Theoretische Aspecten van Programmatuur

HC 2 (week 4)

Bob Diertens

Theory of Computer Science (TCS)

date: Maart, 1

version: February 28, 2017

Wat

- Bestudering programmeer-constructies.
- Uitbreiding PGA (vnl. basis-instructies)
- Functies
- Berekeningsmodellen voor functies
- Concurrency support
- Communicatie
- Berekeningsmodel voor objecten
- Object-orientatie
- Gereedschap: Generalisatie & Abstractie

Berekenings Modellen voor Functies

Introductie

Uitvoering van een Programma

- interpreteren
- vertalen
  interpreteren

Lab-sessie:

- Kennis werking interpretatie is vereist voor het schrijven van een programma.

Vertalen:

- Kennis werking vertaling is vereist voor het schrijven van een programma.
- Kennis werking interpretatie is vereist voor het schrijven van een programma.

Introductie (cont'd)

Functie Uitvoering mbv Stack

Niet transparant:

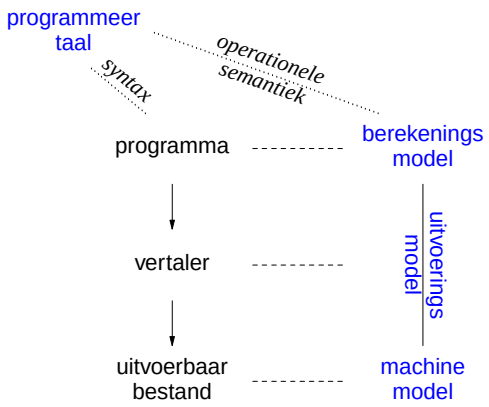
- Recursie niet te diep.
- Tail-recursie eliminatie.
- Geen echte stack.
  niet alleen top van stack (machine support)
- Optimalisatie
  - — niet alleen top van stack
  - — register gebruik
  - — ...
- In-line functie executie.
- ...

Waar staat dat beschreven?

Hoort dat bij de programmeertaal?

...?

Introductie (cont'd)



Berekenings Model

Wat is het berekenings model?

???

Hebben we een berekenings model nodig?

Berekenings Model (cont'd)

```
for (i = 0; i < 10; i ++) {
      .
      .
      .
}
```

Berekeningsmodel?

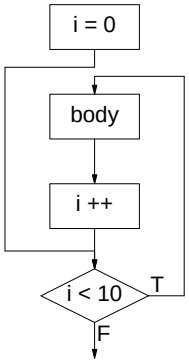
- — eerste keer conditie check?
- — manipulatie van `i` in body mogelijk?
- — waarde `i` na afloop?
- — alle talen hetzelfde?
- — wat als `int i = 0`?

```c
#include <stdio.h>
int main() {
     int i;
     for (i = 0; i < 10; i++) {
          printf("%d\n", i);
     }
}
```



```
.LC0:
          .string "%d\n"
          .text
.globl main
          .type    main, @function
main:
.LFB0:
          .cfi_startproc
          pushq     %rbp
          .cfi_def_cfa_offset 16
          .cfi_offset 6, -16
          movq      %rsp, %rbp
          .cfi_def_cfa_register 6
          subq      $16, %rsp
          movl      $0, -4(%rbp)
          jmp       .L2
.L3:
          movl      $.LC0, %eax
          movl      -4(%rbp), %edx
          movl      %edx, %esi
          movq      %rax, %rdi
          movl      $0, %eax
          call      printf
          addl      $1, -4(%rbp)
.L2:
          cmpl      $9, -4(%rbp)
          jle       .L3
          leave
          .cfi_def_cfa 7, 8
          ret
          .cfi_endproc
.LFE0:
```


Berekenings Model (cont'd)



Berekenings Model (cont'd)

Hoe komen we aan een berekenings model?

onderzoek, generalisatie en abstractie

Berekenings model voor functies

- functie aanroep essentieel onderdeel van programmeertaal
- mechanisme nodig voor berekenen van functies
- gebaseerd op mechanisme in machine model
- mechanisme in machine model maakt gebruik van bepaalde conventie
- beschouw aanroep sequentie voor de programmeertaal C
  [Johnson, Ritchie - technical report 102 - Bell Labs - 1981]
  + gegenereerde assembler code van een aantal C compilers
- baseer berekenings model op uitvoerings model

