構造モジュール　：　型合成のための言語機構

1J－8

久米田　暁文　　田中　譲
北海道大学　工学部

1　はじめに

我々は、語と語を合成して新たな語を定義することによってプログラムを作成していく、語彙に基づくプログラミングを提案している1)。文献1)では抽象データ型をベースにしてプログラミング言語を考えていたため、語と語の合成は可能ではあるが、その言語機構は複雑であり、柔軟性にも欠けていた。そこで、語と語の合成を簡明かつ柔軟に行える言語機構として構造モジュール(Structural Module)を考えたので、それについて報告する。構造モジュールは構造型のスロットの各々に、スロット間の依存関係を表す表現(依存表現)を割当てたものである。構造モジュールには次のような特長がある。(1)複数の既存の構造モジュールを合成して、新たな構造モジュールを定義することができる。(2)スロット間の依存関係によってデータを抽象化することができる。

以下、構造モジュールによるプログラミングについて述べる。

2　構造モジュールによるプログラミング

例として、二端子電気回路を考える。図1のような回路の両端に適当な電圧源(あるいは電流源)を接続したときの各部の電圧、電流、インピーダンスを計算するプログラムについて考える。このプログラムは図2にまとめてあるので、以下でこれについて説明する。

図2(a)では構造モジュールの定義に先だって、スロットの宣言を行っている。スロットの宣言は大域的なものであり、スロット名とその型を宣言している。スロットを大域的に宣言するのは、同名、同型のスロットがいくつかの構造モジュールのなかで使われることを意図しているからである。図2(a)で宣言されているのは、周波数(f)、インピーダンス(z)、電圧(v)、電流(i)、抵抗値やコンデンサの容量など素子の特性を表す値(value)、可変素子の最大値(max)、同じく最大値に対する比率(ratio)、直列回路や並列回路の部分回路(a、b)のそれぞれを表すスロットである。スロットa、bの宣言中の*は型変数である。これらのスロットの型は*&circuitという構造モジュールの合成(後述)によって表されている。

図2(b)は構造モジュールcircuitの定義である。このcircuitと後で定義する構造モジュールを合成することにより、種々の回路を表す構造モジュールが得られる。CIRCUITはcircuitのインスタンスを生成するためのデータ構成子である。このCIRCUITはスロットf、z、v、iをもつことが宣言されている。{}の中はスロット間の依存関係を表す依存表現の定義である。ここではvとiに依存表現が割当てられている。multiply__complex[z, i]などは関数の適用である。この依存関係をすべてのスロットをノードにもつグラフに表すことにより、各データ構成子に対して、図3に示すような依存グラフが得られる。CIRCUITの依存グラフは図3(1)である。グラフの矢は、依存表現に現れているスロットから、その依存表現の定義されているスロットに向けられている。スロットは、依存表現が割当てられているか、いないかで、targetとsourceの二種類に分類される。依存グラフに強連結成分がある場合は、CIRCUIT{v}のような形式で、強連結成分に含まれるtargetをsourceに変えることができる。

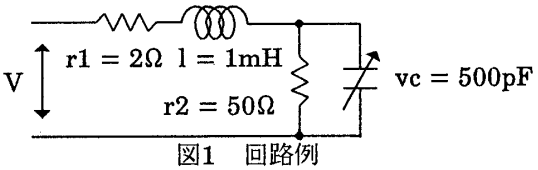

図1　回路例

インスタンスを生成する場合は、CIRCUIT{v}<f←..., z←..., v←...>のような形式で、生成に使うデータ構成子のすべてのsourceに値を代入する。このとき、そのデータ構成子の依存グラフに強連結成分があってはならない。インスタンスのスロットの値は、遅延評価される。

図2(c)は素子を表す構造モジュールdeviceの定義である。図2(d)のRESISTORは、DEVICEをsourceのzに依存表現を割当てることによって、特殊化したデータ構成子であり、やはりdeviceのインスタンスを生成する。このような場合、新たに強連結成分ができるような依存表現を割当てることは禁止される。RESISTORの依存グラフは図3(2)である。素子を回路として見たときの構造モジュールはdevice&circuitという構造モジュールの合成で与えられる。この合成によってdeviceとcircuitの両方のスロットと依存表現をもつ構造モジュールが得られる。device&circuitのデータ構成子はRESISTOR&CIRCUITなどで与えられる。RESISTOR&CIRCUITの依存グラフは図3(1)と図3(2)を合成したものとなる。

Structural Module : a Language Construct for Type Composition
Akefumi KUMETA, Yuzuru TANAKA
Hokkaido Univ.

