Bereitstellung (*Allocation*)

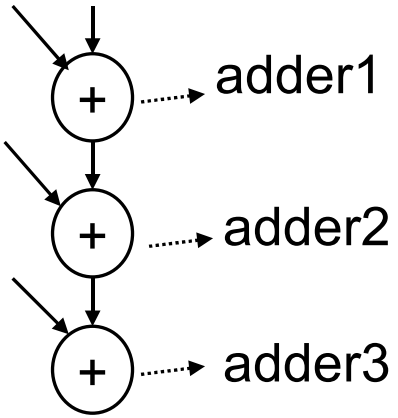
Peter Marwedel
TU Dortmund
Informatik 12

Einfache Verfahren

Methoden:

1. **Vorgabe durch den Benutzer**
   Bezieht Kenntnisse des Benutzers ein.
2. **Für jedes Vorkommen eine eigene Ressource**, als „direkte Compilation“ bezeichnet. Keinerlei „*sharing*“. Entspricht der Codeerzeugung eines Compilers: für jede Operation werden Befehle erzeugt.

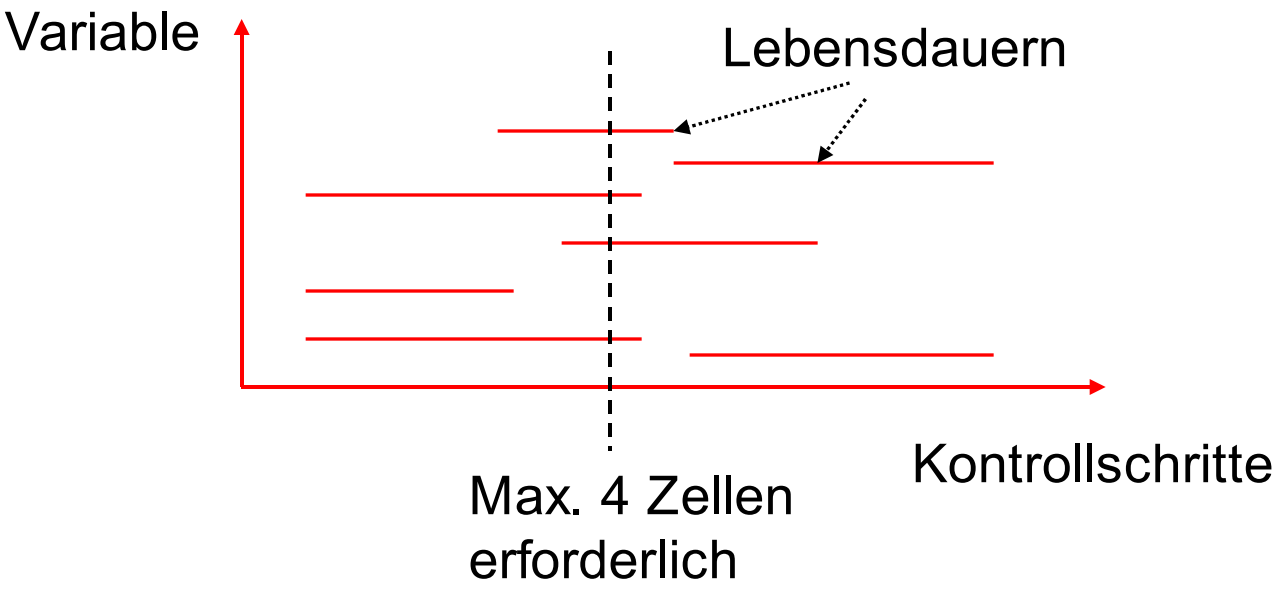
Regs: RAM16x32;



3. Mehrfachausnutzung von Ressourcen mit denselben Fähigkeiten

Minimierung der Anzahl gleichartiger Ressourcen: **Bildung des Maximums der Nutzung über Kontrollschritte**.

