Deklarativ programmering

TDP007 Konstruktion av datorspråk
Föreläsning 7

LINKÖPINGS UNIVERSITET

Pontus Haglund
Institutionen för datavetenskap
2024-02-13

Deklarativ programmering

Hämta gärna filerna från kurssidan redan nu

LINKÖPINGS UNIVERSITET

Pontus Haglund
Institutionen för datavetenskap
2024-02-09

Deklarativ programmering

- Programmet specificerar vad som ska göras snarare än hur det ska göras.
- Exempel på språk som har stöd för deklarativ programmering:
  - logikspråk, t.ex. Prolog
  - funktionellt språk, t.ex. Scheme
  - restriktionsspråk (eng. *constraint*)
- Jämför med *imperativ programmering* eller *objektorienterad programmering*.

Exempel: Prolog

Programmet består av regler och fakta.

```
syskon(X,Y) :- förälder(Z,X), förälder(Z,Y).
förälder(X,Y) :- far(X,Y).
förälder(X,Y) :- mor(X,Y).

mor(tina,sally).
far(tom,sally).
far(tom,erica).
far(mike,tom).
```

Exempel: Prolog

Programmet består av regler och fakta.

```
syskon(X,Y) :- förälder(Z,X), förälder(Z,Y).
förälder(X,Y) :- far(X,Y).
förälder(X,Y) :- mor(X,Y).

mor(tina,sally).
far(tom,sally).
far(tom,erica).
far(mike,tom).
```

Att köra programmet innebär att ställa frågor som besvaras med utgångspunkt i de regler och fakta som gäller.

Exempel: Prolog

Programmet består av regler och fakta.

```
syskon(X,Y) :- förälder(Z,X), förälder(Z,Y).
förälder(X,Y) :- far(X,Y).
förälder(X,Y) :- mor(X,Y).

mor(tina,sally).
far(tom,sally).
far(tom,erica).
far(mike,tom).
```

Att köra programmet innebär att ställa frågor som besvaras med utgångspunkt i de regler och fakta som gäller.

```
?- syskon(sally,erica).
Yes
```

Exempel: Scheme

Programmet består av ett antal funktioner som definierar problemet matematiskt.

```
(define (fakultet n)
  (if (= n 0)
      1
      (* n (fakultet (1- n)))))

(define (kvadrat x)
  (* x x))
```

Exempel: Scheme

Programmet består av ett antal funktioner som definierar problemet matematiskt.

```
(define (fakultet n)
  (if (= n 0)
      1
      (* n (fakultet (1- n)))))

(define (kvadrat x)
  (* x x))
```

Att köra programmet innebär att konstruera uttryck vars värde beräknas med hjälp av funktionsdefinitionerna.

Exempel: Scheme

Programmet består av ett antal funktioner som definierar problemet matematiskt.

```
(define (fakultet n)
  (if (= n 0)
      1
      (* n (fakultet (1- n)))))

(define (kvadrat x)
  (* x x))
```

Att köra programmet innebär att konstruera uttryck vars värde beräknas med hjälp av funktionsdefinitionerna.

```
> (+ (fakultet 5) (kvadrat 2))
124
```

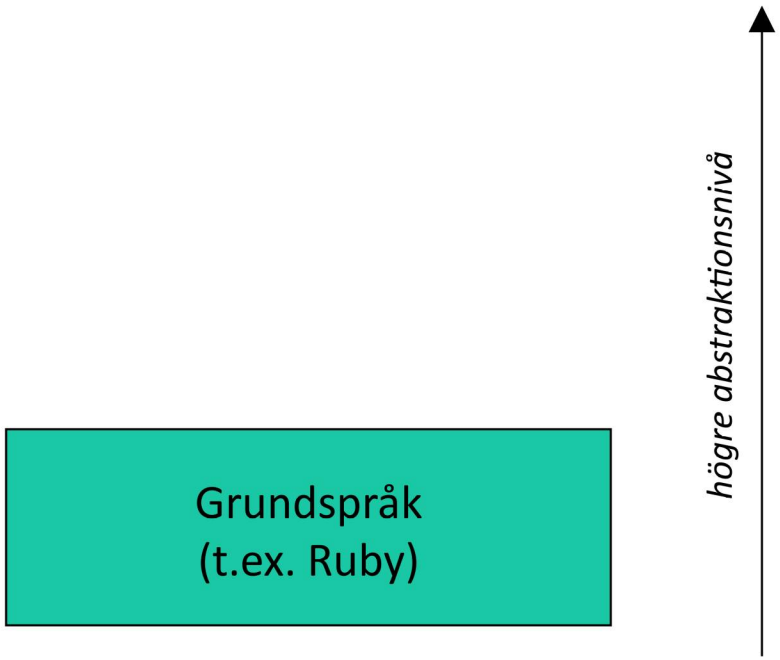
Exempel: HTML

Även om HTML inte är ett programspråk är det ett bra exempel på hur man kan definiera data deklarativt. Med HTML definierar vi (åtminstone som det var tänkt från början) endast innehåll och inte utseende.

```
<h2>Exempel</h2>

<p>Det finns många bra exempel på frukter.
<a href=”banan.html”>Bananen</a> t.ex. är en avlång
frukt med gult skal som är ganska god.</p>
```

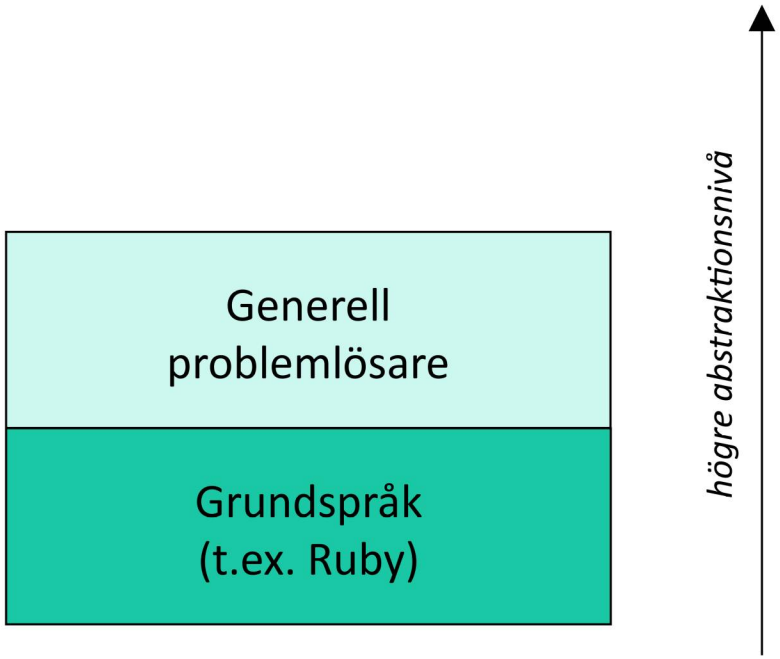
Vem gör grovjobbet?

Även om programmet uttrycks deklarativt måste ”någon” förr eller senare bestämma sig för hur problemet ska lösas.



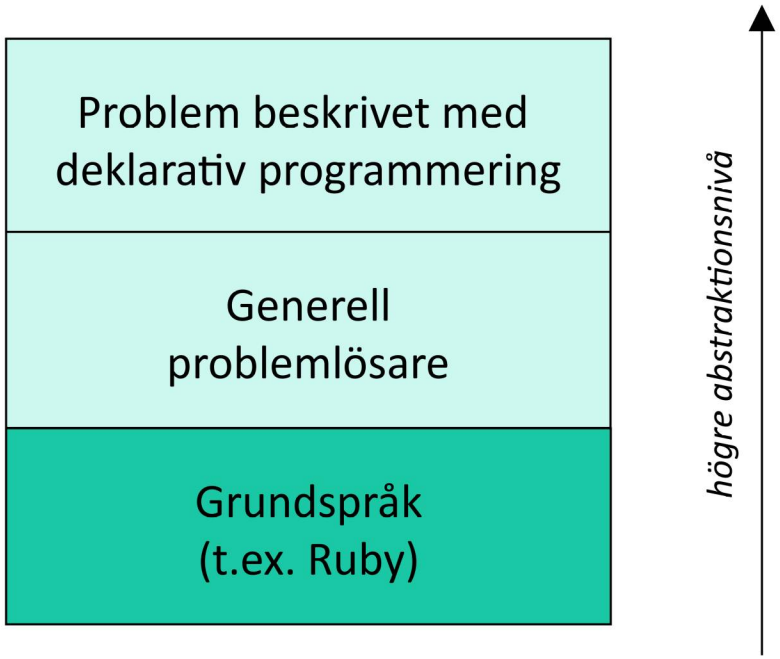
Vem gör grovjobbet?

Även om programmet uttrycks deklarativt måste ”någon” förr eller senare bestämma sig för hur problemet ska lösas.



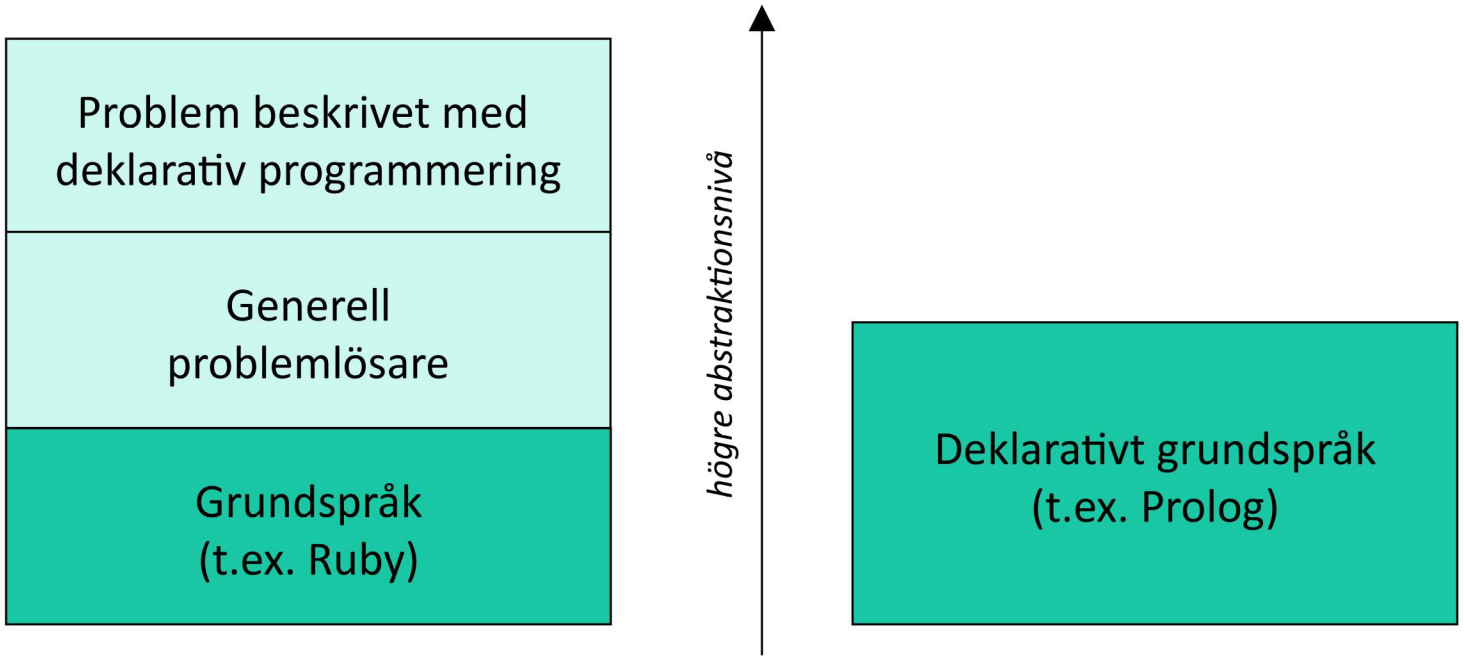
Vem gör grovjobbet?

Även om programmet uttrycks deklarativt måste ”någon” förr eller senare bestämma sig för hur problemet ska lösas.



