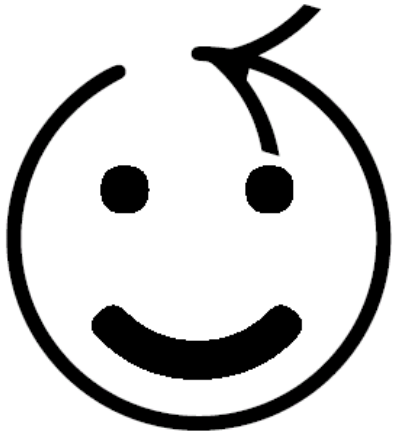
PLDIr#6 (PLDI 2002)



2010-02-11

酒井 政裕

Adoption and Focus: Practical Linear Types for Imperative Programming

Manuel Fähndrich Robert DeLine
Microsoft Research
One Microsoft Way
Redmond, WA 98052-6399

{maf,rdeline}@microsoft.com

Citation Count: 52

背景

- Type-state とか Vault とかみたいな、状態をトラッキングする型システムはエイリアシングと相性が悪い
  - e.g. close(a); read(b) は a と b が同じものを指していたら不正
- なので、状態をトラッキングする型では、エリアシングを禁止 (線形型)
- 線形型をフィールドに持つような型も線形型にしなくてはいけない
  - でないと、 close(a.f); read(b.f) が安全にならない
- が、これはデータ構造の設計への強い制約になってしまって、現実的ではない

- ⇒ この問題を解決する型システムを考えました!

概要

- Adoption/focus という概念を導入することで、非線形な型が線形なフィールドを持つことを許し、かつアクセスの安全性を保証する。

- adoption construct
  - 状態をトラッキングする型や、それらを参照する型の、エイリアスを安全に許す仕組み
- focus construct
  - エイリアスされたオブジェクトに対して、状態のトラッキングが出来る、一時的なスコープを導入する仕組み。

従来ダメだった例

非線形な型fileptrがトラッキングされる型(線形型)を参照

```
struct fileptr { tracked(@open) file f; }

void reset_logs($G:fileptr msgs, $G:fileptr errs) {
  tracked fileptr log = msgs;
  close(log.f);
  log.f = open(MSG_LOG_FILENAME);

  log = errs;
  close(log.f);
  log.f = open(ERR_LOG_FILENAME);
}
```

msgsとerrsがエイリアスしている可能性

解決方法

fileptrは、フィールドを初期化するために線形型で確保、adoptionで非線形な型に変換

```
struct fileptr { tracked(@open) file f; }

void reset_logs($G:fileptr msgs, $G:fileptr errs) {
  tracked fileptr log = msgs;
  close(log.f);
  log.f = open(MSG_LOG_FILENAME);

  log = errs;
  close(log.f)
  log.f = ope
}
```

- focusを推論
- focus内では
  - logは線形型を持つ
  - msgsのエイリアスの可能性があるerrsへのアクセスは禁止 (ガード$Gで判断)

ヒープモデル

- 従来モデルでは、ヒープオブジェクトは線形・非線形に分類されていた
- このモデルでは、
  - すべてのオブジェクトは線形な状態で確保・破棄される
  - Adoptionで、線形な参照を消費して、非線形なエイリアスを生成
  - Focusで、非線形なエイリアスから一時的に線形な参照を取得して、操作

型の扱い方

- トラッキングされている型は tr(ρ)
  - 属するのはそのオブジェクトただ一つだけ
  - ρは静的に決まる名前でcapabilityのキー
- capabilityのエントリ {ρ ↦ τ}
  - そのプログラムポイントで、ρでトラッキングされるオブジェクトが生きていて、型τでアクセス可能
- Guarded type ρ▷τ
  - ρがcapabilityに含まれるときに、(focusを使って)型τとしてアクセス可能。
  - 自由にエイリアス可能

Type language

heap type	h	$::=$	$\langle \sigma_1 .. \sigma_n \rangle \mid \tau[\,]$
type	τ	$::=$	$\mathtt{int} \mid \mathtt{tr}(\rho) \mid G \triangleright h$
		$\mid$	$\forall[\Delta].(C_1, \sigma_1) \rightarrow (C_2, \sigma_2)$
linear type	σ	$::=$	$\exists[\rho \mid \{\rho \mapsto h\}].\, \mathtt{tr}(\rho) \mid \tau$
guard	G	$::=$	ρ
capability	C	$::=$	$\cdot \mid \{\rho \mapsto h\} \otimes C \mid \epsilon \otimes C$
type context	Δ	$::=$	$\cdot \mid \rho, \Delta \mid \epsilon, \Delta$