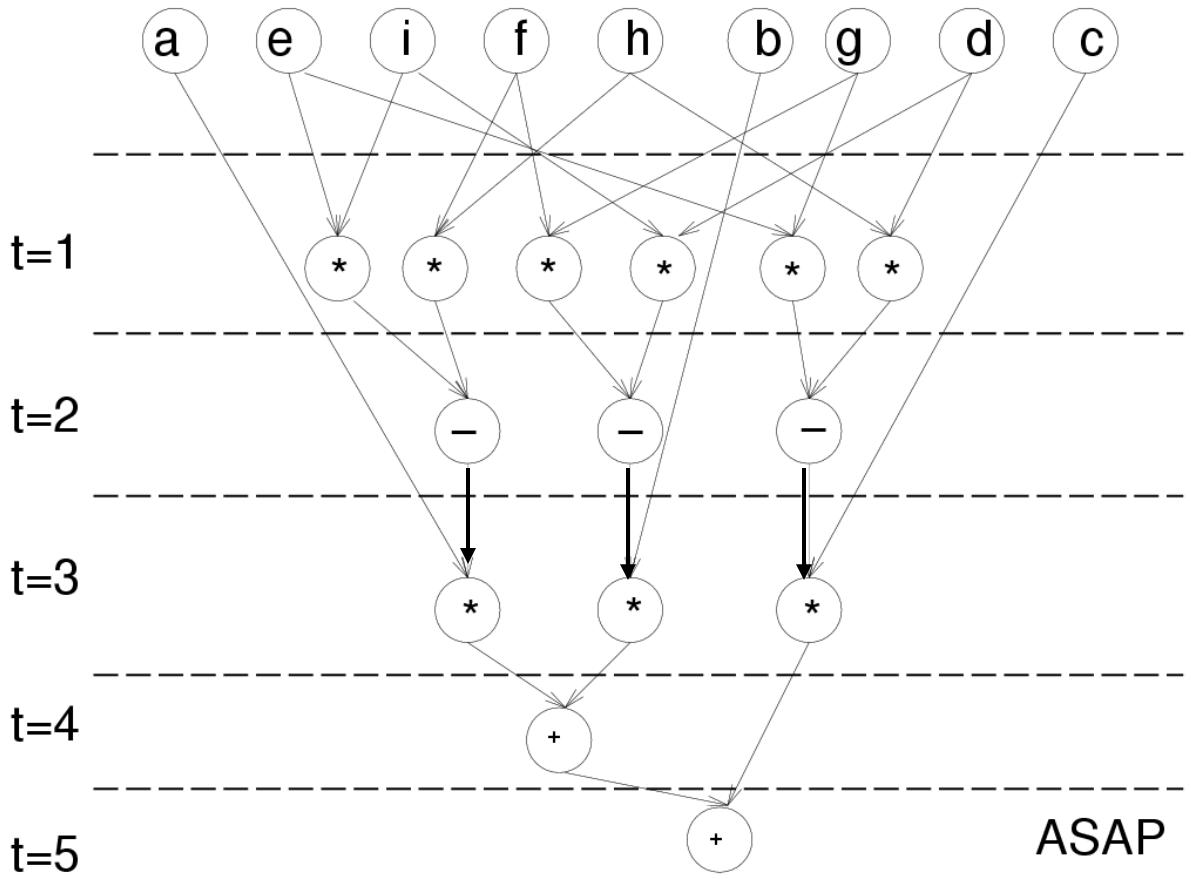
Beispiel: Minimierung der Anzahl benutzter Zellen für Variable innerhalb eines Basisblocks



Abhängigkeit vom Scheduling (1)

Im Beispiel ergeben sich für das ASAP-Schedule:

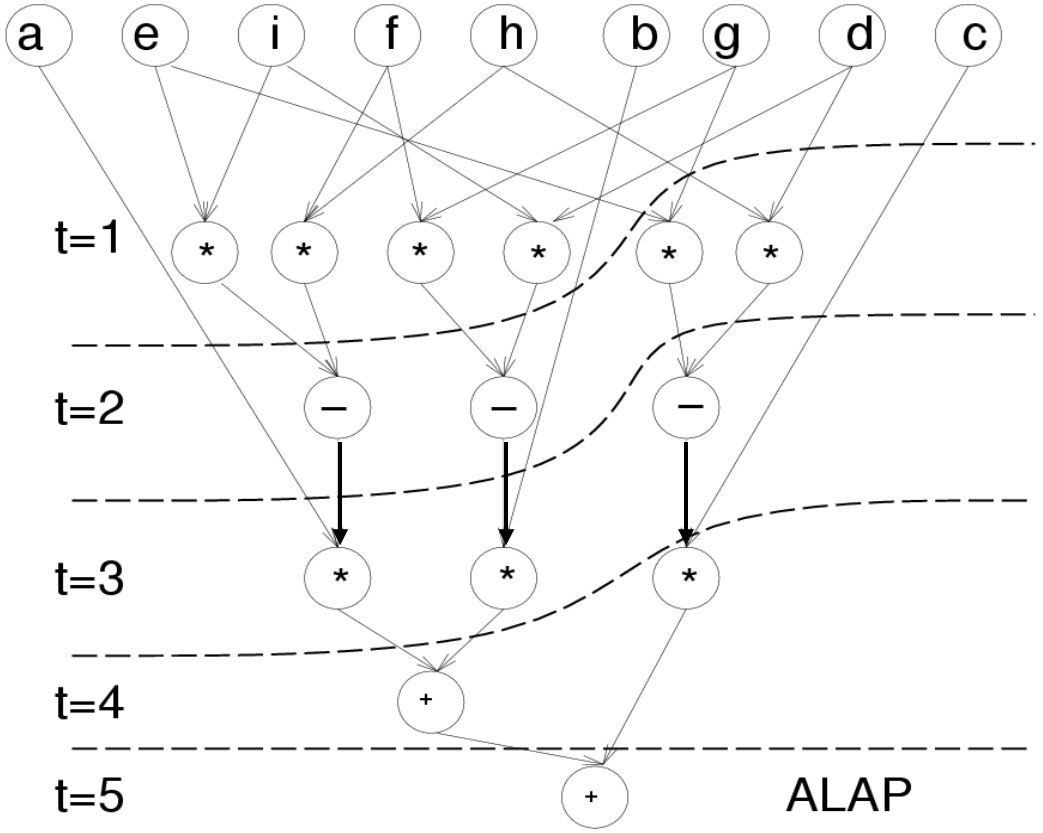
- 6 Multiplizierer
- 3 Subtrahierer
- 1 Addierer



