*August 2022*

# An End-to-End Pipeline from Law Text to Logical Formulas

Aarne Ranta [a] Inari Listenmaa [c] Jerrold Soh [c] Meng Weng Wong [c]

[a] *Department of Computer Science and Engineering, Chalmers University of Technology and University of Gothenburg, aarne.ranta@cse.gu.se*

[b] *SMU and Digital Grammars*

[c] *Yong Pung How School of Law, Singapore Management University*

**Abstract.** This paper develops a pipeline for converting natural English law texts into logical formulas via a series of structural representations. The goal is to show how law-to-logic translation can be achieved through a sequence of simple, well-defined steps. The texts are first parsed using a formal grammar derived from light-weight text annotations designed to minimize the need for manual grammar construction. An intermediate representation, called assembly logic, is then used for logical interpretation and supports translations to different back-end logics and visualisations. The approach, while rule-based and explainable, is also robust: it can deliver useful results from day one, but allows subsequent refinements and variations. While some work is needed to extend the method to new laws, the software presented here reduces the marginal effort necessary. As a case study, we demonstrate our approach on one section of Singapore's Personal Data Protection Act. Our code is available open-source.

**Keywords.** parsing law text

## 1. Introduction

Expressing laws computably is a classic objective of AI and Law [22,34] and a crucial pre-requisite to automating downstream tasks such as compliance checking [26,16], policy support [35,15], information retrieval [6], argumentative reasoning [24], legislative simulation [5,4], and formal verification [15]. But faithfully translating law to logic is challenging, often requiring rare expertise in both legal and formal methods. This "natural language barrier" [23] poses a significant "knowledge bottleneck" [25] to the development of legal expert systems and knowledge bases. Thus in last few decades, researchers have devised numerous strategies for bridging the barrier. These include domain-specific ontologies [27], taxonomies [17], vocabularies [16], standards [28], logics [29], markup languages [3] and programming languages [18], intermediate formalisms for expressing laws [21,20,23], and specialised human workflows [27,37].

Early in the field's history, [6] had already imagined automatic parsers for translating natural language laws into formal logic programs. Several significant

steps have since been taken towards that vision. For instance, McCarty [23] demonstrated how [9]'s statistical parser can extract, from judicial opinion texts, syntax trees which can be further converted into quasi-logical semantic representations of said texts through definite clause grammars implemented in Prolog. Others have examined how far statistical, machine, and deep learning methods are capable of implicitly representing legal principles and concepts [14,10,36,7,8].

In previous literature, the bottleneck has often been the chosen NLP framework [30,38].xs This paper uses Grammatical Framework (GF, [32]). GF has had some earlier uses in the Law domain, e.g. [2,13], typically based on Controlled Natural Language (CNL); see [11,1]. The present approach is novel in addressing uncontrolled real-world law text. But it exploits the same features of GF as have shown fruitful in CNL processing: modularity, precision, and support to semantic back ends via abstract syntax.

As a downside, writing GF grammars requires expertise that is rare in the field of computational law. In order to ease the adoption of GF, we have developed a method to automatically extract a preliminary GF from light-weight annotations that non-programmers can add to texts. The automatically extracted GF grammar is already usable for a rough analysis of law texts. It can also be gradually improved by some manual work, which requires less effort than writing grammars from scratch.

We exemplify the pipeline using a part of Singapore's *Personal Data Protection Act* 2012 (PDPA). Our code is available open-source.[1]

Section 2 details our pipeline. Section 3 explains how we used it to formalise the PDPA. Section 4 discusses future directions and concludes.

## 2. Methodology

### *2.1. Pipeline Overview*

The structure of the pipeline is shown in Fig.ẽfpipeline, while Fig.ẽfpipeline-ex shows a concrete example of its use on a paragraph of text.

The pipeline's input is a statutory text in natural language, which we assume has been properly extracted and stored in a standard text format (e.g. .txt). We also assume it to be stored line by line and tokenized by some standard tool, such as GF's `lextext` converter. The resulting text is converted to **abstract syntax trees** line by line using the GF parser driven by a grammar. The baseline is a context-free grammar, which is automatically derived from the text itself. At a later phase, we can optionally replace this with a richer kind of GF grammar.

