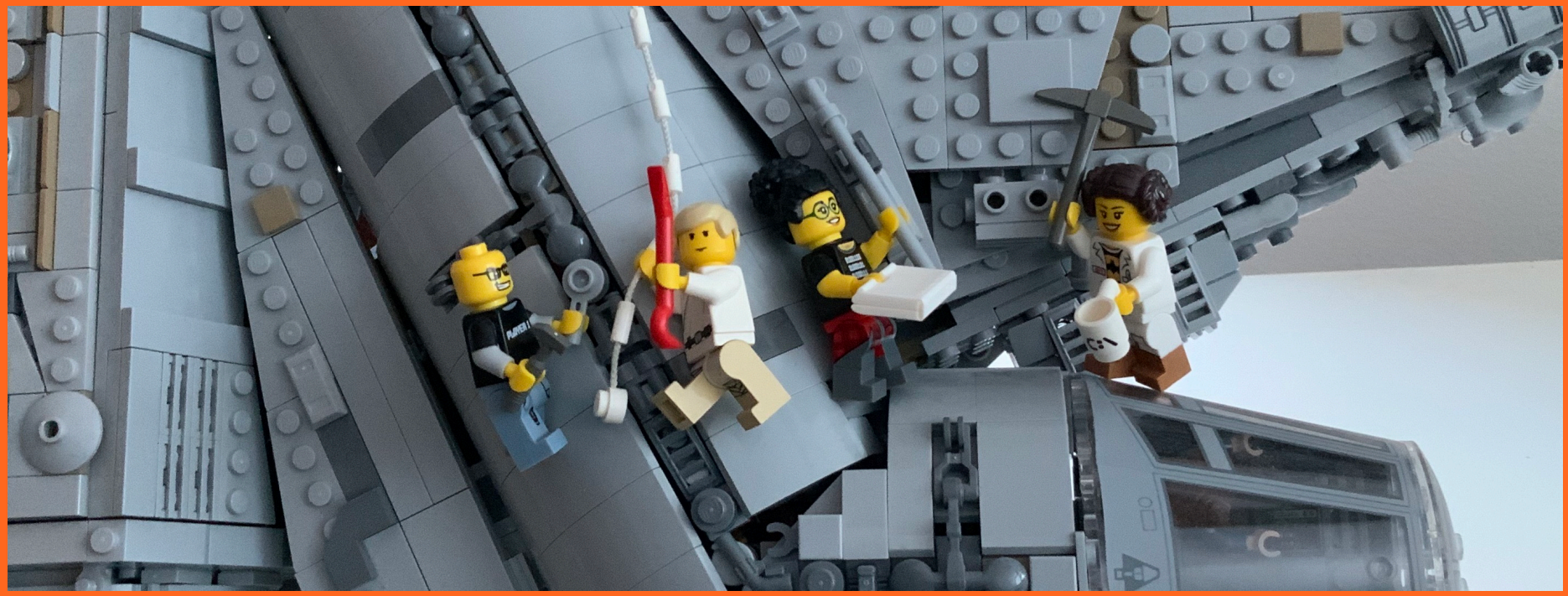
# CS-A1120 Ohjelmointi 2



Aalto-yliopisto
Perustieteiden
korkeakoulu

## Luento 8
## kevät 2026

Sanna Suoranta (suomeksi, T1 klo 14) ja
Lukas Ahrenberg (englanniksi, T1 klo 10)

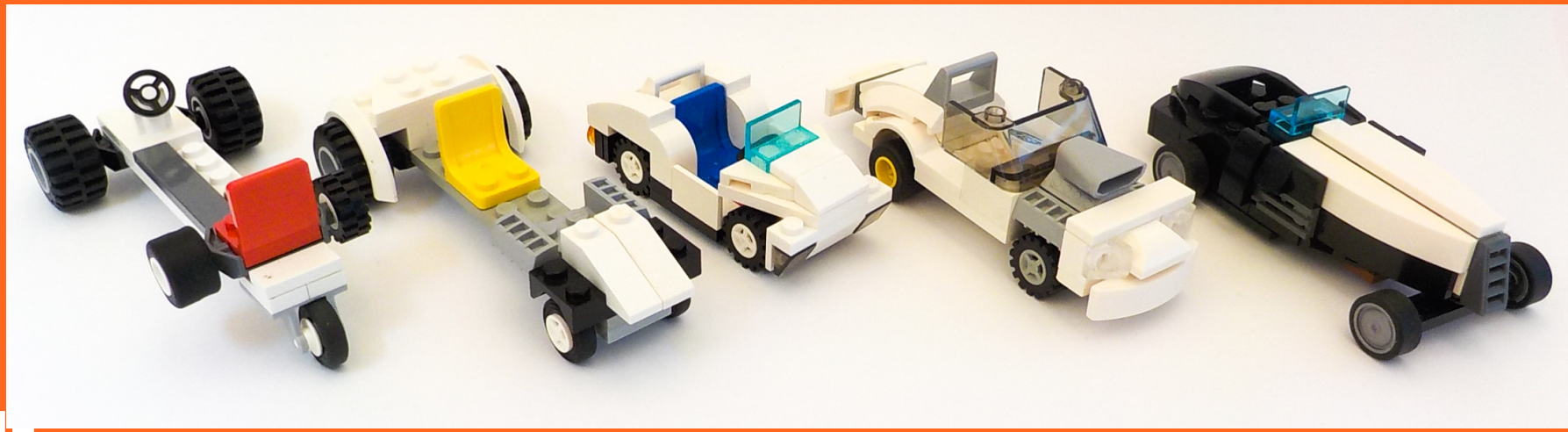
27.4.2026

https://presemo.aalto.fi/o2fi2026

# Tänään O2:ssa

https://presemo.aalto.fi/o2fi2026

## Rekursio (recursion)



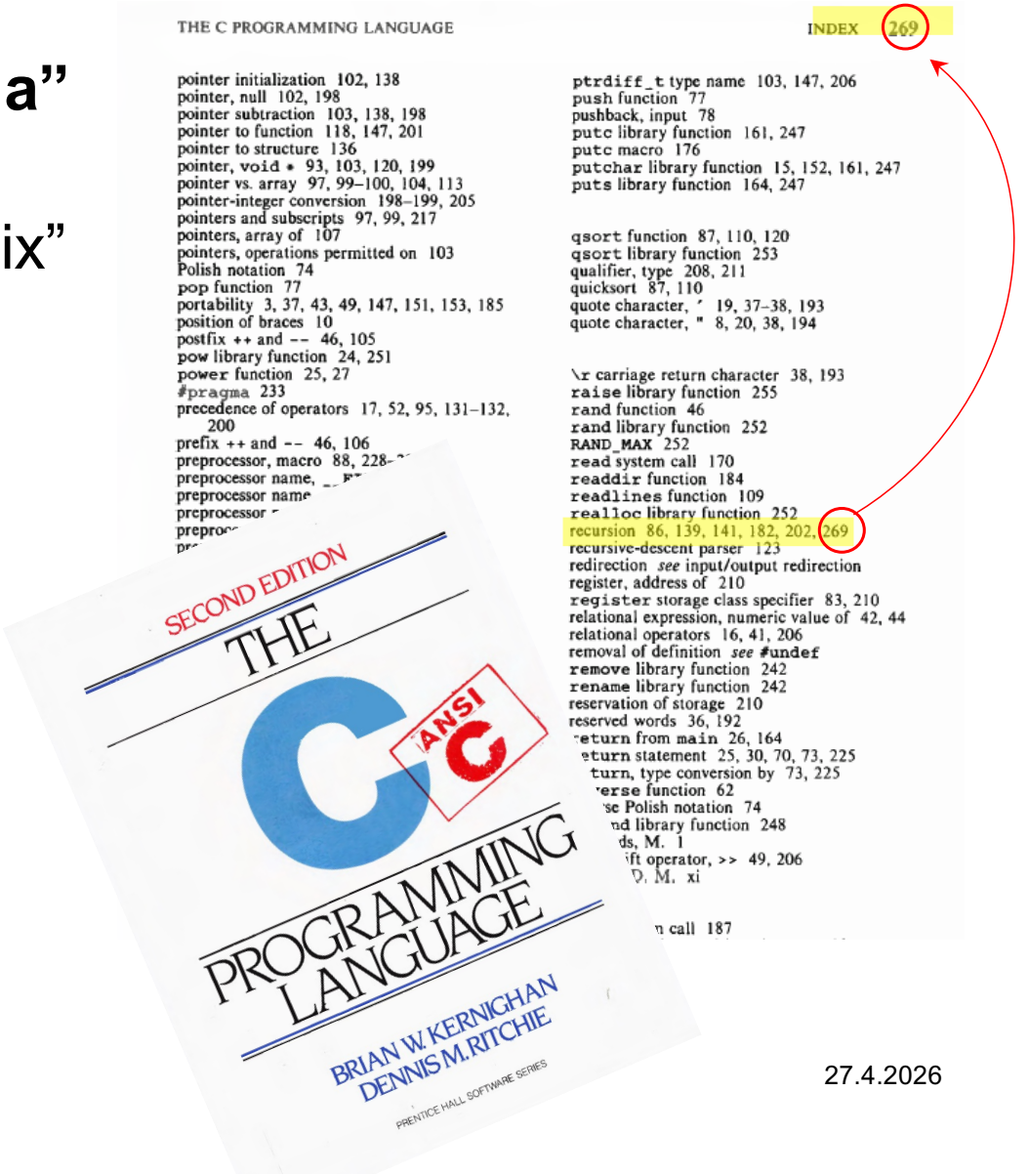
Aalto-yliopisto
Perustieteiden
korkeakoulu

# Kierroksen 8 oppimistavoitteet

- **Tämän viikon jälkeen,**
  - Olet kerrannut rekursion ja osaat määritellä rekursiivisen funktion sekä käyttää sisäfunktioita (inner function) ohjelmissasi
  - Olet tietoinen pinon ylivuoto –virheistä
  - Osaat selittää häntärekursion (tail recursion) ja
    - kirjoittaa häntärekursiivisia funktioita ja
    - näyttää rekursiivisia kutsuja kutsupinon ja laajennuksen avulla
  - Osaat määritellä rekursiivisia tietorakenteita kuten lista ja puu
  - Osaat käyttää rekursiivista ongelmanratkaisua,
    - erityisesti peruuttavaa hakua (backtracking search)

# Rekursion kertaus

- **“Määrittelee jotain itsensä avulla”**
  - Rekursio: katso rekursio
  - GNU stands for “GNU’s Not Unix”
  - $f(x) = x + f(x/2)$
  - Funktio, joka kutsuu itseään
  - Jne
- **Käytännössä rekursiivisella funktiolla on kaksi tapausta:**
  - Rekursiivinen tapaus, jossa funktio kutsuu itseään
  - Perustapaus, joka lopettaa rekursion



# Rekursion kertaus: `ispalindrome`

```
// Checks if String s is a palindrome.
// Assumes s is only lower case and does
// not contain any spaces or punctuation.
def isPalindrome(s: String): Boolean =
  // empty or one character?
  if s.length <= 1 then true  // base case
  // start and end differ?
  else if s.head != s.last then false // another base case
  // substring palindrome?
  else isPalindrome(s.substring(1, s.length - 1))
end isPalindrome
```

- **Yksi tai useampi perustapaus (base case), ja**
- **Funktion soveltaminen pienempään ongelma-instanssiin**

# Rekursion kertaus: sisäfunktiot

- **Useimmiten rekursio-askel (recursive step) on vain osa sitä, mitä haluamme tehdä**
  - Usein tehdään jotain prosessointia ensin syötteelle
- **Tällöin voimme määritellä sen käyttäen sisäfunktiota (inner function)**

# Rekursion kertaus: sisäfunktiot

```
// Checks if String s with punctuation and spaces removed
// is a palindrome. Ignores case and spaces.
def isPalindrome(s: String): Boolean =
// Helper inner function that encapsulates the actual recursive task.
// Assumes that the string contains only lower case characters
// and no spaces or punctuation.
  def actualRecursion(t: String): Boolean =
    if t.length <= 1 then true
    else if t.head != t.last then false
    else actualRecursion(t.substring(1, t.length - 1))
  end actualRecursion
  // Remove spaces and punctuation, transform to lower case
  val sPlain = s.filter(_.isLetterOrDigit).map(_.toLower)
  // Run the actual recursion and return the result
  actualRecursion(sPlain)
end isPalindrome
```

# Rekursion kertaus: kutsupino (call stack)

- **Funktio ottaa parametrejä, sillä voi olla sivuvaikutuksia ja se ylläpitää tilaa**
  - Kun alirutiinia (funktiota) suoritetaan, ohjelman pitää ‘muistaa’ nämä
- **Ohjelmointikielet käyttävät tähän kutsupinoa (call stack)**
  - Ohjelma varaa osan muistista pitääkseen kirjaa ohjelman tilasta funktiokutsun aikana
  - Tila lisätään (push, save) pinoon kun funktioon siirrytään, ja poistetaan (pop, return) sieltä kun funktio on suoritettu

```
isPalindrome("ufo tofu"):
 return actualRecursion("ufotofu")
  actualRecursion("ufotofu"):
   return actualRecursion("fotof")
    actualRecursion("fotof"):
     return actualRecursion("oto"):
      actualRecursion("oto"):
       return actualRecursion("t"):
        actualRecursion("t"):
         return true
```

# Rekursion kertaus: kutsupino (call stack)

- **Edellä esitetty `isPalindrome` havainnollistettuna käsin tehdyllä kutsupinolla**
- **Jos kutsumme tuota syötteellä “ufo tofu”, saamme kutsupinon:**
- **Huom:**
  - Viimeisin kutsu viimeisenä
  - `s.filter(_.isLetterOrDigit)` `.map(_.toLower)` piilotettu

```
isPalindrome("ufo tofu"):
 return actualRecursion("ufotofu")
  actualRecursion("ufotofu"):
   return actualRecursion("fotof")
    actualRecursion("fotof"):
     return actualRecursion("oto"):
      actualRecursion("oto"):
       return actualRecursion("t"):
        actualRecursion("t"):
         return true
```

# Rekursion kertaus: kutsupino (call stack)

- **Kutsupinon listaus (stack trace) on lista nykyisestä kutsupinosta**
  - Kaikki mitä on kutsuttu ennen kuin päädytään “tähän” tilaan
  - Se yleensä tulostetaan, kun tapahtuu ajonaikainen virhe tai poikkeus (eli tuttu juttu)
- **Scalan metodi `Thread.currentThread.getStackTrace` tuottaa kutsupinon listauksen array:nä**
- **Esim. (vähän hassu funktio “lähtölaskentaa” varten):**

```
// Calls itself recursively n times.
// Returns a string of the stack trace as base case
def rec(n: Int): String =
  if n == 1 then Thread.currentThread.getStackTrace.mkString("\n")
  else rec(n-1)+"" // Note: +"" to create need to keep state
```

# Rekursion kertaus: kutsupino (call stack)

```scala
// Calls itself recursively n times.
// Returns a string of the stack trace as base case
def rec(n: Int): String =
  if n == 1 then Thread.currentThread.getStackTrace.mkString("\n")
  else rec(n-1)+"" // Note: +"" to create need to keep state
```

```
scala> rec(4)
val res0: String = java.base/java.lang.Thread
  .getStackTrace(Thread.java:2451)
rs$line$1$.rec(rs$line$1:4)
rs$line$1$.rec(rs$line$1:5)
rs$line$1$.rec(rs$line$1:5)
rs$line$1$.rec(rs$line$1:5)
rs$line$2$.<clinit>(rs$line$2:1)
rs$line$2.res0(rs$line$2)
java.base/jdk.internal.reflect.DirectMethodHandleAccessor
  .invoke(DirectMethodHandleAccessor.java:103)
java.base/java.lang.reflect.Method.invoke(Method.java:580)
dotty.tools.repl.Rendering.$anonfun$4(Rendering.scala:119)
scala.Option.flatMap(Option.scala:283)
dotty.tools.repl.Rendering.valueOf(Rendering.scala:119)
// .. (Plus many more lines left out.)
```

# Laajentaminen (expansion)

- **Jos rekursiivinen funktio on sivuvaikutukseton (side-effect-free), voimme laajentaa (expand) sen**
  - Eli korvata funktion kutsu funktion määritelmällä
- **Analysointia varten hyvä**
- **Esim**

$$f(x) = x + f\left(\frac{x}{2}\right) = x + \frac{x}{2} + \frac{x}{4} + \frac{x}{8} + \dots$$

  - Tai `isPalindrome`:n tapauksessa, kunnes löydämme perustapauksen (base case):

```
isPalindrome("ufo tofu") =
actualRecursion("ufo tofu".filter(_.isLetterOrDigit).map(_.toLower)) =
actualRecursion("ufotofu".map(_.toLower)) =
actualRecursion("ufotofu") =
actualRecursion("fotof") =
actualRecursion("oto") =
actualRecursion("t") =
true
```

# Kutsupinon ylivuoto (stack overflow)

- **Esim: kertomafunktio**

$$n! = \begin{cases} 1 & \text{if } n = 1; \\ n \cdot (n-1)! & \text{if } n \geq 2\,. \end{cases}$$

```
def fact(n : Int): BigInt =
  require(n >= 1, "n should be a positive integer")
  if(n == 1) then BigInt(1)
  else n * fact(n-1)
end fact
```

# Kutsupinon ylivuoto (stack overflow)

- **Esim: kertomafunktio**

$$n! = \begin{cases} 1 & \text{if } n = 1; \\ n \cdot (n-1)! & \text{if } n \geq 2. \end{cases}$$

```
def fact(n : Int): BigInt =
  require(n >= 1, "n should be a positive integer")
  if(n == 1) then BigInt(1)
  else n * fact(n-1)
end fact
```

- **Mitä tapahtuu?**

```
scala> fact(10)
val res0: BigInt = 3628800
```

```
scala> fact(1000000)
java.lang.StackOverflowError
 at fact(<console>:2)
 at fact(<console>:4)
 ...
```

# Kutsupinon ylivuoto (stack overflow)

- **Kutsupinoon lisääminen (pushing) kuluttaa muistia**
- **Ajonaikaiselle kutsupinolle on varattu vain tietty määrä muistia**
- **Liian monta rekursiivista kutsua johtaa virheeseen `StackOverflowError`**
- **Näemme laajentaessamme kutsuja, että Scala ei voi yksinkertaistaa eikä laskea arvoa, ennen kuin koko rekursio purkautuu**

```
fact(4) =
 (4 * fact(3)) =
 (4 * (3 * fact(2))) =
 (4 * (3 * (2 * fact(1)))) =
 (4 * (3 * (2 * 1))) =
 (4 * (3 * 2)) =
 (4 * 6) =
 24
```

- **Tässä tapauksessa muistin käyttö on vähintään Ω (n)**

# Häntärekursio (tail recursion)

- **Eikös tätä voisi tehdä $\mathcal{O}(1)$-(muisti)tilavaatimuksella?**
- **Iteratiivinen ratkaisu pystyy:**

```
def fact(n : Int) : BigInt =
  require(n >= 1, "n should be a positive integer")
  var result = BigInt(1)
  for i <- 2 to n do
    result = result * i
  end for
  result
end fact
```

# Häntärekursio (tail recursion)

- **Ratkaisu: kirjoitetaan funktio käyttäen häntärekursiota (tail recursion)**
  - Scala-kääntäjä optimoi tämän iteraatioksi
- **Häntärekursio tarkoittaa, että rekursio kutsuu häntäkutsua (tail call): kutsutaan funktion viimeistä operaatiota (poislukien return-käsky), niin ei tarvi muistaa sitä enää**
- **Alkuperäisessä (alla) rekursiivisessa esimerkissämme viimeinen operaatio on kertolasku ***

```
def fact(n : Int): BigInt =
  require(n >= 1, "n should be a positive integer")
  if(n == 1) then BigInt(1)
  else n * fact(n-1)
end fact
```

