Representing legislative Rules as Code: Reducing the problems of 'scaling up'

Andrew Mowbray, Philip Chung and Graham Greenleaf*

9 December 2021 – 7,277 words

* Andrew Mowbray is Professor of Law & Information Technology at UTS and Co-Director of AustLII; Philip Chung is Associate Professor of Law at UNSW and Executive Director of AustLII; Graham Greenleaf is Professor of Law & Information Systems at UNSW and Co-Founder of AustLII.

This paper outlines an approach to modelling legal rules, particularly legislative rules, that focusses on representing legislation as a hierarchical set of propositions that records both mechanical and real world meaning. It suggests a methodology for expressing these rules in a way that is machine consumable. The paper describes the progress that has been made by AustLII's[1] DataLex project in leveraging this form of representation in several ways including automated conversion of existing legislative rules, which can also be described as 'scaling up' the production of 'Rules as Code'. Based on this experience, the paper also demonstrates how the drafting of legislation could be changed to make it directly readable and understandable by humans and also usable by machines.

We conclude by suggesting that there is evidence these processes are potentially scaleable and able to deal with the conversion or production of large bodies of legislation. Further work is needed to investigate the extent to which the conversion and creation of 'Rules as Code' can be done at scale and applied to different types of legislation and other classes of legal rules.

1. Introduction: Rules as Code (RaC)

Rules as Code (RaC) is a field of research into making human-made rules usable by machines, to perform useful results. The rules to which RaC can be applied include statutes, regulations and many other types of law-related rules, as well as organisational rules such as codes of practice, codes of conduct and business procedures.

1.1. What do we mean by 'Rules as Code'?

'Rules as Code', in our definition,[2] is the activity of creating or converting a legal text which is in a natural language (legislation, regulations, or other legal instruments – generically, 'law' or 'rules'), in or into a representation in a computer-processable form (code). One application of this allows a human user to input data on a particular fact situation, and thereby produce conclusions which are an accurate statement of the legislative intent of the legal text when applied to that data. Other applications include the embedding of rules in automated systems and to facilitate the communication between systems to determine such matters as design and compliance.

The 'conversion' into code may be simultaneous with the creation of the natural language text (drafting law as code) or retrospective (converting existing laws from a back-set into code).

Rules as Code (RaC) is also often referred to as the creation of machine-consumable ('executable') versions of laws and other type of rules, or 'creating machine-interpretable regulation'.[3]

---

[1] AustLII is the Australasian Legal Information Institute <http://www.austlii.edu.au>, a joint facility of UTS and UNSW Faculties of Law. The DataLex project is at <http://www.datalex.org>.

[2] Terminology in this field is still very unsettled, with no generally accepted definitions for 'Rules as Code', 'Law as Code' or 'Code as Law'.

[3] Productivity Commission (Australia) *Information Paper on Regulatory Technology* (October 2020), p. 13 <https://www.pc.gov.au/research/completed/regulatory-technology/regulatory-technology.pdf>

Examples of the types of legal rules to which RaC can be applied to include legislation, regulations, other forms of delegated legislation (ordinances, by-laws and so forth), court rules, codes of conduct, treaties or technical standards.[4]

There are broadly two possible approaches to achieving RaC in the context of legislative drafting. When rules are made (or subsequently) two versions can be created: one in natural language (for humans) and one in executable form (for machines). Alternatively, the rules can be written from the start in such a way that they are both intelligible and usable by humans as well as by machines. In this article we discuss both approaches.

1.2. Why 'Rules as Code' matters

'Rules as Code' is important to all types of 'stakeholders' in legislation-related activities:

1. RaC can make the law more accessible and more understandable, if implemented with such goals in mind, by allowing anyone to see not only what laws say, but also how they operate in particular situations. The effects of rules can be tested against hypothetical fact situations. RaC can therefore become a natural extension of free access to legal information.[5] For it to do so, implementations of RaC must be made freely available to all. RaC is therefore of value to individual citizens, and to governments making rules.

2. From an economic and business perspective, RaC can lower the very high costs of compliance with legislation or other rules,[6] and is therefore of potential value to the economy as a whole.

No matter which approach to adopting a RaC approach to the creation of new legislative rules, there needs to be some way of dealing with the extensive current sets of existing rules. AustLII currently contains 28,290 Australian statutes and 40,109 regulations, comprised of 1,786,600 legislative sections.

This legislation affects the lives of all citizens and regulates the way that business and the economy function, and it is heavily used. In 2020 there were 96,496,797 accesses to Australian legislation on AustLII alone. To this must be added some comparable amount of use on all the government legislation services combined, plus the commercial providers. Using a rough estimate of 200 million accesses to sections, each of the nearly 2 million sections is being accessed on average 100 times per annum. Of course, actual usage would have high variability.

---

[4] The 'rules' can also include case law which functions as precedents, but this involves different considerations beyond the scope of this article.

[5] RaC is therefore of particular interest to legal information institutes such as AustLII.