The second step is to convert the abstract syntax trees resulting from the parser into formulas in an **assembly logic**. The assembly logic is an itermediate representation between abstract syntax trees and normal "back end" logics, such as predicate logic. It is more abstract than the syntax trees coming from the parser, but preserves more distinctions than e.g. predicate logic. These distinctions are particularly useful when deriving visualizations of the structure, such as **spreadsheets**, two-dimensional representations that make the logical structure

[1] See [url to be provided after blind review].

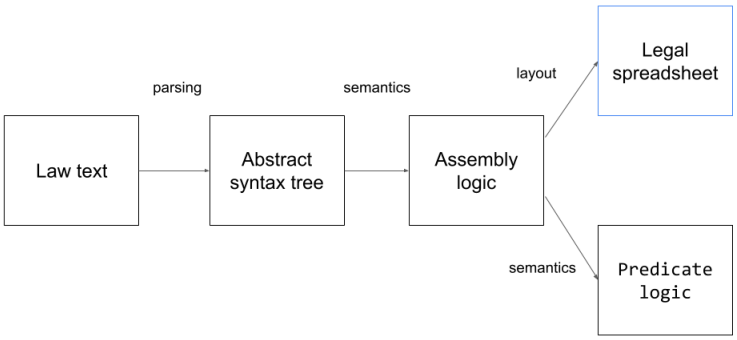



**Figure 1.** The pipeline

explicit without using logical formulas. These visualizations are intended to help understand the text and also to select proper interpretations of it in case of ambiguity. The conversion is performed in Haskell using the methodology described in [33].

**[I: moved this para as per Jerrold's suggestion, but unsure how it fits.]** Notice that the ultimate unit of semantic interpretation is a paragraph consisting of several lines. This is the case, for instance with the example in Figure 6. In the current implementation, the parser reads the document line by line, and a separate process is used for segmenting groups of lines into paragraphs. The reason for this set-up is mainly practical: semantic units can be arbitrarily long sequences of lines, and the parser may get slow in such cases.

### *2.2. Building the Grammars*

Parsing is driven by a grammar, which specifies a relation between strings and abstract syntax trees. In typical GF applications, abstract syntax trees are processed further, usually in translations to other languages but also in logical semantics. GF grammars are usually written by hand, to guarantee a precise correspondance to desired abstract syntax trees. This process is helped by GF's module system and extensive libraries, which reduce the need of manual work to the minimum. In such applications, the language can be made to follow grammar rules defined in the GF Resource Grammar Library (RGL, [31]).

However, law texts contain special constructs that the RGL does not cover, in particular constructs including entire paragraphs and itemized lists, which are significant for the logical structure and which we wanted to capture by the parser. This means that some grammar writing needs to be performed on top of the RGL. In order to make sure to capture everything, we started grammar writing in a data-driven, top-down way, starting from entire lines of text and going forward by stepwise refinements of the grammar rules.

An efficient way to produce the grammar turned out to be a semi-automatic method consisting of manual annotations from which a script generated GF rules. Figure 2 shows the grammar-building workflow with an example of a line of a text, the annotations added to it, and the resulting grammar rules.

**A line in the text:**

*Item (2) without limiting subsection Ref (1)(a), a CN data breach is deemed to VP result in significant harm to an individual —*

**The line annotated with marks for terminals (#) and nonterminals (*):**

```
*Item (2) #without #limiting #subsection *Ref (1)(a) #,
#a *CN data breach #is #deemed #to
*VP result in significant harm to an individual #|
```

**Grammar rules derived by the script:**

```
Line ::= Item "without" "limiting" "subsection" Ref ","
         "a" CN "is" "deemed" "to" VP "-" ;
Item ::= "(2)" ;
Ref  ::= "(1)(a)" ;
CN   ::= "data" "breach" ;
VP   ::= "result" "in" "significant"
         "harm" "to" "an" "individual" ;
```

**The last of the grammar rules manually edited by step-wise refinements:**

```
VP2 ::= "result" "in" NP ;
NP  ::= "significant" "harm" "to" NP ;
NP  ::= "an" CN ;
CN  ::= "individual" ;
```

**Abstract syntax functions in the resulting GF grammar:**

```
fun VP2_result_in : VP2 ;
fun CN_significant_harm_to_NP : NP -> CN ;
fun NP_an_CN : CN -> NP ;
fun CN_individual : CN ;
```

