Beispiel II: Sortieren

- Sortieren: $sort(a, a_0) = perm(a, a_0) \wedge ordered(a_0)$
- Sei $occ(a, x) = card(\{j \mid 0 \leq j < n \Longrightarrow a[j] = x\})$, dann

$$perm(a, a_0) = \forall i. 0 \leq i < n \Longrightarrow \begin{array}{l} occ(a, a[i]) = occ(a_0, a[i]) \wedge \\ occ(a, a_0[i]) = occ(a_0, a_0[i]) \end{array}$$

- Permutation is *transitiv* und *symmetrisch*
- Invariante: *Vertauschen* erhält *Permutation*
- Einfaches Beispiel: Dijkstra's *Dutch National Flag*

Zusammenfassung

- *Programmkonstruktion:* Programm zusammen mit Korrektheitsbeweis entwickeln
- *Rückwärtsbeweis* mit den Regeln des *Hoare-Kalküls*
- *Einfach*: *Sequenzierung*, *Fallunterscheidung*, *Zuweisung*
- *Schwierig*: *Schleifen*
  - Zerlegung in *Invariante* und *Schleifenbedingung*
- *Beispiele*: arithemtische Probleme, Suche, Sortieren
- *Nächste Woche*: fortgeschrittene Techniken — mehr Sortieren

Vorlesung vom 11.07.06: Programmkonstruktion II

Wo sind wir?

- Hoare-Kalkül
- Jenseits von Hoare
- *Programmentwicklung*
  - Programmkonstruktion
  - *Programmentwicklung durch Transformation, Algorithmendesign*

Programmentwicklung durch Transformation

- Von der *Spezifikation* zum *Code*:

$$SP_1 \rightsquigarrow SP_2 \rightsquigarrow \ldots \rightsquigarrow SP_n$$

- Einzelne *Schritte*:
- *Modellverfeinerung* (letztes Semester)
- *Funktionen* implementieren:
  Rekursion $\longrightarrow$ *Endrekursion* $\longrightarrow$ Iteration (letztes Semester)
- Programmentwurfstransformationen
  - Einführung von Rekursion (heute)

Programmentwurfstransformationen

- Idee: *Strukturierung* der *Spezifikation*
  - *Zerlegung*
- Beispieltransformationen:
  - Divide & Conquer
  - Split of Postcondition
  - Global Search

Divide and Conquer

- Ziel: Funktion `f :: D_F ⇒ R_F` implementieren
- Typen: `D_F`, `R_F`, `D_G`, and `R_G`
- Spezifikation als *Vor-* und *Nachbedingung*

```
F_pre        :: "D_F ⇒ bool"
F_post       :: "D_F ⇒ R_F ⇒ bool"
```

- Formalisierung als *parametrisierte Spezifikation*
  - Parameter: Typen, Operationen, Axiome

Divide and Conquer

- Operationen:

```
Measure      :: "D_F ⇒ W"
Primitive    :: "D_F ⇒ bool"
Dir_Solve    :: "D_F ⇒ R_F"
Decompose    :: "D_F ⇒ (D_G × D_F × D_F)"
Compose      :: "(R_G × R_F × R_F) ⇒ R_F"
O_COMPOSE    :: "(R_F × R_G × R_F × R_F) ⇒ bool"
O_DECOMPOSE  :: "(D_F × D_G × D_F × D_F) ⇒ bool"
G            :: "D_G ⇒ R_G"
G_pre        :: "D_G ⇒ bool"
G_post       :: "D_G ⇒ R_G ⇒ bool"
W_r         :: "(W × W) set"
```

Generische Implementation

- Generische *Implementation* von `F`:

```
consts
  F :: "D_F \<Rightarrow> R_F"
recdef F "inv_image W_r Measure"
 "F x = (if (Primitive x) then (Dir_Solve x)
        else (Compose o (f_product G F F) o Decompose) x)"
       (hints recdef_wf: A_WF)
```

- Generische *Korrektheit* von `F`:
  theorem *DaC :* **assumes** *"F_pre x"* **shows** *"F_post x (F x)"*
  - Voraussetzungen als *Axiome*
  - Beweis durch *Instantiierung*

Anwendung der Transformationsregel

- Anwendungsgebiet: *Sortieralgorithmen*
- *Quicksort*
- *Mergesort*
- *Insertsort*
  - *Insert*-Funktion weitere Anwendung

Zusammenfassung

- Programmentwicklung durch *Transformationen*
- *Programmentwurfstransformationen* formalisieren *Entwicklungsentscheidungen*
- Beispiel: *Divide-&-Conquer* für
  - Quicksort, Merge, Insertsort
- Aber: *hochspezielles* Anwendungsgebiet