[6] "According to industry estimates, financial institutions globally spend more than US$70 billion on compliance annually and the costs for regulatory compliance and governance software across the global industry are expected to approach US$120 billion by 2020, more than half of which will occur in consulting and business services. It is estimated that from the 2008 financial crisis through 2015, the annual volume of regulatory publications, changes, and announcements increased a staggering 492 per cent." Australian Prime Minister Scott Morrison, Address to the Fintech Australia Summit, Melbourne (November 2016) <https://ministers.treasury.gov.au/ministers/scott-morrison-2015/speeches/address-fintech-australia-summit-melbourne>

2. Structure and nature of legislative rules

2.1. Coding what rules say, or what they do?

At one level, legislative rules can be seen as being prescriptive, regulating social, business and economic activities. The temptation is to simply model these prescriptions, but this leads to an immediate conceptual fork between what legislative rules "say" and what they are taken to "do". Once there is an attempt to (for example) regard legal rules as expressing obligations or duties (deontic expressions) or to impose any other consequential interpretation, then any form of representation is no longer describing the rules themselves (and so does not represent *them*), but is instead a representation of an *interpretation* of what the rules do.

There is a long history of this type of division in the application of artificial intelligence to law. For example, in 1990, Thorne McCarty, then a leader in the new field of 'AI and law' argued that his "deep conceptual model" of the legal domain was superior to other approaches while not denying those approaches have some uses. But, he said, a more complex representation is needed: "There are many common sense categories underlying the representation of a legal problem domain: space, time, mass, action, permission, obligation, causation, purpose, intention, knowledge, belief, and so on. The idea is to select a small set of these common sense categories, the ones that are most appropriate for a particular legal application, and then develop a knowledge representation language that faithfully mirrors the structure of the selected categories."[7]

In the thirty years since then, many other approaches have been developed to the representation of legal rules which focused on what rules do, or their meaning, including those associated with deontic logic, the semantic web, and legal ontologies.

2.2. Paying attention to what rules say

AustLII's approach is to focus on what rules literally say, and to represent that in code. This includes not just references to things that are external to the rules, but also includes internal references to the applicability or otherwise of other rules and sub-parts of rules.

Legislative rules are usually structured into clauses or sections, each of which is then divided into sub-sections and perhaps even sub-sub-sections. This is done so that lawyers and end-users of legal rules can communicate about which specific aspect of a rule they are discussing. The issues that each rule component relates to are various. Many of these issues are expressed propositionally or the issues can be reduced to a proposition. Some of the examples of the types of matters that legislative rules deal with include:

- setting out principles, objectives or policies
- providing factors to be taken into account in determining an issue
- making declarations

[7] Thorne McCarty 'Artificial Intelligence and Law: How to Get There From Here' (1990) <https://www.researchgate.net/publication/229678099_Artificial_Intelligence_and_Law_How_to_Get_There_from_Here>

- imposing obligations or granting rights
- describing how other sections apply or how they are changed

This list is not exhaustive but is meant to make the point that the subject matter and effect of legislative provisions is diverse and cannot easily be captured by any single ontology or logical system. Rather, legislation is better characterised as being a structured set of statements about any subject matter (including statements about how the rules themselves should operate and inter-relate) and is generally expressed using formal structures containing linked propositions.

The types of propositions in legislation include:

- Substantive propositions
    - *Eg the applicant earns less than $20,000 pa*
- Definitions
    - Eg "*earns" means any form of income*
- Explicit 'structural' propositions
    - *Eg Part II applies*
- Implicit 'structural' propositions
    - *Eg section 1(a) applies and section 1(b) applies*
- External 'structural' propositions
    - *Eg Company has the same meaning as in section 6 of the Corporations Act 2001*

A majority of provisions are usually non-substantive. Instead, most provisions describe how the various parts, divisions, sections and sub-sections relate to each other. In our view, the main goal of any form of representation is to reflect the structure and interrelations of legislative rules. To a large extent, the significance of any real-world associations are secondary.

2.3. Ambiguities in what rules say

The leading textbook in the field of AI and law[8] notes that 'statutory reasoning *should* be easy', but that fifty years of work in the field has 'developed an appreciation of just how difficult the problem is'.[9] Ashley identifies two main sources of the problem:[10]

(i) 'semantic ambiguity, and its cousin, vagueness': 'The regulatory concepts and terms the legislature selects may not be sufficiently well defined to determine if

[8] Kevin D. Ashley *Artificial Intelligence and Legal Analytics* (Cambridge University Press, 2017) (hereafter 'Ashley, 2017')

[9] Ashley, 2017, pp. 38-39.

[10] Ashley, 2017, p.39.

or how they apply.' The relationships between the concepts may also not be clear.

(ii) 'syntactic ambiguity': 'the logical terms legislatures use, such as "if," "and," "or," and "unless," introduce multiple interpretations of even simple statutes.'

The main focus of the DataLex approach is the syntax of legislation (or other rules) and how that syntax may be used to represent machine-interpretable legislation. Although most syntactic elements of a piece of legislation are not ambiguous (for example, in the implicit structural propositions example above, the 'and' can safely be replaced by a logical 'and') there remain significant issues to be addressed where there is ambiguity.

