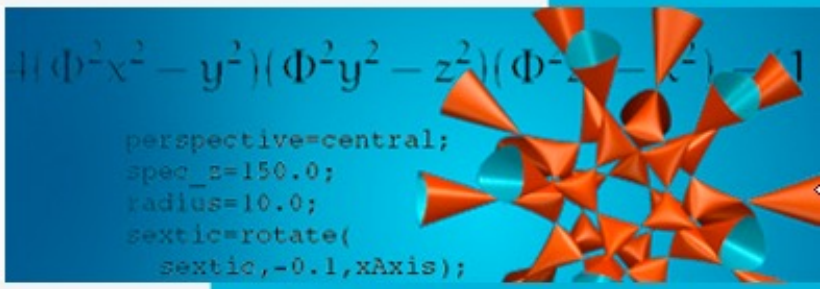
Philipps Universität Marburg

Declarative programming

Summer semester 2024

Prof. Christoph Bockisch, Steffen Dick
(Programming languages and tools)

Imke Gürtler, Daniel Hinkelmann, Aaron Schafberg, Stefan Störmer



[Script 4 - 5.2]

Programs

- Batch program
  - Executed independently
  - Main function
  - Input: Function arguments or input stream
  - Output: Function result or output current
- Interactive program
  - Input/output during execution of the evaluation

Input/output currents

- Input current
  - Endless string
  - Can be read character by character
- Output current
  - Endless string
  - Can be written character by character
- The output stream of one function can serve as the input function of another

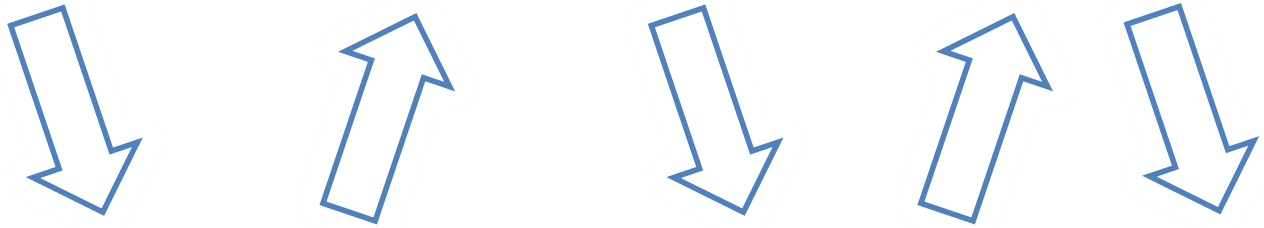
Input/output currents

- Example Unix command line tools
  - ls - writes list of files in the current directory to output stream
  - grep - copies lines corresponding to regular expression to output stream
  - sort - sorts the lines of an input stream

```
ls -1 | grep "05" | sort --reverse
```

Input/output currents

- ls -1 | grep "05" | sort --reverse



```
DP-01-Slides.pdf
DP-01-Slides.pptx
DP-02-Slides.pdf
DP-02-Slides.pptx
DP-03-Slides.pdf
DP-03-Slides.pptx
DP-04-Slides.pdf
DP-04-Slides.pptx
DP-05-Slides.pdf
DP-05-Slides.pptx
```

```
DP-05-Slides.pdf
DP-05-Slides.pptx
```

```
DP-05-Slides.pptx
DP-05-Slides.pdf
```

Batch programs

- Program letter requires no interaction
- Example for batch program
- Possible to start such programs outside of DrRacket
- Function arguments must be taken from command line arguments

Batch programs with Racket

```
(require racket/base)

(define args (current-command-line-arguments))

(if (= (vector-length args) 3)
    (display (letter (vector-ref args 0)
        (vector-ref args 1)
        (vector-ref args 2)))
    (error "Please pass exactly three parameters"))

String String -> String
; generates a scam mail for the victim fst last signed as signature-name
(check-range (string-length (letter "Tillman" "Rendel" "Klaus")) 50 300)

(define (letter fst lst signature-name) ...
```

Use functionality from Packet Base

Primitive constant

Write result to standard output current

Ignore the details for now.

