Automatically Processing Amendments to Legislation

Timothy Arnold-Moore

Multimedia Database Systems Research Group
Collaborative Information Technology Research Institute
Royal Melbourne Institute of Technology
Melbourne, Australia
email: tja@cs.rmit.edu.au

Abstract

This paper proposes an architecture for a system which accepts Amending Acts expressed in SGML and produces a database of resulting versions of the Principal Acts, and describes its implementation. It discusses the core natural language processing module which uses an ATN to parse the components of the Acts into a frame representation of amendment actions. This representation is then used to produce database transactions which add the subsequent versions to the database.

1 Legislation in computers

While considerable effort has been expended in improving the quality of systems which retrieve judicial decisions or case law, the tools for producing and managing legislation fail to take advantage of the potential that is available. Pearce reports from a survey of cases in Australia in 1980 that approximately 40% of reported cases ruled on the meaning of some piece of legislation and a further 30% applied some legislation, a total of 70% [14]. He also reports similar results in England. With the continual expansion of the field of coverage of legislation over time, this percentage can only have grown since. Tools which aid the management of legislation have been described elsewhere including the use of logic to normalize statutes and eliminate ambiguity [1], the logical modelling of legislation using semantic nets [8] and production rules [16], the use of expert systems to capture legislative provisions and act as policy manuals for civil servants [10], and the use of a knowledge-based system to construct links between related sections in legislation [11]. With the limited exceptionof expert systems, none of these tools has been adopted as a major tool of government in the creation, management and distribution of legislation in Australia.

There are two fundamental differences between case law and legislation which raise new issues for the providers of computer-aided legal research (CALR) systems. The first is that legislation has a complex structure which follows predefined rules. All Acts contain numbered sections. These sections can themselves contain subsections, paragraphs, subparagraphs, clauses, subclauses and definitions. In larger Acts these sections may be collected in a combination of chapters, parts, divisions and subdivisions. To avoid confusion with the specific meaning of these terms in legislation we collectively describe these as the *elements* of an Act. Few existing CALR systems support the complete range of opportunities this structure provides for retrieval by the content of particular elements and retrieval of elements at an arbitrary level. This structure allows a formal system of reference (or citation) which identifies each element clearly and unambiguously. Contrast this with case law where citation of part of the decision usually relies on referring to some aspect of presentation (e.g. page number) which changes in different reports of the same case.

The second and more important for the purposes of this paper is that its content can change with the passage of time. Sections (or indeed larger and smaller elements) can be added, removed or altered. In Australia, a Principal Act is created in one of two situations. The first is when a new body of law is reduced to legislation creating a new Act where no other existed. The second is where a large scale restructuring of existing legislation is made creating a new Act (or group of Acts) which completely replaces a previous Act or group of Acts. In between, Amending Acts are passed which make alterations to the Principal Acts. This is further complicated by the fact that Amending Acts may refer to more than one Principal Act and an Act

which is a Principal Act may also repeal another Principal Act. The Australian legislators (comprising the Federal Parliament, the Parliaments of the 6 States and 2 Territories and the subordinate legislative bodies and authorities) have adopted the textual style of amendment where amendments require words to be omitted and others inserted [14]. This means that the law at any particular moment of time is captured as a version of the Principal Act as amended. This is in contrast to the English style of referential amendment which requires the Principal Act to be read in different terms. This requires the Principal Act to be read together with all the amendments [14] rather than simply applying the amendments to the Acts.

The major problem for legislation systems is that updating is slow and inclusion of amendments is complicated [17]. The need for tools for assisting in the drafting of legislation has been recognized widely [5, 6, 17, 18] but little appears to have changed in the field.

While only the Principal Acts and the Amending Acts have legal force, lawyers and legal researchers would prefer to have in front of them the law as it stands at the time relevant for their particular problem. Lawyers in Australia and many other common law jurisdictions will be familiar with consolidations or reprints of legislation. Periodically the authorized printer of government legislation will issue a consolidation of a particular Act which presents the Principal Act with all of the amendments applied i.e. with wording changes, and appropriate elements inserted or deleted. Along with applying the amendments, the consolidations often provide either by annotations in the body of the Act or by means of an index or table at the rear, a list of all the amendments which have affected each section allowing a legal researcher to easily trace back through the history of any section. If not there is usually a third party annotation service provided in a separate volume. The consolidations are currently generated by hand in all Australian jurisdictions and, as considerable work is involved in producing them, they are not printed with every amendment. Many law libraries supplement consolidations by pasting subsequent amendments (usually provided by a third party service) into the most recent consolidation thus ensuring that the users have access to the most recent version.