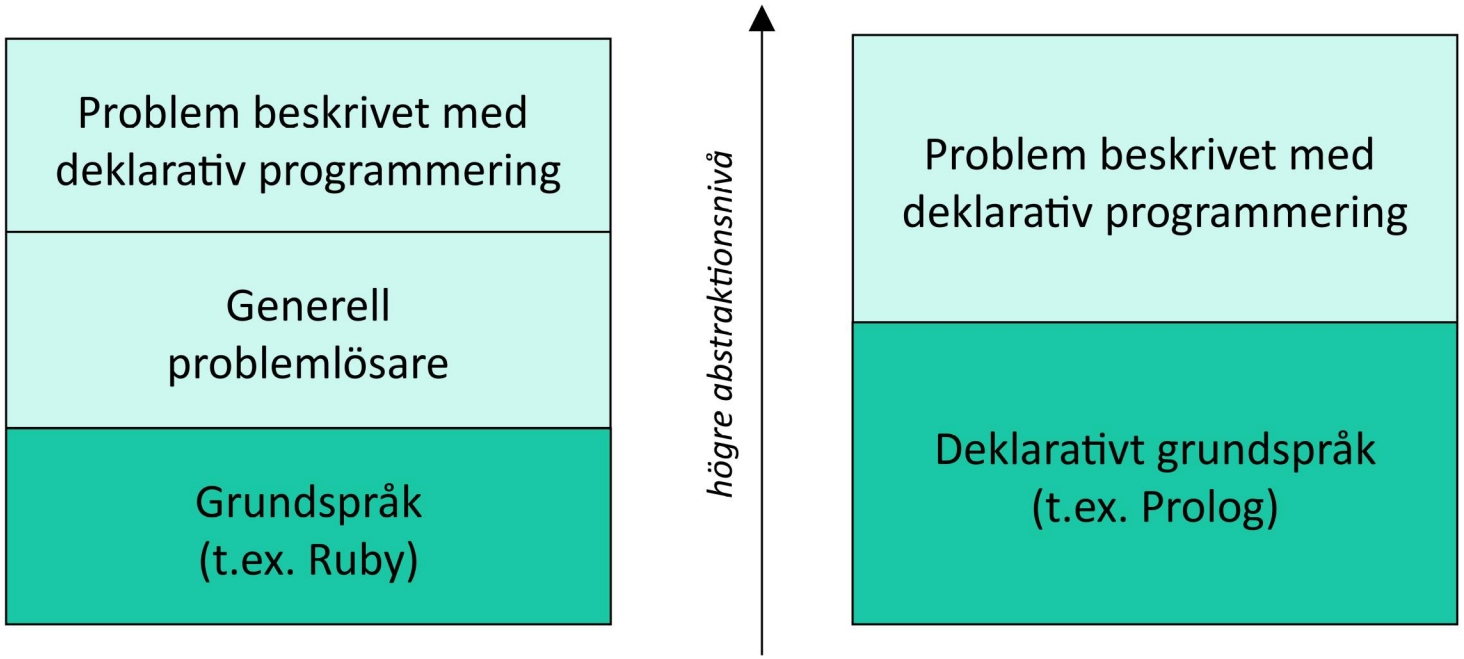
Vem gör grovjobbet?

Även om programmet uttrycks deklarativt måste ”någon” förr eller senare bestämma sig för hur problemet ska lösas.



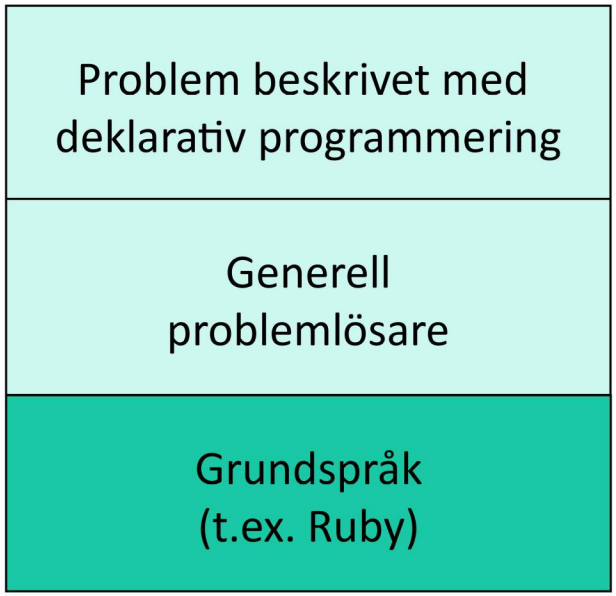
Vem gör grovjobbet?

Även om programmet uttrycks deklarativt måste ”någon” förr eller senare bestämma sig för hur problemet ska lösas.



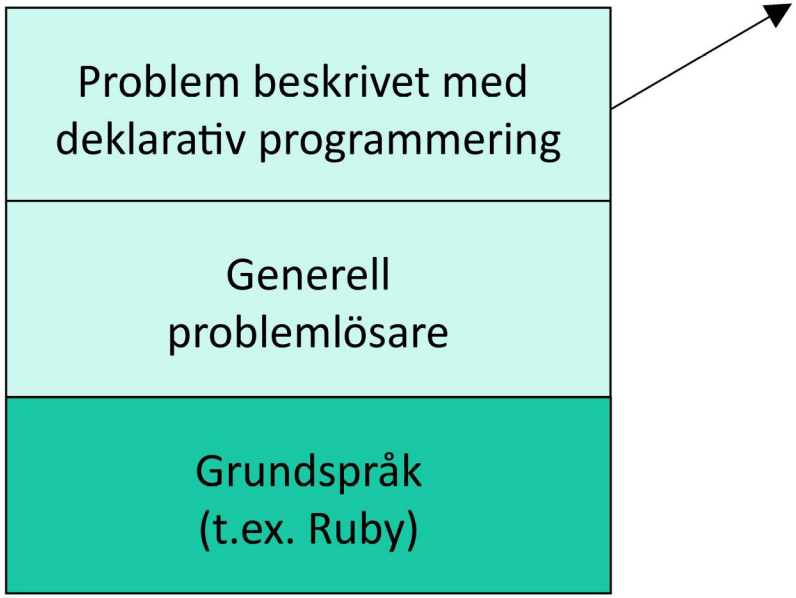
Deklarativ programmering i Ruby

Problemets natur avgör hur generell lösning vi behöver.



Deklarativ programmering i Ruby

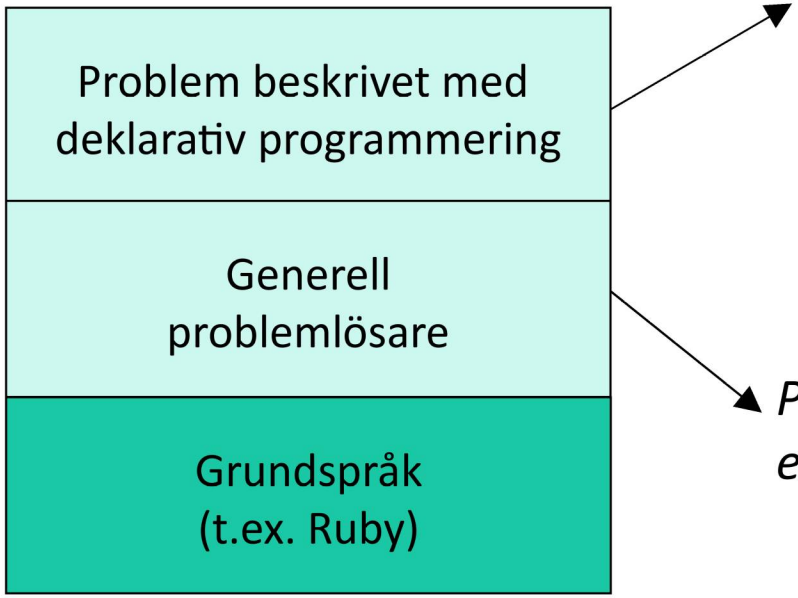
Problemets natur avgör hur generell lösning vi behöver.



Problemet kan t.ex. uttryckas med hjälp av ett nytt (domänspecifikt) språk.

Deklarativ programmering i Ruby

Problemets natur avgör hur generell lösning vi behöver.



Problemet kan t.ex. uttryckas med hjälp av ett nytt (domänspecifikt) språk.

Problemlösaren kan då ses som en interpretator för det nya språket.

Dagens föreläsning

- Vi ska idag titta på två tekniker för generella problemlösare som låter oss definiera problem deklarativt:
  - Icke-deterministisk programmering med hjälp av *continuations*
  - Restriktionsnätverk (eng. *constraint propagation networks*) för att modellera matematiska samband
- Målet är inte att förstå dessa tekniker i alla detaljer, utan att få en känsla för hur de kan användas.

1. Icke-deterministisk programmering

- Betyder egentligen att resultatet inte är förutbestämt.

1. Icke-deterministisk programmering

- Betyder egentligen att resultatet inte är förutbestämt.
- Exempel på problem:

  Kalle köper 20 saker i kiosken och betalar totalt 100 kr. Tuggummi kostar 1 kr, äpplen 7 kr och Coca Cola 12 kr. Hur många köpte han av varje?

Ickedeterministisk problemlösare

LINKÖPINGS UNIVERSITET

1. Icke-deterministisk programmering

- Betyder egentligen att resultatet inte är förutbestämt.
- Exempel på problem:

  Kalle köper 20 saker i kiosken och betalar totalt 100 kr. Tuggummi kostar 1 kr, äpplen 7 kr och Coca Cola 12 kr. Hur många köpte han av varje?

- Vi skulle vilja lösa detta t.ex. så här:

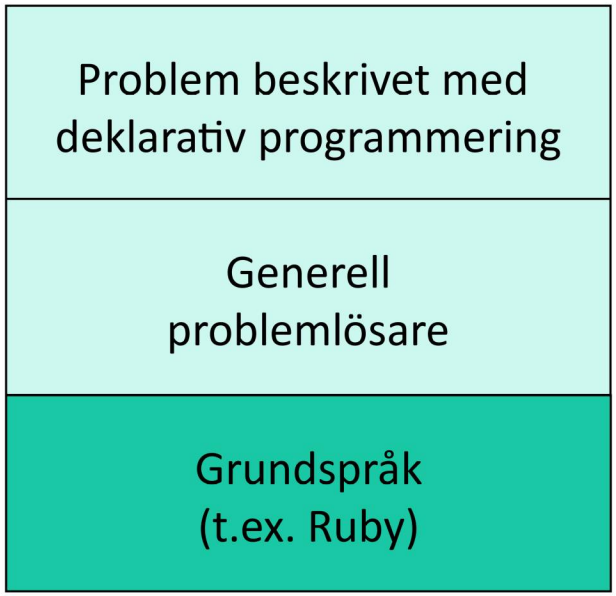
```
gum <- 1..20
apple <- 1..20
coke <- 1..20
gum*1+apple*7+coke*12=100
```

En kommentar

- Haka inte upp dig på ”att resultatet inte är förutbestämt”. Tänk tillbaka till denna:

En kommentar

- Haka inte upp dig på ”att resultatet inte är förutbestämt”. Tänk tillbaka till denna:



Generell problemlösare

- För att kunna deklarera problemen på det viset behöver vi en generell problemlösare som kan:

Generell problemlösare

- För att kunna deklarera problemen på det viset behöver vi en generell problemlösare som kan:
  - tilldela en variabel ett värde ur ett intervall

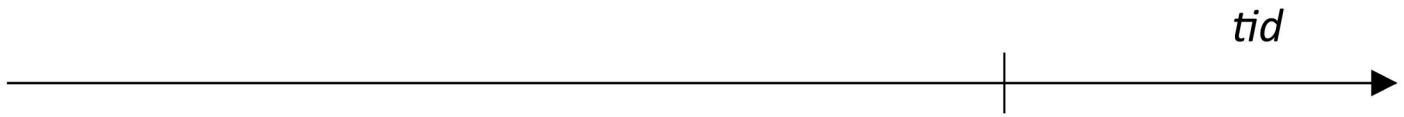
Generell problemlösare

- För att kunna deklarera problemen på det viset behöver vi en generell problemlösare som kan:
  - tilldela en variabel ett värde ur ett intervall
  - ta hand om särskilda villkor på variablerna

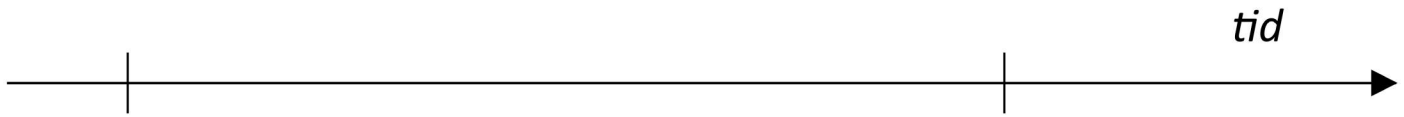
Generell problemlösare

- För att kunna deklarera problemen på det viset behöver vi en generell problemlösare som kan:
  - tilldela en variabel ett värde ur ett intervall
  - ta hand om särskilda villkor på variablerna
  - försöka igen om tilldelningen inte stämmer överens med villkoren

Backtracking

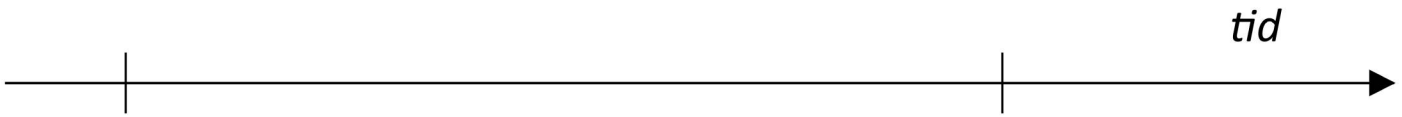


Backtracking



X *får ett av värdena i intervallet 1..20*

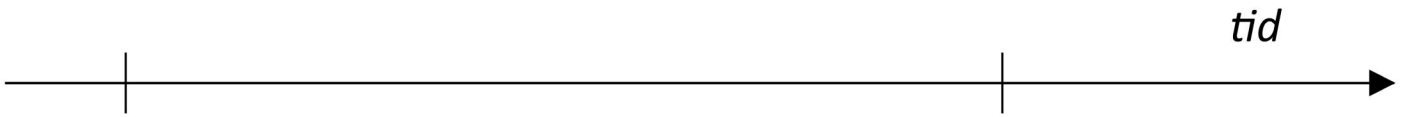
Backtracking



X *får ett av värdena i intervallet 1..20*

Programmet fortskrider.