**Figure 2.** The grammar extraction process.

The grammar produced by the script is a BNF (context-free) grammar, in a notation that is directly usable in GF as it is. Internally, GF adds to each rule an abstract syntax function, whose names is derived from items in the rule itself. Full GF is more expressive and more abstract than BNF, and it can merge together rules that are just morphological variants of each other. In the example of Figure 2, the full power can be used to merge together the function `CN_individual` with another derived function `CN_individuals`, to a single function that covers both the singular and the plural form of the noun. Similarly, the rules `NP_an_CN` can be merged with `NP_a_CN`. An advantage, in addition to getting fewer rules, is that the choice of the sinular and the plural, or *a* vs. *an*, can then be made precisely

**Some assembly logic constructors:**

```
data Cat =
  CProp | CSet | CInd | -- ...
data Formula =
    Atomic Cat Atom
  | Conjunction Cat ConjWord [Formula]
  | Implication Formula Formula
  | Conditional Formula Formula     -- reverse implication
  | Quantification String Formula   -- quantifier + domain
```

**Semantics of abstract syntax trees in the assembly logic:**

```
iNP :: Env -> GNP -> Formula
iNP env np = case np of
  GNP_any_CN cn -> Quantification "ANY" (iCN env cn)
  GNP_each_CN cn -> Quantification "EACH" (iCN env cn)
  -- ...
  _ -> Atomic CInd (lin env (gf np))
```

**Figure 3.** Data structures and conversions related to the assembly logic.

depending on the context.

### *2.3. From abstract syntax trees to assembly logic*

The assembly logic is an intermeriate representation between abstract syntax trees and ordinary logics. It is designed to preserve enough of the syntactic structure to generate visualizations that are still recognizable as representations of the original text. For example, it distinguishes between ordinary an reverse implications (*if A then B* vs. *B if A*). It also preserves quantified noun phrases as units (e.g.*any organization* ) and is neutral between how modalities (such as *must*) are formalized in the logic. At the same time, it is intended to be sufficient to represent much more varied sets of abstract syntax trees, so that when the grammar is extended (e.g. via annotations of new law texts), the assembly logic and its back ends can be kept constant.

Figure 3 shows a sample of the assembly logic, which is implemented as a Haskell datatype `Formula`. It also shows a part of an **interpretation function**, `iNP`, which converts abstract syntax trees of GF type NP (Noun Phrase) to assembly logic. These functions use pattern matching over trees. Each constructor of an abstract syntax tree may have its own pattern, such as for `GNP_any_CN` in Fig. 3. When the grammar is extended, new patterns can be added. But even if this is not done, the function can take care of the new constructors by the catch-all case (`_`), which treats the new expressions as atomic. Atomic expressions are converted to atomic formulas or constants in logics and to single cells in spreadsheets.

**A toy example of a "donkey sentence":**

*if a notification is a data breach , the notification is affected*

**Compositional interpretation in many-sorted logic with iota terms:**

$$(\exists x : \mathit{notification})\mathit{data\_breach}(x) \supset \mathit{affected}(\iota(\mathit{notification}))$$

**Conversions to ordinary predicate logic in TPTP notation:**

```
![X]:(notification(X) => data_breach(X) => affected(X))
```

**Figure 4.** Iota terms are eliminated via anaphora resolution.

### *2.4. From assembly logic to predicate logic*

The pipeline in Figure 1 branches from Assembly Logic to two directions: to spreadsheets, described in the next section, and to predicate logic, to be discussed here.

The predicate logic step itself is divided to two phases:

1. Assembly logic is mapped into a many-sorted logic with Russell's iota terms.
2. The many-sorted logic is mapped into ordinary predicate logic and rendered in the TPTP notation.

The many-sorted logic is chosen because of its better support of compositional translation. Thus quanfication expressed by noun phrases (such as *any storage medium*) are compositionally interpreted as quantifiers with sorts, instead of dividing them into unsorted quantiers and sort predicates. The sorts are eliminated as a part of the conversion from many-sorted to ordinary logic.

Even more crucially, anaphoric expressions (such as *that organization*) are interpreted as definite descriptions formalized as iota terms (notice that we write $\iota(A)$ instead of $(\iota x)A(x)$, leaving possible variable bindings to $A$ itself, as customary in higher-order logic). These iota terms are eliminated in a pass that looks for non-iota terms in their context of use. Figure 4 shows an example of this, with the structure of a "donkey sentence" well-known in linguistic literature (*if a man owns a donkey he beats it*) [12,19] Such a sentence has an existential quantifier in the antecedent, and a pronoun or definite desctiption referring to it in the succedent. To express this reference in ordinary predicate logic, the existential quantifier is changed into a universal one with a wide scope of the implication.

