Repetition

Übersicht

- Organisatorisches: Ausserordentliches Tutorat
- Rekursion: Die Türme von Hanoi
- Listen: Verketten und umdrehen

Ziel

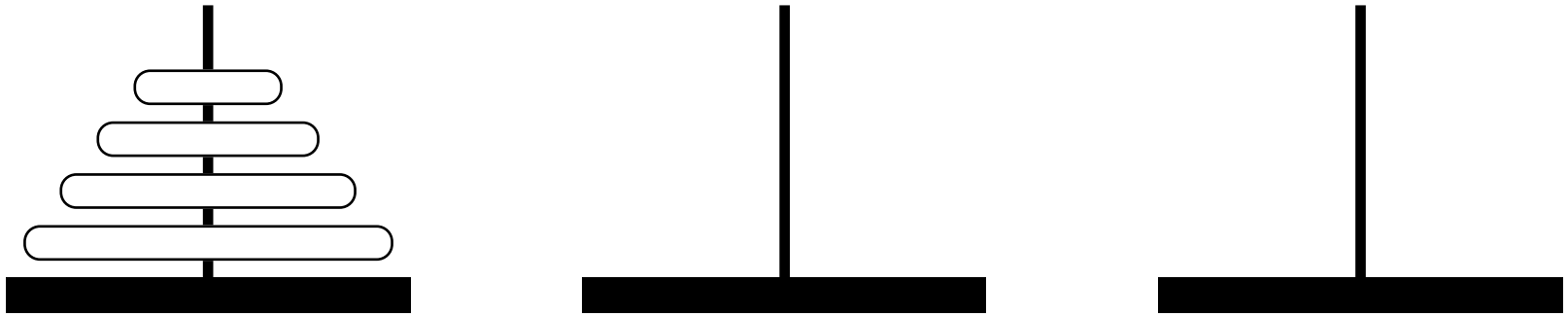
- Wiederholung der bisher eingeführten Konzepte anhand neuer Beispiele
- Nützliche Programmiertechniken mit Listen

Ausserordentliches Tutorat

Weil viele nicht ins Tutorat kommen können, findet einmalig ein ausserordentliches Tutorat statt.

- Mittwoch, 16. Dezember 1998, 10.15 – 12.00 Uhr
- Zimmer U1, Rämistrasse 69
- nur wenn mindestens drei Leute kommen

Die Türme von Hanoi



Eine Anzahl Scheiben soll vom linken zum mittleren Turm verschoben werden.

- Bei jedem Zug *eine* Scheibe (die oberste eines Turms) bewegen
- Zu keiner Zeit darf eine grössere Scheibe auf einer kleineren liegen