However, lawyers' needs are different from the needs of other disciplines which rely heavily on electronic sources in that they are often interested in the law as it was at some distant time in the past [7]. If a lawyer brings a case to court involving an accident which happened on the first of June, 1981, the relevant substantial law is (usually) the law as it was on that date, not the law as since amended. The ideal would be for law libraries to have a copy of the relevant consolidation in which the appropriate amendments are pasted for every different version of a given Act. This could mean storing many hundreds of copies of some Acts and even the best law libraries can't support this level of service. They usually aim to provide the most recent version together with the Principal Acts and the Amending Acts as enacted by parliament. Legal researchers can then make use of the annotations in the most recent version (or annotation service) to get back to the law as it was at a particular moment in the past. This can be a time-consuming exercise if the Act is frequently amended or the matter being researched is quite old.

Current CALR systems reflect these changes only by presenting to the user either the Act as it appeared on a particular date, or by presenting the original Act with all of its subsequent amendments, a solution not even as good as that of paper libraries. Unlike paper libraries which would need to store a copy of each Act in every possible version, electronic libraries need not indulge such wasteful duplication. There is great potential for CALR systems not only to present legislation in a format familiar to lawyers (like that of the paper consolidation) but to present it as it would have appeared at any arbitrary point in time with annotations available with the text.

The problems of how to store these various versions in electronic databases are discussed at length elsewhere [2]. This paper addresses the problem of analysing the language and terms of Amending Acts to produce the appropriate changes to the database. It describes an Amendment Processing System (APS) to process Amending Acts describing its architecture and components. The natural language processing component of the system is analysed in further detail looking at some of its sub-networks. Finally some conclusions are presented and some areas for further exploration are suggested.

2 The APS Architecture

Two of the major problems with natural language processing (NLP) by computer are handling ambiguity and the extent of domain knowledge needed by the NLP system to truly 'understand' the text [3]. Legislation is especially difficult in this respect as the possible subject of legislation covers the whole spectrum of human endeavour making the domain virtually unbounded. Ambiguity is also problematic as the number of cases which involve the determination of the meaning of legislation demonstrates. However, if we restrict a system to considering just the language that makes amendments (and ancillary language relevant to amendments) we have a manageable domain. Most jurisdictions have either formal or informal conventions for the language used to describe amendments to Acts to eliminate ambiguity as ambiguity in the description of amendments cannot

be tolerated. This also leads to a reduced vocabulary and a much more predictable structure in the content of particular amending sections. Amending Acts therefore provide an obvious application for processing by computer.

In order to fit this processing into a CALR system we need to fit the various components together. Three components are required (see Figure 1):

1. a text database management system which encodes the structure of Acts in which to store Principal, Amending, and consolidations of the Acts;
2. an NLP module which produces a structured representation of amendment actions from their English description in the Amending Acts; and
3. a text processing module which produces new consolidations from the structured representation of actions and existing versions of the Principal Act.

The text database system needs to manage multiple versions of a single document and to manage highly structured documents [2]. We have chosen to use the Structured Information Manager (SIM) [13] produced at CITRI and marketed by the Ferntree Computer Corporation which, while not directly supporting all versioning functionality desired, can be used to manage versioned text and allows the retrieval of elements within documents or Acts. SIM stores documents in the Standard Generalized Markup Language (SGML) [9], an ISO Standard for document interchange which provides a grammar for describing the structure of documents. SGML has received broad acceptance as an appropriate tool for encoding legislation as well as a variety of other types of document. Many providers of legislation in electronic form use SGML to encode their distribution. Tools exist to convert electronic versions [22] and even paper versions into SGML text. The structure encoding provided by SGML also proves useful in the processing of amendments so an SGML encoding is used as the text input to the NLP module and the text processing module.

The choice of structured representation is crucial to the complexity of the text processing module. The representation must capture particular features of each amendment including:

- the (*Source*) element which makes the amendment;
- the *Target* element of the amendment;
- the *Time* at which the amendment is to commence;
- the *Type* of amendment to be made;
- any *Old* text to be removed;
- which *Occurrence* of the old text is to be removed;
- any *New* text to be added;
- the text/element *Before* any text to be added; and
- the text/element *After* any text to be added.