Figure 2: Type language

Expressions

$$
\begin{array}{lll}
e & ::= & x \mid i \mid e.i \mid e.i := e \mid e(e) \mid e[c] \\
& \mid & \texttt{new}\langle i \rangle \mid \texttt{free}\, e \mid \texttt{adopt}\, e{:}h\, \texttt{by}\, e \\
& \mid & \texttt{let}\, x = e\, \texttt{in}\, e \\
& \mid & \texttt{let}\, x = \texttt{focus}\, e\, \texttt{in}\, e \\
& \mid & \texttt{fun}\ f[\Delta](x : \sigma) : \sigma\, \texttt{pre}\ C\, \texttt{post}\ C\ \{e\} \\
\\
c & ::= & \rho \mid C \mid G
\end{array}
$$

Figure 3: Expressions

Adoption (養子縁組)

- **adopt** e1 **by** e2
  - e1の線形な参照を消費して、e2からe1への内部的な参照を作成
  - e1が養子、e2が養親
  - 結果はe1への非線形な参照
- 型は、e1：τ, e2：tr(ρ) のとき、**adopt** e1 **by** e2：ρ▷τ

Adoption

$$\frac{\begin{array}{l}\Delta;\Gamma;C \vdash e_1 : \mathtt{tr}(\rho_1);C_1 \\ \Delta;\Gamma;C_1 \vdash e_2 : \mathtt{tr}(\rho_a);\{\rho_1 \mapsto h\}\otimes C_2 \\ \Delta \vdash C_2 \leq \{\rho_a\}\end{array}}{\Delta;\Gamma;C \vdash \mathtt{adopt}\ e_1{:}h\ \mathtt{by}\ e_2 : \rho_a \triangleright h;C_2}$$

Focus

- **let** x = **focus** e1 **in** e2
- e1 が ρ1▷τ であって、capabilityにρ1が含まれているときに、e2を以下の文脈で評価:
  - フレッシュな名前ρを使って、x : tr(ρ) とおいて、capabilityに {ρ ↦ τ} を追加し ρ1 を削除。
  - e2の実行後のcapabilityも {ρ ↦ τ} を含まなくてはならない。

Focus

$$\frac{\Delta;\Gamma;C \vdash e_1 : G \triangleright h; C_1 \otimes C_2 \quad \Delta \vdash C_1 \leq G \quad \rho \text{ fresh} \quad \Delta;\Gamma[x : \mathtt{tr}(\rho)]; C_2 \otimes \{\rho \mapsto h\} \vdash e_2 : \tau_2; C_3 \otimes \{\rho \mapsto h\}}{\Delta;\Gamma;C \vdash \mathtt{let}\, x = \mathtt{focus}\, e_1\, \mathtt{in}\, e_2 : \tau_2; C_1 \otimes C_3}$$

Motivating Example

```
fun dict_lookup(phone:int, ssn:int, d:dictionary) {
... let c = newCell(d) in ...
}
fun add_amount(cell:ref<int[]>, elem:int) {
... resize(cell, newsize) ...
}
fun add_entry(d:dictionary, phone:int, ssn:int,
    amount:int) {
  let cell = dict_lookup(phone, ssn, d)
  in add_amount(cell, amount)
}
```

Motivating Example (2)

- 二種類のインデックスでひけて、値が配列である辞書。
- 配列はサイズを変更するので、ダングリングポインタがないことを保証するために、線形型にしたい。
- しかし、それを持つrefは二種類のインデックスの間でエイリアスされる。

newCell

fun newCell(dct : tr(ρ_d)) : $\rho_d \triangleright \langle \text{int}[]^\bullet \rangle$
  pre {$\rho_d \mapsto$ dictionary}
  post {$\rho_d \mapsto$ dictionary}
{
  let cell : tr(ρ) = **new**<1> **in**
    cell.1 := **newarray**(10);
    adopt cell : $\rho_d \triangleright \langle \text{int}[]^\bullet \rangle$ **by** dct
}

$h^\bullet := \exists[\rho|\{\rho \mapsto h\}].\ \text{tr}(\rho)$

文脈の変化

- **let** cell : tr(ρ) = **new**<1> **in** の実行後:
  - $\{\rho_d \mapsto \text{dictionary}\} \otimes \{\ \rho \mapsto \ \langle \text{int} \rangle\ \}$
  - dct : $\text{tr}(\rho_d)$, cell : $\text{tr}(\rho)$

