www.austlii.edu.au

Australasian Legal Information Institute
A joint facility of UTS and UNSW Faculties of Law

Level 14, 61 Broadway, Ultimo NSW 2007
PO Box 123 Broadway NSW 2007

Tel: +61 2 9514 4921
Fax: +61 2 9514 4908
Email: feedback@austlii.edu.au

Applying the Rule of Law in Automated Decision Systems through Rules as Code (AustLII's Submission to the Robodebt Royal Commission)

Andrew Mowbray, Philip Chung and Graham Greenleaf[*]
Australasian Legal Information Institute (AustLII)

9 February 2023

[*] Andrew Mowbray is Professor of Law & Information Technology, University of Technology Sydney and Co-Director, AustLII; Philip Chung is Associate Professor of Law, UNSW Sydney and Executive Director, AustLII; Graham Greenleaf AM is Professor of Law & Information Systems, UNSW Sydney, and Co-Founder & Senior Researcher, AustLII.

Introduction: Automated Decision Systems (ADS) & the Rule of Law

All administrative decision making including those made on an automated basis should be based on the application of law (and in particular, legislation and other forms of regulation). As part of this requirement, there needs to be transparency and explanation of the processes and reasons for decisions both of fact and of law.

The approach and software used in the context of Robodebt was similar in many respects to other Automated Decision Systems (ADS) that are in everyday and widespread use. These systems are generally built manually or using machine learning to rely on factual artefacts to support decision making. They generally are not directly based on legislation or, if they are, this is not transparently encoded in a way that allows for their legal accuracy to be determined. In most cases, explanation facilities of ADS are limited (or non-existent).

Rules as Code (RaC) is a technology that can be used to explicitly represent and apply rules such as legislation in a range of applications. This approach can be used in automated decision making systems to provide a "rule of law" basis for the operation of this type of software. Apart from ensuring that decision making takes account of, and is driven by, law, this approach also allows for explanations that can be expressed in terms of the legal basis of decisions being made.[1]

In this Submission, we develop recommendations which expand on how RaC-compliant legislation can be developed, and how it can be incorporated in automated decision systems. First, we will set out a number of fundamental principles which we propose should apply to all ADS developed by or for the Australian government. Second, we add a number of important principles which should be achieved wherever possible. Finally, in an Appendix, we illustrate how AustLII's DataLex development environment and yscript representation language implements these principles to provide an example of good practice.

The Terms of Reference for the Commission includes "The Royal Commission will be able to make any recommendations it considers appropriate. This includes ways to prevent any public administration failures identified from happening again." We consider that adoption of AustLII's submissions would be effective to avoid Robodebt-like errors from recurring.

Background to AustLII and DataLex

The Australasian Legal Information Institute (AustLII) is a joint facility of the Faculties of Law at UTS and UNSW, and has been in operation since 1995. AustLII's mission is to promote free and effective access to law in Australian and internationally. As part of its activities, AustLII operates the largest and most relied upon free access legal research service[2] that provides free access to all significant Australian legislation and case law. This service receives over 230 million page accesses annually (about 700,000 accesses per day) from over 6.5 million distinct hosts.

AustLII's research work on Rules as Code has developed DataLex which is an applications development environment that is suitable for building RaC codebases and applications. At the heart of the DataLex approach is a language called "yscript" which supports the declarative

[1] For an explanation of 'Rules as Code' see A. Mowbray, P. Chung and G.Greenleaf 'Representing legislative Rules as Code: Reducing the problems of 'scaling up' " (2021) *Computer Law & Security Review* <https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3981161>;

[2] http://www.austlii.edu.au

representation of rules using a quasi-natural-language ("English-like") codebase syntax. The language facilitates a close relationship between rules and code that makes code easier to write and maintain, and provides for high levels of transparency. This representation allows sophisticated dialogs and explanations to be generated directly from the code.[3]

Fundamental principles in implementing ADS

Any ADS developed and provided by the Australian Government should implement the four following fundamental principles. This submission applies to systems relating to decisions about individuals (and thus involving 'personal information' as understood in the *Privacy Act 198*8) as well as systems that do not deal with personal information.

The nine principles listed in this and the next section are those which it is necessary or desirable to observe when building an Australian government ADS. There are other principles (or differently named versions of the same principles) which have also been suggested, often arising from the requirements of administrative law.[4] These often overlap with the principles proposed below.

(i) ***Acknowledgement*** that an ADS implements aspects of the legal system.