A representation which allows this with some flexibility to add extra features is the frame representation first described by Minsky [12]. A frame stores a number of features of specified types in a structure which allows extraction of each feature when needed. An example frame is shown in the Appendix as the output of the NLP module.

Having constructed such a frame for each amendment, a text processing module can be constructed using a scripted editor, to apply these frames to the appropriate version of the legislation to produce new versions. SIM provides a scripting language which allows the manipulation of complex, structured documents expressed in SGML. This scripting language has been used to implement the text processing module in the prototype system. The only major component which remains is the NLP module.

3 The NLP Module

We wish to incorporate our NLP module in a system for handling a large database of legislation. Unlike most NLP systems which require a great deal of semantic and syntactic information from the text often with multiple passes over the text to ensure that the syntactic and semantic information is consistent, we need only extract the features for each amendment action from the text. The required 'understanding' of natural language is therefore considerably simpler than that required of more general systems. Efficiency of calculation is very important as the throughput is potentially very large (for example building a database of all Commonwealth legislation from 1901 to the present). Therefore we prefer a simple, one pass method for extracting the information. Such a method is the augmented-transition network (ATN) described by Wood [24] formalizing and extending the work of Thorne *et al.* [20] and Bobrow and Fraser [4]. Some of the advantages of the ATN parsing method described by Wood [24] include:

- it is easily readable by humans;
- it can parse in a single pass by postponing decisions about which route to take;
- it is easily extendible;
- it captures regularities in the parsing process; and
- it is very flexible in the structures that can be generated by parsing.

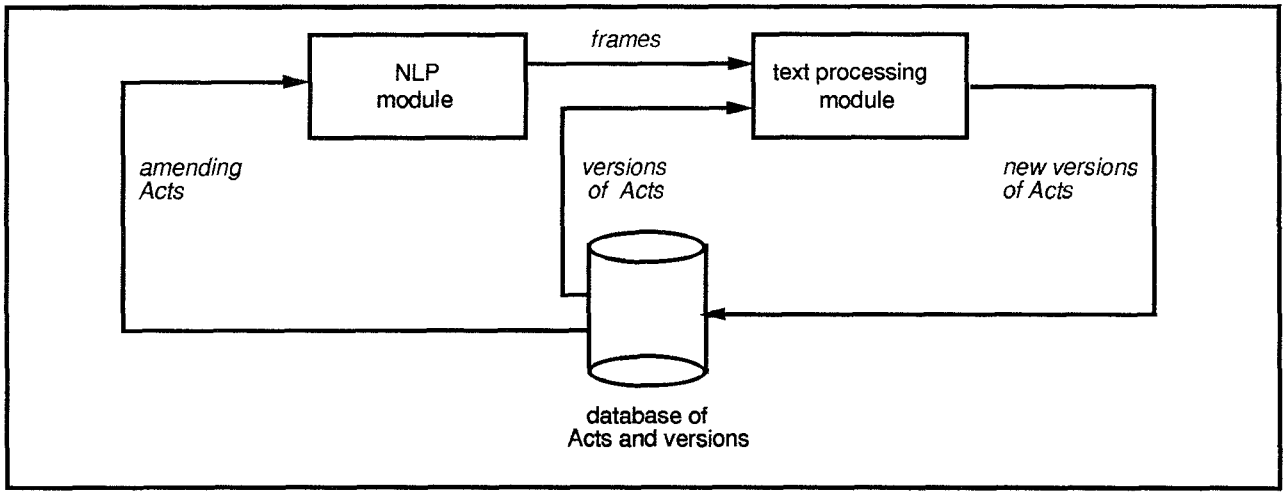

Figure 1: Architecture of the Amendment Processing System

An ATN parser can be easily implemented in the programming language Prolog using the top-down, left-to-right solver and unification that is built into the language [15, 25]. Although it is argued that definite clause grammars (DCGs) are to be preferred over ATNs [15] it is much easier to eliminate the need for backtracking (the trying of alternative paths through the grammar) in ATNs by making use of registers to store intermediate results. Such a deterministic network produces more efficient parsers as time is not wasted trying alternatives which may prove unfruitful.