Backtracking

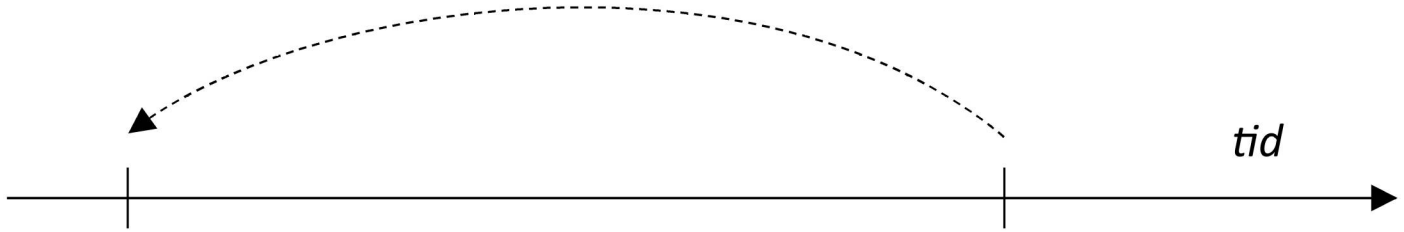


X *får ett av värdena i intervallet 1..20*

Programmet fortskrider.

Ett villkor på X *gör att det valda värdet inte kan accepteras.*

Backtracking

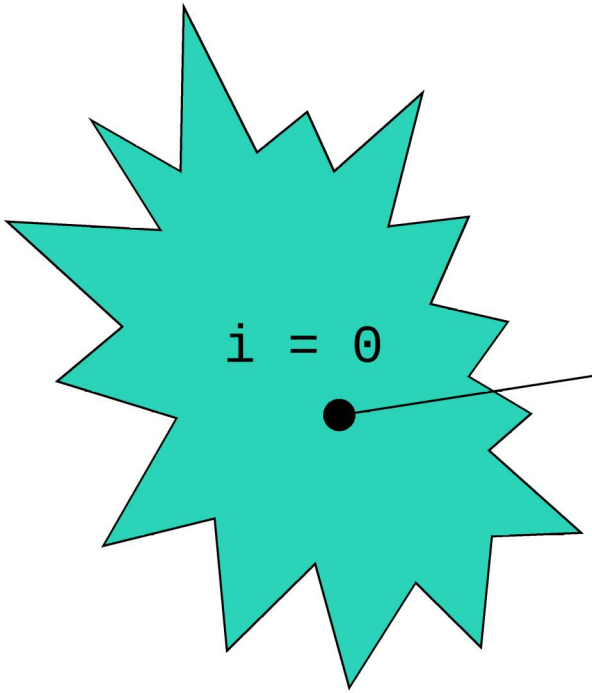


X får ett av värdena i intervallet 1..20

Programmet fortskrider.

Ett villkor på X gör att det valda värdet inte kan accepteras.

Exempel

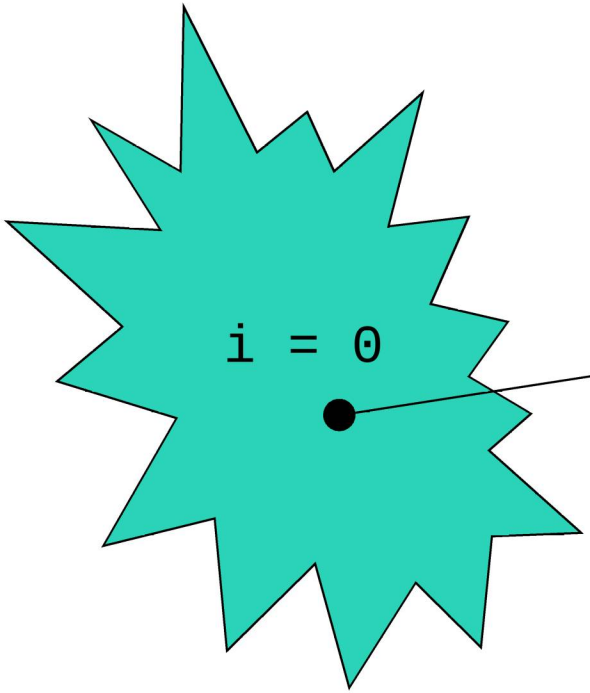



```
require ’continuation’

def start
  i = 0
  callcc { |c| $again = c }
  i = i+1
  i
end
```

Continuation

- En *continuation* representerar ”resten” av en beräkning. Den innehåller all information som behövs för att fortsätta programmet från en viss punkt (som ett savegame).
- Man kan använda en continuation för att åsidosätta normala styrstrukturer, t.ex. för att hoppa ut ur eller upprepa kod.
- Continuations skapas i Ruby med konstruktionen **callcc**, vilket står för *call with current continuation*. *(deprecated)*

Övning




```
require 'continuation'

def start
  i = 0
  callcc { |c| $again = c }
  i = i+1
  i
end
```

Vad händer och varför?
Diskutera gärna med andra i IP-salen...

Övning

Skriv in koden ovan, eller kopiera från kurshemsidan i filen cont.rb.
Pröva att anropa **`start`** *och därefter* **`$again.call`** *några gånger.*

Backtracking med continuations

```
class Amb

  def initialize
    @backtrack_points = []
  end

  def choose(choices)
    choices_array = choices.to_a
    backtrack if choices_array.empty?
    callcc do |cc|
      @backtrack_points.push cc
      return choices_array[0]
    end
    choose(choices_array[1..choices_array.length])
  end

# ...
```

Backtracking med continuations

```
class Amb

  def backtrack
    if @backtrack_points.empty?
      fail ExhaustedError, "Can't backtrack"
    else
      @backtrack_points.pop.call
    end
  end

  def require(condition)
    backtrack unless condition
  end

# ...
```

Problemets lösning

```
def my_problem
  a = Amb.new
  begin
    gum = a.choose 1..20
    apple = a.choose 1..20
    coke = a.choose 1..20

    a.require gum*1+apple*7+coke*12 == 100

    puts "#{gum} tuggummi, #{apple} äpplen, #{coke} cola"

    a.next
  rescue ExhaustedError
    puts "Det var alla möjligheter."
  end
end
```

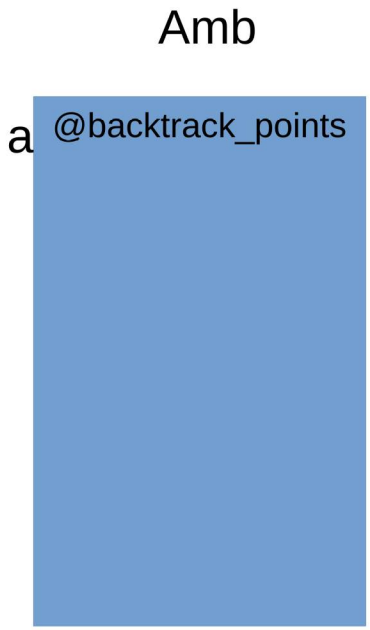
Problemets lösning

```
def my_problem
  a = Amb.new
  begin
    gum = a.choose 1..20
    apple = a.choose 1..20
    coke = a.choose 1..20

    a.require gum*1+apple*7+coke*12 == 100

    puts "#{gum} tuggummi, #{apple} äpplen, #{coke} cola"

    a.next
  rescue ExhaustedError
    puts "Det var alla möjligheter."
  end
end
```

Vad händer?


```
def my_problem
  a = Amb.new
  begin
    gum = a.choose 1..20
    apple = a.choose 1..20
    coke = a.choose 1..20

    a.require gum*1+.. == 100

    puts "#{gum} tuggummi..."

    a.next
  rescue ExhaustedError
    puts "Det var alla..."
  end
end
```

Vad händer?




```
def my_problem
  a = Amb.new
  begin
    gum = a.choose 1..20
    apple = a.choose 1..20
    coke = a.choose 1..20

    a.require gum*1+.. == 100

    puts "#{gum} tuggummi..."

    a.next
  rescue ExhaustedError
    puts "Det var alla..."
  end
end
```

Vad händer?

Amb

a

@backtrack_points


```
def my_problem
  a = Amb.new
  begin
    gum = a.choose 1..20
    apple = a.choose 1..20
    coke = a.choose 1..20

    a.require gum*1+.. == 100

    puts "#{gum} tuggummi..."

    a.next
  rescue ExhaustedError
    puts "Det var alla..."
  end
end
```

Vad händer?

Amb

a

@backtrack_points

```
def my_problem
  a = Amb.new
  begin
    gum = a.choose 1..20
    apple = a.choose 1..20
    coke = a.choose 1..20

    a.require gum*1+.. == 100

    puts "#{gum} tuggummi..."

    a.next
  rescue ExhaustedError
    puts "Det var alla..."
  end
end
```

```
# ...
  def choose(choices)
    choices_array = choices.to_a
    backtrack if choices_array.empty?
    callcc do |cc|
      @backtrack_points.push cc
      return choices_array[0]
    end
    choose(choices_array[1..choices_array.length])
  end
# ...
```

Vad händer?

Amb

a

@backtrack_points

```
def my_problem
  a = Amb.new
  begin
    gum = a.choose 1..20
    apple = a.choose 1..20
    coke = a.choose 1..20

    a.require gum*1+.. == 100

    puts "#{gum} tuggummi..."

    a.next
  rescue ExhaustedError
    puts "Det var alla..."
  end
end
```

```
# ...
  def choose(choices)
    choices_array = choices.to_a
    backtrack if choices_array.empty?
    callcc do |cc|
      @backtrack_points.push cc
      return choices_array[0]
    end
    choose(choices_array[1..choices_array.length])
  end
# ...
```

Vad händer?

Amb

a

@backtrack_points

```
def my_problem
  a = Amb.new
  begin
    gum = a.choose 1..20
    apple = a.choose 1..20
    coke = a.choose 1..20

    a.require gum*1+.. == 100

    puts "#{gum} tuggummi..."

    a.next
  rescue ExhaustedError
    puts "Det var alla..."
  end
end
```

```
# ...
  def choose(choices)
    choices_array = choices.to_a
    backtrack if choices_array.empty?
    callcc do |cc|
      @backtrack_points.push cc
      return choices_array[0]
    end
    choose(choices_array[1..choices_array.length])
  end
# ...
```

Vad händer?

Amb

a

@backtrack_points
&[1, 2, 3, 4, 5, 6...]

```
def my_problem
  a = Amb.new
  begin
    gum = a.choose 1..20
    apple = a.choose 1..20
    coke = a.choose 1..20

    a.require gum*1+.. == 100

    puts "#{gum} tuggummi..."

    a.next
  rescue ExhaustedError
    puts "Det var alla..."
  end
end
```

```
# ...
  def choose(choices)
    choices_array = choices.to_a
    backtrack if choices_array.empty?
    callcc do |cc|
      @backtrack_points.push cc
      return choices_array[0]
    end
    choose(choices_array[1..choices_array.length])
  end
# ...
```

Vad händer?

Amb

a

@backtrack_points
&[1, 2, 3, 4, 5, 6...]

```
def my_problem
  a = Amb.new
  begin
    gum = a.choose 1..20
    apple = a.choose 1..20
    coke = a.choose 1..20

    a.require gum*1+.. == 100

    puts "#{gum} tuggummi..."

    a.next
  rescue ExhaustedError
    puts "Det var alla..."
  end
end
```

```
# ...
  def choose(choices)
    choices_array = choices.to_a
    backtrack if choices_array.empty?
    callcc do |cc|
      @backtrack_points.push cc
      return choices_array[0]
    end
    choose(choices_array[1..choices_array.length])
  end
# ...
```

Vad händer?

