SECLIPS: A Structured English Interface for an Expert System Shell

Frank Hadlock
Computer Science Department
Tennessee Technological University
Cookeville, TN 38501
Ph. 01-931-372-3687
fhadlock@tntech.edu

ABSTRACT

This paper describes the SECLIPS system, which provides a restricted English grammar and vocabulary designed to express the facts and rules of a rules-based knowledge base. SECLIPS is an extension of ECLIPS and allows the user to extend a core vocabulary to express a knowledge base of facts and rules for a particular problem domain. SECLIPS employs an LL(1) based translator with the grammar for the restricted English SECLIPS language to perform a translation to the CLIPS expert system language. SECLIPS extends ECLIPS by supporting negative predicates and modifiers, plural verbs, and list structures. SECLIPS provides a mechanism by which knowledge can be expressed in a readable format and translated into a CLIPS format with a readily apparent correspondence between the SECLIPS input and the CLIPS output.

1. INTRODUCTION

CLIPS is a forward chaining expert system patterned after OPS5. Developed by NASA in 1985, it is available as public domain software. Since the CLIPS inference engine uses the RETE algorithm for pattern matching and rule firing, CLIPS is very efficient. The main obstacle to the use of CLIPS by nonprogrammers is its LISP-like syntax. Fact templates, fact instances, and rules employ patterns that are delimited by parentheses. Nested constructs with missing or extra parentheses are a common source of error. Controlled English was introduced by Fuchs to enable designers to specify software in a readable representation, which could be translated to Prolog for requirements based testing [2], [4] SECLIPS was formerly named ECLIPS and renamed to avoid confusion with the Eclipse IDE. SECLIPS is an adaptation of Controlled English, designed to provide a user-friendly front end for developing knowledge-based systems. Since expert systems have a wide range of applications, a tool for expressing knowledge in the domain of application should prove very useful. Besides expert systems, rule based knowledge is recognized as being of central importance in business. From an information system perspective, a business rule is "...is a statement that defines or constrains some aspect of the business. It is intended to assert business structure, or to control or influence the behavior of the business." from the Business Rules Group [1]. From the "Business Rules Manifesto" (also [1]), "*Rules should be expressed declaratively for the business audience, in natural-language sentences. If something cannot be expressed, then it is not a rule*."

This emphasizes the significance of SECLIPS since it provides a linkage between an easily understood representation of domain knowledge, and an executable representation (CLIPS). Together they provide a user-friendly basis for building knowledge-based systems.

2. CLIPS vs. SECLIPS

2.1 CLIPS

CLIPS employs fact templates along with initial fact instances of these templates. If course requirements are to be asserted for an academic program, a CLIPs fact template might be "(deftemplate is-requirement (slot course) (slot program))" while a fact instance might be "(deffacts requirement-facts (is-requirement (course CSC202) (program CSC)))". Rules employ patterns in both their antecedents and consequents that are patterned after the templates. If the patterns in an antecedent match facts in the current fact list, new facts are asserted or old facts retracted according to the patterns in the consequent when the rule is fired. An example of a CLIPS rule is "(defrule must-take (is-major (student ?X) (program ?Y)) (is-requirement (course ?Z) (program ?Y)) => (assert (must-take (student ?X) (course ?Z))))" The rule asserts that a student ?X who is a major in program ?Y for which course ?Z is a requirement must take course ?Z. Note that variables are introduced by ?'s. Patterns in the antecedent are compared with a fact list, with variables being bound to fields in the facts. If the antecedent is satisfied, the rule is put in a list of rules ready to fire. If this rule is fired, the fact that student ?X must take course Z? would be added to the fact list. Note that fact templates, fact instances, rules and patterns are delimited by parentheses that make CLIPS difficult to read or write. For a detailed description of CLIPS, see "Expert Systems – Principles and Programs" (3).