An ATN comprises a number of states, and a number of arcs between states. Each arc defines the conditions and input required to traverse to a subsequent state and what to output. These it has in common with finite-state transition networks (FSTNs). ATNs also allow the description of named sub-networks. An arc can go to such a sub-network, and after completion of the sub-network, processing continues from the previous state in the main network. These sub-networks can be entered as often as is necessary (i.e. recursively) hence the name, recursive transition network (RTN). Sub-networks allow the description of regularities. An augmented transition network (ATN) is an RTN augmented by the addition of registers in which to hold preliminary results as processing continues. Unlike FSTNs or RTNs, these registers can then later be used as conditions on the arcs facilitating the construction of single pass processing. Registers can also be used to output information in a different order to that in which it appears in the input text. This allows the construction of arbitrary complex structures containing information gleaned from the input text. Thus the amendment action frame can be generated in the parser itself. These characteristics make ATNs ideal for implementing our NLP module. We now describe some of the sub-networks in the NLP module.

4 NLP sub-networks

4.1 Principal Act definitions

The first major problem in implementing the parser is identifying the target element to be modified. Amending sections often refer to an element in 'the Principal Act' which must be defined elsewhere in the Act, either for the whole Act or for a specified element (usually a part or division) within the Amending Act. Such definition sections usually appear at the beginning of the Act or the element to which they refer. So the Act can be processed from beginning to end incorporating these sections as they occur. The main network recognizes a definition of a Principal Act and enters a PRINCIPAL ACT sub-network. This sub-network alters the current definition of "Principal Act" which is stored in a register. Greater error checking is provided by monitoring the scope of each definition and ensuring that a definition is only used in the scope in which it was intended. This is achieved by extending the register to contain a list of Principal Act definitions and their scope.

4.2 Citations

Having identified the Act in which the modification is to occur, the next problem is identification of the element within the Act which is to be modified. Statutes can contain a variety of citation formats [23] however those in amending sections are usually limited to simple citations of a single element, or occasionally a contiguous range of elements which Wilson terms an extended reference (e.g. "sections 1 to 3"). The prototype system handles both the simple and extended references although the later proved somewhat more difficult to implement as Wilson found. Regardless of the citation format, the location information is often split, appearing

in fragments throughout the amending section. Because of the scattering of the information about location, a frame representation was used for storing the target location as was done by Merkl *et al* [11] (and source location for consistency). The frame representation can easily be converted to any citation standard when required. Whenever further clarification of the location is available from the input the LOCATION sub-network is entered. This updates the location frame which is passed to it. It is not necessary to record the short title in the location frame as an Act is uniquely determined by the year and the Act number in that year. If needed the short title can be derived from this.

4.3 Time

The time at which an amendment comes into effect is crucial for ensuring that the correct version of the Principal Act is available for every moment in time. In all Australian jurisdictions where an Act commences on a particular day, it comes into operation immediately on the expiry of the preceding day [14]. Thus the smallest unit of time to be represented is a day. An Act may commence a certain time after it receives the vice-regal assent, on a day fixed by the Act, on the proclamation of the Governor or Governor-General (acting on the advice of his or her Ministers) or at some time fixed by reference to a certain event. Since most authorized versions of Acts have the date of commencement as a footnote to the title page, it seems reasonable to expect this date to appear in the SGML encoding of the amending Act. Unfortunately different elements of the Acts may come into effect at different times. A separate section is used specifying the commencement times of effected elements. While the handling of temporal references can be problematic, the current version of the NLP module includes a TIME sub-network for handling sections which specify commencement times for the Act or different elements of the Act with the default being the commencement date in the encoding of the Act. This sub-network handles most of the more common expressions of commencement times of elements.

It is possible that an amending Act may come into force after a subsequent Act amending the same Principal. Although rare, retrospective legislation is permitted in Australia. Commencement times dependent on certain events occuring can also create an unexpected ordering of amending Acts. The amendments must then be applied to multiple versions of the Principal. The simplest solution is to undo modifications in any subsequent versions, apply the intervening modifications and then reapply the later modifications. More sophisticated management could simply propagate changes to later versions reporting any conflicts between changes. These may actually require judicial interpretation.

4.4 Sections

A single section or sub-section in the Amending Act (the source section) is invariably used for each insertion or repeal of a (target) section, or higher element such as a chapter, part, division or sub-division, in the Principal Act. The normal way of organizing amendments below the target section level is to collect amendments to a single target section in a single source section or sub-section. Therefore most amending sections or subsections begin with "Section (*Target section no*) of (*Target act title*) is amended ...." so we describe this part of the ATN in greater detail. A sample ATN for such sections is provided in Figure 2 (some simplifications have been made for the purposes of presentation although the sub-network is still quite complex).

