4X-3

関数型言語の検証系の試作

鈴木賢三，永田守男
(慶應義塾大学理工学部)

1. はじめに

ＬＩＳＰは、もともとラムダ算法に基づいて設計された言語だが、その後プログラミング言語として独立し、豊富な機能を備えるようになった。このことは、プログラムの生産性を向上させた反面、複雑でわかりにくいプログラムの記述を助長し、保守やデバッグを困難なものにしている。こうした状況を踏まえ、我々は、プログラミングの早さよりも信頼性の高さと保守の容易さに重点を置いて、プログラミングと検証を並行してできる関数型言語，およびその処理系を設計・試作中である。本稿では、その言語の特徴と検証方法の概要について述べる。

2. 言語の特徴

(1) グローバル変数を持たない。

関数の外の変数に値を束縛したり，外の変数の値を参照したりすることは、関数型言語本来の美しさを損ない、プログラムの論理的な扱いを難しくしているので、これを排除する。その結果、言語の記述能力は低下するが、関数のモジュール性は高まる。

(2) 全ての関数の引数と評価値のデータ型を宣言する。

ＬＩＳＰは型宣言を必要としない。このことは、大規模なプログラムを作成するときに混乱をまねく原因となっている。本言語では，関数の呼出しにおけるデータ型の不整合をオートマティックにチェックし、データ型に関連するエラーを未然に防ぐことをねらっている。

(3) 基本関数と基本データ型、およびその公理系が与えられている。

基本データ型とは、本言語があらかじめ用意しているデータ型のことで、自然数，ブール、アトム、リストの４種類だけである。データを操作するのに必要となるいくつかの演算(関数)を基本関数といい、その意味は代数的仕様によって与えられる。検証は、この仕様の公理にもとづいて行われる。

(4) 関数定義は、他の関数の呼出しとＩＦおよび再帰だけでなされる。

ＬＩＳＰには、ＣＯＮＤ，ＤＯなど様々な制御構造があるが、基本的には分岐と繰返しの組合せである。本言語では、分岐にはＩＦ，繰返しには再帰だけを用いて記述する。

(5) 関数だけでなく、データ型も定義できる。

データ型は、代数的仕様および既定義のデータ型による実現に関する記述によ

An Implementation of the verifier on the functional programming language
Kenzo SUZUKI and Morio NAGATA
KEIO Univ. Faculty of Science and Technology

り、階層的に定義できる。

3. 検証方法の概要

(1) 静的解析

定義された関数の構文上の誤りのチェック、およびデータ型に関するチェックを行う。(例1)

例1. $f(x,y,z)$ ≜ IF $p(x,y)$ THEN $g(y,z)$
ELSE $f(h_1(x), h_2(y), z)$

f : $t_1, t_2, t_3 \rightarrow t_4$

(f の入力条件) = $f_{pre}(x,y,z)$

のとき、以下のことを確かめる。

[データ型の制約]

p: $sc(t_1), sc(t_2) \rightarrow bool$
g: $sc(t_2), sc(t_3) \rightarrow t_4$
h_1: $sc(t_1) \rightarrow t_1$
h_2: $sc(t_2) \rightarrow t_2$

[入力条件の制約]

$f_{pre}(x,y,z) \supset p_{pre}(x,y)$
$f_{pre}(x,y,z) \supset g_{pre}(y,z)$
$f_{pre}(x,y,z) \supset h_{1pre}(x)$
$f_{pre}(x,y,z) \supset h_{2pre}(y)$
$f_{pre}(x,y,z) \supset f_{pre}(h_1(x), h_2(y), z)$

(ここで $sc(t)$ とは、データ型 t または t の上位クラスを指す。)

例2. (つづき) $f(x,y,z)$ の評価値を w とする。

(f の出力条件) = $f_{post}(x,y,z,w)$

のとき、以下の2式がいえれば部分正当である。

$f_{pre}(x,y,z) \wedge p(x,y) \rightarrow f_{post}(x,y,z,g(y,z))$
$f_{pre}(x,y,z) \wedge \neg p(x,y) \rightarrow f_{post}(x,y,z,f(h_1(x),h_2(y),z))$

例3. $gcd(x,y)$ ≜ IF $x>y$ THEN $gcd(y,x)$
ELSE IF $rem(x,y)=0$ THEN y
ELSE $gcd(x-y,y)$

$pt(x,y) = x+2y$ として以下がいえれば gcd は停止する。 ($gcd_{pre}(x,y) = x>0 \wedge y>0$)

$gcd_{pre}(x,y) \supset pt(x,y) > 0$
$gcd_{pre}(x,y) \wedge x<y \supset pt(x,y) > pt(y,x)$
$gcd_{pre}(x,y) \wedge \neg x<y \wedge \neg rem(x,y)=0 \supset pt(x,y) > pt(x-y,y)$
$halt(gcd(1,1))$.

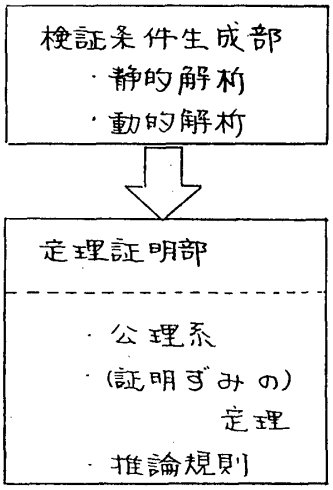

図1. 検証系の構成

(2) 動的解析

関数の定義が仕様を満足することを確かめる。そのため、部分正当性と停止性を証明する。

部分正当性証明のための条件の生成例を例2に示す。

停止性証明のための条件の生成例を例3に示す。ポテンシャル関数を与え、それが繰返しについて減少することと、ポテンシャルが最小になる場合に停止することを示す。

これらの検証条件は、仕様にもとづいて形式的な論理式に変換され、定理証明部で、半自動的に証明がなされる。(図1)

4. おわりに

本検証系は、誤ったプログラムの修正に有益な情報を与えることに重点をおいて設計した。

今後その点について、さらに検討を進める予定である。

参考文献

1) R.S. Boyer and J.S. Moore, A Computational Logic, Academic Press.
2) J.L. Bates and R.L. Constable, Proofs as Programs, ACM TPLAS, Vol.7, No.1 (1985).