TALx86: A Realistic Typed Assembly Language

Greg Morrisett et. al.

December 3, 2000

□ Contents

- Motivations
- Compilation Process
- Popcorn
- TALx86

□ Motivation - The goal of TAL

- Provide a fully automatic way to verify that programs will not violate the type safety of the language.

- Provide a low-level, statically typed target language that is better suited than Java bytecodes for supporting a wide variety of sources langauges and a number of important optimizations.

□ Compilation Process

- Popcorn compiling

```
foo.tal, foo_i.tali, foo_e.tali
main.tal, main_i.tali, main_e.tali
```

- TALx86 type-checking
- Link verfing
- Assembling : MASM
- Linking with garbage collector and I/O

□ Popcorn

□ Popcorn

- Resembles C
- New features
  - Parametric polymorphism
  - Tuples, ML style union, ?struct
  - Exceptions
- Missing features
  - Pointer arithmetic, The address operator (&), Pointer cast

□ Popcorn example

- Tuple

```
*(int, int) x = new (3, 4)
*(int, int) dbl = new (x.1 + x.1, x.2 + x.2)
```

- Parametric polymorphism

```
?struct <‘a> list { ‘a hd; <‘a> list tl;}

<‘b>list map (‘b f(‘a), <‘a>list l){
         if (l == null) return null;
         return new list(f(l.hd), map(f, l.tl));
}
```

- Union

```
union tree { int Numleaf; *(tree,tree) Node };
```

□ TALx86

- Safe
- A realistic typed assembly language
- Executable on the Intel Pentium

□ TALx86 - features

- Import and export interface
- Type construct declaration
- Typing preconditions on code label
- Types on data labels
- Type coercions on instruction operands

□ Typing preconditions on code label

$$\forall \alpha_1 : \kappa_1 \cdots \alpha_m : \kappa_m . \{r_1 : \tau_1, \cdots, r_n : \tau_n\}$$

- α : type variable
- κ : kind
- r : register
- τ : type

□ TALx86 example - Preconditions on code label

```
int sum(int n){
  int i = n+1;
  int s == 0;
  while(--i > 0)
    s += i;
  return s;
}
```

□ TALx86 example - Preconditions on code label

```
int sum(int n){
  int i = n+1;
  int s == 0;
  while(--i > 0)
    s += i;
  return s;
}
```

⇒

```
sum:  {ecx:  B4, ebp:  {eax:  B4}}
      mov     eax,ecx    ; i = n
      inc     eax        ; ++i
      mov     ebx,0      ; s = 0
      jmp     test
body: {eax: B4, ebx: B4, ebp: {eax: B4}}
      add     ebx,eax    ; s += i
test: {eax: B4, ebx: B4, ebp: {eax: B4}}
      dec     eax        ; --i;
      cmp     eax,0      ; i > 0
      jg      body       ; if so, goto body
      mov     eax,ebx    ; otherwise
      jmp     ebp        ; return s
```

□ Stacks and Function Calls

- TALx86 uses stacks for function calls
- `sptr` σ: stack type of σ
- Example

$$\forall\rho\texttt{:Ts.}\{\texttt{eax: B4, esp: sptr B4::}\rho\}$$

□ Heap

- Tracking *initialization* of heap for safety

- Memory Allocation in Popcorn

```
*(int,int) x = new (3, 4);
```

- Memory Allocation in TALx86

```
malloc  8,<[:B4,:B4]> ; ^*[B4^u, B4^u]
mov     [eax+0],3     ; ^*[B4^rw, B4^u]
mov     [eax+4],4     ; ^*[B4^rw, B4^rw]
```

□ Arrays

- Runtime array size check

- Popcorn arrays: Size followed by the array elements
  - `S(s)` : Type represents array size `s`
  - `array(s,B4`^{rw}`)` : Type represents array elements

□ Sums and Recursive Types

- Popcorn list type

```
?struct int_list{
  int hd;
  int_list tl;
}
```

- TALx86 list type

  `type <int_list:T4 = ^.(0)*[B4`^{rw}`,‘int_list`^{rw}`]>`

- Coercion : `int_list` $\rightarrow$ `null * int_list`

□ Making Types Smaller

- Branch : Both code label have same precondtion

- Type abstraction

  type <str = $\exists$s:Sint.^*[S(s)^r,array(s,B1^{rw})]>

  type <F = λ ret:T4 ρ:Ts.
      $\{$esp: sptr $\{$eax: ret, esp: sptr $\rho\}$::$\rho\}$>

□ Check Point

- What's key idea?
- Do we really need it?
- Can we improve it?
- Do we understand it?
- Is anyone skeptical about it?
- What are alternatives?