RECONSTRUCTION OF BASE DATA TYPES IN DECOMPILATION

Peter Matula
Master Degree Programme (2), FIT BUT
E-mail: xmatul01@stud.fit.vutbr.cz

Supervised by: Jakub Křoustek
E-mail: ikroustek@fit.vutbr.cz

Abstract: This paper describes base data type reconstruction method in the decompilation problem. It presents a state of the art approach based on data-flow analysis, its improvements and adjustments in order to work as a part of retargetable decompiler developed within the Lissom project at FIT BUT. The final iterative algorithm operating on the LLVM IR and using results of other analyses is shown.

Keywords: reverse engeneering, decompiler, decompilation, data type analysis, Lissom

1 ÚVOD

Spätný preklad je programová transformácia, ktorej cieľom je zo vstupného programu vytvoriť kód vo vyššom programovacom jazyku. Proces začína dekódovaním strojového kódu a vytvorením jeho internej reprezentácie. Jadrom prekladu je rada analýz, ktorých cieľom je získanie alebo odvodenie maximálneho množstva vysokoúrovňových informácií. Výstupom je kód, ktorý by mal byť v ideálnom prípade čitateľnejší ako jazyk symbolických inštrukcii.

Informácie o datových typoch sú jednou z hlavných vlastností, ktoré odlišujú nízkoúrovňový strojový kód od vysokoúrovňového zdrojového kódu. Tento článok sa zaoberá návrhom a implementáciou analýzy jednoduchých datových typov implementovanej ako súčasť rekonfigurovateľného spätného prekladača projektu Lissom [3].

2 ANALÝZA JEDNODUCHÝCH DATOVÝCH TYPOV

Základy typovej analýzy pri spätnom preklade položil Alan Mycroft [1], ktorý predstavil algoritmus založený na unifikácií typových termov odvodených zo strojových inštrukcií. Pôvodný algoritmus nedokázal rozlišovať znamienka celých čísiel, bol často divergentný a umožňoval rýchle šírenie konfliktov. Táto práca je založená na článku [2], ktorý pracuje na rovnakom princípe, ale pre riešenie typových obmedzení využíva analýzu toku dat. Zároveň odstraňuje niektoré nedostatky pôvodného prístupu.

2.1 ZDROJE INFORMÁCIÍ O TYPOCH

Pretože strojový jazyk nie je typovaný, je pred rekonštrukciou typov nutná identifikácia zdrojov typových informácií v sémantike samotných inštrukcií. Ich vlastnosti vytvárajú obmedzenia na možné typy operandov a výsledku, ktoré môžeme rozdeliť do nasledujúcich kategórií: obmedzenia registrov (napríklad registre pre desatinné čísla), obmedzenia inštrukcií (inštrukcie znamienkového delenia, atď.), obmedzenia príznakov a obmedzenia prostredia (napríklad volania knižničných funkcií).

2.2 ANALÝZA TOKU DAT

Analýza typov pridelí každému výskytu *objektu* (register, globálna/lokálna premenná, parameter/návratová hodnota funkcie) v programe datový typ $T_i = \langle \tau^{core}, \tau^{size}, \tau^{sign} \rangle$ vytvorený pomocou trojice

atribútov $core, size, sign$ (rovnica 1). Výsledný typ objektu je rovný $T = T_1 \sqcap \cdots \sqcap T_n$, kde $\sqcap$ je funkcia definovaná ako prienik jednotlivých atribútov (rovnica 2).

$$core \in \{integer, pointer, float\},\ size \in \{1,8,16,32,64\},\ sign \in \{signed, unsigned\} \tag{1}$$

$$T_1 \sqcup T_2 = \langle \tau_1^{core} \cap \tau_2^{core}, \tau_1^{size} \cap \tau_2^{size}, \tau_1^{sign} \cap \tau_2^{sign} \rangle \tag{2}$$

Ďalej analýza vytvorí zoznam binárnych ($T_{op1} \bigcirc T_{op2} \Leftrightarrow T_{dst}$), unárnych ($\bigcirc T_{op} \Leftrightarrow T_{dst}$) alebo kopírovacích ($T_{src} \Leftrightarrow T_{dst}$) *propagačných rovníc*, kde $\bigcirc$ je operácia korešpondujúca k strojovej inštrukcií ku ktorej rovnica patrí. Tieto propagačné pravidlá sú zostavené na základe operácií jazyka C (nízkoúrovňový, typovaný jazyk, ktorý má blízko k asembleru) a sú následne využité k šíreniu typových informácií v dvoch smeroch: od operandov k výsledku a od výsledku k operandom.

