation of PAS tuples into Knowledge Graph fragments (sets of ontology triples) is done following a set of semantic templates (Section 5.2.).

To extract PAS tuples, we first implemented a deterministic baseline using WatsonX general-purpose deep parsing engine (Kalyanpur et al., 2012), which builds dependency trees for sentences. WatsonX provides a *Pattern Matching* library to characterize subtrees via handcrafted rules. Table 1 shows an example of two rules, based on the simple dependency tree in Figure 4. The rule assigns a *syntactic* role to tokens/spans in the sentence – e.g., 'subject', 'predicate' or 'object' (also referred to as 'slot types'). Other roles such as 'complement', can also be applied as necessary.

However, hand-coding these syntactic rules requires knowledge of computational linguistics. Our aim is for non-linguistic experts (e.g., application developers) to be able to apply this process to new domains. To reduce dependence on hard-to-acquire dependency parsing skills, we have developed an approach based on Spacy (Honnibal et al., 2020) for learning these rules from curated examples (Section 5.1). In Section 6, we compare performance of these learned rules to the baseline hand-coded rules.

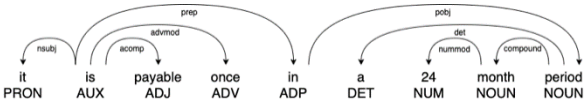

Figure 4. Dependency tree for an example sentence

pattern1→verb[hasPOS('verb'), hasLemma('be')] { nsubj -> **subj** [hasPOS("pron")] } { acomp-> **pred** [ ] } { advmod -> **comp** [hasPOS ("noun")] } { prep -> prepVar [ ] { pobj -> **obj** [hasPOS ("noun")] } } **pattern2** -> **subj** [hasPOS ("noun")] { compound -> **obj** [] { num_mod -> **comp** [] }}
PAS[pattern1]:subj=it,pred=payable,obj=once,comp=period PAS[pattern2]:subj=period, pred=[], obj=month, comp=24

Table 1: Example of written rules to extract PAS.

5.1 PAS Extraction Based on Learned Rules

Using a curated set of sentences with labelled PAS tuples, our framework generates dependency parsing rules that can obtain corresponding tuples from other, similarly-structured sentences. Specifically, we use Spacy's pattern builder to extract *Semgrex* patterns between fully-connected tokens, based on the shortest dependency path between tokens. This path usually contains the necessary information to identify their relation. Semgrex syntax allows us to characterize a subtree (Chambers et al., 2007), it describes nodes with normal token attributes, and how these nodes connect to other nodes in the dependency tree.

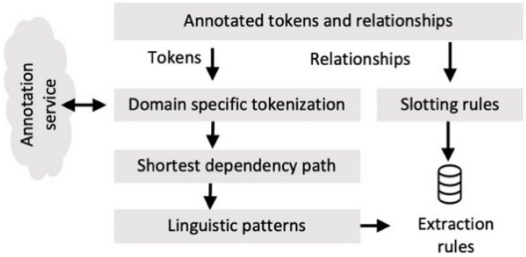

Figure 5. Extraction rule learning phase

We begin by obtaining a collection of sentences that represent the linguistic relationships we want to extract. We then annotate these sentences, identifying the interesting tokens and their syntactic roles (PAS 'slots'). Every annotated example is used to learn extraction rules, as depicted in Figure 5.

To learn the extraction rules for one curated sentence, we start by applying domain-specific tokenization, using the ontology-based entities

annotated during Entity Extraction. We then build a dependency tree for the sentence. These annotations help subsequent dependency parsing by letting the parser know when a complex, multi-token term (e.g., the service name "*Full-mouth debridement to enable comprehensive* ..") can be treated as a simple, single named-entity. The re-tokenization simplifies and lends a degree of consistency to the resulting dependency tree (Finkel and Manning, 2009). The dependency tree is then parsed to find the shortest dependency path between the interesting (annotated) roles/slots for a PAS. Once the subtree consisting of all the desired tokens for a PAS has been identified, we extract a linguistic pattern characterizing that subtree. In addition to the linguistic properties of the tokens, we also extract slotting rules. A slotting rule is used in conjunction with linguistic patterns to assign a syntactic role to an extracted token. This is based on (Choi and Palmer, 2012) where dependency labels can be assigned to arguments – i.e, they indicate the Slot type (subject, predicate, object, etc.) Finally, an extraction rule (linguistic -semgrex - pattern + slotting rules) is captured for every PAS in an annotated sentence.