2.2 SECLIPS

To make CLIPS more readable, it is natural to combine a predicate adjective name with an intransitive

verb such as "is" or "has" for the fact name for facts concerning a state, along with nouns for the slot names. Examples are "is-major" for " . . Student is a major .. " and "has-fever" for " . . patient has a fever . . ". For event facts, it seems natural to combine the transitive verb with the object for the fact names. The subject and prepositional phrase heads would be used for the slots. An example is "passes-course" for the fact name for " . . passes course . .". Examples of SECLIPS source fact templates, fact instances, and rules are shown below with the corresponding CLIPS target constructs. If these conventions are followed in writing a CLIPS program, then it becomes reasonable to assume a restricted English front end from which CLIPS programs can be generated. SECLIPS is an extension of ECLIPS which was introduced as a restricted English front end. Like Controlled English, SECLIPS contains a core vocabulary and is governed by a restricted syntax.

2.3 CLIPS → SECLIPS Correspondence

The following examples provide an overview of the correspondence between SECLIPS source constructs and CLIPS targets.

Fact Templates. A SECLIPS fact template is formed with a declarative sentence beginning with the key word "Define" and a single clause with a transitive verb from the domain dictionary or intransitive verb "is" or "has" from the core dictionary. The template is limited as to noun or prepositional phrases as discussed later. Templates are central to the translation process and model an event or relation with fixed verb component and variable objects. Slot markers *A*, *B*, . . *M* (first half of alphabet) are used to indicate locations where references to specific domain objects can be substituted. Slot markers must occur in the phrase head position.

Example 1: "Define course A is a requirement for program B." for which the CLIPS target generated by the SECLIPS translator is "(deftemplate is-requirement (slot course) (slot program))".

To make the correspondence between SECLIPS source and CLIPS target readily apparent, and to make the generated CLIPS more readable, the translator uses the verb along with the direct or indirect object as template name. Slot names are derived from the corresponding subject, object and prepositional phrase heads.

Fact Instances. A SECLIPS fact instance is formed by a declarative sentences where the subject or object may be a list. The fact instance must correspond to a previously defined template as far as the verb and noun phrase structure. Noun and prepositional phrase heads are specific domain object references. .

Example 2: "Data Structures is a requirement for program CSC." is a fact instance corresponding to the template of Example 1. The CLIPS target generated by the SECLIPS translator is "(deffacts is-requirement-facts (is_requirement (course Data_Structures) (program CSC)))".

As can be seen from the example, the SECLIPS translator has a fact instance name "is-requirement", using the same rules as were used for fact template names. The corresponding template is located in order to retrieve the slot names.

Rules A_SECLIPS rule is formed by a conditional sentence using the words *if*, *then*, *else*, *assert* and *retract* which are part of the SECLIPS core vocabulary. Clauses in the antecedent and consequent must correspond to fact templates. They are conjoined by *and* and *or* in the antecedent and by *and* in the consequent; these conjunctions are part of the core vocabulary. Variables are indicated by capital letters *N*, . . *Z* (second half of alphabet) Additional core words *assert* and *retract* correspond to CLIPS key words which assert new facts or retract old ones depending on the truth of the antecedent.

Example 3: "If student X is a major in program Y and course Z is a requirement for program Y then assert student X must take course Z. . The CLIPS target generated by the SECLIPS translator is

```
(defrule major-requirement-take
(is-major (student ?X) (program ?Y))
(is-requirement (course ?Z) (program ?Y))
        => (assert (must-take (student ?X) (course ?Z))))".
```

The rule states that if a course is required by a program and a student is pursuing the course of study prescribed by the program, then the student must take the course. The conditional element *(is-requirement (course ?Z) (program ?Y))* is an example of the simplest kind of conditional element, referred to as a pattern, which is designed to match the fields of a fact instance of the required_course template. A ? introduces a variable. In order for the patterns in this rule to match facts in the current fact list, the variables ?X, ?Y and ?Z must be bound to fields in the matching facts. The consequent action asserts a new fact as an instance of the must_take template, with fields given by the values of ?X and ?Z variables. Another type of consequent action element is the retract action.

Example 4: (Example of the use of retract)

"If student X must take course Y and student X passes course Y then retract student X must take course Y". The CLIPS target generated by the SECLIPS translator is

```
(defrule take_passes
    ?mt ← (must_take (student ?X) (course ?Y))
    (pass_course (student ?X) (course ?Y))
        =>
    (retract ?mt )).
```

When fired, the ?mt variable is bound to the address of the fact matching the must_take pattern, causing the fact to be retracted or removed from the current fact list. These are some of the basic features of CLIPS and constitute the target language for the former version of SECLIPS. CLIPS features a number of predicates and functions, as well as features for program control. These are not included in the SECLIPS target and the reader is referred to [3] for a full discussion of CLIPS.

