The Problem of Structural Type Tests in a Gradual-Typed Language

John Boyland

University of Wisconsin-Milwaukee

FOOL 2014, Portland, Oregon

Summary

- Structural types classify object behavior
- Gradual typing allows types to be implicit
- Type assertions bridge static and dynamic typed code, but can be delayed.
- Type tests require a yes/no decision, and are not compatible with gradual structural types.

Structural Types (1 of 3)

- Contrasted with *Nominal* Typing:
  - Object gets type from its class:

    ```
    new Window(...) : Window
    ```

  - Name of class is the type, subtyping follows declared extension/implements.
  - E.g. Java

Structural Types (2 of 3)

- The type indicates the structure of a value:

  `{x=3, y="foo"} : {x:int, y:string}`

- We may optionally name the type:

  `type MyType = {x:int, y:string}`

- But the name is merely an abbreviation for the real (structural) type.
- e.g. OCamL

Structural Types (3 of 3)

- A variant of "duck-typing":

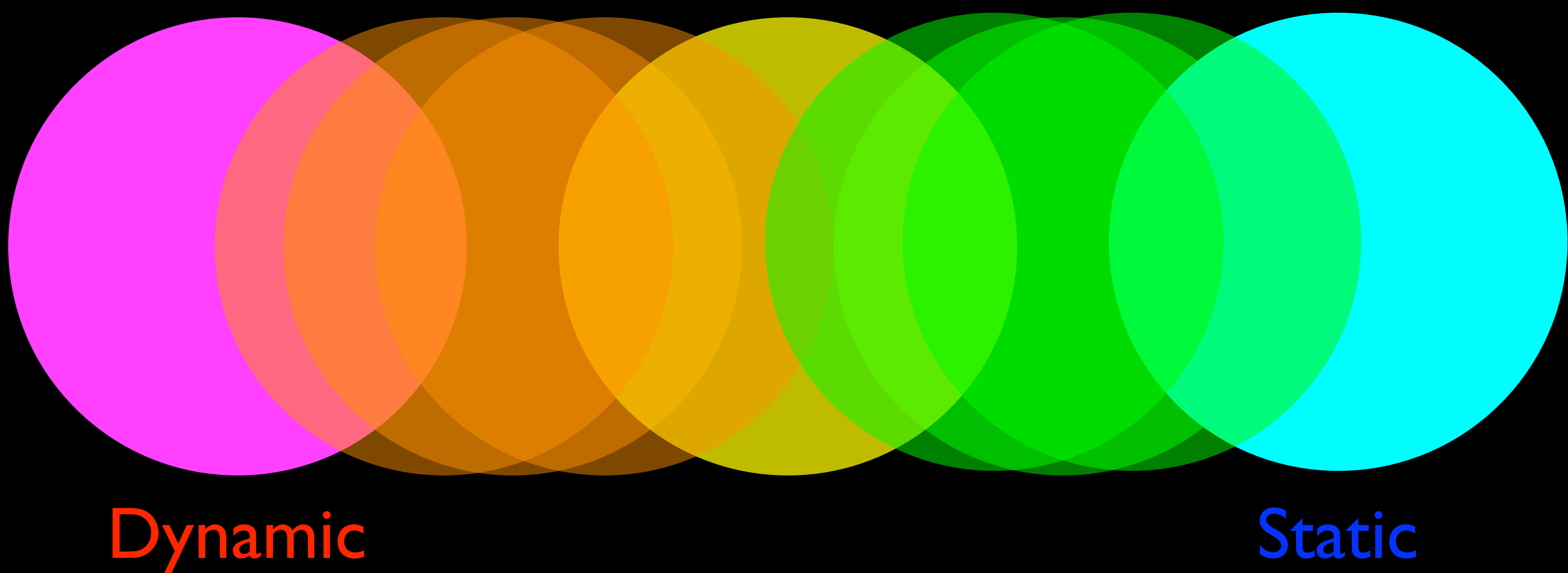
  If it walks likes a duck and quacks like a duck, it is a duck.

