内部DSLの有効範囲を制御するための言語機構の提案

市川 和央　千葉 滋
東京大学情報理工学系研究科

省略できるものは省略する

最近の流行

- 型推論 (Haskell, ML)
- implicit conversion (Scala, C#)
- 多数のシンタックスシュガー (Ruby, Scala)

DSL (ドメイン専用言語)

- 汎用性を犠牲に、本質以外の記述を可能な限り省略

```
select name from scores group by name having avg(score) > 80
```

内部DSL

DSLのように使えるライブラリ

- 本質的な記述だけでプログラムが書けるようデザイン
- ホスト言語との連携で汎用性も維持
- ホスト言語の文法による制限を受ける
  - 省略できない部分が存在

```
create.select(NAME).from(SCORES).groupBy(NAME)
                    .having(avg(SCORE).gt(80))
```

本研究がめざす方向

内部DSLの(表面的な)文法の自由度を向上させる

- ホスト言語の文法による制限を緩める
  - ユーザ定義の文法など
  - 内部構造を変えるわけではない
- ホスト言語との連携は維持
  - 一部だけ別のコンパイラにするといった解法はとらない

前身研究：ProteaJ (Java + protean operators)

ユーザ定義演算子によるDSLの文法の表現

- 演算子 = (引数部 | 名前部)*
  - 並び方や順序には制限がない
- 型を非終端記号のようにみなす
  - 引数及び返り値の型によるオーバーロード

演算子 _ ">" _

```
select name from scores group by name having avg(score) > 80
```

演算子 "select" _ "from" _ "group" "by" _ "having" _

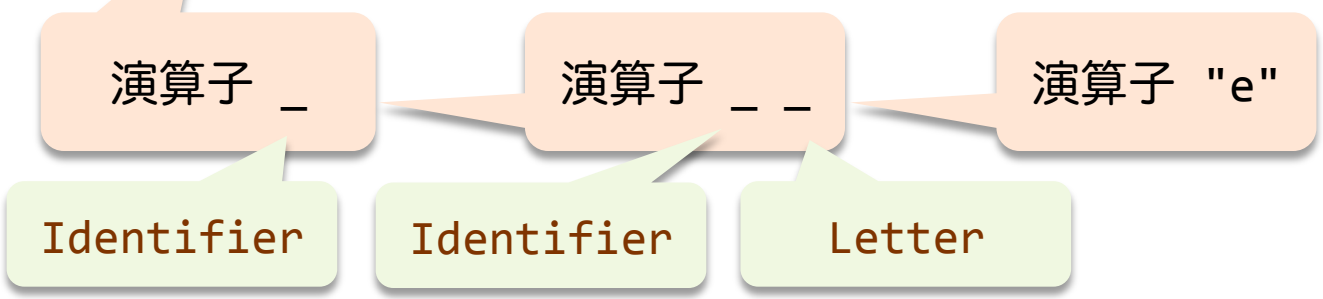
Column　Table　Column　Condition

ProteaJ でできること

式部分で任意の文法のDSLを定義できる

- 表現力 = Parsing Expression Grammar (PEG)
- 大抵の言語の文法は表現できる
- ユーザ定義リテラルも表現可能

```
select name from scores group by name having avg(score) > 80
```



ProteaJ でできなかったこと

式を超えた構造は表現できない

- ブロック構造、制御構造
  - あくまでも演算子なので、文は表現できない
  - ProteaJ は遅延評価をサポートしているため、ある程度は可能
- クラス・メソッド定義
  - 本質的にメタプログラミングが必要

本研究の目標

ProteaJ を拡張して、
式を超えた構造を持つ内部DSLを実現

研究室ミーティングにて

```
ResultSet rs = select name from scores group by name
                                having avg(score) > 80;
while (rs.next) praise(rs[name]:str);
```

こんなふうに書ける内部DSLが作れます！

DBのコネクションはどうしてるの？

静的変数にしてますけど・・・

それじゃ複数あったら駄目じゃん

Iさん

従来の実現手法

DSLの内部状態をユーザ変数を介してやりとり

- 直観的には、省略できそう
- 内部状態はできるだけ隠すべき (カプセル化)

```
ResultSet rs = select name from scores group by name
                              having avg(score) > 80        ;
while (rs.next) praise(rs[name]:str);
```

従来の実現手法

DSLの内部状態をユーザ変数を介してやりとり

- 直観的には、省略できそう
- 内部状態はできるだけ隠すべき (カプセル化)

```
Connection con = connect jdbc:postgresql:database          ;
ResultSet rs = select name from scores group by name
                              having avg(score) > 80 by con;
while (rs.next) praise(rs[name]:str);
close con;
```

従来の実現手法

DSLの内部状態をユーザ変数を介してやりとり

- 直観的には、省略できそう
- 内部状態はできるだけ隠すべき (カプセル化)

"文脈"から明らか

```
Driver driver = org.postgresql.Driver;
Connection con = connect jdbc:postgresql:database by driver;
ResultSet rs = select name from scores group by name
                              having avg(score) > 80 by con;
while (rs.next) praise(rs[name]:str);
close con;
```