Batch programs in Racket

- Saving the racket program (e.g. letter.rkt)
- Call from command line

```
$ racket letter.rkt Tillmann Rendel Klaus
Dear Mr./Mrs. Rendel,

After the last annual calculations of your GNB account
activity we have determined that you, Tillmann Rendel, are
eligible to receive a tax refund of $479.30.

Please submit the tax refund request (http://www...) and allow
us 2-6 days in order to process it.

With best regards,
Klaus
```

Batch programs in Racket

- Output to standard output current
- Can be combined with other command line tools

- wc - Count words

```
$ racket letter.rkt Tillmann Rendel Klaus | wc -w
49
```

Executable programs with Racket

- Menu item Racket -> Program file create an executable file
- Creates native executable file
- Racket no longer needs to be installed on the target computer
- Read/write files: add teachpack 2htdp/batch-io

```
(write-file "letter.txt" (letter "Tillman" "Rendel" "Klaus"))
(read-file "letter.txt")
```

Interactive programs

- Universe teachpack
  - Support for interactive programs
  - Time signals
  - Mouse events
  - Keyboard input
  - Network traffic
  - Graphic output
- Several main functions
  - Handlers are evaluated in response to events

Interactive programs

Case study: see script

States

- Interactive program has condition
  - During execution
  - World has properties with values
  - Condition changes
- For interactive programs
  - Can influence how the condition changes
  - Function calls in response to events
  - Events can be controlled
- Program status: "WorldState"

Event handler

- Program defines event handler
  - Function
  - When the event occurs: Call the function
  - Input: Description of the event + current WorldState
  - Output: new WorldState

- Events:
  - on-mouse-event: Position of the cursor + event type (movement, click, ...) + WorldState
  - on-tick-event

Install handler

- Handler must be linked to event
- Function big-bang from Universe
- Result: last WorldState

```
install all event handlers; initialize world state to 500
(big-bang 500
    (on-tick on-tick-event 0.1)
    (on-mouse on-mouse-event)
    (to-draw render)
    (stop-when end-of-the-world render))
```

Install handler

- Handler must be
- Function big-ba
- Result: last World

```
install all event handle
(big-bang 500
    (on-tick on-tick-event 0.1)
    (on-mouse on-mouse-event)
    (to-draw render)
    (stop-when e
```

Counts down from 500. Initial WorldState

Dealer function for time signal

One counter step after 0.1 second

Dealer function for mouse event

Function for graphically displaying the WorldState

Function for determining whether the program has ended.

Install handler

- Handler must be linked to event
- Function big-bang from Univ
- Result: last WorldState

```
install all event handlers; initial world state to 500
(big-bang 500
    (on-tick on-tick-event 0.1)
    (on-mouse on-mouse-event)
    (to-draw render)
    (stop-when end-of-the-world render))
```

Second argument optional. If omitted, the interval is 1/28 second.

Second argument optional. Function that draws the image for the last WorldState. If omitted, the image is not updated again.

big-bang

- Handler must be linked to event
- Function erse
- Result:

Special form, not an ordinary function.

```
install all event handlers; initiaiz
(big-bang 500
    (on-tick on-tick-event 0.1)
    (on-mouse on-mouse-event)
    (to-draw render)
    (stop-when end-of-the-world render))
```

No function call. Special clause from big-bang.

big-bang

- Optional clauses
  - If no handler is installed for the event type, these events are ignored
- Only to-draw clause is required
- Binding of event to function happens with big bang
  → Names do not matter

```
; install all event handlers;
; initialize world state to 500
(big-bang 500
    (on-tick count-down 0.1)
    (on-mouse on-mouse-event)
    (to-draw render)
    (stop-when end-of-the-world render))
```

```
(define (count-down world)
    (- world 1))
```

Sequence of event handlers

WorldState:

500

(on-tick-event 500) → 499

499

(on-tick-event 499) → 498

498

(on-mouse-event 498 123 456 "move") → 498

(on-mouse-event 498 123 456 "button-down") → 398

398

Sequence of event handlers

- Sequence of events determines sequence of handler function calls
- In functional programming languages: Sequence through nesting

Sequence of event handlers

(on-tick-event 500) → 499

(on-tick-event 499) → 498

(on-mouse-event 498 123 456 "move") → 498

(on-mouse-event 498 123 456 "button-down") → 398

Sequence of event handlers

```
(on-tick-event
   (on-tick-event 500)) → 498

(on-mouse-event 498 123 456 "move") → 498

(on-mouse-event 498 123 456 "button-down") → 398
```

Sequence of event handlers

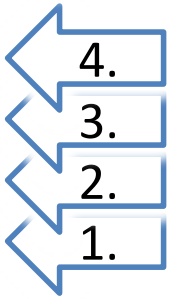
```
(on-mouse-event
   (on-tick-event
      (on-tick-event 500))
   123 456 "move") → 498

(on-mouse-event 498 123 456 "button-down") → 398
```

Sequence of event handlers

This allows event sequences to be simulated in tests.

```
(check-expect
   (on-mouse-event
      (on-mouse-event
         (on-tick-event
            (on-tick-event 500))
         123 456 "move")
      123 456 "button-down")
 398)
```

```
(on-mouse-event
   (on-mouse-event
      (on-tick-event
         (on-tick-event 500))
      123 456 "move")
   123 456 "button-down") → 398
```



