Translating a Subset of English to Defeasible Logic Programs

Cesar V. Dragunsky   Alejandro J. García

Computer Science Department
Universidad Nacional del Sur
Av. Alem 1253, (8000) Bahía Blanca, Argentina
cvd@cs.uns.edu.ar   ajg@cs.uns.edu.ar

Abstract

Defeasible Logic Programming (DeLP) is an extension of Logic Programming capturing common-sense reasoning features. The DeLP language can manage defeasible reasoning, allowing the representation of defeasible and non-defeasible knowledge. Since DeLP syntax is based on a PROLOG-like notation, the task of writing a DeLP program is similar to that of writing a Prolog one. Hence, writing a DeLP program could be hard for someone unfamiliar with this kind of syntax.

In this paper we present a window-based system that assists a user in the writing of DeLP programs. The system allows the user to write sentences in restricted English and then translates these sentences to DeLP syntax. The system is strongly based on a natural language interpretation module for translating a subset of English to DeLP. The translator has the capability of learning new words directly from the sentences being parsed. Thus, the system lexicon is built dynamically while the program is being written. This adaptive feature makes the system usable in any application domain.

KEYWORDS: ARTIFICIAL INTELLIGENCE, NATURAL LANGUAGE UNDERSTANDING

1 Introduction

In this paper we present the system IDLE (Interface for Defasible Logic in English). IDLE is a window-based system that assists a user in the writing of DeLP programs. The system allows the user to write sentences in restricted English and then translates these sentences to DeLP syntax. IDLE is strongly based on a natural language interpretation module for translating a subset of English to DeLP.

Defeasible Logic Programming is an extension of Logic Programming capturing common-sense reasoning features. The DeLP language can manage defeasible reasoning, allowing the representation of defeasible and non-defeasible knowledge. Since DeLP syntax is based on a

PROLOG-like notation, the task of writing a DeLP program is similar to that of writing a Prolog one. Hence, writing a DeLP program could be hard for someone unfamiliar with this kind of syntax.

The IDLE system has the capability of learning new words directly from the sentences being parsed. Thus, the system lexicon is built dynamically while the program is being written. This adaptive feature makes the system usable in any application domain.

This paper is organized as follows: section 2 gives a brief introduction to DeLP, focusing in its syntax. Section 3 introduces the basic English statements that will be used for the translation to DeLP rules. In section 4 the grammar for parsing the mentioned statements is given. Later in section 5 we explain how IDLE will build its own lexicon. Section 6 describes the implementation of the system, and in section 7 include the drawn conclusions.

2 Defeasible Logic Programming Syntax

We include here a brief description of the syntax of DeLP, and refer the interested reader to [4, 5] for details. The DeLP language is defined in terms of two disjoint sets of rules: a set of *strict rules* for representing strict (sound) knowledge, and a set of *defeasible rules* for representing tentative information. Formally,

Definition 2.1 (Strict Rule) *A Strict Rule is an ordered pair, conveniently denoted Head $\leftarrow$ Body, whose first member, Head, is a literal, and whose second member, Body, is a finite set of literals. A literal "L" is an atom "A" or a negated atom "$\sim$A", where "$\sim$" represents the* strong negation. *A strict rule with the head L_0 and body $\{L_1, \ldots, L_n\}$ can also be written as: $L_0 \leftarrow L_1, \ldots, L_n$. As usual, if the body is empty, then a strict rule becomes "$L \leftarrow true$" (or simply "L") and it is called a* fact.

The syntax of strict rules correspond to *basic rules* [7] in Logic Programming, but we call them 'strict' to strengthen the difference with the 'defeasible' ones (see below). Some examples of strict rules follow. Observe that strong negation may be used in the head of the rules.

$$\begin{array}{rcl} bird(X) & \leftarrow & duck(X) \\ \sim innocent & \leftarrow & guilty \\ \sim duck(X) & \leftarrow & \sim bird(X) \\ dead(X) & \leftarrow & \sim alive(X) \end{array}$$

In DeLP *Defeasible Rules* add a new representational capability for expressing a weaker link between the head and the body in a rule. A defeasible rule "*Head* $\prec$ *Body*" is understood as expressing that "*reasons to believe in the antecedent Body provide reasons to believe in the consequent Head*" [8].