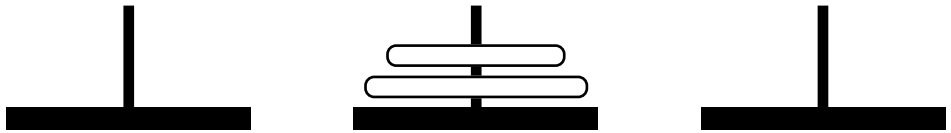
Die Türme von Hanoi

Beispiel mit zwei Scheiben

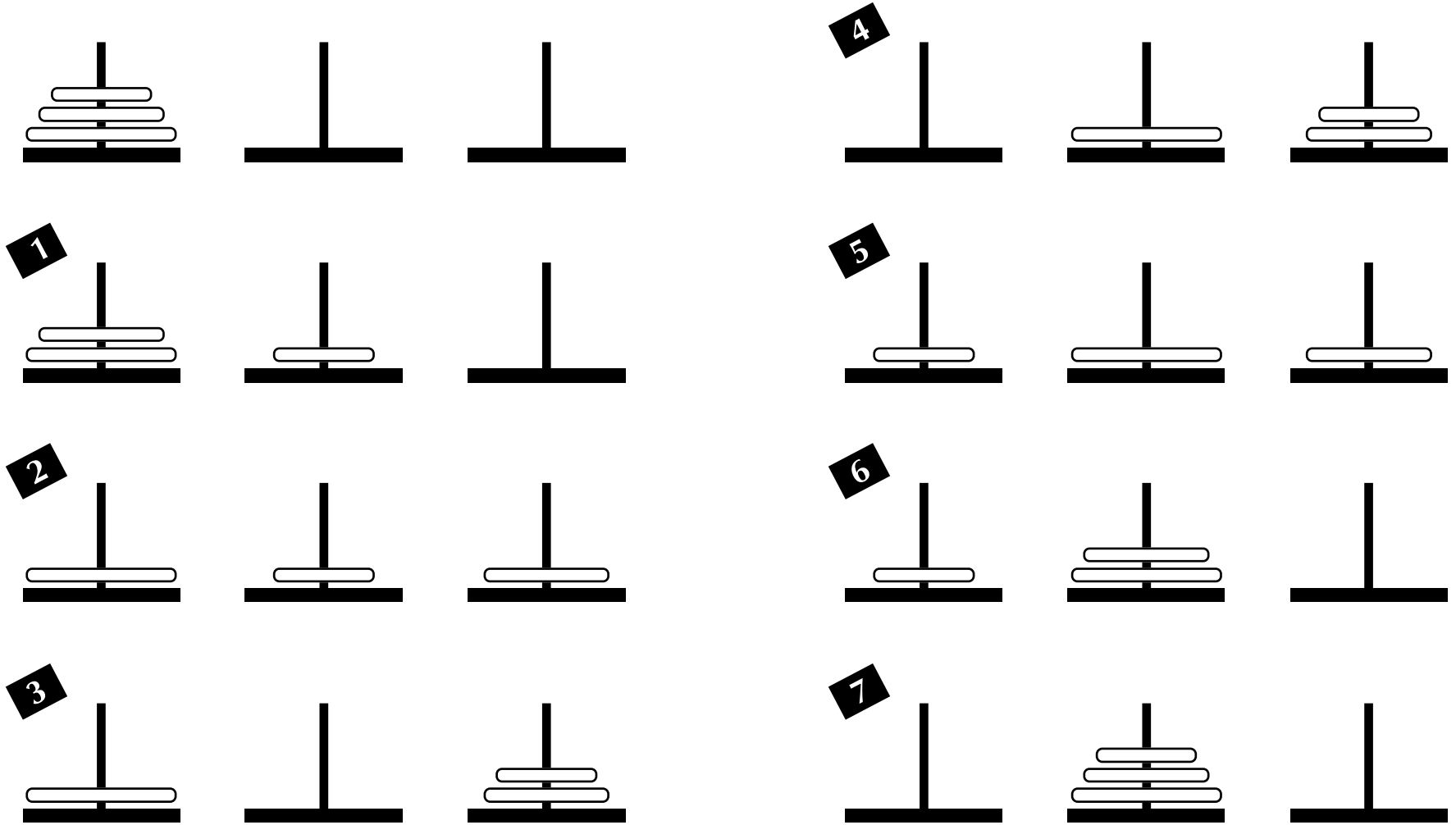
- Bewege Scheibe von *links* nach *rechts*