2.4 SECLIPS Source Language

SECLIPS is an adaptation of Structured English and Controlled English to provide a readable representation for a knowledge base or for software specification. Both vocabulary and grammar are designed to support translation to CLIPS but would serve as a basis for other target languages.

SECLIPS Vocabulary: At the language level, SECLIPS is comprised of a core vocabulary, intended to be common to all application domains, business or medical. Like Attempto Control English (ACE), the SECLIPS core vocabulary contains determiners, quantifiers, prepositions, and logical connectives such as *and*, *or*, *not, if* and *then*, words *define*, *assert* and *retract*, along with intransitive verbs *is* and *has* [5]. However, SECLIPS assumes other domain words (verbs, adverbs, nouns, and adjectives) will be added to the dictionary for a particular application. Any word not found in the dictionary is assumed by SECLIPS to be a noun. The core words *if* and *then* are reserved for use in the SECLIPS source sentences which map into CLIPS rules. The core word *define* is used to introduce a fact template sentence. Words *assert* and *retract* are used to introduce actions in the consequent of a rule. The SECLIPS core vocabulary has been extended to include plural intransitive verbs such as *are* and *have*, as well as multifield slot markers.

SECLIPS Grammar: An attributed phrase structure grammar for SECLIPS has evolved from the ECLIPS grammar [4]. The grammar supports the definition of a SECLIPS sentence list comprised of fact template sentences, fact instance sentences, and conditional sentences. These are translated respectively into CLIPS fact templates, fact instances, and rules.

The SECLIPS grammar is LL(1) with a semantic action (italicized) associated with rules for which the left hand side nonterminal corresponds to CLIPS target code not generated automatically during the topdown parse.

The SECLIPS grammar has been extended to support plural subjects and objects by adding a noun phrase list nonterminal <NPList> as follows.

<NPhrase> → <Det><Adj><NHead><NPList>
<NPList>→ , <NPhrase>
<NPList> → c<NPhrase>
<NPList> → λ

Here <Det> generates a determiner or nothing and <Adj> generates a possibly empty string of adjectives, while <NHead> generates a domain noun or slot marker or variable. <NPList> then generates a comma followed by a noun phrase, or an "and" followed by a noun phrase, or nothing.

The SECLIPS grammar has also been extended to enable negative assertions by modifying the verb rules [4] as shown below:

<VPhrase> → <Aux><Neg><VHead>
<Aux> → x
<Aux> → λ
<Neg> → q
<Neg> → λ
<VHead> → v
<VHead> → λ

This enables a verb phrase to consist of an optional auxilliary verb <Aux> followed by a optional negative assertion <Neg> followed by a verb head <VHead>. This construction enables the SECLIPS grammar to retain its LL(1) property while gaining the added functionality. Words such as "is" and "has" and "does" are given "x" as lexical token in the SECLIPS dictionary unless followed by "not" in which case they are given "q" as lexical token. As a consequence, the lexical analyzer must check for two word phrases before looking for single words.

2.5 SECLIPS Translator

The first stage of the translator performs lexical analysis to reduce the SECLIPS source to a token stream, retaining or transforming the original content words for use in a readable CLIPS target representation. The second stage performs a top down LL(1) parse, building a derivation tree for an SECLIPS paragraph. The third stage of the translator processes the derivation tree in bottom up fashion, synthesizing attributes from the initial token text via the associated semantic actions and transforming it into CLIPS representation.