Semantic ambiguity, or the 'open texture of legal language', on the other hand is not something that representation by itself can easily reduce. The need for human interpretation of statutory terms is not something that can be removed from machine-interpretable legislation, although well-designed computer applications can assist human users with the task of interpretation, through actively assisting them to locate the materials (cases, scholarship etc) on which such interpretation must be based.[11]

2.4. Example of what rules say: Patents Act 1914, s. 114

Consider the following from the Australian Patents Act 1990:

PATENTS ACT 1990 - SECT 114

Priority date of claims of certain amended specifications

(1) This section applies if:

(a) a complete specification has been amended; and

(b) the amendment was not allowable under subsection 102(1); and

(c) as a result of the amendment, a claim of the amended specification claims an invention that:

(i) was not disclosed by the complete specification as filed in a manner that was clear enough and complete enough for the invention to be performed by a person skilled in the relevant art; but

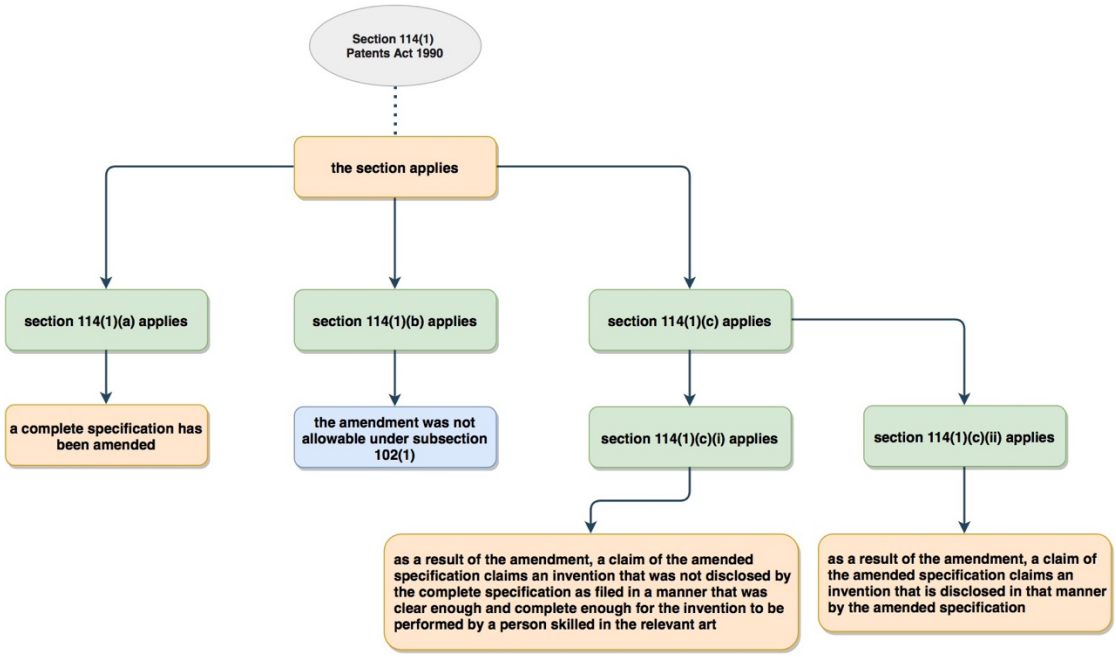
(ii) is disclosed in that manner by the amended specification.

(2) If this section applies, the priority date of the claim must be determined under the regulations.

The section says something about each of ten different concepts (that is, statements or propositions). Sub-section (1) says the section 114 will apply only where three subsequent conditions are met. The first of these (contained in sub-section (1)(a)) is a substantive proposition, namely that "a complete specification has been amended". Subsection (1)(b) relies upon an explicit structural proposition that "the amendment was not allowable under subsection 102(1)". Some structural elements of the section contain more than one proposition (such as (1)(c)(i)).

[11] Greenleaf G, Mowbray A and Chung C, Building Sustainable Free Legal Advisory Systems: Experiences from the History of AI & Law (2018) 34(1) *Computer Law & Security Review* 314

Conceptually, the section can be represented as a directed graph as follows:



At a broader level, the entire Act can be represented in a similar way. Acts, as currently drafted, are seldom complete and will require the insertion of dependencies and relationships between Act components.

3. Representation of legislative rules: *yscript*

yscript (pronounced 'why-script') is a language for representing and manipulating propositions.[12] It can be used to represent real-world rules such as legislation or codes of practice, as well as to create systems based around less formally defined procedures or knowledge.

Syntactically correct yscript rules are able to be run by the *yscript interpreter*[13] to engage a user in a consultation, and to produce a report on conclusions generated during the consultation.[14] The yscript interpreter and library is available under an Affero GPL licence. [15]

---

[12] Mowbray *Coding in yscript* (AustLII, May 2021) <https://austlii.community/foswiki/pub/DataLex/WebHome/ys-manual.pdf>

[13] yscript is an extension of the language used by a previous system called ysh.