- cell.1 := **newarray**(10); の実行後:
  - $\{\rho_d \mapsto \text{dictionary}\} \otimes \{\ \rho \mapsto \ \langle \text{tr}(\rho 2) \rangle\ \} \otimes \{\ \rho 2 \mapsto \text{int}[]\ \}$
  - dct : $\text{tr}(\rho_d)$, cell : $\text{tr}(\rho)$

文脈の変化

- cell.1 := **newarray**(10); の実行後:
  - { $\rho_d \mapsto$ dictionary } $\otimes$ { $\rho \mapsto$ 〈tr(ρ2)〉 } $\otimes$ { $\rho 2 \mapsto$ int[] }
  - dct : tr(ρ_d), cell : tr(ρ)
- Packして、
  - { $\rho_d \mapsto$ dictionary } $\otimes$ { $\rho \mapsto$ 〈int[]$^\bullet$ 〉 }
  - …
- **adopt** cell : $\rho_d \triangleright$ 〈int[]$^\bullet$〉 **by** dct で
  - { $\rho_d \mapsto$ dictionary } という capability と、
  - $\rho_d \triangleright$ 〈int[]$^\bullet$〉という型に

$h^\bullet := \exists[\rho|\{\rho \mapsto h\}].\ tr(\rho)$

resize

```
fun resize[ρd](cell : ρd ▷ ⟨int[]•⟩ , size:int): int
  pre {ρd ↦ dictionary}
  post {ρd ↦ dictionary}
{
  let newa = newarray(size) in
  let fcell = focus cell in
  let olda = fcell.1 in
    copy(olda,newa);
    fcell.1 := newa;
    free olda
}
```

文脈の変化

- Focus前:
  - { $\rho_d \mapsto$ dictionary } $\otimes$ { $\rho 1 \mapsto$ int[] }
  - cell : $\rho_d \triangleright$ $\langle$int[]•$\rangle$, newa : tr(ρ1), size : int
- Focus後:
  - { ρ1 $\mapsto$ int[] } $\otimes$ { $\rho \mapsto$ $\langle$int[]•$\rangle$ }
  - fcell : tr(ρ), newa : tr(ρ1), size : int
- Unpackして:
  - { ρ1 $\mapsto$ int[] } $\otimes$ { $\rho \mapsto$ $\langle$tr(ρ2)$\rangle$ } $\otimes$ { ρ2 $\mapsto$ int[] }
  - fcell : tr(ρ), newa : tr(ρ1), size : int

文脈の変化

- **let** olda = fcell.1 **in** の実行後:
  - { ρ1 ↦ int[] } ⊗ { ρ ↦ ⟨tr(ρ2)⟩ } ⊗ { ρ2 ↦ int[] }
  - fcell : tr(ρ), newa : tr(ρ1), olda : tr(ρ2), size: int
- fcell.1 := newa; の実行後:
  - {ρ1 ↦ int[]} ⊗ {ρ ↦ ⟨tr(ρ1)⟩ } ⊗ {ρ2 ↦ int[]}
- free(olda) の実行後:
  - {ρ1 ↦ int[]} ⊗ {ρ ↦ ⟨tr(ρ1)⟩ }

文脈の変化

- free(olda) の実行後:
  - { ρ1 ↦ int[] } ⊗ { ρ ↦ 〈tr(ρ1)〉 }
- Packして
  - { ρ ↦ 〈int[]• 〉 }
- Focusが終了して
  - { ρ_d ↦ dictionary } ⊗ { ρ ↦ 〈int[]•〉 }