If more than one modification is made to the target section, the source section or subsection is divided into paragraphs each of which contains a complete modification. This corresponds to the parser reaching state 5 on the ATN in Figure 2. Otherwise the network proceeds directly to state 6. When the modification is processed the network will reach state 901 where an amending action frame is output. If there are paragraphs the network will traverse to state 902 from which it can return to state 5 to process the next paragraph or exit to state 999 at the end of the section or subsection. Sections and subsections without paragraphs and hence only one modification go directly from state 901 to 999.

The information for the remaining features is extracted between state 6 and state 901. There are 6 different types of amendment action possible in an amending section or subsection:

Action type	Description
`omit_string`	Deletes all of the designated *Occurrence*(s) of *Old* string in the *Target* element
`omit_element`	Deletes the *Target* element
`replace_string`	Replaces all of the designated *Occurrence*(s) of *Old* string with *New* string in the *Target* element
`replace_element`	Replaces the *Target* element with *New*, a complete element
`insert_string`	Inserts *New* string after the *After* string or before the *Before* string in the *Target* element
`insert_element`	Inserts *New*, a complete element, after the *After* or before the *Before* location

Actions of the first four types are distinguished by the word "omitting" after state 6. Actions of the latter two

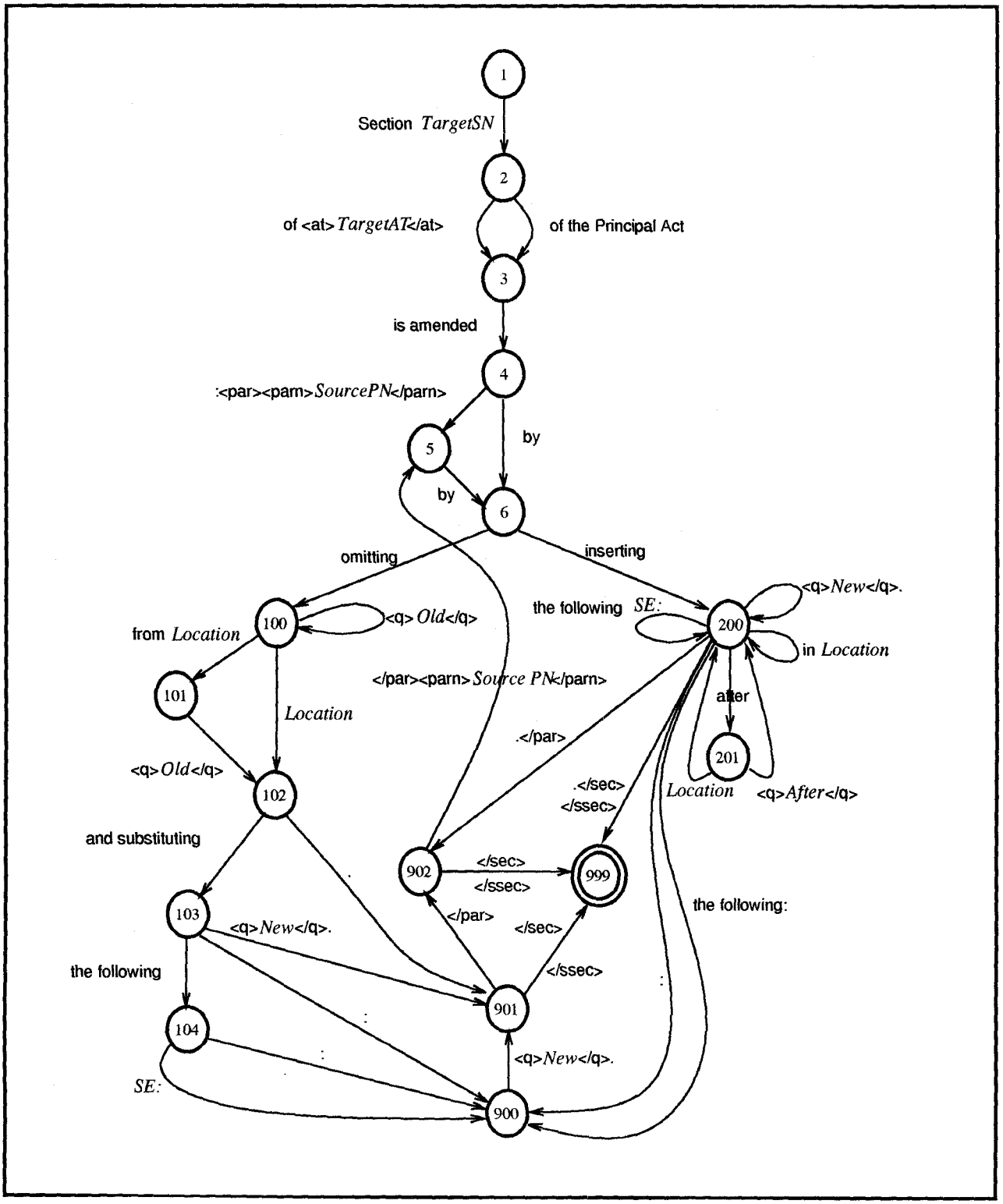

Figure 2: Simplified ATN sub-network for AMENDING SECTIONS

types are distinguished by the word "inserting" after state 6. The remainder of the sub-network identifies the various features using a combination of SGML tags and key words in the text to ultimately distinguish between the various types of amendment action. An example of each type of amendment action appears in its SGML text form and in the resulting Amendment Action frame in Appendix 5.

4.5 Schedules

A collection of minor amendments is often gathered in one or more schedules which appear at the end of the Act. While schedules are strictly speaking part of the Act and have legal force in their own right [14], where such consequential amendments appear in a schedule there is invariably a section or subsection in the body of the Amending Act drawing attention to the amendments in the schedule. Where a single Act is involved the wording is usually "The Principal Act is amended as set out in the Schedule" or similar words referring to a particular Act or a numbered Schedule. The Schedule usually comprises a description of the element to be amended in bold followed by an amendment action similar to those after state 6 in the AMENDING SECTION sub-network (see Figure 2). Alternatively this information is captured in a two column table.