[14] Mowbray A, Chung P and Greenleaf G, Utilising AI in the Legal Assistance Sector – Testing a Role for Legal Information Institutes (2020) 38(1) *Computer Law and Security Review* 105407

[15] An explanation of the Affero AGPL licence is at: https://www.gnu.org/licenses/why-affero-gpl.html .

3.1. Features of the yscript language

Some of the features of *yscript*[16] are that it uses a quasi-natural-language 'English-like' syntax, which is easy to learn and use, supports declarative and imperative coding, produces natural English dialogs (consultations) and explanations without any textual 'baggage' in addition to its rules. Whilst yscript is a flexible general purpose language, it is can be used for representing legislation and other rules which are comprised of a structured set of propositions. It allows isomorphic (one to one) representation of legislation and other types of rules (important for transparency, explanation and maintainability). *yscript* can be directly created and maintained by lawyers.

When yscript code is executed, it results in a dialog or *consultation*. A series of questions are asked, and conclusions are made. Along the way, the user can interrogate the system as to why questions are being asked, to explain how conclusions have been reached and to check hypothetical question answers (that is, *what happens if I answer this way?*). When a session completes, a report is generated to give an answer to the original goals and to explain why this is the case.

Numerous applications written in *yscript* are on the DataLex web pages.[17] The yscript code for each application can be found there, and the applications can be run ('Consultations'). Users can develop and run their own test applications using the *DataLex Application Developer Tools*.[18]

From the outset, one of the central aims in the development of *yscript* was to develop a form of representation that looked as much like natural language as possible. The language syntax manages to almost entirely avoid the use of symbols which are the principal structural elements of most programming languages. This was done partly to make it easier to write code for non-programmers, but also to make the code more transparent. Even if someone cannot write code in yscript, they can probably understand what it is doing and possibly even comment upon whether it accurately encapsulates anything from the real world (such as the text of legislation) that it is meant to reflect.

In formal terms, yscript code consists of *rules* that deal with *facts*. Facts are expressed in their plain English-language form. Individual rules can be imperative but often are just declarative and describe the relationships between facts. Once a rule is being evaluated, other rules that can help determine a value for required facts are automatically executed. Rules are used in a goal-oriented fashion to determine values. Each time that a new fact becomes known, rules are used to check if other fact values can be derived. When required, rules can be specifically called like procedures or functions in other languages.

---

[16] An earlier version of the *yscript* language was originally developed for the expert systems shell *ysh*. Prior to being integrated into AustLII's DataLex platform, yscript was also used as the language and code interpreter for a system called wysh (short for "web-ysh").

[17] DataLex web pages <http://datalex.org/>

[18] *DataLex Application Developer Tools* <http://datalex.org/dev/tools/>

yscript also supports *examples*[19]. An *example* is a set of propositions (facts) supporting a particular outcome. This was implemented in yscript to deal with case-based reasoning in law but probably has application in other fields. Finally, yscript applications can generate *documents*. A *document* is built procedurally but can take advantage of rule-based determination of necessary facts.

One of the advantages of using a quasi-natural language form of coding is that it is often possible to directly adopt the wording of a set of real-world rules that you are trying to represent. This is particularly useful in the legal domain, where it is desirable to use language taken directly from statutes and regulations. In legal documents (legislation, contracts etc), the exact form of words is very often crucial. Specific words and phrases can imply underlying complex meaning or act as a point of reference to a body of interpretative documents and decisions which need to be considered.

The rule-based structure of yscript encourages and supports *isomorphism* (that is, one-to-one mapping) of real-word rules (particularly legislation) into yscript code. This makes it a lot easier to build applications, and of equal importance, it allows for simpler maintenance of the code when source legislation or other rules change, and for legal experts to audit the accuracy of the code without being computing experts.

One of the key ideas is that code should represent what rules **say** rather than what they **do**. yscript can be used to do either, but it is more effective in a RaC context when it used to represent what rules say.

3.2. Example of *yscript* code: Patents Act 1914, s. 114

The following is an example of a codebase that represents section 114 of the *Patents Act 1990* (Cth) (as set out above) using the yscript language:

```
RULE Section 114 - Priority date of claims of certain amended specifications
PROVIDES
SUBRULE Section 114(1)
SUBRULE Section 114(2)

RULE Section 114(1) PROVIDES
this section applies ONLY IF
    section 114(1)(a) applies AND
    section 114(1)(b) applies AND
    section 114(1)(c) applies

RULE Section 114(1)(a) PROVIDES
section 114(1)(a) applies ONLY IF
    a complete specification has been amended

RULE Section 114(1)(b) PROVIDES
section 114(1)(b) applies ONLY IF
    the amendment was not allowable under subsection 102(1)

RULE Section 114(1)(c) PROVIDES
section 114(1)(c) applies ONLY IF
    section 114(1)(c)(i) applies AND
```

