修士学位論文要約（平成30年３月）

関数型言語のためのコードレベルデバッグ環境の実現方式

大野 一樹
指導教員：大堀 淳，　学位論文指導教員：上野 雄大

A Study on Code Level Debugging Environment for Functional Languages

Kazuki ONO
Supervisor: Atsushi OHORI,　Research Advisor: Katsuhiro UENO

SML# is a functional programming language that enables the programmer to exploit the operating system features such as multithreads as well as functional language features such as first class functions. To debug an SML# program using all of such features, it is helpful not only to trace source level control flow but also to check how its machine code runs on a CPU with using low-level memory and registers. Toward realizing a software development environment that assists such the machine code level debugging for functional languages, we extend the SML# compiler with a debug information generation and make GDB (the GNU Project Debbuger) available for SML# programs. This thesis reports the details of the development.

1. はじめに

関数型言語は，C や Java などの手続き型言語に比べて，高い信頼性と安全性を持つという点で優れており，安全で信頼のおけるソフトウェアの生産に大きく貢献すると期待されている．しかし関数型言語には C や C++における GDB デバッガ[1]のような，ソースコードから機械語コードまでの挙動を確認するためのコードレベルデバッグ環境は十分でない．大きなソフトウェアの開発では，ソースコードだけでなく，連携するライブラリや言語の挙動を同時に確認できるような環境が必要不可欠である．そこで，本研究では関数型言語にコードレベルデバッグ環境を構築することを目標とする．環境の構築のための方針として，C 言語との直接連携機能を持つ関数型言語 SML#[2]ならば，C におけるコードレベルデバッグ環境である GDB デバッガを用いることが可能である，という推察を立てる．そこで，関数型言語 SML#を対象言語として，GDB デバッガの機能のうち，ステップ実行，ブレークポイントの設定，ローカル変数の値のプリントの機能の実現を目指す．実現のために，デバッグのためにコンパイラが生成するべき情報について，データフォーマットである DWARF[3]やコンパイラの中間表現である LLVM IR についての調査を行い，既存のデバッガである GDB を SML#から用いるための方針を立て，その方針のもと，コンパイラの LLVM IR 生成フェーズを拡張し，デバッグ情報の生成機能を加え，コンパイラが生成するコードを GDB でデバッグ可能なことを確認した．

2. DWARF の概要

DWARF は，デバッグ情報をオブジェクトファイルに格納する際に用いられるデータフォーマットのひとつである．DWARF のデバッグ情報は，ソースプログラムの構造を抽象する木構造と，ソースコード上の位置とメモリ上の配置を対応づける表（行番号テーブル）の 2 つからなる．木構造の各ノードは，ソースプログラムに由来する様々な情報を表す．DWARF では，これら木構造に含まれる各ノードを，Debug Information Entry (DIE) と呼ぶ．DIE には，木構造の根となるコンパイル単位 (compile unit)，など，ソースコードのブロック構造を表すものや，型情報や変数情報などを表すものが存在する．

3. LLVM における DWARF 情報の取り扱い

SML#がマシンコードを生成する基盤として用いている LLVM[4]には，DWARF の木構造を書くための記法が用意されている．LLVM は中間表現 LLVM IR で書かれたプログラムを入力として受け取り，マシンコードを生成する．LLVM IR プログラムは，関数または変数の宣言の列からなり，各関数は命令列をもつ．LLVM では，それら宣言や命令列に対する注釈として，デバッグ情報を書くことができる．

4. コンパイラの拡張

SML#コンパイラにおいてデバッグ情報を生成するために，コンパイラ内部のメタ情報の伝播と，デバッグ情報の生成の 2 つのステップに分けて拡張を行う．SML#コンパイラの各フェーズにおいては，命令列が由来する行番号やファイル名などのメタ情報が伝播している．ここで，トップレベルのファイル名の情報は十分に伝播されていなかったため，現状の

