Bootstrapping

Компајлери
Миле Јованов

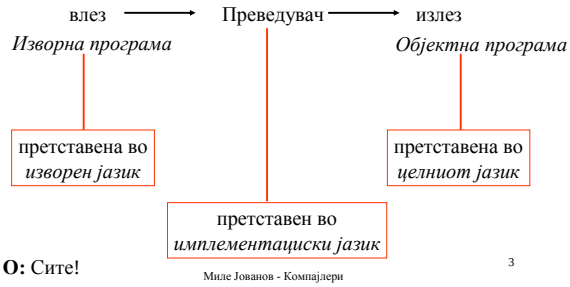
Денес...

- Bootstrapping

- Серија на себе-содржани процеси кои продолжуваат без помош однадвор

Терминологија

П: Кои програмски јазици играат улога на сликава?

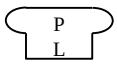


О: Сите!

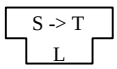
Плочка Дијаграми

- дијаграми кои се состојат од множество на "пазли" кои можеме да ги користиме за резонирање околу јазичните процесори и програмите
- различни видови на парчиња
- правила за комбинирање (сите дијаграми не се "добро составени")

Програма P имплементирана во L



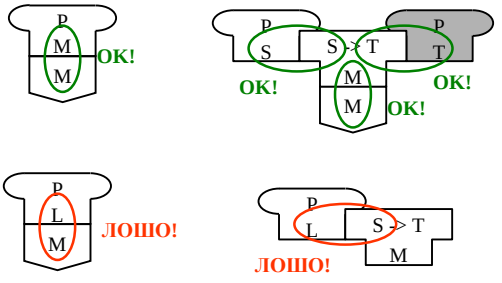
Преведувач имплемент. во L



Машина имплем. во харверот

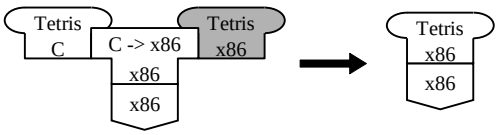
Јазичен интерпретер во L

Плочка Дијаграми: Правила на комбинирање



Компајлирање

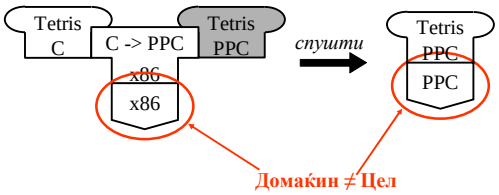
Пример: Компајлирање на C програма на x86 машина



Вкрстена компилација

Пример: C “вкрстен компајлер” од x86 за PPC

Вкрстен компајлер е компајлер кој работи на една машина (***машината домаќин***), а произведува код за друга машина (***целната машина***).

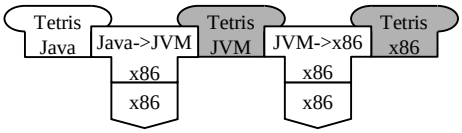


Домаќин ≠ Цел

П: Корисни ли се овие вкрстени компајлери? За што?

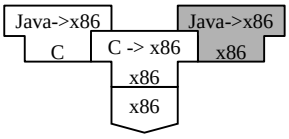
Двофазна компилација

Двофазен преведувач претставува композиција на два преведувачи. Излезот на првиот е влез за вториот преведувач.



Компајлирање на компајлер

Забележи: Компајлерот е програма.
Затоа може да биде влез за јазичен процесор.
Пример: компајлирање на компајлер.

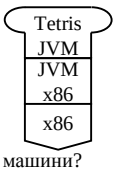


Интерпретери

Интерпретер е јазичен процеор импементиран во софтвер, како програма

Терминологија: *апстрактна (или виртуелна) машина* наспроти *реална машина*

Пример: Java Virtual Machine

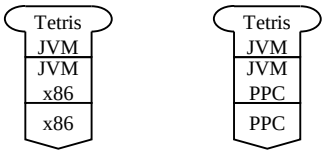


П: За што се корисни апстрактните машини?

Интерпретери

П: За што се корисни апстрактните машини?