- Bewege Scheibe von *links* nach *mitte*

- Bewege Scheibe von *rechts* nach *mitte*



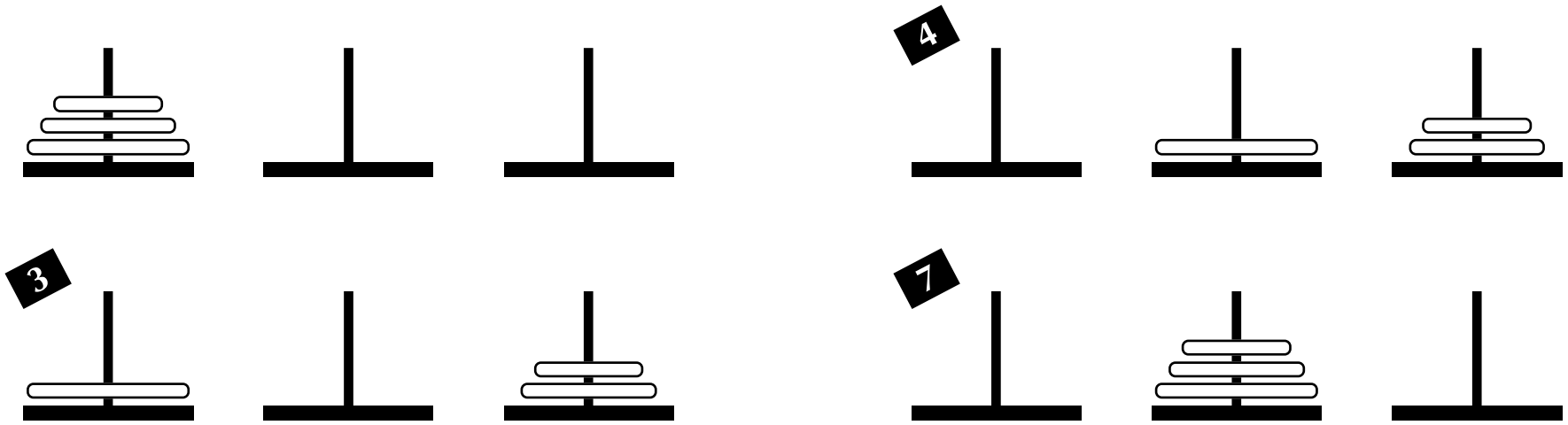
Die Türme von Hanoi



Die Türme von Hanoi: Vorgehen

Es gibt eine einfache rekursive Strategie, um *N* Scheiben von *Start* nach *Ziel* zu bewegen, mit *C* als Hilfs-Turm:

- bewege *N* – 1 Scheiben von *Start* nach *C*
- bewege die letzte Scheibe von *Start* nach *Ziel*
- bewege *N* – 1 Scheiben von *C* nach *Ziel*



Die Türme von Hanoi in Prolog

```
% hanoi(+N) gibt die Züge für Türme-von-Hanoi-Spiele mit
% N Scheiben aus. Es ruft das untenstehende verschiebe/4.
hanoi(AnzahlScheiben) :-
  verschiebe(AnzahlScheiben, links, mitte, rechts).

% verschiebe/4 - Abbruchbedingung
verschiebe(0, _, _, _).

% verschiebe/4 - Rekursiver Fall
verschiebe(N, Start, Ziel, C) :-
  M is N - 1,
  verschiebe(M, Start, C, Ziel),
  eineScheibe(Start, Ziel),
  verschiebe(M, C, Ziel, Start).

% Eine einzelne Scheibe "verschieben": Ausgabe auf Schirm
eineScheibe(Von, Nach) :-
  write(['Bewege', 'Scheibe', von, Von, nach, Nach]),
  nl.
```

Listen verketten

[a, b, c] + [d, e] = [a, b, c, d, e]

Das Verketten zweier Listen wird sehr häufig gebraucht.

- Hierzu definieren wir uns ein Prolog-Prädikat `append/3`
  - Natürlich könnte es auch anders benannt werden
- Beispiel-Anfragen:
  - ```
    ?- append([a, b, c], [d, e], Ergebnis).
    Ergebnis = [a, b, c, d, e]
    ```
  - ```
    ?- append(X, [c, d, e], [a, b, c, d, e]).
    X = [a, b]
    ```

Listen verketten

Wie so oft gibt es eine Abbruchbedingung und einen rekursiven Fall.