Data types

- Set of values with common meaning

- So far:
  - Figures
  - Strings
  - Pictures
  - Truth values
  - Self-defined data types

unlimited

limited

Interpretation possibly limited

Enumeration types

- So far:
  - Self-defined data types as an interpretation of primitive types
  - Only certain values make sense, then:
  - "Enumeration type" or "Enumeration type"

```
A TrafficLight shows one of three colors:
; - "red"
; - "green"
; - "yellow"
; interp. each element of TrafficLight represents which colored
; bulb is currently turned on
```

Enumeration types

- Suitable for case distinctions

```
; TrafficLight -> TrafficLight
; given state s, determine the next state of the traffic light
(check-expect (traffic-light-next "red") "green")

(define (traffic-light-next s)
  (cond
    [(string=? "red" s) "green"]
    [(string=? "green" s) "yellow"]
    [(string=? "yellow" s) "red"]))
```

What happens if a string other than "red", "gren", "yellow" is passed?
(traffic-light-next "blue")

Error: *cond: all question results were false*

Enumeration types

- Another example: MouseEvent

A MouseEvt is one of these strings:

; - "button-down"

; - "button-up"

; - "drag"

; - "move"

; - "enter"

; - "leave"

Design recipe with enumeration types

- Design recipe step 3: Tests
- Function with enumeration type for argument
- A test for each possible value of the argument

```
(check-expect (traffic-light-next "red") "green")
(check-expect (traffic-light-next "green") "yellow")
(check-expect (traffic-light-next "yellow") "red")
```

Design recipe with enumeration types

- Design recipe step 4: Template

- For parameter type with enumeration type:
  Case differentiation with possible values

```
(define (traffic-light-next s)
  (cond
    [(string=? "red" s) ...]
    [(string=? "green" s) ...]
    [(string=? "yellow" s) ...]))
```

If argument with enumeration type is only used in auxiliary function, no case distinction appears here

Interval types

- List of all possible values
  - Impossible with an infinite number of values
  - Nonsensical with a large number of values
  - Then also difficult to read

- Specification of value ranges or "intervals"

Interval types

- Example: Simulation of a landing UFO
  - Using the big-bang function
  - WorldState corresponds to the height of the UFO (from above)

- Extension: Status bar
  - "decending" for height above 1/3 of the image
  - "closing" underneath
  - "lands" when touching down

Interval types

Case study: see script

Interval types

- Numbers and strings are comparable:

```
> ( string<? "a" "b")
#true
> ( > 9 2)
#true
> (< 2 2.1)
#true
> (string>=? "ab" "abc")
#false
```

Intervals

- Defining ranges/intervals using larger/smaller comparisons
- One or two borders
- Closed border
  - Limit value is included
  - >= or <=
- Open border
  - Limit value is not included
  - > or <

Intervals

- Definition of constants for interval limits

```
; constants:
(define WIDTH 300)
(define HEIGHT 100)
...
(define BOTTOM (- HEIGHT (/ (image-height UFO) 2)))
(define CLOSE (* 2 (/ HEIGHT 3)))
A WorldState is a Number. It falls into one of three intervals:
; - between 0 and CLOSE
; - between CLOSE and BOTTOM
; - at BOTTOM
; interp. height of UFO (from top)
```

Definition of interval limits

Use of interval limits

Interval types

- Intervals not relevant for all functions
  - render does not need to be adjusted
- In the example, rendering the status bar depends on the interval
- Functions that are dependent on intervals usually have a case distinction

Design recipe with interval types

- Design recipe step 3: Tests

- For parameters with interval type
  - At least one test per interval
  - Test interval limits in particular

Design recipe with interval types

- Design recipe step 4: Template

- For parameters with interval type:
  Case differentiation with possible values

```
WorldState -> Image
; add a status line to the scene created by render
(define (render/status y)
  (cond
    [(<= 0 y CLOSE) ... ]
    [(< CLOSE y BOTTOM) ... ]
    [(= y BOTTOM) ... ]))
```

Case differentiation not necessarily at the highest level

Design recipe with interval types

- In all three cases
  - Render status bar
  - The only difference: Text
- DRY principle: Case differentiation for the text argument

```
WorldState -> Image
; add a status line to the scene create by render
(define (render/status y)
  (above
    (text
      (cond
        [(<= 0 y CLOSE) "descending"]
        [(< CLOSE y BOTTOM) "closing in"]
        [(= y BOTTOM) "landed"])
      12 "black")
    (render y)))
```

Use in the to-draw clause of big-bang.