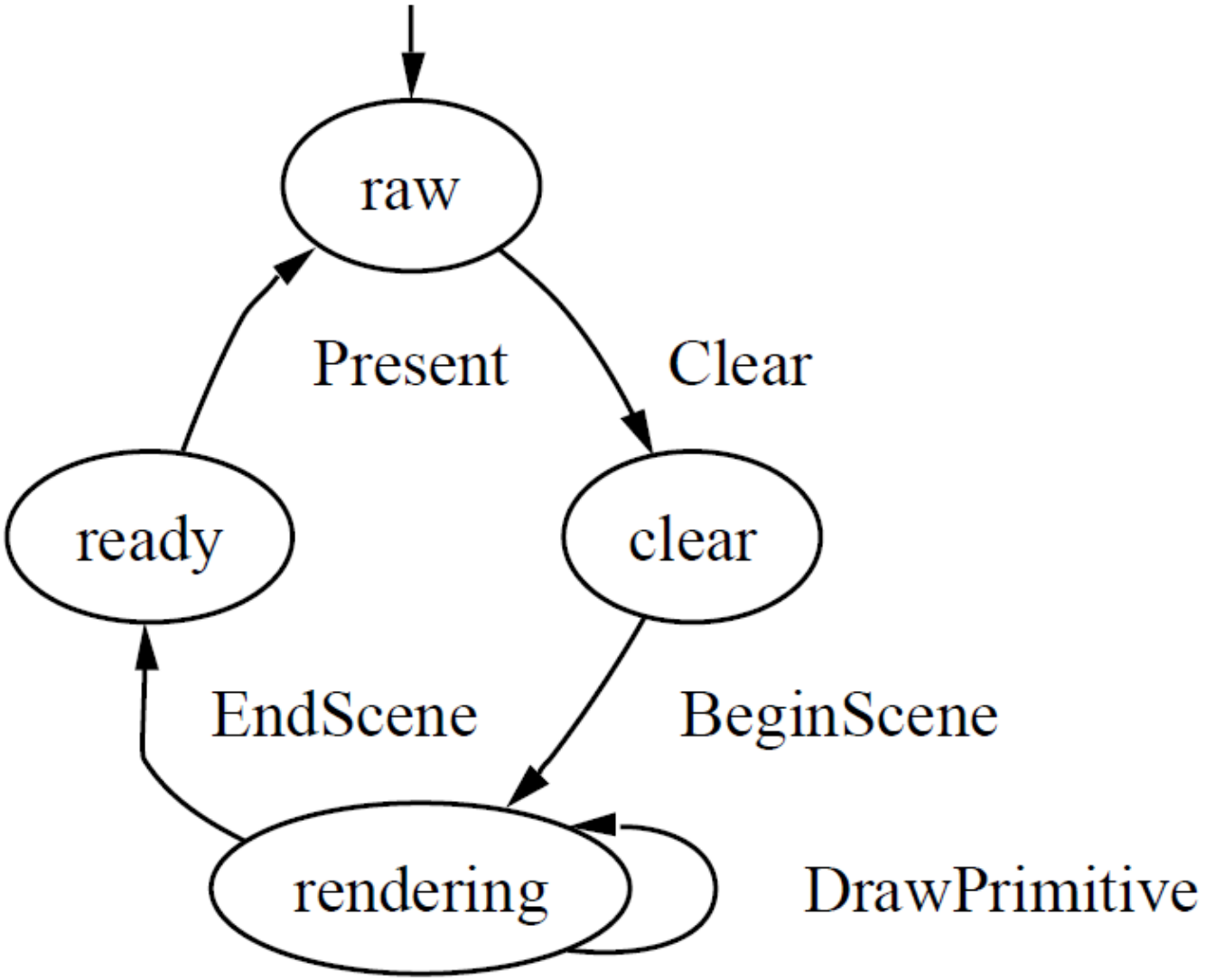
Vaultに実装

Syntax	Precondition	Postcondition
`new $K@s`	C	$C \otimes \{\texttt{\$K@s}\}$ (fresh `$K`)
`+$K@s`	C	$C \otimes \{\texttt{\$K@s}\}$
`-$K@s`	$C \otimes \{\texttt{\$K@s}\}$	C
`$K@s`	$C \otimes \{\texttt{\$K@s}\}$	$C \otimes \{\texttt{\$K@s}\}$
$\texttt{\$K@s}_1\texttt{->s}_2$	$C \otimes \{\texttt{\$K@s}_1\}$	$C \otimes \{\texttt{\$K@s}_2\}$

Direct3Dの例

```
interface VERTEX_BUFFER {
  type buffer;
  tracked($B) buffer CreateVertexBuffer () [new
   $B@raw];
  void Clear (tracked($B) buffer) [$B@raw->clear];
  void BeginScene (tracked($B) buffer) [$B@clear-
   >rendering];
  void DrawPrimitive ($B:buffer, ...) [$B@rendering];
  void EndScene (tracked($B) buffer) [$B@rendering-
   >ready];
  void Present (tracked($B) buffer) [$B@ready-
   >raw];
}
```

