MAU22C00 Lecture 4

John Stalker

Trinity College Dublin

Example language: elementary arithmetic

Combining elements of the other languages we've seen, and quantifiers, gives a language for arithmetic we'll talk more about later.

```
statement : bool_exp ;
bool_exp : ( ¬ bool_exp) | ( bool_exp operator bool_exp )
         | ( quantifier variable . bool_exp )
         | ( num_exp relation num_exp ) ;
operator : ∧ | ∨  | ⊃ ;
quantifier : ∀ | ∃ ;
variable : v | w | x | y | z ... ;
relation : = | ≤ ;
num_exp : 0 | variable | num_exp '
        | ( num_exp + num_exp ) | (num_exp · num_exp ) ;
```

Elementary arithmetic, continued

The new elements here are

- The quantifiers `∀`, read "for all", and `∃`, usually read "there exists".
- We have two types of expressions, boolean and numerical. The former are true or false. The latter have values in the natural numbers, 0, 1, 2, ...
- The ' operator, which increments whatever it's applied to.
- We're not using the usual notation for numbers. There are no `1`, `2`, etc. The first few natural numbers are `0`, `0'`, `0''`, `0'''`, ...

When we return to arithmetic properly later I'll allow [,], {, and } and explain how to allow infinitely many variables.

Elementary arithmetic, continued

An example statement is

$$(((x = y) \wedge (y = z)) \supset (x = z)),$$

which says that if $x = y$ and $y = z$ then $x = z$.

The usual convention is to interpret such statements as having implicit universal quantifiers, i.e. that it means the same as

$$(\forall x.(\forall y.(\forall z.(((x = y) \wedge (y = z)) \supset (x = z))))),$$

In other words, for all natural numbers x, y and z it's true that if $x = y$ and $y = z$ then $x = z$.

Elementary arithmetic, continued

The statement $(((x = y) \wedge (y = z)) \supset (x = z))$ happens to be true, with the intended interpretation, but our language contains plenty of false statements as well, like $0 = 0'$.

The true statements also form a language, which is a subset of our full language.

Later on we'll turn this into a formal system, by introducing axioms and rules of inference. This allows us to give formal proofs of some statements. The set of statements which can be proved will then also be a language.

What types of languages are these?

The full language is context free but not regular. The language of provable statements is recursively enumerable but not context free. The language of true statements is not even recursively enumerable.

Truth is not a grammatical property. Also, either there are true statements which cannot be formally proved or there are false statements which can be formally proved!

What now?

We will return to the study of formal languages later, but for first we'll discuss

- Zeroeth order logic
- First order logic
- Elementary arithmetic
- Set theory

Each of these will have its own formal language. The language of first order logic builds on zeroeth order logic and the languages of elementary arithmetic and set theory build on first order logic.

After we've discussed all of them we'll return to languages in general, and their relation to idealised machines.

Levels of proof

- Formal: a derivation of a statement in a formal language from a set of axioms according to a set of rules of inference.
- Informal: a (convincing?) argument that a statement is true, in its intended interpretation. The statement may be, but usually isn't expressed in a formal language.
- Semiformal: An informal proof that a formal proof exists, without actually producing such a proof.

All of these have their uses, and all will appear in this module.

A *formal system* consists of a language, a set of axioms, and a set of rules of inference, but not an interpretation. You can't really have either a formal or semiformal proof without a formal system.

Logic

Proofs, of any type, generally start from premises (hypotheses) and work towards conclusions.

You can only trust the conclusions when the premise are true and the reasoning leading from them to the conclusions is sound.

Logic deals *exclusively* with the second question, the soundness of the reasoning.

Zeroeth order logic is concerned with Boolean operators, or operators expressible in terms of them.

First order logic adds quantification, i.e. “for all” and “for some”, and some other elements.

Higher order logic also exists, but isn’t popular. Most of mathematics is built on first order logic plus set theory.

The language of zeroeth order logic

```
%start statement
%%
statement  : expression ;
expression : variable
           | ( expression binop expression )
           | [ expression binop expression ]
           | { expression binop expression }
           | ( ¬ expression )
           | [ ¬ expression ]
           | { ¬ expression } ;
variable   : letter | variable ! ;
letter     : p | q | r | s | u ;
binop      : ∧ | ∨ | ⊃ | ⊼ | ⊻ | ≡ | ≢ | ⊂ ;
```

Comments on the language

We'll only use the operators $\neg$, $\wedge$ $\vee$ $\supset$ and, very briefly, $\overline{\wedge}$.

Expressions are always fully parenthesised. We don't rely on precedence or associativity rules.

That makes stating rules of inference much easier, but leads to lots of parentheses. The three types of parentheses exist only to make statements with many nested parentheses easier (for humans) to read.

For reasoning *about* the language it's convenient if there are infinitely many variables. In addition to the letters p, q, r, etc. we also allow $p!$, $q!!!!$, $r!!!$, etc. We'll never need them when reasoning *within* the language though.

Every statement is an expression, but not every expression is a statement. Some rules of inference will operate at the level of statements and others will operate at the level of expressions.