课程信息简介

1 课程内容

这门程序验证课会讲授一系列关于程序正确性的理论知识，并介绍一些实践中较为可行的技术方法。

- 程序语义理论
- 程序逻辑理论
- 程序验证技术与应用

2 课程评分

本课程的期末总评分分为三个部分。

- 课堂参与 10 分
- 平时作业 50 分
- 大作业 40 分

3 语法与语义

下面三组 C 程序语句中，每一组内的两句程序语句是相同的程序语句吗？第一组：

```
y=x+1; // 代码中的空格更少
```

```
y = x + 1; // 代码中的空格更多
```

第二组：

```
y = (x) + 1; // 代码中的包含多余的括号
```

```
y = x + 1; // 代码中无多余的括号
```

第三组：

```
y = 1 + x;
```

```
y = x + 1;
```

对于描述程序行为与程序正确性的理论而言，像第一组例子中的多余空格与第二组例子中的多余括号并不重要，因此，我们一般会以程序的抽象语法树来定义程序。这样看的话，上面三组程序语句中，第一组与第二组都包含了相同的程序语句。而第三组中的两句程序语句则是不同的程序语句。

程序行为则是完全不同于程序语法的概念。有许多种方法可以描述程序行为。例如我们可以描述程序运行前后的整体效果，我们也可以在此基础上进一步描述程序运行过程中的所有中间步骤。

4 例子：布尔表达式的语法与语义

下面以只包含布尔变元以及“与”“或”“非”的布尔表达式为例，分别定义它们的语法与语义。

首先可以如下定义这类布尔表达式的语法，其中 V 表示布尔变元集合 Σ 中的变元名称。

```
E ::= V  |  E && E  |  E || E  |  ! E
```

每个布尔表达式 e 的语义（$[\![e]\!]$）可以定义为从真值指派到真值的函数，即定义为这个表达式在每个的真值指派（从布尔变元到真值的映射）上的真值。具体的，假设 $J : \Sigma \to \{\text{true}, \text{false}\}$ 是一个真值指派，那么

- $[\![P]\!](J) = J(P)$，若 $P \in \Sigma$；
- $[\![e_1 \ \&\& \ e_2]\!](J) = [\![\&\&]\!]([\![e_1]\!](J), [\![e_2]\!](J))$；
- $[\![e_1 \ || \ e_2]\!](J) = [\![||]\!]([\![e_1]\!](J), [\![e_2]\!](J))$；
- $[\![!e_1]\!](J) = [\![!]\!]([\![e_1]\!](J))$。

其中，$[\![\&\&]\!]$、$[\![||]\!]$ 与 $[\![!]\!]$ 表示下面的真值函数：

- $[\![\&\&]\!](\text{true}, \text{true}) = \text{true}$，$[\![\&\&]\!](\text{true}, \text{false}) = \text{false}$，
  $[\![\&\&]\!](\text{false}, \text{true}) = \text{false}$，$[\![\&\&]\!](\text{false}, \text{false}) = \text{false}$；
- $[\![||]\!](\text{true}, \text{true}) = \text{true}$，$[\![||]\!](\text{true}, \text{false}) = \text{true}$，
  $[\![||]\!](\text{false}, \text{true}) = \text{true}$，$[\![||]\!](\text{false}, \text{false}) = \text{false}$；
- $[\![!]\!](\text{true}) = \text{false}$，$[\![!]\!](\text{false}) = \text{true}$。

在此基础上我们可以看到，如果 P 与 Q 是两个不同的布尔变元，那么 $P||Q$ 与 $Q||P$ 是不同的布尔表达式（语法树不同），但是具有相同的语义。

5 Coq 中的函数、谓词与证明

Coq 表达式 1. 包含函数的表达式. 在 Coq 中，某函数 `F` 作用于某参数 `X` 写作 `F X` ，不需要写括号。这一语法类似于 Ocaml 等函数式编程语言。另外，这一语法是左结合的。换言之，表达式 `F X Y` 是 `(F X) Y` 的简写，而表达 `F (G X)` 时必须添加括号。

Coq 证明脚本 1. unfold 指令. 证明指令 `unfold` 表示在待证明的结论中展开某项定义。如果要在证明目标的某前提 `H` 中展开 `X` 的定义，可以使用证明指令 `unfold X in H` 。

Coq 表达式 2. 包含多元函数的表达式. Coq 中的二元函数实质上是接收一个参数后会计算得到一个一元函数的函数。例如，当 `F` 是一个二元函数时，我们通常将“`F` 作用于 `X` 与 `Y` 的结果”写作 `F X Y` ，即 `(F X) Y` 的简写。这是因为 `F X` 实质上是一个一元函数，当他再接收一个参数 `Y` 之后的计算结果就是 `(F X) Y` 。类似的，Coq 中的三元函数实质上是接收一个参数后会计算得到一个二元函数的函数；Coq 中的 n+1 元函数实质上是接收一个参数后会计算得到一个 n 元函数的函数。