The starting point for any Australian Government ADS should be an explicit written statement of the relationship between the ADS and the rule of law. It should acknowledge that the ADS implements specific aspects of the legal system, comprised of specified (named) statutory provisions, regulations, Departmental manuals, interpretations of case law, and the like. The document that sets out the starting point must be available to the public, because of the transparency principle.

(ii) ***Legal requirements*** of explainability must be observed.

As a corollary of (i), any Australian Government ADS must include in this acknowledgement, confirmation that it meets any Australian legal provisions concerning such matters as explainability, FOI status, and privacy legislation compliance, and should itemise such requirements.

Legal requirements of explainability are well known in the EU (particularly GDPR[5] article 22) but are not restricted to the 30 countries of the EU/EEA. Many other countries now have such requirements (or related controls on automated decision systems). At least 25 jurisdictions outside the EU[6] have such controls, influenced by the EU's DPD or GDPR, but often with different terms. So more than 50 countries now have some controls on automated decision systems.

---

[3] Mowbray, Chung and Greenleaf 'Explainable AI (XAI) in Rules as Code (RaC): The DataLex approach' (2022) *Computer Law & Security Report*

[4] See, for example, T. de Sousa, P. Andrews and L. Bennet Moses 'Submission – Royal Commission into the Robodebt Scheme', suggesting that 'trustworthy automated decisions in government' should be Transparent, Traceable, Accountable, Appealable and Beneficial (see submission p3, and Appendix)..

[5] (EU) General Data Protection Regulation (GDPR)

[6] China, Macau, Philippines (Asia - 3); Ghana, South Africa, Morocco, Kenya, Uganda, Algeria (Africa - 6); Brazil, Argentina, Uruguay, Peru (Latin America – 4); California (1); Albania, Turkey, Ukraine, Azerbaijan, Bosnia & Herzegovina, Serbia, Russia (Europe – 7). Only the 'Top 50% by GDP' of jurisdictions with data privacy laws have been considered, so that the actual number will be higher. The UK, and the three Channel Island jurisdictions should also be added.

In a country like Australia, we require 'explainability' of computing systems such as an ADS that have important consequence for our lives, for a number of reasons. As we have pointed out previously,[7] those reasons are highly dependent on our relationship to the ADS:[8]

- Those individuals affected directly by a decision or prediction will want to understand how it was reached, in order to be convinced of its fairness (among other things) and the explanation will need to be understandable by them;
- Organisations utilising such systems need to understand how decisions they administer are determined, and (usually) to keep a record of this, with the explanation being understandable to those who run the system, and also to any parties who review such decisions;
- Such organisational users of ADS also need to encourage individual users of the systems they provide to trust (where justified) the outcomes of use of those systems, and this trust may depend on explainability;
- System designers (and those who have to maintain systems) need to understand how systems work in order to improve them, or to debug them where necessary;
- All parties involved need to be convinced that the system is operating in ways which do not breach legal requirements;
- 'Social licence' for such ADS to be used in making important decisions requires a level of trust from society as a whole.

Australian law does not at present include a general statutory requirements of 'explainability' which would cover all these relationships to an ADS. It would be desirable if the Commission made recommendations for changes which addressed as many of these relationships as possible. As suggested above in (i), this could be part of the acknowledgement accompanying an ADS.

A survey of the provisions under Australian administrative law that could require explainability in some form[9] has shown that there are dozens of provisions in Australian federal law which authorise the making of decisions by automated means, but they then deem those decisions to have been made by a human decision-maker (for example, the Secretary of a government Department), thus bringing into play all the requirements of federal law relating to the making of decisions.

(iii) ***Transparency*** of statutory representation and other aspects of the ADS.

The statutory provisions, and other aspects of the rule of law embodied in the ADS must be represented in an explicit representation so that they can be understood by people other than the system developer, including those responsible for administering the use of the ADS, those auditing it or otherwise critiquing it, and at least some of those to whom it is applied. To the maximum extent possible, other aspects of the ADS should also be represented in a similar way.

As an example of what is possible in this respect, one of the central aims in the development of DataLex's *yscript* language (see Appendix) was to develop a form of representation that looked as much like natural language as possible. This was done partly to make it easier to

[7] Mowbray, Chung and Greenleaf (2022)

[8] Martin Ebers 'Regulating Explainable AI in the European Union. An Overview of the Current Legal Framework(s)' in: L. Colonna and S. Greenstein (eds.), *Nordic Yearbook of Law and Informatics 2020: Law in the Era of Artificial Intelligence*, p.4 <https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3901732>.