Amb

a

@backtrack_points
&[1, 2, 3, 4, 5, 6...]
&[2, 3, 4, 5, 6...]

```
def my_problem
  a = Amb.new
  begin
    gum = a.choose 1..20
    apple = a.choose 1..20
    coke = a.choose 1..20

    a.require gum*1+.. == 100

    puts "#{gum} tuggummi..."

    a.next
  rescue ExhaustedError
    puts "Det var alla..."
  end
end
```

```
# ...
  def choose(choices)
    choices_array = choices.to_a
    backtrack if choices_array.empty?
    callcc do |cc|
      @backtrack_points.push cc
      return choices_array[0]
    end
    choose(choices_array[1..choices_array.length])
  end
# ...
```

Vad händer?

Amb

a

@backtrack_points
&[1, 2, 3, 4, 5, 6...]
&[2, 3, 4, 5, 6...]
&[3, 4, 5, 6...]

```
def my_problem
  a = Amb.new
  begin
    gum = a.choose 1..20
    apple = a.choose 1..20
    coke = a.choose 1..20

    a.require gum*1+.. == 100

    puts "#{gum} tuggummi..."

    a.next
  rescue ExhaustedError
    puts "Det var alla..."
  end
end
```

```
# ...
  def choose(choices)
    choices_array = choices.to_a
    backtrack if choices_array.empty?
    callcc do |cc|
      @backtrack_points.push cc
      return choices_array[0]
    end
    choose(choices_array[1..choices_array.length])
  end
# ...
```

Vad händer?

Amb

a

@backtrack_points
&[1, 2, 3, 4, 5, 6...]
&[2, 3, 4, 5, 6...]
&[3, 4, 5, 6...]
&[4, 5, 6...]

```
def my_problem
  a = Amb.new
  begin
    gum = a.choose 1..20
    apple = a.choose 1..20
    coke = a.choose 1..20

    a.require gum*1+.. == 100

    puts "#{gum} tuggummi..."

    a.next
  rescue ExhaustedError
    puts "Det var alla..."
  end
end
```

```
# ...
  def choose(choices)
    choices_array = choices.to_a
    backtrack if choices_array.empty?
    callcc do |cc|
      @backtrack_points.push cc
      return choices_array[0]
    end
    choose(choices_array[1..choices_array.length])
  end
# ...
```

Vad händer?

Amb

a

@backtrack_points
&[1, 2, 3, 4, 5, 6...]
&[2, 3, 4, 5, 6...]
&[3, 4, 5, 6...]
&[4, 5, 6...]
...

```
def my_problem
  a = Amb.new
  begin
    gum = a.choose 1..20
    apple = a.choose 1..20
    coke = a.choose 1..20

    a.require gum*1+.. == 100

    puts "#{gum} tuggummi..."

    a.next
  rescue ExhaustedError
    puts "Det var alla..."
  end
end
```

```
# ...
  def choose(choices)
    choices_array = choices.to_a
    backtrack if choices_array.empty?
    callcc do |cc|
      @backtrack_points.push cc
      return choices_array[0]
    end
    choose(choices_array[1..choices_array.length])
  end
# ...
```

Vad händer?

Amb

a

@backtrack_points
&[1, 2, 3, 4, 5, 6...]
&[2, 3, 4, 5, 6...]
&[3, 4, 5, 6...]
&[4, 5, 6...]
...
&[20]

```
def my_problem
  a = Amb.new
  begin
    gum = a.choose 1..20
    apple = a.choose 1..20
    coke = a.choose 1..20

    a.require gum*1+.. == 100

    puts "#{gum} tuggummi..."

    a.next
  rescue ExhaustedError
    puts "Det var alla..."
  end
end
```

```
# ...
  def choose(choices)
    choices_array = choices.to_a
    backtrack if choices_array.empty?
    callcc do |cc|
      @backtrack_points.push cc
      return choices_array[0]
    end
    choose(choices_array[1..choices_array.length])
  end
# ...
```

Vad händer?

Amb

a

@backtrack_points
&[1, 2, 3, 4, 5, 6...]
&[2, 3, 4, 5, 6...]
&[3, 4, 5, 6...]
&[4, 5, 6...]
...
&[20]

```
def my_problem
  a = Amb.new
  begin
    gum = a.choose 1..20
    apple = a.choose 1..20
    coke = a.choose 1..20

    a.require gum*1+.. == 100

    puts "#{gum} tuggummi..."

    a.next
  rescue ExhaustedError
    puts "Det var alla..."
  end
end
```

```
# ...
  def choose(choices)
    choices_array = choices.to_a
    backtrack if choices_array.empty?
    callcc do |cc|
      @backtrack_points.push cc
      return choices_array[0]
    end
    choose(choices_array[1..choices_array.length])
  end
# ...
```

Vad händer?

Amb

a

```
@backtrack_points
&[1, 2, 3, 4, 5, 6...]
&[2, 3, 4, 5, 6...]
&[3, 4, 5, 6...]
&[4, 5, 6...]
...
&[20]
```

```
def my_problem
  a = Amb.new
  begin
    gum = a.choose 1..20
    apple = a.choose 1..20
    coke = a.choose 1..20

    a.require gum*1+.. == 100

    puts "#{gum} tuggummi..."

    a.next
  rescue ExhaustedError
    puts "Det var alla..."
  end
end
```

```
# …
  def backtrack
    if @backtrack_points.empty?
      fail ExhaustedError, "Can't backtrack"
    else
      @backtrack_points.pop.call
    end
  end
# …
```

Vad händer?

Amb

a

```
@backtrack_points
&[1, 2, 3, 4, 5, 6...]
&[2, 3, 4, 5, 6...]
&[3, 4, 5, 6...]
&[4, 5, 6...]
...
&[20]
```

```
def my_problem
  a = Amb.new
  begin
    gum = a.choose 1..20
    apple = a.choose 1..20
    coke = a.choose 1..20

    a.require gum*1+.. == 100

    puts "#{gum} tuggummi..."

    a.next
  rescue ExhaustedError
    puts "Det var alla..."
  end
end
```

```
# …
  def backtrack
    if @backtrack_points.empty?
      fail ExhaustedError, "Can't backtrack"
    else
      @backtrack_points.pop.call
    end
  end
# …
```

Vad händer?

Amb

a

```
@backtrack_points
&[1, 2, 3, 4, 5, 6...]
&[2, 3, 4, 5, 6...]
&[3, 4, 5, 6...]
&[4, 5, 6...]
...
&[20]
```

```
def my_problem
  a = Amb.new
  begin
    gum = a.choose 1..20
    apple = a.choose 1..20
    coke = a.choose 1..20

    a.require gum*1+.. == 100

    puts "#{gum} tuggummi..."

    a.next
  rescue ExhaustedError
    puts "Det var alla..."
  end
end
```

```
# …
  def backtrack
    if @backtrack_points.empty?
      fail ExhaustedError, "Can't backtrack"
    else
      @backtrack_points.pop.call
    end
  end
# …
```

Vad händer?

Amb

a

@backtrack_points
&[1, 2, 3, 4, 5, 6...]
&[2, 3, 4, 5, 6...]
&[3, 4, 5, 6...]
&[4, 5, 6...]
...
&[20]

```
def my_problem
  a = Amb.new
  begin
    gum = a.choose 1..20
    apple = a.choose 1..20
    coke = a.choose 1..20

    a.require gum*1+.. == 100

    puts "#{gum} tuggummi..."

    a.next
  rescue ExhaustedError
    puts "Det var alla..."
  end
end
```

```
# …
  def backtrack
    if @backtrack_points.empty?
      fail ExhaustedError, "Can't backtrack"
    else
      @backtrack_points.pop.call
    end
  end
# …
```

Vad händer?

Amb

a

```
@backtrack_points
&[1, 2, 3, 4, 5, 6...]
&[2, 3, 4, 5, 6...]
&[3, 4, 5, 6...]
&[4, 5, 6...]
...
&[20]
```

```
def my_problem
  a = Amb.new
  begin
    gum = a.choose 1..20
    apple = a.choose 1..20
    coke = a.choose 1..20

    a.require gum*1+.. == 100

    puts "#{gum} tuggummi..."

    a.next
  rescue ExhaustedError
    puts "Det var alla..."
  end
end
```

```
# …
  def backtrack
    if @backtrack_points.empty?
      fail ExhaustedError, "Can't backtrack"
    else
      @backtrack_points.pop.call
    end
  end
# …
```

Vad händer?

int

gum 20

a Amb

@backtrack_points
&[1, 2, 3, 4, 5, 6...]
&[2, 3, 4, 5, 6...]
&[3, 4, 5, 6...]
&[4, 5, 6...]
...

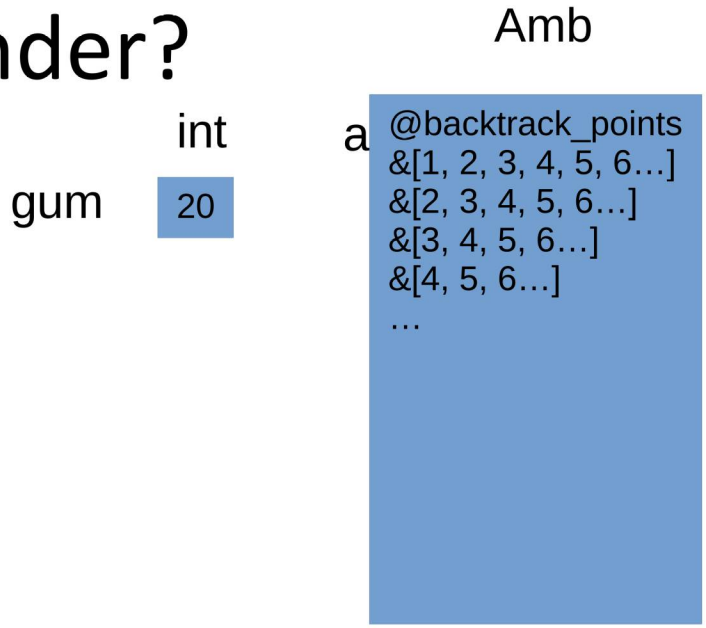
```
def my_problem
  a = Amb.new
  begin
    gum = a.choose 1..20
    apple = a.choose 1..20
    coke = a.choose 1..20

    a.require gum*1+.. == 100

    puts "#{gum} tuggummi..."

    a.next
  rescue ExhaustedError
    puts "Det var alla..."
  end
end
```

```
# ...
  def choose(choices)
    choices_array = choices.to_a
    backtrack if choices_array.empty?
    callcc do |cc|
      @backtrack_points.push cc
      return choices_array[0]
    end
    choose(choices_array[1..choices_array.length])
  end
# ...
```

Stack av backtrack points

- Denna problemlösare skapar en stack av backtrack points rekursivt
- I praktiken kan vi se på det som en stack av funktioner vi kan anropa, och tillsammans representerar dessa funktioner alla möjliga kombinationer

Vad händer?




```
def my_problem
  a = Amb.new
  begin
    gum = a.choose 1..20
    apple = a.choose 1..20
    coke = a.choose 1..20

    a.require gum*1+.. == 100

    puts "#{gum} tuggummi..."

    a.next
  rescue ExhaustedError
    puts "Det var alla..."
  end
end
```

Vad händer?

