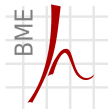


Fordító részei

Kód visszafejtés.



Híradástechnikai Tanszék

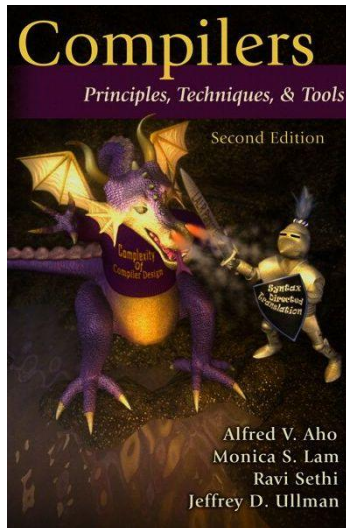
Izsó Tamás

2016. szeptember 29.

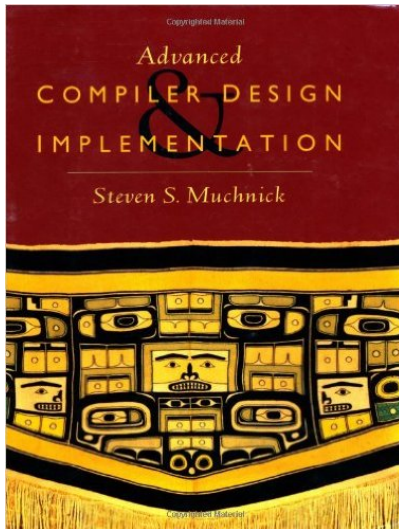
Section 1

Fordító részei

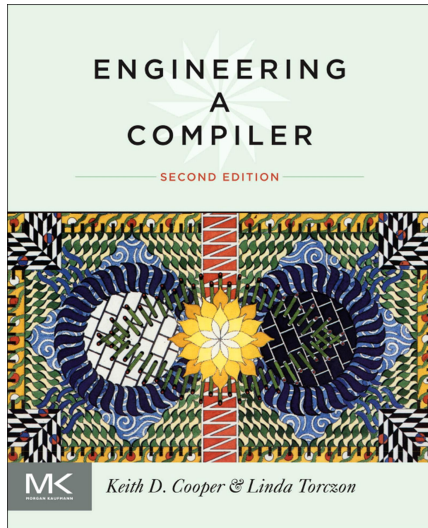
Irodalom



Irodalom



Irodalom



Reverse Compilation Techniques

by

Cristina Cifuentes

Doctor of Philosophy

at the

QUEENSLAND UNIVERSITY OF TECHNOLOGY

July 1994

Miért kell ismerni a fordítót?

- általában a fordító által generált kódot kell visszafejteni;
- ha ismerjük a fordító által alkalmazott kódtranszformációt, hamarabb rájöhethünk az eredeti tartalomra;
- tesztelni kell a fordítót;
- **mert a fordító is azokat az algoritmusokat használja a kód elemzéshez, melyeket a modern decompiler-ek.**
(Miért, amikor a fordító rendelkezésére áll az eredeti forrás?)

Fordító decompiler párhuzamok

Fázis	fordító	decompiler
előfeldolgozó	makrók kifejtése	input file részeinek a beolvasása
lexikai analízis	tokenizálás	assembly utasítás generálás
szintaktikai analízis	absztrakt szintaxisfa készítés	absztrakt szintaxisfa készítés
kód generálás	gépi kód előállítás	magas szintű nyelven írt program előállítása
optimalizálás	felesleges utasítások kiszűrése és a futás gyorsítása	kevesebb forrásprogram sor, még olvashatóbb formába

Ilfak Guilfanov (Hex-Rays) Decompilers and beyond cikke alapján.

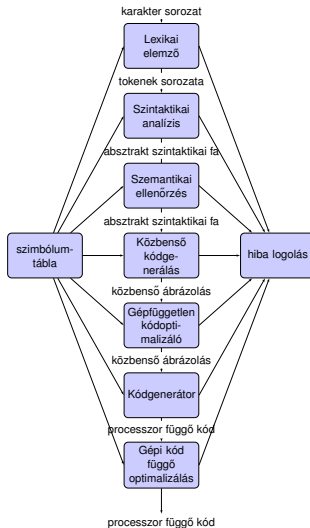
decompiler készítés szempontjai

- Információ gyűjtés a függvényhívási konvencióról, fordító kibocsátójáról, memória modellről, stb. Esetleg a felhasználó adja meg.
- Elméleti alapokon nyugvó problémák megoldása (adatfolyam analízis, regiszterek szerepe az adatterjedésben, alias analízis, use-def lánc, eredményt nem befolyásoló utasítások kiszűrés, adatok élettartama, vezérlésfolyam gráf analízis).
- Nem megoldható problémákra heurisztikák keresése (indirekt ugrások, függvényhatárok, és függvényparaméterek megállapítása).
- Beavatkozási lehetőséget adni a felhasználónak a disassembler által nem kezelhető problémák megoldására.
- Interaktív felhasználói beavatkozás biztosítása.