Assembler code

```
#include <stdio.h>
int add(int x, int y, int z) {
      int k;
      k = x + y + z;
      return k;
}
int main() {
      int x, y, z;
      int s;
      x = 3;
      y = 4;
      z = 5;
      s = add(x, y, z);
      printf("%d\n", s);
}
```



```
          .file     "fun.c"
          .text
.globl add
          .type     add, @function
add:
.LFB0:
          .cfi_startproc
          pushq     %rbp
          .cfi_def_cfa_offset 16
          .cfi_offset 6, -16
          movq      %rsp, %rbp
          .cfi_def_cfa_register 6
          movl      %edi, -20(%rbp)
          movl      %esi, -24(%rbp)
          movl      %edx, -28(%rbp)
          movl      -24(%rbp), %eax
          movl      -20(%rbp), %edx
          leal      (%rdx,%rax), %eax
          addl      -28(%rbp), %eax
          movl      %eax, -4(%rbp)
          movl      -4(%rbp), %eax
          leave
          .cfi_def_cfa 7, 8
          ret
          .cfi_endproc
.LFE0:
          .size     add, .-add
          .section  .rodata
.LC0:
          .string   "%d\n"
          .text
.globl main
          .type     main, @function
main:
.LFB1:
          .cfi_startproc
          pushq     %rbp
          .cfi_def_cfa_offset 16
          .cfi_offset 6, -16
          movq      %rsp, %rbp
          .cfi_def_cfa_register 6
          subq      $16, %rsp
          movl      $3, -16(%rbp)
          movl      $4, -12(%rbp)
          movl      $5, -8(%rbp)
          movl      -8(%rbp), %edx
          movl      -12(%rbp), %ecx
          movl      -16(%rbp), %eax
          movl      %ecx, %esi
          movl      %eax, %edi
          call      add
          movl      %eax, -4(%rbp)
          movl      $.LC0, %eax
          movl      -4(%rbp), %edx
          movl      %edx, %esi
          movq      %rax, %rdi
          movl      $0, %eax
          call      printf
          leave
          .cfi_def_cfa 7, 8
          ret
          .cfi_endproc
.LFE1:
          .size     main, .-main
          .ident    "GCC: (GNU) 4.4.7 20
          .section  .note.GNU-stack,"",@p
```


Model Functie Uitvoering

1. De argumenten voor de functie worden op de stapel (stack) gezet.
2. Het terugkeer-adres wordt op de stack gezet.
3. Controle wordt overgedragen aan de aangeroepen functie (program counter wordt aan het begin van de functie gezet). Argumenten zijn beschikbaar dmv referentie in de stack.
4. De inhoud van in de functie gebruikte registers worden bewaard op de stack.
5. Er wordt ruimte gereserveerd op de stack voor de lokale variabelen van de functie.
6. De eigenlijke functie wordt berekend.
7. De waarde die moet worden teruggeven aan de aanroeper wordt ergens opgeslagen (stack/register).
8. De ruimte op de stack wordt vrijgegeven en de opgeslagen waarden worden teruggezet in de registers.
9. Het terugkeer-adres wordt van de stack gehaald.
10. De controle wordt teruggegeven aan de aanroeper van de functie (program counter wordt op het terugkeer-adres gezet).
11. De teruggeef-waarde wordt opgehaald (van stack / uit register).
12. De argumenten van de aanroep worden van de stack gehaald en verdere berekening wordt voortgezet.

Model Functie Uitvoering (cont'd)

- Uitvoering dmv stack is iets te specifiek.
- Machine (model) kent al een framepointer.
- Wijst naar structuur (op stack).
- generaliseren

Generiek Model Functie Uitvoering

1. De arguments voor de functie en het terugkeer-adres worden in een structuur (frame) geplaatst.
2. De structuur wordt bewaard in een plaats die bereikbaar is voor de functie.
3. Een omgeving voor de functie wordt opgezet.
4. De controle wordt overgedragen aan de aangeroepen functie.
    1. De functie maakt eigen structuur voor opslag van lokale variabelen en inhoud van in functie gebruikte registers.
    2. De argument worden uit de structuur gehaald.
    3. De eigenlijk functie wordt berekend.
    4. De teruggeef-waarde wordt opgeslagen in structuur van aanroeper.
    5. De functie ruimt eigen structuur op.
    6. Het terugkeer-adres wordt uit de structuur gehaald en de controle wordt teruggeven aan de aanroeper.
5. De omgeving voor de function wordt opgeruimd.
6. De teruggeef-waarde wordt uit de structuur gehaald.
7. De structuur wordt opgeruimd.