RESISTOR&CIRCUITのようにデータ構成子を合成する場合、それらに共通なスロットがあるときは、targetは高々一つである必要がある。さもなければ、一つのスロットに二つの依存表現が割当てられるという不都合が生じる。RESISTOR&CIRCUITの場合、共通なスロットとしてfとzがあるが、fについては両方ともがsourceであり、zについてはCIRCUITの方がsourceなので、この条件は満たされている。

図2(e)のvariableは素子の可変性を表す構造モジュールである。データ構成子VARIABLEの依存グラフは図3(3)である。可変素子を表す構造モジュールは、variable&device&circuitで与えられる。このデータ構成子はVARIABLE&RESISTOR&CIRCUITなどで与えられる。

直列回路や並列回路は部分回路の接続として得られる。部分回路の型は、種々の場合があるので、型変数を用いて多相型として表す必要がある。図2(f)のconnectiveは部分回路をもつ回路のための構造モジュールである。部分回路はスロットa、bで表される。*と**が型変数である。

```
f :: real                                          (a)
z :: complex
v :: complex
i :: complex
value :: real
max :: real
ratio :: real
* a :: *&circuit
* b :: *&circuit

circuit ::= CIRCUIT(f, z, v, i){                   (b)
    v←multiply__complex[z, i],
    i←divide__complex[v, z]}

device ::= DEVICE(f, value, z)                     (c)

RESISTOR = DEVICE{                                 (d)
    z←COMPLEX<re←value, im←0.0>}

variable ::=                                       (e)
    VARIABLE(max, ratio, value){
        value←multiply__real[max, ratio]}

* ** connective ::=                                 (f)
    Z1 Z2 CONNECTIVE
          (f, z, v, i,
          * a of Z1&CIRCUIT,
          ** b of Z2&CIRCUIT){
        f↑a←f, f↑b←f}

Z1 Z2 SERIES =                                     (g)
    Z1 Z2 CONNECTIVE{
        z←plus__complex[z↑a, z↑b],
        i↑a←i, i↑b←i}

r1 = RESISTOR{value←1.0}                           (h)
r2 = RESISTOR{value←50.0}
l = COIL{value←1.0^-3}
vc = VARIABLE&CONDENSER{
    max←500.0^-12}
ex = (r1 l SERIES) (r2 vc PARALLEL)
        SERIES
```

図2　プログラム例

部分回路をもつ回路の構造モジュールは、(* ** connective)& circuitで表される。データ構成子CONNECTIVEでは、スロットa、bもデータ構成子である。すなわち、CONNECTIVEはネストしたデータ構成子である。a、bは、部分回路の種類に応じて種々のデータ構成子の場合がある。そこで、データ構成子の変数Z1、Z2をCONNECTIVEにもたせ、スロットの宣言の際にa、bがそれぞれ、Z1&CIRCUIT、Z2&CIRCUITというデータ構成子であることを宣言している。図2(g)のSERIESは直列回路を表すデータ構成子である(依存グラフは図3(4))。Cをデータ構成子、s、tをスロットとするとC{s←...,t←...}はC{s←...}{t←...}の略記である。uが強連結成分に含まれるtargetのとき、C{u}{u←...}は、C{u←...}と略記することができる。また、f↑aはaのfスロットの参照を表す。並列回路の場合も同様に定義することができる。

図2(h)は、図1で示した回路例に対応させて定義したデータ構成子である。exが回路全体に対応している。

3　おわりに

以上で示してきたように、構造モジュールでは、依存関係によってデータの抽象化を行っている。抽象データ型では、これは情報隠蔽によって行われている。そのため抽象データ型を合成して、新たな抽象データ型を得るのは難しい。これに対し、構造モジュールの合成は容易である。また、具体例は示さなかったが、スロットの依存表現や構造モジュールを再帰的に定義することができる。関数も、引数と返値の間の依存関係として考えることにより、構造モジュールによって表現することができる。今後は型推論の規則を確立することが課題であると考えている。

参考文献

1)　久米田暁文,田中譲:語彙に基づくプログラミングを可能とするための関数型言語,電子情報通信学会技術研究報告,SS88-16,1988

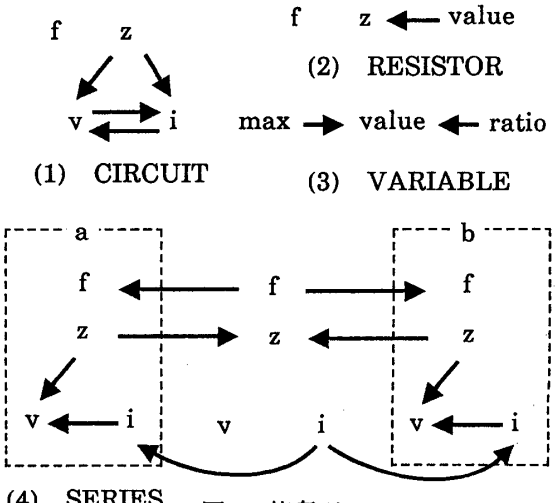

図3　依存グラフ