Pri propagácii *od operandov k výsledku* je na základe datových typov operandov vyvodený záver o type výsledku. Príklad propagácie atribútu *core* pre binárnu rovnicu sčítania $T_{u1}\ ADD\ T_{u2} \Leftrightarrow T_{u3}$ je zobrazený v rovnici 3. Jej prvá časť napríklad vraví, že ak môžu byť oba operandy celočíselné, potom môže byť aj výsledok celočíselný (zlomková čiara tu slúži ako implikácia). Analýza musí obsahovať podobné pravidlo pre každú jednu inštrukciu a všetky tri atribúty.

Princíp propagácie od *výsledku k operandom* je v mnohých prípadoch rovnaký ako v prechádzajúcom príklade. U niektorých inštrukcií ale nastane problém. Uvažujme opäť inštrukciu sčítania a predpokladajme, že typ výsledku môže byť ukazovateľ. Podľa štandardu jazyka C bude jeden z operandov taktiež ukazovateľ a druhý celé číslo. Existujú teda dve možnosti propagácie. Táto komplikácia je riešená použitím takzvaných *alternatívnych elementov*, zaznamenávajúcich vzťah medzi objektami operandov (rovnica 4).

$$\frac{integer \in \tau_{u1}^{core} \wedge integer \in \tau_{u2}^{core}}{integer \in \tau_{u3}^{core}},\ \frac{pointer \in \tau_{u1}^{core} \wedge integer \in \tau_{u2}^{core}}{pointer \in \tau_{u3}^{core}},\ \frac{integer \in \tau_{u1}^{core} \wedge pointer \in \tau_{u2}^{core}}{pointer \in \tau_{u3}^{core}} \tag{3}$$

$$\frac{ptr \in \tau_{u3}^{core}}{\tau_{u1}^{core} \leftarrow \tau_{u1}^{core} \cup \{pointer_{u1}^{u3}?[u2]\} \ \wedge\ \tau_{u2}^{core} \leftarrow \tau_{u2}^{core} \cup \{pointer_{u2}^{u3}?[u1]\}} \tag{4}$$

Výsledný algoritmus zobrazený na zdrojovom kóde 1 začína vytvorením objektov a typov ich výskytov na základe inštrukcií programu. Následne vytvorí rovnice a vykoná počiatočné spojenie všetkých výskytov každého objektu. Algoritmus následne vstúpi do cyklu, ktorý sa bude opakovať až kým dve, po sebe nasledujúce iterácie nebudú mať rovnaký výsledok. V iterácií sa vykonajú obe propagácie a následné zjednotenie výsledkov vracajúce $true$ ak sa aspoň jeden datový typ nejakého objektu zmenil. V opačnom prípade vráti $false$ a cyklus skončí.

3 IMPLEMENTÁCIA ANALÝZY PRE PROJEKT LISSOM

V predchádzajúcej kapitole bol predstavený obecný framework analýzy toku dat určenej pre rekonštrukciu jednoduchých datových typov. Pri jej implementácii do existujúceho projektu spätného prekladača bolo navrhnutých niekoľko zmien a vylepšení popísaných v tejto kapitole.

Analýza začína *prvým priechodom* inštrukciami programu zhora-dole. Každá inštrukcia je preskúmaná a sú vytvorené nové výskyty pre všetky objekty s ktorými pracuje. Táto inicializácia sa od predchádzajúcej odlišuje v dvoch bodoch: 1) Propagačná rovnica je vytvorená len pre tie inštrukcie, cez ktoré je naozaj možné šírenie typovej informácie. Z ostatných je možné typy odvodiť priamo. 2) Počiatočná inicializácia typov výskytov je špecifická pre každý typ inštrukcie.