Generiek Model Functie Uitvoering (cont'd)

- Structuren iets te concreet.
- abstraheren

Berekenings Model voor Functies

1. Aanroeper stelt argumenten en terugkeer-adres beschikbaar.
2. Een omgeving voor de functie wordt opgezet.
3. Controle wordt overgedragen aan de functie.
    1. Functie initialiseert.
    2. Functie verkrijgt argumenten.
    3. Functie voert berekening uit.
    4. Functie stelt teruggeef-waarde beschikbaar.
    5. Functie ruimt op.
    6. Controle wordt teruggeven.
4. Omgeving wordt opgeruimd.
5. Aanroeper ontvangt teruggeef-waarde.

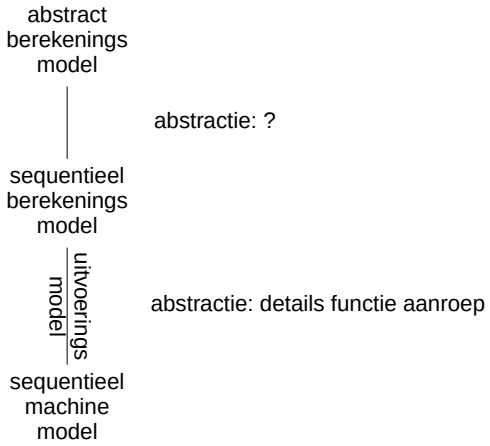
☞ Sequentieel Berekenings Model (voor functies)

Sequentieel Berekenings Model

- Een functie wordt uitgevoerd in een omgeving.

  zie ook bv
  [R.D. Tennent, *Principles of Programming Languages*, 1981]
  (formele semantiek programmeer-constructies)
- Uitleg functie uitvoering zonder implementatie details.
- ...

Verdere Abstractie



Analyse Functie Uitvoering

Aspecten

- aanroeper (caller)
- aangeroepene (callee)
- ...

Aspecten Functie Uitvoering

Sequentieel Berekenings Model Functies

1. Aanroeper stelt argumenten en terugkeer-adres beschikbaar.
2. Een omgeving voor de functie wordt opgezet.
3. Controle wordt overgedragen aan de functie.
    1. Functie initialiseert.
    2. Functie verkrijgt argumenten.
    3. Functie voert berekening uit.
    4. Functie stelt teruggeef-waarde beschikbaar.
    5. Functie ruimt op.
    6. Controle wordt teruggeven.
4. Omgeving wordt opgeruimd.
5. Aanroeper ontvangt teruggeef-waarde.

Gezien vanuit de aanroeper (1, 5)

Gezien vanuit de functie (3-2, 3-3, 3-4)

Rest?

Aspecten Functie Uitvoering (cont'd)

Rest draagt zorg voor:

- omgeving
- controle
- initialisatie
- opruiming

☞ Mechanisme dat de controle heeft over de uitvoering van de functie.

Tevens:

- recursieve functie-aanroep is mogelijk
- voor elke functie een nieuwe specifieke omgeving

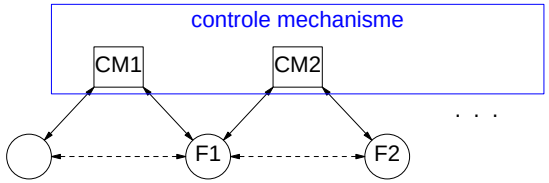
☞ Functie is een instantie (van een generieke omgeving).

Controle Mechanisme voor Functie Uitvoering

1. Aanroeper stuurt functie-naam en argumenten naar controle-mechanisme.
    1. Controle-mechanisme ontvangt functie-naam en argumenten.
    2. Controle-mechanisme creeert een **instantie** van de functie.
    3. Controle-mechanisme stuurt argumenten naar de functie-instantie.
        1. Functie-instantie ontvangt argumenten.
        2. Functie-instantie voert berekening uit.
        3. Functie-instantie stuurt waarde naar controle-mechanisme.
    4. Controle-mechanisme ontvangt waarde.
    5. Controle-mechanisme vernietigt instantie van functie.
    6. Controle-mechanisme stuurt waarde naar aanroeper.
2. Aanroeper ontvangt waarde en continueert.