# Häntärekursio ei kuluta pinosta tilaa

- **Kääntäjä voi laajentaa rekursion silmukan tapaan**
- **Muistathan lähtölaskenta-funktion ja sen kutsupinon listauksen:**

```
// Calls itself recursively n times.
// Returns a string of the stack trace as base case
def rec(n: Int): String =
  if n == 1 then Thread.currentThread.getStackTrace.mkString("\n")
  else rec(n-1)+"" // Note: +"" to create need to keep state
```

- `rec(n-1)+""` **ei ole häntärekursiivinen (+ on häntäkutsu)**
- **Voimme kirjoittaa sen ilman ketjutusta**

```
// Calls itself recursively n times.
// Returns a string of the stack trace as base case
def rec(n: Int): String =
  if n == 1 then Thread.currentThread.getStackTrace.mkString("\n")
  else rec(n-1) // Note: recursion is now a tail call!
```

# Häntärekursio ei kuluta pinosta tilaa

- **Nyt uusi rec-funkiton kutsu, jossa on vain yksi rec-kutsu. Kääntäjä on huomannut häntärekursion ja poistanut kutsupinoon laitetut kehykset**

Vain yksi →

```
scala> rec(4)
val res1: String = java.base/java.lang.Thread
  .getStackTrace(Thread.java:2451)
rs$line$3$.rec(rs$line$3:4)
rs$line$4$.<clinit>(rs$line$4:1)
rs$line$4.res1(rs$line$4)
java.base/jdk.internal.reflect.DirectMethodHandleAccessor
  .invoke(DirectMethodHandleAccessor.java:103)
java.base/java.lang.reflect.Method.invoke(Method.java:580)
dotty.tools.repl.Rendering.$anonfun$4(Rendering.scala:119)
scala.Option.flatMap(Option.scala:283)
dotty.tools.repl.Rendering.valueOf(Rendering.scala:119)
dotty.tools.repl.Rendering.renderVal(Rendering.scala:159)
dotty.tools.repl.ReplDriver.$anonfun$7(ReplDriver.scala:421)
scala.runtime.function.JProcedure1.apply(JProcedure1.java:15)
scala.runtime.function.JProcedure1.apply(JProcedure1.java:10)
scala.collection.immutable.List.foreach(List.scala:334)
dotty.tools.repl.ReplDriver
  .extractAndFormatMembers$1(ReplDriver.scala:421)   … …
```

# Häntärekursio (tail recursion)

- **Häntärekursiivinen versio kertomasta laskee kertolaskun ennen kutsua:**

```
import scala.annotation.tailrec
def fact(n : Int) : BigInt =
  require(n >= 1, "n should be a positive integer")
  // Inner recursive function using two parameters:
  // - i keep track of the recursion 'level
  // - result is the value of (i-1)!
  @tailrec def iterate(i : Int, result : BigInt) : BigInt =
    if i > n then result
    else iterate(i+1, result*i)
  end iterate
  iterate(2, BigInt(1))
end fact
```

# Häntärekursio (tail recursion)

- **Häntärekursiivinen versio kertomasta laskee kertolaskun ennen kutsua:**

```
import scala.annotation.tailrec
def fact(n : Int) : BigInt =
  require(n >= 1, "n should be a positive integer")
  // Inner recursive function using two parameters:
  // - i keep track of the recursion 'level
  // - result is the value of (i-1)!
  @tailrec def iterate(i : Int, result : BigInt) : BigInt =
    if i > n then result
    else iterate(i+1, result*i)
  end iterate
  iterate(2, BigInt(1))
end fact
```

- Perustapaus annetaan kutsuttaessa ikäänkuin alkuarvoksi

# Häntärekursio (tail recursion)

- **Häntärekursiivinen versio kertomasta laskee kertolaskun ennen kutsua:**

```
import scala.annotation.tailrec
def fact(n : Int) : BigInt =
  require(n >= 1, "n should be a positive integer")
  // Inner recursive function using two parameters:
  // - i keep track of the recursion 'level
  // - result is the value of (i-1)!
  @tailrec def iterate(i : Int, result : BigInt) : BigInt =
    if i > n then result
    else iterate(i+1, result*i)
  end iterate
  iterate(2, BigInt(1))
end fact
```

- `@tailrec`-merkintä ei ole välttämätön, mutta auttaa Scala-kääntäjää huomaamaan, että tämän oli tarkoitus olla häntärekursiivinen, jolloin se varoittaa, jos häntärekursio ei ole mahdollinen

# Kutsupinon laajennus kertomalle

- **Tavallinen rekursio**

```
fact(4):
 val temp1= fact(3)
  fact(3):
   val temp2 = fact(2)
    fact(2):
     val temp3 = fact(1)
      fact(1):
       return 1
     return 2*temp3
   return 3*temp2
 return 4*temp1
```

- **Häntärekursiivinen versio**

```
fact(4):
 return iterate(2, 1)
  iterate(2, 1):
   return iterate(3, 1*2)
  iterate(3, 2):
   return iterate(4, 2*3)
  iterate(4, 6):
   return iterate(5, 6*4)
  iterate(5, 24):
   return 24
```

# Rekursiiviset tietorakenteet

## (recursive data structures)

Aalto-yliopisto
Perustieteiden
korkeakoulu

# Rekursiiviset tietorakenteet

- **Funktioiden lisäksi myös tietorakenne (data structures) voidaan määritellä rekursiivisesti**
- **Niiden käsittely on luonnollista rekursiivisten funktioiden avulla**
- **Tällä luennolla katsomme kahta esimerkkiä:**
  - Linkitetty lista (samanlainen kuin `List` Scalassa)
  - Symboliset aritmeettiset lausekkeet (kuten 2x + 4y -7), joka on esimerkki yleisemmästä puumaisesta tietorakenteesta (tree-like data structure)

# Linkitetty lista (linked list)

- **Linkitetty lista on tietorakenne, joka voi esittää sekvenssiä**
- **Se voidaan määritellä rekursiivisesti, sillä lista on**
  - Joko tyhjä tai
  - Elementti, jota seuraa lista
- **Formaalimmin, määritellään `T-List` listana, joka sisältää `T`-tyyppisiä elementtejä:**
  - `Nil` merkitsee tyhjää `T-list`aa, tai
  - `Cons(e, l)`, jossa `e` on elementti tyyppiä `T` ja `l` on `T-list`
  - `e` on listan pää (head) ja häntä (tail) on lista `Cons(e,l)`
  - Nimitykset tulevat Lisp-kielestä: ‘Cons’ = construction eli rakenne ja ‘Nil’ viittaa numeroon nolla

# Linkitetty lista (linked list)

- **Esimerkkejä:**
  - `Nil` on tyhjä lista `[]`
  - `Cons(1, Nil)` on `Int-List[1]`
  - `Cons('c', Cons('a', Cons('b', Nil)))` on `String-List['c', 'a', 'b']`


# Linkitetty lista – miksi vaivautua?

- **Dynaaminen: Elementin lisääminen listan alkuun on $\mathcal{O}(1)$**
  - Lisääminen ja poistaminen ei vaadi datan uudelleenorganisointia (mutta $\mathcal{O}(n)$)
- **Monipuolinen: voi käyttää perusrakenteena muille tietorakenteille, esim.**
  - Pino
  - Jono
  - Puu, …
- **Mutta, kuten aina, yksittäinen tietorakenne ei sovi ihan kaikkiin tarpeisiin**
- **Toteutamme linkitetyn listan oppimistarkoituksissa**

# Linkitetyn listan luokka Scalassa

- **Tyypin T lista on**
  - Nil, tai
  - Sisältää pään ja hännän, jossa
    - pää on elementti tyyppiä T ja
    - häntä on toinen T-lista
- **T on tyyppiparametri***
  - Polymorfinen, yleinen, eli sisältö määritellään vasta käytettäessä

```
abstract class LinkedList[T]:
  // List empty (Nil)
  // or not (Cons)?
  def isEmpty: Boolean
  // Data
  def head: T
  // Rest of the list
  def tail: LinkedList[T] end
LinkedList
```



* https://docs.scala-lang.org/tour/polymorphic-methods.html

# Linkitetyn listan luokka Scalassa

- **Konkreettiset luokat (tapausluokat) määrittelevät `Nil` ja `Cons`:**

```
case class Nil[T]() extends LinkedList[T]:
  // Nil is empty
  def isEmpty = true
  // head is undefined in Nil
  def head = throw new java.util
    .NoSuchElementException("head of empty list")
  // tail is undefined in Nil
  def tail = throw new java.util
    .NoSuchElementException("tail of empty list")
end Nil

case class Cons[T](val head: T, val tail: LinkedList[T])
  extends LinkedList[T]:
  // Cons is not empty
  def isEmpty = false
  // Rest is created automatically
  // in the case class
end Cons
```

# Linkitetyn listan luokka Scalassa – käyttö

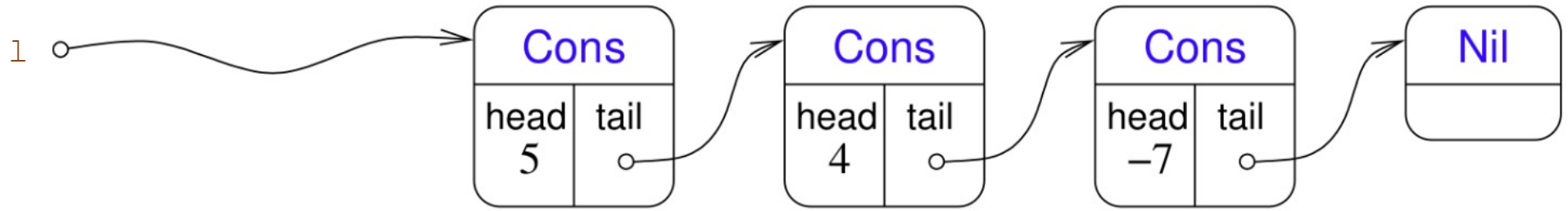
- **Voimme rakentaa listoja:**

```
scala> val l = Cons(5, Cons(4, Cons(-7, Nil())))
val l: Cons[Int] = Cons(5,Cons(4,Cons(-7,Nil())))

scala> l.head
val res7: Int = 5

scala> l.tail
val res8: LinkedList[Int] = Cons(4,Cons(-7,Nil()))
```

- **Huomaa rekurssiivinen rakenne:**



# Linkitetyn listan luokka Scalassa – käyttö

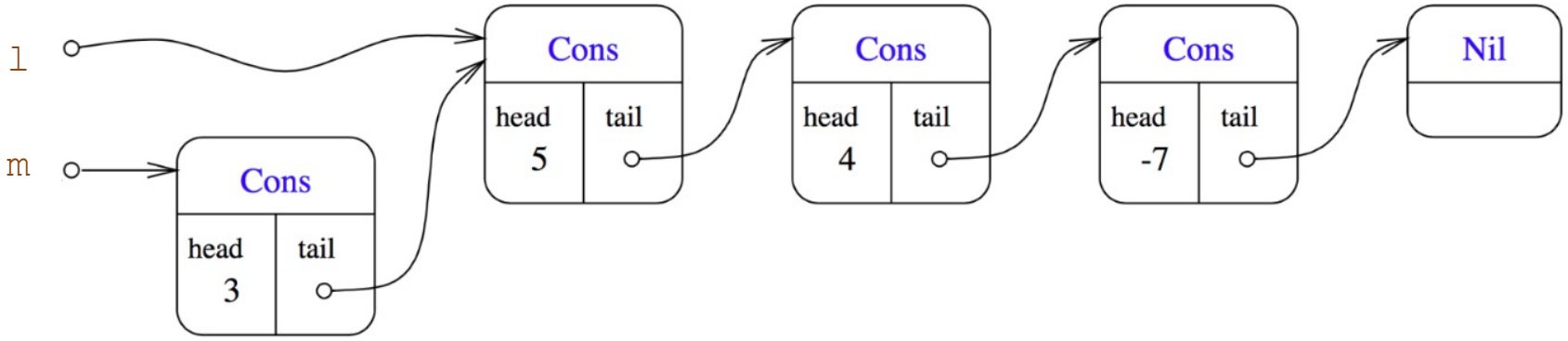
- **Lisääminen alkuun on helppoa**

```
scala> val m = Cons(3,l)
val m: Cons[Int] = Cons(3,Cons(5,Cons(4,Cons(-7,Nil()))))

scala> m.head
val res9: Int = 3

scala> m.tail == l // note, this is letter l, not number 1
val res10: Boolean = true
```

- **Huomaathan, että aikaisempaa listaa `l` ei kopioida!**



# Linkitetyn listan luokka Scalassa – metodit

- **Mitä metodeita tarvitsisimme `LinkedList` –luokkaan?**
- **Mitä haluaisimme pystyä tekemään listalla?**

https://presemo.aalto.fi/o2fi2026

- **Rekursiivisia tietorakenteita – rekursiivisia metodeita**

# Linkitetyn listan pituus – `length`

- **Usein tarvitaan kokoelman elementtien määrää**
- **Perustapaus: tyhjän listan pituus on 0**
- **Rekursiolla: pituus on on 1 + hännän pituus**

```
abstract class LinkedList[T]:
  // Previous code left out ...

  def length : Int = if isEmpty then 0 else 1 + tail.length
end LinkedList
```

- **Onko tämä häntärekursiivinen?**

https://presemo.aalto.fi/o2fi2026

# Linkitetyn listan pituus – `length`

```
abstract class LinkedList[T]:
  // Previous code left out ...

  def length : Int = if isEmpty then 0 else 1 + tail.length
end LinkedList
```

- **Onko tämä häntärekursiivinen?**
- **No ei ole. Laajentamalla saadaan**

```
Cons(1, Cons(2, Cons(3, Nil()))).length =
1 + Cons(2, Cons(3, Nil())).length =
1 + (1 + Cons(3, Nil()).length) =
1 + (1 + (1 + Nil().length)) =
1 + (1 + (1 + 0)) =
1 + (1 + 1) =
1 + 2 =
3
```

# Linkitetyn listan pituus – `length`

- **Häntärekursiivinen versio**

```
def length: Int =
  // Tail-recursive inner function, accumulating the result
  @tailrec def inner(remaining: LinkedList[A], result:Int): Int =
    if remaining.isEmpty then result
    else inner(remaining.tail, result+1)
  end inner
  inner(this, 0)
end length
```

- **Laajentamalla:**

```
Cons(1, Cons(2, Cons(3, Nil()))).length =
inner(Cons(1, Cons(2, Cons(3, Nil()))), 0) =
inner(Cons(2, Cons(3, Nil())), 1) =
inner(Cons(3, Nil()), 2) =
inner(Nil(), 3) = 3
```

# Elementti linkitetyssä listassa – `contains`

- **Onko jokin elementti `e` listassa?**
- **Perustapaus: tyhjä lista ei voi sisältää elementtiä e**
- **Rekursiolla: onko elementti `e` sama kuin listan pää? Muutoin, tarkista onko `e` hännässä**

```
abstract class LinkedList[T]:
  // Previous code left out ...
  def contains(e: T): Boolean =
    if isEmpty then false
    else if head == e then true
    else tail.contains(e)
  end contains
end LinkedList
```

- Onko `contains` häntärekursiivinen?
- Mikä on sen aikavaativuus (time complexity)?

# Elementti linkitetyssä listassa – `contains`

```
abstract class LinkedList[T]:
  // Previous code left out ...
 @tailrec final def contains(e: T): Boolean =
    if isEmpty then false
    else if head == e then true
    else tail.contains(e)
  end contains
end LinkedList
```

- **Onko `contains` häntärekursiivinen? On**
  - Huom. `@tailrec`-metodit pitää määritellä olemaan `final` (estää metodin korvaamisen perittävissä alaluokissa), jos se ei ole sisäfunktio
- **Mikä on sen aikavaativuus (time complexity)? $\mathcal{O}$(n)**

# Vaihtoehtoinen toteutus `contains`-metodille

- **Voimme tehdä saman käyttäen hahmonsovitusta**

```
abstract class LinkedList[T]:
// Previous code left out ...
  @tailrec final def contains(e: T): Boolean =
  this match
    case Nil() => false
    case Cons(h, t) => if (h == e) then true else t.contains(e)
  end match e
end contains end LinkedList
```

- **Kurssimateriaalista löytyy vastaava `length`-metodille, jossa**

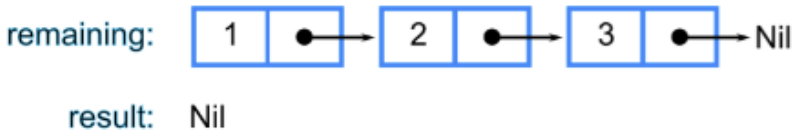
```
case Cons(_, t) => inner(t, result + 1)
```

# Linkitetyn listan kääntäminen – `reverse`

- **Kuinka voimme kääntää listan järjestyksen, esim. 1,2,3→3,2,1?**

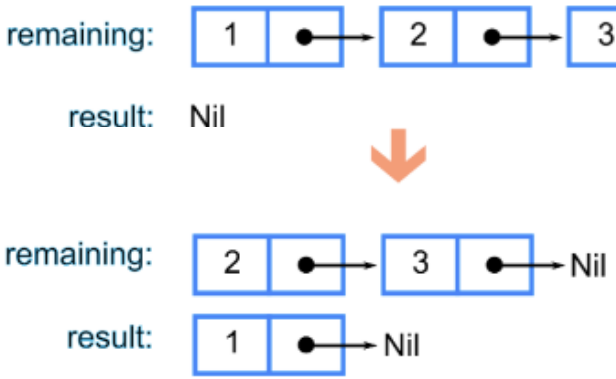
```
abstract class LinkedList[T]:
// Previous code left out ...
  def reverse: LinkedList[T] =
    @tailrec def inner(remaining: LinkedList[T],
      result: LinkedList[T]): LinkedList[T] =
      remaining match
        case Nil() => result
        case Cons(h, t) => inner(t, Cons(h, result))
      end match
    end inner
    inner(this, Nil())
  end reverse
end LinkedList
```

# Linkitetyn listan kääntäminen – `reverse`



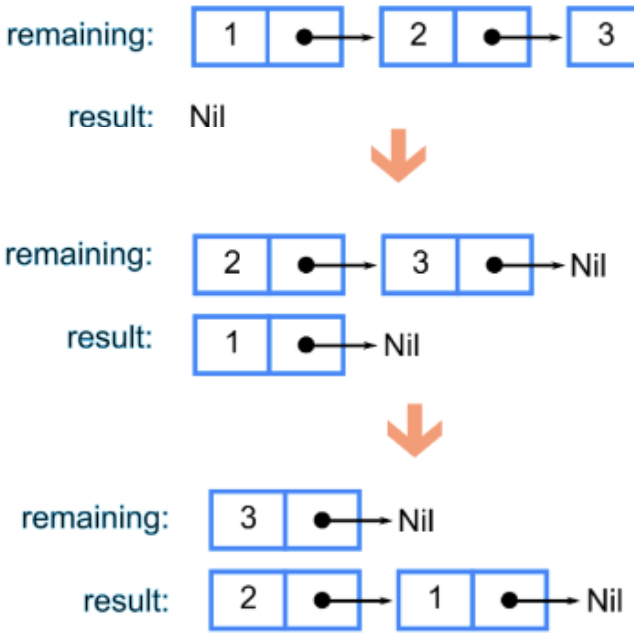

```
abstract class LinkedList[T]:
// Previous code left out ...
  def reverse: LinkedList[T] =
    @tailrec def inner(remaining: LinkedList[T],
      result: LinkedList[T]): LinkedList[T] =
      remaining match
        case Nil() => result
        case Cons(h, t) => inner(t, Cons(h, result))
      end match
    end inner
    inner(this, Nil())
  end reverse
end LinkedList
```