Abhängigkeit vom Scheduling (2)

Für das ALAP-Schedule:

- 4 Multiplizierer
- 2 Subtrahierer
- 1 Addierer



☞ Abhängigkeit zwischen Scheduling und Allokation!

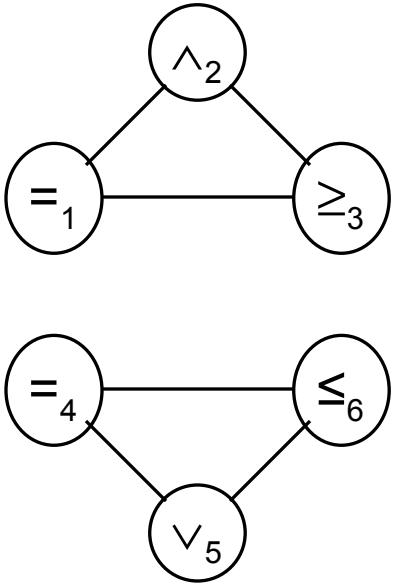
4. Konstruktion von ALUs

Gruppierung aufgrund „gleichzeitigem Vorkommen“

Beispiel (Operationen durchnummeriert):
CS1: … $(a =_1 b) \wedge_2 (c \geq_3 d)$ …

CS2: … $(e =_4 f) \vee_5 (g \leq_6 h)$ …

Relation „gleichzeitiges Vorkommen“



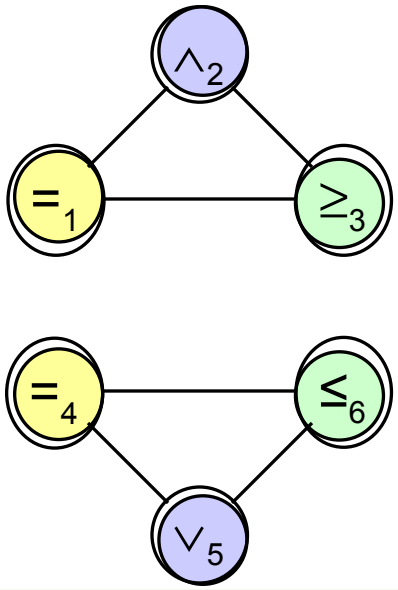
„Konfliktgraph“.

Kann grundsätzlich auch Basisblock-übergreifend erzeugt werden, z.B. kein Konflikt zwischen THEN- und ELSE-Teil.

Färbeproblem

Bestimmung einer minimalen Anzahl von ALUs durch Färben des Konfliktgraphen:

Färbeproblem: Ordne jedem Knoten des Konfliktgraphen eine Farbe zu, wobei durch Kanten verbundene Knoten eine unterschiedliche Farbe haben müssen. Wähle eine Zuordnung mit einer minimalen Anzahl von Farben!



(Die Entscheidungsvariante des) Färbeproblems ist NP-vollständig

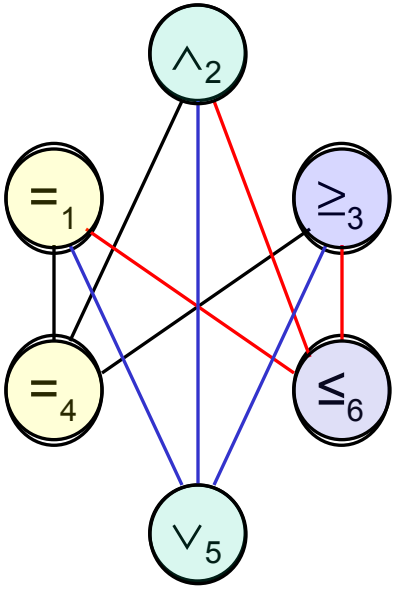
Jede Farbe entspricht einer ALU.