- Useful in prototyping languages (no classes):

- ```
  type Duck = {
    walk(n:Number) -> Location,
    quack : String }
  ```
  ```

Structural Types (3 of 3)

- A variant of “duck-typing”:

  If it walks likes a duck and quacks like a duck, it is a duck.

- Useful in prototyping languages (no classes):

- ```
  type Duck = {
    walk(n:Number) -> Location,
    quack : String }
  ```

Traditionally duck typing doesn’t declare types
  ```

Gradual Typing (1 of 4)

- Gradual typing [Siek&Taha 2006,2007] encompasses dynamic and static typing:



Gradual Typing (2 of 4)

- Encompasses dynamic typing:
  - compiler doesn't require type annotations.
- Encompasses static typing:
  - compiler checks types if given,
  - fully typed implies type safety.
- Supports partially typed programs.

Gradual Typing (3 of 4)

- Uses quasi-type `?` for untyped vars (AKA `Dynamic`, `Unknown`)
- When passing values from untyped to typed code, the type is checked (*type assertion*).
- Exactly how and when these assertions are done depends on technique: inc. Guarded, Transient, Monotone. [Vitousek *et al* 2014]

Gradual Typing (3 of 4)

- Uses quasi-type ? for untyped vars (AKA `Dynamic`, `Unknown`)
- When passing values from untyped to typed code, the type is checked (*type assertion*).
- Exactly how and when these assertions are done depends on technique: inc. Guarded, Transient, Monotone. [Vitousek *et al* 2014]

See DLS Tomorrow!

Gradual Typing (4 of 4)

- Encourage rapid prototyping.
- Encourage static typing:
  - as enforceable documentation;
  - fail-fast (statically, or on dynamic entry) if code used incorrectly;
  - to enable optimization.

Gradual Typing (4 of 4)

- Encourage rapid prototyping.
- Encourage static typing:
  - as enforceable documentation;
  - fail-fast (statically, or on dynamic entry) if code used incorrectly;
  - to enable optimization.

DON'T BREAK CODE!

Gradual Guarantee

- Runtime Semantics is independent of typing
  - A unified language with a single semantics;
  - As in old semantics of Standard ML.
- Type annotations don't break the program.
- A working program does not break if a type annotation is omitted.

Gradual Guarantee

- Runtime Semantics is independent of typing
  - A unified language with a single semantics;
  - As in old semantics of Standard ML.
- ~~Type annotations don't break the program.~~
- A working program does not break if a type annotation is omitted.

Type Assertions

- Easy, if an atomic type (e.g., String)
- Tricky, if a function type
  - argument type depends on use;
  - result type depends on result.
- Some checks left to later
  (for details see DLS talk)
- Failure may happen at a distant time and loc.

