DOWN WITH GENERALISATION

Typechecking lets

let y = E in B

- Typecheck E:
  - Find E's type, T
  - Gather type constraints C from E
- Generalise over as, free in T but not in Γ
- Infer type f :: forall as. C => T

eg (a -> a)

eg (Num a)

eg forall a. Num a => a->a

Typechecking lets

let y = E in B

- Typecheck E:
  - Find E's type, T
  - Gather type constraints C from E
- Generalise over as, free in T but not in Γ
- Infer type f :: forall as. C => T
- **BUT such a type is TOO GENERAL**

Should this program be accepted?

```
data T a where
  C :: T Bool
  D :: T a

f :: T a -> a -> Bool
f = \v.\x. let y = not x
             in case v of
                  C -> y
                  D -> True
```

Should this program be accepted?

```
data T a where
  C :: T Bool
  D :: T a

f :: T a -> a -> Bool
f = \v.\x. let y :: (a~Bool) => Bool
               y = not x
           in case v of
               C -> y
               D -> True
```

Implications

```
let y = E in B
```

- No in-place unification at all
- Gather all constraints (no matter how innocuous)
- Abstract over them
- Result:
  - Large incomprehensible types
  - Type errors postponed to call sites

Should this program be accepted?

```
let y = E in B
```

- Typecheck E:
  - Find E's type, T
  - Gather type constraints C from E
- Simplify C "as much as possible", giving D
- Infer type f :: forall as. D => T

Type functions

- But "as much as possible" may vary depending on how much of the rest of the program we've seen
  - D [a] β   instance D [a] Int where ...
  - F [a] β ~ Int   type instance F [a] Int = Int
- The info about b may come from B; but we can't typecheck B until we've decided a tpye for f.

let y = E in B

Observation

- Nasty cases only occur when there are type variables free in the environment; ie, in nested let/where bindings
- Proposal:
  - Never generalise local let/where bindings (except where there is a type signature)
  - Always generalise top-level bindings
- Note: many consider it good style to provide a type signature on all top level bindings, so Proposal amounts to: all polymorphism is explicitly declared

Observation

- Proposal:
  - Never generalise local let/where bindings (except where there is a type signature)
  - Always generalise top-level bindings
- Questions:
  - How many existing programs would break?
    - answer: 10%
  - How inconvenient would the restriction be?
    - SPJ answer: not inconvenient

Slogan

Give up something
that you are used to having
but don't really use

in exchange for

Substantial simplification of
the language design