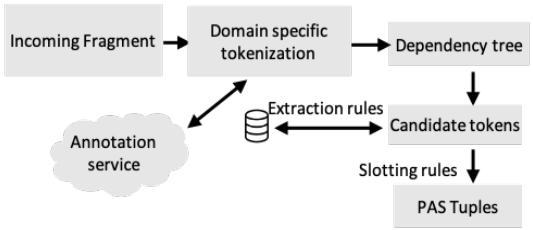

Figure 6. Runtime extraction of candidate PAS tuples

At runtime, incoming sentences are processed to extract PAS tuples (Figure 6). As before, domain-specific tokenization is applied to the sentence prior to building a dependency tree. The extraction rules learned earlier are then applied, to obtain candidate PAS tokens. Lastly, syntactic role labels are assigned to these candidate tokens by applying slotting rules.

5.2 Graph Building from PAS

PAS tuples enable us to extract meaningful relationships, even in text with challenging, long-range dependencies. However, PAS tuples require some translation to match ontology entities and relationships, e.g., a linguistic predicate may not directly translate to an ontological property. Implicit arguments may also be missing from a PAS. PAS can contain ternary relations that need to be aligned to one or more binary relations. And finally, not all PAS tuples are relevant.

To translate PAS tuples into semantically consistent Knowledge Graph fragments, we start by only keeping PAS that contain one or more ontology-based entity annotations. Then, for each subset of connected PAS tuples, we search for non-ambiguous semantic paths in the ontology that connect these entities, based on *parametrized* templates - implemented using the Jena API (Carrol et al., 2004). We use the annotated semantic types of the PAS entities (e.g.: class, instance, property, datatype, etc.) to select the templates to be executed. If a PAS token was annotated with more than one ontology annotation, then all combinations are tried. Non-relevant candidates will likely not yield any meaningful graph fragments. In here we provide an illustrative example of the process. Further details on the different templates can be found in Appendix C.

Consider the example in Figure 1, the first sentence yields the following PAS (among others):

```
<:d4355, :hasApplicableService, :Service>
<:d4355, :hasNoRequirement, :PAR>
```

The first PAS fires a template pattern that checks if the class *Service* is both the type of the instance *d4355* and the range of the object property *hasApplicableService*. If so, these entities are semantically connected and can be translated into the corresponding knowledge graph fragments. The following semantic triples are created:

```
:br1 rdf:type :ServiceLimitationBR
:br1 :hasApplicableService :d4355
```

For the second PAS, the range of the property *hasNoRequirement* corresponds to the type of the object instance *PAR*. This PAS links to the previous through the subject instance *:d4355* and can be translated into the knowledge graph fragment:

```
:br1 :hasNoRequirement :PAR
```

Thus, a knowledge graph is built by joining together all graph fragments obtained from the subset of connected PAS tuples.

Finally, a consolidation step pulls together the collection of graphs extracted from different portions of a policy paragraph. Here, all the constraints expressed in the ontology are enforced, e.g. *disjointness* between two properties, *min* and *max cardinality,* to ensure semantically-meaningful rules and discard nonsense rules (e.g. there can only be one applicable time period per BR – due to max cardinality = 1). Graph fragments are also merged, if the resulting BR graph does not

violate any ontological constraint. Since there can more than one extractor, any duplicate graphs derived from the same policy text are discarded, partial graphs can also be merged if doing so does not violate any ontological constraint.

Figure 7 shows the consolidated graph extracted from sentence 1 and 2. This graph can be 'flattened' to obtain human-readable 'Benefit Rules' – such as the Service Limitation rule shown on the left of Figure 2 and Mutually Exclusive rule shown on the right. Both subtypes inherit the properties of their parent Benefit Rule class, but are not merged, because each contains conditions that are only meaningful for that rule subtype, e.g., the property *hasServiceLimitation* is only relevant for a 'Service Limitation' rule.

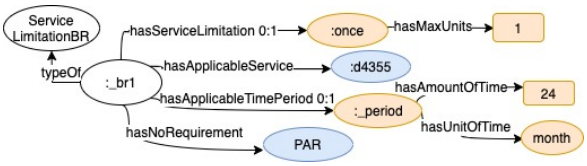

Figure 7. Consolidated graph from sentence 1 and 2

6 Evaluation Methodology

We concentrate on the system's ability to exploit the rich information contained in both a domain ontology and dependency trees with respect to a gold-standard created in consultation with our policy investigators. In particular, we aim to compare the impact of first addressing the need for dependency rules to extract PAS by proposing an approach that generates these rules from examples, which fit the domain-specific characteristics of new policy text. Policy-aware users can add these examples as there is no requirement to manually write new dependency parsing rules. Second, the impact of using a BERT-based classifier. Labeled training data is expensive to acquire as it requires domain expertise. However, as policy investigators review Benefit Rules, we investigated the use of this curated, small labeled dataset to fine-tune a classifier that filters out paragraphs that do not appear to contain any rules.