Amb

int

gum 20

int

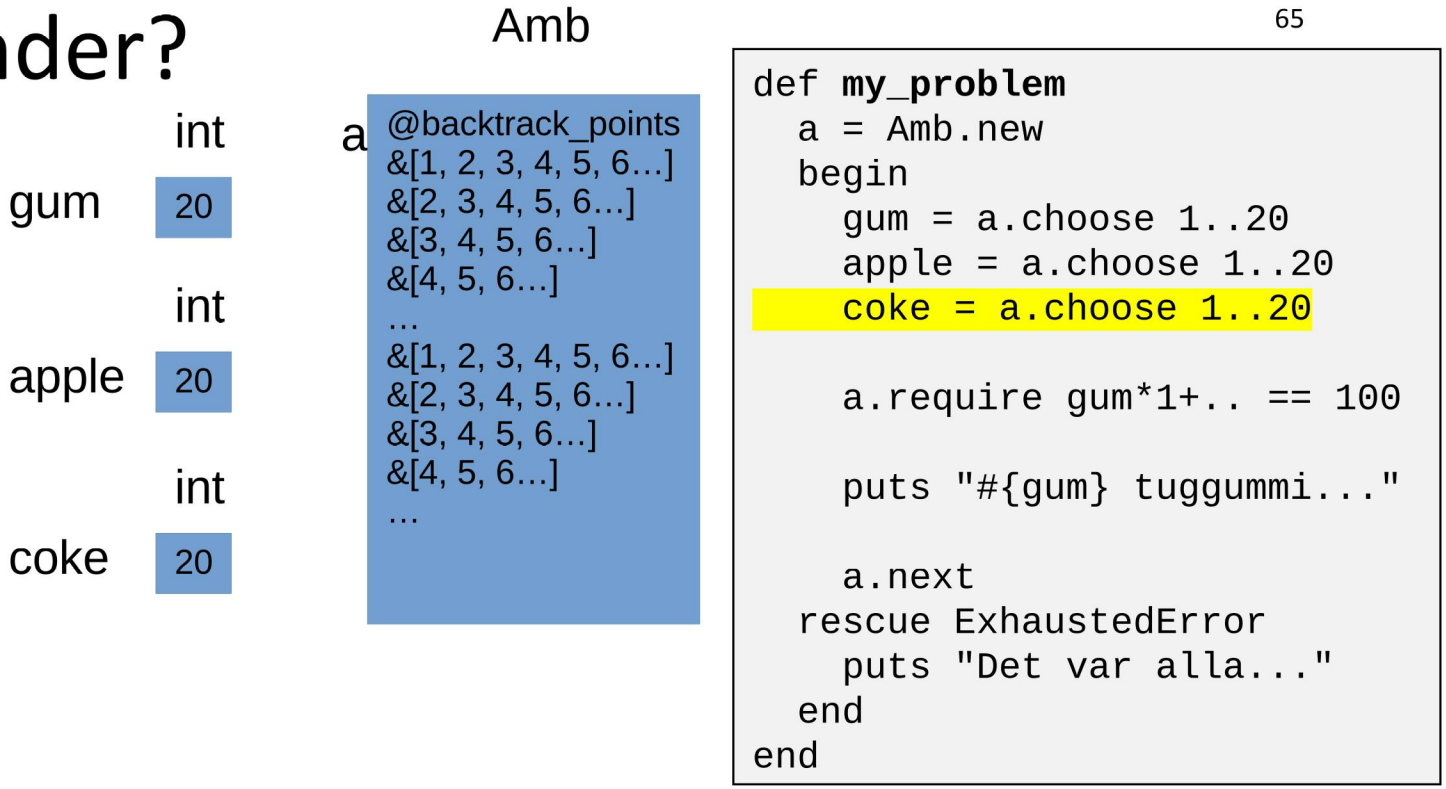
apple 20

a

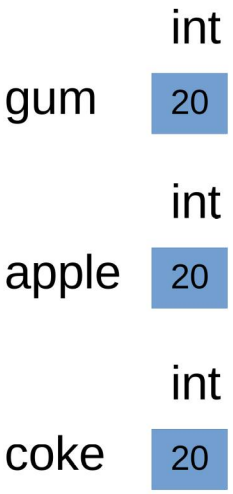
@backtrack_points
&[1, 2, 3, 4, 5, 6…]
&[2, 3, 4, 5, 6…]
&[3, 4, 5, 6…]
&[4, 5, 6…]
…
&[1, 2, 3, 4, 5, 6…]
&[2, 3, 4, 5, 6…]
&[3, 4, 5, 6…]
&[4, 5, 6…]
…

```
def my_problem
  a = Amb.new
  begin
    gum = a.choose 1..20
    apple = a.choose 1..20
    coke = a.choose 1..20

    a.require gum*1+.. == 100

    puts "#{gum} tuggummi..."

    a.next
  rescue ExhaustedError
    puts "Det var alla..."
  end
end
```

Vad händer?


Amb
int
gum
20
int
apple
20
int
coke
20
a
@backtrack_points
&[1, 2, 3, 4, 5, 6…]
&[2, 3, 4, 5, 6…]
&[3, 4, 5, 6…]
&[4, 5, 6…]
…
&[1, 2, 3, 4, 5, 6…]
&[2, 3, 4, 5, 6…]
&[3, 4, 5, 6…]
&[4, 5, 6…]
…
def my_problem
a = Amb.new
begin
gum = a.choose 1..20
apple = a.choose 1..20
coke = a.choose 1..20
a.require gum*1+.. == 100
puts "#{gum} tuggummi..."
a.next
rescue ExhaustedError
puts "Det var alla..."
end
end

Vad händer?



Amb

a

@backtrack_points
&[1, 2, 3, 4, 5, 6…]
&[2, 3, 4, 5, 6…]
&[3, 4, 5, 6…]
&[4, 5, 6…]
…
&[1, 2, 3, 4, 5, 6…]
&[2, 3, 4, 5, 6…]
&[3, 4, 5, 6…]
&[4, 5, 6…]
…

```
def my_problem
  a = Amb.new
  begin
    gum = a.choose 1..20
    apple = a.choose 1..20
    coke = a.choose 1..20

    a.require gum*1+.. == 100

    puts "#{gum} tuggummi..."

    a.next
  rescue ExhaustedError
    puts "Det var alla..."
  end
end
```

Vad händer?

Amb

int gum 20

int apple 20

int coke 20

a

@backtrack_points
&[1, 2, 3, 4, 5, 6…]
&[2, 3, 4, 5, 6…]
&[3, 4, 5, 6…]
&[4, 5, 6…]
…
&[1, 2, 3, 4, 5, 6…]
&[2, 3, 4, 5, 6…]
&[3, 4, 5, 6…]
&[4, 5, 6…]
…

```
def my_problem
  a = Amb.new
  begin
    gum = a.choose 1..20
    apple = a.choose 1..20
    coke = a.choose 1..20

    a.require gum*1+.. == 100

    puts "#{gum} tuggummi..."

    a.next
  rescue ExhaustedError
    puts "Det var alla..."
  end
end
```

Nu kollar vi om det stämmer

Vad händer?

Amb

int
gum 20

int
apple 20

int
coke 20

a

@backtrack_points
&[1, 2, 3, 4, 5, 6…]
&[2, 3, 4, 5, 6…]
&[3, 4, 5, 6…]
&[4, 5, 6…]
…
&[1, 2, 3, 4, 5, 6…]
&[2, 3, 4, 5, 6…]
&[3, 4, 5, 6…]
&[4, 5, 6…]
…

```
def my_problem
  a = Amb.new
  begin
    gum = a.choose 1..20
    apple = a.choose 1..20
    coke = a.choose 1..20

    a.require gum*1+.. == 100

    puts "#{gum} tuggummi..."

    a.next
  rescue ExhaustedError
    puts "Det var alla..."
  end
end
```

```
# ...
  def require(condition)
    backtrack unless condition
  end
# ...
```

Vad händer?

int gum 20

int apple 20

int coke 20

a

@backtrack_points
&[1, 2, 3, 4, 5, 6…]
&[2, 3, 4, 5, 6…]
&[3, 4, 5, 6…]
&[4, 5, 6…]
…
&[1, 2, 3, 4, 5, 6…]
&[2, 3, 4, 5, 6…]
&[3, 4, 5, 6…]
&[4, 5, 6…]
…

```
def my_problem
  a = Amb.new
  begin
    gum = a.choose 1..20
    apple = a.choose 1..20
    coke = a.choose 1..20

    a.require gum*1+.. == 100

    puts "#{gum} tuggummi..."

    a.next
  rescue ExhaustedError
    puts "Det var alla..."
  end
end
```

```
# …
  def backtrack
    if @backtrack_points.empty?
      fail ExhaustedError, "Can't backtrack"
    else
      @backtrack_points.pop.call
    end
  end
# …
```

Vad händer?

Amb

int gum 20

int apple 20

int coke 20

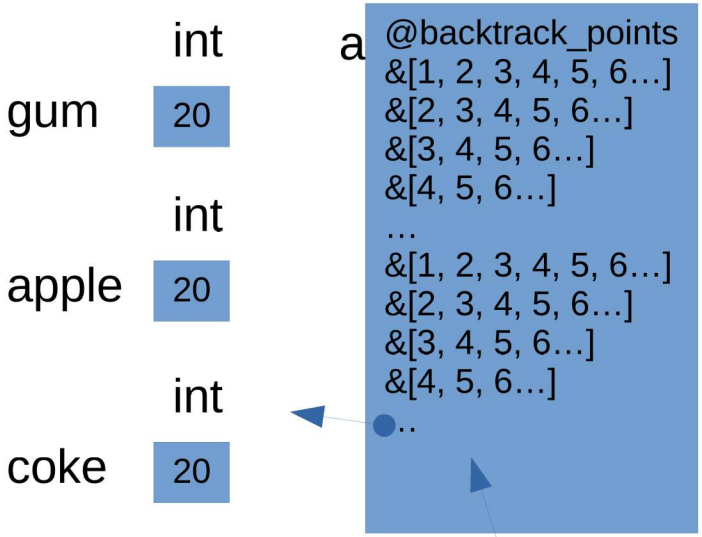
a

@backtrack_points
&[1, 2, 3, 4, 5, 6…]
&[2, 3, 4, 5, 6…]
&[3, 4, 5, 6…]
&[4, 5, 6…]
…
&[1, 2, 3, 4, 5, 6…]
&[2, 3, 4, 5, 6…]
&[3, 4, 5, 6…]
&[4, 5, 6…]
…

```
def my_problem
  a = Amb.new
  begin
    gum = a.choose 1..20
    apple = a.choose 1..20
    coke = a.choose 1..20

    a.require gum*1+.. == 100

    puts "#{gum} tuggummi..."

    a.next
  rescue ExhaustedError
    puts "Det var alla..."
  end
end
```

```
# …
  def backtrack
    if @backtrack_points.empty?
      fail ExhaustedError, "Can't backtrack"
    else
      @backtrack_points.pop.call
    end
  end
# …
```

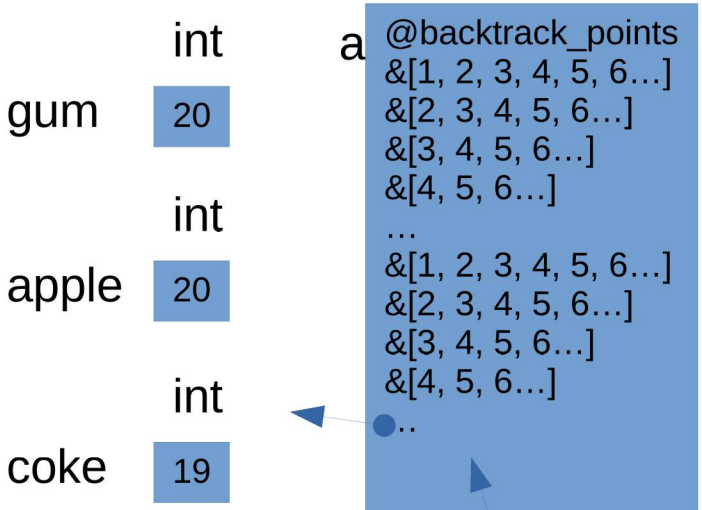
Vad händer?




```
def my_problem
  a = Amb.new
  begin
    gum = a.choose 1..20
    apple = a.choose 1..20
    coke = a.choose 1..20

    a.require gum*1+.. == 100

    puts "#{gum} tuggummi..."

    a.next
  rescue ExhaustedError
    puts "Det var alla..."
  end
end
```

```
# …
  def backtrack
    if @backtrack_points.empty?
      fail ExhaustedError, "Can't backtrack"
    else
      @backtrack_points.pop.call
    end
  end
# …
```

Vad händer?




```
def my_problem
  a = Amb.new
  begin
    gum = a.choose 1..20
    apple = a.choose 1..20
    coke = a.choose 1..20

    a.require gum*1+.. == 100

    puts "#{gum} tuggummi..."

    a.next
  rescue ExhaustedError
    puts "Det var alla..."
  end
end
```

```
# …
  def backtrack
    if @backtrack_points.empty?
      fail ExhaustedError, "Can't backtrack"
    else
      @backtrack_points.pop.call
    end
  end
# …
```

Övning


```
def my_problem
  a = Amb.new
  begin
    gum = a.choose 1..20
    apple = a.choose 1..20
    coke = a.choose 1..20

    a.require gum*1+apple*7+coke*12 == 100

    puts "#{gum} tuggummi, #{apple} äpplen, #{coke} cola"

    a.next
  rescue ExhaustedError
    puts "Det var alla möjligheter."
  end
end
```

Övning

Hämta koden från amb_test.rb *och komplettera den med det villkor som saknas. Lägg också till en räknare som kollar hur många alternativ vi testar!*

Constraint Network

2. *Constraint networks*

- Modellering av matematiska samband
- Exempel: Antag att vi ska bygga en spelmotor och på något sätt vill använda Newtons lagar om gravitation.
- Hur kan vi få in följande ekvation i vårt program?