decompiler működésének a lépései

- Gépi utasítások visszafejtés.
- Gépi utasítás mikrokóddá alakítása. Algoritmusnak könnyebb figyelembe venni az utasítások szemantikáját. Például `pop eax` utasítás csökkenti az `esp` regiszter értékét.
- Lokális optimalizálás egyszerűsíti az utasításokat, figyelembe veszi `peephole` optimalizáció hatását.
- Globális optimalizáció. Adatfolyam analízis a regiszterek szerepének a tisztázásában. Függvényhívás és paraméter átadás analízis. Pointer és alias analízis. Függvény stack frame határainak a megállapítása.
- Lokális adatok feltérképezése.
- Vezérlésszerkezet feltérképezése. A feltételes elágazások alapján az (`if`, `switch`, `for`, `stb...`) utasítások előállítása.
- Pszeudokód generálás.
- Pszeudokód transzformáció, hogy még olvashatóbb legyen a program.
- Típusok felderítése.

Fordító részei

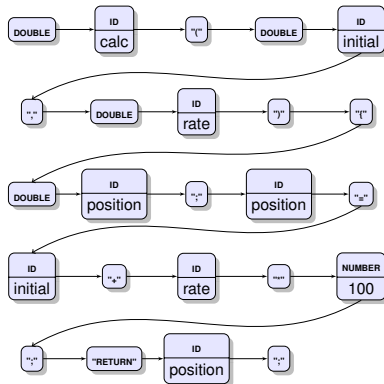


Lexikai elemző

```
1  double calc( double initial, double rate )
2  {
3      double position;
4      position = initial+rate*100;
5      return position;
6  }
```

Lexikai elemző

```
double calc( double initial, double rate ) \n
{ \n double position; \n position = initial+
rate*100; \n return position; \n } \n
```



Token

- token – szintaktikai kategória, csoport;
 - élő nyelvben: ige, főnév, melléknév, tárgy;
 - programozási nyelvekben: egész konstans, valós konstans, string, azonosító, operátor, kulcsszó, írásjelek;
- lexéma – konkrét megvalósulása egy tokennek;
- attribútum – tokenhez rendelt tulajdonság, érték
 - elmaradhat (pl. T_FOR)
 - típus
 - egész konstans , valós konstans , sztring értéke;
 - méret;
 - élettartam;
 - láthatóság
 - stb.
- nem token: szóköz, megjegyzés

Blokk struktúra

```
int main()
{
    int a=1;
    int b=1;
    {
        int a=3;
        printf("%d %d\n",a,b);
    }
    {
        int b=4;
        printf("%d %d\n",a,b);
    }
    printf("%d %d\n",a,b);
    return 0;
}
```

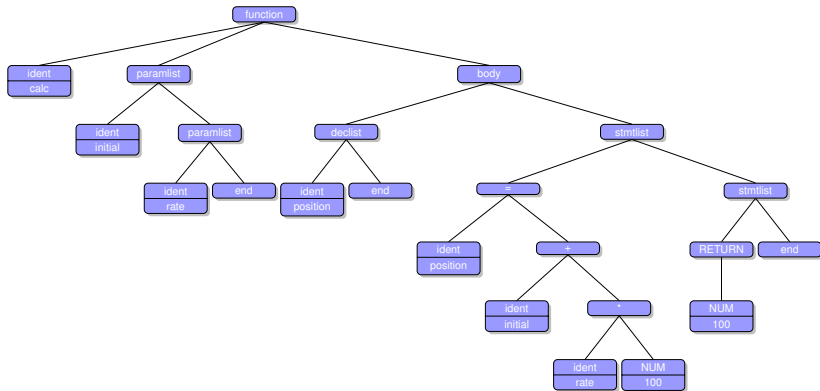
Szimbólum tábla

- a tokenizálás során keletkezett azonosítókat, és az azokhoz kapcsolódó attribútumokat tárolja;
- megőrzi a blokkstruktúrát;
- OOP esetén a származtatást;
- gyorsan lehet benne keresni;
 - sokszor a kulcsszavakat is tartalmazza (pl. for, while).

Szintaktikai analízis

- lexikai elemzés után tokenek sorozatát kapjuk;
- cél a program szerkezetének az értelmezése a környezetfüggetlen nyelvtani szabályok (context-free grammar CFG) alapján (műveletek, precedencia);
- cél a hibák felderítése;
- absztrakt szintaxisfa (abstract syntax tree – AST) elkészítése.

Szintaktikai analízis eredménye



Szemantikai analízis célja

- Olyan hibák felderítése, amit a CFG-ben nehéz, vagy nem lehet megadni.
- További információk gyűjtése.

```

1  extern void f(int n);
2  double calc( )
3  { int *p1, *p2;
4      int i, j;
5      ....
6      p1 = p1 + i;      /* OK */
7      p1 = p1 + p2; /* Hiba */
8      if( f(*p1) )      /* Hiba f() void */
9      {
10         i = j[2];      /* Hiba */
11         i = j[p1];      /* OK */
12     }
13     /* Nincs visszatérési érték */
14 }
```