Propositional Logic

Declarative Sentence

we need to develop a language in which we can express sentences in such a way that brings out their logical structure. The language we begin with is the language of propositional logic. It is based on propositions, or declarative sentences which one can, in principle, argue as being true or false.

Examples of declarative sentences are:

(1) The sum of the numbers 3 and 5 equals 8.

(2) Jane reacted violently to Jack's accusations.

(3) Every even natural number >2 is the sum of two prime numbers.

(4) All Martians like pepperoni on their pizza.

(5) Albert Camus ´etait un ´ecrivain fran¸cais.

(6) Die W¨urde des Menschen ist unantastbar.

These sentences are all declarative, because they are in principle capable of being declared 'true', or 'false'.

Sentence (1) can be tested by appealing to basic facts about arithmetic (and by tacitly assuming an Arabic, decimal representation of natural numbers).

Sentence (2) is a bit more problematic. In order to give it a truth value, we need to know who Jane and Jack are and perhaps to have a reliable account from someone who witnessed the situation described. In principle, e.g., if we had been at the scene, we feel that we would have been able to detect Jane's violent reaction, provided that it indeed occurred in that way.

Sentence (3), known as Goldbach's conjecture, seems straightforward on the face of it. Clearly, a fact about all even numbers >2 is either true or false. But to this day nobody knows whether sentence (3) expresses a truth or not. It is even not clear whether this could be shown by some finite means, even if it were true. However, in this text we will be content with sentences as soon as they can, in principle, attain some truth value regardless of whether this truth value reflects the actual state of affairs suggested by the sentence in question.

Sentence (4) seems a bit silly, although we could say that if Martians exist and eat pizza, then all of them will either like pepperoni on it or not. Again, for the purposes of this text sentence (4) will do. Et alors, qu'est-ce qu'on pense des phrases (5) et (6)?

Sentences (5) and (6) are fine if you happen to read French and German a bit. Thus, declarative statements can be made in any natural, or artificial, language.

The kind of sentences we won't consider here are non-declarative ones, like

Could you please pass me the salt?

Ready, steady, go!

May fortune come your way.

Primarily, we are interested in precise declarative sentences, or statements about the behaviour of computer systems, or programs. Not only do we want to specify such statements but we also want to check whether a given program, or system, fulfils a specification at hand. Thus, we need to develop a calculus of reasoning which allows us to draw conclusions from given assumptions, like initialised variables, which are reliable in the sense that they preserve truth: if all our assumptions are true, then our conclusion ought to be true as well. A much more difficult question is whether, given any true property of a computer program, we can find an argument in our calculus that has this property as its conclusion. The declarative sentence (3) above might illuminate the problematic aspect of such questions in the context of number theory. The logics we intend to design are symbolic in nature. We translate a certain sufficiently large subset of all English declarative sentences into strings of symbols. This gives us a compressed but still complete encoding of declarative sentences and allows us to concentrate on the mere mechanics of our argumentation. This is important since specifications of systems or software are sequences of such declarative sentences. It further opens up the possibility of automatic manipulation of such specifications, a job that computers just love to do1. Our strategy is to consider certain declarative sentences as being atomic, or indecomposable, like the sentence 'The number 5 is even.' We assign certain distinct symbols p, q, r,..., or sometimes p1, p2, p3,... to each of these atomic sentences and we can then code up more complex sentences in a compositional way.

For example, given the atomic sentences

p: 'I won the lottery last week.'

q: 'I purchased a lottery ticket.'

r: 'I won last week's sweepstakes.'

we can form more complex sentences according to the rules below:

¬: The negation of p is denoted by ¬p and expresses 'I did not win the lottery last week,' or equivalently 'It is not true that I won the lottery last week.'

∨: Given p and r we may wish to state that at least one of them is true: 'I won the lottery last week, or I won last week's sweepstakes;' we denote this declarative sentence by p ∨ r and call it the disjunction of p and r2.

∧: Dually, the formula p ∧ r denotes the rather fortunate conjunction of p and r: 'Last week I won the lottery and the sweepstakes.'

→: Last, but definitely not least, the sentence 'If I won the lottery last week, then I purchased a lottery ticket.' expresses an implication between p and q, suggesting that q is a logical consequence of p. We write p → q for that3. We call p the assumption of p → q and q its conclusion.

Of course, we are entitled to use these rules of constructing propositions repeatedly. For example, we are now in a position to form the proposition p ∧ q → ¬r ∨ q which means that 'if p and q then

not r or q'. You might have noticed a potential ambiguity in this reading. One could have argued that this sentence has the structure 'p is the case and if q then . ..' A computer would require the insertion of brackets, as in $(p \wedge q) \rightarrow ((\neg r) \vee q)$ to disambiguate this assertion. However, we humans get annoyed by a proliferation of such brackets which is why we adopt certain conventions about the binding priorities of these symbols.