6.1 Set up: Data and Metrics

The proposed extraction pipeline was evaluated using Benefit Rules extracted from unstructured policies from two different states in the US. The ground truth of BRs for each policy was manually created by a team of three FWA investigators using our prototype User Interface: 90 rules were provided for State1 and 51 rules for State 2.

The ontology used in the experiments consists of 34 classes and 43 properties. Once the domain-specific instance data is 'lifted' in, 4954 individuals are added along with 23250 lexicalizations (i.e., labels used to annotate textual entities). The ground truth Benefit Rules presented in the experiments (not commercially sensitive or in production) are made available together with the ontology.

Each BR comprises of a policy text and a corresponding set of condition-values describing the text. Precision (P) measures the proportion of extracted rules that match the ground truth (GT) for the same policy text. Recall (R) measures the proportion of GT rules correctly extracted. F_1 combines the two. In addition, each matched rule gets a pairing 'score'. When all extracted condition-value pairs for a rule exactly match the GT, the score is 1. For partial matches, the score is between 0 and 1. A score of 0 indicates a missed rule. Table 2 shows the average pairing score across all rules. Details on the calculation and execution environment can be found in the appendixes A and B.

Jupyter notebooks showcasing the dependency tree (before and after tokenization) and PAS tuple extraction implementation with the learned rule for the example used in this paper, can be found in the git repository.

6.2 Results and Discussion

We run the evaluations using the two approaches presented for the PAS tuple extraction. The first – based on 54 manually-coded linguistic rules in WatsonX - acts as a baseline. The second are the learned syntactic rules described in Section 5.1. A total of 55 sentences were annotated with PAS examples.

State	Extractor	R	P	Avg score	F1
1	Learned rules	0.51	0.90	0.56	**0.65**
	Manual rules	0.48	0.73	0.57	0.58
2	Learned rules	0.45	0.82	0.41	0.58
	Manual rules	0.57	0.71	0.54	**0.62**

Table 2: Metrics when different extractors are applied over policies from different states

Results are summarized in Table 2. Overall, figures indicate that the proposed pipeline is a promising step towards automated extraction of BRs from unstructured policies. Extraction using learned

rules is comparable to using manual rules. Considering the number of curated examples and rules added to both extractors, there is a potential to further improve this by identifying missed rules and adding them as examples. This requires far less skill than manually hand-crafting linguistic extraction rules. We believe this takes a distinct step towards empowering development teams to tailor extraction to customer needs.

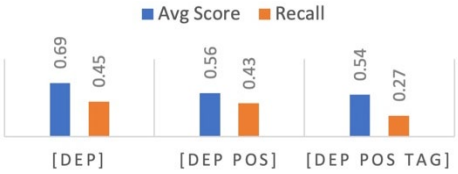

Table 3: Performance when various combinations of linguistic features are used to characterize subtree

Spacy provides several linguistic features on which to characterize a subtree, including syntactic dependency (DEP), part-of-speech (POS) and detailed part-of-speech (TAG). When settling on which of these to use (Table 3), we used Recall as the key metric and found that syntactic dependency (DEP) yielded best performance.

The use of a filtering classifier increases both performance (execution times shown in appendix A) and Precision (from 0.88 to 0.96 in State 1 and 0.82 to 0.95 in State 2) at the cost of a small drop in Recall (from 0.51 to 0.48 in State 1 and 0.45 to 0.41 in State 2). This is expected, since there are inevitably some false negatives in the classification. The model used is a BERT-based text classifier, fine-tuned on 70% of the paragraphs of the two policies, with the remaining 30% used for validation and early stopping during training. The hyperparameters were selected using Optuna (Takuya et al., 2019) optimization framework using 5-fold cross validation settings. The generalization capacity of the model is difficult to assess due to the small amount of labelled ground truth. With that caveat, it has been verified with cross-validation, where the model obtains an average accuracy of 96% on the various folds. The training data size is currently small due to the cost of manual policy labelling. However, it is expected this will expand as users review and curate extractor output.

While these benefit rule extraction results are promising, there is clearly scope to enhance our models and extractors to improve coverage. Most Benefit Rules are self-contained across one or more sentences in a paragraph, However, further work is needed to automatically capture the knowledge from headings, tables or co-references that span paragraphs. Improvements in service annotation could also reduce the incidence of partially-extracted rules being discarded during consolidation. Of course, there can always be implicit information, available in the minds of policy consumers but not present in the policy text for extractors to see. For example, the meaning of *'fair to good'* expressed in the rule *"Restorative services are payable when there is a fair to good prognosis for maintaining the tooth"*. While these details cannot currently be extracted automatically, our system gives policy analysts the ability to capture them in other consistent ways. For example, simple options like attaching a 'medical necessity' label can make cases like these amenable to machine learning and offer real benefits to Program Integrity workers seeking to size and prioritize work.