Algoritmus ďalej používa takzvané *lenivé pravidlá* (anglicky lazy evaluation) namiesto alternatívnych elementov, ktoré sú zbytočne komplikované. Pravidlá sa aktivujú len v prípade, že môžu byť aplikované. Propagácia pre sčítanie (rovnica 5) je vykonaná len v prípade, že je možné rozhodnúť, ktorý operand je ukazovateľ. Je aplikované vždy len jedno (prvé použiteľné) pravidlo.

$$\frac{\{ptr\} = \tau_{u3}^{core} \wedge ptr \notin \tau_{u1}^{core}}{\tau_{u2}^{core} \leftarrow \{ptr\}}, \frac{\{ptr\} = \tau_{u3}^{core} \wedge ptr \notin \tau_{u2}^{core}}{\tau_{u1}^{core} \leftarrow \{ptr\}}, \frac{\{ptr\} = \tau_{u3}^{core}}{\tau_{u1}^{core} \leftarrow \{integer\} \wedge \tau_{u2}^{core} \leftarrow \{integer\}} \quad (5)$$

Rekonfigurovateľný spätný prekladač projektu Lissom transformuje strojový kód na LLVM IR reprezentáciu. Na nej sú následne vykonávané všetky analýzy vrátane analýzy typov, ktorá je zaradená medzi poslednými. V čase jej behu už IR obsahuje množstvo vysokoúrovňových informácií o programe, vrátane jeho rozdelenia na funkcie. Kedže pôvodný algoritmus s funkciami nepočítal, je nutná jeho modifikácia. Nový algoritmus v jednom okamihu nepracuje na všetkých inštrukciách programu, ale len na tých, patriacich aktuálne spracovávanej funkcii. Pri vstupe do funkcie sú sledované aj typy jej parametrov a pri výstupe typ návratovej hodnoty. Nový algoritmus implementovaný podľa tohto návrhu je zobrazený na zdrojovom kóde 2.

```
objects = makeObjects(instructions);
equations = makeEquations(instructions);
objects.joinAll();

flag = true;
while (flag)
{
   flag = false;

   equations.opsToDstPropagation();
   flag |= objects.joinAll();
   equations.dstToOpsPropagation();
   flag |= objects.joinAll();
}
```

Zdrojový kód 1: Pôvodný algoritmus rekonštrukcie základných datových typov.

```
forall f in functions do:
   TypeFnc &tf=typeFnc.push_back(TypeFnc(f));
   forall i in instructions do:
      if f.belong(i) then tf.addInstr(i)
   tf.initializeArguments(); tf.firstPass();
   tf.mergeObjects(); tf.joinAllObjects();
flag = true;
forall tf in typeFunctions do:
   while (flag) do:
      flag = false;
      equations.opsToDstPropagation();
      flag |= objects.joinAll();
      equations.dstToOpsPropagation();
      flag |= objects.joinAll();
```

Zdrojový kód 2: Navrhnutý algoritmus rekonštrukcie základných datových typov.

4 ZÁVER

V tomto článku bol predstavený iteratívny, konvergentný algoritmus rekonštrukcie jednoduchých datových typov založený na analýze toku dat. Boli vysvetlené jeho vylepšenia a úpravy nutné pre jeho zapracovanie do existujúceho spätného prekladača projektu Lissom. Výsledný algoritmus by mal byť schopný perfektne rekonštruovať jednoduché datové typy za predpokladu, že vstupný program dodržiava štandard jazyka C. Algoritmus bude základom ďalšej práce na rekonštrukcií zložených datových typov.

POĎAKOVANIE

Tento príspevok vznikol za podpory grantu TAČR TA01010667 a výzkumného zámeru MSM0021630528.

REFERENCE

[1] Mycroft, A.: Type-Based Decompilation. In: Programming Languages and Systems, 8th European Symposium on Programming, Amsterdam, Springer 1999, ISBN 3-540-65699-5

[2] Dolgova, E. N., Chernov, A. V.: Automatic reconstruction of data types in the decompilation problem. Program. Comput. Softw., ročník 25, č. 2, 2009, s. 105–119, ISSN 0361-7688

[3] Ďurfina, L., Křoustek, J., Zemek, P., Kolář, D., Hruška, T., Masařík, K., Meduna, A.: Design of a Retargetable Decompiler for a Static Platform-Independent Malware Analysis. In: The 5th International Conference on Information Security and Assurance, s. 72–86, Springer Verlag, 2011, ISBN 978-3-642-23140-7