

そろそろ線形型を かじっておくか

2019/11/25

びしょ〜じょ

本日の内容

(Haskellの)線形型
を
知った気になる

自己紹介



こんにちは、びしょ〜じょです

- 筑波大学大学院 M2
型とか関数型言語の研究室でプログラム変換してる
- 株式会社HERP エンジニア
つくばオフィスでTSとかたまにHaskellを書いている

We're hiring!

線形型って急になんですか

- リソースを必ず1回だけ使うという性質を
型で表すすごい奴なんです

線形型って急になんですか

- リソースを必ず1回だけ使うという性質を型で表すすごい奴なんですが
- GHCの新たな拡張として議論されている



線形型

リソースを必ず1回だけ使うという性質を**型で表す**
すごい奴がいると...

- プログラマがハッピー
 - より安全なコードが書ける
- コンパイラがハッピー
 - 情報が増えて効率的なコード生成

Haskellの型システムの拡張



型システムの拡張は容易ではない!

- 既存のコードが壊れない安全な拡張ですか?



型システムにやってほしいことが

証明されてれば安全そう

- ✓ 型の付いたプログラムはランタイムエラーしない

Haskellの型システムの拡張



型システムの拡張は容易ではない!

- 既存のコードが壊れない安全な拡張ですか?



型システムにやってほしいことが

証明されてれば安全そう

- ✓ 型の付いたプログラムはランタイムエラーしない



論文を書いてしっかり型システムの証明もしている

Linear Haskell

Practical Linearity in a Higher-Order Polymorphic Language

$$\lambda_{\rightarrow}^q$$

- **Linear Haskell** (Haskell2010 + Linear types extension)
のcore calculus
- STLC + α + **multiplicity**
引数を使える回数に指定がある関数型

λ_{\rightarrow}^q

- **Linear Haskell** (Haskell2010 + Linear types extension)
のcore calculus
- STLC + α + **multiplicity**
引数を使える回数に指定がある関数型

$$\lambda_{\pi} (x : A) . e : A \rightarrow_{\pi} B$$

λ_{\rightarrow}^q

- **Linear Haskell** (Haskell2010 + Linear types extension)

のcore calculus

- STLC + α + **multiplicity**

引数を使える回数に指定がある関数型

$$\lambda_{\pi}(x : A) . e : A \rightarrow_{\pi} B$$

eの型付けにxを π 回過不足無く使うべし



今回はdata constructorとmultiplicity abstractionは省略します

λ_{\rightarrow}^q

multiplicity

$$\pi, \mu ::= 1 \mid \omega \mid p \mid \pi + \mu \mid \pi \cdot \mu$$

- $(+)$ と (\cdot) はcommutative, associative
- (\cdot) は $(+)$ に対してdistributive
- 1 は (\cdot) の単位元
- $\omega \cdot \omega = \omega$ (任意回)
- $1 + 1 = 1 + \omega = \omega + \omega = \omega$

λ_{\rightarrow}^q

型環境の操作(1) context addition

(multiplicityの(+))とは異なるので注意)

$$\begin{aligned}(x :_{\pi} A, \Gamma) + (x :_{\mu} A, \Delta) &= (x :_{\pi+\mu} A), (\Gamma + \Delta) \\(x :_{\pi} A, \Gamma) + \Delta &= (x :_{\pi} A), (\Gamma + \Delta) && (x \notin \Delta) \\() + \Delta &= \Delta\end{aligned}$$

λ_{\rightarrow}^q

型環境の操作(1) context addition

(multiplicityの(+))とは異なるので注意)

$$\begin{aligned}(x :_{\pi} A, \Gamma) + (x :_{\mu} A, \Delta) &= (x :_{\pi+\mu} A), (\Gamma + \Delta) \\(x :_{\pi} A, \Gamma) + \Delta &= (x :_{\pi} A), (\Gamma + \Delta) \quad (x \notin \Delta) \\() + \Delta &= \Delta\end{aligned}$$