[9] M. Guihot and L. Bennett Moses *Artificial Intelligence, Robots and the Law* (Lexis Nexis Australia , 2020), pp 166-175..

represent legal rules, but also to make the code more transparent. Even if someone does not have a detailed knowledge of the system, 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.

(iv) ***Separate representation*** of statutory provisions in an ADS.

The statutory provisions, and other aspects of the legal system embodied in the ADS must be represented separately from other aspects of the ADS (such as provisions controlling order of implementation) so that they can be understood and considered separately from the implementation aspects.

Desirable principles in implementation of an ADS

There are other highly desirable principles which should be adopted wherever possible in the development of an ADS, but none of them are essential, unlike the four fundamental principles above. Each of these desirable aspects of a government ADS is first described in general terms, and this is then followed by an example of how it is implemented in the DataLex environment (with further details in the Appendix).

(v) ***Open source*** representation languages are preferable.

If all the components of the environment in which an ADS is developed are open source, this prevents the requirement of transparency being frustrated by the use of proprietary systems.

In the DataLex environment (see Appendix) the yscript language is open source. The yscript interpreter (the software that runs apps written using the yscript language) and yscript library are available as open source[10] under an Affero GPL licence.[11] (see Appendix)

(vi) ***Isomorphic representations*** between statutory provisions and code are desirable.

There are many ways by which an ADS system can encourage developers to maintain a close relationship between the code of the application and the legislation being represented, preferably a one-to-one relationship at the section level (or equivalent). Such isomorphism makes the code easier to build, to maintain, to audit and to understand. This is important for transparency, explanation and maintainability. It also 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.

In the DataLex environment (see Appendix), the rule-based structure of yscript encourages and supports isomorphism of real-word rules (particularly legislation) into yscript code. It is intended that yscript applications can be directly created and maintained by lawyers.

(vii) ***Explanatory features*** of the codebase should be supported.

When a government ADS uses a dialog with either a subject expert (an administration official) or a member of the public (often then called a 'chatbot'), it should as far as possible provide explanatory features which show why questions are being asked, and how interim conclusions

[10] The yscript source is available via the main DataLex page < main http://datalex.org/> under "Source". The link is: https://datalex.org/src/ys/ys-latest.tar.gz

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

have been arrived at. It should allow user to test out hypothetical answers. It should generate a final report which sets out all steps necessary to the final conclusion being reached.

Using the DataLex system as an example, yscript does not require questions and explanations to be written separately from the rules that make up an application. Questions and explanations (responses to 'Why?', 'How' and 'Conclusions') are generated on the fly by the yscript interpreter from the rules when the app is running. There is therefore no textual 'baggage' which must be written (and maintained) in addition to its rules.

When yscript code is interpreted and run, it results in a dialog or *consultation* (see Appendix for examples). A series of questions are asked, and conclusions are drawn. During the consultation, the user can interrogate the system as to why questions are being asked (*Why?),* to explain how conclusions have been reached (*How?),* and to delete facts so that all conclusions depending on the fact are re-evaluated. Facts previously provided by the user can be deleted (*'Forget*'). The system also uses all information available to it, from the codebase and user-supplied facts, to suggest other relevant *Related Materials* which it extracts from the whole AustLII system and displays the most relevant results. Users can test a hypothetical answer to a question through selection of '*What if?'* When a session completes, a *Report* is generated to give an answer to the original goals set for the consultation, and to explain why this answer follows from the user-provided information given during the consultation. These reports may be quite lengthy. They demonstrate the value of an 'English-like' interface by showing how various and complex the interface to a government ADS can be.

(viii) ***Links from the codebase*** to specific statutory materials.

In all interactions with users of a government ADS, as far as possible there should be access to primary legal materials (legislation and cases) so that the user can (a) understand those materials in order to better answer questions being asked, and (b) understand conclusions being drawn during the consultation. This makes the ADS more transparent to the user.

The DataLex interface automatically links all references to primary materials (legislation and caselaw) in the consultation dialogs to those primary materials on AustLII.

(ix) ***Public availability*** of the codebase.

It is perhaps obvious, but worth re-stating, that the whole codebase for a government ADS should be available to the public. It is information about how government programs are being administered, and should be pro-actively provided to the public, not only via a FOI request.