$$F = \frac{Gm_1m_2}{r^2}$$

Modellera matematiska samband

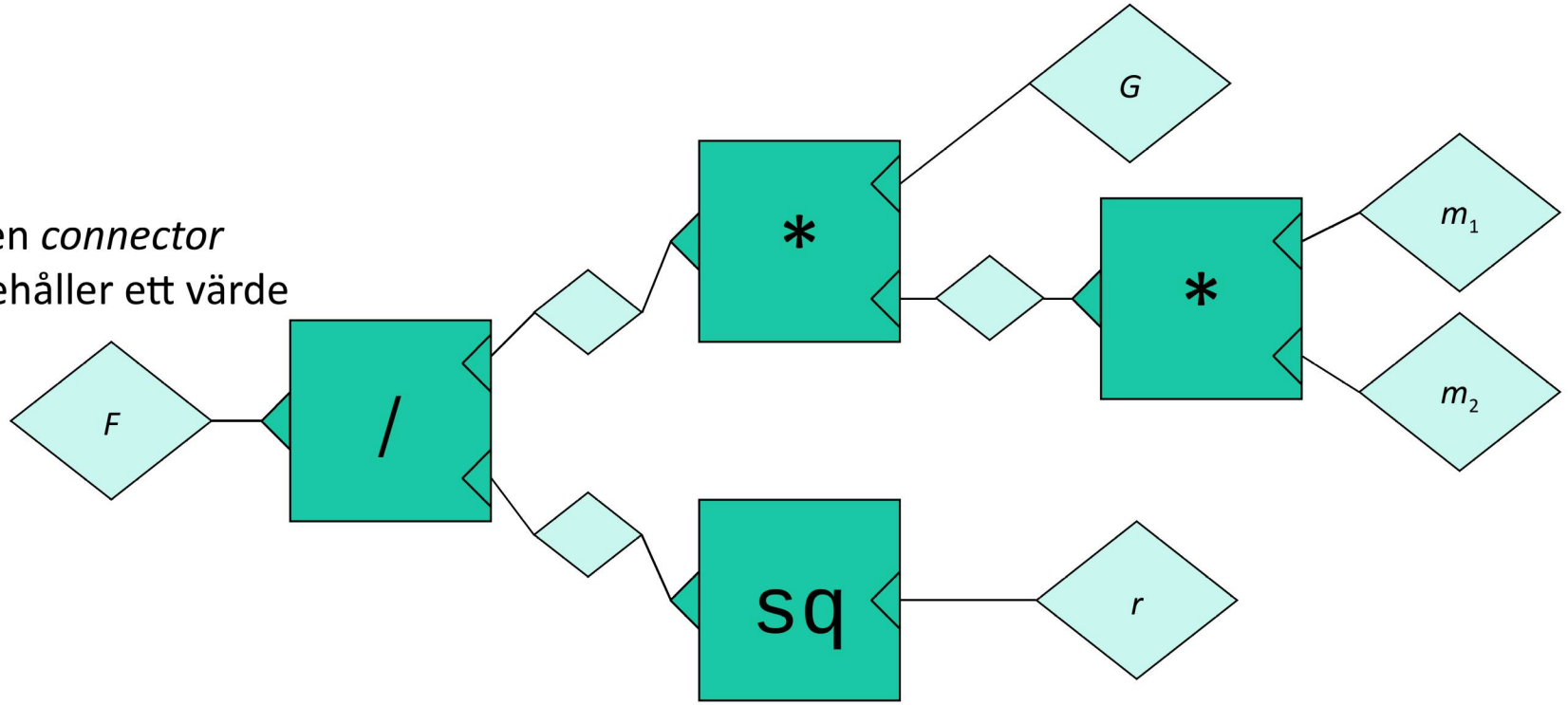
- *Som en ekvation:*

```
irb(main):107:0> f == g*m1*m2/r**2
true
```

- *Som ett antal metoder:*

```
def calculate_f(g,m1,m2,r)
  g*m1*m2/r**2
end
def calculate_m1(f,g,m2,r)
  f*(r**2)/(g*m2)
end
```

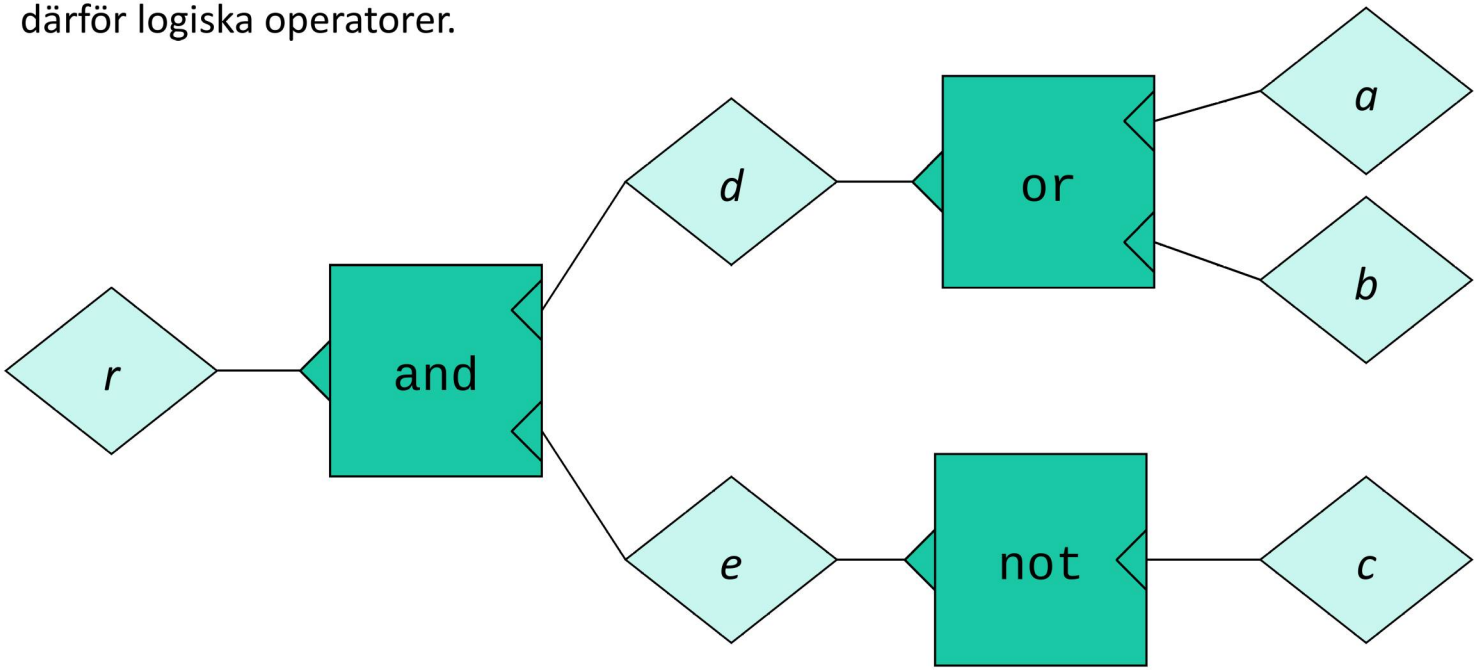
Ekvationen som ett *constraint network*



ett *constraint* uttrycker
förhållandet mellan värden

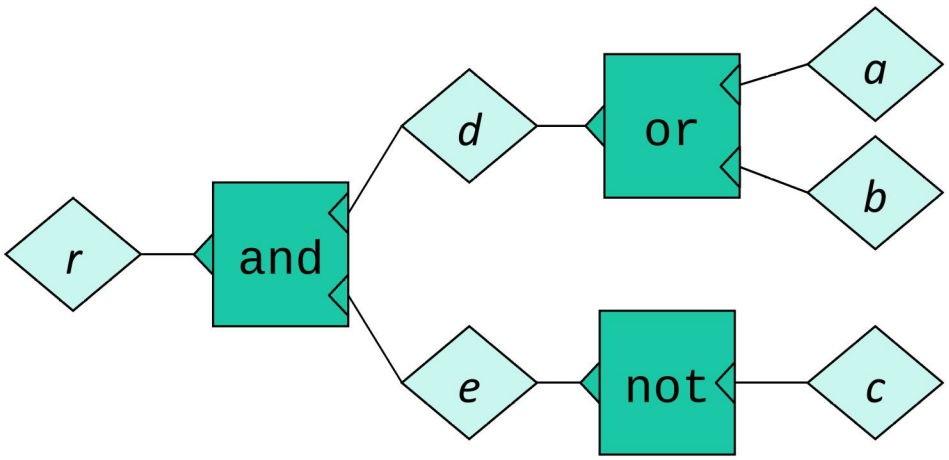
Sanningsvärden

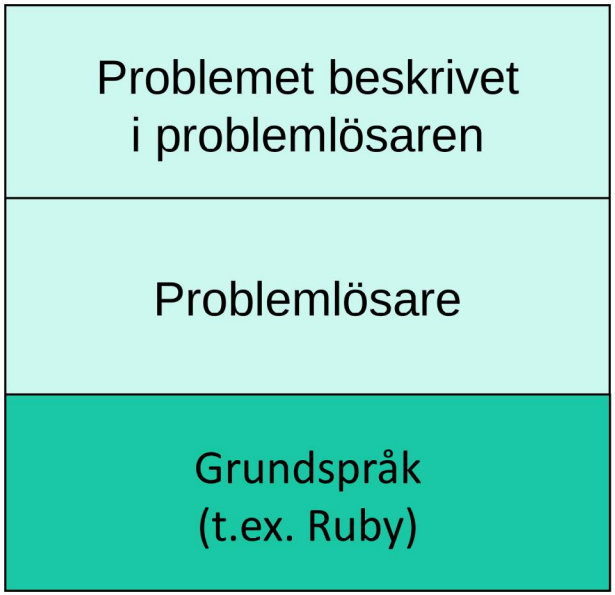
Detta är ett constraint network som jobbar med logiska värden istället för tal. Alla constraints är därför logiska operatorer.



Sanningsvärden

Detta är ett constraint network som jobbar med logiska värden istället för tal. Alla constraints är därför logiska operatorer.





Kan representeras med OOP

- Vi kan representera ett nätverk av denna typ på många sätt
- OOP är ett som vi är bekanta med
- Constraints (noder som är portar) kan vara instanser av klasser
- Wires (noder som är värden) kan vara instanser av klasser
- Själva kopplingen behöver inte vara sin egen typ (kan vara referenser)
- Vad har vi för behov?
  - Constraints måste kunna skicka signaler (känna till Wires/Constraints(?))
  - Wires måste kunna vidarebefordra signaler (känna till Constraints/Wires(?))

Definition av en *connector*

```
class Wire

  def initialize(name,value=false)
    @name=name
    @value=value
    @constraints=[]
  end

  def add_constraint(gate)
    @constraints << gate
  end

  def value=(value)
    @value=value
    @constraints.each { |c| c.new_value }
  end

end
```

För de logiska *constraint*-nätverken kallar vi *connectors* för *Wire*.

Definition av generell binär *constraint*


```
class BinaryConstraint

  def initialize(input1,input2,output)
    @input1=input1
    @input1.add_constraint(self)
    @input2=input2
    @input2.add_constraint(self)
    @output=output
    new_value()
  end

end
```

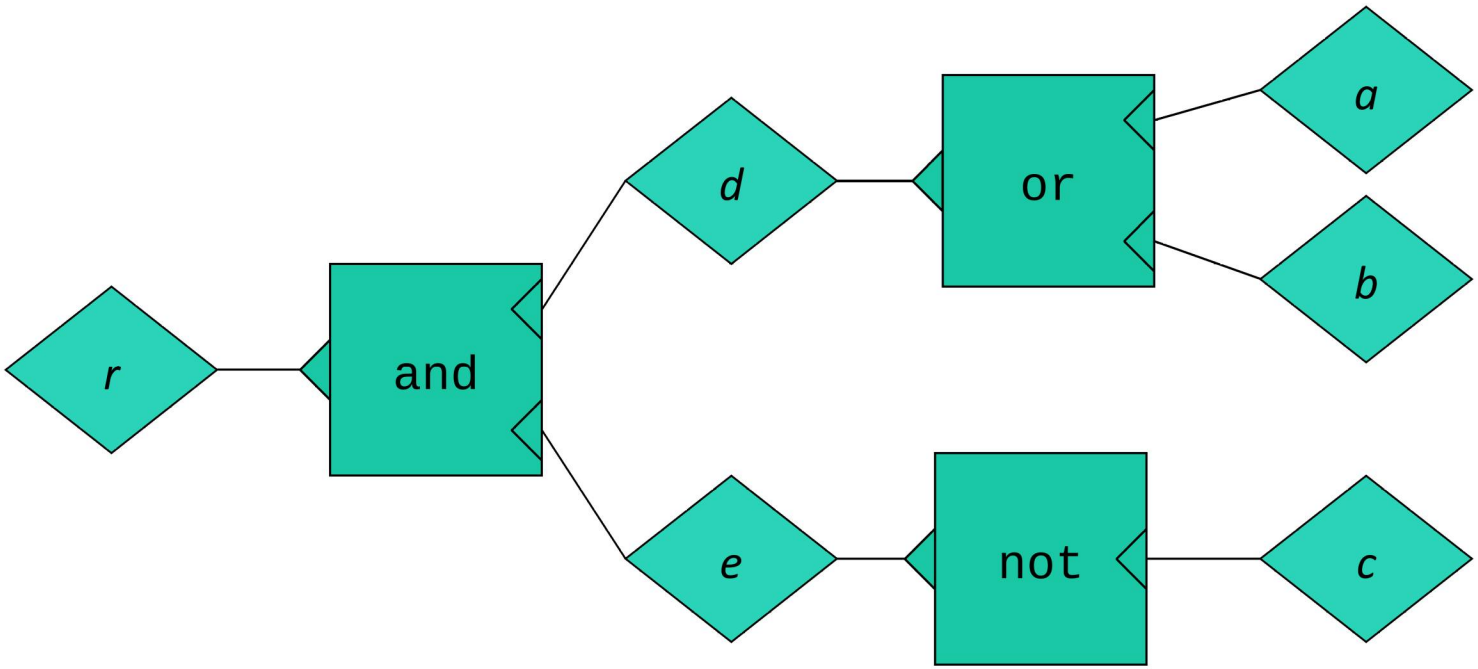
Implementation av *and* och *or*

```
class AndGate < BinaryConstraint

  def new_value
    sleep 0.2
    @output.value=(@input1.value and @input2.value)
  end

end

class OrGate < BinaryConstraint

  def new_value
    sleep 0.2
    @output.value=(@input1.value or @input2.value)
  end

end
```

