2.4. 表示的意味論

表示的意味論

プログラムを
ネイティブ言語(この場合 SIMPLE)から
別の表現に変換すること
に関心

英語の walk をフランス語の marcher に変化するがごとく

元々は数学的オブジェクトに変換するためにつかうもの

SIMPLE という言語を Ruby という言語(※)で
記述することで「できること(できないこと)」を
表現している

※ ここでは誰もが
完全に知っている言語という扱い

2.4.1. 式

ルール

環境は e で渡す

実行は proc 内で行う

もっとシンプルに実装可能でも、
proc および e を使う

元の言語の構成要素の特性を
うまく表現するため

たとえ一部の式が環境を使わなくても、
一般的に式は環境を必要とするため。

実例

ほぼ直接書くもの

Number

```
class Number
  def to_ruby
    "-> e { #{value.inspect} }"
  end
end
```

Boolean

```
class Boolean
  def to_ruby
    "-> e { #{value.inspect} }"
  end
end
```

変数を展開するもの

Variable

```
class Variable
  def to_ruby
    "-> e { e[#{name.inspect}] }"
  end
end
```

合成

Add

```
class Add
  def to_ruby
    "-> e { (#{left.to_ruby}).call(e) + (#{right.to_ruby}).call(e)}"
  end
end
```

Multiply

```
class Multiply
  def to_ruby
    "-> e { (#{left.to_ruby}).call(e) * (#{right.to_ruby}).call(e)}"
  end
end
```

LessThan

```
class LessThan
  def to_ruby
    "-> e { (#{left.to_ruby}).call(e) < (#{right.to_ruby}).call(e)}"
  end
end
```

2.4.2. 文

Assign

解説

操作的意味論では値ではなく新しい環境を生成していた

表示的意味論では環境のハッシュを更新する

```
class Assign
  def to_ruby
    "-> e { e.merge({ #{name.inspect} => (#{expression.to_ruby}).call(e)} ) }"
  end
end
```

DoNothing

```
class DoNothing
  def to_ruby
    '-> e { e }'
  end
end
```

If

```
class If
  def to_ruby
    "-> e { if (#{condition.to_ruby}).call(e) " +
      " then (#{consequence.to_ruby}).call(e) " +
      " else (#{alternative.to_ruby}).call(e) " +
      " end } "
  end
end
```

Sequence

解説

1番目の文の評価結果は、
2番目の文を評価するための
環境として使われる

```
class Sequence
  def to_ruby
    "-> e { (#{second.to_ruby}).call((#{first.to_ruby}).call(e))}"
  end
end
```

While

```
class While
  def to_ruby
    "-> e {" +
      " while (#{condition.to_ruby}).call(e); e = (#{body.to_ruby}).call(e); end; " +
      " e " +
      " } "
  end
end
```

コラム

意味論のスタイルの比較

While 実装に現れている

スモールステップ意味論

抽象機械のための簡約規則

ビッグステップ意味論

最終的な環境を
直接計算する方法を示した評価規則として
書かれている。

表示的意味論

Ruby によって書き直す方法。

すなわち Ruby の While を使って
書き直すことを示している。

2.4.3. 応用

表示的意味論は環境を?、
具体的な Ruby オブジェクトとして表現することで、
その意味を明確にしている。

このような表示的定義が
現場で使われた例

過去の Scheme 標準

http://www.schemers.org/Documents/Standards/R5RS/HTML/r5rs-Z-H-10.html#%25_sec_7.2

XSLT ドキュメント変換言語

XSLT パターン

XPath 式

表示的に意味論を使って
正規表現の意味論を記述する例

「3.3.2. 意味論」

楽しみ