Definition 2.2 (Defeasible Rule) *A Defeasible Rule is an ordered pair, conveniently denoted Head $\prec$ Body, whose first member, Head, is a literal, and whose second member, Body, is a finite set of literals. A defeasible rule with head L_0 and body $\{L_1, \ldots, L_n\}$ can also be written as: $L_0 \prec L_1, \ldots, L_n$. If the body is empty, we write "$L \prec true$" and we call it a* presumption.

Syntactically, the symbol "$\prec$" is all that distinguishes a defeasible rule from a strict one. Pragmatically, a defeasible rule is used to represent defeasible knowledge, i. e., tentative information that may be used if nothing could be posed against it. Thus, whereas a strict rule is used to represent non-defeasible information such as " $bird(X) \leftarrow penguin(X)$ ", which expresses that *"all penguins are birds."*, a defeasible rule is used to represent defeasible knowledge such as " $flies(X) \prec bird(X)$", which expresses that *"birds are presumed to fly"* or *"usually, a bird can fly."* A DeLP program is a finite set of defeasible and strict rules (see Example 2.1).

Example 2.1 Here follows a DeLP program:

$$\begin{array}{l}
dog(spike) \\
bull(joe) \\
{\sim}carnivore(X) \leftarrow bull(X) \\
has\_horns(X) \leftarrow bull(X) \\
{\sim}dangerous(X) \prec {\sim}carnivore(X) \\
dangerous(X) \prec {\sim}carnivore(X), has\_horns(X) \\
dangerous(X) \prec not{\sim}dangerous(X)
\end{array}$$

In DeLP, answers to queries will be supported by arguments. Informally, an argument is a derivation that uses strict and defeasible rules, and satisfies certain properties. In DeLP, an argument may be *defeated* by other arguments. A query q will succeed if the supporting argument for it is not defeated; it then becomes a *warranted* conclusion. Since the goal of this paper is to define a translation of English sentences to DeLP syntax, the details of the inference mechanism of DeLP is clearly out of the scope of this paper. We refer the interested reader to [5, 4] for details.

Here follows another example of a DeLP program used in a different application domain.

Example 2.2

$$\begin{array}{l}
sell\_stock(T) \prec advice(T, sell) \\
{\sim}sell\_stock(T) \prec advice(T, sell), speculate(T) \\
speculate(T) \prec negative\_profit(T) \\
{\sim}speculate(T) \prec negative\_profit(T), too\_risky(T) \\
too\_risky(T) \prec market(down) \\
too\_risky(T) \prec breaking(T) \\
negative\_profit(T) \leftarrow pricepaid(T, P), lastsale(T, L), L < P \\
pricepaid(msft, 101)
\end{array}$$

3 Characterization of English Sentences needed for constructing DeLP rules

According to the definitions presented in the previous section, a rule has the form "$L_0 \leftarrow L_1, L_2, \ldots, L_n$" or "$L_0 \prec L_1, L_2, \ldots, L_n$", where each L_i is a predicate. In other words, predicates are the atomic elements for building DeLP rules. In order to introduce how to translate restricted English to DeLP syntax, we will first show how simple statements can be translated to DeLP predicates. Then, facts, presumptions and rules will be obtained generalizing these ideas.

To understand how the categories for the underlying grammar were identified, observe that a predicate like $dog(spike)$ intends to mean the sentences "spike is a dog", and a predicate like $eats(john, grapes)$ represents "john eats grapes". Similarly, we have identified seven classes of sentences that can be translated to DeLP predicates. These classes are shown in the first column of Table 1. In each row we also include an example in English and the corresponding DeLP predicate. We include below a brief explanation of each statement class.

Statement Type	English example	translated DeLP predicate
ISA Statement	John is a parrot	$parrot(john)$
Adjectival Statement	John is green	$green(john)$
Intr. Verbal Statement	John sleeps	$sleeps(john)$
Trans. Verbal St. w/Subject	John eats grapes	$eats(john, grapes)$
Trans. Verbal St. w/o Subject	Shut the door	$shut(door)$
Composite Statement	John is a green parrot	$green(john)$ $parrot(john)$
Prepositional Statement	John is the father of Mary	$father(john, mary)$

Table 1: Representative statements and their translations to DeLP predicates

ISA Statements

These statements have the form "*X is a C*", stating that an individual *X* belongs to class *C*, just like in "*John is a Parrot*". Statements of this kind are important and very frequent.

