Halsketting

Operator overloading bij zelfgedefinieerde types

Als Python de expressie

```
o1 + o2
```

moet evalueren, dan wordt die expressie omgezet naar

```
type(o1).__add__(o1, o2)
```

Op die manier kan je voor zelfgedefinieerde types vastleggen hoe de `+`-operator moet uitgevoerd worden. Dit wordt *operator overloading* genoemd. Dit blijft echter niet beperkt tot de `+`-operator. Python vertaalt elke ingebouwde operator (wiskundige operatoren en vergelijkingsoperatoren) naar een methode waarvan de naam is vastgelegd door de ontwikkelaars van Python (de naam begint en eindigt telkens met twee underscores). Hier is een overzicht van enkele van deze *magische* methoden:

operator	methode
`+`	`__add__`
`-`	`__sub__`
`*`	`__mul__`
`/`	`__truediv__`
`//`	`__floordiv__`
`**`	`__pow__`

Bij operator overloading wordt de *magische* methode dus aangeroepen op het linker operand `o1`. Maar wat als de klasse van het linker operand `o1` de magische methode niet definieert voor objecten van het type `o2`? In dat geval wordt er een *exception* opgeworpen, en probeert Python vervolgens een andere *magische* methode (waarbij de naam van de *magische* methode wordt voorafgegaan door de letter `r`) aan te roepen op het rechter operand `o2`.

De optelling van hierboven wordt dus in tweede instantie omgezet naar

```
type(o2).__radd__(o2, o1)
```

Let hierbij op het feit dat de naam van de methode `__radd__` geworden is, in plaats van `__add__`, en dat de volgorde van de argumenten omgekeerd is. Dat laatste is vooral belangrijk voor bewerkingen die niet symmetrisch zijn.

Een verwijzing naar het huidige object teruggeven

Sommige objecten geven na een bewerking een verwijzing naar zichzelf terug. Stel voor dat we het spelletje Tic-Tac-Toe willen implementeren:

```
class TicTacToe:
  def __init__(self):
    self.rooster = [
      [ None, None, None ],
      [ None, None, None ],
      [ None, None, None ]
    ]
    self.speler = 'O'

  def zet(self, i, j):
    self.rooster[i][j] = self.speler
```

```
    self.speler = 'O' if self.speler == 'X' else 'X'
    return self
```

Dit laat ons toe om het spel als volgt te spelen:

```
>>> spel = TicTacToe().zet(1, 1).zet(0, 0).zet(0, 1).zet(1, 0)
>>> spel.rooster
[
  [ 'X', 'X', None ],
  [ 'O', 'O', None ],
  [ None, None, None ]
]
```

Het belangrijke hier is dus de `return self`.

Datacompressie

Specifieke info

Voor het decoderen van een bitstring b kun je de volgende procedure uitvoeren totdat de bitstring b de lege string is:

- vind de korste prefix p van de bitstring b die een symbool s voorstelt dat gedecodeerd kan worden
- voeg het symbool s toe aan de gedecodeerde boodschap
- verwijder de prefix p aan het begin van de bitstring b

Racetrack Playa

Specifieke info

Figuur 1 laat zien wat er gebeurt als een blok verschoven wordt.

Figuur 2 laat zien wat er gebeurt als een blok gekanteld wordt.