# Linkitetyn listan kääntäminen – `reverse`



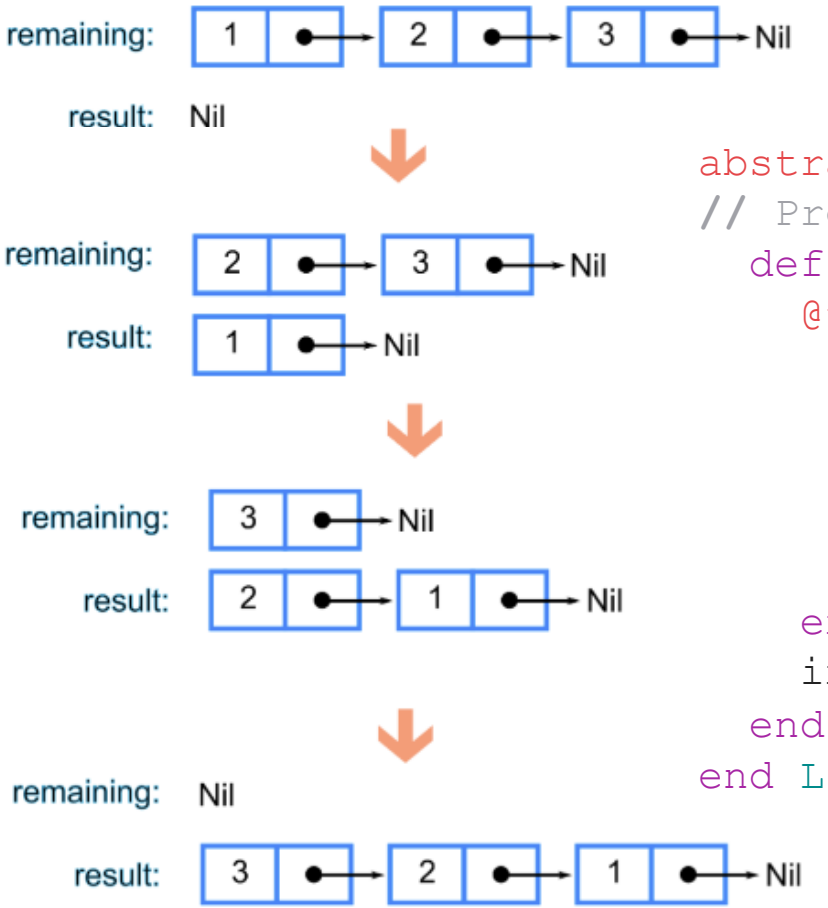

```
abstract class LinkedList[T]:
// Previous code left out ...
  def reverse: LinkedList[T] =
    @tailrec def inner(remaining: LinkedList[T],
      result: LinkedList[T]): LinkedList[T] =
      remaining match
        case Nil() => result
        case Cons(h, t) => inner(t, Cons(h, result))
      end match
    end inner
    inner(this, Nil())
  end reverse
end LinkedList
```

# Linkitetyn listan kääntäminen – `reverse`




```
abstract class LinkedList[T]:
// Previous code left out ...
  def reverse: LinkedList[T] =
    @tailrec def inner(remaining: LinkedList[T],
      result: LinkedList[T]): LinkedList[T] =
      remaining match
        case Nil() => result
        case Cons(h, t) => inner(t, Cons(h, result))
      end match
    end inner
    inner(this, Nil())
  end reverse
end LinkedList
```

# Linkitetyn listan kääntäminen – `reverse`




```
abstract class LinkedList[T]:
// Previous code left out ...
  def reverse: LinkedList[T] =
    @tailrec def inner(remaining: LinkedList[T],
      result: LinkedList[T]): LinkedList[T] =
      remaining match
        case Nil() => result
        case Cons(h, t) => inner(t, Cons(h, result))
      end match
    end inner
    inner(this, Nil())
  end reverse
end LinkedList
```

# Toinen esimerkki: lausekkeet


Aalto-yliopisto
Perustieteiden
korkeakoulu

# Lausekkeet (expressions)

- **Toinen rekursiivisesti määriteltävissä oleva tietorakenne on symboliset aritmeettiset lausekkeet (symbolic arithmetic expression)**
- **Nämä koostetaan**
  - Numeerisista vakioista (numerical constants)
  - Muuttujista (variables)
  - Operaatioista (operations)
- **Esim**
  - 2,5
  - $x$
  - $3*x + 8*(x - y) - 5$

# Lausekkeet – määritelmä

- **Oletetaan, että operaatioita, joita haluamme käyttää, ovat**
  - lisäys, vähennys, tulo, ja negaatio
- **Meillä on seuraavat rekursiiviset määritelmät lausekkeelle:**

1. **Vakio (kuten 3.0) on lauseke**
2. **Muuttuja (kuten `x`) on lauseke**
3. **Jos $e_1$ ja $e_2$ ovat lausekkeita, sitten $(e_1 + e_2)$, $(e_1 - e_2)$, $(e_1 * e_2)$ ja $-e_1$ ovat myös lausekkeita**
4. **Ei ole muita lausekkeita**

# Lausekkeet – esimerkki:

- `2.0` on lauseke
- `x` on lauseke
- `y` on lauseke
- `(2.0 * x)` on lauseke
- Niinpä `(2.0 * x) + y` on lauseke
- Niinpä `-((2.0 * x) + y)` on lauseke

- Scalalla tämän toteuttaessamme voimme käyttää abstraktia luokkaa ja tapausluokkia erityyppisille lausekkeille

# Lausekkeet – Scala-toteutus

```
abstract class Expr:
// More here soon..
end Expr
/** Variable expression, like "x" or "y". */
case class Var(name: String) extends Expr:
  override def toString = name
end Var
/** Constant expression like "2.1" or "-1.0" */
case class Num(value: Double) extends Expr:
  override def toString = value.toString
end Num
/** Expression formed by multiplying two expressions, like "x * (y+3)" */
case class Multiply(left: Expr, right: Expr) extends Expr:
  override def toString = "(" + left + "*" + right + ")"
end Multiply

// continues…
```

# Lausekkeet – Scala-toteutus

```
abstract class Expr:
// More here soon..
end Expr

// continues…
/** Expression formed by adding two expressions, like "x + (y*3)" */
case class Add(left: Expr, right: Expr) extends Expr:
  override def toString = "(" + left + " + " + right + ")"
end Add
/** Expression formed by subtracting an expression from another, "x - (y*3)" */
case class Subtract(left: Expr, right: Expr) extends Expr:
  override def toString = "(" + left + " - " + right + ")"
end Subtract
/** Negation of an expression, like "-((3*y)+z)" */
case class Negate(p: Expr) extends Expr:
  override def toString = "-" + p
end Negate
```

# Lausekkeet – Scala-toteutus

- **Nyt voimme määritellä lausekkeen `-((2.0 * x) + y)`**

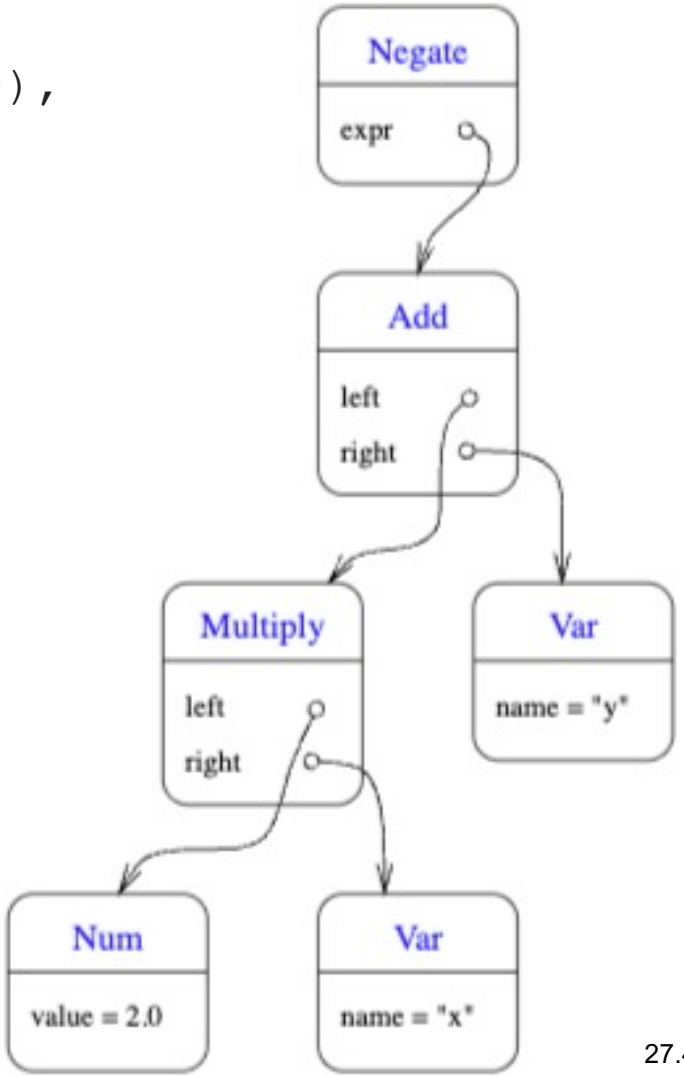
```
scala> val e = Negate(Add(Multiply(Num(2.0),
     |  Var("x")),
     |  Var("y")))
     |
val e: Negate = -((2.0*x) + y)
```

- **Tapausluokkien `toString`-metodin korvaamisen (override) ansiosta tämä jopa printtautuu nätisti**
- **Huomaa, että 'pää' on tyyppiä `Negate`, koska se on uloin lauseke**
- **Tämä toimii esimerkkinä puumaisesta tietorakenteesta**

# Lausekkeet – Scala-toteutus

```
scala> val e = Negate(Add(Multiply(Num(2.0),
     | Var("x")),
     | Var("y")))
     |
val e: Negate = -((2.0*x) + y)
```

- **Osalla lausekkeista on kaksi operandia, osalla vain yksi**
  - Kaksi ‘häntää’ yhteen-, vähennys- ja kertolaskulla
  - Negaatiolla yksi ‘häntä’



# Lausekkeet – Scala-toteutus

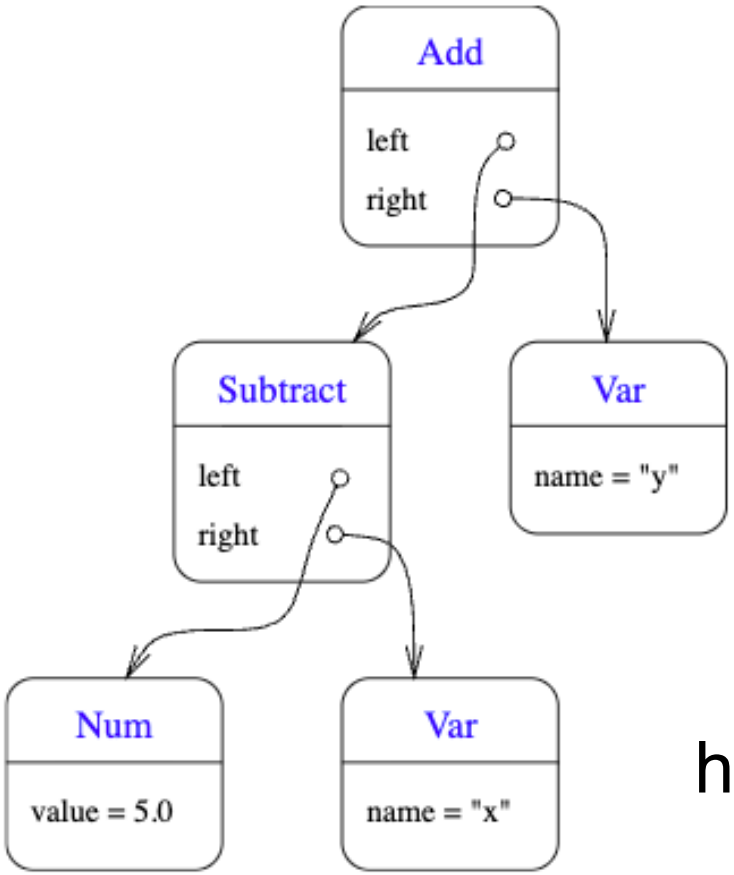
- **Voimme korvata operaattorit `Expr`-luokassa:**

```
abstract class Expr:
  /* Overriding these operators enable us to construct
     expressions with infix notation */
  def +(other: Expr) = Add(this, other)
  def -(other: Expr) = Subtract(this, other)
  def *(other: Expr) = Multiply(this, other)
  def unary_- = Negate(this)
end Expr
// Case-classes as before here...
```

**helpottaaksemme lausekkeiden kirjoittamista:**

```
scala> -( Num(2.0) * Var("x") + Var("y"))
val res: Negate = -((2.0*x) + y)
```

# Lauseke-quizz




- **Mikä Scala-lauseke luo tämän puun?**

https://presemo.aalto.fi/o2fi2026

# Lausekkeet – Evaluointi (evaluation)

- **Metodi lausekkeen arvon määrittämiseksi eli evaluoimiseksi olisi näppärä. Eli, jos kaikille muuttujille on määritelty arvo, `Expr` palauttaisi numeerisen arvon**
  - Annettuna: kuvaus (map) muuttujien nimistä niiden arvoiksi
  - Perustapaus: `Num`:n arvo on sen arvo
  - Perustapaus: `Var`:n arvo on kuvauksessa annettu arvo
  - Rekursiolla: operaattorilausekkeen (operator expression) arvo saadaan soveltamalla operaattoria operandilausekkeiden (operand expressions) arvoon

# Lausekkeet – Evaluointi (evaluation)

```
abstract class Expr:
  // Previous code left out ... /
  ** Custom exception for when a variable isn't assigned. */
  class VariableNotAssignedException(message: String)
    extends java.lang.RuntimeException(message)

    def evaluate(p: Map[String, Double]): Double =
    this match case Var(n) => p.get(n) match {
      case Some(v) => v
      case None => throw new VariableNotAssignedException(n)
    }
    case Num(v) => v
    case Multiply(l, r) => l.evaluate(p) * r.evaluate(p)
    case Add(l, r) => l.evaluate(p) + r.evaluate(p)
    case Subtract(l, r) => l.evaluate(p) - r.evaluate(p)
    case Negate(t) => -t.evaluate(p)
  end evaluate
end Expr
```

# Lausekkeet – Evaluointiesimerkki

- **Evaluoi `-((2.0*x)+y)`, kun `x = 4.5` ja `y = 1.2`:**

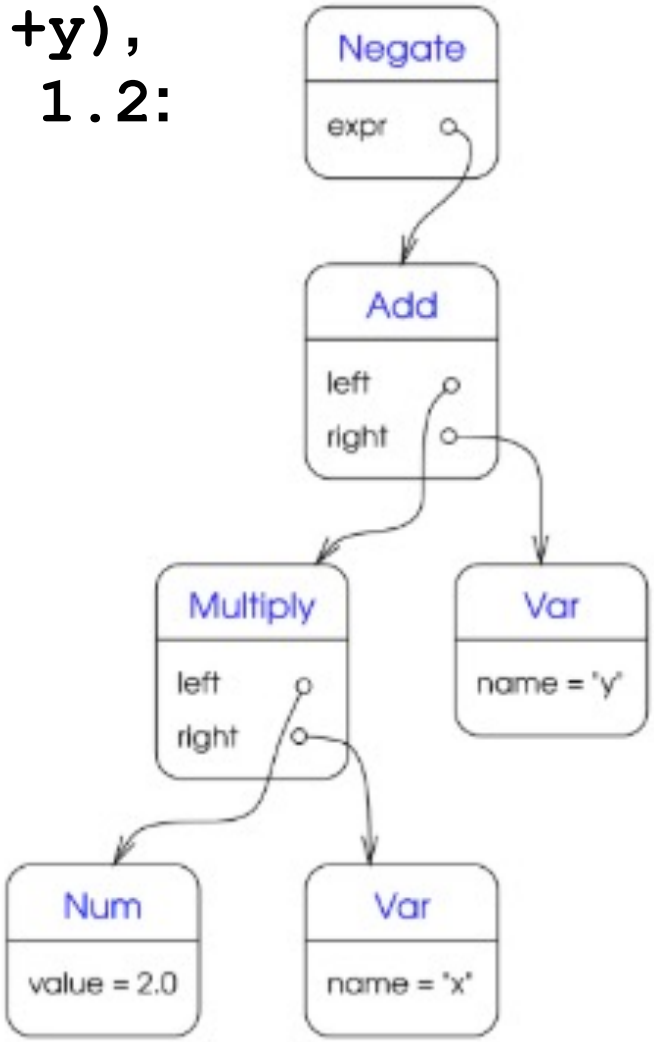
```
scala> val e1 = -( Num(2.0) * Var("x") + Var("y"))
val e1: Negate = -((2.0*x) + y)

scala> val p = Map("x" -> 4.5, "y" -> 1.2)
val p: scala.collection.immutable.Map ...

scala> e1.evaluate(p)
```

