プログラミング言語処理系のための抽象型による対応関係の実現

2009SE190 村津 貴大
指導教員: 横山 哲郎

1 はじめに

プログラミング言語処理系の実現において, 環境や記憶域を始めとして対応関係で表現すると自然なデータ構造はたくさんある. したがって, プログラミング言語処理系の実現を行うためには対応関係の効果的な実現が必要である. 対応関係の実現方法のひとつとして抽象型の利用が考えられる.

本研究の目的は, 対応関係を抽象型を用いて実現することと, その実現を通して抽象型の理解を深めることである. 具体的には, 文献[1]で抽象型の理解を深めて, プログラミング言語処理系の実現に必要な対応関係を抽象型で実現する例をふたつ挙げて, 抽象型の利点およびそれぞれの実現の利点と欠点を確かめる.

2 抽象型による対応関係の実現

抽象型とは, 問題に応じて用意されるデータに対して, 必要な機能を実現する関数を用いてその性質を規定するデータ型である.

抽象型のメリットは, 必要な機能を実現する関数だけを用いて表現しているため, データの実現法とそれらの関数を変更しても, そのデータを用いるプログラムには影響がでないことである.

本節では対応関係を抽象型で表してその実現を2つ示す. 抽象型を規定するためにはその代数的仕様を定める必要がある. 一般には, 抽象型の実現には複数の実現がある. それぞれの具体的な実現が確かに対応関係を規定できていることを確かめるためには, 具体的な実現がこの代数的仕様を満たす必要がある.

2.1 対応関係

対応関係とは, ある型の値を他の型の値に対応づけているものであり, プログラミング言語の処理系の実現において用いられることが多い. たとえば, 変数と値の対応関係である状態や, 記憶場所の領域と記憶可能値の領域の対応関係である記憶域などが用いられている.

状態と記憶域の具体例を図1, 図2に示す. 図1は, 変数 **a**, **b**, **c** がそれぞれ値 3, 4, 2 に対応づけられた状態を表している. 図 2 は, 記憶場所 loc1, loc2, loc3 がそれぞれ値 5, –2, 8 に対応づけられた記憶域を表している. 両者は異なる要素を対応づけているが対応関係であることに違いはない.

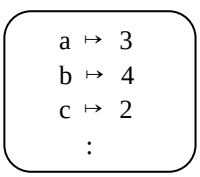

図1. 状態の例

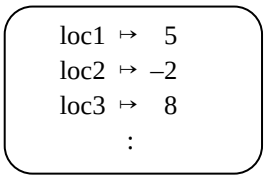

図2. 記憶域の例

2.2 関数による対応関係の実現例

一般に, 型 **a** を型 **b** に対応づける記述ができれば, 個々の対応関係は特殊化によって得られる. たとえば, 前小節では変数と値の対応関係および記憶場所と記憶可能値の対応関係を考えた. しかし, こうした対応関係は, これらを抽象化した多相データ型

```
type Assoc a b
```

を特殊化したものと見なすことができる. すなわち, 変数 **Ide** と値 **Val** の関係 **Assoc Ide Val** および記憶場所 **Loc** の領域と記憶可能値 **Val** との対応関係 **Assoc Loc Val** である.

対応関係の代数的仕様を図3に示す. 関数 **none** は, 空の対応関係を表す. すなわち, 型 **a** の任意の要素に対応する型 **b** の要素が存在しないような対応関係を表す. 関数 **lookup h x** は, 対応関係 **h** と型 **a** の要素 **x** を受け取り, **h** において **x** と対応づけられてる値を返す. 関数 **update h x v** は, 対応関係 **h** の中で **x** に対応づけられている先の値をあらたに **v** に定める関数である. 第 1 の等式は, 空の対応関係の中で **x** に対応する値が存在しないため値が定められないことを示している. ここで **undefined** は値が定められないことを表すものである. 第 2 の等式は, **x** が **v** に対応するよう更新された対応関係では **x** に対応するものは **v** であることを示している. 第 3 の等式は, **x** が **v** に対応するよう更新された対

```
none :: Assoc a b
lookup :: Assoc a b -> a -> b
update :: Eq a =>
          Assoc a b -> a -> b ->
          Assoc a b

lookup none x = undefined
lookup (update h x v) x = v
lookup (update h x v) x′ = lookup h x′
```

図3.対応関係の代数的仕様

応関係では, **x** と異なる **x′** と対応する値は, 元々の対応関係で **x′** と対応する値であることを示している.