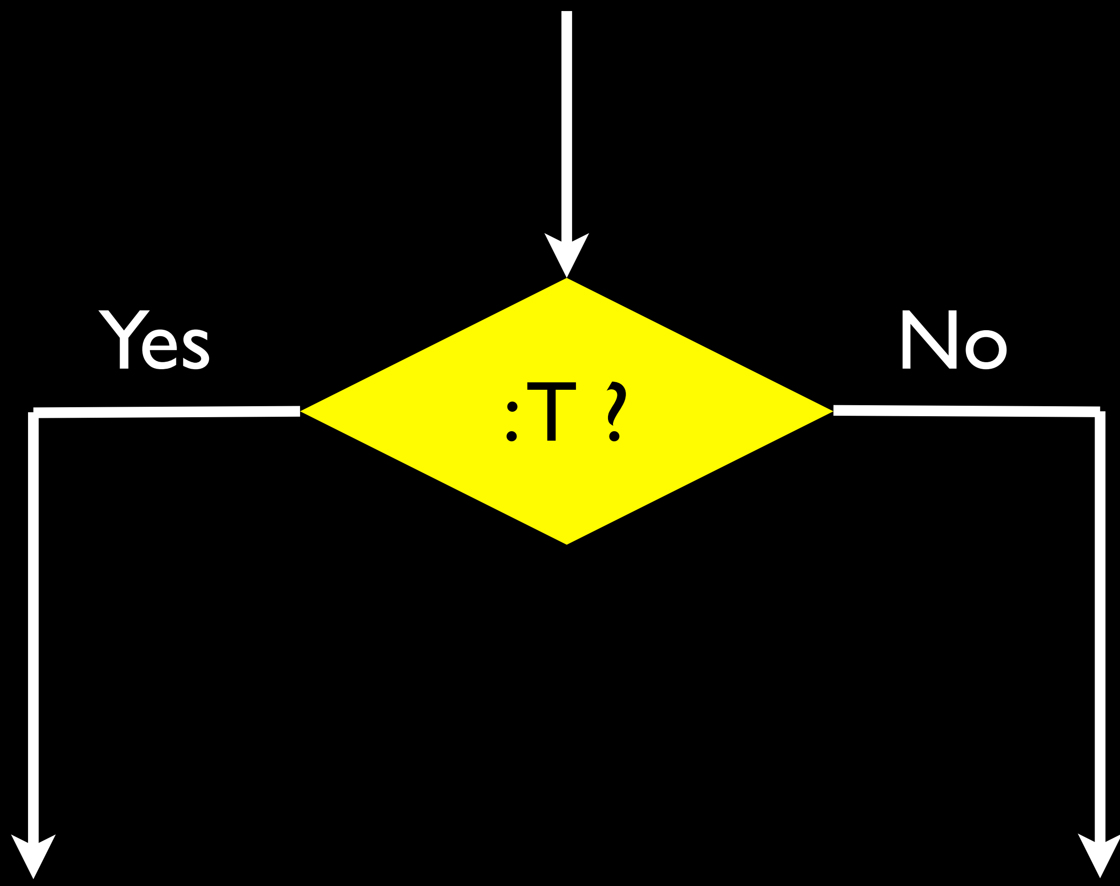
Type Tests

- Check the type of a value dynamically:
  - e.g. `instanceof` in Java;
  - `match` in Scala (also used for patterns).
- If test fails, program can try something else:
  - failure is not an error.
- Nominal type test simply examines class tag.

