AT-TaPL 4: Typed Assembly Language pt.1

河原 悟

2017 年 6 月 6 日

P142〜

0 introduction

適当に引っ張ってきたコードは正しく動くのか？

- *proof-carrying code(PCC)* by Necula amd Lee
  コードの性質の証明のチェックが簡単できて、proof-checking engine が小さい

PCC をうまく使っていくために、以下の問題を解決したい:

1. コードが満たすべき性質とは？
2. コードが満たすべき性質の証明をプログラマーはどう作るか？

1. は文脈やアプリケーションに依存し、2. は自動的にはできない。ではどうするか？

type-preserving compilation をベースにしたアプローチを考えてみる。コードが満たすべき性質として型安全であることにフォーカスする。

この方法論では、コンパイルのプロセスとして型付きマシンコードにコンパイルされる型付きの中間言語をデザインする必要がある。

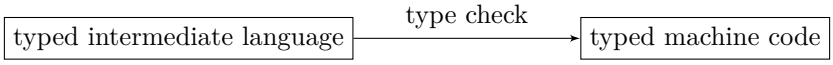

図 1　必要となる工程

CISC ライクではなく、RISC ライクな、高級言語の機能をエンコードでき種々の最適化が適用できる言語を考える。

1 TAL-0: Control-Flow-Safety

まず RISC スタイルの型付きアセンブリー言語を考えるにあたり、*control-flow safety* という性質にフォーカスしてみる。直感として、意図しないアドレスへのジャンプを防ぎ、適切なエントリーポイントにのみ飛ぶことができるように制御する。

r ::=		*registers* :	
	r1 | r2 | $\cdots$ | rk		
v ::=		*operands* :	
	n	*integer literal*	
	ll	*label or pointer*	
	r	*registers*	

i ::=		*instructions* :	
	r_d:=v		
	r_d:=r_s+v		
	if r jump v		
I ::=		*instruction sequences* :	
	jump v		
	$i; I$		

図 2　Instructions and operands for TAL-0

mov や add のような表記より familiar な表記を用いる。

R ::=		*register files* :	
	$\{\mathtt{r}_1 = v_1, \ldots, \mathtt{r}_k = v_k\}$		
h ::=		*heap values* :	
	I	*code*	

H ::=		*heaps* :	
	$\{l_1 = h_1, \ldots, l_m = h_m\}$		
M ::=		*machine states* :	
	(H, R, I)		

図 3　TAL-0 abstract machine syntax

マシンの状態 M は 3 つの状態をもつ:

1. ヒープ (H)
   ラベルからヒープ値 (h) への有限の部分写像
2. レジスターファイル (R)
   レジスター (r) から値 (v) への完全写像
3. 現在の命令列 (I)
   実際のマシンがプログラムカウンターで指している命令列

マシンの演算を jump までの命令のプリフェッチ、また jump をコントロールの移動が必ず起きる場所と考えることができる。

Listing 1　example

```
1 prod: r3 := 0;
2   jump loop
3
4 loop: if r1 jump done;
5   r3 := r2 + r3;
6   r1 := r1 + -1;
7   jump loop
8
9 done: jump r4
```

改めて TAL-0 の書き換え規則を記す。

$$\frac{H(\hat{R}(v)) = I}{(H, R, \texttt{jump } v) \to (H, R, I)} \quad \text{(JUMP)}$$

$$(H, R, r_d{:=}v; I) \to \Big(H, R[r_d = \hat{R}(v)], I\Big) \quad \text{(MOV)}$$

$$\frac{R(r_s) = n_1 \qquad \hat{R}(v) = n_2}{(H, R, r_d{:=}r_s + v; I) \to (H, R[r_d = n_1 + n_2], I)} \quad \text{(ADD)}$$

$$\frac{R(r) = 0 \qquad H(\hat{R}(v)) = I'}{(H, R, \texttt{if } r \texttt{ jump } v; I) \to (H, R, I')} \quad \text{(IF-EQ)}$$

$$\frac{R(n) = n \qquad n \neq 0}{(H, R, \texttt{if } r \texttt{ jump } v; I) \to (H, R, I)} \quad \text{(INF-NEQ)}$$