The DataLex interface displays the whole codebase at any time, on user request. There is also a mode in which the interface can be run during a consultation so that it displays the particular part(s) of the codebase that are in use at each point in the consultation.

Recommendations

AustLII submits that the Royal Commission should propose that any Australian Government ADS:

1. Must commence with an explicit written acknowledgement that an ADS implements specific aspects of the legal system, which are identified, thereby stating its relationship to the rule of law.
2. Must include in this acknowledgement, confirmation that it meets any Australian legal requirements concerning such matters as explainability, FOI status, and privacy legislation compliance.
3. Must be implemented in a transparent way so as to maximise the extent to which its effects can be understood.
4. Must represent the aspects of the legal system embodied in the ADS separately from other aspects of the ADS.
5. Should be developed in an environment in which all the components are open source, and are not provided by proprietary software.
6. Should maintain a close relationship between the code of the application and the legislation being represented, if possible close to a one-to-one relationship at the section level.
7. Should as far as possible provide explanatory features such as those which show why questions are being asked, and how interim conclusions have been arrived at.
8. Should as far as possible provide links to primary legal materials (legislation and cases) from the dialogs generated by the ADS.
9. Should pro-actively make the code of the ADS available to the public.

Appendix: DataLex implementations

AustLII has created an applications development environment ('DataLex') which can be used to develop ADS systems, using the *yscript* language, which implement the principles set out in this submission. Various running examples of applications can be tested at http://datalex.org ,[12] including examples on legal subjects such as the Mandatory Bargaining Code (Cth), NSW Community Gaming Regulation, the Modern Slavery Act 2018 (Cth), and the NSW Hairdressers Act. The yscript code for each application can be found on the DataLex site, and the applications can be run ('Consultations'). Users can develop and run their own test applications using the *DataLex Application Developer Tools*.[13] Papers explaining the DataLex approach and software are available.[14]

The following extract is from 'Explainable AI (XAI) in Rules as Code (RaC): The DataLex approach' (2022) *Computer Law & Security Review*[15]

5. DataLex: RaC meets XAI, in the public domain

DataLex is an applications development environment created by the university-based Australasian Legal Information Institute (AustLII) which is suitable for developing RaC codebases and applications.[16]

There are many languages and approaches for representing legislation and other forms of rules as code. Some systems are interactive and use dedicated rule editors. The BLAWX system developed by Jason Morris, for example, uses a "drag and drop" graphical environment for constructing rules.[17] In OpenFisca, rules are represented using a subset of the Python programming language.[18] Other systems use languages to represent rules symbolically. The Defensible Deontic Logic (DDL) language developed by CSIRO's Data61, for example, uses a symbolic approach to represent rules in an extended deontic logic format.[19]

DataLex uses a language called "yscript" which facilitates the declarative representation of rules using a quasi-natural-language codebase syntax (ie one resembling English). The language facilitates a close relationship between rules and code that makes coding easier and provides for high levels of transparency. As explained below, this allows sophisticated dialogs and explanations to be generated directly from the code.

---

[12] DataLex applications built with yscript <http://datalex.org>

[13] *DataLex Application Developer Tools* http://datalex.org/dev/tools/

[14] For more details, see 'Representing legislative Rules as Code: Reducing the problems of 'scaling up' " (2021) *Computer Law & Security Review* < https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3981161>; 'Building DataLex decision support systems: A tutorial on rule--based reasoning in law' (2017) <https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3034430>; 'Law as Code: Introducing AustLII's DataLex AI' (2021) <ttps://papers.ssrn.com/sol3/papers.cfm?abstract_id=3971919>

[15] The full article 'Explainable AI (XAI) in Rules as Code (RaC): The DataLex approach' is at <https://papers.ssrn.com/sol3/papers.cfm?abstract_id=4093026 >

[16] For an overview of the DataLex components, see A. Mowbray, G, Greenleaf, and P. Chung, *Law as Code: Introducing AustLII's DataLex AI* (November 16, 2021). UNSW Law Research Paper No. 21-81, <https://ssrn.com/abstract=3971919>.

[17] See https://www.blawx.com.