Geabstraheerd van een mogelijke implementatie voor het berekenen van een functie.

Controle Mechanisme voor Functie Uitvoering (cont'd)



Controle Mechanisme voor Functie Uitvoering (cont'd)

Beperkingen:

- Aanroeper moet wachten op de waarde die wordt teruggegeven.
- Controle-mechanisme moet onmiddelijk in actie komen bij een functie-aanroep.

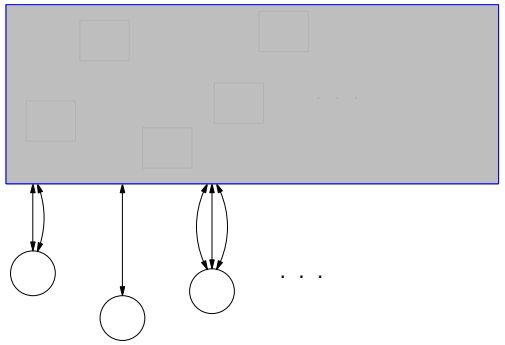
Vanuit controle-mechanisme zijn beperkingen niet nodig.

Opheffen beperkingen (generaliseren):

- Uitvoering van instructies mag doorgaan na een functie-aanroep tot het moment waarop de teruggeef-waarde nodig is.
- Controle-mechanisme mag wachten met uitvoering functie tot dat een bepaald criterium is bereikt, zoals
    - beschikbaar komen benodigde hulpmiddelen
    - teruggeef-waarde is nodig
    - . . .

Controle-mechanisme heeft nu de typische taken van een uitvoerings-regelaar (scheduler).

Uitvoerings-regelaar



Generalisatie

Abstractie

Uitvoerings-regelaar

Nieuwe mogelijkheden:

- meer dan een functie in wachtrij voor uitvoering
- uitvoering niet noodzakelijk in de volgorde van aanroep
- niet noodzakelijk te wachten op teruggeef-waarde
- intussen kunnen andere aanroepen gedaan worden

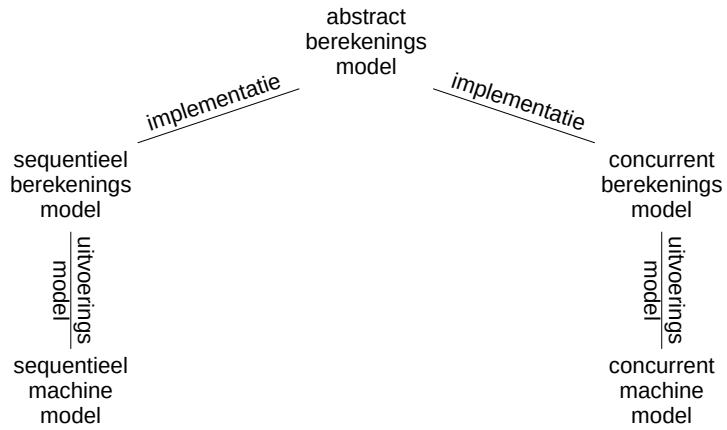
☞ concurrente berekening van functies

parallelle berekening - meerdere sequentiele instructie uitvoerders nodig

sequentiele berekening is: in rij van instructie geregelde uitvoering (inline scheduling)

☞ Abstract Berekenings Model (voor Functies)

Raamwerk



Concurrency

mogelijke implementatie:

- sequentieel + threads
- problemen met threads en compiler
  vanwege sequentieel uitvoerings model
- [P.A. Buhr, *Are Safe Concurrency Libraries Possible?*, 1995]
  [H.J. Boehm, *Threads Cannot Be implemented As A Library*, 2005]

beter:

- functionaliteit van threads naar de programmeertaal
- implementatie van threads naar machine model

MSP met Functies

Uitbreiding MSP met functies

Functies:

- parameters
- teruggeefwaarde
- lokale variabelen

Implementatie op basis van:

- eval (MSPea)
- argumenten via stack
- variabelen via stack

MSPfunc

`function fname(parameters):type string`

Defineert functie fname met optionele parameters en optioneel return type.