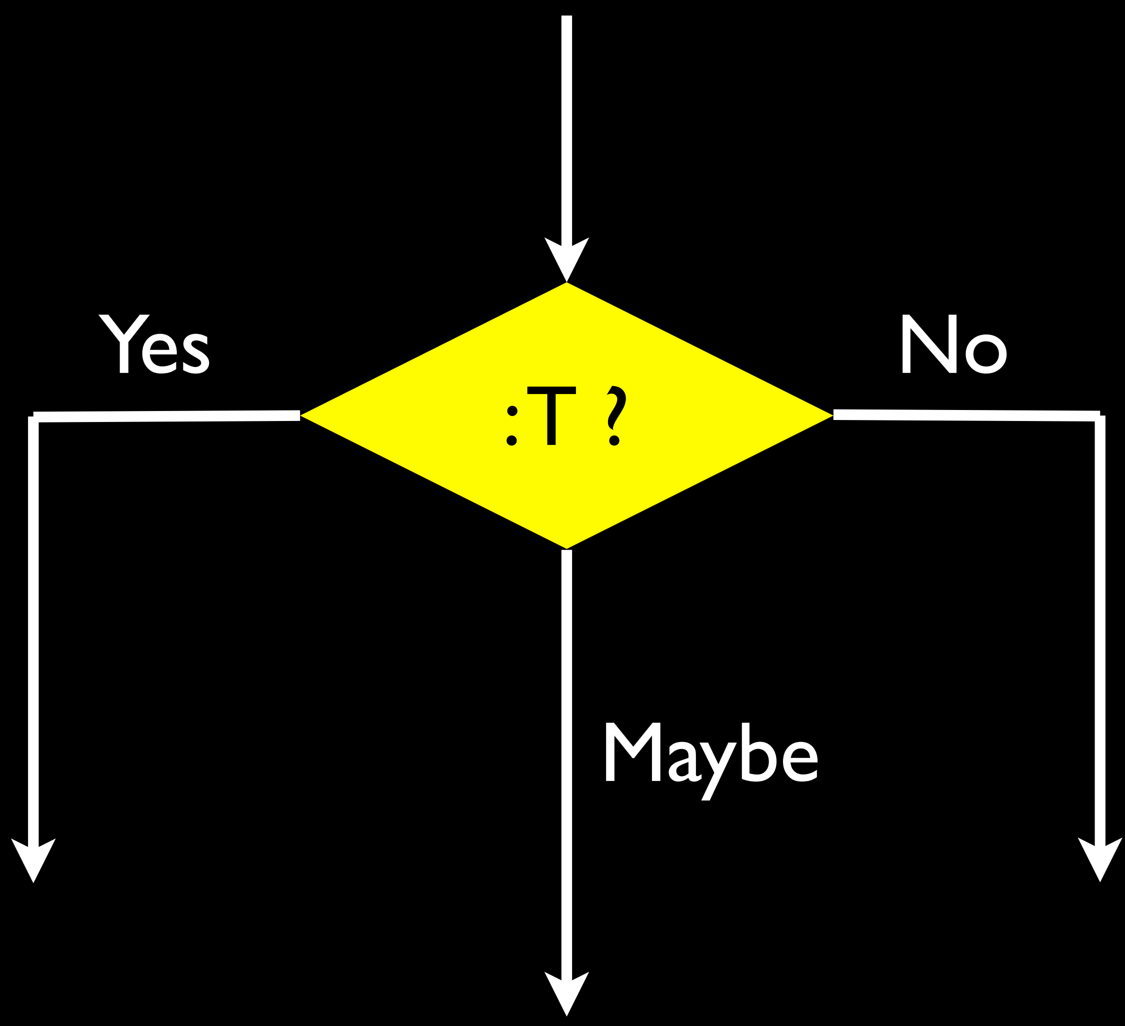
Structural Type Test

- Few languages have them:
  - OCaML does not
  - Java/Scala omit type tests for generics.
- TinyGrace [Jones&Noble 2014]: type test implemented by preserving declared type in values.
- Grace [Black *et al* 2013] is gradual too.

Gradual Type Test? (1)



Gradual Type Test? (1)



Gradual Type Test? (2)

- Gradual guarantee cannot be honored:
  - Omitting a type annotation can obscure a type match success
  - Omitting a type annotation can obscure a type match failure
  - Choosing wrong way changes the semantics of the program.

Gradual Type Test? (3)

- Optimistic: If cannot prove, assume true.
  - Can lead to type errors at run-time.
- Trust-but-verify: Check as much as possible, and then perform type assertion.
  - Can lead to assertion failures later.
- Pessimistic: If cannot prove, assume false.
  - Can lead to missing match at run-time.

Example

```
type Window = { draw(g:Gfx); }

type Cowboy = { draw(g:Gun); }

match (object
        { method draw(g:Gun)
            { g.fire(10); } })

  case {w : Window -> ...}

  case {c : Cowboy -> ...}
```

Example

```
type Window = { draw(g:Gfx); }

type Cowboy = { draw(g:Gun); }

match (object
          { method draw(g:Gun)
              { g.fire(10); } })

  case {w : Window -> ...}

  case {c : Cowboy -> ...}
```

Example

```
type Window = { draw(g:Gfx); }

type Cowboy = { draw(g:Gun); }

match (object
        { method draw(g: ? )
           { g.fire(10); } })

  case {w : Window -> ...}

  case {c : Cowboy -> ...}
```

Example

```
type Window = { draw(g:Gfx); }

type Cowboy = { draw(g:Gun); }

match (object
          { method draw(g: ? )
              { g.fire(10); } })
```

Optimistic

```
  case {w : Window -> ...}
```
ERROR

```
  case {c : Cowboy -> ...}
```

Example

```
type Window = { draw(g:Gfx); }

type Cowboy = { draw(g:Gun); }

match (object
          { method draw(g: ? )
              { g.fire(10); } })

   case {w : Window -> ...}

   case {c : Cowboy -> ...}
```

Pessimistic   ERROR

Responses (1 of 3)

- Type tests examine types and so no one should be surprised that erasing a type annotation changes semantics.

  - James Noble

- With this principle, the pessimistic approach is needed to preserve type safety.

Responses (2 of 3)

- Type tests are a form of reflection, and thus don't need to honor the gradual guarantee.

  - Andrew Black

- With this principle, type tests should not be involved with the static type system (as in `match`), but should simply be predicate calls.

Responses (3 of 3)

- Type tests break parametricity and thus should not be supported.

  - John Boyland

- With this approach, *match* would be limited to pattern matching, with hand-written RTTI if needed.

Conclusions

- There is no satisfactory way to give a semantics to structural type tests in a gradually-typed language (such as Grace).
- Erasing a type annotation can lead to a type error (if optimistic) or a match error (if pessimistic).
- Problem is because types of gradual objects are not known immediately.