Various approaches can be used to look beyond co-occurrence of entities in sentences and explore how the terms are linguistically and semantically connected. For instance (Roth and Lapata, 2015) (Lopez et al., 2019) use semantic role labeling to identify actions and roles in a sentence (agent, theme, polarity, etc.) and reason over these to expose relation-entity/value pairs. Other complementary extraction approaches can be leveraged to look beyond co-occurrence of entities in the sentence and explore how the terms are linguistically and semantically connected, which could improve further on rule extraction coverage, such deep learning models. While the training data set is currently small, due to the cost of manual policy labeling, it is expected this will expand as users review and curate the extractors' output.

In this paper, we have explored building over dependency PAS, which can be exploited to capture fine-grained and distant relationships. Using a 'learned-rule' approach to obtain intermediate representations from a sentence enables non-linguistic experts to extend this process to more policies, without requiring manual production of syntactic-rules to address syntactical variability.

We made a small analysis to quantify the similarity between our two dental policy texts P_1 and P_2. We extracted sentences (removing stop words) from P_1 and P_2 using a standard sentence tokenizer (NLK Tokeniser, n.d.). Following (Reimers et al., 2019), we computed embeddings for each sentence using a model optimized for

Semantic Textual Similarity (STS) (specifically, we used stsb-roberta-large -STSb performance:86.39 (SBert, n.d.)). Let Ei be the list of sentence embeddings for the sentences of policy Pi. We have computed pairwise cosine similarity between E_1 and E_2. Let E_{12_max} be the list of the maximum values of the cosine similarity between each embedding in E_1 and all the embeddings in E_2 (the i-th value of E_{12_max} gives the best cosine similarity between the i-th sentence in P_1 and some sentence in P_2). We found that the mean of E_{12_max} is 0.65 with a standard deviation of 0.12; about 45% of the sentences in P_1 have a cosine similarity score with some sentence in P_2 that is above the average value. We repeated the same experiment using a third policy text from a different domain P_3, and found that the similarity scores for sentences in P_1 with respect to P_3 were consistently lower than between P_1 and P_2. This small experiment gives an indication that policy texts from different states but in the same domain (e.g., dental) have a good degree of similarity. The Benefit Rule Ground Truth built for each of the two policies can be seen in the git repository, each Benefit Rule shows the policy paragraph of text from where it is extracted and condition-values.

7 Conclusions and Future Work

Within regulated organizations, visibility over any part of the 'compliance landscape' is valuable information for identifying Program Integrity risks and prioritizing follow-up. With healthcare insurance policies, typically only a small fraction policy is automated (hand-coded), and the rest of the landscape remains opaque. So, unlike many AI applications that require high Recall before they are useful, any Recall here provides concrete, actionable value that claim workers can use both for prioritization and for follow-up. In a dark room, all light helps.

The ontology abstraction supports transferability, e.g. from one US state to another that semantically express similar compliance rules concepts or conditions (e.g.: eligible members, places of service, maximum billable units of service, services that should not be billed together, etc.) even if the wording differs between the texts. It also supports incremental enhancement to cover more rules and/or policy areas, which enables a scalable market. Dependency trees capture fine-grained, distant connections, which guided by the ontology are used to automatically build and consolidate a semantically meaningful Knowledge Graph. This increases precision by allowing for paragraphs that mention relevant entities, but which don't contain Benefit-Rules, to be discarded. If syntactical variability is high, using a 'learned rule' approach to obtain PAS representation enables non-linguistic experts to apply this process to more policies without requiring manual production of syntactic rules.

We see interesting avenues for follow-up NLP / AI work that combines neural and symbolic approaches, e.g., training deep learning models as our users validate more policy rules, to recognize rule fragments (spans), then assemble them into rules using the ontology as a blueprint. We also plan to perform user-based evaluations to measure how our pipeline impacts the time taken by investigators to identify and resolve FWA leads, as well as the quality and scope of the compliance information provided.

Finally, while our methodology is shown in a Healthcare setting, we believe it is applicable to other regulated domains, such as Finance. These extracted computable policy representations have the potential to open up many new opportunities – from reducing compliance costs through automated checking (particularly where common regulations apply to many) to 'what-if' analyses of proposed policy to automatically identifying gaps/loopholes that undermine Program Integrity.