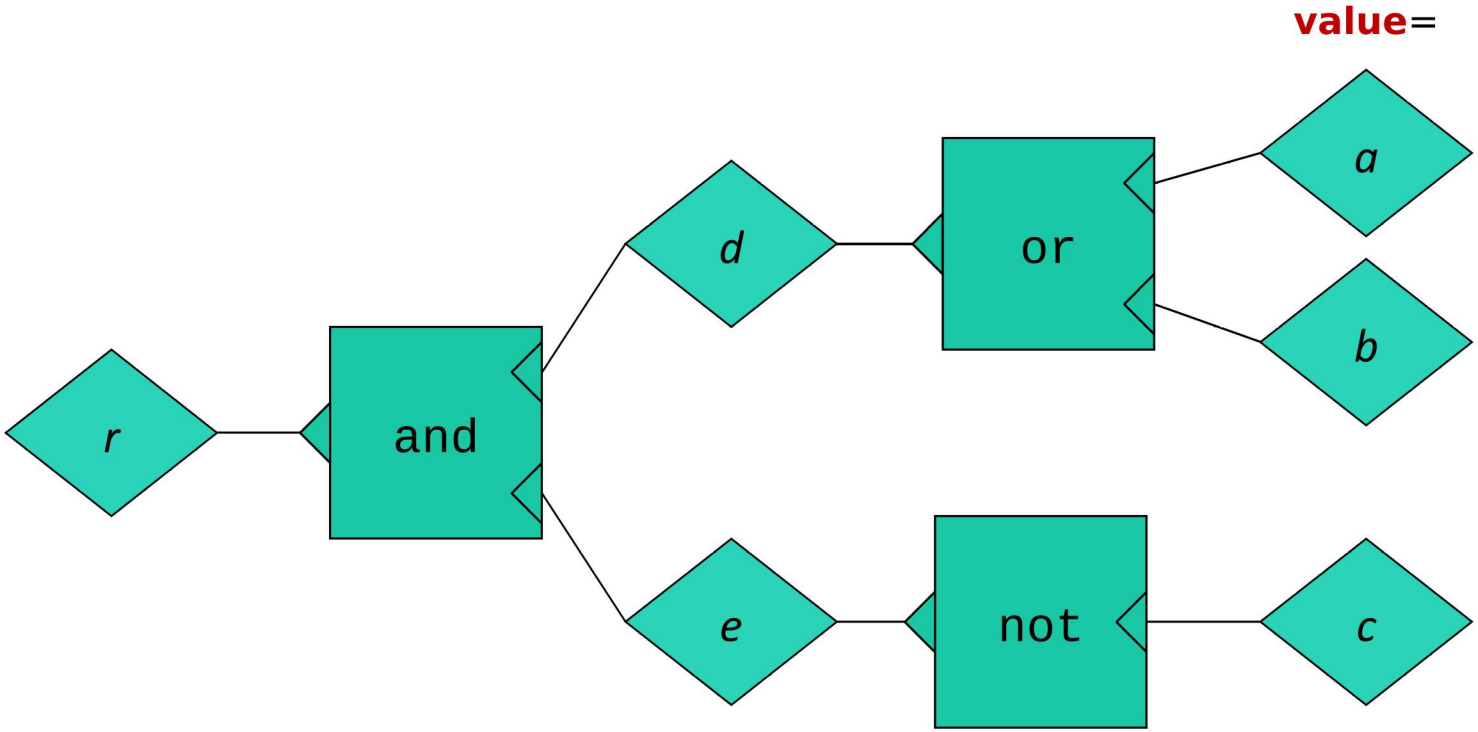
Implementation av *not*

```
class NotGate

  def initialize(input,output)
    @input=input
    @input.add_constraint(self)
    @output=output
    new_value()
  end

  def new_value
    sleep 0.2
    @output.value=(not @input.value)
  end

end
```