具体的な実現法のひとつとして, 対応関係を型 **a** から型 **b** への関数で実現する方法を考える:

```
type Assoc a b = a -> b
```

空の対応関係 **none** は, 任意の **x** について **undefined** となるものとして実現する:

```
none :: Assoc a b
none x = undefined
```

対応づけられている値を返す関数 **lookup** は関数適用で実現する:

```
lookup :: Assoc a b -> a -> b
lookup h x = h x
```

対応関係を更新する関数 **update** は対応関係を表す関数を更新することで実現する:

```
update :: Eq a =>
          Assoc a b -> a -> b ->
          Assoc a b
update h x v y
   | x==y      = v
   | otherwise = lookup h y
```

この実現方法が抽象型の具体的な実現になっていることを図 3 の代数的仕様を満たしていることにより示す.

まず, 空の対応関係において任意の値 **x** に対応づけられる値が定められないことを等式変形により示す:

```
  lookup none x
= { lookup の定義 }
  none x
= { none の定義 }
  undefined
```

次に, **x** に対応する値を **v** に更新した後に, **x** に対応する値が **v** になっていることを示す:

```
  lookup (update h x v) x
= { lookup の定義 }
  update h x v x
= { update の定義 }
  v
```

最後に, **x** に対応する値を **v** に更新した後に, **x′** に対応する値は元々の対応関係から求められることを示す:

```
  lookup (update h x v) x′
= { lookup の定義 }
  update h x v x′
= { update の定義 }
  lookup h x′
```

以上の3つの等式変形により, 本節における関数による対応関係の実現が図3の代数的仕様を満たしていることが確認できた.

2.3 値の組のリストによる対応関係の実現例

本節では, 前節とは異なった具体的な実現を考える. すなわち, 対応関係を表す抽象型 **Assoc a b** を値の組のリストによって実現することを考える:

```
type Assoc a b = [(a,b)]
```

Assoc a b の実現リストは, 型 **a** の **x** と型 **b** の **y** が対応する場合, **(x,y)** を要素に含む. 空の対応関係 **none** は, こうした要素を全く含まない空リスト **[]** となる.

```
none :: Assoc a b
none = []
```

関数 **lookup** は実現リスト中の組の第 1 要素に **x** が出現するものを見つけ, その第 2 要素を返すことで実現する:

```
lookup :: Assoc a b -> a -> b
lookup [] x = undefined
lookup ((x′,v′) : h′) x
   | x==x′ = v′
   | otherwise = lookup h′ x
```

この具体的な実現法が図 3 に定められた代数的仕様を満たしていることは以下の3つの等式変形によって示される:

```
  lookup none x
= { none の定義 }
  lookup [] x
= { lookup の定義 }
  undefined
```

```
  lookup (update h x v) x
= { update の定義 }
  lookup ((x,v) : h) x
= { lookup の定義 }
  v
```

```
  lookup (update h x v) x′
= { update の定義 }
  lookup ((x,v) : h) x′
= { lookup の定義 }
  lookup h x′
```

2.4 対応関係の使用例

本節では, 簡単な状態を抽象型を用いた対応関係を用いて実際に実現する. 例として, 図 4 の対応関係を実現する. 空の対応関係 **none** から **update** 関数を用いて対応関係を構築する:

```
update (update none "a" 3) "b" 4
```

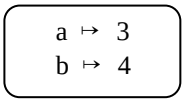

図 4. 対応関係の実現例

作成した対応関係を用いて **"a"** に対応する値を見つけることは以下のようにできる:

```
  lookup (update (update none "a" 3) "b" 4) "a"
= {代数的仕様の適用}
  lookup(update none "a" 3) "a"
= {代数的仕様の適用}
  3
```

3 おわりに

本研究で次のことの理解が得られた. 抽象型は, データの実現法に依らずに利用でき, 使用され方に依らずに実現できる. また, プログラミング言語処理系の実現で良く出てくる対応関係を実現することに活用できる.

参考文献

[1] 武市正人:プログラミング言語, 岩波書店 (1994).