[19] The yscript library uses an analogous reasoning approach developed by Alan Tyree called PANNDA. This is described in his book *Expert Systems in Law*, Prentice Hall, 1990.

```
    section 114(1)(c)(ii) applies

RULE Section 114(1)(c)(i) PROVIDES
section 114(1)(c)(i) applies ONLY IF
     as a result of the amendment, a claim of the amended specification claims
an invention that was not disclosed by the complete specification as filed in a
manner that was clear enough and complete enough for the invention to be
performed by a person skilled in the relevant art

RULE Section 114(1)(c)(ii) PROVIDES
section 114(1)(c)(ii) applies ONLY IF
     as a result of the amendment, a claim of the amended specification claims
an invention that is disclosed in that manner by the amended specification

RULE Section 114(2) PROVIDES
if this section applies, the priority date of the claim must be determined
under the regulations
```

4. From legislation to code: (i) Automating conversion of existing legislation to code

Since the earliest attempts to use AI to convert legislation into code, one of the main problems impeding development of 'real world' applications (in contrast to 'toy' applications) has been what is described in this paper as the problem of 'scaling up'. Australia's Productivity Commission[20] gives a succinct description:

> There are a number of factors that may limit the benefits of machine-interpretable regulation, at least in the short-term, including that existing options for creating such code may not readily scale up due to resource constraints, may require significant levels of validation to check for accuracy and may also necessitate standardisation that has not yet been developed. Nevertheless, it could be practicable for machine-interpretable text to be developed for selected components of regulation — where coded rules would help create service efficiencies, automate existing manual processes, or attract multiple uses in coded form.

ylegis is a program that converts the text of legislation into *yscript* rules that reflect the legislation's structure.

Until recently, development of the code for 'Rules as Code' applications, including those written using *yscript*, has been a completely manual process, with no automated assistance available, other than some error-checking tools.

To make the coding process easier, the DataLex project has developed a pre-processor program (*ylegis*), which takes a section of legislation (or multiple sections, or even a whole Act), and converts it automatically into a 'first draft' of *yscript* code for those legislative provisions. The pre-processed yscript code can immediately be run by the yscript interpreter, to test (via observing consultations, and by use of error-checking tools) how well the conversion to code has worked.

---

[20] Productivity Commission (Australia) *Information Paper on Regulatory Technology* (October 2020), p. 13 <https://www.pc.gov.au/research/completed/regulatory-technology/regulatory-technology.pdf>

ylegis can take legislation drafted in 'UK/Westminster style' format (as used in many Commonwealth jurisdictions such as Australia), and produce from it yscript rules that reflect the legislation's structure. It can be used to produce *yscript* code from unmodified original source legislation, with no human intervention. Technically, it can be described as a filter, or as a pre-processor.

4.1. Three possible outputs from ylegis

Three outcomes of the use of ylegis to convert existing legislation are possible:

1. In many cases, depending on the structure of the selected legislative provisions, the pre-processed code will need no change, because it is considered to accurately reflect the meaning of the legislation. In these cases, the goal of automated conversion of 'legislation into code' has been achieved. Examples are given below of sections 15KP and 15KQ of the *Crimes Act 1914* (Cth).
2. In many other cases, the pre-processed code will provide a useful 'first draft' of how the legislative provisions can be converted into the code required by the yscript interpreter. Further human editing of the pre-processed code will be necessary before it will run in a way which accurately represents the legislation.
3. In a minority of cases, the pre-processed code will not be a useful first draft, and it will be more efficient to start from scratch with manual encoding of the legislation into yscript code.

The effectiveness of *ylegis* in 'scaling up' the conversion of existing legislation into code is, to a large extent, the question of what percentages of all legislation tested fall into categories 1, 2 and 3 respectively. We want a high enough percentage to fall into category 1 for it to be very clear that this is a very efficient way in which to covert legislation into code. We do not expect that the percentage in category 3 will be zero, but it should be small enough that the use of ylegis is not considered to be 'hit and miss', with the difficulties of 'starting from scratch' being very time-consuming. While a significant percentage of 'useful first drafts' (category 2) might be an acceptable result, it will not be efficient (and might be dangerously misleading) if it is difficult and time-consuming to identify and fix shortcomings in such drafts.

There are several other factors which must also be taken into account in assessing the effectiveness of ylegis:

- The scope over which such percentage-based assessments should be made is arguable. Should it be the percentage of sections which fall into each category, irrespective of which Act they are in? Alternatively, if one section is in category 3, from an Act with 100 sections, does that mean the Act should be in category 3? Our view is that the most objective measure is the first, by individual sections.
- The question of whether the *yscript* code for a converted section falls into category 1 or 2 will often be arguable. For example, as discussed below in relation to section

> 15KP of the *Crimes Act 1914* (Cth), the ylegis conversion can be said to be legally correct, but the *yscript* code will be more precise in what it reports, if one change is made to the code.

The ylegis pre-processor is a program that is undergoing rapid development, with the range of less standard legislative structures than are described below coming within the scope of its conversion abilities. This makes it premature at the moment to make any definitive assessments of its effectiveness, but progress is sufficiently positive to be worth reporting.

4.2. Input requirements for ylegis

Input text to ylegis requires no special markup other than that sections and sub-sections should start at the beginning of a line. Any HTML or XML tags are ignored. The emphasis is on always producing some output *yscript* code. That code can be edited if not correct.

