Class 26: Type Checking (1) Introduction

Held: Monday, 31 October 2011

Summary: Today we begin our consideration of a key semantic step: type checking. That is, we consider where and how one might verify types in a program. We also consider related issues, such as the definition of type.

Related Pages:
- EBoard.
- Reading: Aho et al, 6.3, 6.6.

Notes:
- Exams returned.
- Most of you did worse on the exam than you or I would have liked. I expect that you were not expecting this kind of exam. I will offer a makeup exam on Friday. It will again have four problems. I will take the best of your grades on corresponding problems. I will then drop the computed grade by 1/3 a letter, unless that gives you a lower grade than exam 1.

Overview:
- Status, reviewed.
- Introduction to Type Checking.
- When Do You Type Check? Compile-time vs. Run-time Type Checking.
- Common Types.
- Where Do You Type Check?

Where We Are And Where We’re Going
- We can now write parsers that build parse trees!
  - We can even write simple parser generators, although we have not tried to do so.
- It is time to move from the front end to the back end.
- What do we have to do before we start generating code?
  - Build symbol tables.
  - Do some semantic analysis to validate the program.
  - Reflect on the structure of the code we will generate.
- Today we start moving forward.
An Introduction to Type Checking

- The problem of type checking is one of the first important semantic aspects of compilation. (What we’ve been doing so far is more syntactic.)
- Essentially, type checking
  - allows the programmer to limit what types may be used in certain circumstances;
  - assigns types to values;
  - determines whether those values are used in an appropriate way.
- Type checking is also useful because it has relevance to generated code (e.g., you will often generate a different assembly instruction for adding reals and adding ints)
- There are many variants of type checking. For example, if the type of an object is “incorrect”, you may coerce it to a different type.
- The definition of types is subtle; it’s often worth asking when two values should be considered as having the same type.

Compile-Time vs. Run-Time Type Checking

- In many modern languages, type checking is done at compile time.
- In some, it’s done at run time.
- What are some advantages for type checking at compile time?
  - Catches many common errors.
  - Can result in faster code.
  - ...
- What are some advantages for type checking at run time?
  - Frees the programmer.
  - Can result in more robust code.
  - Can result in clearer code.
  - ...
- Pascal type checking is traditionally done at compile time.

Common Types

- Built-in types (integers, floating-point numbers, strings, ...)
- Compound types (arrays, lists, records, classes)
- User-defined enumerated types.
- ...

Where Should You Type Check?

- Where in the program do you look at types?
- We’ll leave this for class discussion.
Computing Types

- In Pascal, the types of variables are declared in advance.
- However, we still need to type other things (such as functions and ...)
- We’ll consider as a class what kinds of things should be typed and where we use types.