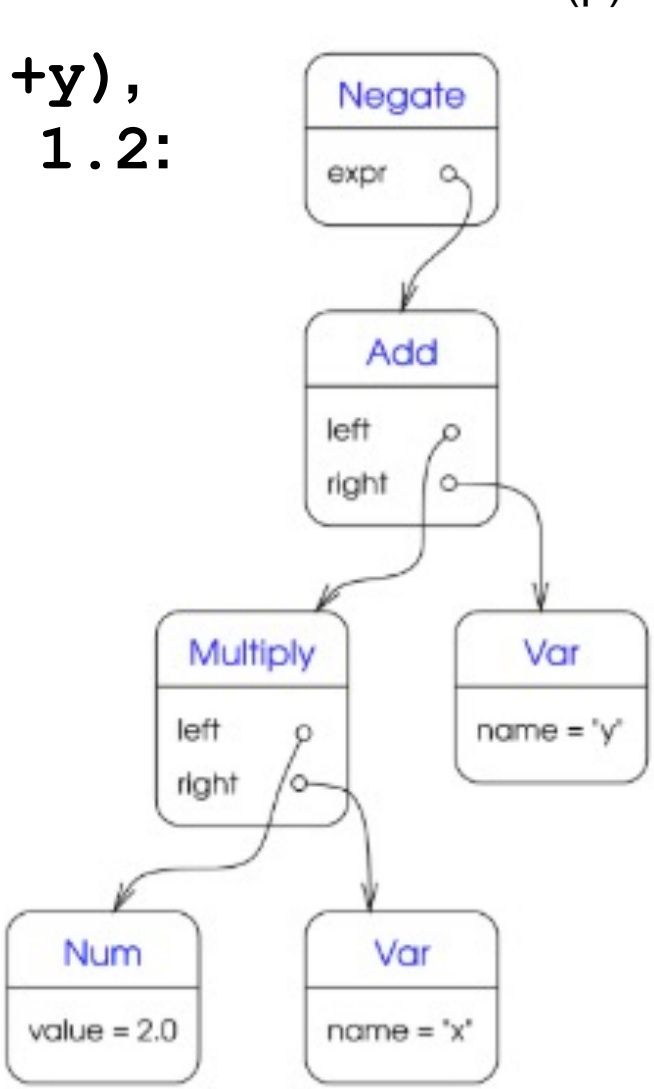
# Lausekkeet – Evaluointiesimerkki

- **Evaluoi `-((2.0*x)+y)`, kun `x = 4.5` ja `y = 1.2`:**



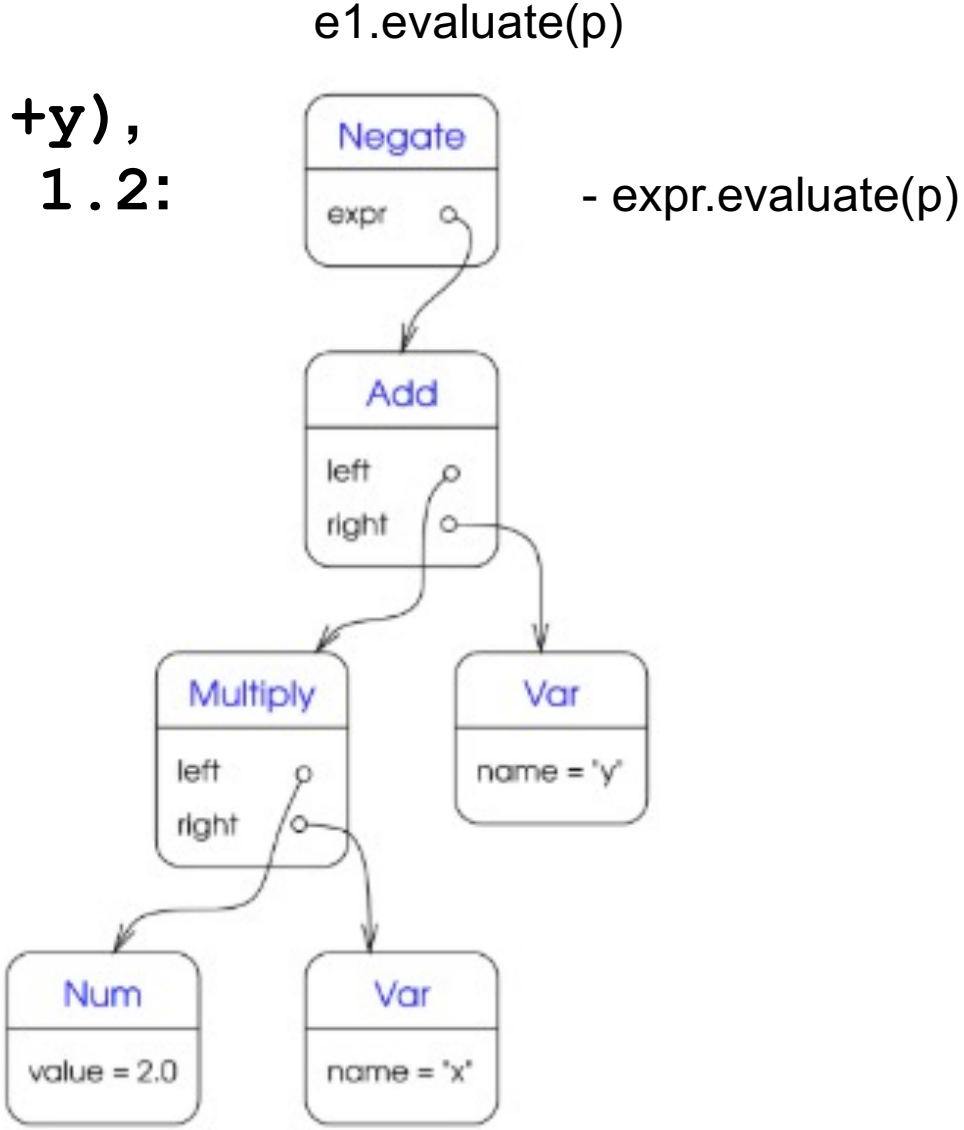
# Lausekkeet – Evaluointiesimerkki

- **Evaluoi `-((2.0*x)+y)`, kun `x = 4.5` ja `y = 1.2`:**



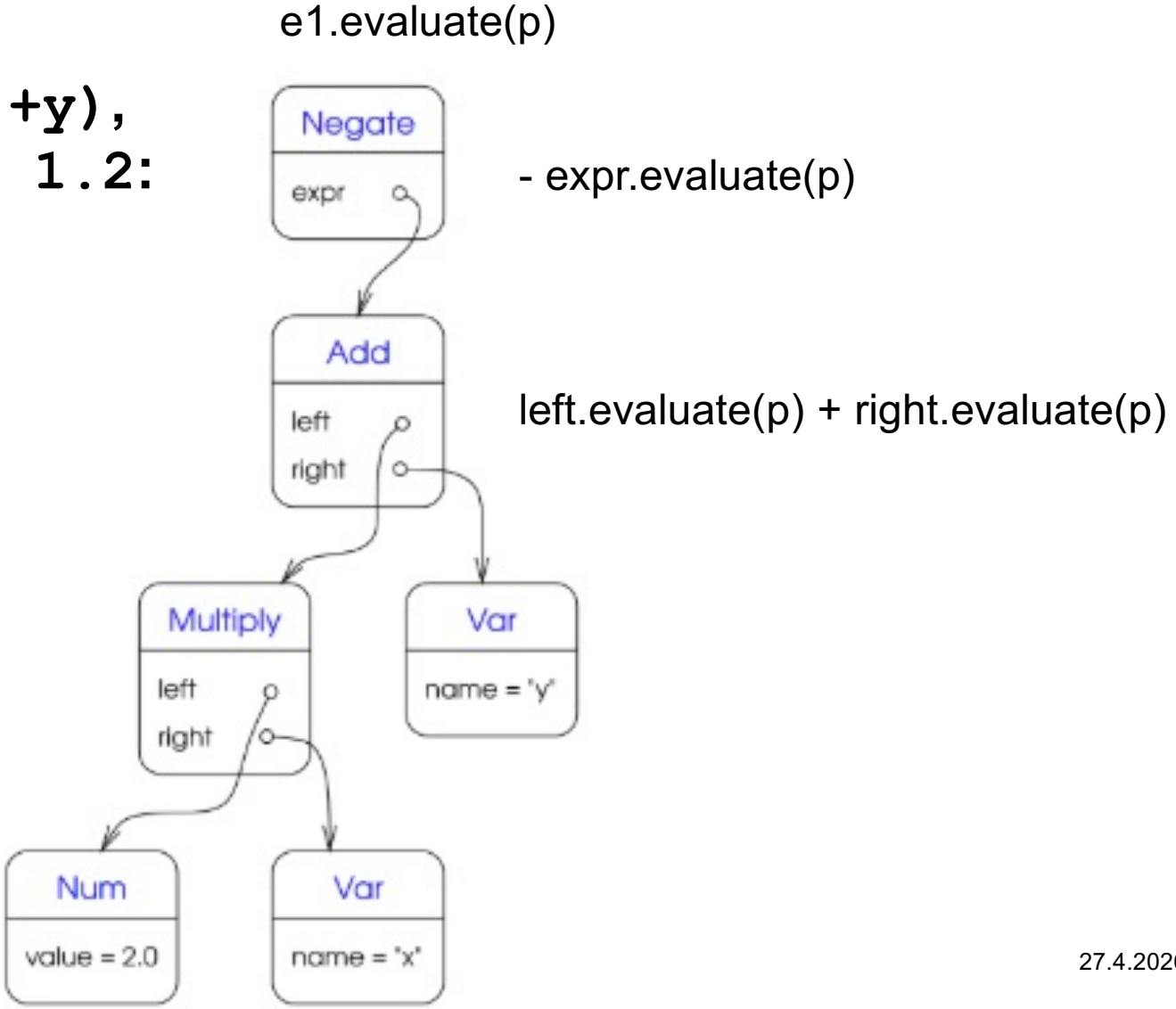
# Lausekkeet – Evaluointiesimerkki

- **Evaluoi `-((2.0*x)+y)`, kun `x = 4.5` ja `y = 1.2`:**



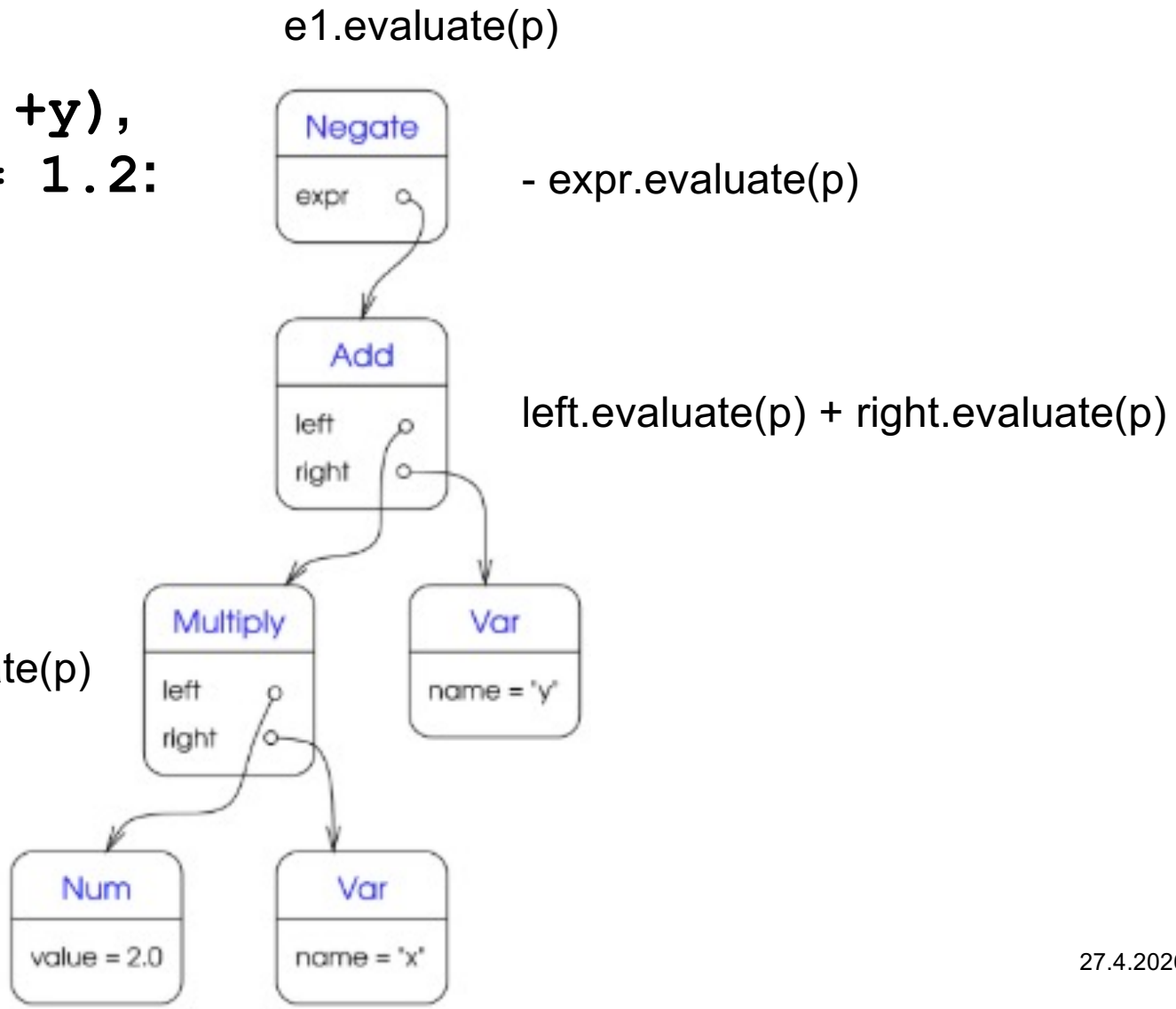
# Lausekkeet – Evaluointiesimerkki

- **Evaluoi `-((2.0*x)+y)`, kun `x = 4.5` ja `y = 1.2`:**



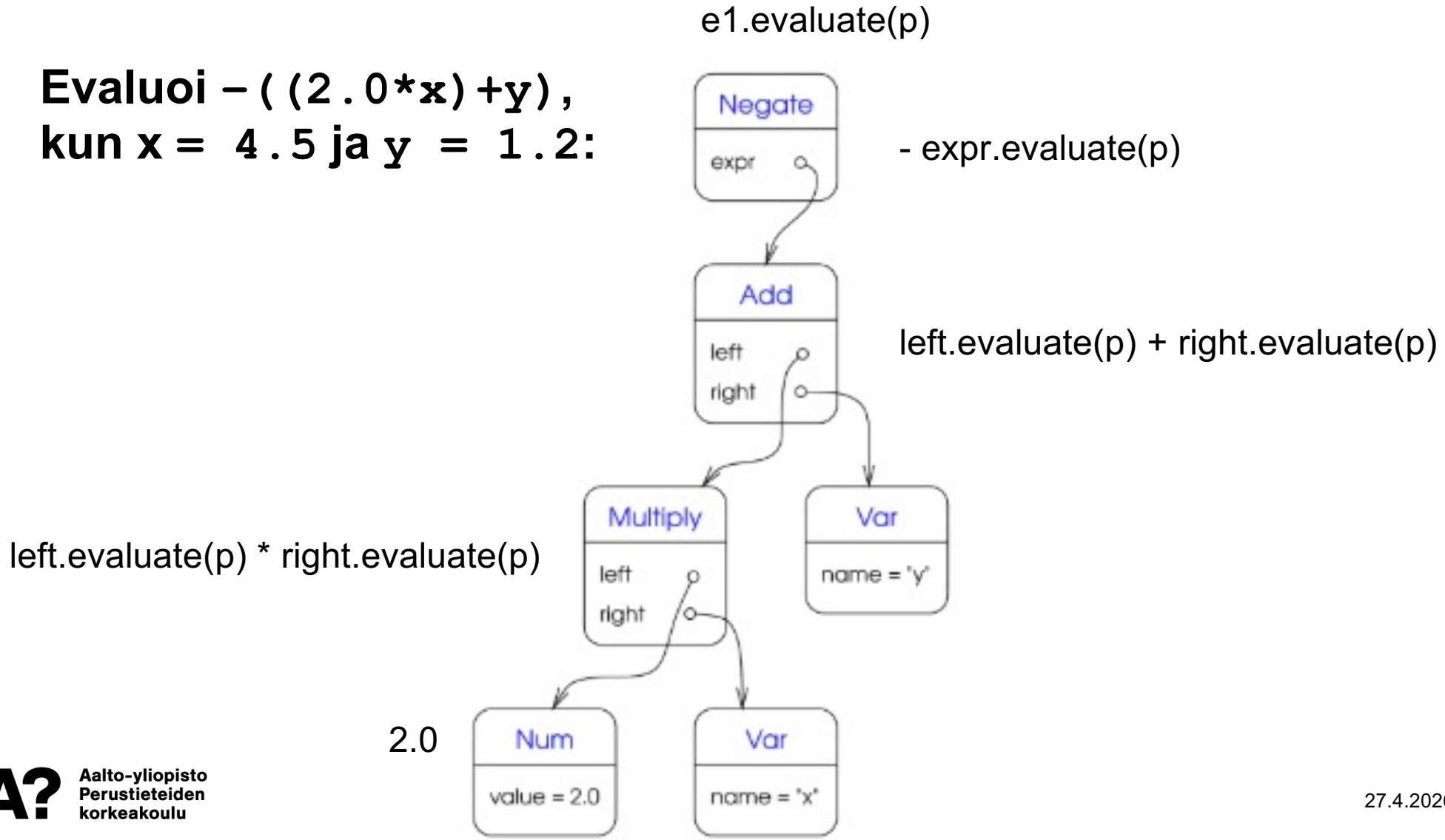
# Lausekkeet – Evaluointiesimerkki

- **Evaluoi `-((2.0*x)+y)`, kun `x = 4.5` ja `y = 1.2`:**



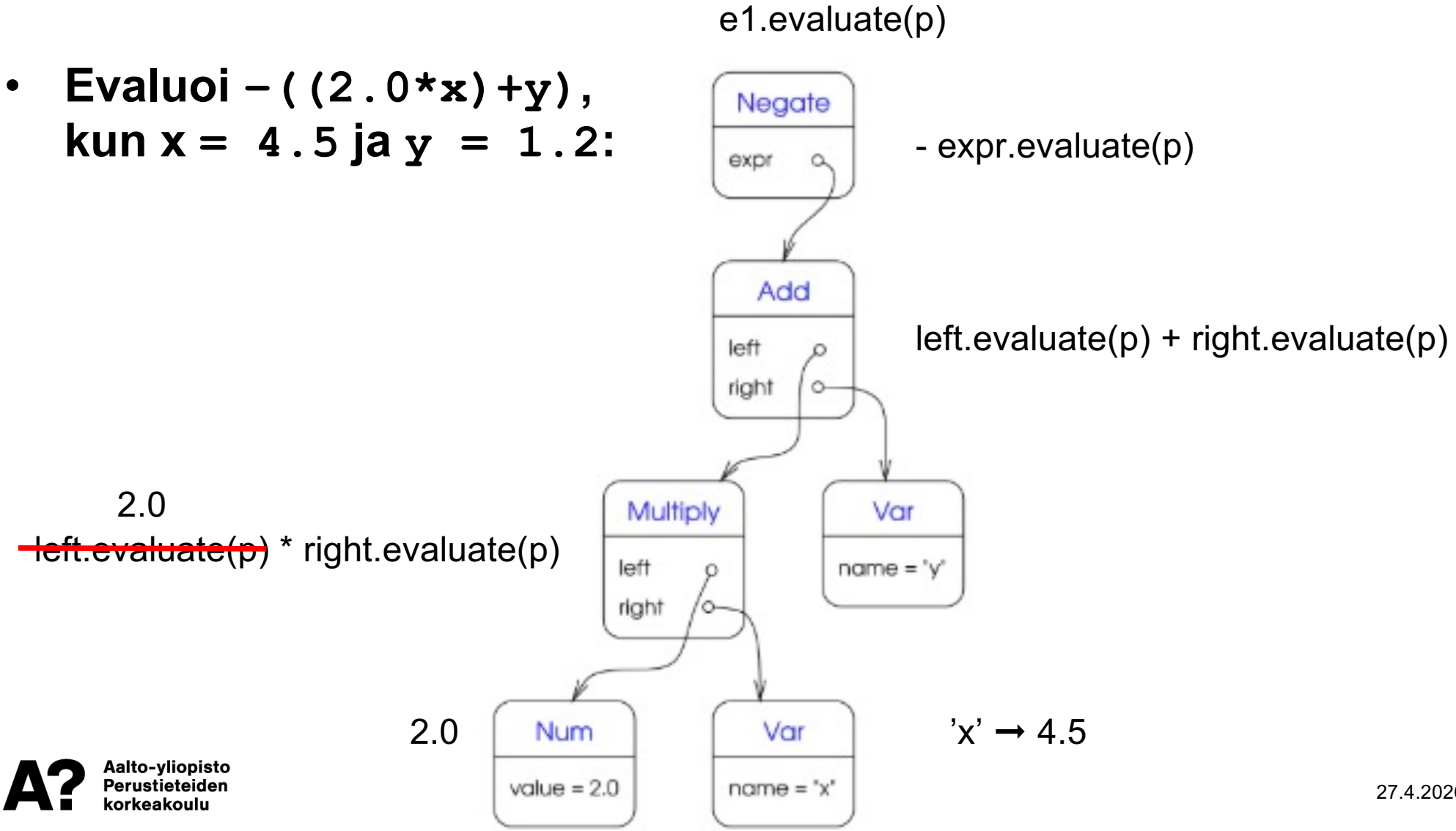
# Lausekkeet – Evaluointiesimerkki

- **Evaluoi `-((2.0*x)+y)`, kun `x = 4.5` ja `y = 1.2`:**



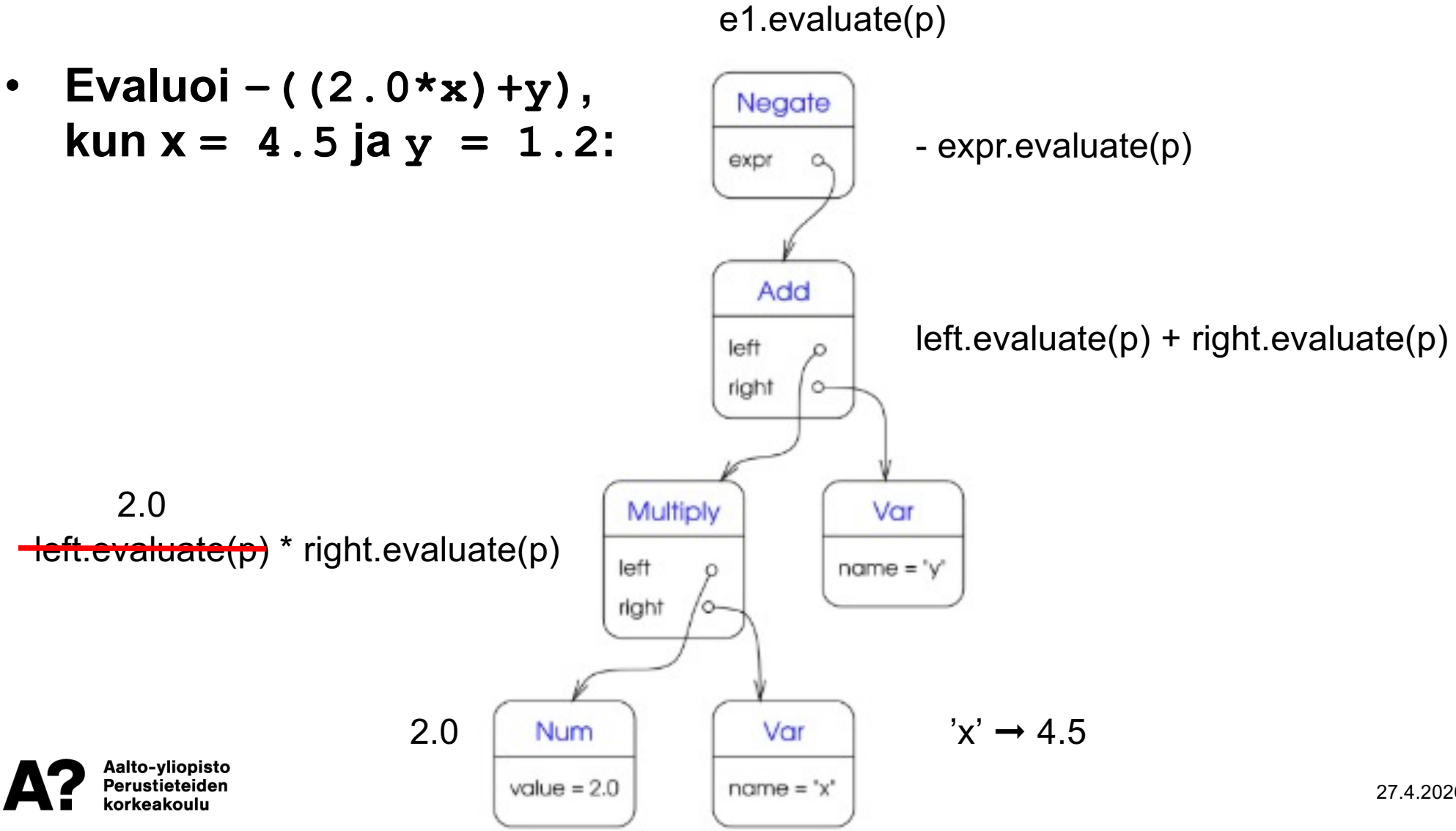
# Lausekkeet – Evaluointiesimerkki

- **Evaluoi `-((2.0*x)+y)`, kun `x = 4.5` ja `y = 1.2`:**



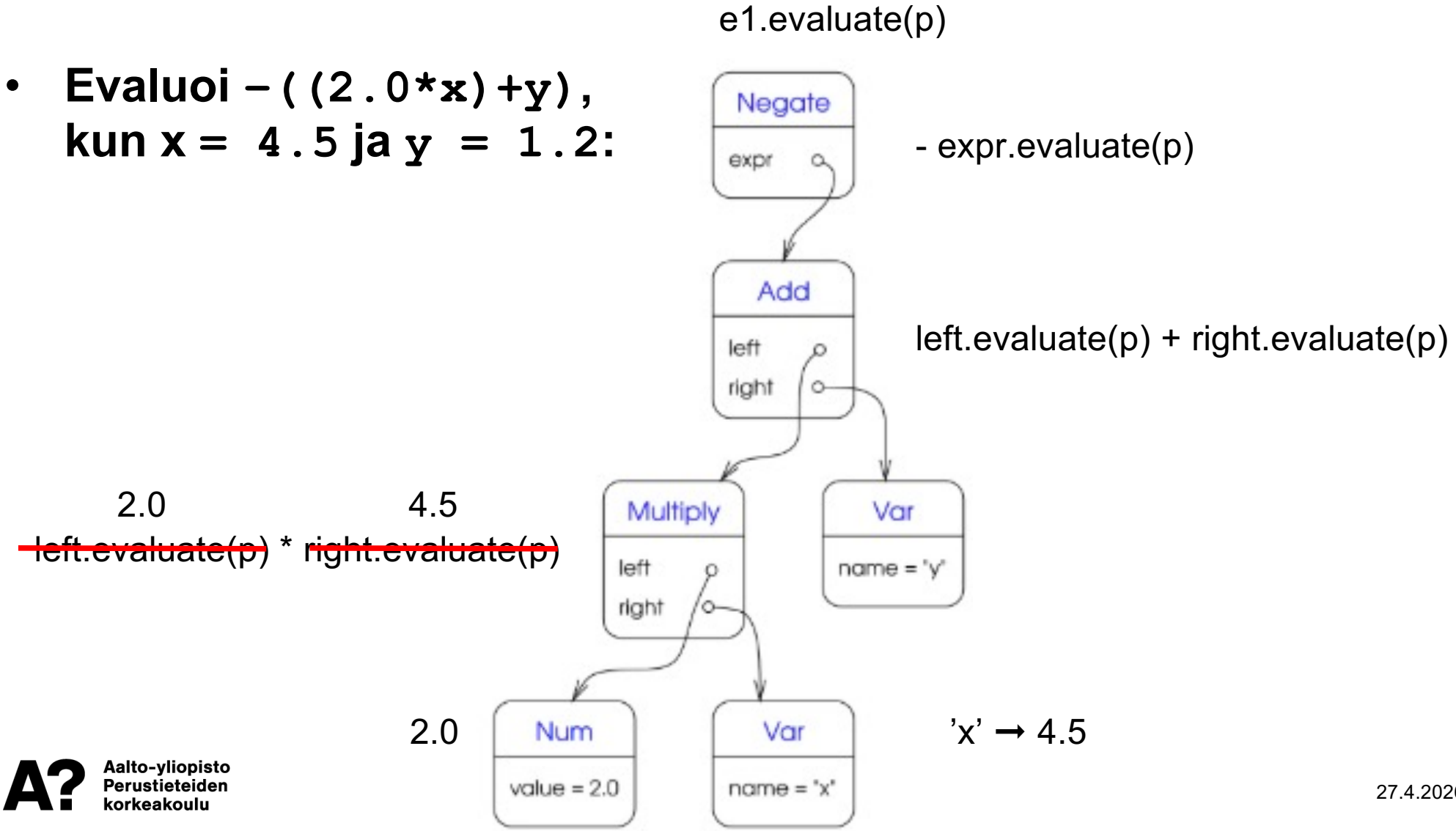
# Lausekkeet – Evaluointiesimerkki

- **Evaluoi `-((2.0*x)+y)`, kun `x = 4.5` ja `y = 1.2`:**



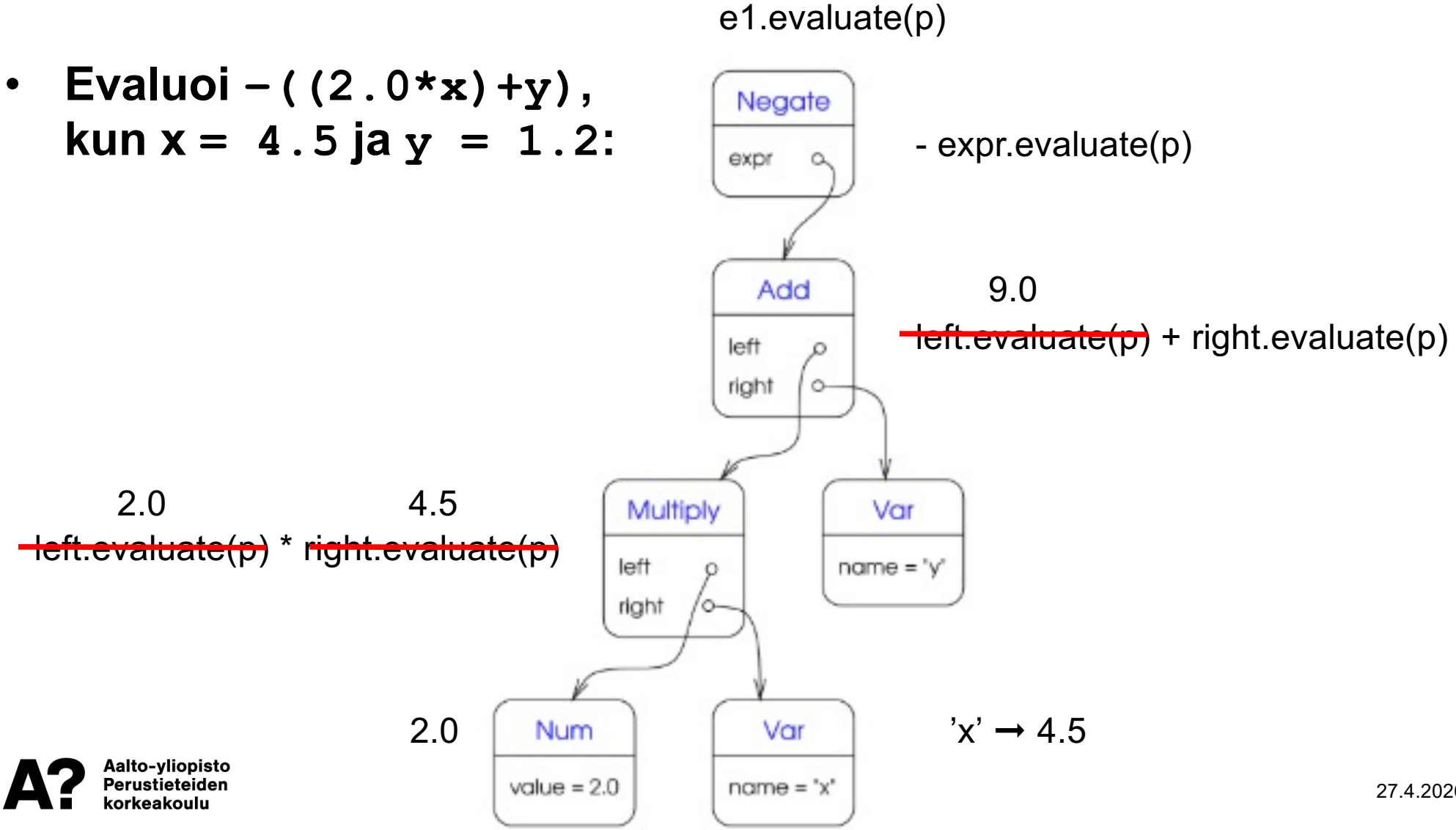
# Lausekkeet – Evaluointiesimerkki

- **Evaluoi `-((2.0*x)+y)`, kun `x = 4.5` ja `y = 1.2`:**



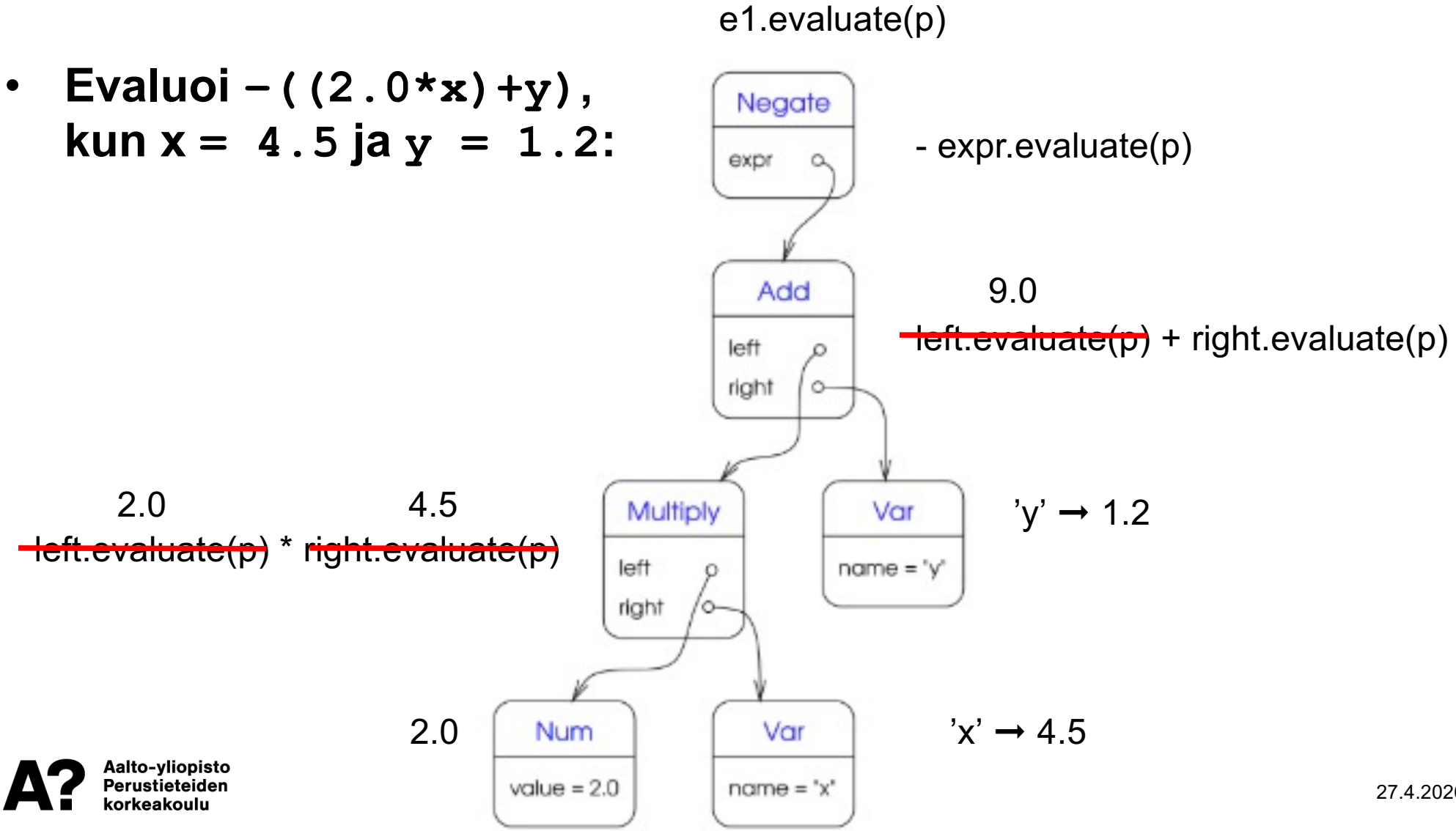
# Lausekkeet – Evaluointiesimerkki

- **Evaluoi `-((2.0*x)+y)`, kun `x = 4.5` ja `y = 1.2`:**



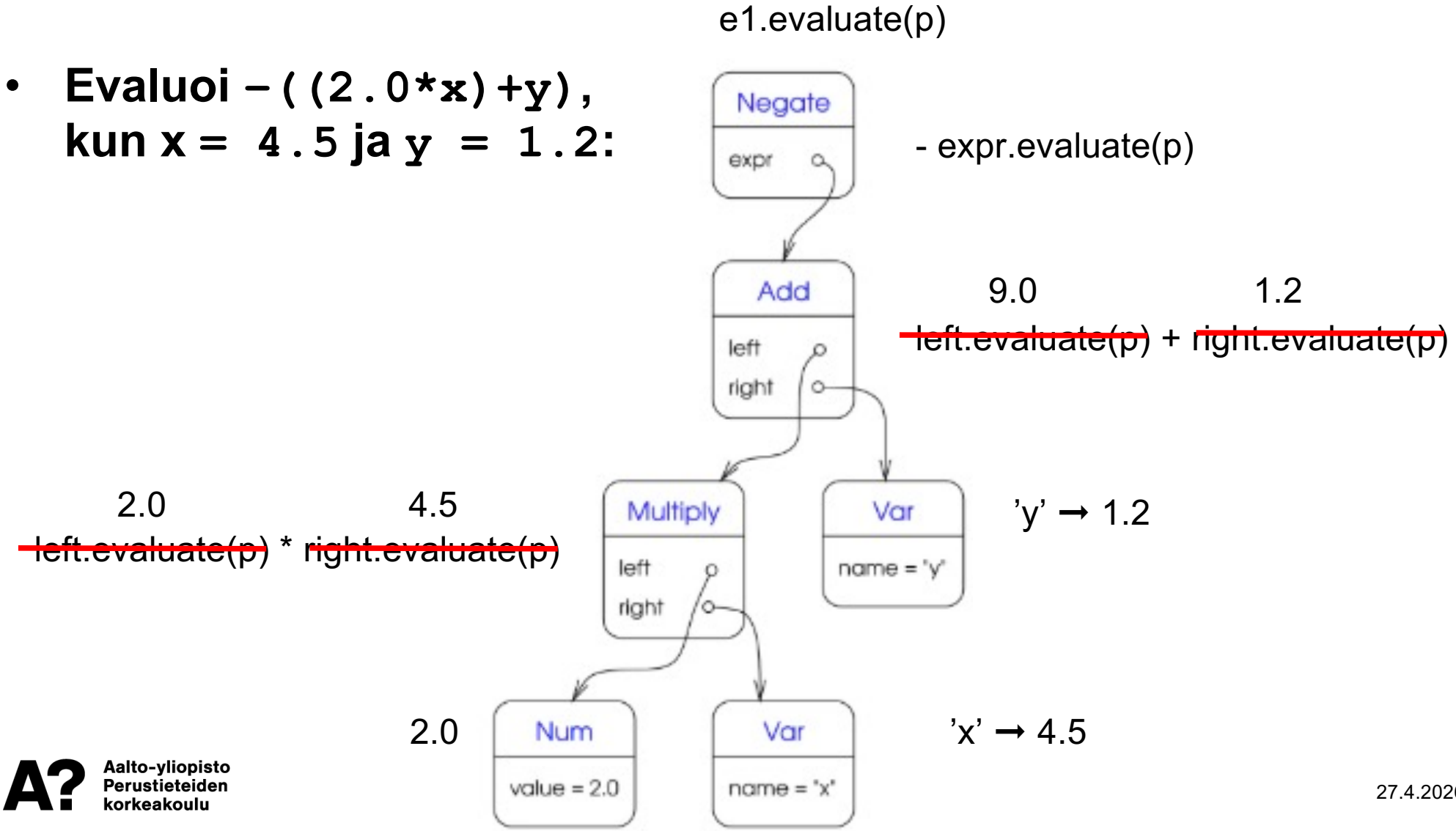
# Lausekkeet – Evaluointiesimerkki

- **Evaluoi `-((2.0*x)+y)`, kun `x = 4.5` ja `y = 1.2`:**



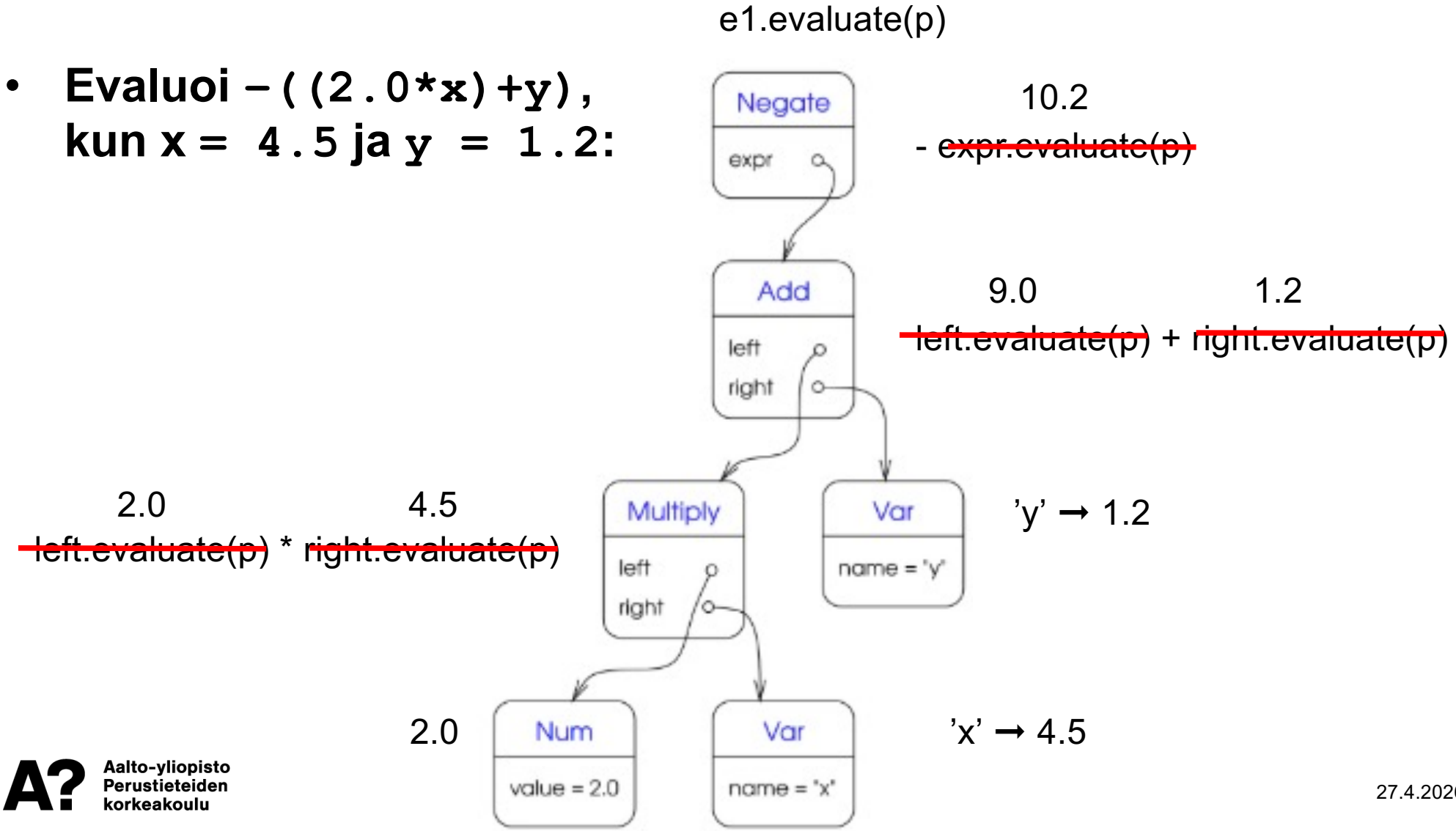
# Lausekkeet – Evaluointiesimerkki

- **Evaluoi `-((2.0*x)+y)`, kun `x = 4.5` ja `y = 1.2`:**



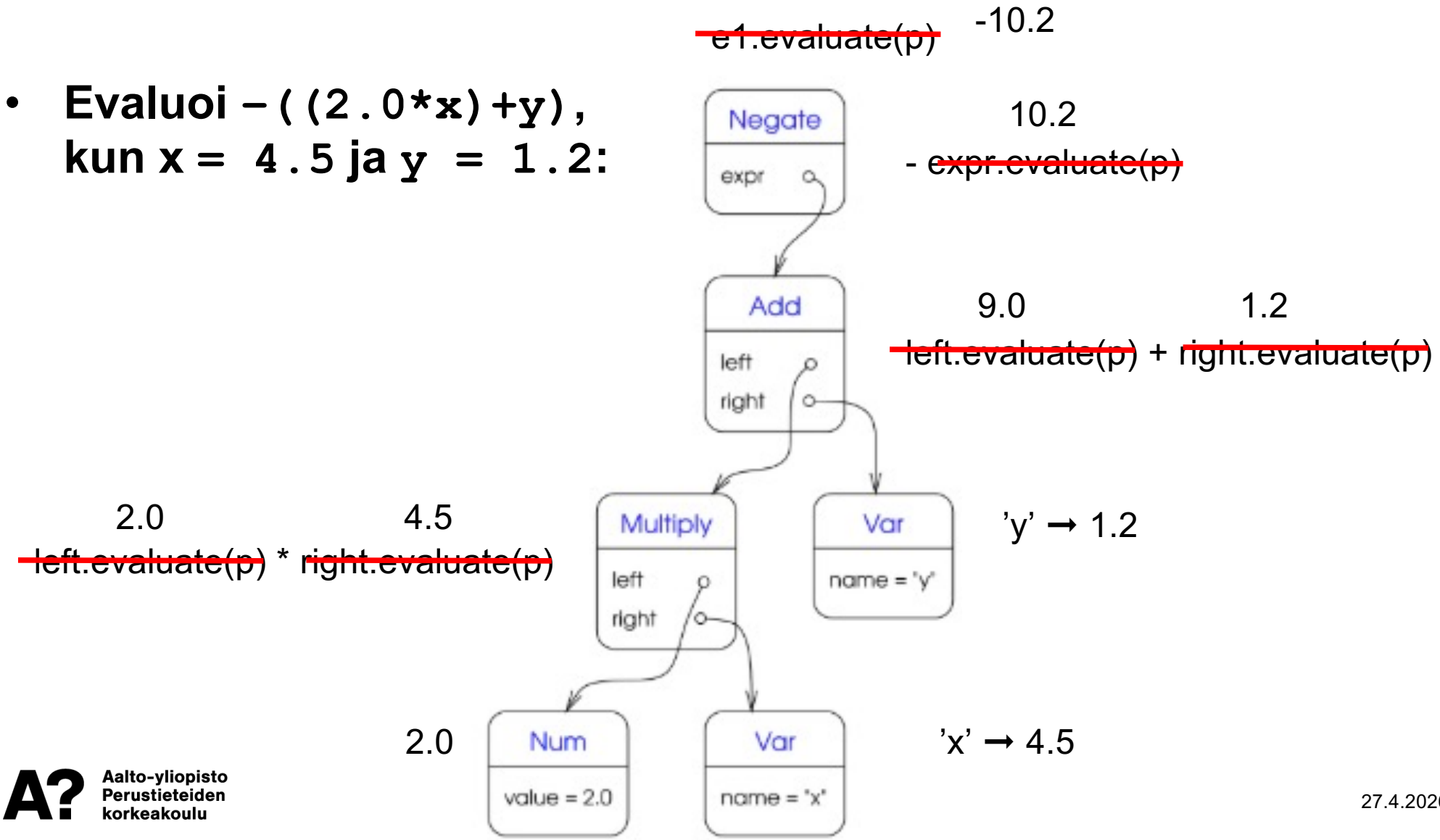
# Lausekkeet – Evaluointiesimerkki

- **Evaluoi `-((2.0*x)+y)`, kun `x = 4.5` ja `y = 1.2`:**



# Lausekkeet – Evaluointiesimerkki

- **Evaluoi `-((2.0*x)+y)`, kun `x = 4.5` ja `y = 1.2`:**



# Rekursiivinen ongelmanratkaisu

## (recursive problem solving)

Aalto-yliopisto
Perustieteiden
korkeakoulu

# Rekursiivinen ongelmanratkaisu

- **Jotkin ongelmatyypit luonnostaan johtavat rekursiiviseen lähestymistapaan, kun mietimme niiden ratkaisua**
  - Tyypin “järjestä johdonmukaisesti x ‘palasta’ y ‘paikkaan/askeleeseen’ tavoitteen saavuttamiseksi”

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 5 | 3 | | | 7 | | | | |
| 6 | | | 1 | 9 | 5 | | | |
| | 9 | 8 | | | | | 6 | |
| 8 | | | | 6 | | | | 3 |
| 4 | | | 8 | | 3 | | | 1 |
| 7 | | | | 2 | | | | 6 |
| | 6 | | | | | 2 | 8 | |
| | | | 4 | 1 | 9 | | | 5 |
| | | | | 8 | | | 7 | 9 |