λ_{\rightarrow}^q

型環境の操作(2) context scaling

型環境を π でスケーリングする

$$\pi (x :_{\mu} A, \Gamma) = (x :_{\pi\mu} A) , \pi\Gamma$$

λ_{\rightarrow}^q

型環境の操作(2) context scaling

型環境を π でスケーリングする

$$\pi(x :_{\mu} A, \Gamma) = (x :_{\pi\mu} A), \pi\Gamma$$

λ_{\rightarrow}^q 型付け

練習問題: var

$$\overline{\omega\Gamma + (x :_1 A) \vdash x : A}$$

λ_{\rightarrow}^q 型付け

練習問題: var

$$\overline{\omega\Gamma + (x :_1 A) \vdash x : A}$$

- context scalingにより $\omega\Gamma$ に1回しか使えない型は無い
- ところで、 $(x :_\omega A) + (x :_1 A) = x :_\omega A$ より
 $\omega\Gamma$ に x は含まれうる

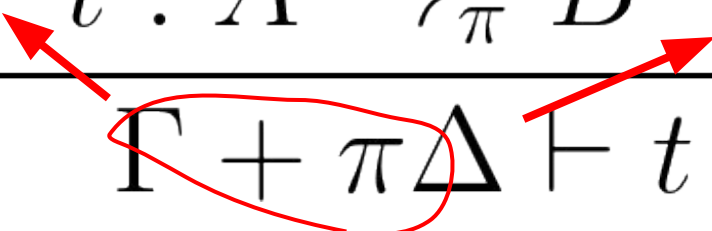
λ_{\rightarrow}^q 型付け

練習問題: application

$$\frac{\Gamma \vdash t : A \rightarrow_{\pi} B \quad \Delta \vdash u : A}{\Gamma + \pi \Delta \vdash t \ u : B}$$

λ_{\rightarrow}^q 型付け

練習問題: application

$$\frac{\Gamma \vdash t : A \rightarrow_{\pi} B \quad \Delta \vdash u : A}{\Gamma + \pi \Delta \vdash t u : B}$$


- context additionを**分解する方向**で使う(下から上に見る)

λ_{\rightarrow}^q 型付け

練習問題: application

$$\frac{\Gamma \vdash t : A \rightarrow_{\pi} B \quad \Delta \vdash u : A}{\Gamma + \pi \Delta \vdash t u : B}$$

- context additionを分解する方向で使う(下から上に見る)
- Δ を π で制約付ける
tの引数は π 回しか使えないので、 π 回使える変数の型環境 $\pi\Delta$ からのみ変数を参照する

線形型とパフォーマンス

"1回しか使わない" が型より明らかだと...

- 外部データの効率的な読み取り
1回しか使わないならメモリに展開しなくてよい
- **継続**を効率的なコードに変換
 - スタックセグメントのコピーが不要
 - 継続の実行がポインタへのジャンプ
 - 継続を使いまくる(`((>>=))`)Haskellには非常に朗報
- インライン化の補助
 - コードサイズ爆発を見切りやすい

線形型よもやま - via arrows vs via kinds

- Linear Haskellは**linearity via arrows**
- 他の選択肢: **linearity via kinds**

線形型よもやま - via arrows vs via kinds

- Linear Haskellは**linearity via arrows**
- 他の選択肢: **linearity via kinds**

お手元のATTaPL1章を参照ください

$$\begin{aligned}(\textit{type}) \tau &::= q \ p \\(\textit{prime type}) p &::= atom \mid \tau \rightarrow \tau \\(\textit{qualifier}) q &::= \text{un} \mid \text{lin}\end{aligned}$$

- **pros** 1回しか使えない値を表現しやすい
linear arrowの場合は継続を使う $(A \multimap r) \rightarrow r$
- **cons** 互換性を保つ拡張ができない
既存の型を任意回使える型に変換する必要

まとめ

- Haskellに線形型入るっぽいぞ
- 線形型面白いですね
- コンパイラもコード生成にやりがいを感じられる

Haskellほとんど関係なくてワロタ

參考資料

- Jean-Philippe Bernardy, Mathieu Boespflug, Ryan R. Newton, Simon Peyton Jones, and Arnaud Spiwack. "*Linear Haskell*."
- Carl Bruggeman, Oscar Waddell, and R. Kent Dybvig. "*Representing Control in the Presence of One-Shot Continuations*."