$\hat{R}$ は以下のような動作を表す。

- $\hat{R}(r) = R(r)$ レジスターの値
- $\hat{R}(n) = n$ 即値
- $\hat{R}(ll) = ll$ ラベル

`jump` と `if-jump` は invalid なラベルを渡すと stuck してしまう。

1.1 Exercise: レジスターファイルも初期値 R_0 を $\{\texttt{r1} = 2, \texttt{r2} = 2, \texttt{r3} = 0, \texttt{r4} = \texttt{exit}\}$、`exit` を特別なアドレスとしたとき、listing 1 が $(H, R_0, \texttt{jump prod})$ から $(H, R, \texttt{jump r4})$、$R(\texttt{r3}) = 4$ になるまでの評価を示せ。

$$M_0 = (H, R_0, \texttt{jump prod})$$

JUMP 規則により、

$$M_1 = (H, R_0, \texttt{if r1 jump done; r3 := r2 + r3; r1 := r1 + } -1\texttt{; jump loop})$$

$R_0(\texttt{r1}) = 2$ から IF-NEQ 規則により、

$$M_2 = (H, R_0, \texttt{r3 := r2 + r3; r1 := r1 + } -1\texttt{; jump loop})$$

$R_0(\texttt{r2}) = 2$ $\hat{R_0}(\texttt{r3}) = 0$ ADD 規則により、

$$R_1 = \{\texttt{r1} = 2, \texttt{r2} = 2, \texttt{r3} = 0, \texttt{r4} = \texttt{exit}\}$$
$$M_3 = (H, R_1, \texttt{r1 := r1 + } -1\texttt{; jump loop})$$

JUMP 規則により、

$$I_{loop} = \texttt{if r1 jump done; r3 := r2 + r3; r1 := r1 + } -1\texttt{; jump loop}$$
$$M_4 = (H, R_1, I_{loop})$$

$R_1(\texttt{r1}) = 2$ から、IF-NEQ 規則により、

$$M_5 = (H, R_1, \texttt{r3 := r2 + r3; r1 := r1 + } -1\texttt{; jump loop})$$

$R_1(\texttt{r2}) = 2$、$\hat{R_1}(\texttt{r3}) = 0$ から、ADD 規則により、

$$R_2 = \{\texttt{r1} = 2, \texttt{r2} = 2, \texttt{r3} = 2, \texttt{r4} = \texttt{exit}\}$$
$$M_6 = (H, R_2, \texttt{r1 := r1 + } -1\texttt{; jump loop})$$

$R_2(\texttt{r}_1) = 2$、$\hat{R_2}(-1) = -1$ から、ADD 規則により、

$$R_3 = \{\texttt{r1} = 1, \texttt{r2} = 2, \texttt{r3} = 2, \texttt{r4} = \texttt{exit}\}$$
$$M_6 = (H, R_3, \texttt{jump loop})$$

JUMP 規則により、

$$M_7 = (H, R_3, I_{loop})$$

$R_3(\texttt{r1}) = 1$ から、IF-NEQ 規則により、

$$M_8 = (H, R_3, \texttt{r3 := r2 + r3; r1 := r1 + } -1\texttt{; jump loop})$$

$R_3(\texttt{r2}) = 2$、$\hat{R_3}(\texttt{r3}) = 2$ から、ADD 規則により、

$$R_4 = \{\texttt{r1} = 1, \texttt{r2} = 2, \texttt{r3} = 3, \texttt{r4} = \texttt{exit}\}$$
$$M_9 = (H, R_4, \texttt{r1 := r1 + } -1\texttt{; jump loop})$$

$R_2(\texttt{r}_1) = 1$、$\hat{R_2}(-1) = -1$ から、ADD 規則により、

$$R_5 = \{\texttt{r1} = 0, \texttt{r2} = 2, \texttt{r3} = 4, \texttt{r4} = \texttt{exit}\}$$
$$M_{10} = (H, R_5, \texttt{jump loop})$$

JUMP 規則により、

$$M_{11} = (H, R_5, \texttt{jump r4})$$

以上。

1.2 Exercise: **実装してみよう**

Typed Racket で実装した。`https://bitbucket.org/nymphium/attapl/` chap4-typed-assembly-language/src/tal0

Listing 2 test.rkt

```
1 #lang typed/racket
2
3 (require "interpreter.rkt")
4
5 (: r RegFiles)
6 (define r (hash 1 (Imm 2) 2 (Imm 2) 3 (Imm 4) 4 (Label "exit")))
7
```

```
(: h Heaps)
(define h (hash
          "prod" (list
                  (Assign (Reg 3) (Imm 0))
                  (Jmp (Label "loop")))
          "loop" (list
                  (IfJmp (Reg 1) (Label "done"))
                  (Add (Reg 3) (Reg 2) (Reg 3))
                  (Add (Reg 1) (Reg 1) (Imm -1))
                  (Jmp (Label "loop")))
          "done" (list
                  (Jmp (Reg 4)))))

(eval (MachineStatus h r (list (Jmp (Label "prod")))))
```