Acknowledgments

We would like to acknowledge our business developers, offering managers, subject matter experts and accredited fraud investigators that make this project and its evaluation possible: Morten Kristiansen, Conor Cullen, Denisa Moga, Jillian Scalvini, John Davis, Shannon Ware, Mark Gillespie and Mark Goodhart.

Ethics / Broader Impact Statement

Governments must routinely automate policy rules in order to deliver services at population-scale, and this brings opportunities for both good and harm. Consistent application of policy rules at population-scale ensures fairer distribution of healthcare and social care services, as well as defending limited resources from Fraud, Waste and Abuse. (This latter point is often missed, but is critical to ensuring that resources are available when vulnerable people need them). At the same

time, defects in the code increase the potential for failure to reimburse healthcare providers for delivering necessary services. Getting from policy, to business requirements, to coded rules is a long, multi-translation process, with error and omission failure modes at every step. Gaps or biases in the original policy may also remain unnoticed through this long journey, only to be discovered at the end when vulnerable people are impacted.

A recent global movement known as '*Rules as Code'* (Mohun et al., 2020) identifies several methods to tackle this. The one our system uses involves extracting rules from policy text via NLP, to minimize translation steps. There is potential for misuse here, should machine-extracted rules be executed blindly, without checking they faithfully represent policy intent. Hence, our system treats human-in-the-loop oversight as an essential, non-optional part of the process. Together - automated extraction plus human oversight have the potential to reduce translation failures, as well as enable discovery of *policy* errors and biases (by facilitating earlier testing and iteration).

Further to this, we believe that effective oversight demands more than a review process or 'AI explainability' add-on. It requires human-understanding and ability-to-correct to be first-class design goals. To this end, we use an ontology to represent extracted rules in a form (Figure 2) that policy-aware users find familiar, understandable and correctable (as well as traceable back to their policy origin). This representation is at the heart of the system. When presented side-by-side with the policy text to a reviewer, both success and failure scenarios are clearly visible. For example, in a failure scenario (ontology does not model the policy well), few rules are extracted, and rule conditions are missing. Here the reviewer can discard the rule, or fill-in the missing information. Whether well-formed or poorly formed, at no point are rules blindly applied to citizen data automatically.

We believe that shifting the focus towards validation of policy and its digital expression will facilitate the production of better-quality, more humane policy.

References

Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. 2019. Optuna: A Next-generation Hyperparameter Optimization Framework. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, Association for Computing Machinery, pages 2623-2631. https://doi.org/10.1145/3292500.3330701

American Medical Association. 2000. *Current procedural terminology: CPT 2001*. American Medical Association, Chicago, IL.

Luigi Assom. 2020. Impact of Global food Trade: Food tractability and sustainability indicators. In *1st Workshop on Knowledge Graphs for Social Good*. Data retrieved from https://knowledgegraphsocialgood.pubpub.org/schedule

Olivier Bodenreider. 2004. The Unified Medical Language System (UMLS): integrating biomedical terminology. *Nucleic Acids Research*. 32(1) (Database issue)):D267-70. https://doi.org/10.1093/nar/gkh061

Jeremy Carroll, Dave Reynolds, Ian Dickinson, Chris Dollin, Andy Seaborne, Kevin Wilkinson. 2004. Jena: Implementing the Semantic Web Recommendations. In *Proceedings of the 13th International World Wide Web Conference on Alternate Track Papers &* Posters, Association for Computing Machinery , pages 74-83. https://doi.org/10.1145/1013367.1013381

Centers for Medicare & Medicaid Services (U.S.). 2020. *Healthcare Common Procedure Coding System (HCPCS)*. Centers for Medicare & Medicaid Services.

Centers for Medicare & Medicaid Services (U.S.). n.d. *Place of Service Codes*. Data retrieved from https://www.cms.gov/Medicare/Coding/place-of-service-codes/Place_of_Service_Code_Set

Nathanael Chambers, Daniel Cer, Trond Grenager, David Hall, Chloe Kiddon, Bill MacCartney, Marie-Catherine de Marneffe, Daniel Ramage, Eric Yeh, and Christopher D. Manning. 2007. Learning alignments and leveraging natural logic. In *Proceedings of the ACL-PASCAL Workshop on Textual Entailment and Paraphrasing*. Association for Computational Linguistics, pages 165– 170.

Compare and Comply. n.d. https://cloud.ibm.com/docs/compare-comply

Jinho D. Choi, and Martha Palmer. 2012. Guidelines for the clear style constituent to dependency conversion. *Technical report 01-12*, Institute of Cognitive Science, University of Colorado Boulder, Boulder CO, USA.