*Given as input a set $W = \{w_1, w_2, \ldots, w_n\}$ of $n$ integers (the **steps**) and an integer $t$ (the **target**), is there a subset $S \subseteq W$ with $\sum_{z \in S} z = t$?*

Osajoukkojen summa –ongelma,
Katso kurssimateriaali

**May**

| Wk | Mon | Tue | Wed | Thu | Fri | Sat | Sun |
|---|---|---|---|---|---|---|---|
| | 25 | 26 | 27 | 28 | 29 | 30 | 1 |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 2 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| 3 | 16 | 17 | 18 | 19 | 20 | 21 | 22 |
| 4 | 23 | 24 | 25 | 26 | 27 | 28 | 29 |
| 5 | 30 | 31 | 1 | 2 | 3 | 4 | 5 |

Jan: iltapäivät parittomina viikkoina
Saana: kaikki paitsi ei 25. ja 26.5
Timo: vain ma-to
Eeva:…

  - Jne. matemaattiset ongelmat ja pelit



https://a1120.cs.aalto.fi/current/notes/round-recursion--solving.html#the-subset-sum-problem

# Rekursiivinen ongelmanratkaisu

"järjestä johdonmukaisesti x 'palasta' y 'paikkaan/askeleeseen' tavoitteen saavuttamiseksi"
("consistently arrange x 'pieces' in 'y' locations/steps' to reach a target")

- **Johdonmukaisesti/yhdenmukaisesti/säännönmukaisesti (consistently) tarkoittaa noudattaen jotain sääntöjä tai rajoituksia**
  - "John ei voi istua Sallyn vieressä",
  - "Sama ihminen ei voi istua kahdessa paikassa", ja
  - "Jokainen istuin voidaan käyttää vain kerran" jne
- **Askeleet (steps) tarkoittaa, että voimme parantaa mahdollista ratkaisua yksi pala kerrallaan (kunhan se pysyy johdonmukaisena)**
  - "valitse yksi ruutu Sudokusta ja laita numero siihen"
- **Kohde (target) tarkoittaa että tunnistamme täydellisen ratkaisun**

# Rekursiivinen ongelmanratkaisu

"järjestä johdonmukaisesti x 'palasta' y 'paikkaan/askeleeseen' tavoitteen saavuttamiseksi"
("consistently arrange x 'pieces' in 'y' locations/steps' to reach a target")

- **Voimme ratkaista tällaiset ongelmat etsimällä ratkaisua**
- **Rekursio on luonnollinen tapa toteuttaa kattava haku (exhaustive search)**
- **Se käy läpi jokaisen mahdollisen konfiguraation yksi askel kerrallaan, kunnes oikea vastaus löytyy (tai jokainen mahdollinen kombinaatio on käyty läpi, eli ratkaisua ei ole olemassa)**
- **Huom. Tällaiset ongelmat voivat olla todella vaikeita ratkaista yleisessä tapauksessa, jopa $\mathcal{O}(x^y)$**

# Peruuttava haku (backtracking search)

**Mahdolliset ratkaisut rakennetaan rekursiivisesti ja peruuttavaa hakua tehdään, jos huomataan, että ratkaisu ei tule toimimaan**

**Yleinen idea:**

- **Jokaisella askeleella meillä on konfiguraatio (configuration)**
- **Onko nykyinen konfiguraatio johdonmukainen?**
  - Jos se on myös täydellinen (complete), olemme saavuttaneet päämäärän ja löytäneet ratkaisun