2 TAL-0 Type System

TAL-0 の型システムが目指すところは well-formed な抽象機械が stuck しないこと–つまり progress と preservation を満たしているということである。あるラベルに制御が移行するとき、レジスターの値がどうなっているかなどが明らかになっている必要がある。

$\tau ::=$		*operand types* :	$\Gamma ::=$		*register file types* :
	$\texttt{int}$	$word-sized\ integers$		$\{\mathtt{r_1} : \tau_1, \ldots, \mathtt{r_k} : \tau_k\}$	
	$\texttt{code}(\Gamma)$	*code labels*	$\Psi ::=$		*heap types* :
	α	*type variables*		$\{l_1 : \tau_1, \ldots, l_n : \tau_n\}$	
	$\forall\alpha.\tau$	*universal polymorphic types*			

図 4　TAL-0 type syntax

$\texttt{code}(\Gamma)$ はレジスターファイルの型 Γ におけるラベルの型と見ることができる。!!Γ をレジスターから型への全域関数と見ると、ラベルを、値のレコード Γ を引数に取る継続とみなすことができる。!!

Values $\boxed{\Psi \vdash v : \tau}$

$$\Psi \vdash n : \texttt{int} \quad \text{(S-INT)}$$

$$\Psi \vdash ll : \Psi(ll) \quad \text{(S-LAB)}$$

Operands $\boxed{\Psi; \Gamma \vdash v : \tau}$

$$\Psi; \Gamma \vdash r : \Gamma(r) \quad \text{(S-REG)}$$

$$\frac{\Psi \vdash v : \tau}{\Psi; \Gamma \vdash v : \tau} \quad \text{(S-VAL)}$$

$$\frac{\Psi; \Gamma \vdash v : \forall \alpha.\tau}{\Psi; \Gamma \vdash v : \tau[\tau'/\alpha]} \quad \text{(S-INST)}$$

Instructions $\boxed{\Psi \vdash i : \Gamma_1 \to \Gamma_2}$

$$\frac{\Psi; \Gamma \vdash v : \tau}{\Psi \vdash r_d{:=}v : \Gamma \to \Gamma[r_d : \tau]} \quad \text{(S-MOV)}$$

$$\frac{\Psi; \Gamma \vdash r_s : \texttt{int} \qquad \Psi; \Gamma \vdash v : \texttt{int}}{\Psi \vdash r_d{:=}r_s + v : \Gamma \to \Gamma[r_d : \texttt{int}]} \quad \text{(S-ADD)}$$

$$\frac{\Psi; \Gamma \vdash r_s : \texttt{int} \qquad \Psi; \Gamma \vdash v : \texttt{code}(\Gamma)}{\Psi \vdash \texttt{if } r_s \texttt{ jump } v : \Gamma \to \Gamma} \quad \text{(S-IF)}$$

Instruction Sequences $\boxed{\Psi \vdash I : \tau}$

$$\frac{\Psi; \Gamma \vdash v : \texttt{code}(\Gamma)}{\Psi \vdash \texttt{jump } v : \texttt{code}(\Gamma)} \quad \text{(S-JUMP)}$$

$$\frac{\Psi \vdash i : \Gamma \to \Gamma_2 \qquad \Psi \vdash I : \texttt{code}(\Gamma_2)}{\Gamma \vdash i; I : \texttt{code}(\Gamma)} \quad \text{(S-SEQ)}$$

$$\frac{\Psi \vdash I : \tau}{\Psi \vdash I : \forall \alpha.\tau} \quad \text{(S-GEEN)}$$

Register Files $\boxed{\Psi \vdash R : \Gamma}$

$$\frac{\forall r.\Psi \vdash R(r) : \Gamma(r)}{\Psi \vdash R : \Gamma} \quad \text{(S-REGFILE)}$$

Heaps $\boxed{\vdash H : \Psi}$

$$\frac{\begin{array}{c}\forall ll \in dom(\Psi).\Psi \vdash H(ll) : \Psi(ll) \\ FTV(\Psi(ll)) = \Phi\end{array}}{\vdash H : \Psi} \quad \text{(S-HEAP)}$$

Machine States $\boxed{\vdash M}$

$$\frac{\vdash H : \Psi \qquad \Psi \vdash R : \Gamma \qquad \Psi \vdash I : \texttt{code}(\Gamma)}{\vdash (H, R, I)} \quad \text{(S-MATCH)}$$