Adjective Statements

These are of the form "*X is Q*", and state that individual *X* has the quality *Q*. Phrases like "*John is green*" belong in this kind.

Intransitive Verbal Statements

These statements are of the form "*Noun verb*", and they express that the action *verb* is performed by the individual *Noun*. Here, 'Intransitive' means that no third party is involved in the action. An example of this class is "*John sleeps*".

Transitive Verbal Statements with Subject

These are the typical simple sentences in English. They have the form "*Subject verb Noun*" and express that *Subject* exerts action *verb* affecting individual *Noun*. In the example, the sentence "*John eats grapes*", shows *John* performing an action — *eating* — on the *grapes*.

Transitive Verbal Statements without Subject

This kind of sentences are used for expressing either an imperative statement (as in "*shut the door*") or an action to be executed by the agent that interprets the program, like in "$sell\_stock(T) \prec advice(T, sell), speculate(T)$" in Example 2.2.

Composite Statements

These sentences are a mixture of adjectival and ISA statements, that simplify data input. For example, instead of writing the following four statements:

> *John is a parrot.*
>
> *John is green.*
>
> *John is lazy.*
>
> *John is naughty.*

it is possible to write directly *John is a green, lazy, naughty parrot.*

Prepositional Statements

These statements assert that a given relation holds among certain individuals, like in "*John is the father of Mary*". Another example is "*the last sale of T was P*", represented as "$lastsale(T, P)$" in Example 2.2. This class of statements are fundamental because they introduce multi-ary predicates.

4 A Grammar for IDLE

Automatic translation requires a parser, which in turn must be defined based on a grammar. In this section we will first introduce a grammar for IDLE, and then we will explain how to use it for parsing English sentences that will be translated into DeLP rules.

4.1 The Grammar

Here follows a context free grammar obtained from the statement classification described in the previous section. Observe that each production of the grammar is commented on same line.

Stat	$\rightarrow$ (ISAS|AdjS|IVS|TVSwS|TVSwoS|CS|PS)	{Statement}
ISAS	$\rightarrow$CW is a ISAP	{ISA Statement}
AdjS	$\rightarrow$CW is AdjP	{Adjectival Statement}
IVS	$\rightarrow$CW IV	{Intransitive Verbal Statement}
TVSwS	$\rightarrow$CW TV (N$*$ CW)$^+$	{Transitive Verbal Statement with Subject}
TVSwoS	$\rightarrow$TV (N$*$ CW)$^+$	{Transitive Verbal Statement without Subject
CS	$\rightarrow$CW is a AdjP$^+$ ISAP	{Composite Statement}

PS	$\rightarrow$CW is [the|a] PP (Prep CW)$^+$	{Prepositional Statement}
ISAP	$\rightarrow$CW	{ISA Predicate}
AdjP	$\rightarrow$SW	{Adjectival Predicate}
IV	$\rightarrow$SW	{Intransitive Verb}
TV	$\rightarrow$SW	{Transitive Verb}
N	$\rightarrow$SW	{Noise}
PP	$\rightarrow$SW	{Prepositional Predicate }
Prep	$\rightarrow$about|above|after|against|as|around| at|before|below|from|for|in|into|on|onto| since|to|under|upon$\cdots$	{Preposition}
CW	$\rightarrow$[A-Z]([a-z]|[A-Z])*	{Word starting with capital Letter}
SW	$\rightarrow$[a-z]([a-z]|[A-Z])*	{Word starting with small letter}

The richness provided by this system does not lie in its language's syntax, but rather in its lexicon. The lexicon is left out of the specification on purpose. It grows as the system gets acquainted with the language through usage by means of interaction with the end-user, as we will see in section 5.

Example 4.1 To illustrate how a simple sentence is parsed by this grammar, consider for example the Composite Statement "*John is a green Parrot*". For the sake of simplicity, we will assume in this example that the lexicon of the system has an entry for the nouns *John* and *Parrot*, and for the adjetive *green*. In section 5 we will show how the lexicon will be updated dynamically.