Donkey sentences are ubiquitous in law text, as in many other types of text. To our surprise, also "inverted donkey sentences" occurred in our sample text — ones in which the existential appears in the succedent and is referred to in the antecedent. What makes this sound natural is that the whole implication appears in reverse: *a man beats a donkey if he owns it.* Figure 5

26B . — (1) a data breach is a notifiable data breach if the data breach —
(a) results in , or is likely to result in , significant harm to an affected individual ; or
(b) is , or is likely to be , of a significant scale .

```
(∃x : significant_harm_to_an_affected_individual)
  (results_in(ι(data_breach),x) ∨
   is_likely_to_result_in(ι(data_breach),x)) ∨
   (∃x : of_a_significant_scale)(is(ι(data_breach),x) ∨
   is_likely_to_be(ι(data_breach),x))
⊃ (∃x : data_breach)is_a_notifiable_data_breach(x)

![X]:(data_breach(X) =>
  ?[Y]:(significant_harm_to_an_affected_individual(Y) & (results_in(X,Y) |
    is_likely_to_result_in(X,Y))) |
    ?[Y]:(of_a_significant_scale(Y) & (is(X,Y) |
    is_likely_to_be(X,Y)))
      => is_a_notifiable_data_breach(X))
```

**Figure 5.** An inverted donkey sentence from the actual law text.

### *2.5. Visualisation in Spreadsheets*

Aside from predicate logic, the AST allowed us to visualise the statute's structure in spreadsheet form. Presently, the spreadsheet only serves as a form of visualisation, similarly to [24]. However, we plan to use the spreadsheet format as an input to a low-code programming platform, to help non-technical users learn, understand, and write (intermediate) legal formalisms more easily. This work is currently under development by our research team and will be more formally described in a separate paper.

## 3. Formalising the Personal Data Protection Act

We demonstrate our proposed pipeline on Part 6A the PDPA. Like the EU's *General Data Protection Regulation* (GDPR), the PDPA is Singapore's primary personal data protection statute and prescribes obligations surrounding the collection, use, disclosure, and deletion of personal data. Part 6A in particular stipulates when and how organisations are expected to notify regulators of personal data breaches (e.g. a hospital's patient databases are leaked). We chose the PDPA as a case study for three reasons. First, modelling the PDPA was a practical suggestion from the Personal Data Protection Commission (the Commission), Singapore's primary data regulator, based on their experience having to field numerous questions and prepare volumes of public-friendly guidelines on Part 6A. There was thus a clear use case for our technology. Second, while the PDPA has not been examined in prior AI & Law literature, its subject matter connects it to established work modelling the GDPR [26,**?**,16].

Third, the notification sections we examine are sufficiently complex to demonstrate the utility of a computational law approach in general and our pipeline in particular. Counter-intuitively, organisations are not always expected to immediately disclose data breaches upon discovering them. Rather, breaches are only

| | |
|---|---|
| lines | 66 |
| characters | 6154 |
| tokens | 1072 |
| unique tokens | 229 |
| tokens per line on average | 16 |
| tokens on the longest line | 56 |
| logical units | 46 |

**Table 1.** Statistics about Personal Data Protection Act (PDPA), Part 6A

notifiable if they (are likely to) result in significant harm affected individual(s), or if they are of "significant scale" (PDPA s 26B(1)). If so, there is a general duty to report the breach to the Commission "as soon as practicable" (s 26D(1)). The organisation must thereafter notify each affected individuals "in any manner that is reasonable in the circumstances" (s 26B(2)) — unless the Commission directs them not to (s 26D(2)). The Commission's published guidelines explain that this may be directed if the breach is of such severity that public responses to it may need to be carefully managed **[J: Meng please confirm and provide a cite]**. Modelling these rules allowed us to surface the implicit race condition between notifying the Commission and the affected individuals: an organisation which proactively informed both groups at the same time might inadvertently flout the statute's design.

Table 1 provides a statistical summary of the primary input text.

To formalise Part 6A, we first... **[J: can someone write a one para summary of the actual process here? That is, how exactly did we convert Part 6A into code? How did we build the specific grammars, etc?]** Next, we... Finally, ...