提案："文脈"を表す言語機構

文脈情報: プログラムの上から下に伝播する情報

- 内部DSLの式により暗黙的に追加・削除される
- 文脈により使える文法が変化する
- 文脈情報を表すオブジェクト = 文脈オブジェクト

```
use org.postgresql.Driver;
connect jdbc:postgresql:database;
select name from scores group by name having avg(score) > 80;
while (each result) praise(name:str);
disconnect;
```

Connection を文脈に追加

Connection が文脈に含まれるため利用可能

文脈から Connection を削除

工夫：情報は動的だが文法は静的

文脈オブジェクトの追加は動的

- Connection は動的な情報

文脈によって静的な文法が変化

- どんな文脈なのかコンパイラが知る必要

```
void "connect" _ (DB db)
                          {
  activate new Connection(...);
}
```

```
use org.postgresql.Driver;
connect jdbc:postgresql:database;
select name from scores group by name having avg(score) > 80;
while (each result) praise(name:str);
disconnect;
```

制約：静的に文脈の型が決まる

動的に追加・利用・削除する文脈の型を指定

- throws 節のようなもの

文脈オブジェクトの型が静的に一意に決まる必要

- どのパスを通っても同じでなければならない
- 文脈は静的に伝播

```
void "connect" _ (DB db)
  activates Connection {
  activate new Connection(...);
}
```

```
use org.postgresql.Driver;
connect jdbc:postgresql:database;
select name from scores group by name having avg(score) > 80;
while (each result) praise(name:str);
disconnect;
```

文脈クラス

コンストラクタ・フィールド・演算子を持つ

- フィールド = ある文脈における内部DSLの内部状態
- 演算子 = この文脈でのみ使える内部DSLの文法

文脈クラスのインスタンス = 文脈オブジェクト

- 普通のクラスと同様に new でインスタンス化

```
context Connecting {
  Connecting (DB db, Driver driver) {
    con = driver.connect(db.path, new Properties());
  }
  void "select" _ "from" _ "group" "by" _ "having" _
    (Column s, Table f, Column g, Condition h)
    { con.executeQuery(...); }
  private Connection con;
}
```

コンストラクタ

演算子

フィールド

文脈オブジェクトの追加・削除

activates 節・deactivates 節

- 追加・削除する文脈オブジェクトの型を指定
- 演算子の呼び出し側の文脈に文脈オブジェクトを追加

activate 文

- 実際に追加する文脈オブジェクトを指定する

```
// Connecting 文脈クラス外部、driver を持つ文脈クラス内部
void "connect" _ (DB db) activates Connecting
  { activate new Connecting(db, driver); }

// Connecting 文脈クラス内部
void "disconnect" () deactivates Connecting
  { con.close(); }
```

文脈の利用

requires 節

- 利用する文脈オブジェクトの型を指定
- その型の文脈オブジェクトを含む文脈でのみ利用可能
- ボディはその型の文脈オブジェクトを含む文脈で評価

```
void selectStudents () requires Connecting {
  select name from scores group by name having avg(score) > 80;
}

use org.postgresql.Driver;
connect jdbc:postgresql:database;
selectStudents();
```

DSLモジュール

文脈によらない内部DSLを定義するモジュール

- 文脈オブジェクトを最初に追加する演算子を定義
- import dsl でインポートして利用

```
dsl SQL {
  void "use" _ (Driver d) activates UseDriver {
    activate new UseDriver(d);
  }
}
```

発展的機能

autoclose・propagate 修飾子

- 文脈クラスにつける
- ホスト言語のスコープを抜けた時の動作を指定
- autoclose: 文脈オブジェクトを削除
- propagate: 外側のスコープに伝播
- autoclose の場合は削除時の後処理を指定できる
- propagate でメソッドの外へ伝播する場合、activates 節が必要

関連研究

deep embedding

- 一連の処理を表現するオブジェクトを作り、評価器に渡す手法
- 内部DSLをホスト言語と明確に区別

implicit parameter

- 自明な引数を省略、静的なスコープに登録された値を利用
- 文脈ごとに異なる値を使いたければユーザが登録する必要

import

- インポートしたメソッドをレシーバ無しで記述
- ユーザが明示的にインポート文を記述する

関連研究

SugarJ [Erdwegら, OOPSLA'11]

- シンタックスシュガーをライブラリにできる言語
- 文脈に応じて文法を変えることはできない

Wyvern [Omarら, ECOOP'14]

- ユーザ定義リテラルを実現できる言語
- generic literal と呼ばれる特別な構文が必要

Context-oriented programming

- 文脈によってメソッドディスパッチを変化させる
- 文法を変化させるようなものではない

まとめと今後の課題

文脈を表す言語機構を提案

- 複数の文からなるDSLの状態を隠蔽
- 文脈により文法が変化
- 文脈の型が静的に一意に決まらなければならない

今後の課題：コンパイラの実装および評価

- 現在鋭意実装中
- 評価方法は検討中

動的な情報、静的な伝播

文脈オブジェクト自体は動的

- DSLの内部状態を表現

文脈オブジェクトの伝播は静的な範囲に限定

- 静的な解析を可能とするため

```
void praiseGoodStudents () {
  use org.postgresql.Driver;
  connect jdbc:postgresql:database;
  selectGoodStudents();
  while (each result) praise(name:str);
  disconnect;
}
void selectGoodStudents () {
  select name from scores group by name having avg(score) > 80;
}
```

呼び出し先には文脈は伝播しない

Connection が文脈にないためエラー