While parsing, the system first identifies a word starting with a capital letter, and then tries to use any of the productions starting with 'CW'. The word "*is*" after "*John* " suggest that one of the following productions should be used: ISAS, AdjS, CS, or PS. However, the next word "*a*", allows the system to leave the AdjS hypothesis out.

Since "*green*" is known to be an adjective, this sentence must have the form of a CS, or the parser will reject it. The parser will then find the noun 'Parrot', and the sentence is correctly parsed as a Composite Statement.

4.2 Combining Grammatical Statements to Form English Sentences

We have shown above how to generate a predicate from a statement. Generating a fact or a presumption is simple if the type of knowledge being represented (strict or defeasible) is given beforehand. The same will happen for distinguishing strict and defeasible rules. In section 6 we will show how the user will specify the type of knowledge of the sentence. But first we must explain how the system identifies a rule.

Writing a rule involves specifying its head and body. The head is a statement so the system knows how to translate it to a predicate. The head will be separated from the body by the keyword 'if', and the body is a list of statements separated from each other by the keyword 'and'. Each of these statements will be processed as explain before. According to this, the grammar introduced above will be extended with the following production:

$$\text{Rule} \rightarrow \text{Stat} \mid \text{Stat if Stat (and Stat)}^*.$$

Let us illustrate this ideas with some examples:

Example 4.2 Suppose that a user writes the sentence *John is a Parrot* and specifies that the type of knowledge is "strict", then the system generates the fact: $parrot(john)$.

However, if a user writes the sentence *John is a Parrot* but specifies that the type of knowledge is "defeasible", then the system generates the presumption: $parrot(john) \prec true$.

In the case that a user writes the sentence *John is a bird if John is a parrot* and specifies that the type of knowledge is "strict", then the system is able to generate the strict rule: $bird(john) \leftarrow parrot(john)$.

When a user writes the sentence *John is a Parrot if John is green and John talks* and specifies that the type of knowledge is "defeasible", then the system has enough information for generating the defeasible rule: $parrot(john) \prec green(john), talks(john)$.

5 Learning in IDLE: an Adaptive Lexicon

The meaning of a word usually depends on the context. Hence, it is not possible to build a general-purpose lexicon for a natural language interpretation system: the word database is inevitably restricted by the application domain. As this system is not tailor-made for any particular application, it is difficult to provide in advance a complete lexicon for it.

In IDLE the lexicon will be built dynamically by interacting with the user every time an unknown word is found. However, the system has a very reduced built-in lexicon with enough entries for identifying the different classes of sentences, for example words like *"is"*, *"a"*, *"the"*, *"if"*, etc.

For updating the lexicon, the interpreter has the capability of learning new words by interacting with the user at the time it encounters a word it does not know. When the system finds an unknown word, the user is prompted for information about it, according to what the parser thinks the word might be. All possible categories for the word (noun, adjective, verb) will be guessed by the place in the sentence that the word occupies. The system will first ask whether its guess was correct, and the user may reject or accept the system proposal. In case of accepting, other information will be required depending on the word category. The other categories will be attempted as the user rejects system proposals.