Laura Chiticariu, Marina Danilevsky, Yunyao Li, Frederick Reiss, Huaiyu Zhu. 2018 SystemT: Declarative text understanding for enterprise. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language*

Technologies, Volume 3 (Industry Papers). Association for Computational Linguistics, pages 76-83. http://dx.doi.org/10.18653/v1/N18-3010

Data61. 2019. *Case study on CSIRO's Data61, Australia: Contribution to the OECD TIP Digital and Open Innovation project*. Commonwealth Science and Industrial Research Organisation (CSIRO) Data 61. Data retrieved from https://www.csiro.au/en/News/News-releases/2019/OECD-spotlights-CSIROs-Data61-as-global-blueprint-for-digital-innovation

Deloitte. 2014. Get out of your own way – Unleashing productivity. *Building the Lucky Country: Business imperatives for a prosperous Australia* series. Data retrieved from https://www2.deloitte.com/au/en/pages/building-lucky-country/articles/get-out-of-your-own-way.html

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Association for Computational Linguistics, pages 4171–4186. http://dx.doi.org/10.18653/v1/N19-1423

Digital Government NZ. 2018. *Better Rules for Government Discovery Report*. Digital Government New Zealand. Data retrieved from https://www.digital.govt.nz/dmsdocument/95-better-rules-for-government-discovery-report/

Mauro Dragoni, Serena Villata, Williams Rizzi, and Guido Governatori. 2016. Combining NLP approaches for rule extraction from legal documents. In *Proceedings of the 1st Workshop on MIning and REasoning with Legal texts (MIREL 2016) collocated with the 29th International Conference on Legal Knowledge and Information Systems*, IOS Press, pages 1–13.

William D. Eggers, Mike Turley, Pankaj Kishnani. 2018. The Regulator's New Toolkit: Technologies and Tactics for Tomorrow's Regulator. *Reducing compliance costs with RegTech Deloitte Insights*. Data retrieved from https://www2.deloitte.com/us/en/insights/industry/publicsector/reducing-compliance-costs-with-regtech.html

European Commission. 2019. *Study on the costs of compliance for the financial sector*. Publications Office of the European Union. Data retrieved from https://op.europa.eu/en/publication-detail/-/publication/4b62e682-4e0f-11ea-aece-01aa75ed71a1

Jenny Rose Finkel and Christopher D. Manning. 2009. Joint parsing and named entity recognition. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*. Association for Computational Linguistics, pages 326-334.

Graham Greenleaf, Andrew Mowbray and Philip Chung. 2020. Strengthening Development of Rules As Code: *Submission to the OECD's OPSI on Cracking the Code*. Australasian Legal Information Institute (AustLII). http://dx.doi.org/10.2139/ssrn.3638771

Matthew Honnibal, Ines Montani, Sofie Van Landeghem, and Adriane Boyd. 2020. *spaCy: Industrial-strength Natural Language Processing in Python*. Explosion AI. https://doi.org/10.5281/zenodo.1212303

Iowa Department of Human Services (DHS). 2013. *Dental Services Provider Manual*. Data retrieved from https://dhs.iowa.gov/sites/default/files/Dental.pdf?012520210943

Hossein Joudaki, Arash Rashidian, Behrouz Minaei-Bidgoli, Mahmood Mahmoodi, Bijan Geraili, Mahdi Nasiri and Mohammad Arab. 2015. Using data mining to detect health care fraud and abuse: a review of literature. *Global journal of health science*, 7(1):194–202. https://doi.org/10.5539/gjhs.v7n1p194

Aditya Kalyanpur, Branimir Boguraev, Siddharth Patwardhan, J. William Murdock, et al. 2012. Structured data and inference in DeepQA. *IBM Journal of Research and Development*, 56(3):10. https://doi.org/10.1147/JRD.2012.2188737

Mayank Kejriwal and Peter Szekely. 2017. Knowledge Graphs for Social Good: An Entity-centric Search Engine for the Human Trafficking Domain. In *IEEE Transactions on Big Data*, https://doi.org/10.1109/TBDATA.2017.2763164

Vivek Khetan. 2020. SDGs and Knowledge Graph Extraction from Unstructured text. In *1st Workshop on Knowledge Graphs for Social Good*. Data retrieved from https://knowledgegraphsocialgood.pubpub.org/schedule

Nadzeya Kiyavitskaya, Nicola Zeni, Travis D. Breaux, Annie I. Antón, James R. Cordy, Luisa Mich, and John Mylopoulos. 2008. Automating the Extraction of Rights and Obligations for Regulatory Compliance. In *Proceedings of the 27th International Conference on Conceptual Modeling (ER '08)*. Springer-Verlag, pages 154–168. https://doi.org/10.1007/978-3-540-87877-3_13