[18] See https://openfisca.org/doc/coding-the-legislation/10_basic_example.html

[19] Governatori, G., Rotolo, A., Riveret, R. (2018). A Deontic Argumentation Framework Based on Deontic Defeasible Logic. In: Miller, T., Oren, N., Sakurai, Y., Noda, I., Savarimuthu, B., Cao Son, T. (eds) PRIMA 2018: Principles and Practice of Multi-Agent Systems. PRIMA 2018. Lecture Notes in Computer Science, vol 11224. Springer, Cham. https://doi.org/10.1007/978-3-030-03098-8_33

For RaC codebases, human rules (such as sections contained in legislation or regulations that are being encoded) are generally represented isomorphically (that is, human rules are mapped on a one-to-one basis to yscript rules). Each rule element (be it a premise or conclusion) is represented as a "fact" which is often in the form of a proposition. Rules set out the relationships between facts (for example, "if some fact applies as well as another fact then some conclusion can be drawn").

There is no separate coding of what a codebase should "do", nor for specific explanations or of other system dialog. All interactions are generated automatically from the facts contained in the rules, in dialogues generated 'on the fly' when the system is in operation.

The DataLex approach does not require the involvement of software experts and codebases can be created directly by lawyers or legal drafters.

Applications and codebases can be collaboratively developed and maintained within the AustLII Communities environment and integrate with AustLII using automated hypertext links to Australian legislation and cases.[20] Access to the DataLex development environment and documentation are available from the DataLex website.[21]

RaC based systems built using the DataLex system meet the seven desirable features of XAI discussed in the previous section as follows:

the yscript language is open source

The yscript interpreter (the software that runs apps written using the yscript language) and yscript library are available as open source[22] under an Affero GPL licence.[23]

yscript uses a quasi-natural language representation for applications

yscript[24] uses a quasi-natural-language 'English-like' syntax, which is easy to learn and use, supports declarative and imperative coding, and produces natural English dialogs (consultations). While yscript is a flexible general purpose language, it is particularly useful for representing legislation and other rules which are comprised of a structured set of propositions.

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.

---

[20] DataLex Community web pages <http://austlii.community/wiki/DataLex/>.

[21] See http://datalex.org

[22] The yscript source is available via the main DataLex page < main http://datalex.org/> under "Source". The link is: https://datalex.org/src/ys/ys-latest.tar.gz

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

[24] 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").

yscript encourages isomorphic representation

The rule-based structure of yscript encourages and supports isomorphism of real-word rules (particularly legislation) into yscript code, important for transparency, explanation and maintainability). It also 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.

An extract from the Australia's Foreign Relations (State and Territory Arrangements) Act 2020 (Cth) codebase is shown in Figure 2.

```
RULE Section 10 - Core foreign arrangements PROVIDES
the arrangement is a "core foreign arrangement" under section 10(2) ONLY IF
    the arrangement is a "foreign arrangement" under section 6 AND
    the Australian entity is a "core State/Territory entity" under
    section 10(3) AND
    the non-Australian entity is a "core foreign entity" under section 10(4)

RULE Section 10(3) PROVIDES
the Australian entity is a "core State/Territory entity" under section 10(3)
ONLY IF
    section 7(a) applies OR
    section 7(b) applies OR
    section 7(c) applies
```

Figure 1 Extract of a DataLex codebase written in yscript

yscript does not require questions and explanations to be written separately from the rules that make up an application. Questions and explanations (responses to 'Why?', 'How' and 'Conclusions') are generated on the fly by the yscript interpreter from the rules when the app is running. There is therefore no textual 'baggage' which must be written (and maintained) in addition to its rules.

It is intended that *yscript* applications can be directly created and maintained by lawyers.

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 more often are 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.

Codebases in yscript are generally publicly available

It is desirable that apps written in yscript, so that they can be run with the interpreter, should be licensed under some form of open content licence, and made available for any user to read,[25]

Explanatory features of yscript apps, when running

When yscript code is interpreted and run, it results in a dialog or *consultation*. A series of questions are asked, and conclusions are made. During the consultation, the user can interrogate the system as to why questions are being asked (Why?), to explain how conclusions have been

[25] The yscript code for many apps is available at <http://datalex.org>.

reached (How?), and to delete facts so that all conclusions depending on the fact are re-evaluated.

The DataLex user interface uses the DataLex software and codebases, the linkages provided by the Communities environment, and user input, to provide legal advisory systems in operation.

