Détection des types dans un programme

Matthieu Renard

9 avril 2014

1 Exemples de programmes

1.1 Détection de pointeurs

On détermine qu'une variable est un pointeur s'il y a déréférencement :

```c
#include <stdlib.h>


int main(int argc, char *argv[])
{
        int a = 0;
        int *b = &a;
        int c = *b;

        return 0;
}
```

```
00000000 <main>:
   0:   push   ebp
   1:   mov    ebp,esp
   3:   sub    esp,0x10
   6:   mov    DWORD PTR [ebp-0xc],0x0 ;
      a = 0
   d:   lea    eax,[ebp-0xc]
  10:   mov    DWORD PTR [ebp-0x4],eax ;
     b = &a
  13:   mov    eax,DWORD PTR [ebp-0x4]
  16:   mov    eax,DWORD PTR [eax]
  18:   mov    DWORD PTR [ebp-0x8],eax ;
     c = *b
  1b:   mov    eax,0x0
  20:   leave
  21:   ret
```

1.2 Détection de tableaux

```c
#include <stdlib.h>

int main(int argc, char *argv[])
{
        int tab[10];
        int i;

        for (i = 0 ; i < 10 ; i++)
        {
                tab[i]= 0;
        }

        return EXIT_SUCCESS;
}
```

```
00000000 <main>:
   0:   push   ebp
   1:   mov    ebp,esp
   3:   sub    esp,0x30
   6:   mov    DWORD PTR [ebp-0x4],0x0 ;
      i = 0
   d:   jmp    1e <main+0x1e>
   f:   mov    eax,DWORD PTR [ebp-0x4]
  12:   mov    DWORD PTR
     [ebp+eax*4-0x2c],0x0 ; tab[i] = 0
  19:
  1a:   add    DWORD PTR [ebp-0x4],0x1 ;
     i++
  1e:   cmp    DWORD PTR [ebp-0x4],0x9
  22:   jle    f <main+0xf> ; i <= 9 ?
  24:   mov    eax,0x0
  29:   leave
  2a:   ret
```

1.3 Détection de structures

1.3.1 En paramètre à une fonction

```c
#include <stdlib.h>

struct S
{
        int a;
        int *b;
};

void init_struct(struct S *s)
{
        s->a = 0;
        s->b = NULL;
}

int main(int argc, char *argv[])
{
        struct S s;

        init_struct(&s);

        return EXIT_SUCCESS;
}
```

```
00000000 <init_struct>:
   0:   push   ebp
   1:   mov    ebp,esp
   3:   mov    eax,DWORD PTR [ebp+0x8]
   6:   mov    DWORD PTR [eax],0x0      ;
       s->a = 0
   c:   mov    eax,DWORD PTR [ebp+0x8]
   f:   mov    DWORD PTR [eax+0x4],0x0  ;
       s->b = NULL
  16:   pop    ebp
  17:   ret

00000018 <main>:
  18:   push   ebp
  19:   mov    ebp,esp
  1b:   sub    esp,0x14
  1e:   lea    eax,[ebp-0x8]
  21:   mov    DWORD PTR [esp],eax
  24:   call   25 <main+0xd>          ;
      init_struct(&s)
  29:   mov    eax,0x0
  2e:   leave
  2f:   ret
```

1.3.2 Dans un tableau

```c
#include <stdlib.h>


struct S
{
        int a;
        int *b;
};

int main(int argc, char*argv[])
{
        struct S tab[10];
        int i;

        for (i = 0 ; i < 10 ; i++)
        {
                tab[i].a = 0;
                tab[i].b = NULL;
        }

        return EXIT_SUCCESS;
}
```

```
00000000 <main>:
   0:   push   ebp
   1:   mov    ebp,esp
   3:   sub    esp,0x60
   6:   mov    DWORD PTR [ebp-0x4],0x0 ;
        i = 0
   d:   jmp    29 <main+0x29>
   f:   mov    eax,DWORD PTR [ebp-0x4]
  12:   mov    DWORD PTR
        [ebp+eax*8-0x54],0x0 ; tab[i].a = 0
  19:
  1a:   mov    eax,DWORD PTR [ebp-0x4]
  1d:   mov    DWORD PTR
        [ebp+eax*8-0x50],0x0 ; tab[i].b =
        NULL
  24:
  25:   add    DWORD PTR [ebp-0x4],0x1 ;
        i++
  29:   cmp    DWORD PTR [ebp-0x4],0x9
  2d:   jle    f <main+0xf> ; i <= 9 ?
  2f:   mov    eax,0x0
  34:   leave
  35:   ret
```

1.3.3 Détection d'une liste chainée

```c
#include <stdlib.h>

struct List
{
        void *val;
        struct List *next;
};

void init_list(struct List *l)
{
        l->next = l;
}
```

```
00000000 <init_list>:
   0:   push   ebp
   1:   mov    ebp,esp
   3:   mov    eax,DWORD PTR [ebp+0x8]
   6:   mov    edx,DWORD PTR [ebp+0x8]
   9:   mov    DWORD PTR [eax+0x4],edx ;
        l = l->next
   c:   pop    ebp
   d:   ret
```

