# Rule Specification Language for Domain Experts

Stanford Logic proposes to develop a Rule Specification Language that can be directly used by the domain experts (e.g., income tax analysts, biologists, etc.), and can also provide a basis for reasoning, calculations, user interface dialogs, and other compliance related tasks. We will test and evaluate this language in two application domains by building sample applications. These domains are: payroll taxes and tax-motivated personal finance advice. We begin by explaining the features of our proposed language and then discuss its reasoning capabilities.

# Features of the Rule Specification Language

The semantics of our proposed language will be defined in terms of logic programming, but it will support a higher-level of abstraction than a typical logical program. Three examples of such higher level of abstraction are as follows:

1. Minimize the need to use variables. For example, our language will allow the direct authoring of mathematical expressions such as: a1 = (5000 / 52 \* (code - 1) div 500). In a traditional logic programming, the right hand side of this equation needs to be converted to: ratio(5000,52,X1) minus(code, 1, X2) & times(X1,X2,X3) & div(X3,500,X4).
2. Support if-then-else rules. For example:

IF L[n] is less than M

I[n] = L[n] - L[n-1]

ELSE

I[n] = M \* (pn-pbik)

In a traditional logic program, the above knowledge has to be expressed using two separate rules:

value(I[n],X) :-

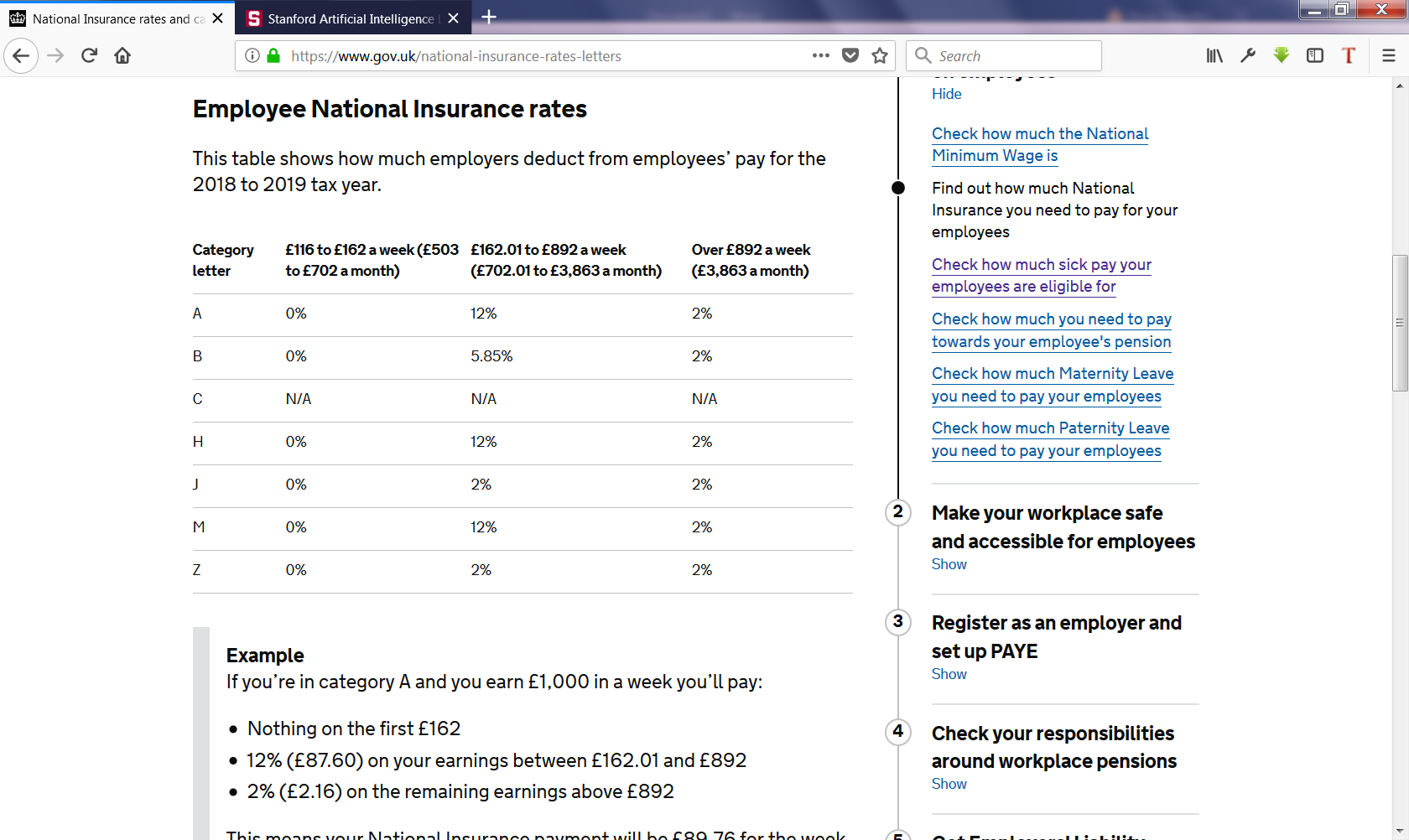
value(L[n],X1) & value(M,X2) & less\_than(X1,X2) & value(L[n-1],X3) & minus(X1,X3,X)

value(I[n],X) :-

value(L[n],X1) & value(M,X2) & ~less\_than(X1,X2) & value(pn,X3) & value(pbik,X4) & minus(X3,X4,X5) & times (X5,M,X)

The complexity of the required rules in a logic program increases significantly as the if-then-else structure becomes more complex, and therefore, a direct support for such knowledge in the Rule Specification Language will make it much easier for the domain experts to author their knowledge.

1. Sometimes the rules that are expressed in the form of tables in a source document. For example, consider the example shown on the next page. Each cell in this table corresponds to a rule in which the body is a conjunction of the logical sentence representing the column heading and a literal corresponding to the row label. In a rule specification language, a domain expert will be able to directly specify such a table without any need to write individual rules for each table entry.



1. Rule Inheritance. Rule specification will provide constructs to make use of an ontology. This will be useful for situations when a rule applies to different kinds of objects that can be organized in a class hierarchy. A facility to use an ontology while writing rules can allow more compact set of rules by stating them at the level of the most general class they are applicable to.

# Reasoning with Rule Specification Language

We will develop several tools based on the rule specification language. Some initial ideas are as follows.

1. User Interface / Dialog Generation. Given a set of rules, we will develop an algorithm that can compute a dialog to obtain the desired inputs from the user and present the result of the computation. For example, in the first rule of the previous section, we know that the only required input is *code,* and once its value is known, a1 can be computed. For the second rule, initially required inputs are L[n] and M, and depending on their relative values, we either need to know L[n-1] or pn and pbik. With a complex set of rules, the reasoning required to produce such an interview dialog can be complex.
2. Advice and What-If analysis: We will analyze several use cases of the reasoning required for financial advice domain and identify most prominent situations that can benefit from logical reasoning. Such use cases may leverage the ontology relationships in the domain, and/or may leverage constraints that are not straightforward to capture in a traditional spreadsheet.
3. Rule Authoring Support: The rule specification will be executed by translating it into a lower level programming language such as Epilog. We will develop a translator/compiler to accomplish this goal. To support the domain expert in authoring rules in the specification language, we will provide an integrated development environment (IDE), through which as they author rules, they are provided productivity tools, e.g., predicate suggestion and completion, results of rule application, and debugging support.