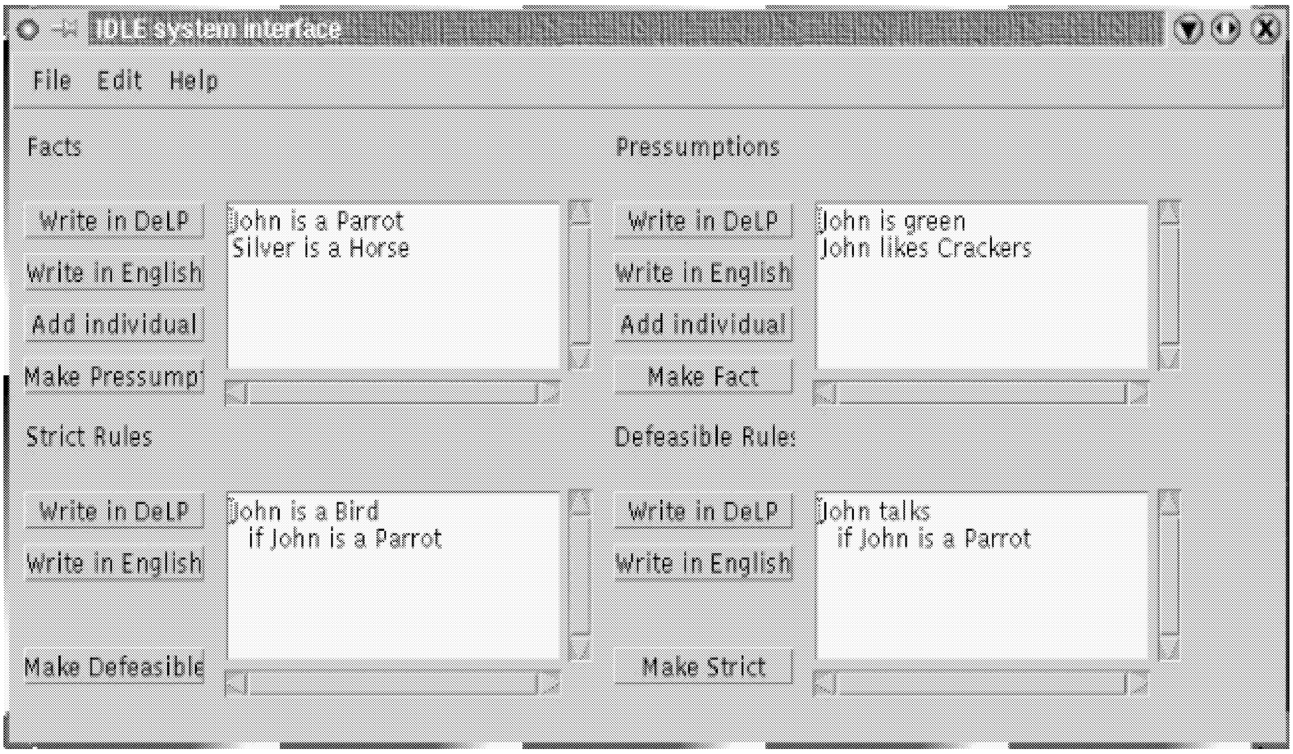
Example 5.1 To illustrate the lexicon updating process, suppose that the system is parsing the sentence "*John is a green Parrot*", and the word "*green*" is not in the lexicon. When the system finds the word "*green*" after having parsed up to "*John is a*", it will use the grammar for inferring the possible categories for the unknown word. If the sentence is parsable at all, then "*green*" must be either a prepositional predicate (using the PS production) or an adjectival one (using the CS production). First the system will prompt the user whether it can assume that "*green*" is a prepositional predicate. If the user says 'yes', the sentence will not parse because the remainder of it does not have the form of a Prepositional Statement (PS). But if the user says 'no', a new proposal will be made. The grammar dictates that this word could also be an Adjectival Predicate (AdjP), so the user will be prompted as to whether this is the correct category for this word. If the answer is 'no', the sentence will be rejected because there are no more productions to try. However, if 'yes' is answered, the new word "*green*" will be incorporated to the lexicon as an adjective, and the sentence will be parsed successfully.

6 The IDLE System

In a DeLP program two different kinds of knowledge may be represented: strict or defeasible. In turn, for each kind, the programmer may write rules or unconditional statements. As stated before, generating a fact or a presumption is simple if the kind of knowledge being represented is given beforehand. The same holds for distinguishing strict and defeasible rules. In this section we will show how the user will write a DeLP program and how s/he will specify the type of knowledge of a sentence.

6.1 An Interface for Knowledge Acquisition

For writing a DeLP program the user will have several options: s/he may write rules directly in DeLP syntax, write sentences in English, use the "Add individual" feature explained below, or transform strict knowledge to defeasible and vice versa. The distinction among facts, presumptions, strict rules and defeasible rules, will be accomplished by using the graphical interface.



John is a Parrot
Silver is a Horse

John is green
John likes Crackers

John is a Bird
  if John is a Parrot

John talks
  if John is a Parrot

Figure 1: IDLE Systems's Main Window

The system interface will provide menus for all the basic file management and text edition tasks (see Figure 6.1). The main window consists of four areas, one for each kind of program clause. Each area provides the following options:

"Write in DLP". This option enables a skilled DLP programmer to write clauses directly in DLP syntax.

"Write in English" This is the option that constitutes IDLE's raison d'etre. It allows the user to write the English phrases that will be translated into DLP by the IDLE system.