1.3.4 Le problème des codes équivalents

```c
#include <stdlib.h>

struct S
{
        int a;
        int *b;
};

struct T
{
        struct S s;
        int c;
};

struct U
{
        int a;
        int *b;
        int c;
};

void init_S(struct S *s)
{
        s->a = 0;
        s->b = NULL;
}

void init_T(struct T *t)
{
        init_S(&(t->s));
        t->c = 0;
}

void init_U_ab(struct U *u)
{
        u->a = 0;
        u->b = NULL;
}

void init_U(struct U *u)
{
        init_U_ab(u);
        u->c = 0;
}
```

```
00000000 <init_S>:
   0:   push   ebp
   1:   mov    ebp,esp
   3:   mov    eax,DWORD PTR [ebp+0x8]
   6:   mov    DWORD PTR [eax],0x0 ;
        s->a = 0
   c:   mov    eax,DWORD PTR [ebp+0x8]
   f:   mov    DWORD PTR [eax+0x4],0x0 ;
        s->b = NULL
  16:   pop    ebp
  17:   ret

00000018 <init_T>:
  18:   push   ebp
  19:   mov    ebp,esp
  1b:   sub    esp,0x4
  1e:   mov    eax,DWORD PTR [ebp+0x8]
  21:   mov    DWORD PTR [esp],eax
  24:   call   25 <init_T+0xd> ;
        init_S(&(t->s))
  29:   mov    eax,DWORD PTR [ebp+0x8]
  2c:   mov    DWORD PTR [eax+0x8],0x0 ;
        t->c = 0
  33:   leave
  34:   ret

00000035 <init_U_ab>:
  35:   push   ebp
  36:   mov    ebp,esp
  38:   mov    eax,DWORD PTR [ebp+0x8]
  3b:   mov    DWORD PTR [eax],0x0 ;
        u->a = 0
  41:   mov    eax,DWORD PTR [ebp+0x8]
  44:   mov    DWORD PTR [eax+0x4],0x0 ;
        u->b = NULL
  4b:   pop    ebp
  4c:   ret

0000004d <init_U>:
  4d:   push   ebp
  4e:   mov    ebp,esp
  50:   sub    esp,0x4
  53:   mov    eax,DWORD PTR [ebp+0x8]
  56:   mov    DWORD PTR [esp],eax
  59:   call   5a <init_U+0xd> ;
        init_U_ab(u)
  5e:   mov    eax,DWORD PTR [ebp+0x8]
  61:   mov    DWORD PTR [eax+0x8],0x0 ;
        u->c = 0
  68:   leave
  69:   ret
```

1.3.5 Différenciation de structures

```c
#include <stdlib.h>

struct S
{
        int a;
        int *b;
};

struct T1
{
        struct S s;
        int c;
};

struct T2
{
        struct S s;
        int *c;
};

void init_S(struct S *s)
{
        s->a = 0;
        s->b = NULL;
}

void init_T1(struct T1 *t)
{
        init_S(&(t->s));
        t->c = 0;
}

void init_T2(struct T2 *t)
{
        init_S(&(t->s));
        t->c = NULL;
}
```

```
00000000 <init_S>:
   0:   push   ebp
   1:   mov    ebp,esp
   3:   mov    eax,DWORD PTR [ebp+0x8]
   6:   mov    DWORD PTR [eax],0x0 ;
        s->a = 0
   c:   mov    eax,DWORD PTR [ebp+0x8]
   f:   mov    DWORD PTR [eax+0x4],0x0 ;
        s->b = 0
  16:   pop    ebp
  17:   ret

00000018 <init_T1>:
  18:   push   ebp
  19:   mov    ebp,esp
  1b:   sub    esp,0x4
  1e:   mov    eax,DWORD PTR [ebp+0x8]
  21:   mov    DWORD PTR [esp],eax
  24:   call   25 <init_T1+0xd> ;
        init_S(&(t->s))
  29:   mov    eax,DWORD PTR [ebp+0x8]
  2c:   mov    DWORD PTR [eax+0x8],0x0 ;
        t->c = 0
  33:   leave
  34:   ret

00000035 <init_T2>:
  35:   push   ebp
  36:   mov    ebp,esp
  38:   sub    esp,0x4
  3b:   mov    eax,DWORD PTR [ebp+0x8]
  3e:   mov    DWORD PTR [esp],eax
  41:   call   42 <init_T2+0xd> ;
        init_S(&(t->s))
  46:   mov    eax,DWORD PTR [ebp+0x8]
  49:   mov    DWORD PTR [eax+0x8],0x0 ;
        t->c = NULL
  50:   leave
  51:   ret
```

2 Algorithme

input : Un programme binaire p
output : L'ensemble des types des variables du programme

types ← tableau dont les indices sont les adresses du programme;
pour *Chaque execution s de p* **faire**
  types ← typer_execution(*s*, *types*);
fin