`string`
code die de functie uitvoert in {PGLA/PGLEc}.MSPfunc.

`parameters`
een door komma's gescheiden lijst van identifier:type.

De `string` wordt gecompileerd naar een molekuul en toegekend aan focus fname.

De `parameters` en het `type` van de functie worden toegevoegd aan het molekuul.

Als geen type (geen `:type`) dan is het type atom.

Als PGLEc de set van primitieve instructies is, dan ook voor de functie.
En PGLA in de andere gevallen.

MSPfunc (cont'd)

Extra:

— aan een parameter kan gerefereerd worden met `<>.identifier`.

— lokale variabelen kunnen worden toegevoegd met `<>.+identifier` en `<>.+identifier:type`.

— als de functie een return type heeft dan kan de instructie `return object/value` gebruikt worden.

— return instructie beeindigt de uitvoering van de functie

- NIET (PGLA)
- wel (PGLEc)

MSPfunc (cont'd)

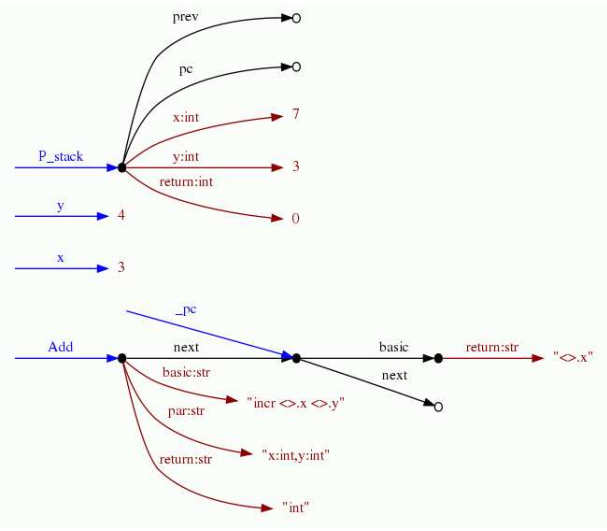
`fname(arguments)`
Voert de functie fname uit door het binden van de argumenten aan de parameters en het evalueren van het molekuul in focus fname.

`object = fname(arguments)`
Voert de functie fname uit door het binden van de argumenten aan de parameters en het evalueren van het molekuul in focus fname.
Na de uitvoering wordt de waarde uit de return instructie toegekend aan object.

MSPfunc (cont'd)

voorbeeld

```
function Add(x:int, y:int):int "incr <>.x <>.y; return <>.x";
x = 3;
y = 4;
x = Add(y, x);
```

MSPfunc with PGLA / PGLEc

PGLA

```
function Add(x:int, y:int):int "
- <>.y == 0;
#3;
return <>.x;
!;
incr <>.x;
decr <>.y;
\\\\#6"
```

PGLEc

```
function Add(x:int, y:int):int "
L0;
+ <>.y == 0 {;
      return <>.x;
}{;
      incr <>.x;
      decr <>.y;
      ##L0;
}"
```


CONTENTS

MSPfunc with PGLA / PGLEc 39
Theoretische Aspecten van Programmatuur 1
Wat 2
Introductie 4
Introductie (cont'd) 5
Introductie (cont'd) 6
Berekenings Model 7
Berekenings Model (cont'd) 8
Berekenings Model (cont'd) 11
Berekenings Model (cont'd) 12
Model Functie Uitvoering 15
Model Functie Uitvoering (cont'd) 16
Generiek Model Functie Uitvoering 17
Generiek Model Functie Uitvoering (cont'd) 18
Berekenings Model voor Functies 19
Sequentieel Berekenings Model 20
Verdere Abstractie 21
Analyse Functie Uitvoering 22
Aspecten Functie Uitvoering 23
Aspecten Functie Uitvoering (cont'd) 24
Controle Mechanisme voor Functie Uitvoering 25
Controle Mechanisme voor Functie Uitvoering (cont'd) 26
Controle Mechanisme voor Functie Uitvoering (cont'd) 27
Uitvoerings-regelaar 28
Uitvoerings-regelaar 29
Raamwerk 30
Concurrency 31
Uitbreiding MSP met functies 33
MSPfunc 34
MSPfunc (cont'd) 35
MSPfunc (cont'd) 36
MSPfunc (cont'd) 37