Predicate logic - The need for a richer language

we developed propositional logic by examining it from three different angles: its proof theory (the natural deduction calculus), its syntax (the tree-like nature of formulas) and its semantics (what these for-mulas actually mean). From the outset, this enterprise was guided by the study of declarative sentences, statements about the world which can, for every valuation or model, be given a truth value.

We begin this second chapter by pointing out the limitations of propo-sitional logic with respect to encoding declarative sentences. Propositional logic dealt quite satisfactorily with sentence components like not, and, or and if ... then, but the logical aspects of natural and artificial languages are much richer than that. What can we do with modifiers like there exists..., all ..., among . . . and only ... ? Here, propositional logic shows clear limitations and the desire to express more subtle declarative sentences led to the design of predicate logic, which is also called first-order logic.

Let us consider the declarative sentence

Every student is younger than some instructor.

In propositional logic, we could identify this assertion with a propositional atom p. However, that fails to reflect the finer logical structure of this sen-tence. What is this statement about? Well, it is about being a student, being an instructor and being younger than somebody else. These are all proper-ties of some sort, so we would like to have a mechanism for expressing them together with their logical relationships and dependences.

We now use predicates for that purpose. For example, we could write S(andy) to denote that Andy is a student and I(paul) to say that Paul is an instructor. Likewise, Y (andy, paul) could mean that Andy is younger than Paul. The symbols S, I and Y are called predicates. Of course, we have to be clear about their meaning. The predicate Y could have meant that the second person is younger than the first one, so we need to specify exactly what these symbols refer to.

Having such predicates at our disposal, we still need to formalise those parts of the sentence above which speak of every and some. Obviously, this sentence refers to the individuals that make up some academic community (left implicit by the sentence), like Kansas State University or the University of Birmingham, and it says that for each student among them there is an instructor among them such that the student is younger than the instructor.

These predicates are not yet enough to allow us to express the sentence in (2.1). We don't really want to write down all instances of S(·) where · is replaced by every student's name in turn. Similarly, when trying to codify a sentence having to do with the execution of a program, it would be rather laborious to have to write down every state of the computer. Therefore, we employ the concept of a variable. Variables are written u, v, w, x, y, z,... or x1, y3, u5,... and can be thought of as place holders for concrete values (like a student, or a program state). Using variables, we can now specify the meanings of S, I and Y more formally:

S(x) : x is a student

I(x) : x is an instructor

Y (x, y) : x is younger than y.

Note that the names of the variables are not important, provided that we use them consistently. We can state the intended meaning of I by writing

I(y) : y is an instructor

or, equivalently, by writing

I(z) : z is an instructor.

Variables are mere place holders for objects. The availability of variables is still not sufficient for capturing the essence of the example sentence above. We need to convey the meaning of 'Every student x is younger than some instructor y.' This is where we need to introduce quantifiers ∀ (read: 'for all') and ∃ (read: 'there exists' or 'for some') which always come attached to a variable, as in ∀x ('for all x') or in ∃z ('there exists z', or 'there is some z'). Now we can write the example sentence in an entirely symbolic way as

$\forall x\,(S(x) \rightarrow (\exists y\,(I(y) \wedge Y(x, y))))$

Actually, this encoding is rather a paraphrase of the original sentence. In our example, the re-translation results in For every x, if x is a student, then there is some y which is an instructor such that x is younger than y.

Different predicates can have a different number of arguments. The predi-cates S and I have just one (they are called unary predicates), but predicate Y requires two arguments (it is called a binary predicate). Predicates with any finite number of arguments are possible in predicate logic.

Another example is the sentence Not all birds can fly.

For that we choose the predicates B and F which have one argument ex-pressing

B(x) : x is a bird

F(x) : x can fly.

The sentence 'Not all birds can fly' can now be coded as

$\neg(\forall x\,(B(x) \rightarrow F(x)))$

saying: ‘It is not the case that all things which are birds can fly.’ Alterna-tively, we could code this as

$\exists x\,(B(x) \wedge \neg F(x))$

meaning: ‘There is some x which is a bird and cannot fly.’ Note that the first version is closer to the linguistic structure of the sentence above. These two formulas should evaluate to T in the world we currently live in since, for example, penguins are birds which cannot fly. Shortly, we address how such formulas can be given their meaning in general. We will also explain why formulas like the two above are indeed equivalent semantically.

Coding up complex facts expressed in English sentences as logical formulas in predicate logic is important – e.g. in software design with UML or in formal specification of safety-critical systems – and much more care must be taken than in the case of propositional logic. However, once this translation has been accomplished our main objective is to reason symbolically () or semantically () about the information expressed in those formulas.

we extend our natural deduction calculus of propositional logic so that it covers logical formulas of predicate logic as well.