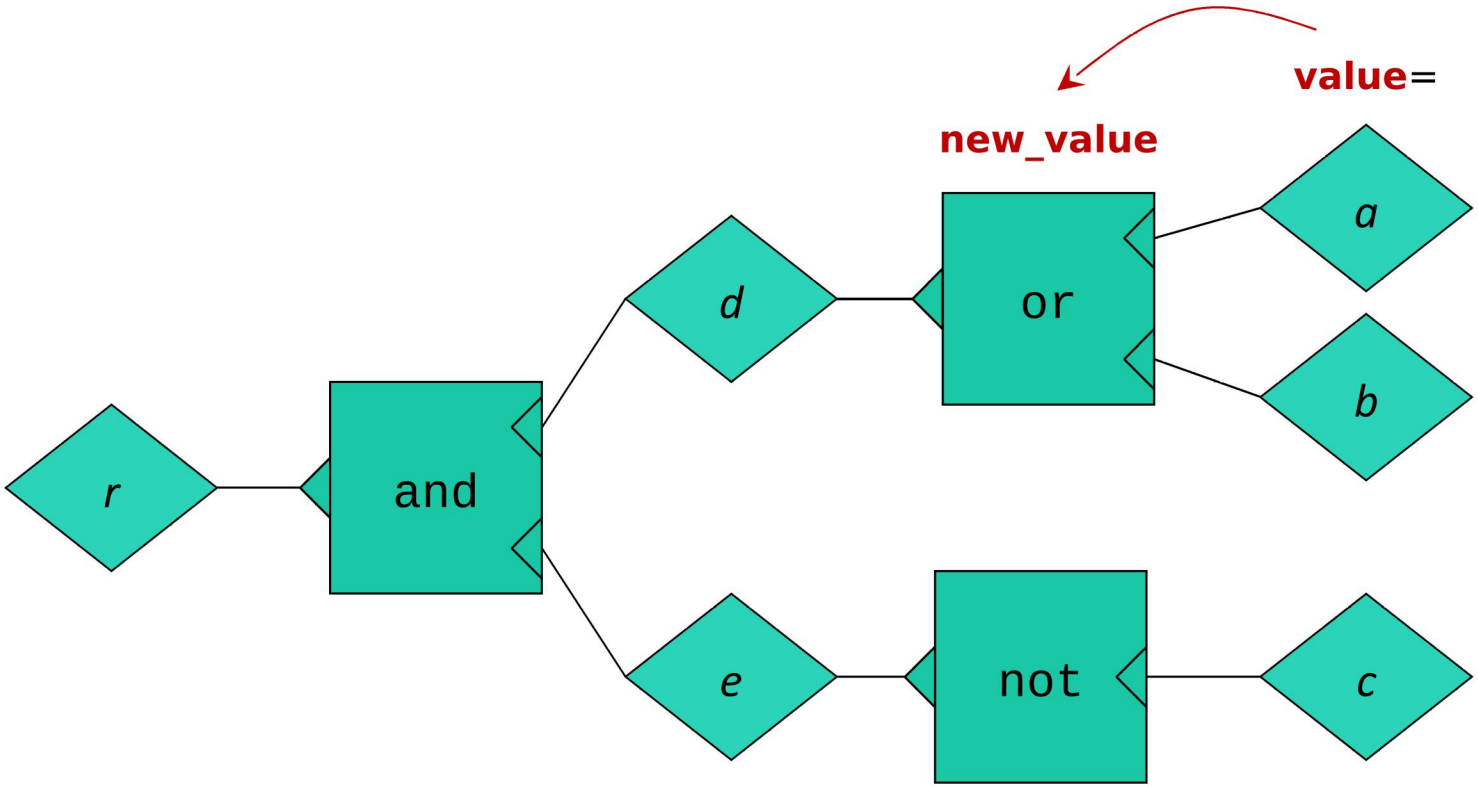
Hur värden propageras genom nätet



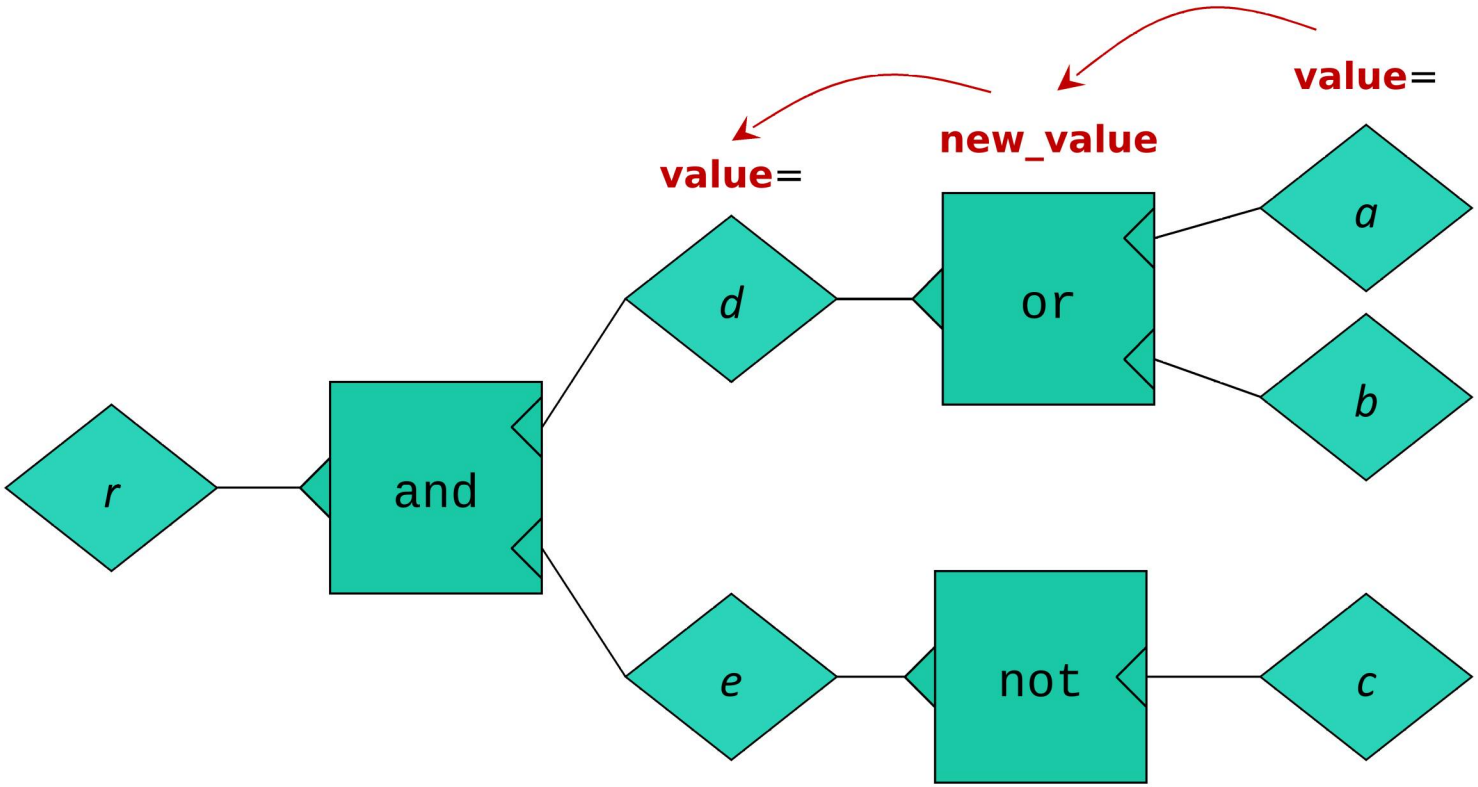
Hur värden propageras genom nätet



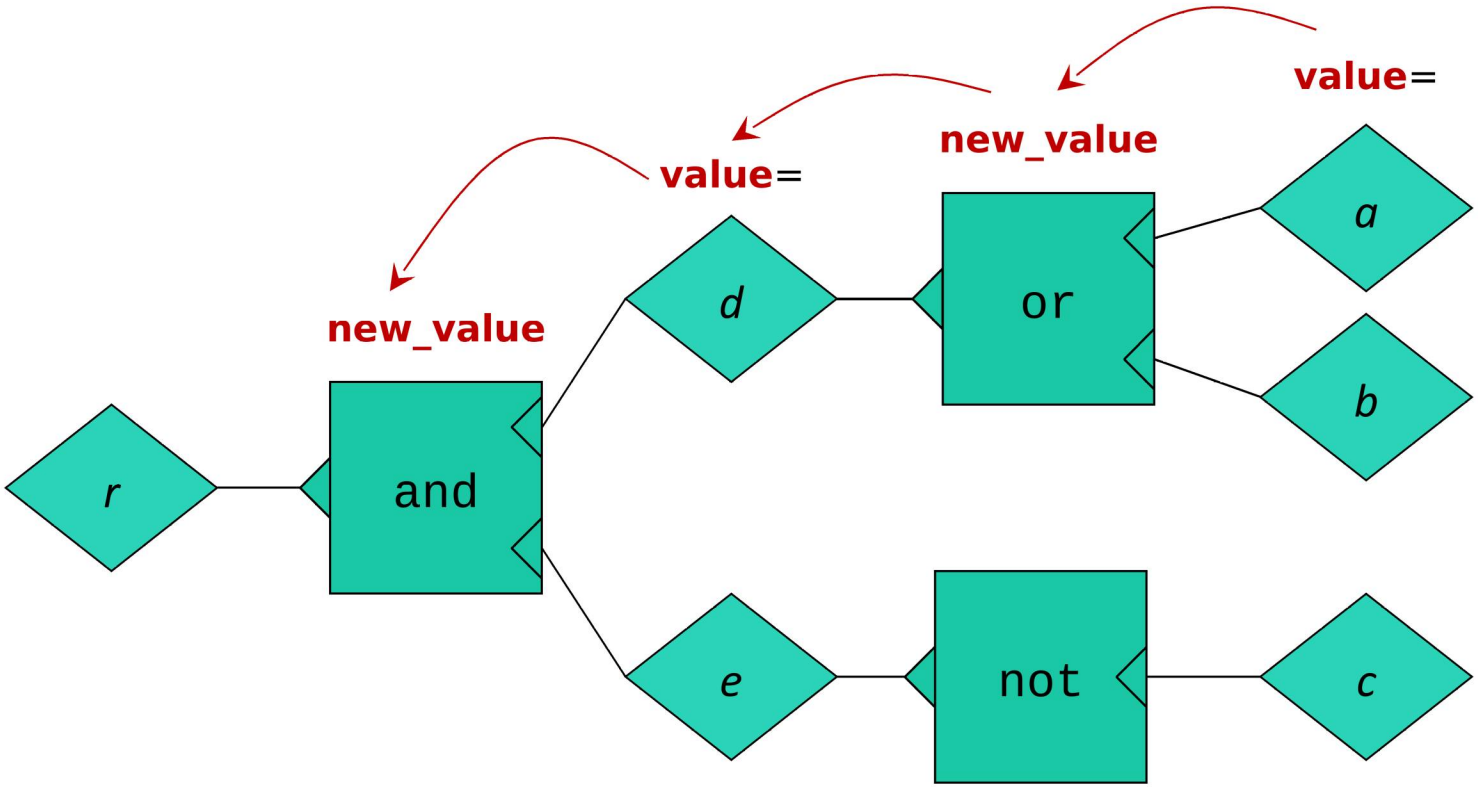
Hur värden propageras genom nätet



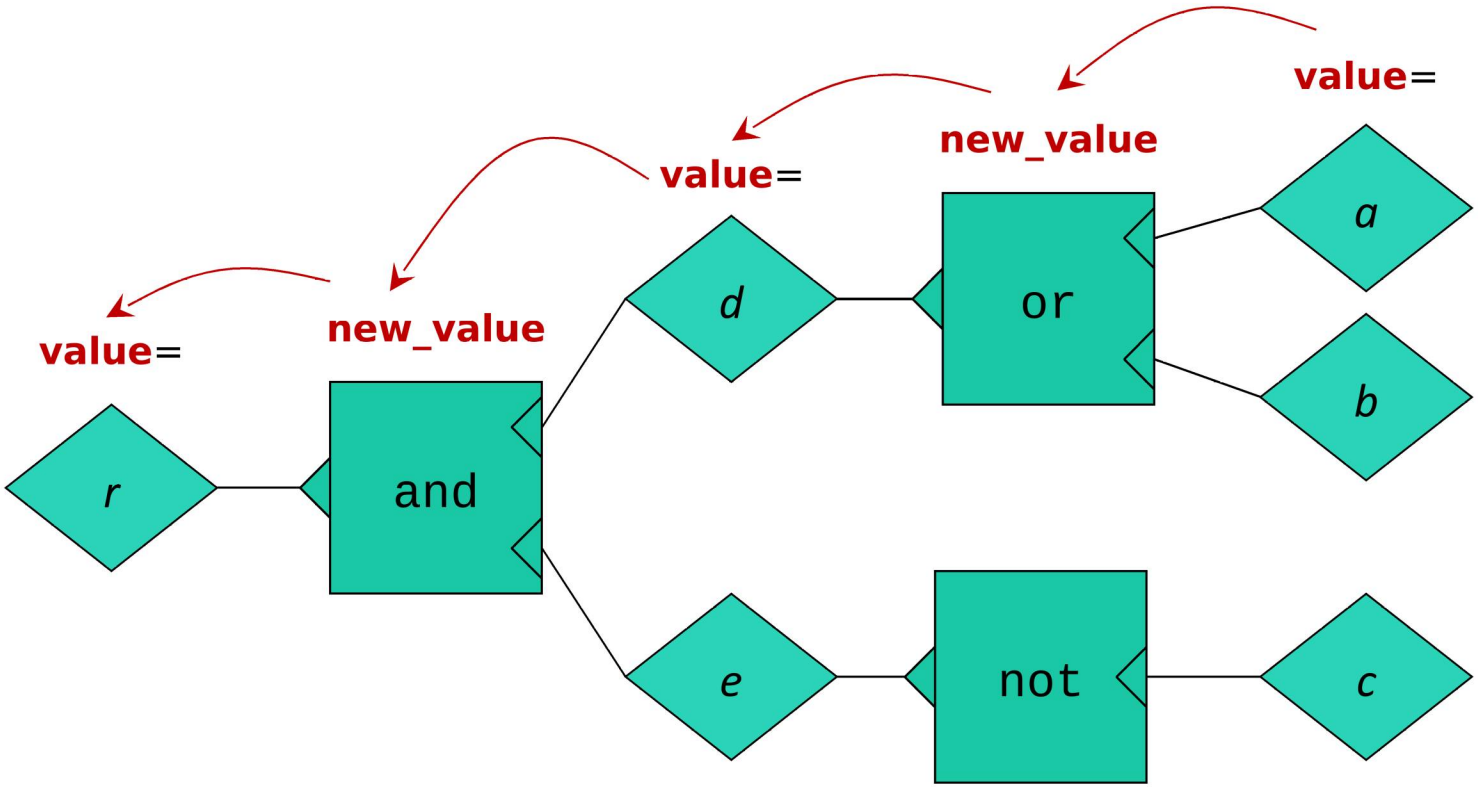
Hur värden propageras genom nätet



Hur värden propageras genom nätet

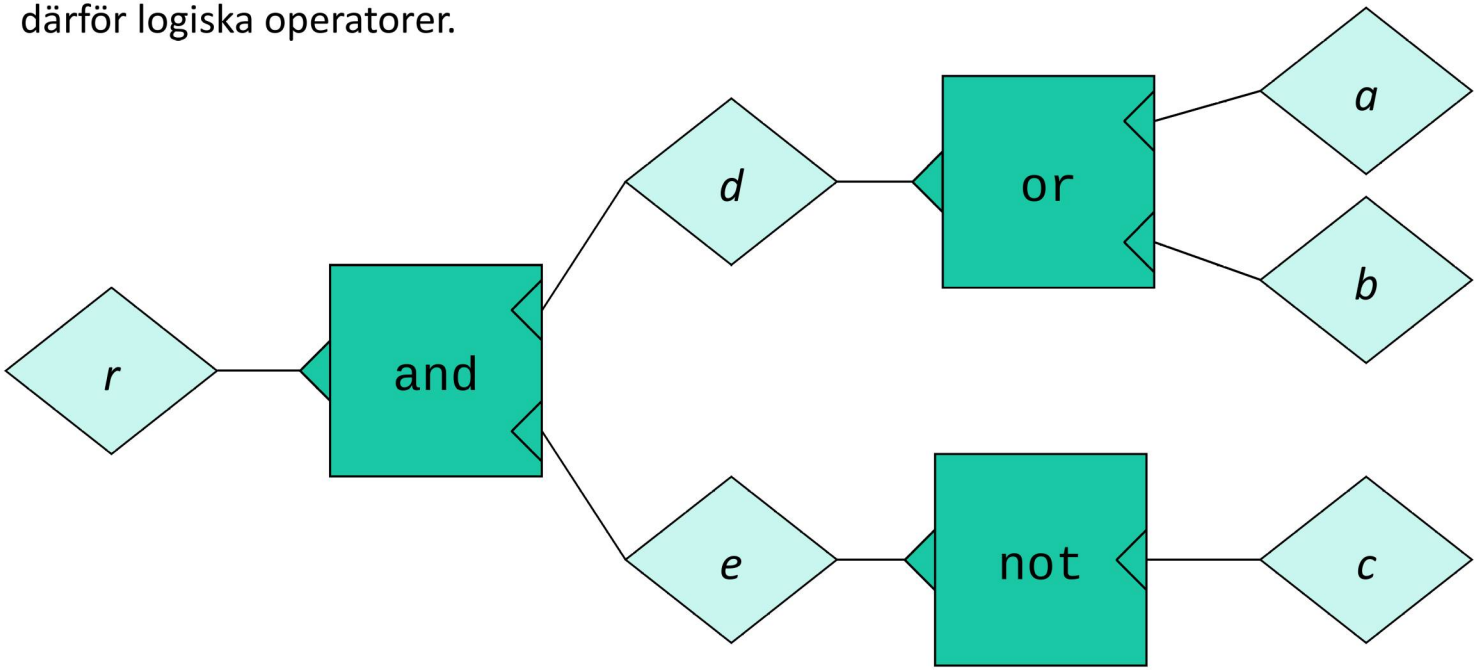


Hur värden propageras genom nätet



Övningsexempel

Detta är ett constraint network som jobbar med logiska värden istället för tal. Alla constraints är därför logiska operatorer.



Övning

```
def my_test
  $a = Wire.new('a')
  $b = Wire.new('b')
  $c = Wire.new('c')
  d = Wire.new('d')
  e = Wire.new('e')
  $r = Wire.new('r')

  ng = NotGate.new($c,e)
  og = OrGate.new($a,$b,d)
  ag = AndGate.new(d,e,$r)
end
```

Övning

Ladda in koden från constraint_test.rb *och kör metoden* my_test.
Vad händer om man ändrar a *till* true*? Och hur gör man det?*

Om uppgifterna inför seminariet

- Uppgift 1
  - Skriv tester för *constraints Adder* och *Multiplier*. Rätta eventuella fel som ni hittar.
  - Implementera ett nätverk för att konvertera mellan olika fysikaliska storheter
  - Skillnader gentemot logiska nätverket: Information ska kunna gå åt båda hållen och vi har inga default-värden.
- Uppgift 2
  - Gör färdigt en parser som kan läsa in en textuell beskrivning av *constraint*-nätverk och bygga upp ett sådant (inklusive tester).

Om uppgifterna inför seminariet

- Ickedeterministiska problemlösaren är inte med i seminariet.
- Kan fortfarande komma på dugga/tenta

Tips Uppgift 2

- Uppgift2 är inte ”stor” men den är klurig
- Testa först Uppgift 1 utförligt, var **SÄKRA** på att nätverket fungerar korrekt. Är viktigt.
- För att lösa uppgiften helt och hållet så krävs det att man förstår nätverket och vad parsern ska göra.
- Det går **INTE** (i min erfarenhet) att IP-bruteforca denna genom att jaga stacktraces och testa sig fram.
- Ta hjälp av assistenter på vägen

Sista föreläsningen

- Överblick av ”maskinkoden” i java (jvm)
- Återkoppling av kursens delar och mål
- Återkoppling om olika datorspråk
- Populäritet av språk
- Tips för förberedelser av opposition sem4 och dugga/tenta
- Föreläsningen är kortare än 2x45
- Slutet är hängivet till mer TDP019 diskussioner för de som vill

www.liu.se

LINKÖPINGS UNIVERSITET