The structure of input text is assumed to be in the traditional UK/Westminster legislative section/sub-section hierarchy 1(1)(a)(i)(A). The second level (eg (4)) is optional and can be omitted.

Explicit definitions are also recognised where a line starts with a "quoted term" (for example, ' "Prescribed person" means …' or ' "Material form" includes …'). Each definition can have an associated hierarchy that is similar to a section.

A section may be preceded by a description. When not in formal mode (see 5.1), text following a section tag for lines with no operator is also treated as a description.

Text to be processed with ylegis can be combined with traditional yscript by running ylegis as a pre-processor to the yscript interpreter. Sections should start with the keyword "SECTION" and finish with the keyword "END-SECTION". Otherwise input (yscript) passes through the pre-processor unchanged. Text to be processed as sections can be in plain format (ie just as it is set out in an Act or Regulation).

4.3. Syntax that ylegis does not automatically convert

The ylegis pre-processor is still being developed, and the range of legislative structures that it can usefully convert continues to expand.

The current version of ylegis does not attempt to use natural language processing to analysis syntactic elements of legislation which indicate logical components which are below the structural level. As Ashley notes, the 'the logical terms legislatures use, such as "if," "and," "or," and "unless," introduce multiple interpretations of even simple statutes.'[21] The interpretation of these terms is at present left to the person creating the rulebase, who will need to decide whether to edit the draft yscript rules produced by ylegis.

4.4. Examples of *ylegis* conversion: Crimes Act 1914, ss. 15KP and 15KQ

An example of the type of provision which ylegis can now successfully convert is section 15KP of the *Crimes Act 1914* (Cth):

---

[21] Ashley, 2017, p. 39; see Allen and Engholm, 1978

```
CRIMES ACT 1914 – SECT 15KP
Assumed identity may be acquired and used
A person may acquire or use an assumed identity if:
(a) the person is an authorised person (other than an authorised
    civilian) and the acquisition or use is:
    (i) in accordance with an authority; and
    (ii) in the course of duty; or
(b) the person is an authorised civilian and the acquisition or use is
    in accordance with:
    (i) an authority; and
    (ii) any direction by the person's supervisor under the authority.
```

The ylegis pre-processor automatically converts this section into the following yscript code:

```
RULE Section 15KP – Assumed identity may be acquired and used PROVIDES
the person may acquire or use an assumed identity ONLY IF
    section 15KP(a) applies OR
    section 15KP(b) applies

RULE Section 15KP(a) PROVIDES
section 15KP(a) applies ONLY IF
    section 15KP(a)(i) applies AND
    section 15KP(a)(ii) applies

RULE Section 15KP(a)(i) PROVIDES
section 15KP(a)(i) applies ONLY IF
    the person is an authorised person (other than an authorised
    civilian) and the acquisition or use is in accordance with an authority

RULE Section 15KP(a)(ii) PROVIDES
section 15KP(a)(ii) applies ONLY IF
    the person is an authorised person (other than an authorised
    civilian) and the acquisition or use is in the course of duty

RULE Section 15KP(b) PROVIDES
section 15KP(b) applies ONLY IF
    section 15KP(b)(i) applies AND
    section 15KP(b)(ii) applies

RULE Section 15KP(b)(i) PROVIDES
section 15KP(b)(i) applies ONLY IF
    the person is an authorised civilian and the acquisition or
    use is in accordance with an authority
```

```
RULE Section 15KP(b)(ii) PROVIDES
section 15KP(b)(ii) applies ONLY IF
    the person is an authorised civilian and the acquisition or
    use is in accordance with any direction by the person's
    supervisor under the authority
```

This section has been converted by ylegis into 'accurate' yscript code, in the sense that the code will run and produce a dialogue. This can be verified by copying the above rules, going to the *DataLex Application Developer Tools,*[22] pasting the rules in the 'Application' box, and running the Consultation. Further checks can also be done by running the three other Checks that the Tools provide.

4.5. How can *ylegis* conversions from existing legislation be tested?

The criteria against which it would be reasonable to claim that ylegis does make it possible to 'scale up' the creation of rules as code are discussed earlier. It is clear that 100% successful conversion of existing legislation will not be achieved:

- *Non-systematic examples of effectiveness* – At present, we are providing non-systematic examples of how well, or badly, the software does convert legislation into code.
- *Systematic tests of a sample of legislation* – We could attempt to convert a randomly-selected, significant sample of sections of Australian legislation, and have the accuracy of the results of the ylegis conversions checked by independent parties with sufficient expertise. This has not yet been done.
- *The eyeballs test*[23] – One unusual aspect of the DataLex approach is that it provides for free access to the software and data by which anyone can test the use of *ylegis* and ys*cript* (via the DataLex Development Tools) to convert any existing Australian legislation (as provided via AustLII). All existing codebases are freely available and the yscript interpreter is available under an open source licence. If anyone wishes to demonstrate flaws in the ylegis/yscript approach, they can do so. AustLII is therefore positioning its claims similar to open source software. Availability for testing does not guarantee that limitations will be found, but at least it makes this possible.
- *Comparison with alternatives* – Do any other approaches demonstrate that they produce superior results in accurately representing legislation, and/or that they do scale up to any significant extent? We are not aware of examples.

