

# Informatik 2

---

Sommersemester 2015

Eberhard Karls Universität Tübingen

Prof. Dr. Torsten Grust  
[torsten.grust@uni-tuebingen.de](mailto:torsten.grust@uni-tuebingen.de)

# Willkommen!

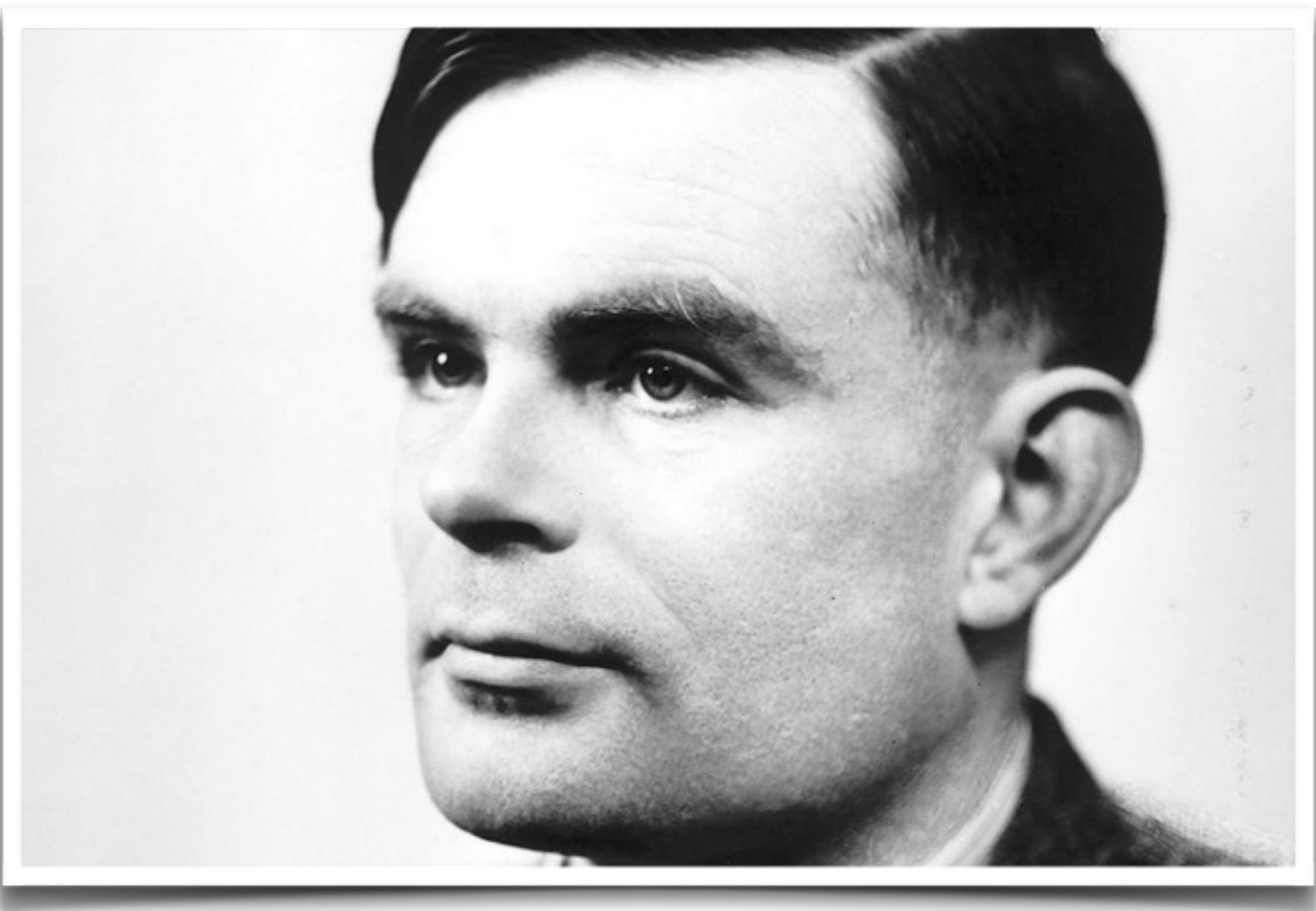
# Torsten Grust?

---

1989–1994	Studium der Informatik an der TU Clausthal
1994–1999	Promotion an der U Konstanz
2000	<i>Visiting Researcher</i> , IBM (USA)
2000–2004	Habilitation an der U Konstanz
2004–2005	Professor für Datenbanksysteme an der TU Clausthal
2005–2008	Professor für Datenbanksysteme an der TU München
2008–heute	Professor für Datenbanksysteme an der U Tübingen

# Alan vs. Alonzo

---



Alan M. Turing (1912–1954)

# Alan vs. Alonzo

---



Alonzo Church (1903–1995)

# Zustandsänderung vs. Auswertung

---

- Programmieren  $\equiv$  **Zustand** der Maschine kontrollieren:

**i := i + 1**

- Jetzt: Programmieren  $\equiv$  Berechnung von **Werten**:

**f(x,y)**

# Zustandsänderung vs. Auswertung

---

- Programmieren  $\equiv$  **Zustand** der Maschine kontrollieren:

**i := i + 1**

- Jetzt: Programmieren  $\equiv$  Berechnung von **Werten**:

**(f x y)**

# Sequenzen vs. Komposition

---

- Programmkonstruktion  $\equiv$  **Sequenzen** von Anweisungen:

$$\begin{array}{l} i := i + 1; \\ x := xs[i] \end{array}$$

- Programmkonstruktion  $\equiv$  Komposition von **Funktionen**:

$$g(f(x, y))$$

# Sequenzen vs. Komposition

---

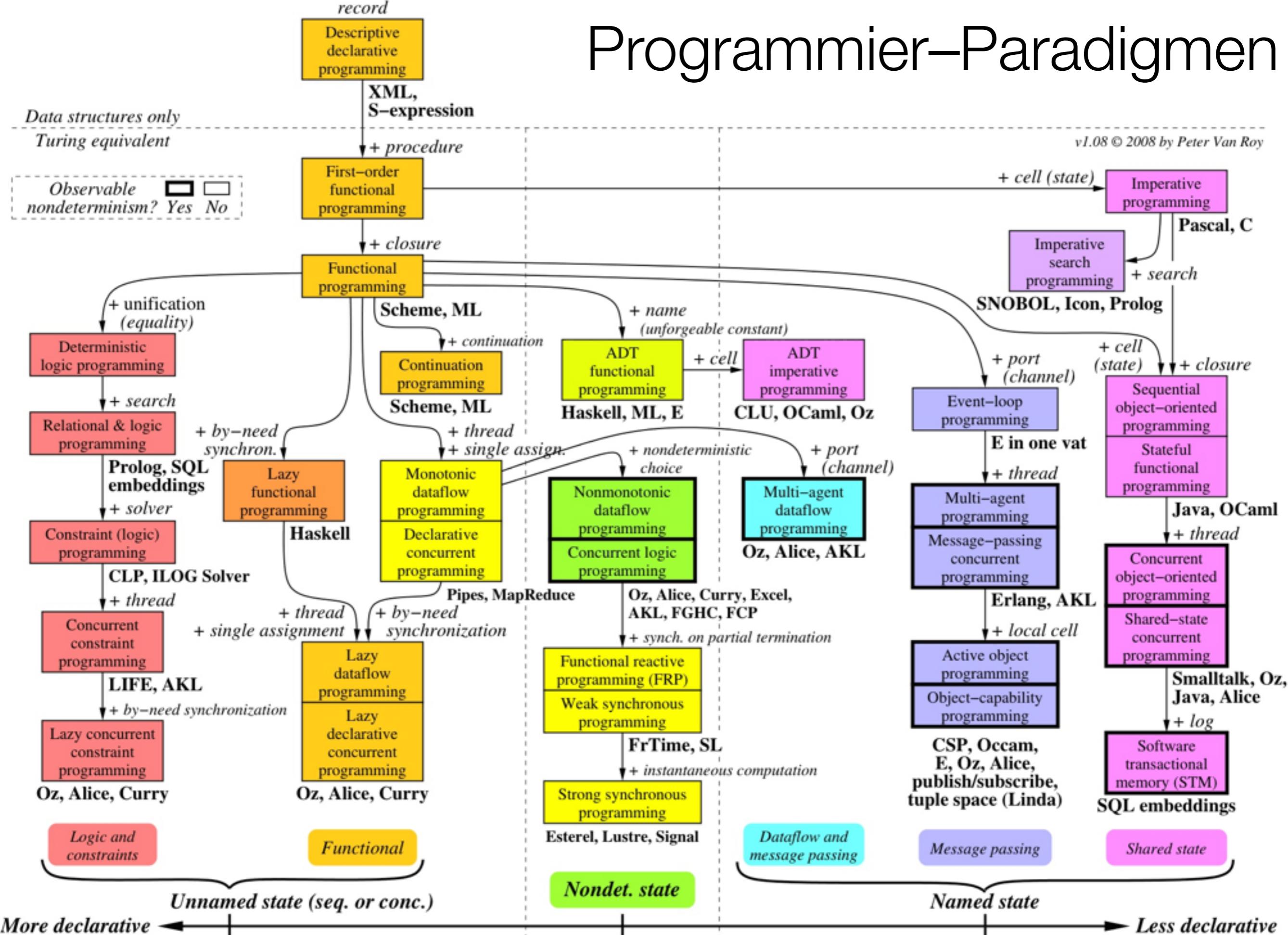
- Programmkonstruktion  $\equiv$  **Sequenzen** von Anweisungen:

$$\begin{aligned} i &:= i + 1; \\ x &:= xs[i] \end{aligned}$$

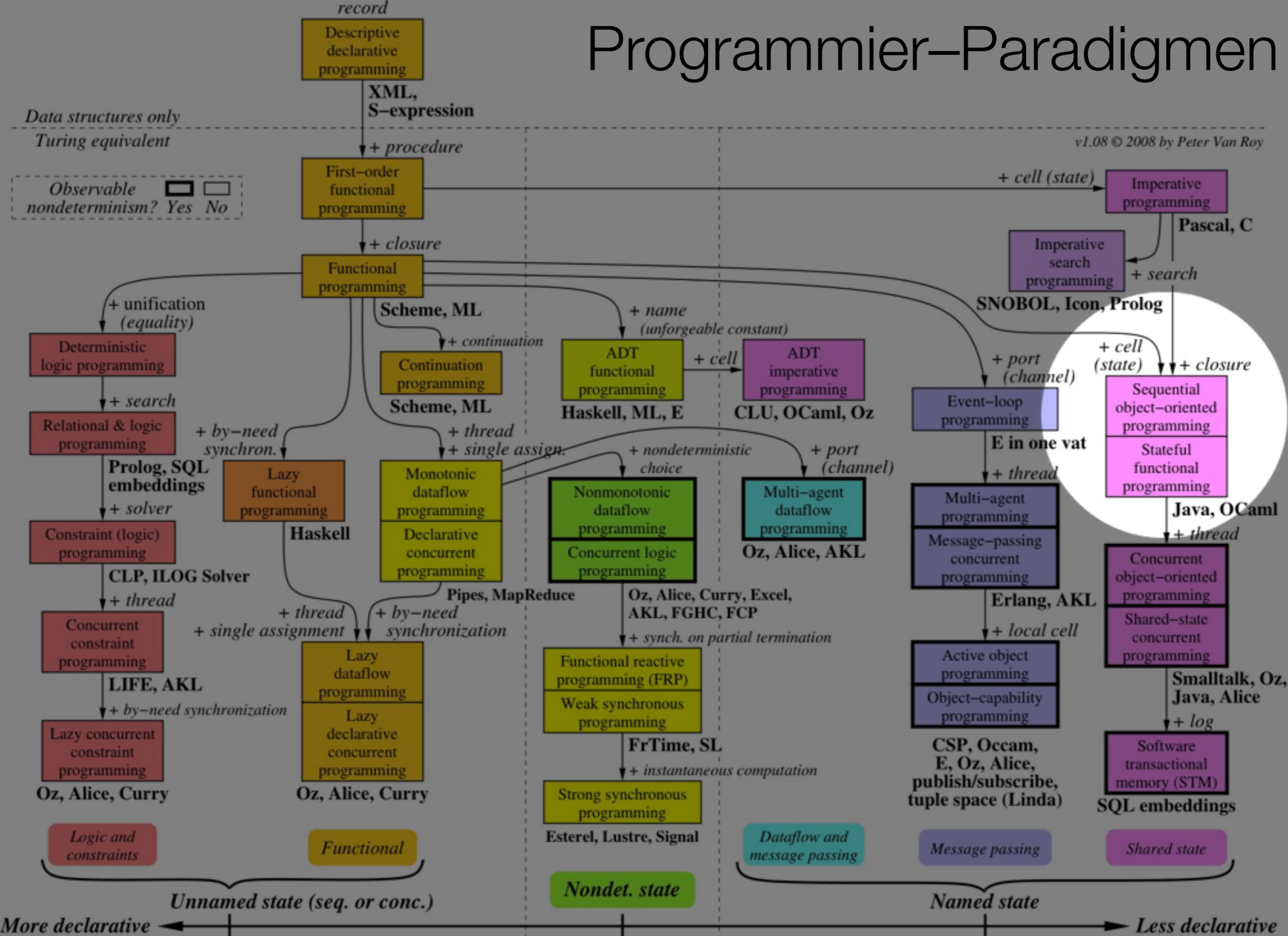
- Programmkonstruktion  $\equiv$  Komposition von **Funktionen**:

$$(g (f x y))$$

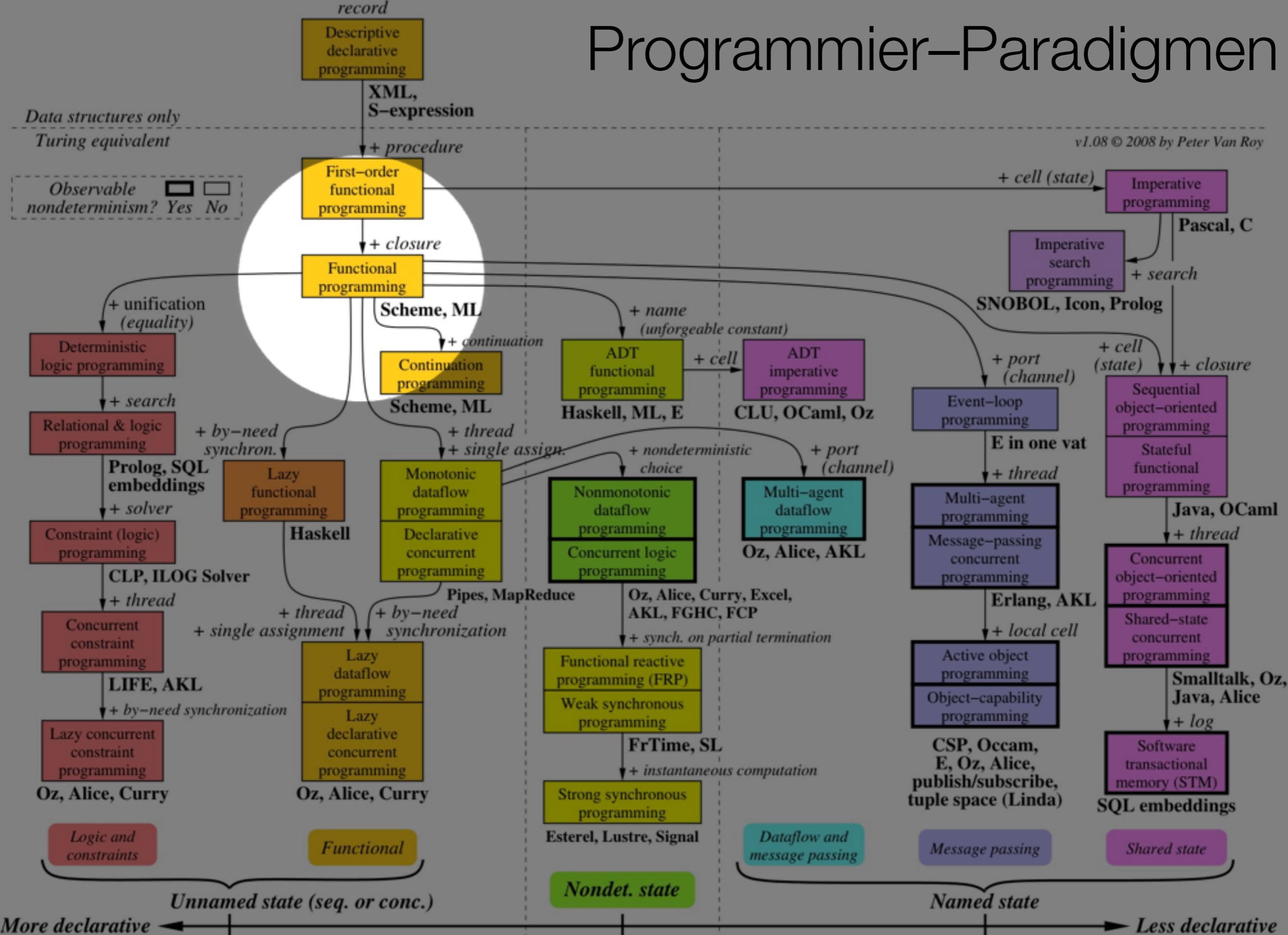
# Programmier-Paradigmen



# Programmier-Paradigmen



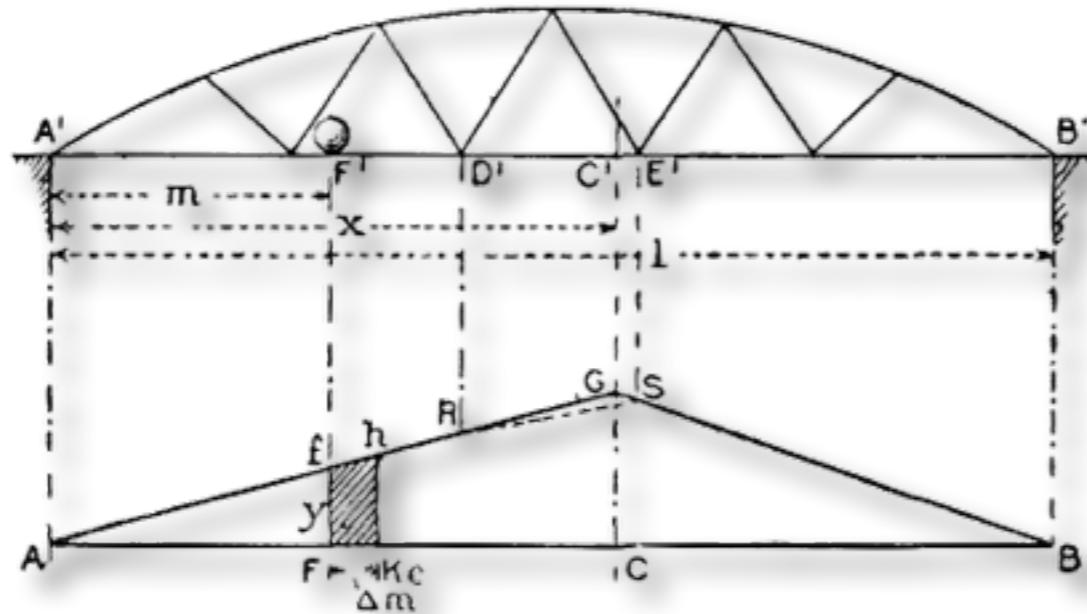
# Programmier-Paradigmen



# Informatik: Ingenieurwissenschaft



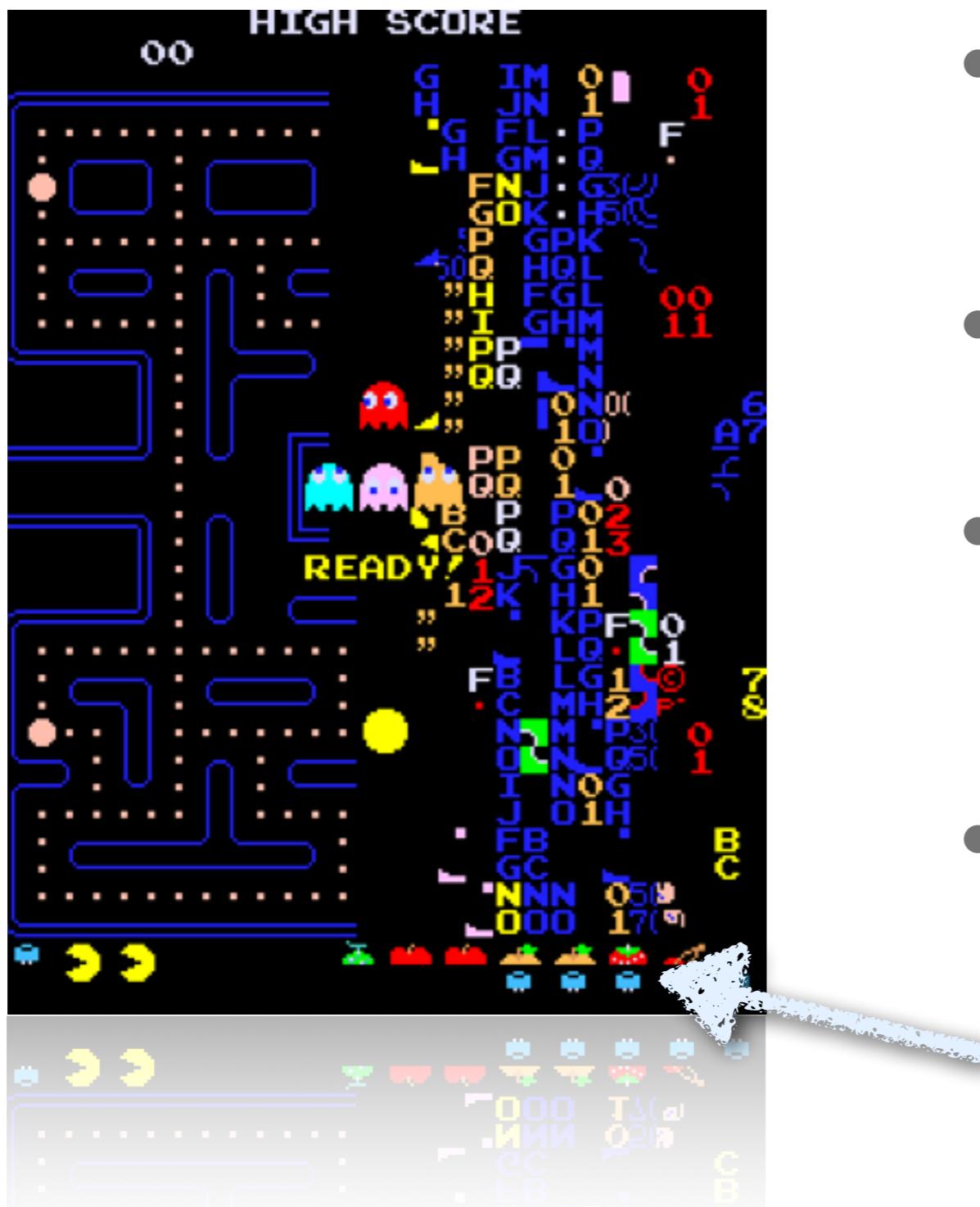
# Informatik: (auf dem Weg zur) Ingenieurwissenschaft



- Anwendung erprobter (teils formal korrekt bewiesener) **Konstruktionsprinzipien** für Software vs. “Let’s hack!”
- Systematisches, erschöpfendes **Testen** und Verifizierung mathematischer (oft: algebraischer) **Eigenschaften** von Programmen

# The Pac-Man Kill Screen

---



- Pac-Man: Level  $\ell$  in 8-Bit-Register
- 8 Bit:  $0 \leq \ell \leq 255$
- Nach Level 255: Überlauf ( $255 + 1 \rightarrow 0$ )
- Eigentliches Problem:  
Prozedur zum Zeichnen  
von  $1 \dots \ell - 1$  Früchten

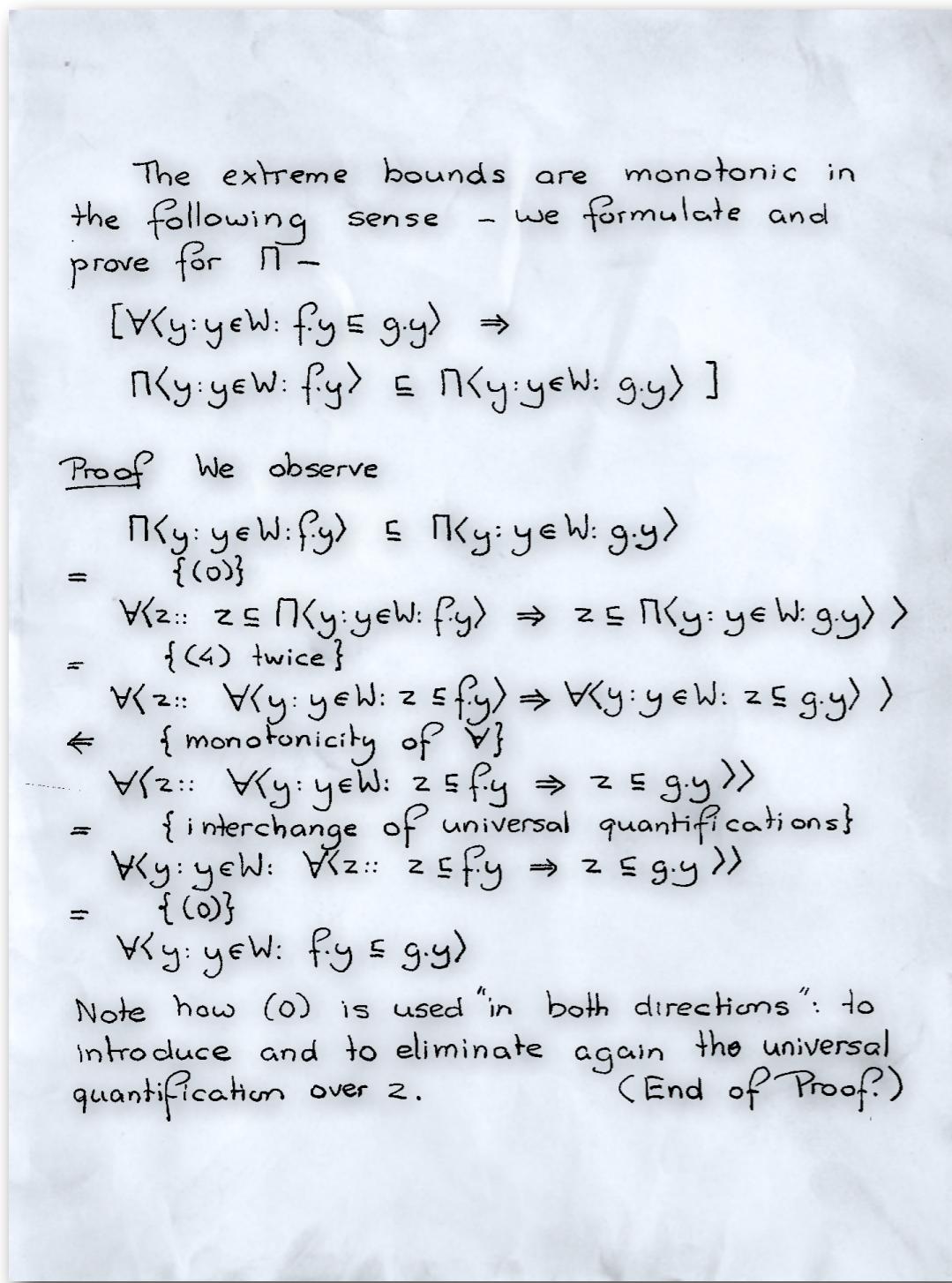
# Ariane 5: 10 Jahre, $7 \times 10^9$ US\$

---



- Erster “Flug”, Juni 1996
  - Umwandlung einer 64-Bit Fliesskommazahl in eine 16-Bit Ganzzahl löst Ausnahmebehandlung aus
- ⇒ Ausfall der Horizontalsteuerung

# Informatik: Formale Wissenschaft



- Viele Objekte der Informatik (Programme, Daten) zeigen interessante, mathematische Strukturen
- Bereits ein relativ kleiner mathematischer Werkzeugkasten kann entscheidende Einsichten bringen
- "...  $f(f(x)) = f(x)$ ." Gut zu wissen.

# Ein Parkplatz–Problem



“Auf einem Parkplatz stehen PKWs und Motorräder (ohne Beiwagen). Zusammen seien es  $n$  Fahrzeuge mit insgesamt  $r$  Rädern. Bestimme die Anzahl  $P$  der PKWs.”

- Dies beschreibt gleich eine ganze Menge von Problemen, für jede Wahl von  $n$  und  $r$ .
- Also haben wir es mit den folgenden Funktionen zu tun:

$P(n,r)$  Anzahl der PKWs

$M(n,r)$  Anzahl der Motorräder

# Ein Parkplatz–Problem



- PKWs und Motorräder machen alle  $n$  Fahrzeuge aus.  
PKWs steuern jeweils 4, Motorräder jeweils 2 Räder zu den  $r$  Rädern bei:

$$P + M = n$$

$$M = n - P$$

$$4P + 2M = r$$

$$4P + 2(n - P) = r$$

$$4P + 2n - 2P = r$$

$$2P = r - 2n$$

$$P = 1/2 (r - 2n)$$

# Kein Parkplatz–Problem mehr?

---

- Eine Funktion als Lösung:  $P(n,r) = 1/2 (r - 2n)$ .  
Sieht gut aus.
- Oder?
  - $P(3,9) = 1.5$  Halbe PKWs?
  - $P(5,2) = -4$  Noch 4 PKWs einparken, dann ist der Parkplatz leer...
  - $P(2,10) = 3$  Subtil  
(mehr PKWs als Fahrzeuge).

# Verträge!

---

- Mit der Funktion  $P(n,r)$  haben wir soweit abstrahiert und uns damit vom tatsächlichen Parkplatz entfernt, bis wir in der Realität geltende wichtige Einschränkungen aus den Augen verloren haben.
- $P(n,r)$  erfüllt seine Aufgabe. Aber nur, sofern wir uns **bei der Anwendung einschränken**:  
 $r$  gerade und  $2n \leq r \leq 4n$
- Funktion  $P(n,r)$  kommt quasi mit **Vertragsbedingungen**.

# Programmieren mit Funktionen

---

- **Funktionen** gehören zu den zentralen Bausteinen der Mathematik:

$$f(x_1, x_2, \dots, x_n)$$

- In **Scheme** adoptieren wir diese Idee und nutzen **Funktionen als Bausteine**, aus denen wir ganze Programme konstruieren:

$$(f\ x_1\ x_2\ \dots\ x_n)$$

# Programmieren mit Funktionen

---

- Diese einfache Idee des *Programmierens mit Funktionen* begleitet uns durch das gesamte Semester:
  1. **Konstruiere** (oder nutze die eingebaute) Funktion  $f$ , dann
  2. **wende**  $f$  auf die benötigte Anzahl von **Argumenten an**:

(+ 40 2)

(odd? 42)

# Das Leibniz–Prinzip

---

- Aber hinter Funktionen steckt mehr als nur ihre kompakte Notation: ein nachvollziehbares, verlässliches Verhalten!
- In der Mathematik verlassen wir uns auf das folgende (wähle Funktion  $f$  und die Argumente  $x, y$  beliebig):

$$x = y \text{ und } f(x) = z \Rightarrow f(y) = z$$

- In Worten: um das Verhalten von  $f$  vorherzusagen, benötigen wir **ausschliesslich** das Argument (hier:  $x, y$ ).

# Gottfried Wilhelm von Leibniz

*Leibniz*

- 1646–1716
- Wesentliche Beiträge zur Mathematik, Paläontologie, Philosophie, Physik, Politik, Rechtswesen, ...
- 1673: Rechenmaschine



# Das Leibniz–Prinzip

---

- ... ist so “hartverdrahtet” in unserem Denken, das wir es teilweise nicht mehr explizit wahrnehmen (das ist gut so!). Zum Beispiel:
  - Wähle jetzt ein bestimmtes  $f$ , nämlich  $f = id$  mit  $id(x) = x$  (für jedes beliebige  $x$ ):

$$x = y \text{ und } id(x) = z \Rightarrow id(y) = z$$

# Das Leibniz–Prinzip

---

- ... ist so “hartverdrahtet” in unserem Denken, das wir es teilweise nicht mehr explizit wahrnehmen (das ist gut so!). Zum Beispiel:
  - Wähle jetzt ein bestimmtes  $f$ , nämlich  $f = id$  mit  $id(x) = x$  (für jedes beliebige  $x$ ):

$$x = y \text{ und } x = z \Rightarrow y = z$$

# Programmieren “ohne Leibniz”?

---

- Natürlich ist das möglich!
- Und es ist vielleicht erstaunlich: die weitaus meisten Programmiersprachen sichern den Entwicklern das Leibniz–Prinzip **nicht** zu.
- Die Vorhersagbarkeit des Verhaltens einer Funktion allein mittels ihres Argument verloren:

$$x = y \text{ und } f(x) = z \not\Rightarrow f(y) = z$$



# “Ohne Leibniz”: Konsequenzen

---

- Viele intuitive Annahmen gelten bei der Programmkonstruktion i.a. nicht mehr. Etwa:

$$f(x) + f(x) \neq 2 * f(x)$$

- Das Verhalten von  $f$  kann von einer unübersichtlichen **Vielzahl von (impliziten) Parametern** abhängen:
  - vorhergegangene Anwendungen weiterer Funktionen (inkl.  $f$ ), Systemzustand, Zeit, Eingaben des Users, Mondphase, ...

# Heisenbugs

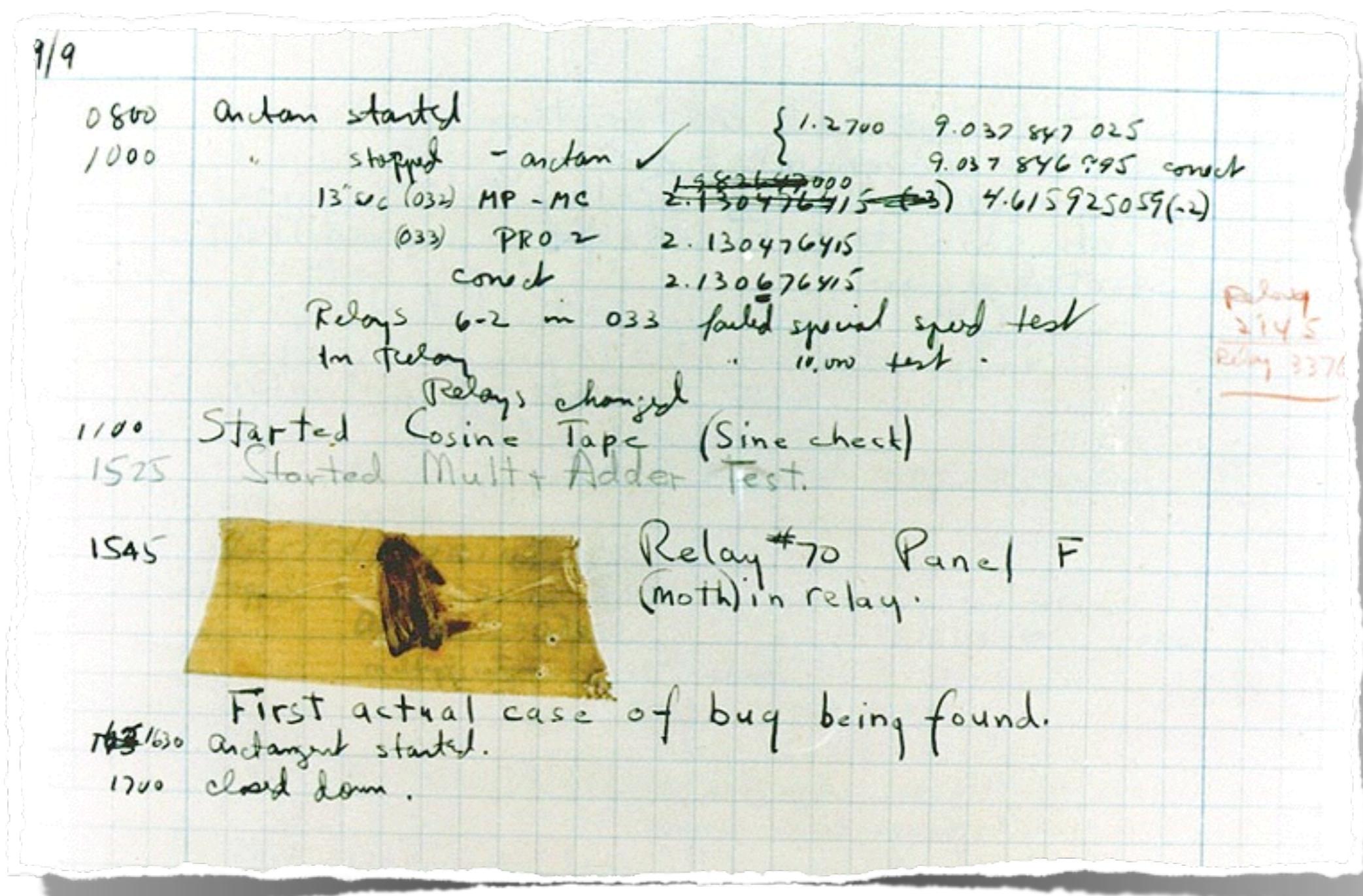
---

- Aus dem *Jargon File*:

heisenbug /hi: 'zen-buhg/ /n./ [from Heisenberg's Uncertainty Principle in quantum physics] A bug that disappears or alters its behavior when one attempts to probe or isolate it. (This usage is not even particularly fanciful; the use of a debugger sometimes **alters a program's operating environment** significantly enough that buggy code, such as that which relies on the values of uninitialized memory, behaves quite differently.)

- Eines ist klar: Programmkonstruktion unter Einfluss von/ mit Wirkung auf den Systemzustand bedarf besonderer Sorgfalt.

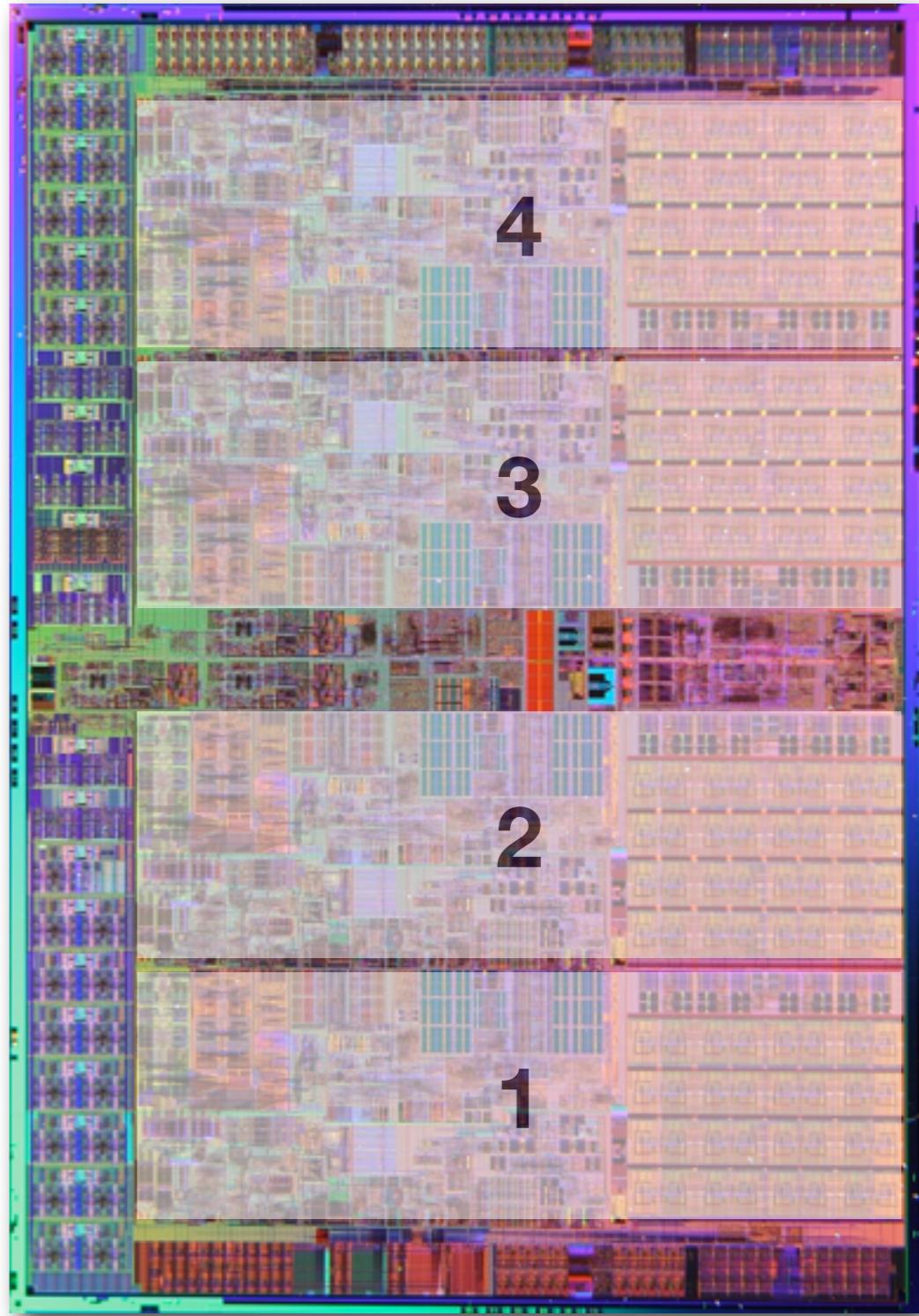
# Programmfehler (Bugs)



Motte im Relay 70 einer Mark II Aiken,  
entdeckt durch Grace Murray Hopper, 1945

# Leibniz und moderne Rechner

- Sei  $(f \ x \ y)$  eine Funktionsanwendung eines Programms.
- Laut Leibniz **beeinflussen** sich die Berechnungen von  $x$  und  $y$  **nicht** gegenseitig.
- Warum also nicht  $x$  und  $y$  **parallel** berechnen?



Intel Core i7 (Nehalem), 4-core CPU

# And: All the Cool Guys are Doing it

---

- Programmieren mit Funktionen (FP oder *functional programming*) gewinnt seit Jahren besonderen Einfluss:
  - Industrielle Forschung & Entwicklung baut auf FP:  
Verizon, Bloomberg LP, Comcast, S&P Capital, Facebook, Netflix, Twitter, Jane Street, Standard Chartered, Citrix, Nokia, Alcatel Lucent, Google, Credit Suisse, Deutsche Bank, McGraw-Hill, ING, ...
  - Ericsson's AXD301 ATM Switch basiert auf Erlang:  
The measured reliability is quoted as being 99.999999% (9 nines) corresponding to a down time of 31 ms/year.

# And: All the Cool Guys are Doing it

---

- Die grössten Datenmengen dieser Welt (bspw. Googles Web–Index) werden mit Hilfe von *MapReduce*—einer “Schablone” zur **systematischen Konstruktion** von funktionalen, parallelen Programmen—analysiert.
- In modernen Programmiersprachen (*C#*, *Clojure*, *Javascript*, *Perl 6*, *Python*, *Ruby*, *Scala*, ...) finden sich wesentliche Ideen der funktionalen Programmierung.
- Derzeit baut Apple mit *Swift* seine Entwicklungsplattform auf der Basis von FP–Prinzipien um.



# And: All the Cool Guys are Doing it

- Die grössten Datenbanken (z.B. Google's Web-Index) werden mit einer “Schablone” zur Entwicklung von funktionalen, parallelisierten Algorithmen programmiert.
- In modernen Programmiersprachen (z.B. Javascript, Python, F#) sind wesentliche Ideen der funktionalen Programmierung integriert.
- Derzeit baut Apple seine mobile Software-Plattform auf der Basis von FP-Prinzipien um.

*objc ↑↓*  
*Functional*  
*Programming*  
*in Swift*



*Chris Eidhof, Florian Kugler, and Wouter Swierstra*

pw. Googles  
Büro—einer  
Funktion von  
analysiert.

*Clojure,*  
...) finden sich  
funktionaler Programmierung.

Entwicklungs-  
umgebung um.



# Das Info 2-Team



# Das Info 2–Team

---

Alex Ulrich,  
Dennis Butterstein

Wissenschaftliche Mitarbeiter (Doktoranden)

Organisation des gesamten Übungsbetriebes  
(Übungsblätter, Präsenzübungen)

## Die Info 2–Tutoren

Felix Bartusch, Jonas Benn, Alexander Blöck, Tim Häring, Yannik Heuper, Jan-Peter Hohloch, Lukas Holländer, Felix Holzwarth, Jakob Koschel, Sophia Mersmann, Robin Mesaric, Janis Plöger, Peter Richter, Andreas Rist, Yves Röhm, Cornelia Schulz, Tobias Stumpp, Steffen Tacke, Marc Weitz

Durchführung des Übungsbetriebes, Korrektur – ansprechbar und generell **hilfswillig**

# Der Info 2–Übungsbetrieb

---

## Übungsblätter

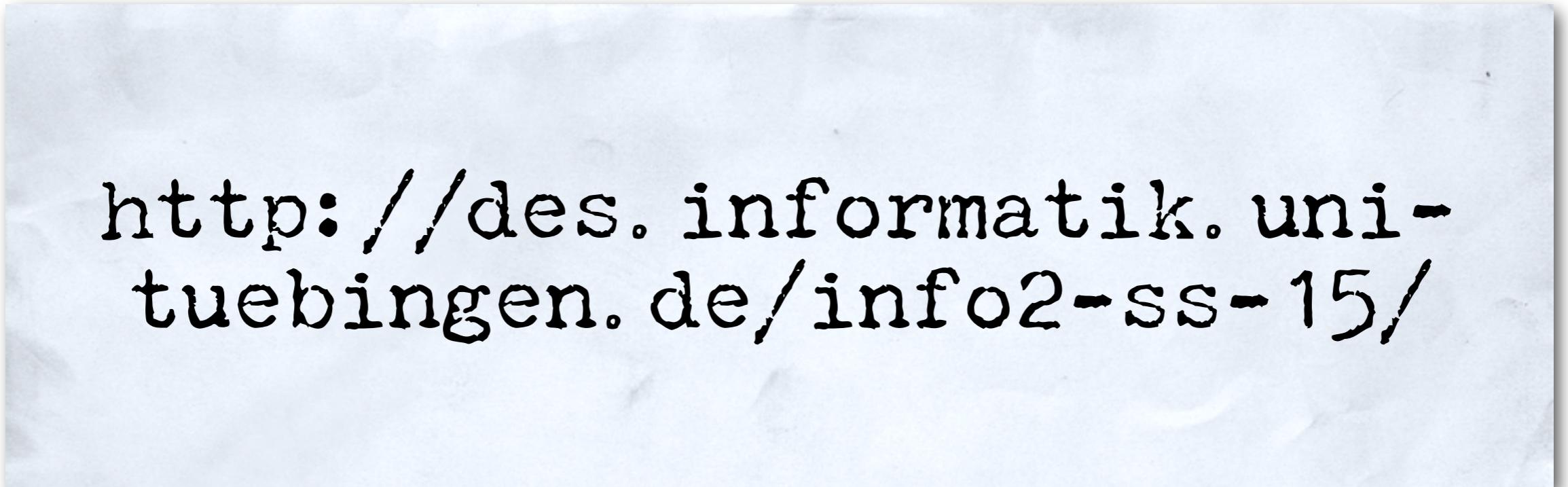
- Ausgabe wöchentlich  
(1. Ü-blatt: 21.4.)
- Bearbeitung:  
Dienstag – Dienstag
- Feste Zuordnung zu  
Tutor/Übungsgruppe
- Bildung von 2er-Teams  
innerhalb der Ü-gruppe

## Präsenzübungen (PÜ)

- Wöchentlich  
(1. PÜ: 27.4. – 30.4.)
- Bearbeitung: in einer ca.  
2-stündigen PÜ im H33
- Buchung eines PÜ-  
Termins (wöchentlich)
- Individuelle Bearbeitung,  
mit Tutorunterstützung

# Das Info 2-Web

---



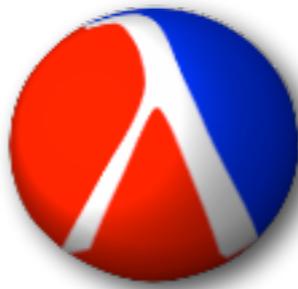
`http://des.informatik.uni-tuebingen.de/info2-ss-15/`

- Neueste Informationen zu V und Ü, Termine, Downloads (Folien, Dokumente, Code), Abgabe Code zu Ü-blättern, Buchung von PÜ-Terminen, Link zum Forum, ...
- *Visit often, visit early!* (am besten gleich heute abend)

# Programmieren, Live.



# DrRacket



- Programmierumgebung für **Scheme**, einer Sprache die funktionales Programmieren (und mehr) unterstützt.
- Interaktiv, hilfsbereit, konfigurierbar, kostenfrei, quasi überall verfügbar.

```
(define-contract even-natural (combined natural (predicate even?)))

; Anzahl PKWs auf einem Parkplatz mit n Fahrzeugen und r Rädern
(: parking-lot-cars (natural even-natural -> natural))
(check-expect (parking-lot-cars 6 12) 0)
(check-expect (parking-lot-cars 3 12) 3)

(define parking-lot-cars
  (lambda (n r)
    (/ (- r (* 2 n))
      2)))

(parking-lot-cars 2.5 3)
```

Willkommen bei [DrRacket](#), Version 6.1.1 [3m].  
Sprache: [Die Macht der Abstraktion](#); memory limit: [4096 MB](#).  
-1.0  
> |

2 Tests geläufen.  
Alle ~a Tests waren erfolgreich!

2 Signaturverletzungen.

Signaturverletzungen:

bekam 2.5 [in parking-lot-cars.rkt, Zeile 13, Spalte 0](#), Signatur [in p](#)  
bekam 3 [in parking-lot-cars.rkt, Zeile 13, Spalte 0](#), Signatur [in par](#)

Ausblenden      Ablegen

Die Macht der Abstraktion ▾

4:2      301.71 MB