Figure 6 illustrates selected portions of the text through various stages of the pipeline.

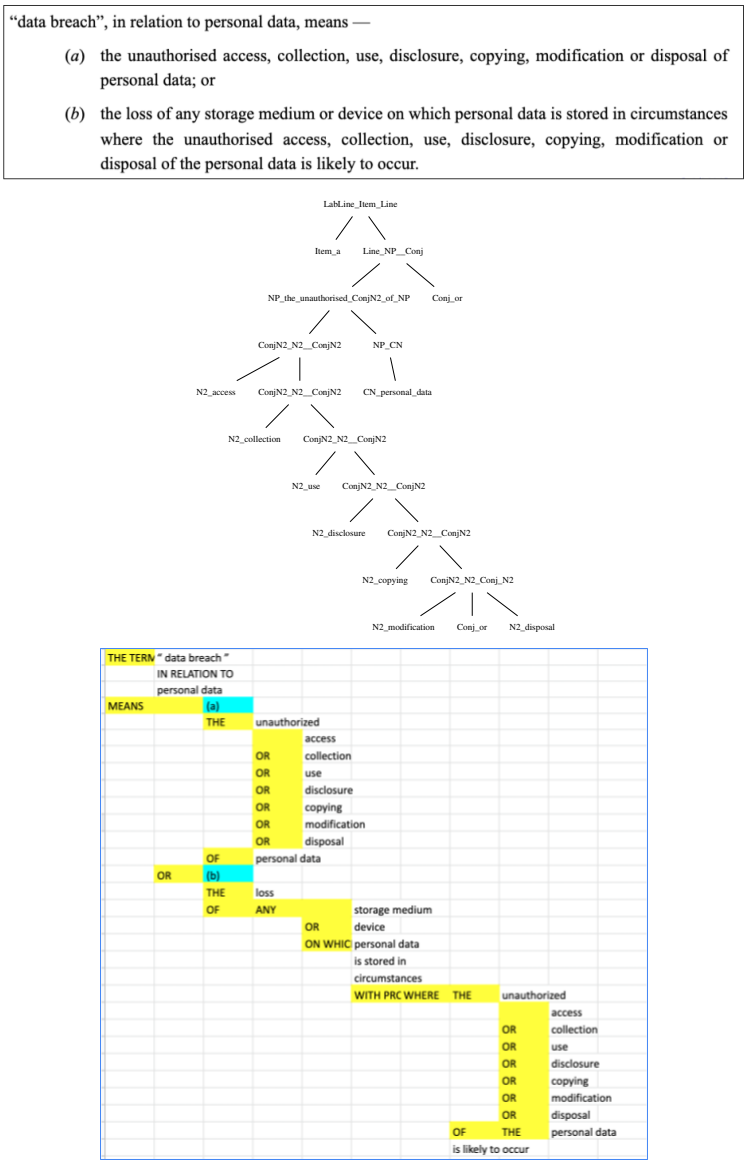
## 4. Discussion and future work

**[J: We will need to offer some insights on how well the thing worked. How did we know if we got a good logical output? Were there any standards we checked it against?]**

The most important tasks for future work are to link this pipeline with the previous work at CCLAW, which have much more precision and detail: the spreadsheets and the grammar of smaller units.

As regards spreadsheets, and in fact the interpretation in logic, the current experiment is simple-minded in the sense that it only looks at the document at hand. The real spreadsheets also take into account other documents that affect the interpretation, and show this information in the spreadsheets.

As for the grammar, the bulk of the work will be in the detailed structure of noun phrases and predicates that appear in single cells. It also remains to see how much of the line and paragraph structures is already included in the grammar based on the sample, but the variation there can be expected to be numerically smaller.




```
![X]:(data_breach(X) & ?[Y]:(personal_data(Y) & IN_RELATION_TO(X,Y)) <=>
(personal_data(X) & ?[Y]:((access(Y,X) | collection(Y,X) | use(Y,X) |
disclosure(Y,X) | copying(Y,X) | modification(Y,X) | disposal(Y,X)) &
unauthorized(Y))) | (((storage_medium(X) | device(X)) & (personal_data(X) &
?[Y]:((circumstances(Y) & (((unauthorized(Y) & (access(Y) | collection(Y) |
use(Y) | disclosure(Y) | copying(Y) | modification(Y) | disposal(Y))) &
is_likely_to_occur(Y))) & is_stored_in(X,Y)))) & loss(X))))
```

**Figure 6.** An example through the pipeline: text, abstract syntax tree (of the second line), spreadsheet, and formula in TPTP notation.

We do not claim to have completely solved the natural language barrier. But given the tedium which necessarily accompanies manual translations of law to logic, we hope to have shown a productive way to split the process into separate steps. Some of these steps can work out of the box when the scope is extended, whereas some — the text annotations for extending the grammar and the conversion to assembly logic — are lightweight enough to make the system feasible to apply.

## References


[1] K. Angelov and A. Ranta. Implementing Controlled Languages in GF. In *CNL-2009, Controlled Natural Language Workshop, Marettimo, Sicily, 2009*, 2009.

[2] Krasimir Angelov, John Camilleri, and Gerardo Schneider. A framework for conflict analysis of normative texts written in controlled natural language. *The Journal of Logic and Algebraic Programming*, 82:216–240, 2013.

[3] Tara Athan, Harold Boley, Guido Governatori, Monica Palmirani, Adrian Paschke, and Adam Wyner. OASIS LegalRuleML. In *Proceedings of ICAIL*, pages 3–12, Rome, Italy, 2013. ACM Press.

[4] T J M Bench-Capon. SUPPORT FOR POLICY MAKERS: PROSPECTS FOR KNOWLEDGE BASED SYSTEMS. In *Proceedings of JURIX*, pages 41–50, 1992.

[5] T. J. M. Bench-Capon, G. O. Robinson, T. W. Routen, and M. J. Sergot. Logic programming for large scale applications in law: A formalisation of supplementary benefit legislation. In *Proceedings of the first international conference on Artificial intelligence and law - ICAIL '87*, pages 190–198, Boston, Massachusetts, United States, 1987. ACM Press.

[6] J. Bing. Designing text retrieval systems for conceptual searching. In *Proceedings of ICAIL*, pages 43–51, Boston, Massachusetts, United States, 1987. ACM Press.

[7] Ilias Chalkidis, Ion Androutsopoulos, and Nikolaos Aletras. Neural Legal Judgment Prediction in English. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4317–4323, Florence, Italy, 2019. Association for Computational Linguistics.

[8] Ilias Chalkidis, Abhik Jana, Dirk Hartung, Michael Bommarito, Ion Androutsopoulos, Daniel Martin Katz, and Nikolaos Aletras. LexGLUE: A Benchmark Dataset for Legal Language Understanding in English. In *Proceedings of ACL*, pages 4310–4330, 2022.

[9] Michael Collins. Head-Driven Statistical Models for Natural Language Parsing. *Computational Linguistics*, 29(4):589–637, December 2003.

[10] Emile de Maat and Radboud Winkels. Automatic Classification of Sentences in Dutch Laws. In *Proceedings of JURIX*, pages 207–216, 2008.

[11] Norbert E. Fuchs, Kaarel Kaljurand, and Tobias Kuhn. Attempto Controlled English for Knowledge Representation. In Cristina Baroglio, Piero A. Bonatti, Jan Małuszyński, Massimo Marchiori, Axel Polleres, and Sebastian Schaffert, editors, *Reasoning Web, Fourth International Summer School 2008*, number 5224, pages 104–124. Springer, 2008.

[12] Peter Geach. *Reference and Generality*. Cornell University Press, Ithaca, New York, 1962.

[13] Digital Grammars and Signatu. GDPR Lexicon. `https://gdprlexicon.com/`, 2018.

[14] C Groendijk. NEURAL SCHEMATA IN AUTOMATED JUDICIAL PROBLEM SOLVING. In *Proceedings of JURIX*, pages 147–157, 1992.

[15] N Den Haan. TRACS: A SUPPORT TOOL FOR DRAFTING AND TESTING LAW. In *Proceedings of JURIX*, pages 63–70, 1992.

[16] David Hickey and Rob Brennan. A GDPR International Transfer Compliance Framework Based on an Extended Data Privacy Vocabulary (DPV). In *Proceedings of JURIX*, pages 161–170. IOS Press, December 2021.

[17] Joris Hulstijn. A Taxonomy for the Representation of Privacy and Data Control Signals. In *Proceedings of JURIX*, pages 23–32. IOS Press, December 2020.

[18] Liane Huttner and Denis Merigoux. Catala: Moving towards the future of legal expert systems. *Artificial Intelligence and Law*, August 2022.