  - Jos se ei ole täydellinen (mutta on johdonmukainen), päivitetään konfiguraatiota yhden askeleen verran ja jatketaan eteenpäin
- **Jos konfiguraatio ei ole johdonmukainen, sitten peruutetaan edelliseen johdonmukaiseen konfiguraatioon ja yritetään jotain muuta**
  - (jos sellaista ei ole, ongelmalle ei ole ratkaisua)

# Peruuttava haku

- **Algoritmi pseudokoodina**

```
bsolve(config):
  if config is consistent then
    if config is complete then
      return Success(config)
    else:
      for every possible action given config do
        s = bsolve(config updated by action)
        if s is Success then return s
      else:
        return Fail
end bsolve
```

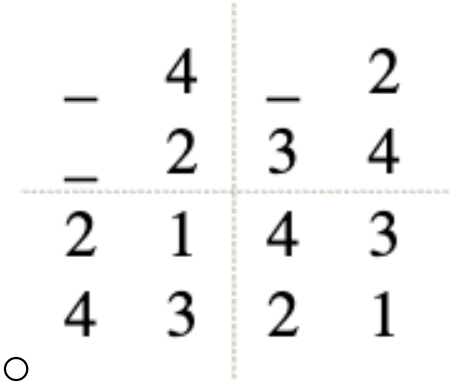
- **Esim Sudokussa:**
  - `config` on n*n Sudoku-ruudukko
  - Pelilauta on *johdonmukainen* jos se noudattaa Sudokun sääntöjä
  - Pelilauta on *täydellinen* jos kaikissa ruuduissa on numero
  - Toiminto on valita tyhjä ruutu ja kokeilla siihen joku numero *johonkin* kriteeriin pohjaten

# Peruuttava haku

- **Algoritmi pseudokoodina**

```
bsolve(config):
  if config is consistent then
    if config is complete then
      return Success(config)
    else:
      for every possible action given config do
        s = bsolve(config updated by action)
        if s is Success then return s
    else:
      return Fail
end bsolve
```

- **4*4-Sudoku-esimerkki**




Tässä esimerkissä:

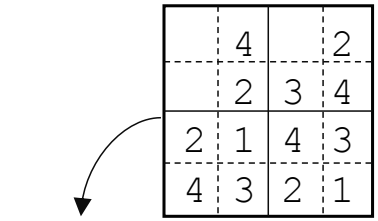
- toiminto on, että otetaan ensimmäinen tyhjä ruutu ylävasemmasta kulmasta lukien, ja
- kokeillaan numeroita järjestyksessä ruutuihin

# Peruuttava haku – 4*4-Sudoku

| | | | |
|---|---|---|---|
| | 4 | | 2 |
| | 2 | 3 | 4 |
| 2 | 1 | 4 | 3 |
| 4 | 3 | 2 | 1 |

Tässä esimerkissä: toiminto on, että otetaan ensimmäinen tyhjä ruutu ylävasemmasta kulmasta lukien, ja kokeillaan numeroita järjestyksessä ruutuihin

# Peruuttava haku – 4*4-Sudoku

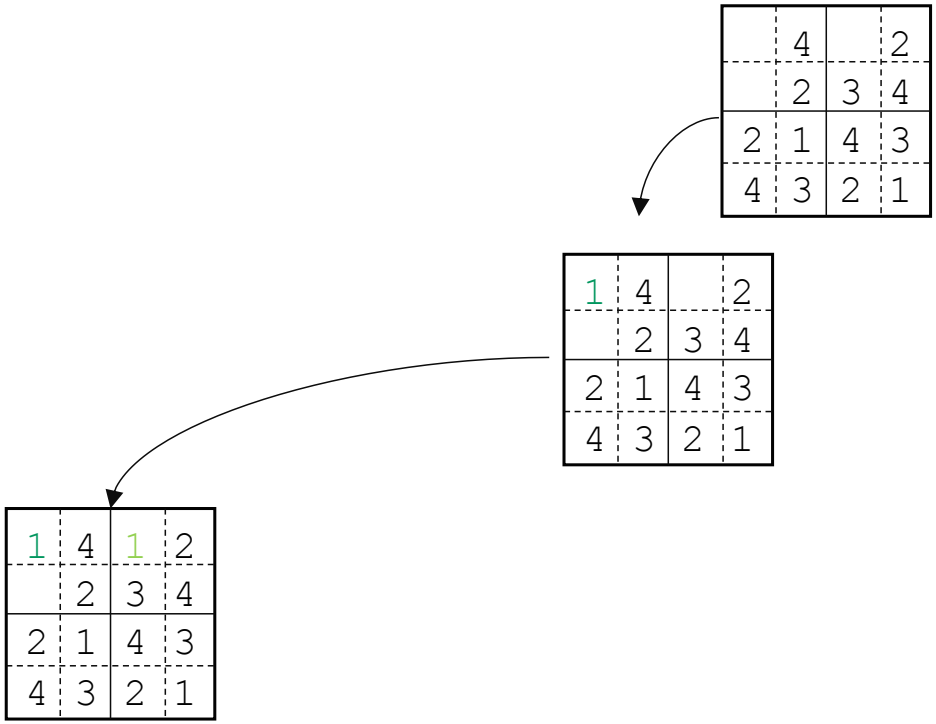


| | | | |
|---|---|---|---|
| | 4 | | 2 |
| | 2 | 3 | 4 |
| 2 | 1 | 4 | 3 |
| 4 | 3 | 2 | 1 |

| | | | |
|---|---|---|---|
| 1 | 4 | | 2 |
| | 2 | 3 | 4 |
| 2 | 1 | 4 | 3 |
| 4 | 3 | 2 | 1 |

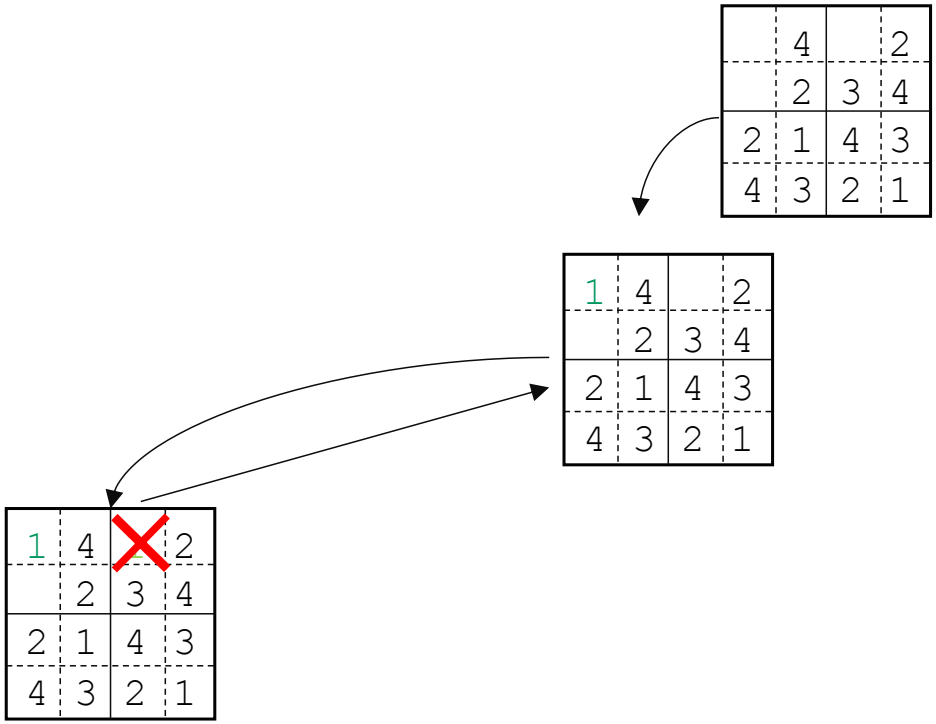
Tässä esimerkissä: toiminto on, että otetaan ensimmäinen tyhjä ruutu ylävasemmasta kulmasta lukien, ja kokeillaan numeroita järjestyksessä ruutuihin

# Peruuttava haku – 4*4-Sudoku



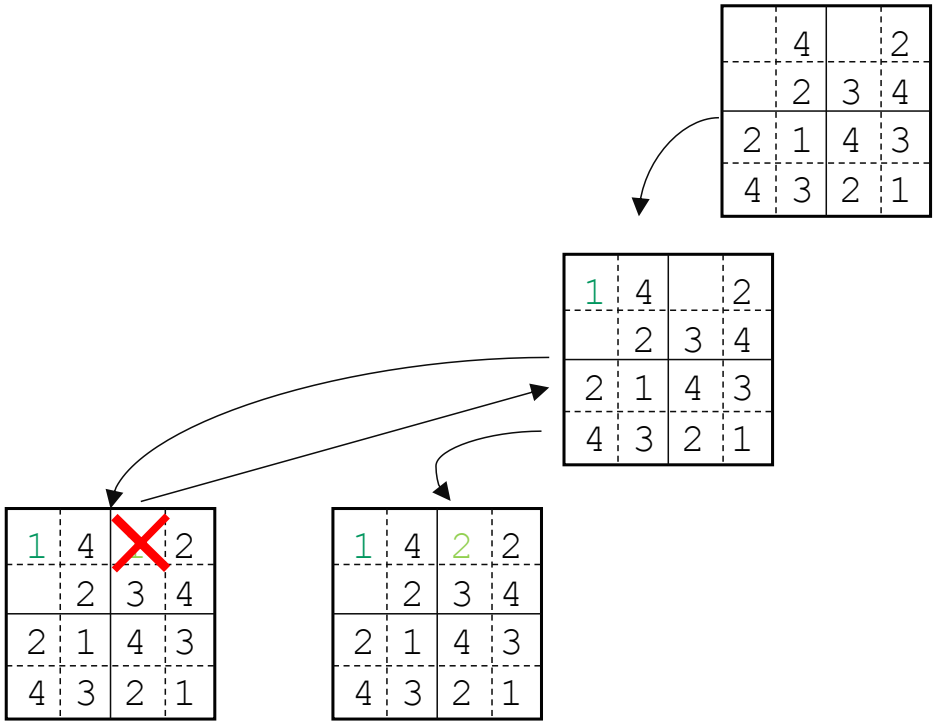

Tässä esimerkissä: toiminto on, että otetaan ensimmäinen tyhjä ruutu ylävasemmasta kulmasta lukien, ja kokeillaan numeroita järjestyksessä ruutuihin