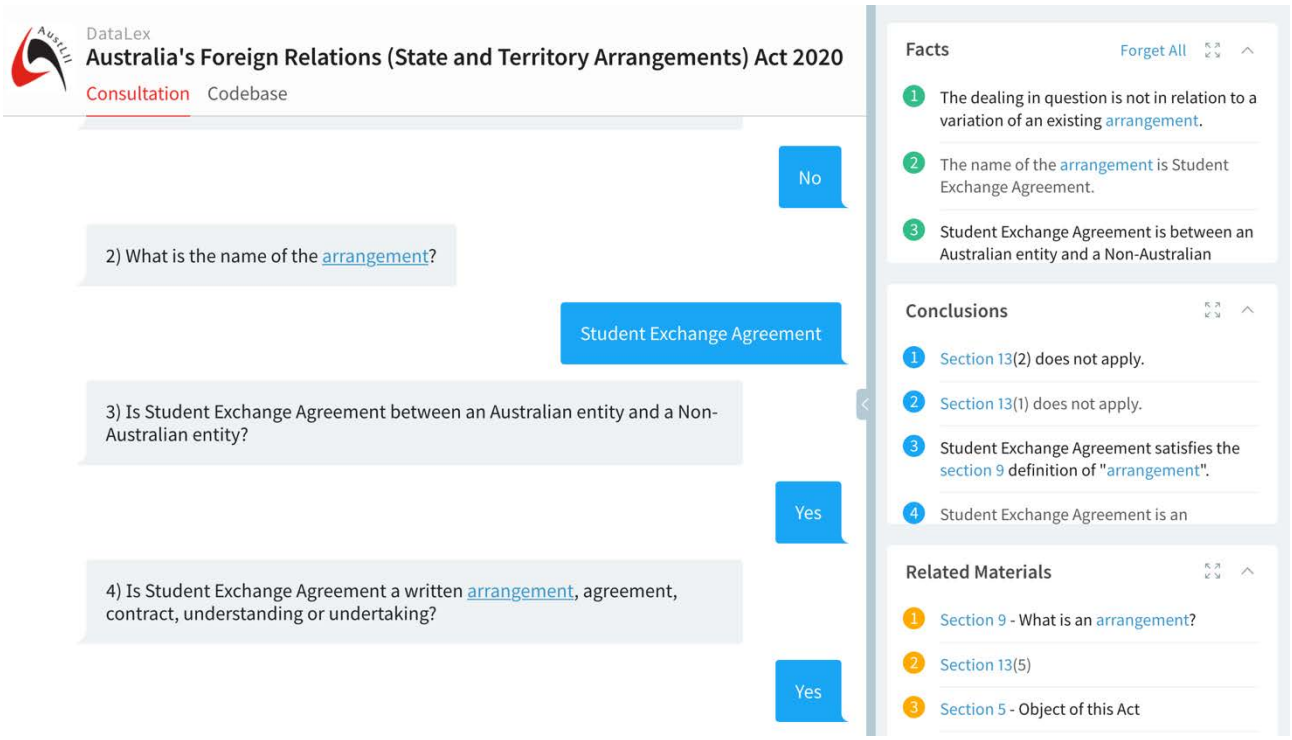

Figure 2 DataLex user interface features: Consultations, Facts, Conclusions, Related Materials

From the above user interface extract, and further interface extracts below, eight elements of DataLex's 'explainability' are shown:

1. Questions, Facts, Conclusions, and Reports are all generated from the codebase plus user-provided facts, are in an understandable form ***('English-like')*** and are available on screen at all times.
2. Facts previously provided by the user (shown under Facts on the right side) can be deleted (***'Forget?'***), by selecting the number of user-provided fact, and questions are then re-asked to re-establish a value for that fact.

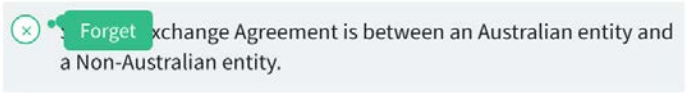

Figure 3 DataLex 'Forget' a user-provided fact during consultation

3. Reasons for why a Question is being asked can be requested ***('Why?')***, with the reasons being generated on the fly from the relevant rules.
4. Conclusions are shown on the right-hand side. Selection of a numbered conclusion results in a ***'How'*** explanation of that conclusion being presented, as shown in Figure 5.

