Agda 言語による スクリプト検証 について

湯 浅 能 史[†1] 木 下 佳 樹[†1]

関数型プログラミング言語 Agda は証明記述言語でもある。本講演では Agda 言語を用いた ソフトウェア検証の方法を紹介する。スクリプト 処理検証への応用についても論じる。

Verification by Agda in script programming

Yoshifumi Yuasa[†1] and Yoshiki Kinoshita[†1]

Agda, a functional programming language, is a powerful proof description langage as well. In this talk, we introduce basic methods to verify software by Agda, and argue their applications to script programming.

1. はじめに

本講演では Agda を用いた ソフトウェア検証の方法を紹介する。Agda とは 関数型プログラミング言語とその処理系の呼称である。これらはスウェーデン Chalmers 工科大を中心に開発が進められている。[1)]

Agda は証明記述言語としての側面も持つため、定理証明法によるソフトウェア検証ツールとしても利用される。アルゴリズムを Agda プログラムとして記述してその性質を証明する、という方法が良くとられるが、だからといって、検証できる対象が関数型言語に限られるわけではない。Agda 言語の豊かな表現力を用いれば、言語の意味論を与える数学的モデルを表現することができ、これにより、スクリプト処理に用いられるような命令型言語の検証も可能となる。

2. Agda 言語 と定理証明

プログラミング言語としての Agda は Martin-Löf 型理論に基づく強力な型システムを持っている。他の良く知られた関数型言語とは異なる極立った特徴は、関数型 “$X \to Y$” を一般化した、依存積型 “$(x : X) \to Y(x)$” が導入されている点である。値域を表す型表現にパラメータが許されており、定義域中の x が向かう先を x に “依存して” 決められる、というのが依存積たる結縁である。

型を命題と捉えて、その型を持つプログラムを証明と捉える Curry-Howard 同型対応の原則では、関数型は含意命題、依存積は述語の全称束縛に解釈される。連言や選言のような命題演算や、述語の特称束縛も、データ型により定義することができる。これらにより Agda 言語は高階述語論理に相当する表現力を持つ。

Agda による証明記述の一例を図 1 にあげた。ここでは、まず自然数とその上の等号を定義している。Agda では 等号 は組込まれたプリミティブではなく、定義されるべきものである。このような述語記号の定義はデータ型として導入することが多い。その後、加法の定義を通常の方法で行い、簡単な補題を導入して、

```
data nat : Set where
 zero : nat
 succ : nat → nat

data _==_ (n : nat) : nat → Set where
 refl : n == n

_+_ : nat → nat → nat
zero   + m = m
succ y + m = succ (y + m)

succ-eq :
 {x y : nat} → x == y → succ x == succ y
succ-eq refl = refl

assoc :
 (n m l : nat) → (n + m) + l == n + (m + l)
assoc  zero    m l = refl
assoc (succ y) m l = succ-eq (assoc y m l)
```

図 1 Agda による加法の結合律の証明

†1 (独) 産業技術総合研究所
National Institute of Advanced Industrial Science and Technology (AIST)

```
[_] : {S : Set} → Trans S → Prop S → Prop S
[ P ] A s =  (t : S) → P s t → A t

_#_ : {S : Set} → Trans S → Trans S → Trans S
_#_ P Q s u = Σ S (λ t → (P s t) × (Q t u))

seq : {S : Set} → {P Q : Trans S} → {A : Prop S} →
      (s : S) → [ P # Q ] A s → [ P ] ([ Q ] A) s
seq s x =
 λ t p u q → x u (t , (p , q))
```

図 2　様相記号とプログラム逐次実行

結合法則を示している。

3. 命令型言語におけるプログラム検証

命令型言語などを検証する場合には、前例のように直接的な方法はとれない。対象言語に意味論を与える数学モデルを Agda 言語で表現し、プログラムの振舞いをこのモデル上の命題として証明することになる。ここでは一例として、状態遷移モデルを紹介する。

命令型言語 L とその実行機械 M が与えられたとき、M の取り得る内部状態の集合を S とする。プログラムの実行は M の状態遷移を引き起こすものなので、その遷移関係を表す S 上の2 引数述語として捉えられる。また、ある時点の機械 M に関する “命題” は、それを満すような状態の集合を指定するものである。よって、これは S 上の1 引数述語と捉えられる。この様な定式化の下、プログラム P と命題 A に対して、様相命題 “$[P]\,A$” を次のように定める。

$$[P]\,A\,s \overset{\triangle}{\Longleftrightarrow} \forall t, P\,s\,t \Rightarrow A\,t$$

これは「プログラム P の実行後には必ず A が成り立っている」ことを表す命題である。

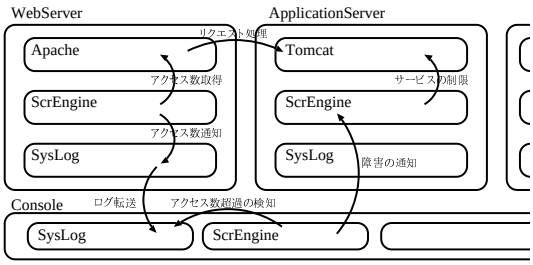
図 2 に示した Agda コードでは、上記の様相記号を定義して、プログラムの逐次実行を表す演算 ‘#’ との関係を証明している。この命題 *seq* は、命令型言語の検証に使われる論理体系である動的論理 (Dynamic Logic)[3] の公理の一つである。この他、条件分岐や繰り返し実行、選択的実行についての公理もあり、これらを証明してライブラリ化しておけば、この体系に基くプログラム検証を Agda 上で行うことができる。

4. スクリプト処理と Agda

前節で述べたモデル化では、実行機械 M やその状態集合 S の具体的な構造は、必ずしも明かである必要はない。実行機械の振舞いを外からの観察により特定して、それを公理として仮定すれば、それを前提とした検証を行える。

スクリプト言語は、既存のアプリケーションソフトウェアや機械システム等の制御を行う目的で利用される。対象となるシステムの内部構造は不明であることが多い。このような Black Box を含む複合的システムにおいて、スクリプト処理が適切に行われていることを検証するのは、モデル検査のような自動ツールでは困難であり、Agda 等を用いた定理証明法による検証が有用なのではないかと考える。

文献[4] ではこのようなシステムの検証例として、複数のサーバからなるネット販売システムの自動障害対応を扱っている。各サーバには幾つかのソフトウェアが稼働しており、複数のスクリプトがこれらを制御して障害に対処する。各コンポーネント（ サーバ, ソフトウェア）の遷移モデルを作り、状態集合間の射影を利用して、それぞれの語彙や公理を相互引用しながら、性質の証明を行った。



5. ま と め

関数型プログラミング言語 Agda とそれによるソフトウェア検証法を紹介した。またスクリプト処理検証への応用について考察した。

参 考 文 献

1) “The Agda wiki”, `http://wiki.portal.chalmers.se/agda/pmwiki.php`.
2) Howard, William A., ”The formulae-as-types notion of construction”, in Seldin, Jonathan P.; Hindley, J. Roger, To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism, Boston, MA: Academic Press, 1980, pp. 479490
3) Vaughan Pratt, “Semantical Considerations on Floyd-Hoare Logic”, Proc. 17th Annual IEEE Symposium on Foundations of Computer Science, 1976, 109-121.
4) 湯浅能史, 木下佳樹. システムのコンポーネント構成に着眼した CD ネット販売における障害対応の Assurance Case 作成, 算譜科学研究速報 (AIST-PS-2012-003), 産業技術総合研究所