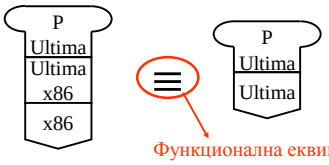
1) Апстрактните машини обезбедуваат подобра независност од платформа



Интерпретери

П: За што се корисни апстрактните машини?
2) Тие се корисни за тестирање и дебагирање.

Пример: Тестирање на процесорот “Ultima” со *хардверска емулација*



Функционална еквивалентност

Забелешка: нема потреба да го имплементираме емулаторот за Ultima во x86. Може да искористиме виш јазик и да го искомпајлираме.

Интерпретери наспроти компајлери

П: Компајлирање наспроти интерпретирање?

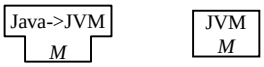
Компајлерите најчесто нудат предности кога
- програмите се пуштаат во продукција
- програмите се "повторливи"
- инструкциите на прог. јазик се сложени

Интерпретерите се најчесто подобар избор кога
- кога сме во фаза на развој/тестирање/дебагирање
- програмите се пуштаат еднаш и толку
- инструкциите на јазикот се едноставни
- брзината на извршување е засенета од други фактори
  - на пр. на веб сервер каде комуникациските трошоци се многу повисоки од брзината на извршување

Интерпретирачки компајлери

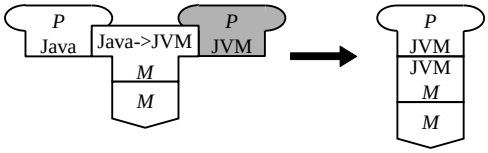
Зошто?
Компромис помеѓу (по)брза компилација и разумна изведба при извршување.
Како?
Употреби "меѓу јазик"
- по-виш од машински код => полесен да се компајлира до него
- по-ниж од изворен јазик => лесен да се имплементира како интерпретер

Пример: "Java Development Kit" за машината *M*



Интерпретирачки компајлери

Пример: Еве како го користиме "Java Development Kit" да стартуваме програма *P* напишана во Java



Преносливи компајлери

Пример: Две различни "Java Development Kit"-а

Kit 1:

Kit 2:

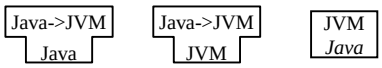
П: Кој е "по-преносли"?

Преносливи компајлери

- Во претходниот пример се виде дека преносливоста не функционира на принцип на „се или ништо"
- Треба да се зборува за „степен на преносливост" како процент на кодот кој треба да се пренапише кога се мигрира на неслична машина
- Во пракса 100% преносливост е невозможна

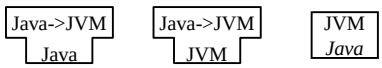
Пример: „пренослив" компајлерски сет (kit)

Пренослив компајлерски сет *:*

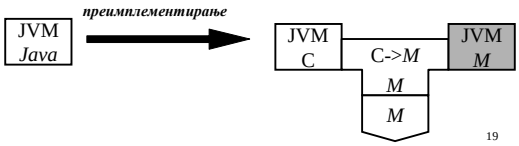


П: Нека сакаме да го стартуваме овој сет на некоја машина *M*. Како да го реализираме тоа? (со најмалку труд)

Пример: „преносliв" компајлерски сет (kit)



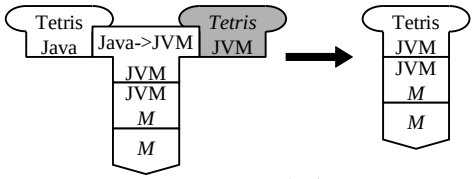
П: Нека сакаме да го стартуваме овој сет на некоја машина *M*. Како да го реализираме тоа? (со најмалку труд)



Пример: „преносliв" компајлерски сет (kit)

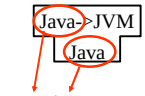
Сега го имаме ова:

Е сега, како ја извршуваме програмата Тетрис?



Bootstrapping

Нашиот „преносliв компајлерски сет":



П: Што можеме да направиме со компајлер напишан „во самиот себе"? Дали воопшто тоа може да се користи?