For this first method of using ylegis to convert legislation to code, we can therefore only make modest claims based on successful examples.

---

[22] *DataLex Application Developer Tools* <http://datalex.org/dev/tools/>

[23] Attributed to Linus Torvald (Linux) is the assertion that "given enough eyeballs, all bugs are shallow". Eric Raymond *The Cathedral and the Bazaar* (1999, O'Reilly)

5. From legislation to code: (ii) Writing Legislative Rules as Code

The second and perhaps a more significant use for *ylegis* is for writing legislation in a natural language format that can be also executed as code.

5.1. Formal mode of ylegis

ylegis can be used to write legislation in a form which is close to that used in conventional legislative drafting, but which can be converted into executable *yscript* code. This requires that the relationship between separate propositions be defined by a formal set of connectors (operators) such as "and:" and "or:". In this mode, text can use a set of formal operators to connect sub-sections and sub-clauses each of which contains a single proposition (be it a requirement or a conclusion). Each sub-clause of a section is assumed to be in the format:

```
tag [fact] [operator]
```

For example:

```
15KP the person may acquire an identity if:
```

In formal mode, the following five operators are recognised by ylegis:

and:	"fact" is AND'd with following fact
and/or:	"fact" is AND/OR'd with following fact
if:	sub-clauses are a condition for "fact"
or:	"fact" is OR'd with following fact
provides:	sub-clauses should just be considered in sequence

The formal mode of ylegis has some similarities to the systematic 'normalization' process for identifying syntactic ambiguities in a statute, developed by Layman Allen.[24]

5.2. Example of writing legislation as code: Crimes Act 1914, s. 15KP

The following is an example from the Crimes Act 1914 (Cth) as it appears in the original Act:

```
CRIMES ACT 1914 - SECT 15KP
Assumed identity may be acquired and used
A person may acquire or use an assumed identity if:
(a) the person is an authorised person (other than an authorised civilian)
and the acquisition or use is:
(i) in accordance with an authority; and
(ii) in the course of duty; or
```

[24] Described in Ashley, 2017, p. 41.

```
(b) the person is an authorised civilian and the acquisition or use is
in accordance with:
(i) an authority; and
(ii) any direction by the person's supervisor under the authority.
```

The same section might be expressed using formal mode and included in a larger piece of yscript code using pre-processor mode (-p) as:

```
    Assumed identity may be acquired and used
    15KP the person may acquire or use an assumed identity under
    section 15KP if:
    (a) the person is an authorised person and the person is not
        an authorised civilian and:
        (i)  the acquisition or use is in accordance with an authority and:
        (ii) the acquisition or use is in the course of duty or:
    (b) the person is an authorised civilian and:
        (i)  the acquisition or use is in accordance with an authority and:
        (ii) the acquisition or use is at the direction by the person's
             supervisor under the authority
```

As can be seen from the above, this *ylegis* mode of presenting the codification of a section is even closer to how legislation normally appears than are the equivalent set of rules written using *yscript*.

If legislation was drafted in this manner, it would be *both* legislation and code. For a more detailed example, the complete NSW *Hairdressers Act 2003* can be written in *ylegis* as follows:

```
1. Name of Act
This Act is the Hairdressers Act 2003.

2. Commencement
This Act commences on a day or days to be appointed by proclamation.

3. Hairdressers must be qualified
An individual may not act as a hairdresser in New South Wales if—
   (a) the individual receives a fee, gain or reward for hairdressing
       services; and
   (b) the individual is not qualified to act as a hairdresser; and
   (c) section 3 applies.

4. When is an individual "qualified to act as a hairdresser"?
(1) An individual is qualified to act as a hairdresser if—
    (a) the individual has been awarded an authorised qualification by a
        registered training organisation; or
    (c) a determination has been made under section 37 of the Apprenticeship
        and Traineeship Act 2001 that the individual is adequately trained
        to pursue the recognised trade vocation of hairdressing (because the
        hairdresser has acquired the competencies of the recognised trade
        vocation); or
    (d) the individual has held, or been taken to have held, a licence under
        Part 6 (Regulation of the hairdressing trade) of the Shops and
        Industries Act 1962 and the licence was limited to carrying out beauty
        treatment only.
(2) An individual has been awarded an authorised qualification by a registered
    training organisation if—
    (a) the individual holds a Certificate III in Hairdressing and: the
        Certificate III in Hairdressing is nationally endorsed; or
```

(b) the individual holds a qualification prescribed by the regulations.

5. Prohibition on unqualified hairdressers does not apply to apprentices, health care professionals or certain others
Section 3 does not apply if—
(a) the individual is acting as an apprentice hairdresser under the direct control and supervision of qualified hairdresser; or
(b) the individual is acting as a hairdresser when engaged in their practice as a medical or health care professional; or
(c) the individual is acting as a hairdresser when providing care for elderly or disabled people; or
(d) the individual is acting as a hairdresser in accordance with other circumstances as prescribed by the regulations.

6. Apprenticeship and Traineeship Act 2001 not affected
The operation of the Apprenticeship and Traineeship Act 2001 is not affected by this Act.

7. Information and documents may be required
(1)(a) The authorised officer may serve a notice requiring the individual to produce specified documents for inspection or copying at any place nominated in the notice.
(b) The authorised officer may serve a notice requiring the individual To provide the information specified in the notice.
(2) the individual is guilty of an offence under section 7(2) if:
an authorised officer has served a notice under section 7 and:
the individual has not complied with the notice and:
the time for compliance specified in the notice has expired.
If: the individual is guilty of an offence under section 7(2) then:
the individual is liable to a penalty of 20 penalty units under section 7
(3) 'authorised officer' means an investigator appointed under section 18 of the Fair Trading Act 1987 or an officer of a Government Department who is authorised by the Minister for the purposes of this section.

8. Proceedings for offences
(1) Proceedings for an offence under this Act may be dealt with summarily before the Local Court.
(2) Proceedings for an offence under this Act may be instituted only by the Minister or by a person duly authorised by the Minister in that behalf, either generally or in a particular case.

9. Regulations
The Governor may make regulations, not inconsistent with this Act, for or with respect to any matter that by this Act is required or permitted to be prescribed or that is necessary or convenient to be prescribed for carrying out or giving effect to this Act.

11. Repeal of Hairdressing Regulation 1997
The Hairdressing Regulation 1997 is repealed.

12. Review of Act
(1) The Minister is to review this Act to determine whether the policy objectives of the Act remain valid and whether the terms of the Act remain appropriate for securing those objectives.
(2) The review is to be undertaken as soon as possible after the period of 5 years from the date of assent to this Act.
(3) A report on the outcome of the review is to be tabled in each House of Parliament within 12 months after the end of the period of 5 years.

ylegis can process this version of the *Hairdressers Act* without further modification and produce equivalent executable *yscript* code.[25]

5.3. Practicalities of implementation: Authoritative status of code

The issues involved in having legislative drafting offices adopt this style of drafting, simultaneously as legislation and code, and the advantages that could be obtained from so doing, are beyond the scope of this article.

Legislation provided in online official databases usually now has authoritative status.[26] For this approach of 'drafting rules as code' to have its full effect, the authoritative status of legislation would have to extend to when it is used as code. Governments would have to commit to standing by such conclusions as authoritative statements of the effect of the legislation. At the least there would need to be provision that any person who relied upon such an authoritatively-generated conclusion could not be prejudiced because of that reliance.

6. From legislation to code: (iii) Converting existing legislation to authoritative code

A hybrid of the two previous approaches, the conversion of existing legislation (as discussed in (i)) into authoritative code (as discussed in (ii)) would be most likely to apply only to selected existing statutes, where governments saw very large-scale benefits arising from making a code version authoritative and were willing to give a converted version the authoritative status discussed above. This could occur with legislation on which there was very high consumer reliance, and very high levels of consultation could be expected.[27]

7. Conclusions and future work

In this paper we have proposed an approach to analysing the nature of existing legal rules, particularly legislative rules that regards legislation as being fundamentally just a set of related propositions.

Taking that approach, we propose a method for representing these rules, using the *yscript* language, that allows the *yscript* program to process these rules so as to produce 'consultations' to determine the values of goals which the legislation is capable of determining.

We have described progress that has been made in automating this process to create *yscript* rules from existing legislation, using the *ylegis* program.

---

[25] This can be tested with the DataLex Application Development Tools <http://datalex.org/dev/tools/>.

[26] For example, see <https://pco.nsw.gov.au/accessing-legislation.html>.

[27] For example, this could be done with the NSW Government Community Gaming app <https://www.fairtrading.nsw.gov.au/community-gaming>. The DataLex version of the same legislation is explained at AustLII Media Release 'Smart AI: AustLII's DataLex turns NSW gaming Regulations into code in 24 hours,' 2 October 2020 <http://classic.austlii.edu.au/austlii/announce/2020/1.pdf>.

Based on this experience, we also demonstrated how the drafting of legislation could be changed such that appropriately drafted legislation is readable and understandable by humans and also usable by machines.

We conclude by suggesting that there is now some evidence that these processes can be generalised ('scaled up') to deal with the conversion or production of large bodies of legislation, and that this has considerable value.

7.1. Future work

The *ylegis* software is evolving rapidly, in terms of the variety of structural forms of legislation that it can convert into *yscript* code, and this work will continue.

The overall methodology for dealing with existing legislation will continue to improve to make the conversion of existing material faster and more efficient. This will facilitate larger-scale legislative conversions (a whole single statute, or a set of statutes) and potentially make a significant contribution that will help realise the potential of Rules as Code as a technology generally.

In relation to new legislation, it is hoped that the approaches set out in this paper may also assist in drafting laws that are simultaneously authoritative legislative rules and code.