Fact templates are needed to generate fact instances as well as conditional elements in rule antecedents. To ensure that the templates are available when needed during the translation process, the lexical analyzer bin sorts sentences by the first word of the sentence: the key word "define" identifies template sentences while the key word "if" identifies rule sentences. By default remaining sentences are assumed to be fact instance sentences. Template sentences are placed at the end in order that in the bottom-up processing of the derivation tree, they are processed first. During lexical analysis, placeholders A,B,C,D,E,F are discarded by the analyser since the slot name will be formed by the nominal compound preceding the placeholder. Each parse tree node has an associated target CLIPS code attribute. Terminal nodes are assigned the source word as attribute value. Nonterminal node values are assigned values synthesized from the values of

descendant nodes by the semantic action associated with the production used to expand the nonterminal. One of the semantic actions, *Phrase*, discards prepositions and determiners and forms a hyphenated string of adjectives and nouns. Another semantic action, *MakeTemplate*, is described below in terms of an example parse. The SECLIPS source is "Define course A is a requirement for program B." and the CLIPS target is "(deftemplate is_requirement (slot course) (slot program))" The source is parsed as ((define) (course A)$_{Noun\_Phrs}$ (is)$_{Verb\_Phrs}$ (a requirement)$_{Noun\_Phrs}$ (for program B)$_{Prep\_Phrs}$). The semantic action *MakeTemplate* is invoked to build the CLIPS representation of a ‹Sentence› consisting of the "define" key word and followed by ‹Frame›. Templates are saved in a Templates list with each template consisting of a name along with two to three slots depending on the number of clause phrases and verb type. To create a new template, *MakeTemplate* invokes *MakeFrame* which accesses the descendant nodes, ‹Frame› → <Noun_Phrs> <Verb_Phrs> <Noun_Phrs> <Prep_Phrs>. and forms a template name as described, along with the number of slots. The same process is followed in processing a fact instance sentence, or the patterns in a rule sentence to form a corresponding template name. The template list, created from the Deftemplate source sentences, is searched by template name to find a template corresponding to the fact instance. The deffacts construct is then generated from a SECLIPS fact instance sentence and the corresponding template by filling the template slots with the corresponding fact instance constants. Defrule constructs are constructed as follows. Each clause in the antecedent and consequent of a SECLIPS conditional sentence is processed to create a template name. The clause and the template are then processed to generate a pattern element of the defrule construct by filling the template slots with the variables from the sentence antecedent or consequent clause. Processing of the derivation tree is bottom up and a list of template names of facts to be retracted is formed. As a conditional pattern element is formed, the list of template names of facts to be retracted is processed. If the corresponding fact is to be retracted, To illustrate, consider the SECLIPS conditional statement, "If student X must take course Y and student X passes course Y then retract student X must take course Y.".

The must_take and passes_course templates are formed for the antecedent clauses and the must_take for the consequent clause. Since must_take is to be retracted in the consequent, an assignment of a fact address to the variable ?mt is generated. The CLIPS target is:

(defrule satisfy_requirement
?mt ← (must_take (student ?X) (course ?Y))
(pass_course (student ?X) (course ?Y)) =>
(retract ?mt) (passed (student ?X) (course ?Y))

The translator is implemented through procedural attachment. Rules in the grammar have the potential of having a procedure attached that synthesizes the CLIPS output to be associated with the left hand side nonterminal. An example is the *MakeTemplate* procedure attached to the template sentence rule. Translation proceeds in bottom up fashion, since the CLIPS output is assembled partially from the SECLIPS source words associated with terminal nodes of the parse tree. Templates are constructed first (parsed last) and a list maintained. The key to high level translation is:

- process a fact instance, rule pattern element or action
- formulate a template name from the verb (+ predicate adjective if verb is relational "is", "has", etc.)
- retrieve the **unique** template with this name from the template list
- populate the slots of the template to corresponding CLIPS construct.

This points a fundamental assumption (and limitation) that the SECLIPS input is structured so that transitive verbs and intransitive verb/predicate adjective combinations are used so that they correspond to unique events or states.

3. Extensions of SECLIPS

The current version of SECLIPS **does not** include Controlled English core vocabulary or grammatical constructs corresponding to CLIPS predicate, mathematical, and procedural functions which would greatly enhance the usefulness of SECLIPS. An extension to SECLIPS would be to introduce core vocabulary words to support commonly used functions.

The extensions that have been implemented are those to support negative assertions and to allow plural verbs. The negative assertion extension is critical to the use of SECLIPS in developing expert systems. The plural verbs extension greatly enhances SECLIPS as far as compressing the number of assertions needed in the SECLIPS source.

