Class 29: Type Checking (3) Wrapup

Held: Monday, 7 November 2011

Summary: Today we continue our explorations of type checking and type equivalence. We consider type equivalence for records as a way of thinking more deeply about the subject. Then we attempt to generalize.

Related Pages:
- EBoard.
- Due: Project, Phase 3: Symbol Table.

Notes:
- Where was Sam on Friday? Talking to a publisher about the introductory Java course.
- What should you be reading? 5.1 and 5.2 are about attributes. 6.3 and 6.5 have further information about type checking. 7.1 through 7.3 are about stacks, our next topic.
- Are there questions on the current homework assignment?
- Project, Phase 4 (coming soon): Type Checking.

Overview:
- Type Equivalence: Records.
- Type Equivalence, Generalized.

Type Equivalence: Records

- Consider the following type and variable declarations:

```plaintext
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 := vi)
- Which of v0 ... v9 can we assign v1 to? (vi := 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.
- How do your answers change if our types are as follows?

```plaintext
type
  Point = record x: real; y: real end;
  GridPoint = record x: integer; y: integer end;
  Money = record dollars: integer; cents: integer end;
  Vector = record radius: real; theta: real end;
```

**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).
- What do we do about unnamed types? E.g.,

```plaintext
var x: record a: integer; b: integer end;
```

- How do we deal with structural equivalence for recursively defined types? E.g.,

```plaintext
type
  node1 = record value: int; next: ^node1; end;
  node2 = record value: int; next: ^node2; end;
  node3 = record value: int; next: ^node4; end;
  node4 = record value: int; next: ^node3; end;
```

- Are arrays much different? We should consider
  - Size
  - Bounds
  - Name
- What are the restrictions on writing constants?