"Add individual" (This option is only valid for facts and presumptions) Observe that 'Adjectival' and 'ISA' statements can be seen as defining classes. Adding an individual to a so defined class using the "write in English" option involves specifying a class c and the individual i, that will be translated to the fact $c(i)$ or presumption $c(i) \prec \mathit{true}$. Using this feature the system will offer an alphabetically sorted list of classes to select from, and the user will input an individual to add to the selected class.

"Make ...". Usually knowledge about the real world changes, and very often this change affects only the type of rule. Thus, a strict rule may become a defeasible one, or vice versa, and similarly, a fact may become a presumption, or vice versa. This option enables these transformations to be effected by a few simple mouse clicks.

6.2 Rule Construction Assistance

Writing facts or presumptions is not too error-prone. However, writing complex rules usually is. For example, one of the most common sources of error in Prolog programs is misspelling. Misspelling can be perceived by an automatic system by detecting singleton (unbound) variables, or atom or predicate names appearing for the first time in a rule.

A well-known warning policy built into many plain Prolog interpreters in existence today consists of issuing a warning message for every occurrence encountered of a variable appearing a single time within the scope of the rule that contains it, along with the name(s) of the offending variable(s). This policy will be implemented in IDLE.

It is widely known that Prolog — as well as DLP — does not allow for the declaration of symbols: they are just interpreted as they are encountered, and all binding of equally named symbols happens through unification. In IDLE we will keep track of symbols that have already been used in previous rules. When a new symbol appears in a statement that is being input, the system will look it up in its tables, remaining silent if it shows up, but warning the user and prompting whether s/he wishes to declare the symbol as new in case it does not. This may make the system a bit too talkative while the first statements are being input, but it will tend to stay more and more quiet as it gets acquainted with the program lexicon. Anyway, a symbol declaration feature will be available, for declaring symbols without writing a single rule, and the warning and prompting feature will have a toggle command to turn it off altogether while the first statements are being written if so desired. When turned on, it will check the so-far-written program and automatically declare the symbols it finds there as preexistent.

7 Conclusions

In this paper we have presented IDLE, a window-based system that assists a user in the writing of DeLP programs. The system allows the user to write sentences in restricted English and then translates these sentences to DeLP syntax. IDLE is strongly based on a natural language interpretation module for translating a subset of English to DeLP. The IDLE system has the capability of learning new words directly from the sentences being parsed. Thus, the system lexicon is built dynamically while the program is being written. This adaptive feature makes the system usable in any application domain. A graphical interface will offer the user several options for writing a DeLP program. Rules may be written directly in DeLP syntax, in English, by using the "Add individual" feature, or by making strict knowledge defeasible and vice versa. Therefore, by using this system, the range of potential users for DeLP is expected to grow substantially. The system will also allow people without solid logic programming background the possibility of using by themselves a very powerful, but rather complex reasoning system.

References

[1] James Allen. *Natural Language Understanding.* Benjamin Cummings, 1987.

[2] Veronica Dahl. Translating spanish into logic through logic. *American Journal of Computational Linguistics*, 13:149–164, 1981.

[3] Veronica Dahl. Natural language processing and logic programming. *Journal of Logic Programming*, 12:1–80, 1994.

[4] Alejandro J. García. Defeasible logic programming: Definition and implementation. Master's thesis, Dep. de Ciencias de la Computación, Universidad Nacional del Sur, Bahía Blanca, Argentina, July 1997.

[5] Alejandro J. García, Guillermo R. Simari, and Carlos I. Chesñevar. An argumentative framework for reasoning with inconsistent and incomplete information. In *Workshop on Practical Reasoning and Rationality.* 13th biennial European Conference on Artificial Intelligence (ECAI-98), August 1998.

[6] Gerald Gazdar and Chris Mellish. *Natural Language Processing in Prolog: An Introduction to Computational Linguistics.* Addison-Wesley, 1989.

[7] Vladimir Lifschitz. Foundations of logic programs. In G. Brewka, editor, *Principles of Knowledge Representation*, pages 69–128. CSLI Pub., 1996.

[8] Guillermo R. Simari and Ronald P. Loui. A Mathematical Treatment of Defeasible Reasoning and its Implementation. *Artificial Intelligence*, 53:125–157, 1992.