ATS 言語による組込アプケーションの検証

岡部 究†

Verification of embedded application by ATS language

Kiwamu Okabe

ねらい: 本書は ATS 言語の持つ定理証明の機能を小規模組込アプリケーションに応用することで、当該組込アプリケーションの安全性を向上する手法を示し、当該言語の組込領域による利用促進を促す。
キーワード: 定理証明，ATS，依存型，線形型，組込アプリケーション

Target: This abstract is to introduce theorem proving on embedded application, and provide a method improving safety of the application.

Keywords: Theorem proving, ATS, dependent types, linear types, embedded application

１．想定する読者・聴衆

本書は第一に小規模なマイコン上で動作する組込アプリケーションの設計者を想定している。そのような設計者は日常的にバッファオーバーランに代表される不具合に悩まされており、その予防策を探している。ATS 言語[1]の備える線形型によるポインタの安全な使用はそのような予防策として有効である。

第二にメニーコアアーキティクチャのような特殊なハードウェア上で動作するアプリケーションの設計者を想定している。そのような設計者は通常の OS とはかけはなれたハードウェアの特性に起因する不具合に悩まされており、またそのハードウェアの上では通常のプログラミング言語が動作しない。ATS 言語の備える定理証明の機能はそのような OS のサポートがない環境下におけるソフトウェアの挙動を静的に検証することができる。

第三に言語処理系設計者を想定している。ATS 言語は 2004 年に初出論文[2]が発表されたまだ若い言語で、その機能の多くは言語設計者の間でもよく理解されていない。本書で実際の使用例を示すことで、ATS 言語の持つ機能を備える別の新しい使いやすい言語が登場することを期待できる。

2. 背景

組込アプリケーションに対する多機能性、高信頼性と納期短縮の要求は年々向上している。ところが明確にこの 2 つの要求は相反している。これらの要求を同時に満たすには高い信頼性を作りこむ手法の整備が急務である。通常このような信頼性の担保には設計と分離した形式手法が用いられる。このような形式手法はその検査と設計を同期させる必要があり多くの場合その同期は人力によって行なわれる。その結果、アプリケーションが継続的に開発される場合には検査と設計の乖離が容易に起きうる。

一方、設計に用いる言語は進化を続けており、Haskell[3]や OCaml[4]のような型推論を持つ静的型付け言語は汎用 OS の存在下において安全なプログラミングを行なうことができる。また Coq[5]や Isabelle[6]のような定理証明器は GC を用いたプログラムを静的に検査することができる。しかしこれらの言語の機能は主に汎用 OS の上で動作するアプリケーションを対象にしており、組込 OS 上で動作するアプリケーションには適用することが難しかった。

一般にあまり知られていない ATS 言語を用いることで OS が不在であっても型推論を用いたプログラミングを行なうことができ、またこの言語は定理証明の機能も持っている。この言語設計は明確に組込アプリケーションの設計を意図しているが、そのような組込アプケーションの具体例は少なく、広く認知されていなかった。

3. 課題

組込 OS 上で動作するアプケーションに対すて、ATS 言語を用いた検証手法の実例を示し、言語処理系に対する知識がない設計者にもその効果と具体例をわかりやすく紹介する。これまで ATS 言語の機能に対する知見は言語処理系に対する詳しい知識を要求していた。本書では、その手法を広く社会に紹介するために、具体例の紹介に重きを置く。

†理化学研究所 計算科学研究機構
RIKEN Advanced Institute for Computational Science 7-1-26 Minatojima-minami-machi, Chuo-ku, Kobe, Hyogo 650-0047, Japan
E-mail: kiwamu@debian.or.jp, twitter: @masterq_mogumog

4. 提案・実験

ATS 言語を用いたアプリケーションは 3 つの形式があり、それらは以下である。

A) GC を使用するアプリケーション

B) GC を使用せず malloc を使用するアプリケーション

C) GC も malloc も使用しないアプリケーション

このとき A 形式は GC を使うため主に汎用 OS の上で動作するアプリケーションで採用される。B 形式は GC を使用しないが malloc を使用するためそのような API を組込 OS に要求する。C 形式は GC も malloc も使用しないため、OS を要求せずにハードウェアで直接動作するようなアプリケーションを構築できる。本書では B と C の形式のアプリケーションについて取り扱う。

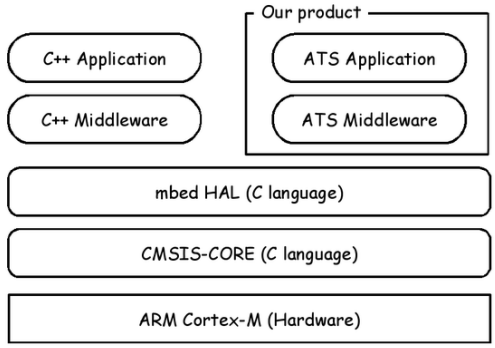

図 1 mbed OS 上で動作する ATS アプリケーション

B 形式のアプリケーションの実用例として、mbed OS[7]上で動作するアプリケーションを ATS 言語で試作した[8]。mbed OS には malloc API があるため、この形式を実現できる。図 1 はそのソフトウェアアーキィティクチャである。

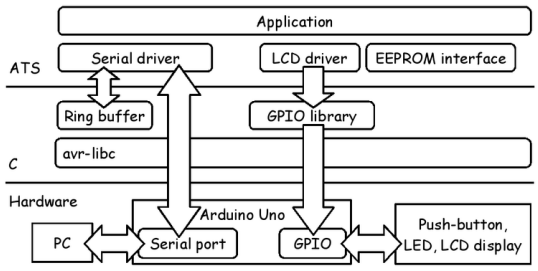

図 2 Arduino Uno 上で動作する ATS アプリケーション

C 形式のアプリケーションの実用例として、Arduino Uno ボード[9]上で OS なしに動作するアプリケーションを ATS 言語で試作した[10]。この成果は発表済みの論文「Arduino programing of ML-style in ATS」[11]で詳細に解説している。図 2 はそのソフトウェアアーキィティクチャである。

5. 効果

B 形式について、ATS 言語における安全な設計と malloc を利用した線形型の設計効率を組込 OS の上で両立可能であることを示した。

C 形式について、以前の Ajhc Haskell コンパイラ[12]を用いた試作[13]ではスマートポインタの使用が必須のために 8bit アーキティクチャや malloc が使用できないメモリ量(Arduino Uno ボードは 2kB)が障害となっていたが、ATS 言語を用いることでこのようなハードウェア上でも強い型を使用することができ、さらにはその定理証明の機能が有効に使用できることを示した。

ATS 言語の証明機能を用いる効果の一例として、長い文字列の切り出しにおけるバッファオーバーランの防止が上げられる。以下のコードは長さ n の文字列 str を開始位置 i から長さ j 切り出す関数で、この関数シグニチャは元の文字列 str の長さを超えて切り出さないことをコンパイル時に強制している。そのため文字列 str を超えた長さをこの lcd_print 関数を用いて切り出すコードはコンパイルエラーとなり、バッファオーバーランはコンパイル時に防止できる。

```
fun lcd_print {n:int}{i:nat}{j:nat | i + j <= n}
  (lcd: !lcd_t, str: string (n), start: size_t (i),
  len: size_t (j)): void
```

6. まとめ

本書では強い静的な型と定理証明の機能を持つ ATS 言語を用いて組込 OS 上で動作するアプリケーションと非力なハードウェアの上で直接動作するアプリケーションを設計できることを試作によって示した。これらのアプリケーションはそのソースコードをインターネット上から入手でき、追試可能である。

文　献

[1] The ATS Programming Language, http://www.ats-lang.org/

[2] Hongwei Xi, “Applied Type System (extended abstract)”, http://www.ats-lang.org/Papers.html#Applied_Type_System

[3] Haskell Language, https://www.haskell.org/

[4] OCaml, https://ocaml.org/

[5] The Coq Proof Assistant, https://coq.inria.fr/

[6] Isabelle, https://isabelle.in.tum.de/

[7] mbed, http://www.mbed.com/

[8] mbed-ats, https://github.com/fpiot/mbed-ats

[9] Arduino Uno, https://www.arduino.cc/en/Main/arduinoBoardUno

[10] arduino-ats, https://github.com/fpiot/arduino-ats

[11] Kiwamu Okabe, Hongwei Xi, “Arduino programing of ML-style in ATS”, http://www.mlworkshop.org/ml2015

[12] Ajhc - Haskell everywhere, http://ajhc.metasepi.org/

[13] Kiwamu Okabe, “How to rewrite the OS using C by strong type”, http://www.slideshare.net/master_q/20140117-11th-wocs2

略 号 一 覧

GC: Garbage collection