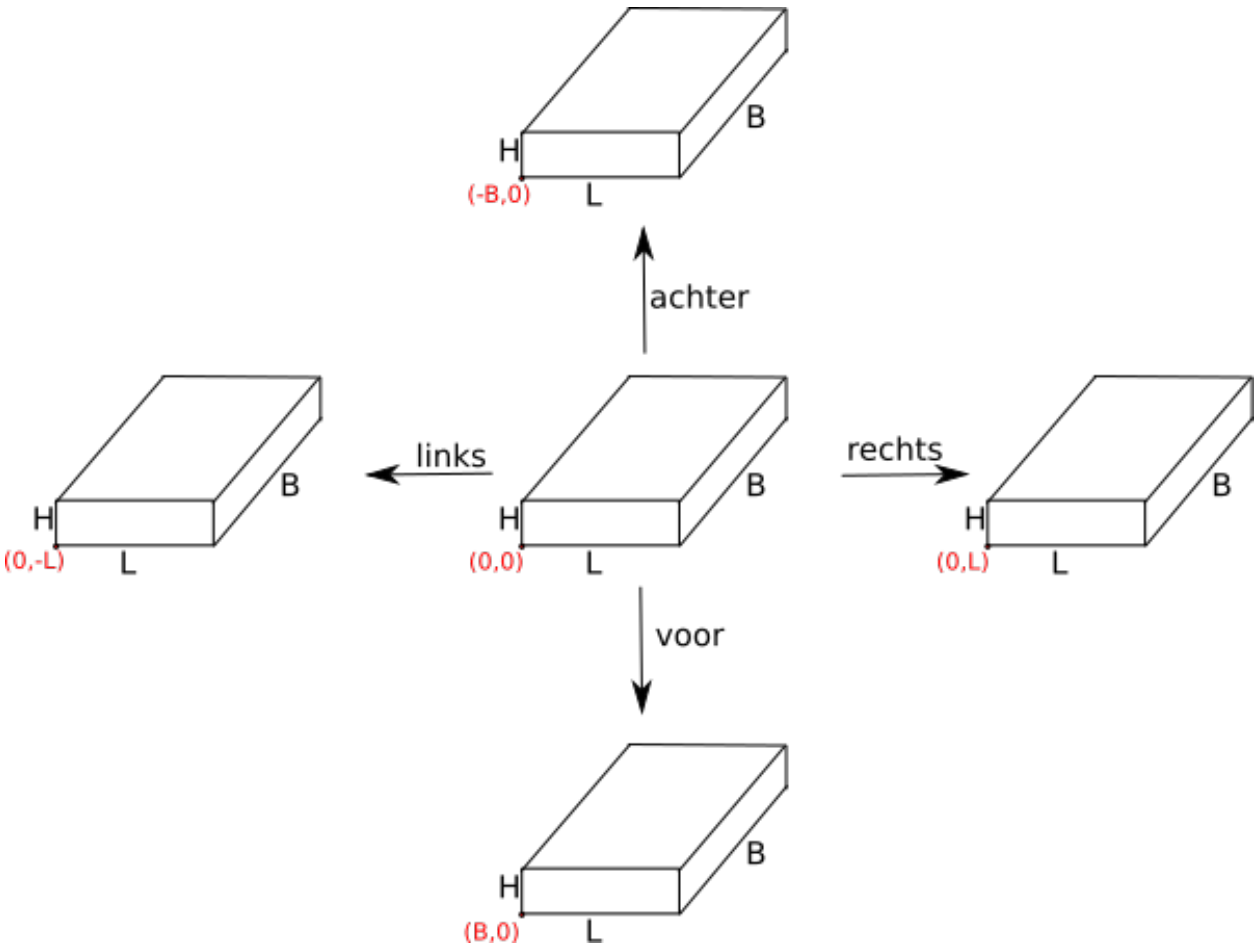

Figure 1: Schuiven

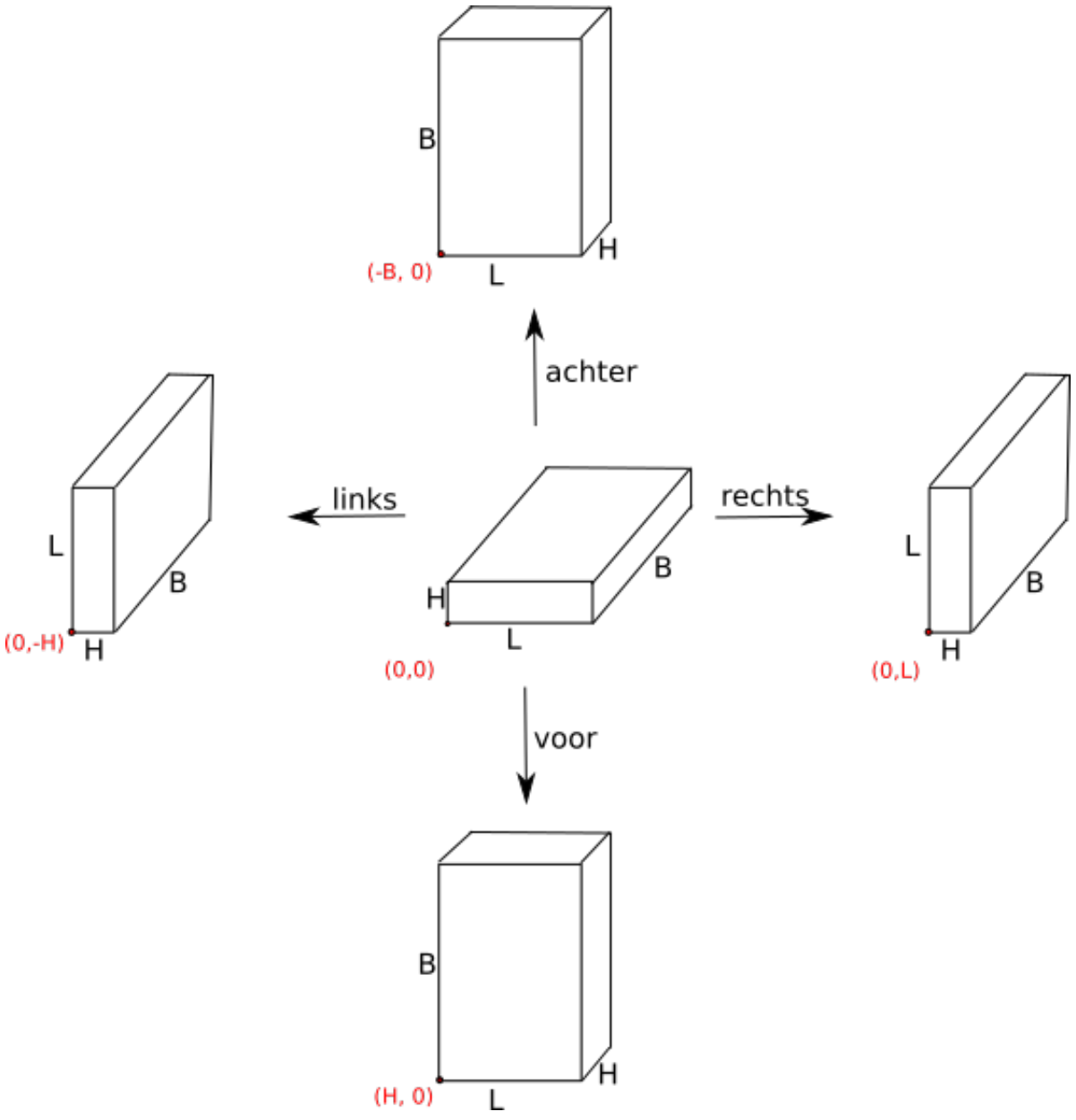

Figure 2: Kantelen