Bootstrapping

П: Што можеме да направиме со компајлер напишан „во самиот себе"? Дали воопшто тоа може да се користи?

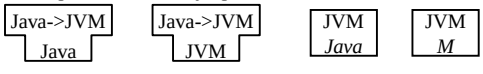
- Со имплемент. на компајлер во (подможество од) сопствениот јазик, се станува помалку зависен од целната платформа => по-преносliва имплементација.
- Но… "проблемот со кокошка и јајце"? Како да се заобиколи?

=> BOOTSTRAPPING: треба да се помачиме за првото "јајце".

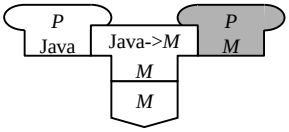
Има многу можни варијации како да се „покрене" компајлер напишан во сопствен јазик.

Bootstrapping на интерпретативен компајлер за да се генерира код за *M*

Нашиот „преносliв компајлерски сет":

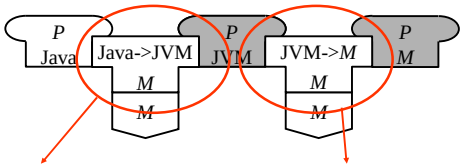


Цел: сакаме да добиеме "комплетно домашен" Java компајлер на машината *M*

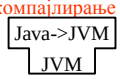


Bootstrapping на интерпретативен компајлер за да се генерира код за *M*

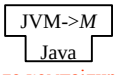
Идеја: ќе направиме двофазен компајлер Java -> *M*.



Ова ќе се направи со компајлирање



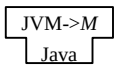
За ова ќе имплементираме



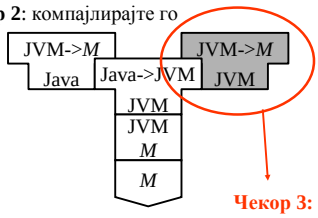
и ќе го компајлираме

Bootstrapping на интерпретативен компајлер за да се генерира код за *M*

Чекор 1: имплементирајте



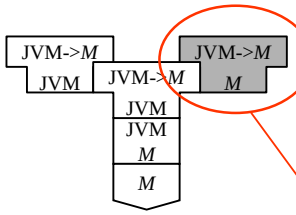
Чекор 2: компајлирајте го



Чекор 3: компајлирај го ова

Bootstrapping на интерпретативен компајлер за да се генерира код за *M*

Чекор 3: "Себе компајлирај" ја JVM-та (во JVM) компајлер

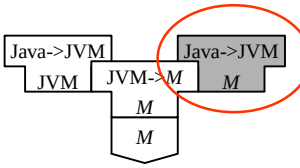


Ова е втората фаза на нашиот компајлер!

Step 4: искористи го ова за компајлирање на Java компајлерот

Bootstrapping на интерпретативен компајлер за да се генерира код за *M*

Чекор 4: Компајлирај го Java->JVM компајлерот во машински код



Прва фаза на нашиот компајлер!

ГОТОВО!

Комплетен Bootstrap

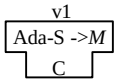
Комплетен bootstrap е неопходен кога се гради нов компајлер „од нула".

Пример:

Сакаме да имплементираме Ada компајлер за маш. *M*. Немаме пристап до никаков Ada компајлер (за *M*, ниту за друга машина).

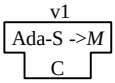
Идеја: Ada е многу голем, ќе го имплементираме компајлерот во подмножество од Ada и ќе направиме bootstrap од подмножество од Ada компајлерот во друг јазик. (на пр. C)

Чекор 1: Направи компајлер за Ada-S во друг јазик

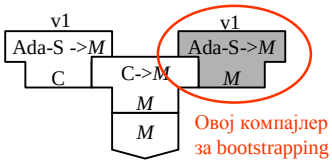


Комплетен Bootstrap

Чекор 1a: направи компајлер (v1) за Ada-S во друг јазик.



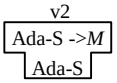
Step 1b: Компајлирај го v1 компајлерот на *M*



Овој компајлер може да се користи за bootstrapping на M не сакаме да се потпираме постојано на него!

