AT-TaPL 4: Typed Assembly Language pt.1

河原 悟

2017 年 6 月 4 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。

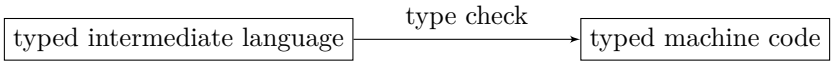

図 1　必要となる工程

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* :	i ::=		*instructions* :
	`r1` | `r2` | $\cdots$ | `rk`			r_d:=v	
v ::=		*operands* :		r_d:=r_s+v	
	n	*integer literal*		`if` r `jump` v	
	l	*label or pointer*	I ::=		*instruction sequences* :
	r	*registers*		`jump` v	
				$i; I$	

図 2　Instructions and operands for TAL-0

R ::=		*register files* :	H ::=		*heaps* :
	$\{\mathtt{r}_1 = v_1, \ldots, \mathtt{r}_k = v_k\}$			$\{l_1 = v_1, \ldots, l_k = v_k\}$	
h ::=		*heap values* :	M ::=		*machine states* :
	I	*code*			

`mov` や `add` のような表記より familiar な表記を用いる。

```
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
```