動的型推論を行う
漸進的型付け言語のコンパイル手法

大志万　優生（京都大学）　　五十嵐　淳（京都大学）

背景：動的型推論を伴う漸進的型付け言語

動的型推論

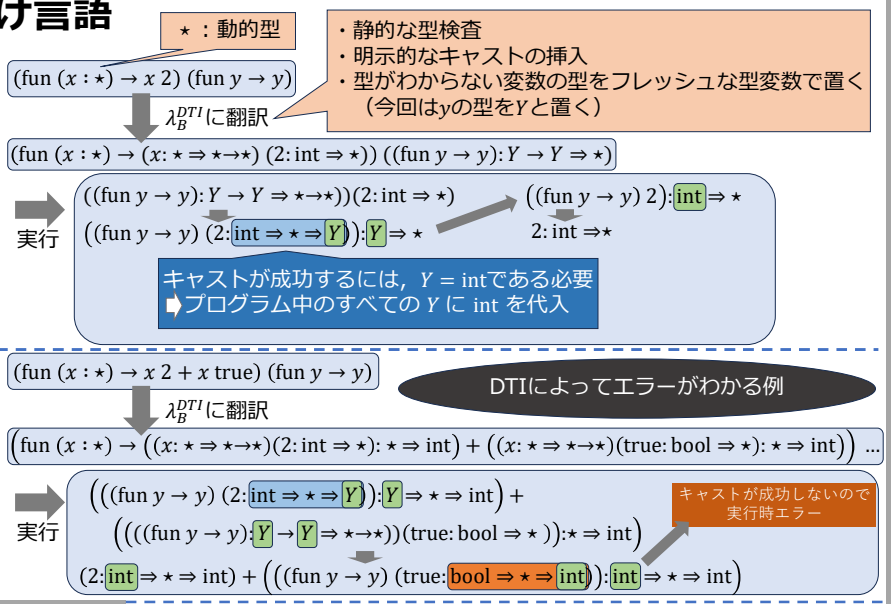
①型がわからない変数の型として**型変数**を置く
②プログラムの実行時に，型変数に**型を代入**
➧型変数で実行時に型の整合性をチェックできる

λ_B^{DTI} [Miyazaki et al. 2019]

動的型推論を伴う漸進的型付け言語を評価するための中間表現

Miyazakiらはλ_B^{DTI}のインタプリタを実装

・動的型以外の型注釈がない漸進的型付け言語のプログラムを実行可能
・標準的な算術・比較演算，関数などに加え，再帰関数，型アノテーション，let多相に対応



動機

・動的型推論を行う漸進的型付け言語のコンパイラがない
・コンパイラでどのように動的型推論を実装するかが知りたい

漸進的型付け言語：
◎ 静的型付けと動的型付けを一つの言語で同時に扱える
✕ 実行時に型を検査するオーバーヘッドが課題に

本研究：λ_B^{DTI}のプログラムをCプログラムへと変換するコンパイラを実装

①K正規化・クロージャ変換などを実装
・MinCaml[Sumii 2005]に基づいて実装
・型についてもネスト構造を除去
➧関数定義と同じようにプログラム中から取り出す

②クロージャ変換されたλ_B^{DTI}のプログラムをCプログラムに変換
・λ_B^{DTI}の型を表すC言語上の型tyを定義
・λ_B^{DTI}でのキャストの意味論を行うC上の関数castを用意
・cast関数内で動的型推論を実現

```c
value cast(value x, ty *t1, ty *t2, ran_pol r_p) {
  if (t1->tykind == DYN && t2->tykind == TYVAR) {
    switch(x.d->g) {
    case(G_INT):
      printf("DTI : int was inferred¥n");
      *t2 = tyint;
      retx = *x.d->v;
      break;
    ……
    }
  }
}
```

x内のGround typeを調べる
➧t2の指す先を**破壊的に**int型に変更
➧プログラム中のt2に相当する型がすべてint型になった
➧xに格納された値を返す

```c
typedef struct ty {
  enum tykind {
    DYN,
    BASE_INT,
    BASE_BOOL,
    BASE_UNIT,
    TYFUN,
    TYVAR,
  } tykind;
  struct tyfun {
    ty *left;
    ty *right;
  } tyfun;
} ty;
```

```c
typedef union value {
  int i_b_u;
  dyn *d;
  fun *f;
} value;
```

```c
typedef struct ran_pol {
   …
} ran_pol;

typedef enum ground_ty {
  G_INT,
  G_BOOL,
  G_UNIT,
  G_AR,
} ground_ty;

typedef struct dyn {
  value *v;
  ground_ty g;
  ran_pol r_p;
} dyn;
```

Cプログラムに必要な様々な型を定義

```
1  #include <gc.h>
2  #include "../lib/cast.h"
3
4  decl_name({T_1, …, T_n})
5
6  int set_tys() {
7    T_1, …, T_n
8    return 0;
9  }
10
11 decl_name({D_1, …, D_n})
12
13 D_1, …, D_n
14
15 int main() {
16   set_tys();
17   e
18   return 0;
19 }
```

出力されるCプログラムの概略図

1行目：
Garbage Collectionのために，
Boehm GC[Boehm et al. 1988]を利用

型や値の定義で動的なメモリ確保が出現

4~9行目：
$T_1, …, T_n$はプログラムから取り出した型
型名を宣言➧set_tys関数内で型を定義

型のポインタはヒープで管理

11~13行目：
$D_1, …, D_n$はプログラムから取り出した関数
関数名を宣言した後，関数を定義

17行目：
式本体

cast関数やcast.hで定義した型などを用いる

実装の状況

https://github.com/SoftwareFoundationGroupAtKyotoU/lambda-dti-compiler

✅ λ_B^{DTI}のコンパイラが完成
✅ let多相をサポート
⬜ さらなる最適化の実装と適用

今後の課題

コアーション[Herman et al. 2010][Kuhlenschmidt et al. 2019]という構造を使って，実行時のオーバーヘッドを少なくする．