Where more than one Act is involved the wording is usually "The Acts set out in the Schedule are amended as set out in the Schedule" or similar. Schedules of this type usually contain an Act title followed by a list of elements with the amendment action for each element, followed by another Act title and so on.

All of these forms can be handled by the existing parser which produces a separate frame for each amending action as with the amending sections.

5 Conclusion and further work

The NLP processor has been constructed in NU-Prolog [19] and tested on a relatively small set of example Federal Acts (including the Grape Research Levy Act 1986 and the States Grants (Schools Assistance) Act 1989 and the Acts which amend them – a total of 10 Acts) which were selected because they are relatively small and recent Acts that provide a rich variety of amendment types. The NLP module attempts to process the whole Amending Act, recognising typical sections which form part of Amending Acts but require no amending action as well as the ones mentioned above. Sections which cannot be parsed or recognized are flagged for the user to handle (either by extending the NLP module or by performing the database transactions directly). Some small extensions were required to the ATN with the first four amending Acts but subsequent amending Acts were handled with only one exception (which was an amending section with alternative wording to be inserted depending on the time of commencement of another Act). While variations which are likely to appear more than once should be incorporated into the NLP module it was thought that unique and complex amending sections are probably better handled individually anyway.

More complex time sections and schedule arrangements can be found in other Acts of the Commonwealth Parliament so wider testing is needed with a larger sample set. Work is presently under way to encode the Family Law Act 1976, which is a much larger Principal Act with more than 42 Amending Acts to date. A few of these Amending Acts amend more than a third of the sections in the Principal Act. Pilot work is also under way to adapt the NLP module for Tasmanian legislation (Tasmania is one of the Australian States) for possible inclusion in the government's integrated legislation drafting and management system. This system would mainly be used for including prior legislation in the legislation database as the drafters are keen to produce the new versions directly and derive the Amending Acts rather than derive the versions from the Amending Acts.

It was considered that using a DCG formalism might make the NLP module more easily modified [15]. Although it is claimed that parsers in each of these formalisms (ATN and DCG) are easy to extend and modify, prototypes in both formalisms seemed to require considerable programming expertise to modify. It is intended to develop a graphical manipulation tool for constructing appropriate ATNs for subsequent versions of the prototype as ATNs lend themselves to graphical representation more easily than DCGs.