# Peruuttava haku – 4*4-Sudoku



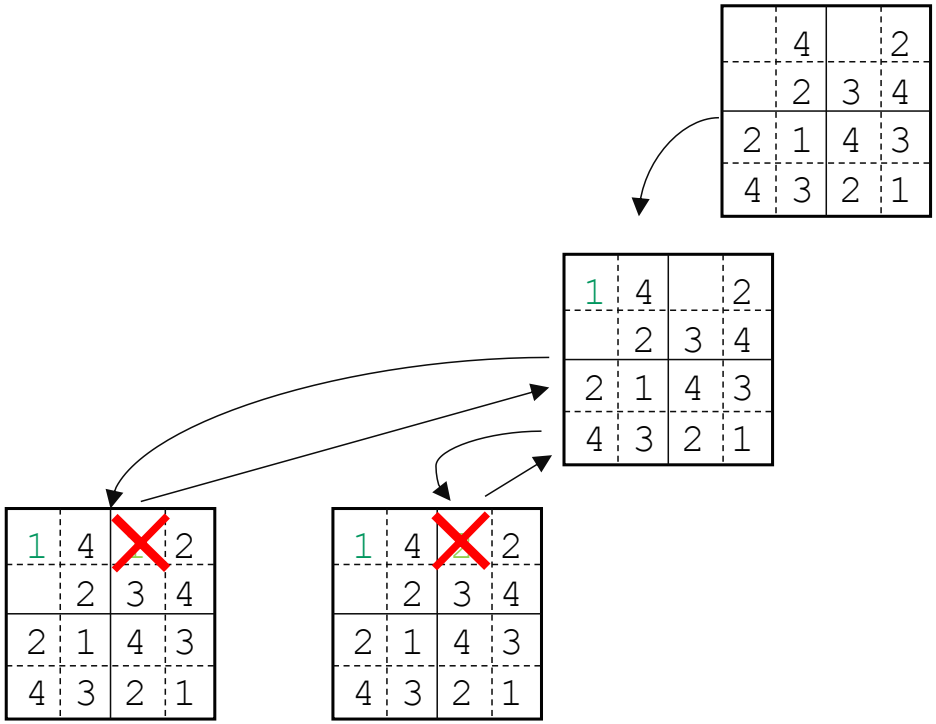

Tässä esimerkissä: toiminto on, että otetaan ensimmäinen tyhjä ruutu ylävasemmasta kulmasta lukien, ja kokeillaan numeroita järjestyksessä ruutuihin

# Peruuttava haku – 4*4-Sudoku



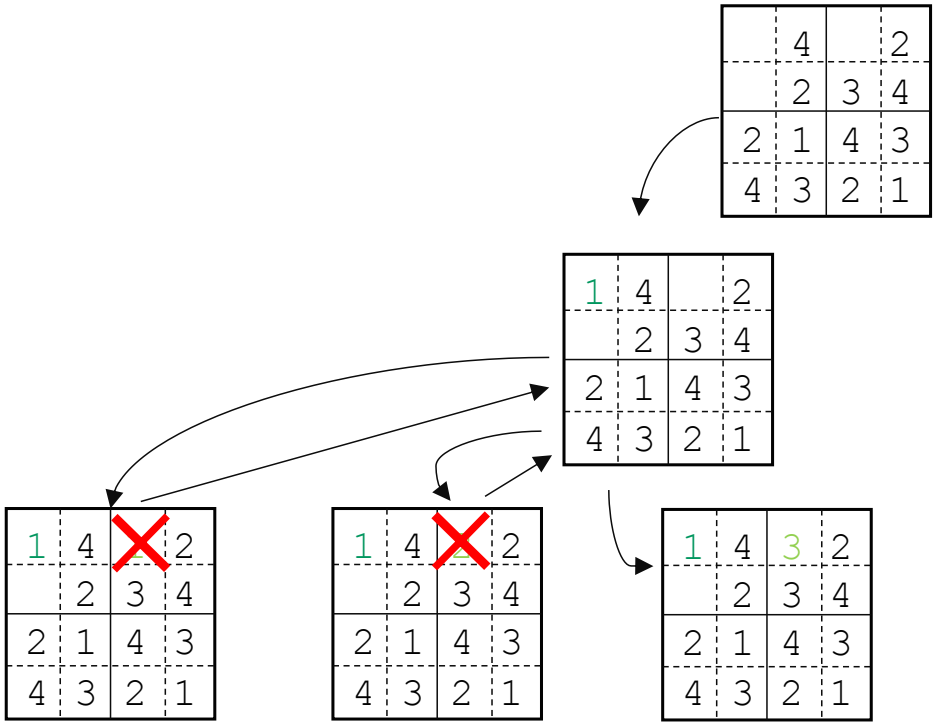

Tässä esimerkissä: toiminto on, että otetaan ensimmäinen tyhjä ruutu ylävasemmasta kulmasta lukien, ja kokeillaan numeroita järjestyksessä ruutuihin

# Peruuttava haku – 4*4-Sudoku



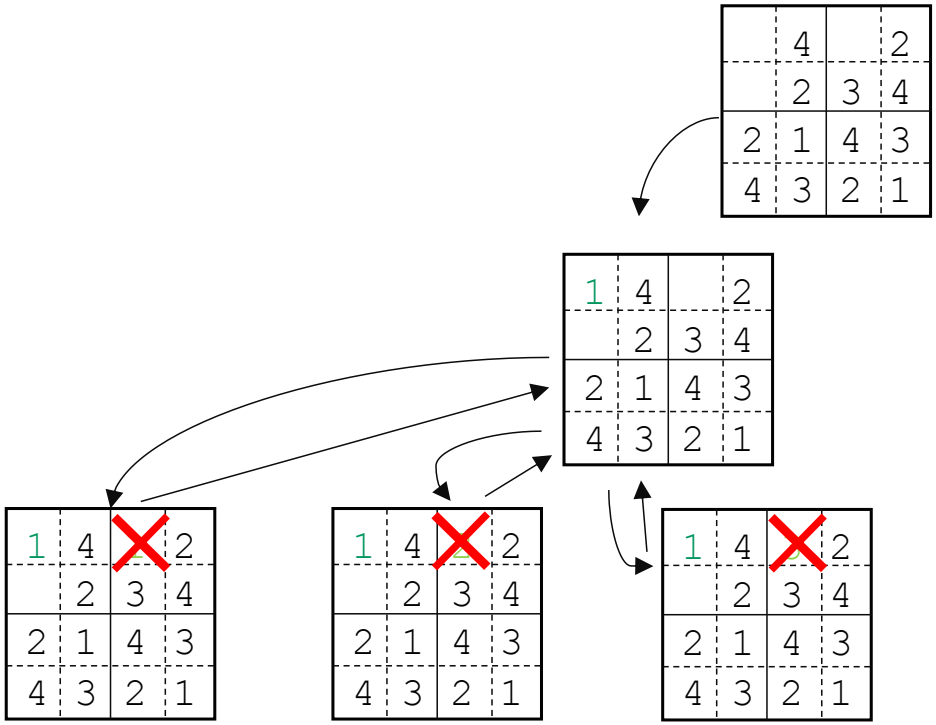

Tässä esimerkissä: toiminto on, että otetaan ensimmäinen tyhjä ruutu ylävasemmasta kulmasta lukien, ja kokeillaan numeroita järjestyksessä ruutuihin

# Peruuttava haku – 4*4-Sudoku



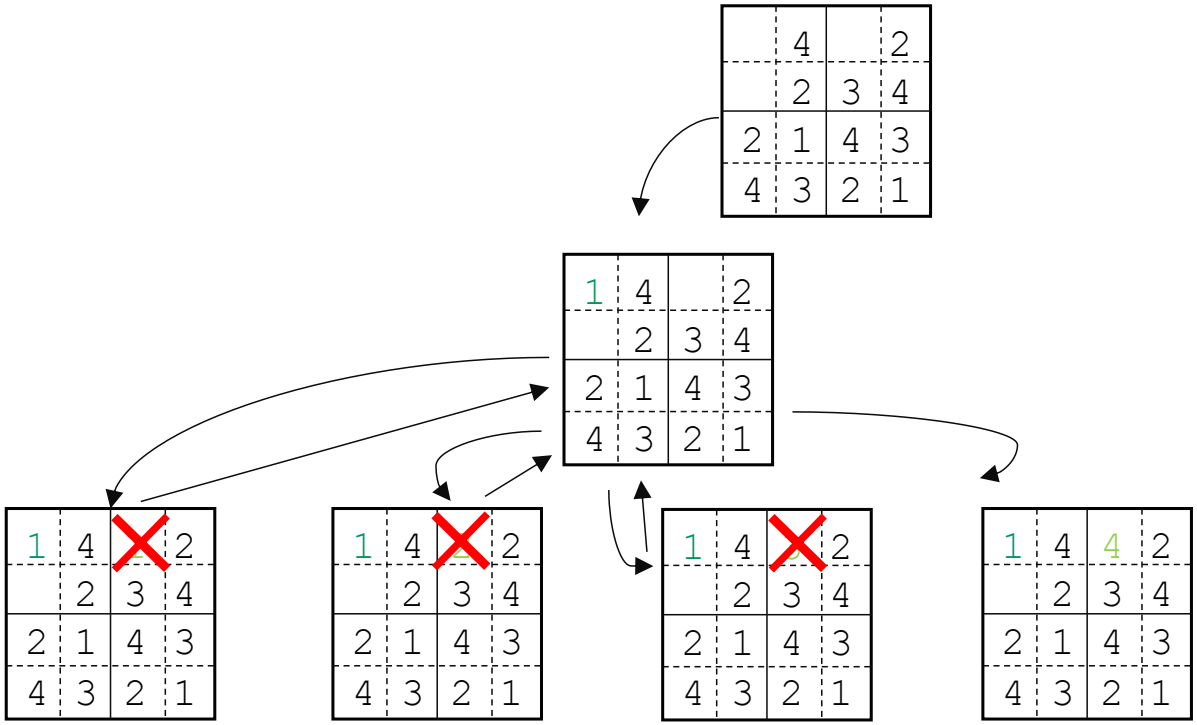

Tässä esimerkissä: toiminto on, että otetaan ensimmäinen tyhjä ruutu ylävasemmasta kulmasta lukien, ja kokeillaan numeroita järjestyksessä ruutuihin

# Peruuttava haku – 4*4-Sudoku



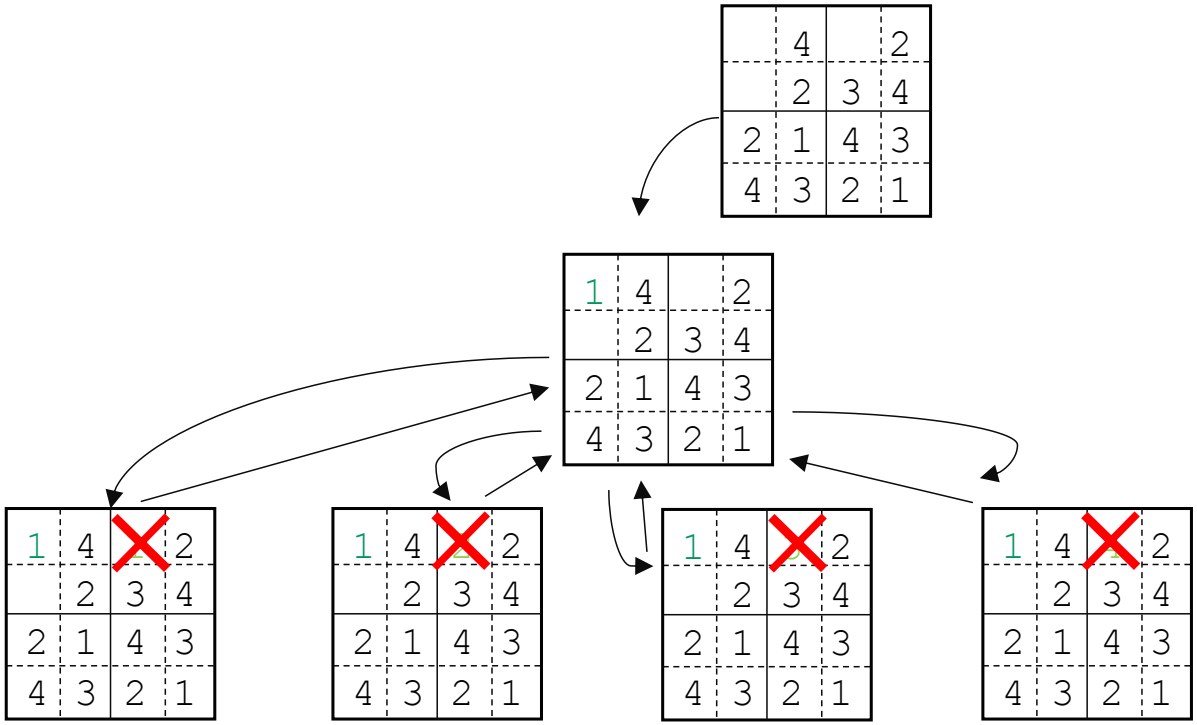

Tässä esimerkissä: toiminto on, että otetaan ensimmäinen tyhjä ruutu ylävasemmasta kulmasta lukien, ja kokeillaan numeroita järjestyksessä ruutuihin

# Peruuttava haku – 4*4-Sudoku



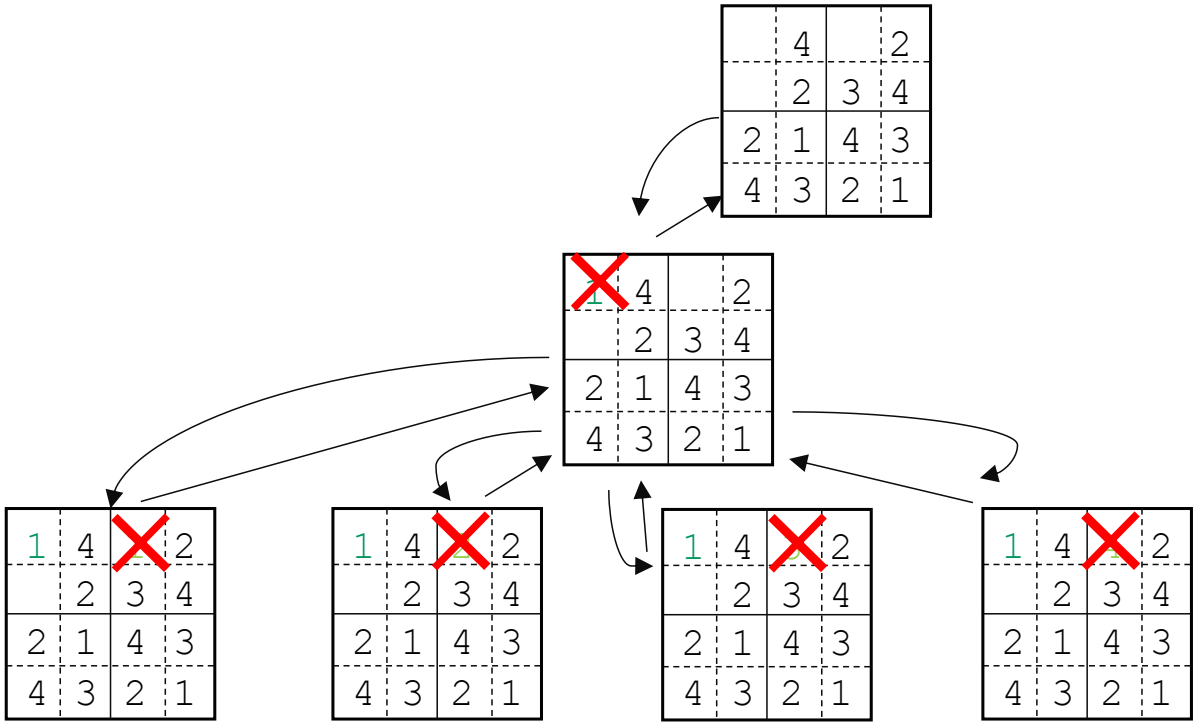

Tässä esimerkissä: toiminto on, että otetaan ensimmäinen tyhjä ruutu ylävasemmasta kulmasta lukien, ja kokeillaan numeroita järjestyksessä ruutuihin

# Peruuttava haku – 4*4-Sudoku



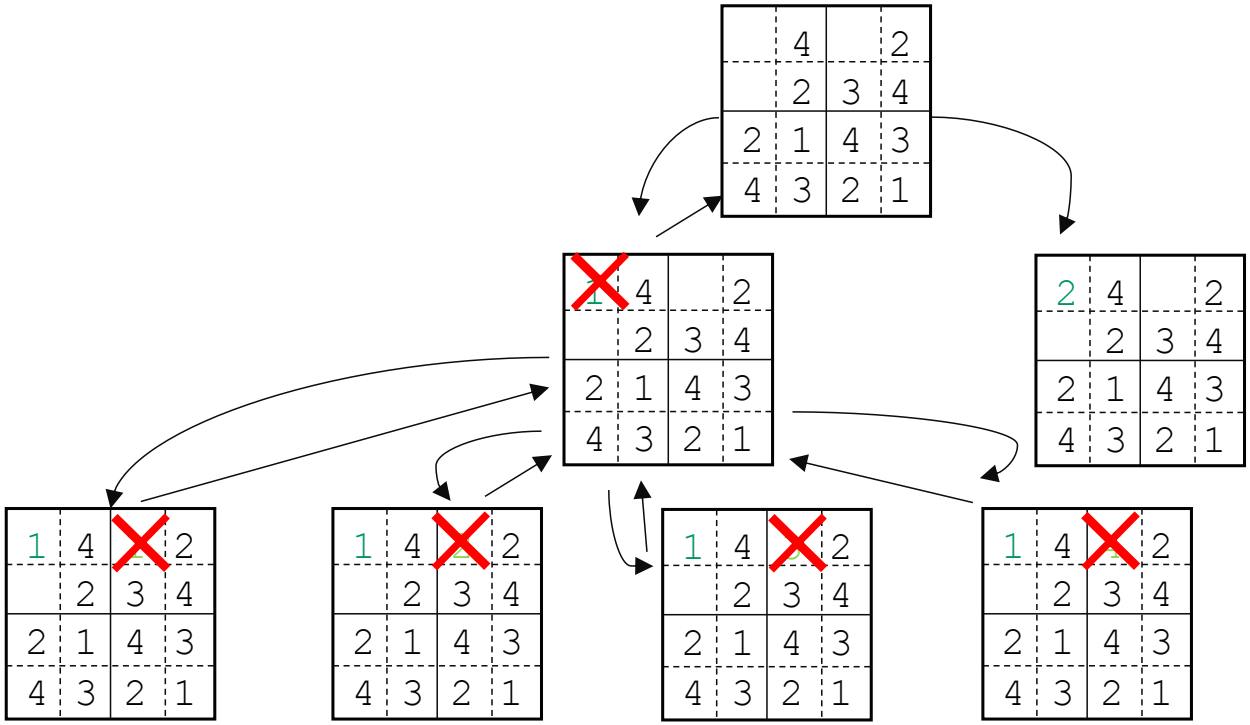

Tässä esimerkissä: toiminto on, että otetaan ensimmäinen tyhjä ruutu ylävasemmasta kulmasta lukien, ja kokeillaan numeroita järjestyksessä ruutuihin

