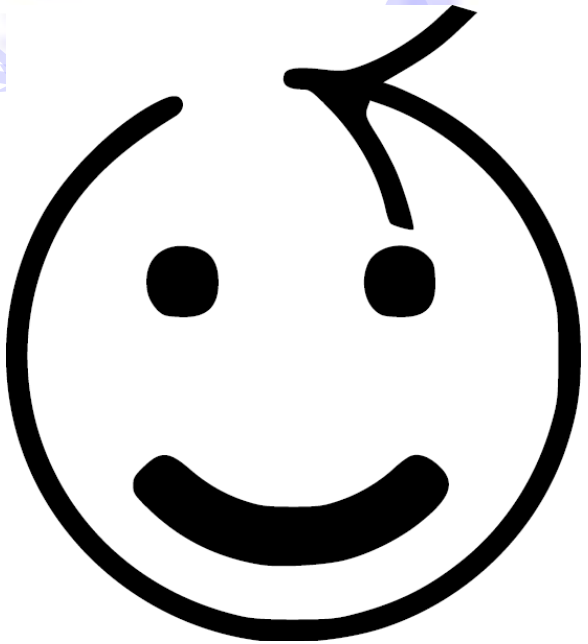
Relaxed Dependency Analysis

～Haskell 2009年末の集い～

酒井 政裕

自己紹介

酒井 政裕

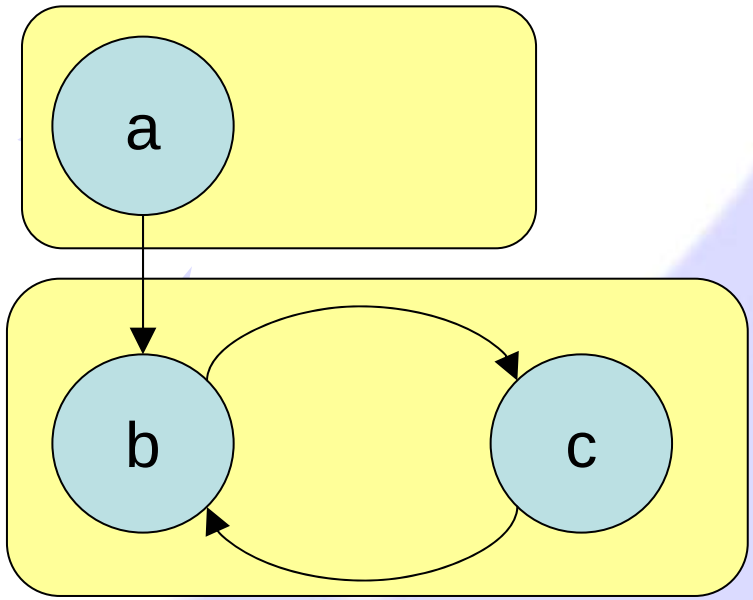


- Blog: ヒビルテ
- Twitter: @masahiro_sakai

- 2001年末ごろにHaskellに遭遇
- 好きな関数: unfoldr

背景: Haskellの型推論の流れ

- 依存関係(の強連結成分)で定義をグループ化
- 末端のグループから順に型推論
- 例: 以下では b, c の型をまず推論し、その結果を使って a の型を推論
  - a = … b …
  - b = … c …
  - c = … b …



Relaxed Dependency Analysis

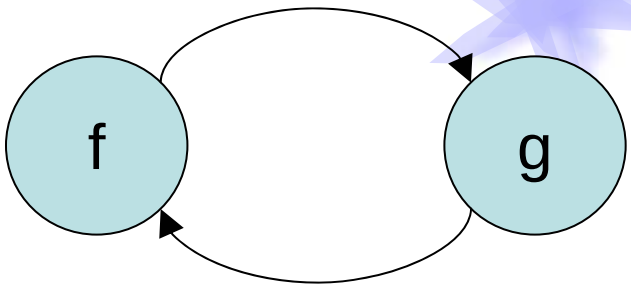
- 依存関係の定義の変更
  - Haskell98では、aがbを参照していれば、aはbに依存していた
  - Haskell2010では、bに型宣言がある場合には、aがbを参照していても、依存とはみなさない
- 効果
  - より多相的な型を推論出来るようになる
  - これまで型宣言が必要だったのが、不要になることがある

例: Haskell98の場合

```
f :: Eq a ⇒ a → Bool
f x = (x == x)
      ∨ g True
      ∨ g “Yes”

g y = (y ≤ y) ∨ f True
```

- fとgは相互に依存しているので、同時に型推論



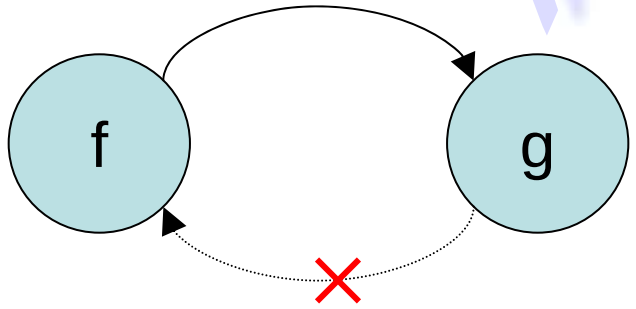
- g が二通りの型で使われている
  - Bool → Bool
  - String → Bool
- ⇒ 型エラー

例: Haskell2010の場合

```
f :: Eq a ⇒ a → Bool
f x = (x == x)
      ∨ g True
      ∨ g “Yes”

g y = (y ≤ y) ∨ f True
```

- fには型宣言があるので、gはfには依存しない



- gを先に型推論して、g :: ∀a. Ord a ⇒ a → Bool
- fの型推論では、gの型変数aをBoolとStringに具体化

何が起こっているのか?

- Haskellの基づいているHindley-Milnerのアルゴリズムでは、ある定義グループの型推論の最後の段階で型を多相的にする。

- なので、先に型推論が済んでいる部分は、型が多相的な型になっているので、それを任意に具体化出来る。
- 一方、型推論中の部分は、型が多相的になっていないので、複数の型で使うことが出来ない。