Because of the fact that legislation can be repealed by Principal Acts and not just amending Acts, the repeal of whole Acts has not been considered in this study. Further complications arise from the fact that an Act can be impliedly repealed by a later inconsistent Act [14] or be ruled unconstitutional. State legislation can also be overridden by inconsistent Federal legislation. Some mechanism for including this information would be required of a commercial system although this will be difficult if not impossible to automate as it involves semantic analysis in a much wider domain.

Despite these limitations, this study demonstrates the viability of automating significant portions of the application of amendments to legislation and opens the door for truly intelligent legislative drafting and management environments.

References

[1] L. Allen and C. Saxon. Exploring computer-aided generation of questions for normalizing legal rules. In Walter [21].

[2] T. Arnold-Moore and R. Sacks-Davis. Databases of legislation: The problems of consolidations. Technical Report CITRI TR/94-9, Collaborative Information Technology Research Institute (CITRI), 1994. To appear in the *Law Library Journal.*

[3] D. Berman and C. Hafner. Obstacles to the development of logic-based models of legal reasoning. In Walter [21].

[4] D. G. Bobrow and J. B. Fraser. An augmented state transition network analysis procedure. In *International Joint Conf. on AI (IJCAI)*, pp. 557–567, 1969.

[5] A. Clark and K. Economides. Technics and praxis: Technological innovation and legal practice in modern society. *Yearbook of Law Computers and Technology*, 4:16, 1989.

[6] M. Corbett. Indexing and searching statutory text. *Law Library Journal*, 84:759–67, 1992.

[7] G. Greenleaf, A. Mowbray, and D. Lewis. Teaching lawyers information retrieval: the AIRS training system. In *Information Online'88: Australian Online Information Conference*, Sydney, 1988.

[8] C. D. Hafner. *An Information Retrieval System based on a Computer Model of Legal Knowledge.* UMI Research Press, Ann Arbor, MI, 1981.

[9] International Organization for Standardization. Information processing – text and office systems – Standard Generalised Markup Language (SGML), 1986. ISO/IEC 8879:1986.

[10] P. Johnson and D. Mead. Legislative knowledge base systems for public administration – some practical issues. Technical report, Softlaw Corporation, Canberra, 1991.

[11] W. Merkl, S. Vieweg, and A. Karapetjan. KELP: a hypertext oriented user-interface for an intelligent legal fulltext information retrieval system. In *International Conference on Database and Expert System Applications*, p. 399, Vienna, 1990.

[12] M. Minsky. A framework for representing knowledge. In P. H. Winston and B. K. P. Horn, editors, *The Psychology of Computer Vision*, pp. 211–277. McGraw-Hill, New York, 1975.

[13] CITRI. *Structured Information Manager (SIM) Manual, Vols 1-15.* Melbourne, 1994.

[14] D. C. Pearce. *Statutory Interpretation in Australia.* Butterworths, Sydney, 2nd edition, 1981.

[15] F. C. N. Pereira and D. H. D. Warren. Definite clause grammars for language analysis – a survey of the formalism and a comparison with augmented transition networks. *Artificial Intelligence*, 13:231–278, 1980.

[16] M. Sergot, F. Sadri, R. A. Kowalski, and F. Kriwaczek. The British Nationality Act as a logic program. *Communications of the ACM*, 29:370–85, 1986.

[17] R. L. Stoyles. The unfulfilled promise: Use of computers by and for legislatures. *Computer/Law Journal*, 9:73, 1987.

[18] C. F. H. Tapper. Computers and legislation. *Alabama Law Review*, 23(1):1–42, 1970.

[19] J. A. Thom and J. Z. (Eds). *NU-Prolog Reference Manual.* Australia, 1990. Available as Deparment of Computer Science Technical Report No 86/10 Version 1.5.24.

[20] J. Thorne, P. Bratley, and H. Dewar. The syntactic analysis of English by machine. In D. Michie, editor, *Machine Intelligence 3*, pp. 281–309. Elsevier, New York, 1968.

[21] C. Walter, editor. *Computer Power and Legal Language: The Use of Computational Linguistics, Artificial Intelligence, and Expert Systems in the Law.* Quorum Books, New York, 1988. Papers from the 2nd Annual Conference on Law and Technology, Houston.

[22] E. Wilson. Electronic books: the automatic production of hypertext documents from existing printed sources. In *Fourth Annual Conference of the UW Centre for the New Oxford English Dictionary: Information in Text*, pp. 29–45, Ontario, Canada, 1988.