Abbruch

```
append([], L, L).
```

Rek. Fall

```
append([X|L1], L2, [X|L3]) :-
   append(L1, L2, L3).
```

Listen verketten

Abbruchbedingung

- Verkettet man die leere Liste und irgendeine Liste *L*, ist das Ergebnis *L*

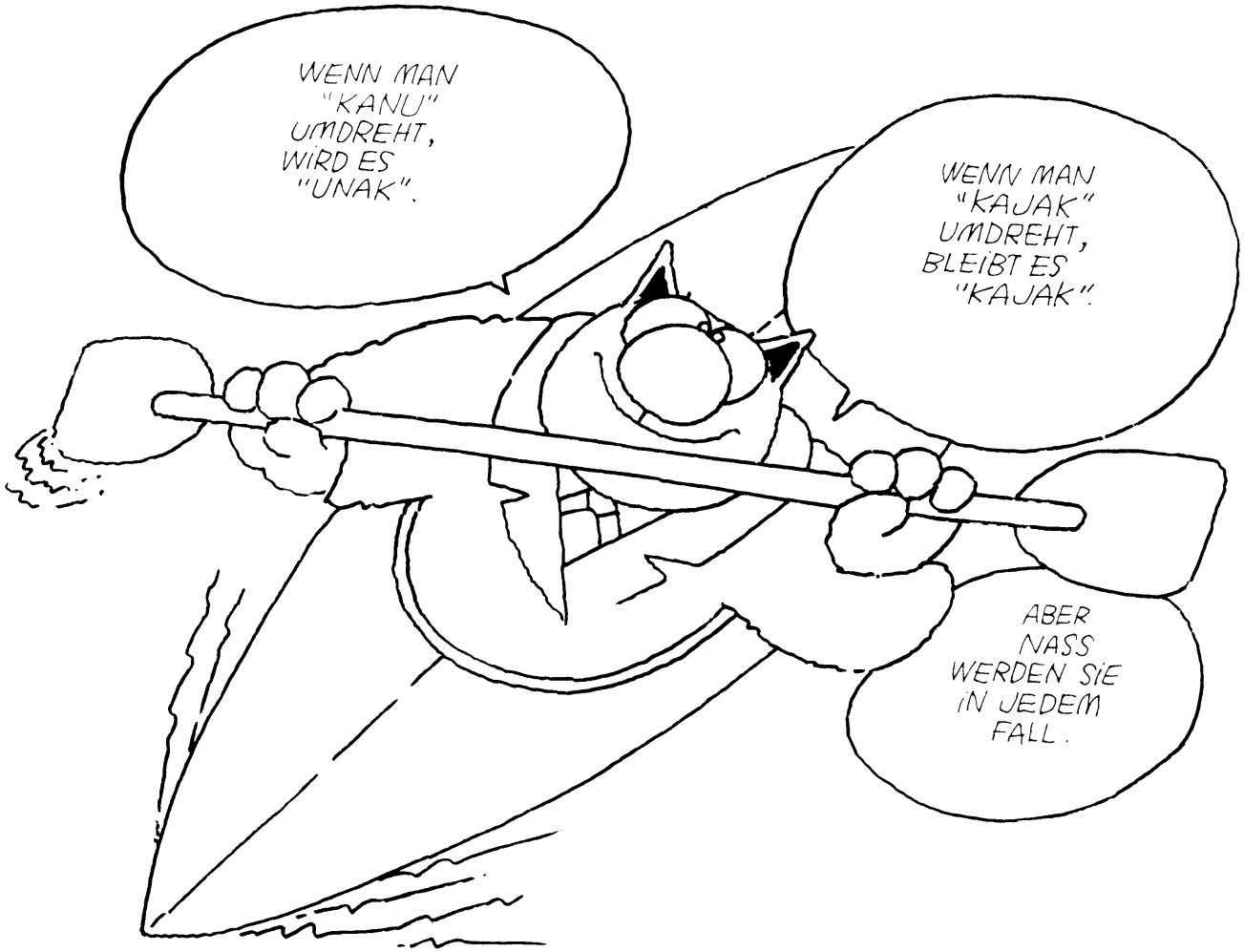
```
append([], L, L).
```

Rekursiver Fall

- Erstes Element der dritten Liste *L3* = erstes Element *X* der ersten Liste *L1*
- Rest der dritten Liste *L3* = Rest von *L1*, verkettet mit *L2*

```
append([X|L1], L2, [X|L3]) :-
    append(L1, L2, L3).
```

Listen umkehren



Aus: Philippe Geluck: Ich Sokrates [Originaltitel: »Le Chat«].
Edition Olms, Zürich 1991. ISBN 3-283-00247-9.

Listen umkehren

[k, a, n, u] → [u, n, a, k]

Gelegentlich ist es nötig, eine Liste umzudrehen.

- Verfahren 1: »naives Umkehren«
  - nachfolgend vorgestellt
- Verfahren 2: schneller, dafür etwas komplizierter zu verstehen
  - verwendet eine bestimmte Prolog-Programmiertechnik (»Akkumulatoren«), die erst später erläutert werden wird

Listen umkehren: »Naive« Version

Wie so oft gibt es eine Abbruchbedingung und einen rekursiven Fall.

Abbruch

```
reverse([], []).
```

Rekursiver Fall

```
reverse([Anfang|Rest], Ergebnis) :-
  reverse(Rest, UmgekehrterRest),
  append(UmgekehrterRest, [Anfang],
    Ergebnis).
```

Listen umkehren: »Naive« Version

Abbruchbedingung

- Die Umkehrung von `[]` ist `[]`

```
reverse([], []).
```

Rekursiver Fall

- Die Umkehrung von `[Anfang|Rest]` ist die Verkettung von
  - Umkehrung von `Rest`
  - `[Anfang]`

```
reverse([Anfang|Rest], Ergebnis) :-
    reverse(Rest, UmgekehrterRest),
    append(UmgekehrterRest, [Anfang], Ergebnis).
```