Direct3Dの例


raw
Present
Clear
ready
clear
EndScene
BeginScene
rendering
DrawPrimitive

MaJIC: Compiling MATLAB for Speed and Responsivenes

George Almási and David Padua
galmasi,padua@cs.uiuc.edu

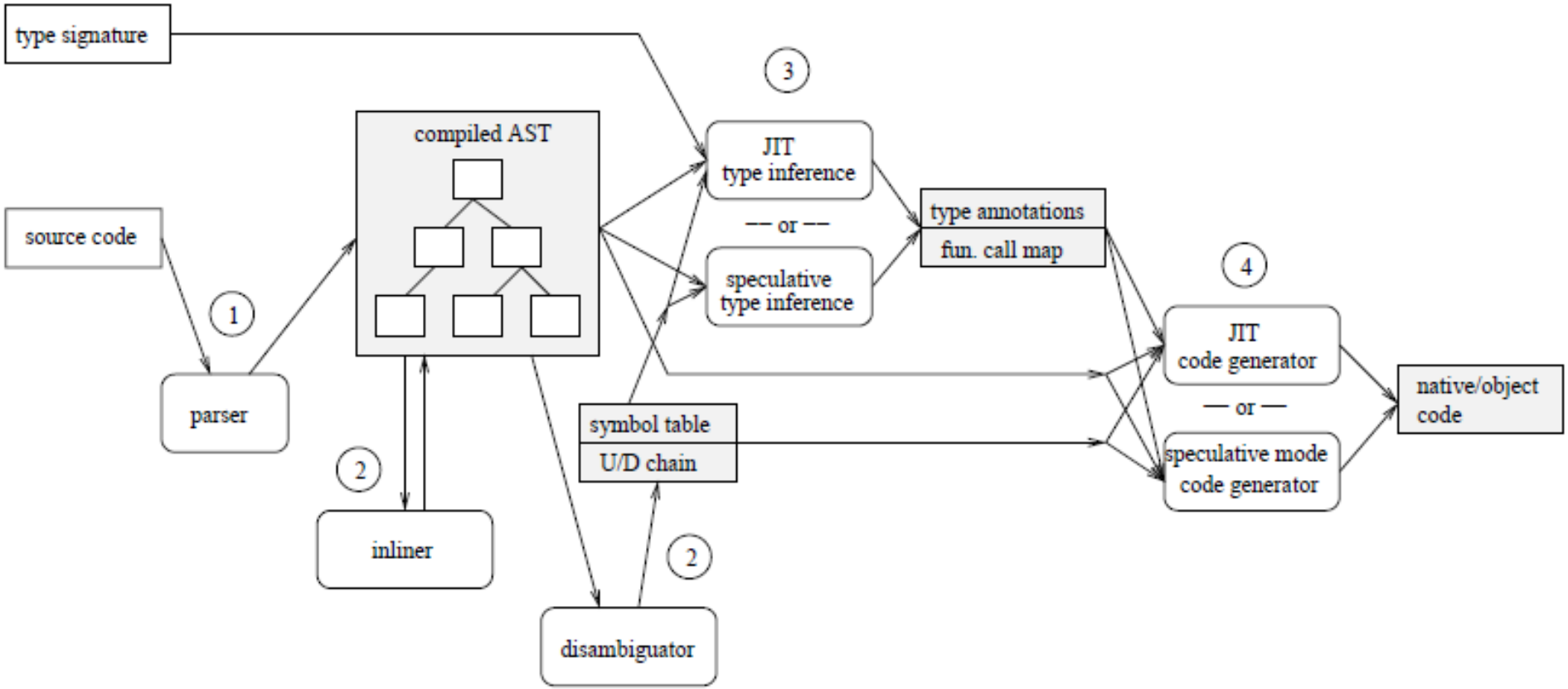
Department of Computer Science
University of Illinois at Urbana-Champaign

Citation Count: 8

概要

- MATLAB用の JIT/AOT コンパイラ
- MATLABは型無しの言語なので、型推論
  - 値の範囲、配列・行列・ベクトルのサイズも推論
  - JITでは引数と文脈から推論
  - AOTでは型を投機的に決定
    - 予測が外れたらJITの方にフォールバック

MaJIC compiler passes



Type speculation で使うヒント

- インデックス操作では、虚部は無視され、実部は丸められるので、普通は整数型しか用いられない
- 比較演算は、虚部を無視するし、ベクトルの比較も稀
- ブラケットの引数は、同じ行数もしくは同じ列数のベクトル。特に一個でもスカラーなら全部スカラー。
- Fortran由来のインデックス操作の構文なら、Fortranっぽい使い方をしているに違いない
- ビルトイン関数の引数 (特定の型以外は警告が出る)

インタプリタと比較しての性能向上