KPMG. 2018. There's a revolution coming - embracing the challenge of RegTech 3.0. Data retrieved from https://home.kpmg/content/dam/kpmg/uk/pdf/2018/09/regtech-revolution-coming.pdf

Vanessa Lopez, Valentina Rho, Theodora Brisimi, John Segrave-Daly, Morten Kristiansen and Fabrizio Cucci. 2019. Benefit graph extraction from healthcare policies. *In Proceedings of the International Semantic Web Conference.*

James Mohun, Alex Roberts. 2020. Cracking the code: Rulemaking for humans and machines. *OECD Working Papers on Public Governance*, No. 42, OECD Publishing, Paris, https://doi.org/10.1787/3afe6ba5-en.

NLTK tokeniser, n.d., https://www.nltk.org/api/nltk.tokenize.html

Galia Nordon, Gideon Koren, Varda Shalev, Eric Horvitz, and Kira Radinsky. 2019. Separating wheat from chaff: Joining biomedical knowledge and patient data for repurposing medications.

OpenFisca. .n.d. Homepage, OpenFisca, https://fr.openfisca.org/ (Accessed 27 January 2021)

Bethany Percha and Russ B. Altman. 2018. A global network of biomedical relationships derived from text. *Bioinformatics*, 34(15):2614-2624. https://doi.org/10.1093/bioinformatics/bty114

Colin Puri. 2020. Building A Simple Knowledge Graph with UN Data: Quick Start Example, Common Methodologies, and Tooling. In *1st Workshop on Knowledge Graphs for Social Good.* https://knowledgegraphsocialgood.pubpub.org/schedule

Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. *In Proceedings of EMNLP/IJCNLP.*

Michael Roth and Mirella Lapata. 2015. Context-aware Frame-Semantic Role Labeling. *Transactions of the Association for Computational Linguistics* (3):449-460 http://dx.doi.org/10.1162/tacl_a_00150

Zheyuan Ryan Shi, Claire Wang and Fei Fang. 2020. *Artificial Intelligence for Social Good: A Survey. Computing Research Repository,* ArXiv, abs/2001.01818.

SBERT Sentence Embedding Models, n.d., https://www.sbert.net/docs/pretrained_models.html

William H. Shrank, Teresa L. Rogstad, and Natasha Parekh. 2019. Waste in the US Health Care System: Estimated Costs and Potential for Savings. *JAMA*, 322(15):1501-1509. https://doi.org/10.1001/jama.2019.13978.

SpaCy Pattern Builder (nd). Accessed January 30 2020 from https://pypi.org/project/spacy-pattern-builder/

World Health Organization. 2010. Health System Financing. The path to universal coverage. Chapter 4: More health for the money. *The World Health Report 2010*. https://www.who.int/whr/2010/en/

Supplementary Material

Appendix A. Evaluation environment

Our experiments were run using multiple microservices representing the modules described in the paper. Kubernetes was used to orchestrate the deployment of containers (microservices) in a cloud infrastructure with approximately 16 CPUs, 16 GB of RAM, and 64 GB of disk space.

The processing time of the policy for state 1 (27 pages) is approximately 74 minutes using the "learned rules" approach, while if using the classifier, the time is reduced to 41 minutes. The "manual rules" approach instead takes 42 minutes without classifier and 23 minutes with the classifier. For the policy of state 2 (35 pages), the times of the "learned rules" approach with and without classifier are 104 minutes and 35 minutes respectively. For the "manual rules" approach the processing time with and without classifier are 70 minutes and 10 minutes respectively.

Appendix B. Definition of metrics

Precision (P) measures the proportion of extracted rules that match the GT. Recall (R) measures the proportion of GT rules correctly extracted. f_1 combines these two. Specifically, they are defined as follows:

$$\boldsymbol{Precision} = \frac{\boldsymbol{n°exact\ matches + n°\ partial\ matches}}{\boldsymbol{n°\ extracted\ rules}}$$

$$\boldsymbol{Recall} = \frac{\boldsymbol{n°exact\ matches + n°\ partial\ matches}}{\boldsymbol{n°\ GT\ rules}}$$

$$\boldsymbol{f_1} = 2 * \frac{\boldsymbol{P * R}}{\boldsymbol{P + R}}$$

For a pair consisting of a ground truth Benefit Rule (BR) and an output BR we calculate a similarity score as follows, assuming each BR is a list of conditions c_i with corresponding values v_i taken out of a set with cardinality C_i.

