Class 27: Type Checking (2) Type Equivalence

Held: Wednesday, 2 November 2011

Summary: Today we continue to consider issues of type equivalence. That is, we explore when two types can be treated as equivalent for purposes of assignment and function/procedure calls.

Related Pages:
  - EBoard.

Notes:
  - Are there questions on phase 3 of the project?
  - Are there questions in preparation for Friday’s examination?
  - EC for Jin Feng’s talk tomorrow at 4:15 p.m.

Overview:
  - Why Type Check?
  - Detour: Type Coercion in C.
  - Type Equivalence: Subranges and Enumerated Types.
  - Type Equivalence; Records.

Why Type Check?
  - Type checking has many uses, although there are two big ones.
  - We type check to verify that code is “correct”
    - Usually, when programmers fail to match types, they’ve made a mistake.
  - We also type check to let us generate code
    - Most architectures have one instruction for adding two integers and a different one for adding two floating point numbers.
    - If we’re coercing types, we need to include the coercion instructions.
  - So, we have a few basic activities for type checking
    - Determining the type of an expression
    - Checking whether that type is appropriate in the context in which it is used
    - If not, recovering appropriately

Fun with C

I thought you might enjoy this quote from Kernighan & Ritchie (2nd edition, p. 198).
Many operators cause conversions and yield result types in a similar way. The effect is to bring operands into a common type, which is also the type of the result. This pattern is called the usual arithmetic conversions.

First, if either operand is a `long double`, the other operand is converted to a `long double`.

Otherwise, if either operand is a `double`, the other is converted to `double`.

Otherwise, if either operand is a `float`, the other is converted to `float`.

Otherwise, the integral promotions are performed on both operands; then, if either operand is `unsigned long int`, the other is converted to `unsigned long int`.

Otherwise, if one operand is `long int` and the other is `unsigned int`, the effect depends on whether a `long int` can represent all values of an `unsigned int`; if so, the `unsigned int` operand is converted to `long int`; if not, both are converted to `unsigned long int`.

Otherwise, if one operand is `long int`, the other is converted to `long int`.

Otherwise, if one operand is `unsigned int`, the other is converted to `unsigned int`.

Otherwise, both operands have type `int`.

Wasn’t that fun? And it doesn’t even deal with reals.

**Type Equivalence: Subrange Types**

- A key issue in language design is deciding whether two types are “the same” for purposes of calls and assignment.
- These questions often have to do with user-defined types.
- Consider the following type and variable declarations.

```plaintext
type
    day = (Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday);
    weekday = Monday .. Friday;
    daynum = 0 .. 6;
    grade = 0 .. 100;
var
    d: day;
    w: weekday;
    n: daynum;
    g: grade;
    i: integer;
```

- Which of the following do you think should be legal (or illegal)? Should the checking be done at compile time or run time?
Type Equivalence: Records

- Consider the following type and variable declarations:

```pascal
type
    t0 = record a: integer; b: integer end;
t1 = record a: integer; b: real end;
t2 = record a: integer; b: real end;
t3 = record b: real; a: integer end;
t4 = record c: integer; b: real end;
t5 = record a: integer; b: real; c: integer; end;
t6 = t1;

var
    v0: t0;
v1: t1;
v2: t2;
v3: t3;
v4: t4;
v5: t5;
v6: t6;
v7: record a: integer; b: real end;
v8, v9: record a: integer; b: real end;
```

- Which of v0 ... v9 can we assign to v1? (v1 := v*)
- Which of v0 ... v9 can we assign v1 to? (v* := v1)
- Can we assign v7 and v8 to each other?
- Pascal gives one set of specifications, but there are others possible (which depend on the language).
- We can ask similar questions about enumerated types, array types, and pointer types.
- Pointer types can introduce some fairly complicated questions, which may lead us to make different decisions about pointer types than we make about record types.

More Type Equivalence

- Kinds of type equivalence:
  - Structural: Two types are the same if they have the same structure.
  - Name: Two types are the same if they have the same name (or can be reduced to the same name through type aliases).
- We’ll continue our example from the records examples considering some unnamed types.
- We’ll also look at some odd consequences of structural equivalence for recursively defined types.
- We’ll consider related issues for arrays
  - Size
- Bounds
- Name

- We’ll also consider the implications of type names on someone’s ability to write “constants” in a type.