# Peruuttava haku – 4*4-Sudoku



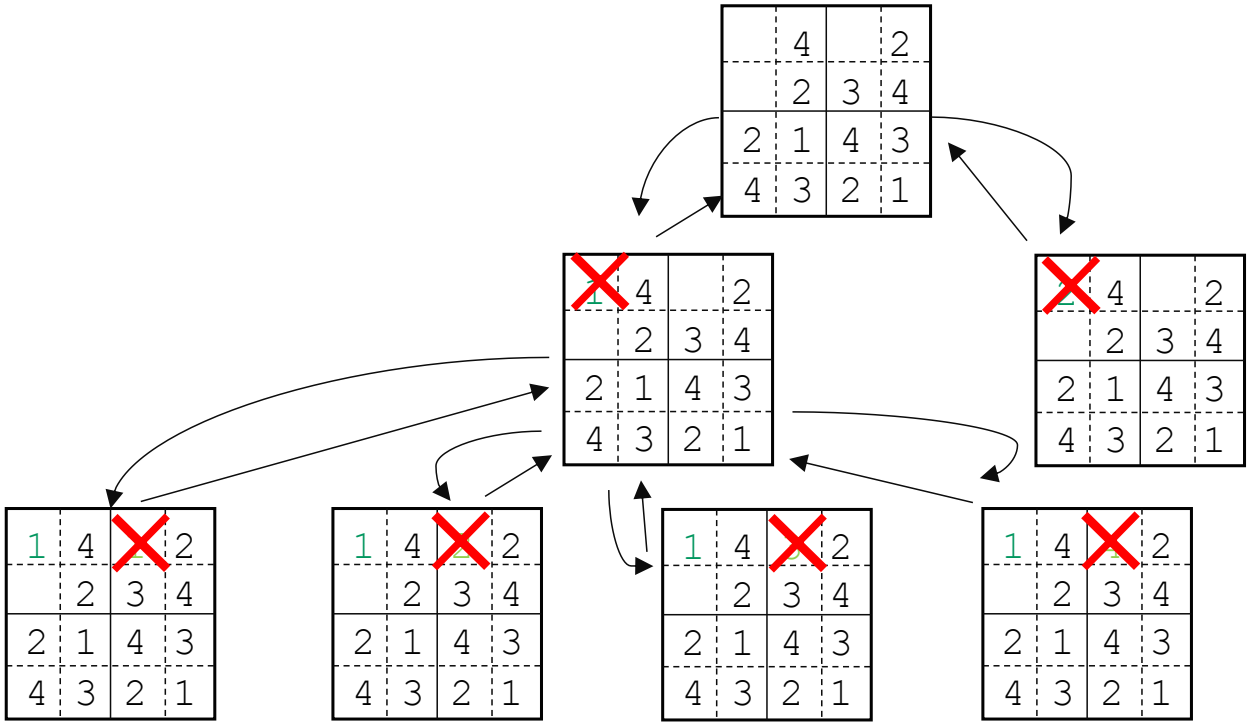

Tässä esimerkissä: toiminto on, että otetaan ensimmäinen tyhjä ruutu ylävasemmasta kulmasta lukien, ja kokeillaan numeroita järjestyksessä ruutuihin

# Peruuttava haku – 4*4-Sudoku



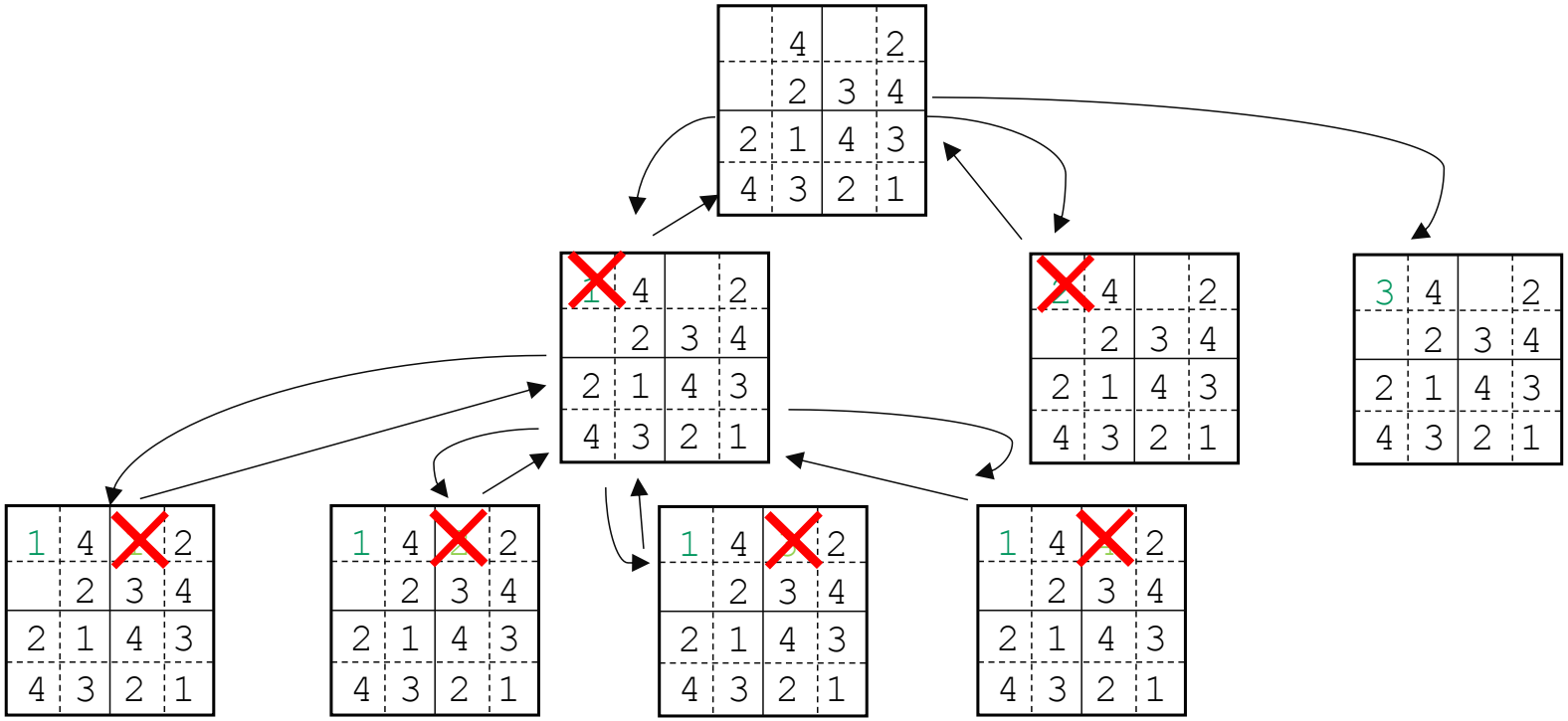

Tässä esimerkissä: toiminto on, että otetaan ensimmäinen tyhjä ruutu ylävasemmasta kulmasta lukien, ja kokeillaan numeroita järjestyksessä ruutuihin

# Peruuttava haku – 4*4-Sudoku



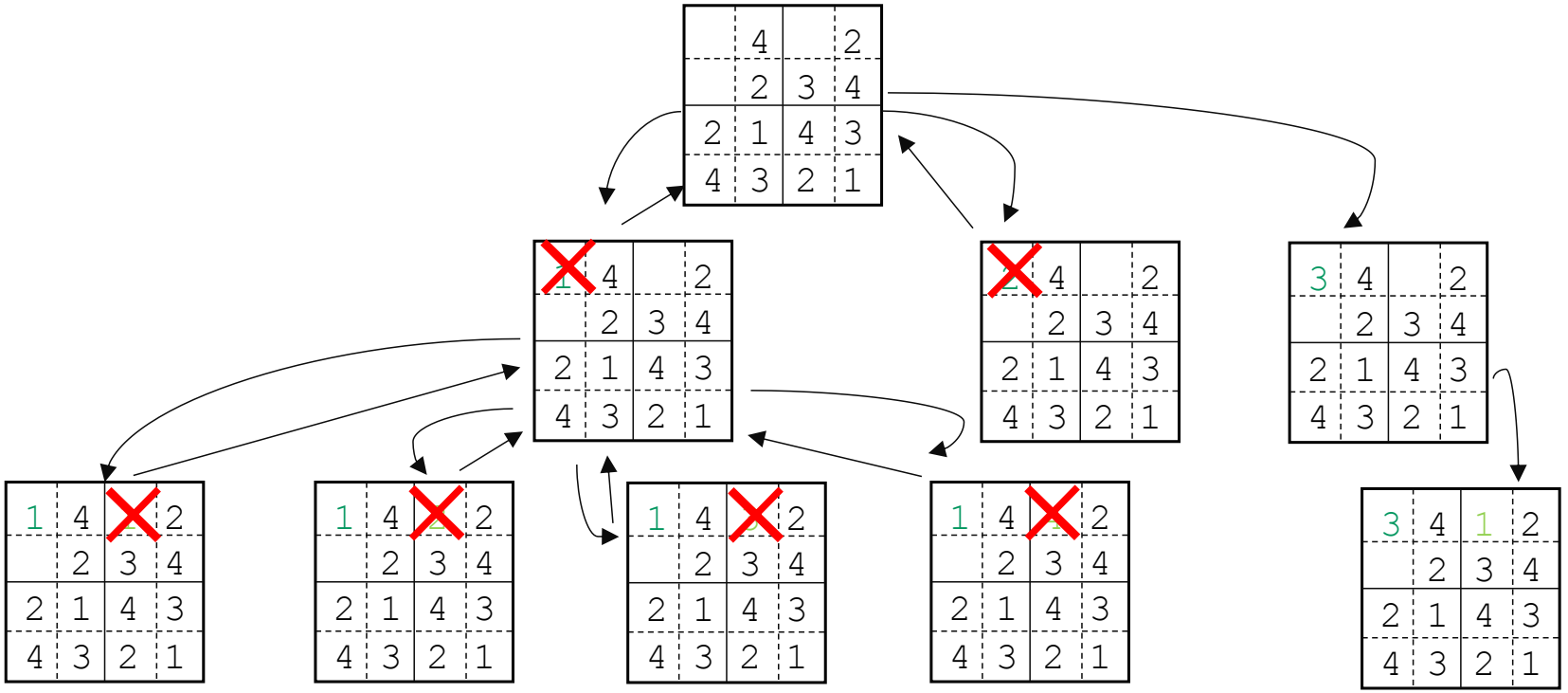

Tässä esimerkissä: toiminto on, että otetaan ensimmäinen tyhjä ruutu ylävasemmasta kulmasta lukien, ja kokeillaan numeroita järjestyksessä ruutuihin

# Peruuttava haku – 4*4-Sudoku



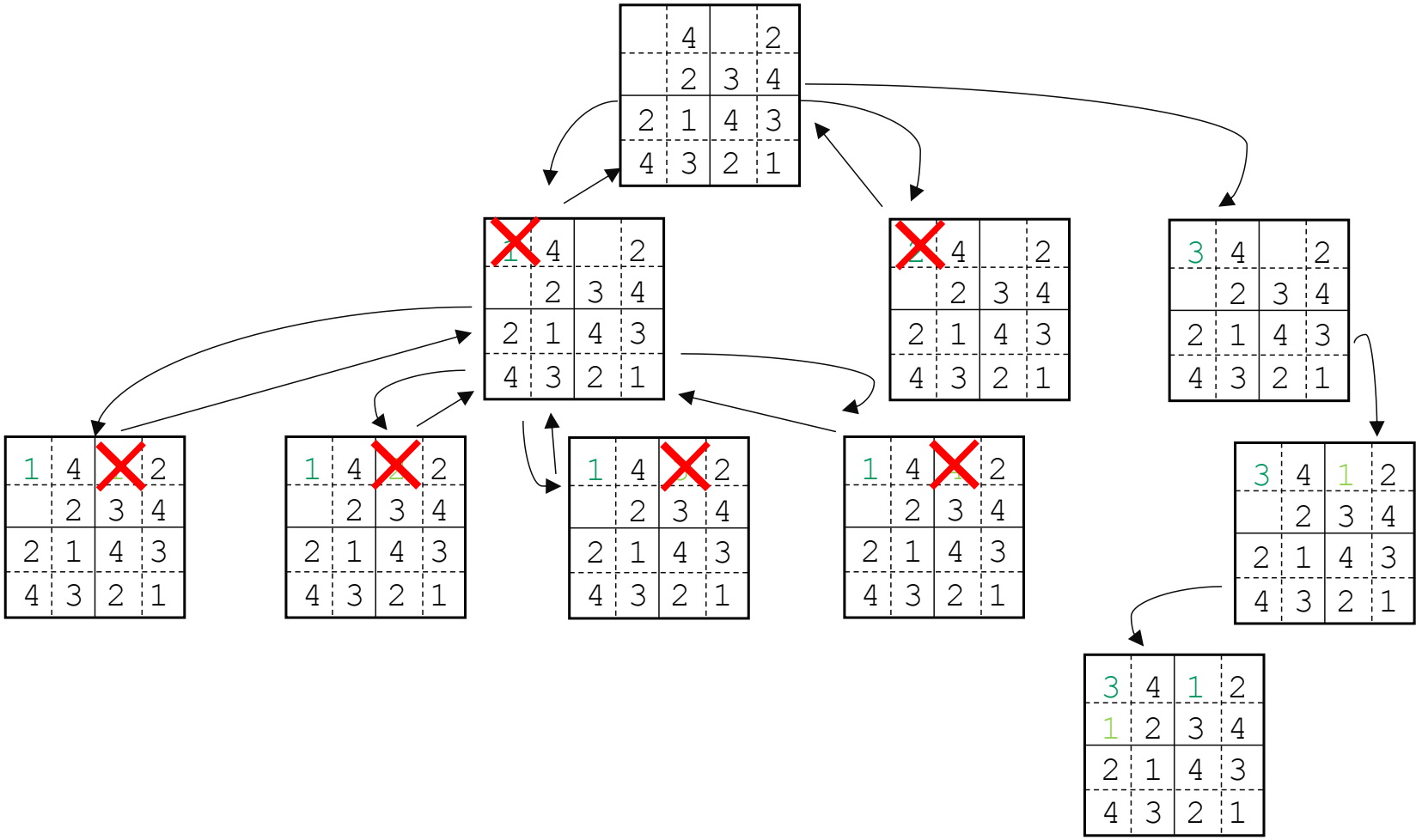

Tässä esimerkissä: toiminto on, että otetaan ensimmäinen tyhjä ruutu ylävasemmasta kulmasta lukien, ja kokeillaan numeroita järjestyksessä ruutuihin

# Peruuttava haku – 4*4-Sudoku



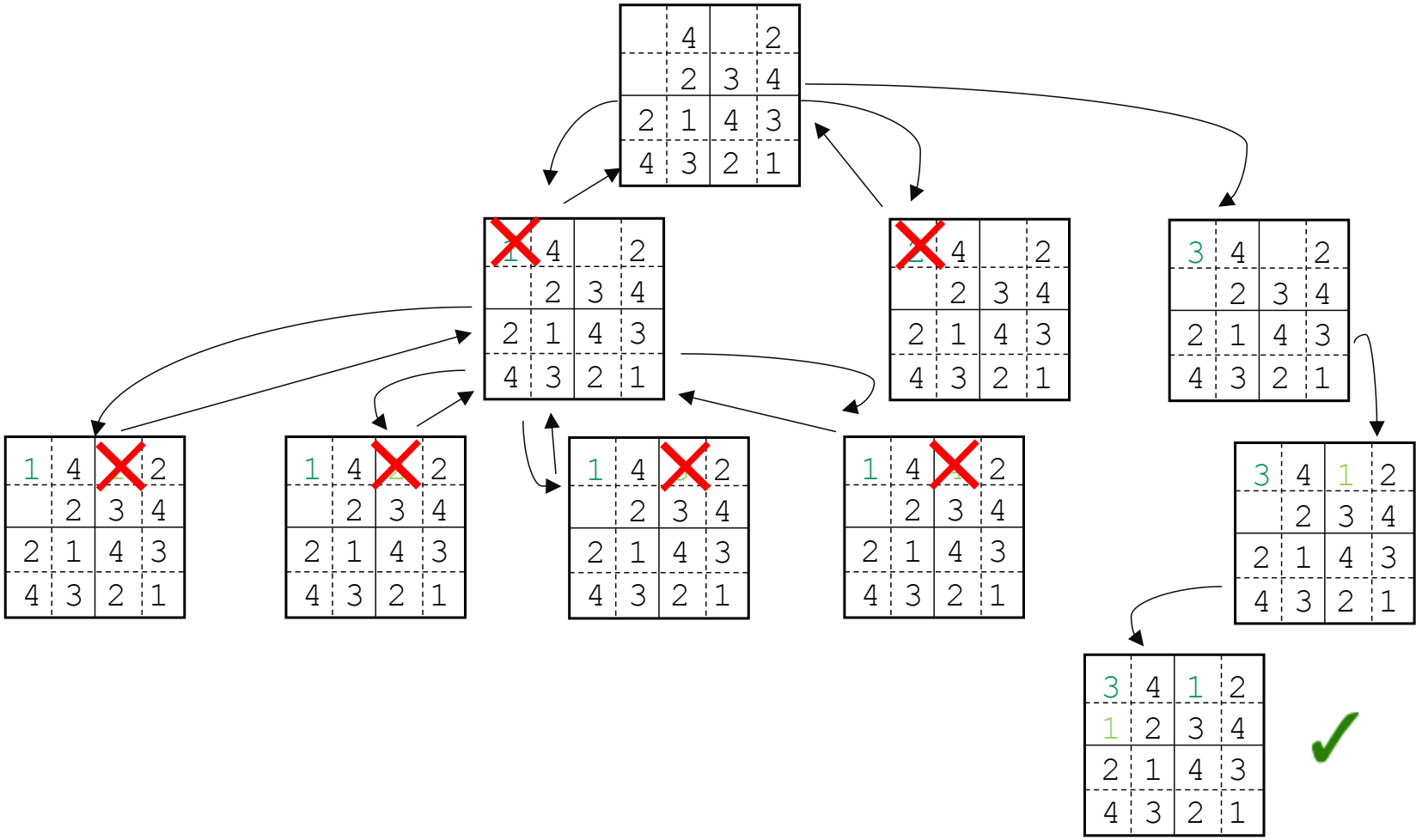
Tässä esimerkissä: toiminto on, että otetaan ensimmäinen tyhjä ruutu ylävasemmasta kulmasta lukien, ja kokeillaan numeroita järjestyksessä ruutuihin

# Peruuttava haku – 4*4-Sudoku



Tässä esimerkissä: toiminto on, että otetaan ensimmäinen tyhjä ruutu ylävasemmasta kulmasta lukien, ja kokeillaan numeroita järjestyksessä ruutuihin

# Peruuttava haku – 4*4-Sudoku



Tässä esimerkissä: toiminto on, että otetaan ensimmäinen tyhjä ruutu ylävasemmasta kulmasta lukien, ja kokeillaan numeroita järjestyksessä ruutuihin

# Kierroksen 8 tehtävät – ei harjoituksia to&pe

1. **Linkitetty lista**
2. **Lausekkeita**
3. **Aikataulutus ryhmälle**
4. **Sudoku**
5. **Haaste: subset sum dynaamisella ohjelmoinnilla**

- **Joissain tehtävissä on rajoitteita: ei saa käyttää var, Array, etc**
  - Tarkkana tehtävänannon kanssa
- **Muistakaa häntärekursio**
- **Katsokaa läpi ennen tehtävän tekemistäannetut metodit**
- **Jos jokin osatehtävä on liian vaikea, voi osasta saada pisteitä**
- **Scheduling ja Sudoku: harkitse peruuttavan haun lähestymistapaa**
- **Sudokussa on mukana käyttöliittymä, jolla voi pelata**