Modellierung als Färbeproblem ignoriert sinnvolle Gruppierung der Operationen; ok für Variable.

Cliquenproblem

Möglicher Übergang auf Kompatibilitätsgraph:
Kante, wenn Operationen kompatibel sind.

„kein gleichzeitiges Vorkommen“

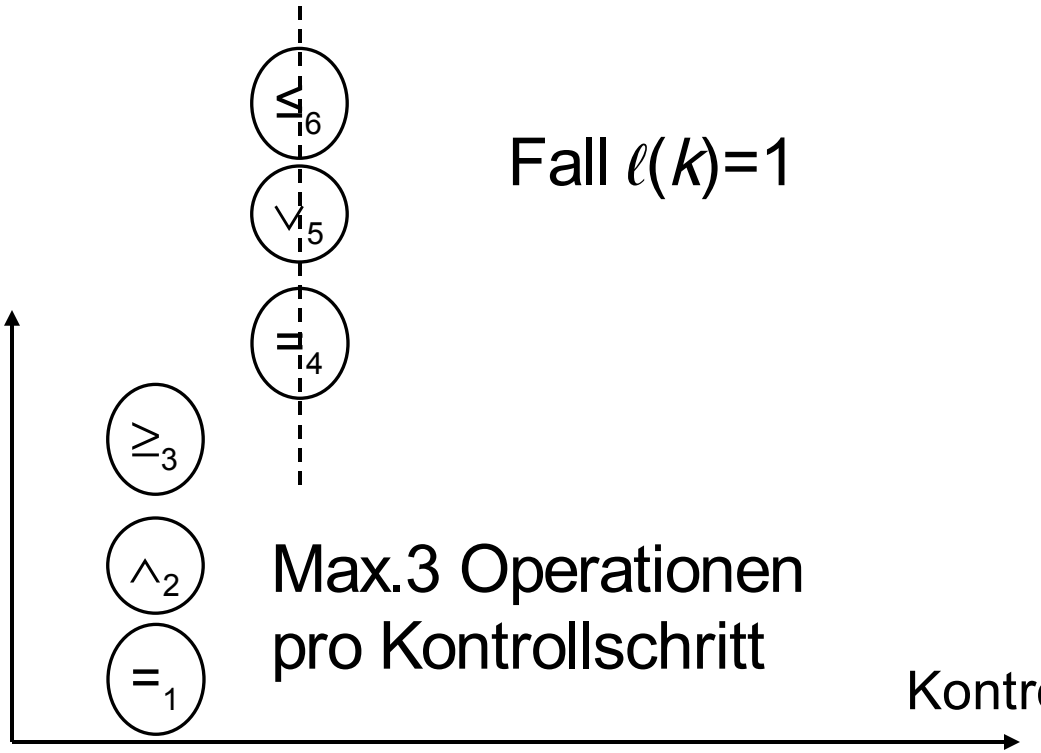


Minimierung der Anzahl der ALUs entspricht jetzt einem Cliquenproblem:
Ein Graph heißt vollständig, wenn alle seine Knoten mit Kanten verbunden sind.
Cliquenproblem: Finde eine Überdeckung der Knoten eines Graphen mit einer minimalen Anzahl vollständiger Teilgraphen.

Dieselben Nachteile wie beim Färbeproblem.

Cliquen- und Färbungsmodell bei Beschränkung auf Basisblöcke

Bei Beschränkung auf Basisblöcke reicht es aus, einfach nur die maximal notwendige Anzahl von Operationen pro Kontrollschritt der Kontrollschritt-Sequenz zu bestimmen.



Fall $\ell(k)=1$

Wenn Multizyklen-operationen vorkommen, entsteht wieder dieselbe Situation wie bei der Mehrfachausnutzung von Ressourcen mit denselben Fähigkeiten.

Zur Komplexität siehe Allokation.

5. Mehrfachausnutzung von Ressourcen mit unterschiedlichen Fähigkeiten

Annahme: Ressourcen haben unterschiedliche Fähigkeiten. Statt der Anzahl müssen die Kosten minimiert werden.