[23] E. Wilson. Integrated information retrieval of law in a hypertext environment. In *Proceedings of the ACM International Conference on Research and Development in Information Retrieval (SIGIR)*, pp. 663–77, Grenoble, France, 1988.

[24] W. A. Woods. Transition network grammars for natural language analysis. *Communications of the ACM*, 13(10):591–606, 1970.

[25] W. A. Woods. Grammar, augmented transition network. In S. C. Shapiro and D. Eckroth, editors, *Encyclopedia of Artificial Intelligence*, pp. 323–333. John Wiley & Sons, Inc., New York, 1987.

Appendix: Example output of the NLP module

Below is a representation of some output from the NLP module. First we provide a simplified SGML encoding of a fictitious Amending Act derived from various Acts which amend the *States Grants (Schools Assistance) Act* 1989 (Cth) No 1 and then we list the resulting output frames.

```
<act>
<at>States Grants Amendment Act 1990</at><an>67</an>
<commence>1 Jan 1991</commence>
<sec><sn>1</sn>
For the purposes of this Act <deft>Principal Act</deft> means the <at>States Grants
(Schools Assistance) Act 1989</at>.
</sec>
<sec><sn>2</sn>
Section 4 of this Act shall commence on the 30 April 1991 and the remainder of the Act
shall commence on a date to be fixed by proclamation.
</sec>
<sec><sn>3</sn>
Section 4 of the Principal Act is amended by omitting the definition of
<deft>Minister</deft> in subsection (2).
</sec>
<sec><sn>4</sn>
Section 14 of the Principal Act is amended:
<par><parn>a</parn>
by omitting from subsection (1) <q>some uninteresting text</q> and
substituting <q>an alternative</q>.
</par>
<par><parn>b</parn>
by omitting from subsection (3) <q>some redundant text</q>.
</par>
</sec>
<sec><sn>5</sn>
Section 25 of the Principal Act is amended:
<par><parn>b</parn>
by omitting subsection (2) and substituting the following:
<q><ssec><ssn>2</ssn>This is a substitute subsection.</ssec>
</q>.
</par>
<par><parn>b</parn>
by inserting <q>some replacement text</q> in subsection (4) after
<q>some text for location purposes</q>.
</par>
<par><parn>c</parn>
by inserting the following subsection after subsection (5):
<q><ssec><ssn>5A</ssn>This is a newly inserted subsection.</ssec>
</q>.
</par>
</sec>
</act>
```

Which produces the following output frames (those features with a nil entry have been left out):

```
Source: loc(1990, 67, n, n, n, n,  3, n, n, n, n, n, n)
Target: loc(1989,  1, n, n, n, n,  4, 2, "Minister", n, n, n, n)
  Time: 1 Jan 1990
  Type: omit_element
 Occur: 1

Source: loc(1990, 67, n, n, n, n,  4, n, n, a, n, n, n)
Target: loc(1989,  1, n, n, n, n, 14, 1, n, n, n, n, n)
  Time: 30 Apr 1990
  Type: replace_string
   Old: "some uninteresting text"
 Occur: 1
   New: "an alternative"

Source: loc(1990, 67, n, n, n, n,  4, n, n, b, n, n, n)
Target: loc(1989,  1, n, n, n, n, 14, 3, n, n, n, n, n)
  Time: 30 Apr 1990
  Type: omit_string
   Old: "some redundant text"
 Occur: 1

Source: loc(1990, 67, n, n, n, n,  5, n, n, a, n, n, n)
Target: loc(1989,  1, n, n, n, n, 25, 2, n, n, n, n, n)
  Time: 1 Jan 1990
  Type: replace_element
   New: "<ssec><ssn>2</ssn>This is a substitute subsection.</ssec>"

Source: loc(1990, 67, n, n, n, n,  5, n, n, b, n, n, n)
Target: loc(1989,  1, n, n, n, n, 25, 4, n, n, n, n, n)
  Time: 1 Jan 1990
  Type: insert_string
 Occur: 1
   New: "some replacement text"
 After: "some text for location purposes"

Source: loc(1990, 67, n, n, n, n,  5, n, n, c, n, n, n)
  Time: 1 Jan 1990
  Type: insert_element
   New: "<ssec><ssn>5A</ssn>This is a newly inserted subsection.</ssec>"
 After: loc(1989,  1, n, n, n, n, 25, 5, n, n, n, n, n)
```