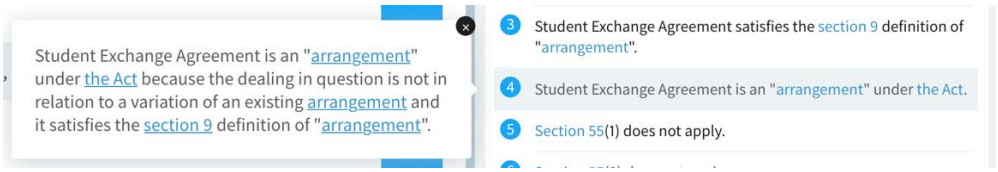

Figure 4 DataLex 'How' explanation during consultation

5. The system also uses all information available to it, from the codebase and user-supplied facts, to suggest other relevant ***Related Materials*** which it extracts from the whole AustLII system and displays the most relevant results.
6. Users can test a hypothetical answer to a question through selection of '***What if?***' (that is, 'what happens if I answer this way?'). When *What If?* is turned off, the original question will be asked again.

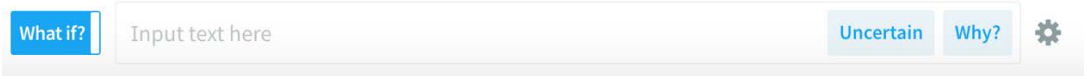

Figure 5 DataLex 'What if?' option selected during consultation

7. The consultation can be set in ***Verbose*** mode, so that each of the rules that are fired is displayed as they are fired, enabling the user to see the relationship between the rules and the interface.

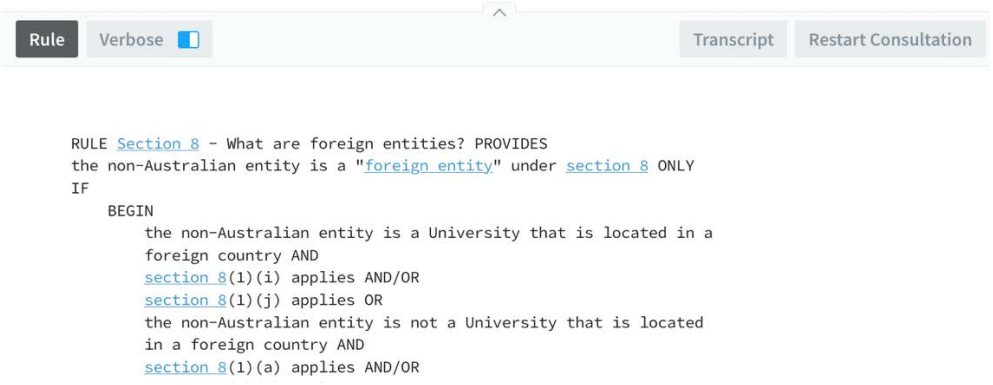

Figure 6 DataLex 'Verbose' mode enabled showing current rule being considered

8. When a session completes, a ***Report*** is generated to give an answer to the original goals set for the consultation, and to explain why this answer follows from the user-provided information given during the consultation. These reports may be quite lengthy.

These features are important to achieving the objective of 'explainable AI'. They demonstrate the value of an 'English-like' interface by showing how various and complex the interface to an AI system can be. *Why?* and *How?* explanations – and *What If?* – show how the interactions in a consultation can be constantly explaining aspects of the system's reasoning to a user. The explanations available for each Conclusion, and their being amalgamated into a lengthy Report at the end of the consultation, both generated from the rules that have been fired, and the user-provided answers to questions.

Explanations to link to source

The DataLex interface automatically links all references to primary materials (legislation and caselaw) to AustLII where available.

Using ylegis to make law and code identical

The Hairdressers Act 2003 (NSW) example also illustrates two ways of representing Rules as Code in DataLex. One way is to code existing legislation in yscript as part of a conversion

process (the majority of current example codebases are written in yscript). Another approach is to write the legislation in a natural language format that can also be executed as code (the ylegis format). A pre-processor program (ylegis) takes a section of legislation (or multiple sections) and converts it automatically into a 'first draft' of yscript code for those legislative provisions, which can immediately be run by the yscript interpreter.[26]

The ylegis format can be used to write legislation in a form that is close to existing conventional legislative drafting but requires relationship between separate propositions be defined by a formal set of connectors (such as 'and:' and 'or:') and these operators are used to connect subsections and subclauses each of which contains only a single proposition.[27]

Using s4(1) of the Hairdressers Act 2003 (NSW) as an example, the following three figures show the original drafting (Figure 8), the section being coded in yscript (Figure 9) and the section rewritten in the ylegis format (Figure 10).