For every pair of partial matches BR (R_j^{GT}, R_i^E), a similarity score s_{ji} is calculated based on:

$$s_{ji} = \frac{\min(L_i,L_j)}{\max(L_i,L_j)} * \frac{1}{L_j} * \sum_{k=1}^{L_j} score_{c_k}$$

where L_j and L_i correspond to the sizes of R_j^{GT} and R_i^E (i.e., how many conditions each BR consists of), and $\frac{\min(L_i,L_j)}{\max(L_i,L_j)}$ represents a penalizing factor when the sizes of the two BR are not the same (rule length similarity).. The score $score_{c_k}$ for each condition value pair $\{c_k: v_k\}$ is calculated as:

$$\begin{cases} 0, & if\ c_k\ is\ captured\ in\ R_j^{GT}, but\ not\ in\ R_i^E \\ 1, & if\ c_k\ is\ captured\ in\ both\ R_j^{GT}, R_i^E\ and\ v_k is\ the\ same \\ 0.5 + 0.5 * f_1 * \left(1 - \frac{1}{Ca_k}\right), & if\ c_k\ is\ in\ R_j^{GT}, R_i^E\ and\ v_k differ \end{cases}$$

Here, f_1 is the harmonic P-R mean generated by comparing the values of c_k in R_j^{GT} and R_i^E and $\{Ca_k\}$ is the number of semantically compatible candidate values a condition may have in the ontology (i.e., instances of the same type, such as all known medical programs), for datatypes where $Ca_k\ is\ 1$

Appendix C. Semantic patterns - templates

For each PAS tuple containing at least two entities annotated with the ontology model (as classes, instances, properties or datatype), consistent semantic statements (subject, predicate, object) are created to build the knowledge graph. Connections are found across the entities according to their semantic types and the templates described in Table 1. Different templates are executed for each meaningful combination of candidate matches in the PAS.

Table 4: semantic templates to check if entities of the given types can be semantically connected. They consist of parameters to substitute by the candidate entities of the type sought - a class, property, instance or datatype (in between <>) - and variables (preceded by '?') that must bind to an ontological resource. If all the constraints apply (that is semantically consistent), the pattern executes to build new statements, creating anonymous resources (blank nodes) as needed.

Pattern 1: <class, object property, instance>
• <class> is one of the domains of <property> • <instance> type is on the range of <property> If consistent, create a blank resource _:b such: _:b rdf:type <class>. _:b <property> <instance>
• <class> is in the range of <property> • <class> is the type of <instance> (or a superclass, that is <instance> rdf:type <class>) • <obj property> has a domain ?domain If consistent, create a blank resource _:b such: _:b rdf:type <domain>. _:b <property> <instance>.
Pattern 2: <class, data property, datatype>
• <class> is one of the domains of <property> • the range of <property> is consistent with the <datatype> (e.g., string, numerical, currency) If consistent, create an blank resource _:b such: _:b rdf:type <class>. _b: <property> 'datatype'
Pattern 3: <class, object property, datatype>
• <class> is one of the ranges of <property> • <class> is the domain of a ?property2, which range is compatible with <datatype> • <property> has a domain ?domain If consistent, create two blank resources _:bi such: _:bx rdf:type <domain>. _:bx <property> :_by. _:by rdf:type <class >. _:by <property2> <datatype>.
Pattern 5: <instance, data property, datatype>
• the <instance> type is one of the domains of <property> • the range of <property> is compatible with <datatype> If consistent, create a new statement such: <instance> <property> 'datatype'
• <property> has a range consistent with the datatype • <property> has a domain ?domain • ?domain is the domain of ?property2, which range ?range is compatible with the type of < instance> If consistent, create a blank resource _:b such: _:b1 <property> 'datatype'. _:b1 rdf:type <domain> _:b1 <property2> <instance>
Pattern 6: <instance1, object property, instance2>
• <property> has <instance2> as value • <property> has a domain ?domain • ?domain is the domain of ?property2, which range ?range is compatible with the type of < instance> If consistent, create a blank resource _:b such: _:b <property> <instance2>. _:b rdfs:type <domain>. __:b <property2> <instance2>
Pattern 7: <class1, object property, class2>
• <class1> is the domain of <property> • <class2> is the range of <property> If consistent, create a blank resource _:bi such: _:bx rdf:type <class1>. _:by rdf:type <class2> _:bx <property>_:by.
Pattern 8: <class, instance1, instance2>
• <class> is the domain of ?property1 and ?property2 • ?property1 has as range the type of <instance1> • ?property2 has as range the type of <instance2> If consistent, create a blank resource _:b1 such: _:b1 rdf:type <class>. _:b1 <property1> <instance1>. _:b1<property2> <instance2>