Negative Assertions ('not' predicate)

A rule which asserts which required courses a student must take realistically would include in its antecedent an assertion that the student had not already taken the required course. In a medical domain, diagnostic rules discriminate between different diagnoses by asserting not only that the patient has certain symptoms, but that he not have other symptoms. A CLIPS rule for deciding which courses a student must take is:

Example 5: "If student X is a major in program Y and course Z is a requirement for program Y and student X has not taken course Z then assert student X must take course Z.". The CLIPS target generated by the SECLIPS translator is

(defrule major-requirement-take
(is-major (student ?X) (program ?Y))
(is-requirement (course ?Z) (program ?Y))
(not (taken-course (student ?X) (course ?Z))

=> (assert (must-take (student ?X) (course ?Z))))".

CLIPS operates under the closed world assumption that if there is not a fact in the database, the "not" predicate is evaluated as true. In this case, if there is not a fact that asserts that the student has taken the course, the (not (taken-course (student ?X) (course ?Y))) pattern will be true for that student and course. The SECLIPS source language and grammar has been extended to support the "not" predicate. The SECLIPS source for this rule is "If student A is a major in program B and course C is required for program B and student A has not taken course C then student A must take course C."

Plural Verbs and Object Lists.

In order to compress the expression of "parallel" or similar knowledge, the plural *are* is added to the core vocabulary of SECLIPS as well as "," as a list delimiter. The conjunction *and* now also plays a role as list terminator. CLIPS supports multifield slots which can have zero or more fillers in a slot. They are introduced in a CLIPS fact template using the key word multislot instead of slot as in "(multislot courses) ".

Example 6: "(deftemplate are-requirements (multislot courses) (slot program))". The SECLIPS fact template is "Define the courses A~ are requirements for program B." The multislot markers are *A~,B~, . . M~* . Note the tilda following each multislot marker. A corresponding fact instance is

Example 7: "(deffacts requirement-facts (are-requirements (courses CSC2010 CSC2020 MA1810) (program CSC))). The SECLIPS fact instance is "CSC2010, CSC2020 and MA1810 are requirements for program CSC. To support matching of several fields in a multifield slot, CLIPS uses multifield variables such as $?X. The "$" designates $?X as multifield as opposed to ?X which is single field. To pick an item out of a list, the combination of multifield, single field, multifield is used. Since multifield variables can match zero or more items, the single field variable can be bound to any item/field in the list/multifield.. This construction is used in the following rule:

Example 8: (defrule major-requirement-take
(is-major (student ?X) (program ?Y))
(are-requirements (courses $R1 ?Z $R2) (program ?Y))
(not (taken-course (student ?X) (course ?Z))
=> (assert (must-take (student ?X) (course ?Z))))".

The SECLIPS source rule is

"If student X is a major in program Y and course Z is a requirement for program Y and student X has not taken course Z then assert student X must take course Z" If the SECLIPS translator does not find an "is-requirement" template, it looks for an "are-requirements" template in order to generate a pattern.

4. Conclusions

Because English (or any natural language) is ambiguous, and has an extensive vocabulary, various efforts have been initiated to restrict or control English, including Attempto Controlled English [2], Simple English Wikipedia [7], Common Logic Controlled English [6], ECLIPS [4] and SECLIPS. SECLIPS and its predecessor ECLIPS provide a user friendly, natural language like interface for facts and rules for an expert system implemented in CLIPS which, while LISP like in it's syntax, is an efficient and publicly available expert system shell.

References

1. Business Rules Group**,** http://www.businessrulesgroup.org/brghome.htm

2. Attempto Controlled English (ACE), designed by Norbert Fuchs - Department of Computer Science, University of Zurich

3. Expert Systems – Principles and Programming, Giarratano and Riley, PWS, 1998

4. ECLIPS, Hadlock, Proceedings of 41st SEACM, 2003

5. Sowa,http://users.bestweb.net/~sowa/misc/ace.htm

6. Sowa, Common Logic Controlled English, http://www.jfsowa.com/logic/clce.htm

7. Simple English Wikipedia, http://simple.wikipedia.org/wiki.cgi?HomePage