4 When is an individual "qualified to act as a hairdresser"?

(1) For the purposes of this Act, an individual is **"qualified to act as a hairdresser"** if any one or more of the following applies to the individual--

(a) the individual has been awarded an authorised qualification by a registered training organisation,

(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 individual has acquired the competencies of the recognised trade vocation),

(d) the individual has at any time held, or been taken to have held, a licence under Part 6 (Regulation of the hairdressing trade) of the *Shops and Industries Act 1962* , other than a licence limited to carrying out beauty treatment only.

Figure 7 Section 4(1) of the Hairdressers Act 2003 (NSW)

[26] For details and examples, see A. Mowbray, P. Chung, and G. Greenleaf, 'Representing legislative Rules as Code: Reducing the problems of 'scaling up'' (December 9, 2021). <https://ssrn.com/abstract=3981161>

[27] See Mowbray, Chung, Greenleaf "Representing legislative Rules as Code: Reducing the problems of 'scaling up'"

```
RULE Section 4 - When is an individual "qualified to act as a hairdresser"
PROVIDES
SUBRULE Section 4(1)
SUBRULE Section 4(2)

RULE Section 4(1) PROVIDES
the hairdresser is qualified to act as a hairdresser ONLY IF
    section 4(1)(a) applies OR
    section 4(1)(c) applies OR
    section 4(1)(d) applies

RULE Section 4(1)(a) PROVIDES
section 4(1)(a) applies ONLY IF
    the hairdresser has been awarded a hairdressing qualification AND
    the qualification was awarded by a registered training organisation AND
    the qualification is an "authorised qualification" under section 4(2)

RULE Section 4(1)(c) PROVIDES
section 4(1)(c) applies ONLY IF
    a determination has been made under section 37 of the Apprenticeship and
    Traineeship Act 2001 that the hairdresser is adequately trained to pursue
    the recognised trade vocation of hairdressing (because the hairdresser has
    acquired the competencies of the recognised trade vocation)

RULE Section 4(1)(d) PROVIDES
section 4(1)(d) applies ONLY IF
    the hairdresser 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
```

Figure 8 Section 4(1) of the Hairdressers Act 2003 (NSW) in yscript format

```
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.
```

Figure 9 Section 4(1) of the Hairdressers Act 2003 (NSW) in ylegis format

When drafting legislation using the ylegis approach, the resulting text is both legislation and code.[28] This means that if a piece of legislation is written using the ylegis format, an intermediate yscript version will be generated automatically during execution on which explanations will be based without having to code a separate version of the legislation in yscript.

This is a different approach to explainability: the legislation *is* the code. Legislation drafted in ylegis is simultaneously the human-readable version of the legislation, and also the machine-processable version of the legislation. Both versions will give rise to the same questions of

[28] See <http://austlii.community/foswiki/DataLex/HairdressersActKB>.

statutory interpretation in relation to the meaning of terms that are used. Legislation drafted in ylegis format has the same desirable features of explainability as legislation represented in yscript (as set out in the following section).

DataLex apps as explainable AI

Explainability is demonstrated in the numerous applications written in *yscript* that are on the DataLex web pages,[29] including examples on legal subjects such as the Mandatory Bargaining Code (Cth), NSW Community Gaming Regulation, the Modern Slavery Act 2018 (Cth), and the NSW Hairdressers Act. The yscript code for each application can be found on the DataLex site, and the applications can be run ('Consultations'). Users can develop and run their own test applications using the *DataLex Application Developer Tools*.[30]

In summary, we can now check which of the seven desirable features of explainability (see section 3.4) are provided in DataLex's implementation of RaC:

> *Transparency* – yscript interpreter software is open source; yscript language for app development is humanly-understandable ('quasi-natural language' or 'English like'); When yscript apps run, they are able to explain their questions, reasoning and conclusions while running.
>
> *Traceability* – yscript rules can have as close to an isomorphic relationship as is achievable to the sources on which the coded rules are based.
>
> *Availability* – yscript code bases for apps should be publicly available, and re-usable by others (and are for many examples).
>
> *Sustainability* – Run-time explanations in yscript apps are generated from the logic and text of the app (not from pre-formulated separate answers or questions).
>
> *Links to the legal sources* justify all run-time explanatory features.

The same qualities apply to legislation written in ylegis format.

One desirable feature ('accountability') requires an answer to the question of whether legal authorities that make codebases available also make themselves *accountable* for the codebase being an accurate counterpart to its legal text equivalent. It is a question for governments, not RaC developers.

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

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