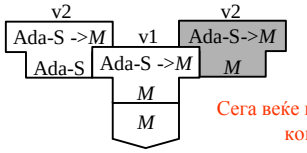
Комплетен Bootstrap

Step 2a: Имплементирај v2 за Ada-S компајлерот во Ada-S



П: Дали е тешко да се пренапише компајлер во Ada-S?

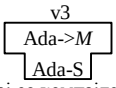
Чекор 2b: Компајлирај го v2 компајлерот со компајлерот v1



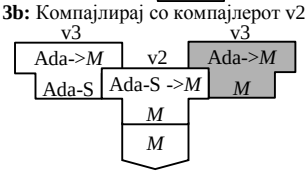
Сега веќе не ни треба достапен компајлер за C!

Комплетен Bootstrap

Чекор 3a: Направи комплетен Ada компајлер во Ada-S



Чекор 3b: Компајлирај со компајлерот v2

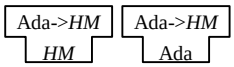


Од овој момент може да го одржуваме компајлерот во Ada.
Секоја наредна верзија v4,v5,... на компајлер во Ada може да се компајлира во претходната верзија.

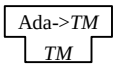
Делумен Bootstrap

П: Што ако имаме пристап за компајлер за нашиот јазик за некоја друга машина *HM*, а сакаме да направиме за *TM* ?

Имаме:



Сакаме:

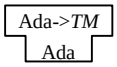


Идеја: Може да користиме вкрстена компилација од *HM* до *TM* за да направиме bootstrap на *TM* компајлерот.

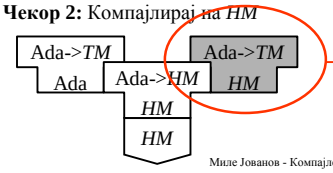
Делумен Bootstrap

Идеја: Можеме да користиме вкрстена компилација од *HM* до *M* за да направиме bootstrap *M* компајлерот.

Чекор 1: Имплементирај Ada->TM компајлер во Ada



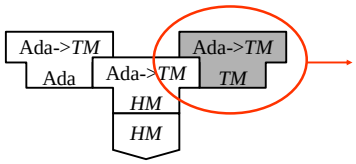
Чекор 2: Компајлирај на *HM*



Вкрстен компајлер: се извршува на *HM*, но продуцира *TM* код

Делумен Bootstrap

Чекор 3: Вкрстено компајлирај го *TM* компајлерот.



ГОТОВО!

Од сега може да правиме нови верзии на компајлерот комплетно на *TM*

Bootstrapping за подобрување на ефикасноста

Ефикасност на програмите и компајлерите:

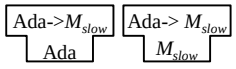
Ефикасност на програмите:
- употреба на меморија
- време на извршување

Ефикасност на компајлерите:
- Ефикасност на самиот компајлер
- Ефикасност на произведениот код

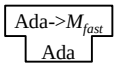
Идеја: Почнуваме од едноставен компајлер (кој генерира неефикасен код) и развиваме пософистицирана негова верзија. Потоа, може да користиме bootstrapping за да ја подобриме изведбата на компајлерот.

Bootstrapping за подобрување на ефикасноста

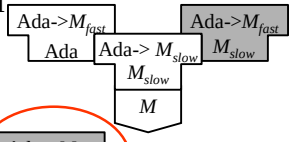
Имаме:



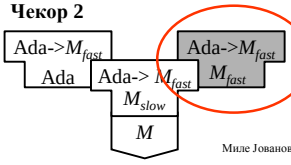
Имплементираме:



Чекор 1



Чекор 2



Брз компајлер кој продуцира брз код!

Заклучок

- За да се напише добар компајлер можно е прво да напишете неколку поедноставни
- Мора да мислите на изворниот јазик, целниот јазик и јазикот на имплементација.
- Работата на пишувачот на компајлер никогаш не е завршена, секогаш има верзија 1.x, па верзија 2.0 па …

За следниот час: