Theoretische Informatik

Berechenbarkeit

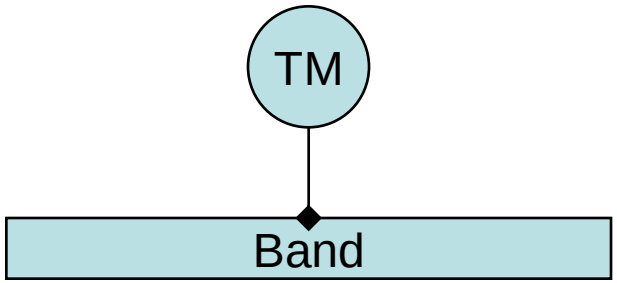
Turing Maschine

- Endlicher Automat mit unendlichem Speicher
- Ein **Modell** eines realen Computers
- Was ein Computer berechnen kann, kann auch eine TM berechnen.
- Was die TM *nicht* berechnen kann, kann auch ein Computer *nicht* berechnen.

IKS

Turing Maschine

- Zusatzfunktionen gegenüber einem DFA:
  - Lesen und Schreiben auf einem Band
  - Band nach links und rechts bewegen
  - Band ist unendlich lang
  - Endzustände (für OK oder Fehler) beenden, auch wenn noch Eingaben vorliegen



IKS

Turing Maschine

- Neue Übergangsfunktion:

  $\delta : Q \times T \rightarrow Q \times T \times [L,R]$

  Q : Zustandsmenge

  T : Alphabet auf dem Band

  [L,R] : Richtung der Bandbewegung

- Die Turingmaschine liest und schreibt bei jedem Schritt auf dem Band und bewegt es ständig hin und her.

IKS

Turing Maschine

- Beispiel einer Berechnung:
  1. Prüfe die Eingabe, ob die Form a*b*c* stimmt
  2. Fahre zum Anfang des Bandes zurück
  3. Ersetze erstes a durch # und suche erstes b
  4. Ersetze jedes c in $b^n\#*c^n$ durch #
  5. Gehe zurück zu 2, bis kein a mehr da ist
  6. Prüfe ob kein c mehr da ist, dann OK
- Was tut das Programm?

IKS

Chomsky Hierarchie

- Typ 3 – reguläre Sprachen
  - Endlicher Automat
- Typ 2 – kontextfreie Sprachen
  - Stackautomat
- Typ 1 – kontextsensitive Sprachen
  - Turingmaschine mit endlichem Band
- Typ 0 – unbeschränkte Sprachen
  - Turingmaschine mit unendlichem Band

IKS

Registermaschine

- DFA mit Zusatzeigenschaften:
  - Unendliche Anzahl von Registern R_i ($i \geq 0$)
  - Inkrementieren eines Registers
  - Dekrementieren eines Registers bis 0
  - Test eines Registers auf 0
- Registermaschinen **modellieren** reale Rechner etwas natürlicher

Registermaschinen

- Register- und Turingmaschinen sind äquivalent
- Beweis durch Simulation:
  - Für alle Operationen einer Turing Maschine existiert ein Registermaschinenprogramm
  - Für alle Operationen einer Registermaschine existiert ein Turingmaschinenprogramm
  - Idee: Zahl im Register = Anzahl von bestimmten Symbolen auf dem Band

IKS

λ - Kalkül

- **Modellierung** realer Computer durch Chomsky (parallel zu Turing)
  - λ x → x + 2 * x (x ist gebundene Variable)
  - α – Konvertierung: Variablenumbennung

    λ x → x + 2 * x <=> λ y → y + 2 * y
  - β – Reduktion: Variablenersetzung

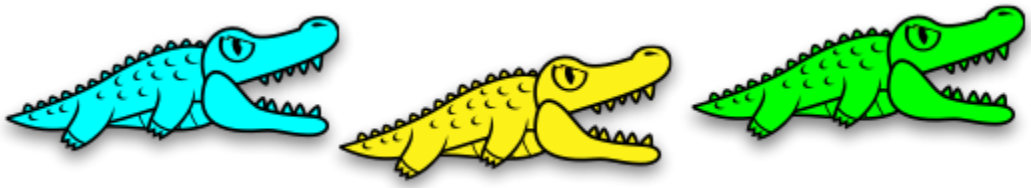
    (λ x → x + 2 * x) (4 – 1) <=> (4 – 1) + 2 * (4 – 1)
  - η – Reduktion: Variablenentfernung

    λ x → f x <=> f
- http://worrydream.com/AlligatorEggs/

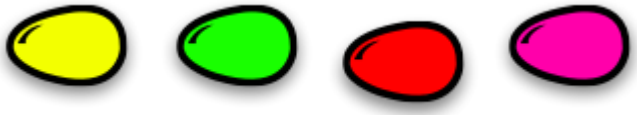
IKS

λ – Kalkül: Spielfiguren

- Hungrige Krokodile fressen alles

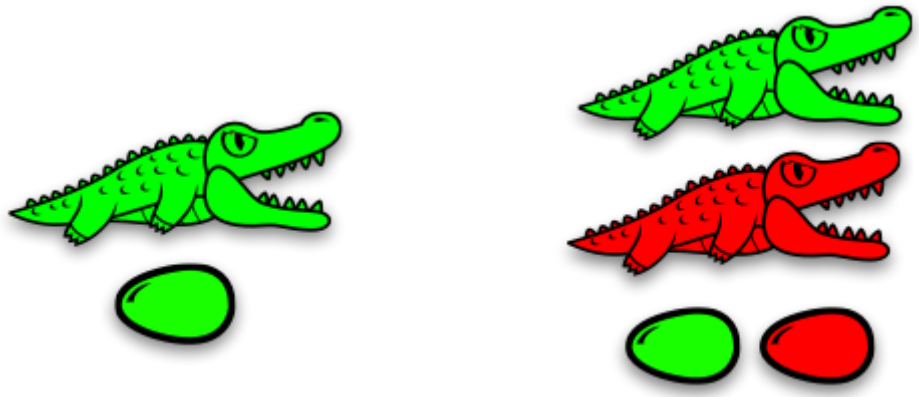


- Eier müssen beschützt werden



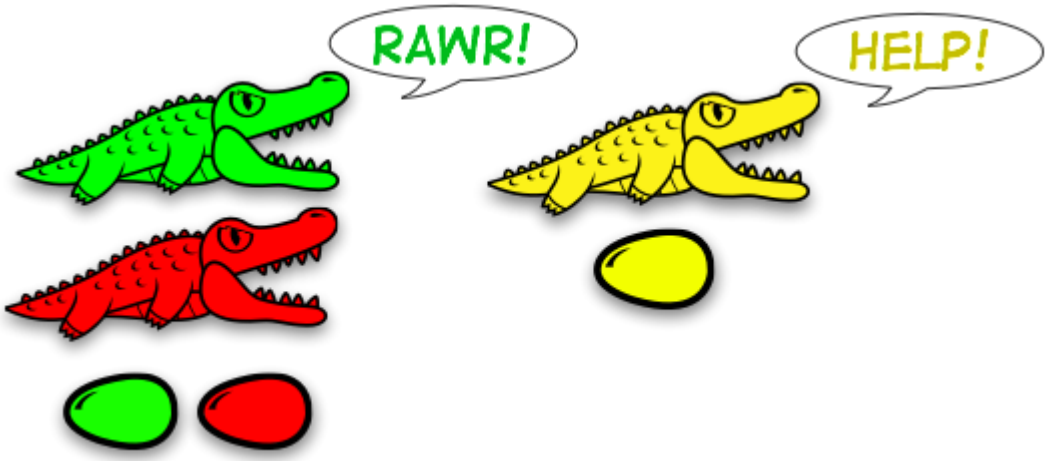
λ – Kalkül: Familien

- Krokodile schützen die unter ihnen stehenden Familienmitglieder
- Jedes Ei braucht ein Krokodil



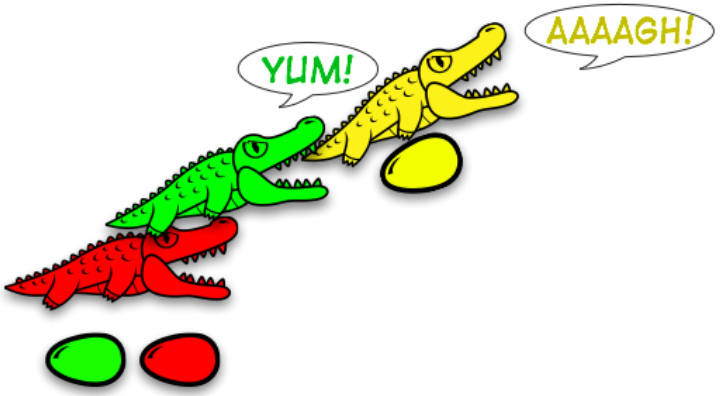
λ – Kalkül: β – Reduktion

- Hungrige Krokodile essen, was vor ihrem Maul steht.
- Das oberste Krokodil darf zuerst essen

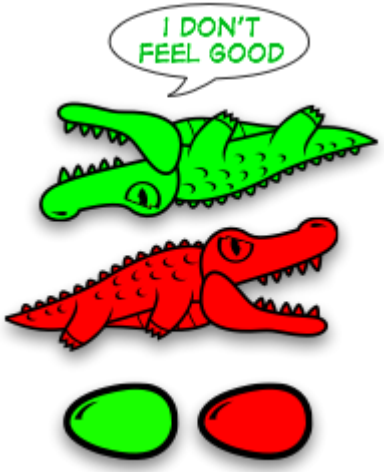


λ – Kalkül: β – Reduktion

- Es werden ganze Familien gefressen



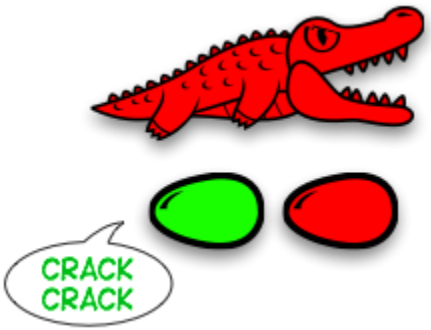
- Wer zuviel frißt, stirbt



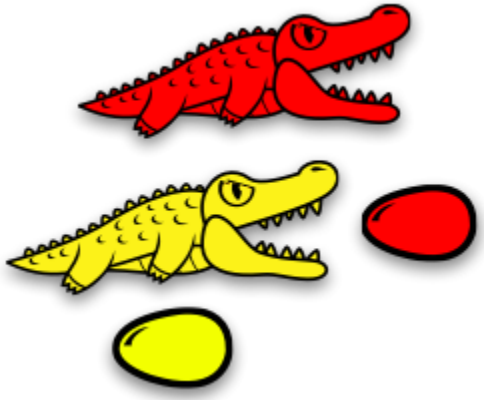
IKS

λ – Kalkül: β – Reduktion

- Ungeschützte Eier schlüpfen

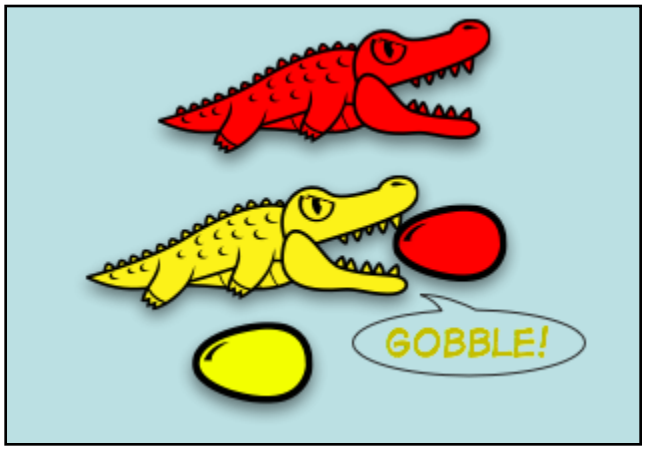


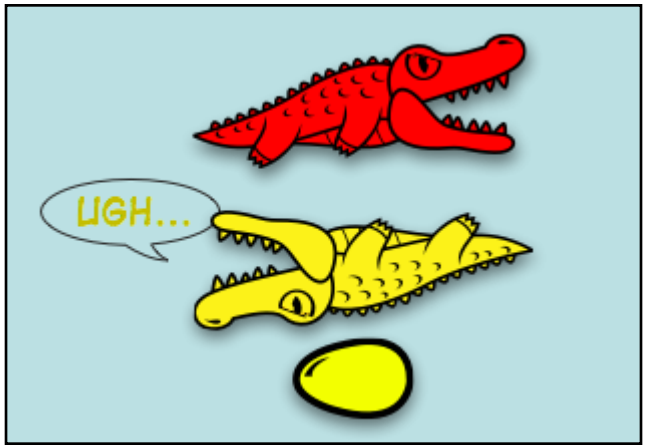
- Es schlüpft genau das, was gefressen wurde

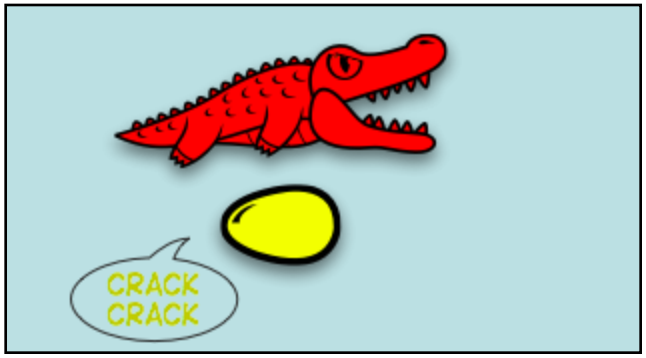


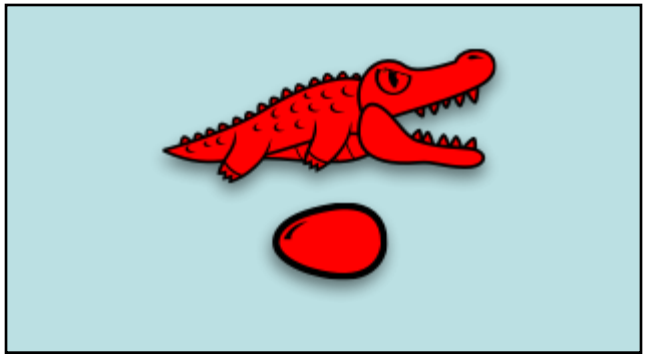
λ – Kalkül: β – Reduktion

- Und es wird weiter gefressen



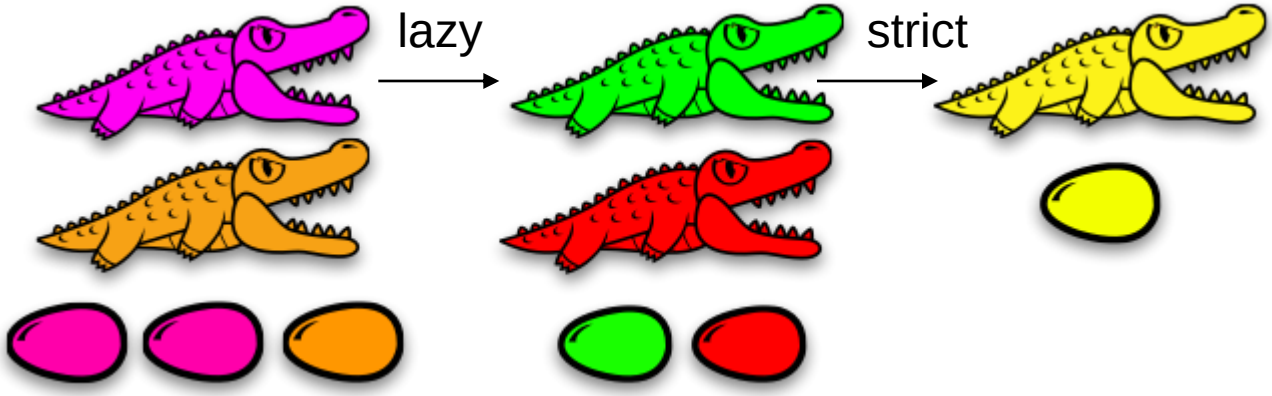




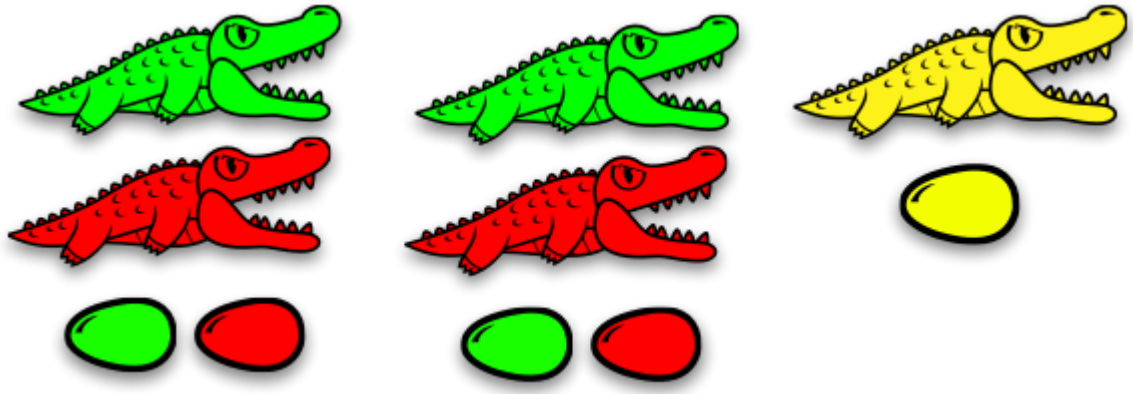


λ – Kalkül: β – Reduktion

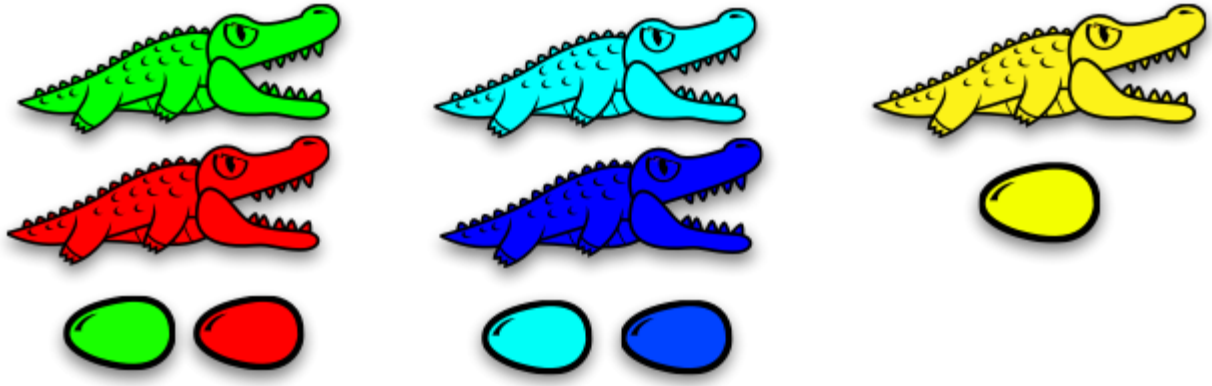
- Längere Ausdrücke sind verschieden reduzierbar
- Lazy (verzögert) oder Strict (mit Werten)



λ – Kalkül: α – Konvertierung



- Doppelte Farben müssen entfernt werden.



λ – Kalkül

- Das λ – Kalkül ist Basis aller modernen Programmiersprachen
- Turing Maschinen und λ – Kalkül sind äquivalent
- Beide Verfahren definieren *Algorithmen*

IKS

Berechenbarkeit

- Was eine TM akzeptiert oder ablehnt, ist *berechenbar* (Church-Turing-These)
- Berechenbare Probleme <=> Algorithmus
- Aber: Endlosschleifen sind möglich
- **Halteproblem**: Kann man allein vom Sourcecode einer TM entscheiden, ob diese immer anhält?

IKS

Theoretische Informatik

Komplexitätstheorie

Komplexität

- Ermittlung des Zeitverhaltens und des Platzbedarfes in Abhängigkeit von der Eingabe
- Drei Fälle:
  - Bester Fall: Trivialität
  - Schlechtester Fall: Schwerste Berechnung
  - Durchschnittlicher Fall: Typisches Verhalten
- Nur asympthotische Abschätzung

IKS

Komplexität

- f=O(g) (Groß-O)
  - Obere Abschätzung
  - f(x) bleibt bei großen Werten unter g(x)
  - Meist sieht f wie g aus
  - Konstante Faktoren (Skalierung) entfällt
- O(1) – Konstantes Verhalten
- O(n) – Lineares Verhalten
- O(log n) – Logarithmisches Verhalten

IKS

Komplexitätsklassen

- **P** – Alle Algorithmen die $O(n^k)$ sind
  - – Schnelles Finden der Lösung
- **NP** – Alle Probleme deren Lösungs*prüf*-algorithmus in **P** liegt
  - – Schnelles Überprüfung der Lösung
- Offenes Problem der Informatik:

  Ist **P** und **NP** gleich?

IKS

Komplexitätsklassen

- **PSPACE** – Probleme, die $O(n^k)$ Platz mit deterministischen Algorithmen brauchen
- **NPSPACE** – Probleme, die $O(n^k)$ Platz mit nichtdeterministischen Algorithmen brauchen
- **EXPTIME** – Probleme, die $O(2^n)$ Zeit brauchen
- **$P \leq NP \leq PSPACE = NPSPACE \leq EXPTIME$**

IKS