© 2018 東北大学電気通信研究所

```
 1: fun sumlist x =
 2:     case x of
 3:       nil => 0
 4:       | a::t =>
 5:         a + sumlist t
 6: fun makelist x =
 7:     case x of
 8:       0 => []
 9:       | n => n :: makelist (n - 1)
10: fun main () =
11:     let
12:       val x = makelist 2
13:       val y = sumlist x
...
17:     end
18:val _ = main ()
```

図 1　動作の確認に用いるプログラム

コンパイルフェーズの連鎖とは別に，情報を LLVM IR 生成フェーズに送るようにコンパイラを拡張する．次に，伝播したメタ情報をもとに，LLVM IR 生成フェーズにおいて，ソースコードの構造と各要素の位置を表すデバッグ情報を生成する．本フェーズにおいて，コンパイラは SML#のバリアント型として定義された LLVM IR の構文木を構築する．構文木の構築は，トップレベルや関数の生成，命令列の生成の 3 段階で行われる．このいずれの段階においてもデバッグ情報の生成を行う．ここで，命令列がどの関数に含まれるか，などのスコープ情報を生成するために，関数を生成する際に，生成したデバッグ情報を保持し，インストラクションを生成する関数の引数として与える．本フェーズで生成した構文木を LLVM IR に翻訳するためには，LLVM が提供する C の API を用いる．SML#の C との直接連携機能により，SML#コンパイラからこの API をそのまま利用することができる．ただし，いくつかの API については C++でのみ提供されているため，C 言語でスタブライブラリを実装した．

5. 構築したデバッグ環境の実行例

本拡張により，デバッグ情報が埋め込まれたオブジェクトファイルを生成する SML#コンパイラを得た．以下に，SML#プログラムを GDB でステップ実行する例を示す．対象プログラム (list.sml) を図 1 に示す．

このプログラムをコンパイルし，生成したファイルを GDB で実行すると，ブレークポイントを設定し，ステップ実行することができる．

```
(gdb) break list.sml:5
Breakpoint 1 at 0x4032df:
 file list.sml, line 5.
(gdb) run
...
Breakpoint 1, _SMLF7sumlist_27 ()
at list.sml:5
5  a + sumlist t
(gdb) step
_SMLF7sumlist_27 () at list.sml:1
1  fun sumlist x =
```

ブレークポイントを設定した場所で，info local コマンドを用いて，スコープ内のプリント可能な変数を調べると，以下のように表示される．

```
(gdb) info local
X1253 = 0x6d6650 <_SMLZN23SMLSharp
__SMLNJ__POSIX__IO5openfE>
X1254 = 0x412f47 <_SML_top+2231>
```

GDB のプリントコマンドを用いることで，格納されている値がそのまま出力されているのが確認できる．本実験によって，ソースプログラムにある変数に対して値をプリントできることが確認できた．

6. まとめと今後の課題

関数型言語の実用化のために必要不可欠であると考えられるコードレベルデバッグ環境の実現の第一歩として，関数型言語 SML#で書かれたプログラムに対して GDB デバッガを用いて，ブレークポイントの設定，ステップ実行の機能を実現した．ローカル変数のプリントについては，実験的な実装を行った．今後の課題としては，関数型言語における理想的なステップ単位の考察や，コンパイラの保持するべき情報，多相型の値のプリントなどが挙げられる．これらの課題に取り組むことで，関数型言語のコードレベルデバッグ環境の完成に近づくと考えられる．

文献

1) GDB: The GNU Project Debugger
http://www.gnu.org/software/gdb/

2) SML# Compiler
http://www.pllab.riec.tohoku.ac.jp/smlsharp/

3) DWARF3.0 ドキュメント
http://www.dwarfstd.org/doc/Dwarf3.pdf

4) LLVM ドキュメント
http://releases.llvm.org/3.7.0/docs/
SourceLevelDebugging.html