Type checking and compatibility

Amer Diwan

Recall..

- A type system is a suit of armor that protects the groups of values
- What does this protection entail?
  - When are types checked?
  - How “completely” are types checked?
  - What does type checking involve?
When to do “type checking”

- **Static type checking**: all checking at compile/link time
- **Dynamic type checking**: all checking at run time
- Most languages fall somewhere in between e.g., Java and Modula-3

Examples of type checking

- `int i;
  int j;
  i = j;`
- `char *p;
  void *q;
  p = q;`
- `int i;
  char c;
  c = i;`
Static versus dynamic typing

- Expressiveness
- Simplicity
- Safety
- Ease of implementation
- Efficiency

How strong is the checking

- **Strongly-typed**: All expressions are checked (and guaranteed) to be type-consistent at compile or run time (type-safe).
  - Pascal, Modula-2, Modula-3, Java, ...
- **Weakly-typed**: Some expressions are not completely typechecked and unchecked type violations may happen at run time
  - C, C++, assembler
Strong and weak typing example

• char *p;
  void *q;

  p = q;
• int i;
  char c;

  c = i;

Strong versus weak typing

• Expressiveness
• Simplicity
• Safety
• Ease of implementation
• Efficiency
What does type-checking involve?

- When is $x:T = y:U$ legal?
- When is $x:T \text{ op } y:U$ legal?

Type equality

- When are two types, T1 and T2, equal?
  - **Name equivalence**: when T1 and T2 have the same name. *Anonymous* types have unique “names”.
  - **Structural equivalence**: when T1 and T2 have the same structure.

- Structural equivalence: Modula-3, Algol
- Name equivalence: Modula-2, Java, Ada
- A bit of each: C, C++
- Undefined: Pascal
Type equality example

TYPE
T1 = RECORD i: INTEGER; b: BOOLEAN; END;
T2 = RECORD i: INTEGER; b: BOOLEAN; END;
T3 = T2;

• T1 = T2? T1 = T3? T2 = T3?

A simple algorithm for structural equivalence

• T1 = T2 =>
  – Replace all names in T1 and T2 with their expansion until T1 and T2 do not contain any type names
  – T1 = T2 if their expansions are identical
Structural equivalence example

```
TYPE T1 = RECORD i: INTEGER; b: T3; END;
TYPE T2 = RECORD i: INTEGER; b: T4; END;
TYPE T3 = RECORD x: BOOLEAN; END;
TYPE T4 = RECORD x: BOOLEAN; END;
```

T1 ?= T2 expands into:
```
RECORD i: INTEGER; b: T3; END; ?=
RECORD i: INTEGER; b: T4; END;
```
Which further expands into
```
RECORD i: INTEGER; b: RECORD x: BOOLEAN; END; END; ?=
RECORD i: INTEGER; b: RECORD x: BOOLEAN; END; END;
```
which are identical

Another example

```
TYPE T1 = RECORD i: INTEGER; b: T2; END;
TYPE T2 = RECORD i: INTEGER; b: T1; END;
```
Another examples

• TYPE
  Score = [0..100];
  AssignNum = [0..100]

Another example

• TYPE
  Employee = RECORD i: INTEGER;
  AssignNum = [0..100]
Type equivalence and distributed environments

Program Producer()
  a: ARRAY [1..1024] OF INTEGER
  send(Consumer, a)

Program Consumer()
  a: ARRAY [1..1024] OF INTEGER
  receive(a)

Name versus structural equivalence

- Expressiveness
- Simplicity
- Safety
- Ease of implementation
- Efficiency
More examples

- x: INTEGER; y: INTEGER;
  x := y;
- x: INTEGER; y: [1..100]
  x := y;
- Subtyping captures the non-equality cases of type-checking

Subtyping

- Type $S$ is a subtype of type $T$ if every value of type $S$ is also a value of type $T$
- Written as $S <: T$
One way to think about subtyping

- Subsetting of values
- E.g.,
  - \([10..20]\) <: INTEGER?
  - INTEGER <: \([10..20]\)?
  - INTEGER <: INTEGER?
  - OBJECT i: INTEGER; END <: OBJECT j: INTEGER; END ?
  - \(T = \text{OBJECT } i: \text{INTEGER}; \text{END}\)
  - \(T <: T \text{ OBJECT } j: \text{INTEGER}; \text{END}\)?

Another way to think about subtyping

- Substitutability of types
- E.g.,
  - \(\text{PROCEDURE } f(p: \text{INTEGER}) = \ldots\)
    - \(\text{VAR } i: \text{INTEGER}; j: \text{[1..10]};\)
    - is \(f(i)\) legal?
    - is \(f(j)\) legal?
- E.g., subclass used when a supertype is expected
Yet another way to think about subtyping

• A subtype has more stringent membership requirements than a supertype
• E.g.,
  – \( v \in \text{INTEGER} \)
  – \( v \in [10..20] \)
  – \( \text{TYPE T = OBJECT } i: \text{INTEGER}; \text{END;} \)
  – \( S = \text{T OBJECT } j: \text{INTEGER}; \text{END;} \)
  – \( v \in T \)
  – \( v \in S \)

When is type \( A <: \) type \( B \)

• Trivially if \( A = B \)
• Transitively if \( A <: C \) and \( C <: B \)
• Subtyping between integers and subranges is easy: directly apply value inclusion
• How about sets?
  \( A = \text{SET of } \{ \text{red, blue} \} \)
  \( B = \text{SET of } \{ \text{red, blue, yellow} \} \)
When is type A <: type B

Arrays

• A = ARRAY[1..10] OF INTEGER
  B = ARRAY[10..20] OF INTEGER
• A = ARRAY[1..10] OF INTEGER
  B = ARRAY[1..10] OF [1..100]

When is A <: B

Objects

• TYPE T = OBJECT i: INTEGER; END;
  TYPE S = T OBJECT j: CHAR; END;
• (More on this later)
Using subtyping for checking assignments

• When is b: B := a: A legal?

A few spanners in the works...

• VAR x: [10..20];
  y: INTEGER;
  x := y;
• VAR x: [10..20];
  y: [18..30];
  x := y;
Using subtyping for checking expressions

- PROCEDURE f(p: INTEGER) = ... 
- What can we pass to p? 
- Does parameter passing mode matter?

Subtyping and conversions

- Substitutability caused by subtyping gives a form of conversion (widenning):
  - [10..20] -> INTEGER
- Why is it called a widening conversion? 
- Does a widening conversion require a check at run time?
More examples of widening conversions

- From $A = \text{ARRAY}[1..10] \text{ OF } [1..100]$ to $B = \text{ARRAY}[1..10] \text{ OF INTEGER}$

- From $A = \text{SET} \text{ of } \{\text{red, blue}\}$ to $B = \text{SET} \text{ of } \{\text{red, blue, yellow}\}$

Narrowing conversion

- From supertype to subtype
- INTEGER -> [10..20]
- Why is it called a narrowing conversion?
- Do narrowing conversions require checks at run time?
Non-converting type conversions

• Narrowing and widening conversions are “safe”: they only convert between incompatible types
• non-converting casts reinterpret the bits of one value to be another type
• e.g.,
  – i: INTEGER; j: REF INTEGER;
    i := LOOPHOLE(j, INTEGER)
• Useful for low-level coding

Next topic: Examples from Java and Modula-3

• How are types incorporated into some common languages