[19] Hans Kamp. A theory of truth and semantic representation. In J. Groenendijk, T. Janssen, and M. Stokhof, editors, *Formal Methods in the Study of Language, Part 1*, pages 277–322. Mathematisch Centrum, Amsterdam, 1981.

[20] Robert Van Kralingen and Eduard Oskamp. NORM FRAMES IN THE REPRESENTATION OF LAWS. In *Proceedings of JURIX*, pages 11–21, 1993.

[21] L. T. McCarty. A language for legal Discourse I. basic features. In *Proceedings of ICAIL*, pages 180–189, Vancouver, British Columbia, Canada, 1989. ACM Press.

[22] L. Thorne McCarty. Reflections on "Taxman": An Experiment in Artificial Intelligence and Legal Reasoning. *Harvard Law Review*, 90(5):837, March 1977.

[23] L. Thorne McCarty. Deep semantic interpretations of legal texts. In *Proceedings of ICAIL*, pages 217–224, 2007. Journal Abbreviation: Proceedings of the International Conference on Artificial Intelligence and Law Pages: 224 Publication Title: Proceedings of the International Conference on Artificial Intelligence and Law.

[24] Raquel Mochales and Marie-Francine Moens. Study on the Structure of Argumentation in Case Law. In *Proceedings of JURIX*, pages 11–20, 2008.

[25] Adeline Nazarenko, François Lévy, and Adam Wyner. A Pragmatic Approach to Semantic Annotation for Search of Legal Texts – An Experiment on GDPR. In *Proceedings of JURIX*, pages 23–32. IOS Press, December 2021.

[26] Monica Palmirani and Guido Governatori. Modelling Legal Knowledge for GDPR Compliance Checking. In *Proceedings of JURIX*, pages 101–110, 2018.

[27] Monica Palmirani, Michele Martoni, Arianna Rossi, and Livio Robaldo. Legal Ontology for Modelling GDPR Concepts and Norms. In *Proceedings of JURIX*, pages 91–100, 2018.

[28] Monica Palmirani and Fabio Vitali. Akoma-Ntoso for Legal Documents. In Giovanni Sartor, Monica Palmirani, Enrico Francesconi, and Maria Angela Biasiotti, editors, *Legislative XML for the Semantic Web*, pages 75–100. Springer Netherlands, Dordrecht, 2011.

[29] Henry Prakken. A logical framework for modelling legal argument. In *Proceedings of ICAIL*, pages 1–9, Amsterdam, The Netherlands, 1993. ACM Press.

[30] Paulo Quaresma and Irene Pimenta Rodrigues. A Question Answer System for Legal Information Retrieval. In *Proceedings of JURIX*, pages 91–100, 2005.

[31] A. Ranta. The GF Resource Grammar Library. *Linguistics in Language Technology*, 2, 2009.

[32] A. Ranta. *Grammatical Framework: Programming with Multilingual Grammars*. CSLI Publications, Stanford, 2011.

[33] Aarne Ranta. Translating between language and logic: What is easy and what is difficult. In Nikolaj Bjørner and Viorica Sofronie-Stokkermans, editors, *Automated Deduction – CADE-23*, pages 5–25, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[34] M. J. Sergot, F. Sadri, R. A. Kowalski, F. Kriwaczek, P. Hammond, and H. T. Cory. The British Nationality Act as a logic program. *Communications of the ACM*, 29(5):370–386, May 1986.

[35] J S Svensson, P J M Kordelaar, and J G J Wassink. EXPERTISZE, A TOOL FOR DETERMINING THE EFFECTS OF SOCIAL SECURITY LEGISLATION. In *Proceedings of JURIX*, pages 51–61, 1992.

[36] Radboud Winkels and Rinke Hoekstra. Automatic Extraction of Legal Concepts and Definitions. In *Proceedings of JURIX*, pages 157–166, 2012.

[37] Alice Witt, Anna Huggins, Guido Governatori, and Joshua Buckley. Converting copyright legislation into machine-executable code: interpretation, coding validation and legal alignment. In *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Law*, pages 139–148, São Paulo Brazil, June 2021. ACM.

[38] Adam Wyner and Guido Governatori. A Study on Translating Regulatory Rules from Natural Language to Defeasible Logic. In *Proceedings of the 7th International Web Rule Symposium*, pages 16.1–16.8, 2013.