Def.:

- B_m: Anzahl der Bausteine vom Typ m (gesucht)
- $\beta(m)$: Vom Typ m bereitgestellte Operationen
- $f_{i,g}$: Häufigkeit der Operation g in Kontrollschritt i

Beispiel:

3 Bausteintypen:
$\beta(1)=\{+\}$, $\beta(2)=\{-\}$, $\beta(3)=\{+,-\}$

2 Kontrollschritte
CS1: $f_{1,+}=1$, $f_{1,-}=1$
CS2: $f_{2,+}=2$,

Welche Gleichungen werden benötigt, um eine ausreichende Anzahl von Bausteinen sicherzustellen?

Hinreichende Bedingungen

3 Bausteintypen:
$\beta(1)=\{+\}$, $\beta(2)=\{-\}$, $\beta(3)=\{+,-\}$

2 Kontrollschritte
CS1: $f_{1,+}=1$, $f_{1,-}=1$
CS2: $f_{2,+}=2$,

In CS1 müssen die Bausteine ausreichen, also:
$B_1+ \quad B_3 \geq 1$ (wegen der Zahl für {+})
$B_2+B_3 \geq 1$ (wegen der Zahl für {-})
$B_1+B_2+B_3 \geq 2$ (wegen der Gesamtzahl für {+,-})

In CS2 müssen die Bausteine ausreichen, also:
$B_1+ \quad B_3 \geq 2$ (wegen der Zahl für {+})

Dabei haben wir in jedem Kontrollschritt alle Kombinationen von Operationen betrachtet.

Zusammenfassung der Ungleichungen

Zusammenfassung über Kontrollschritte hinweg:
für jede vorkommende Kombination von Operationen die minimal erforderlich Anzahl von Bausteinen bestimmen!
Im Beispiel:

$\{+\}$ $B_1 + B_3 \geq 2$ (Zusammenfass. 1. & 4. Ungleichung)
$\{-\}$ $B_2 + B_3 \geq 1$
$\{+,-\}$ $B_1 + B_2 + B_3 \geq 2$

Allgemein: Für jede Kombination der jeweils vorkommenden Operationen muss die Gesamtzahl der Ressourcen, die mindestens eine der Operationen ausführen können, mindestens gleich der Gesamtzahl der Operationen sein.

Allgemeine Formulierung hinreichender Bedingungen (1)

Def.:

- F_i: Menge der in i ausgeführten Operationen $\{g|f_{i,g}>0\}$
- $F_i^* = \wp(F_i)$: deren Potenzmenge (ohne das leere Element)

Dann muss gelten (für alle Kontrollschritte) :

$$\forall\, i : \forall\, h \in F_i^* : \sum_{\substack{m \in M \\ h \cap \beta(m) \neq \{\}}} B_m \geq \sum_{g \in h} f_{i,g}$$

Also: für jede Kombination der jeweils vorkommenden Operationen muss die Gesamtzahl der Ressourcen, die mindestens eine der Operationen ausführen können, mindestens gleich der Gesamtzahl der Operationen sein.

Allgemeine Formulierung hinreichender Bedingungen (2)

Ungleichungen für alle Kontrollschritte zusammenfassen: Betrachtung des Maximums der Anzahl der Operationen, die für eine Kombination von Operationen vorkommen.

Sei : $F^* = \bigcup_i F_i^*$ (Menge der Kombinationen von Operationen)

sowie S_h die max. Anzahl von Vorkommen einer Kombination h :

$$\forall h \in F^* : S_h = \max_i \left(\sum_{g \in h} f_{i,g} \right)$$

Dann lassen sich die Ungleichungen zusammenfassen zu:

$$\forall h \in F^* : \sum_{m \in M} a_{h,m} B_m \geq S_h \qquad \text{mit } a_{h,m} = \begin{cases} 1, \text{falls } h \cap \beta(m) \neq \{\} \\ 0, \text{sonst} \end{cases}$$

[Marwedel, 1986]

Gesamtmodell

$$\text{Minimiere } C = \sum_{m \in M} B_m c_m$$

$$\text{für die Randbedingungen } \forall\, h \in F^* : \sum_{m \in M} a_{h,m} B_m \geq S_h$$

Problem der **Ganzzahligen (linearen) Programmierung** *(*engl. *integer (linear) programming,* I(L)P*)*-Problem in den ganzzahligen Variablen B_m.

Potenziell einzelne Ungleichungen redundant.

Anzahl von Operationstypen i.d.R. relativ klein (<10), $\rightarrow |F^*|$ klein genug, um praktikabel zu sein.

Ausnahme: Große Anzahl unterschiedlicher Bitbreiten.

Beispiel

Zusammenfassung

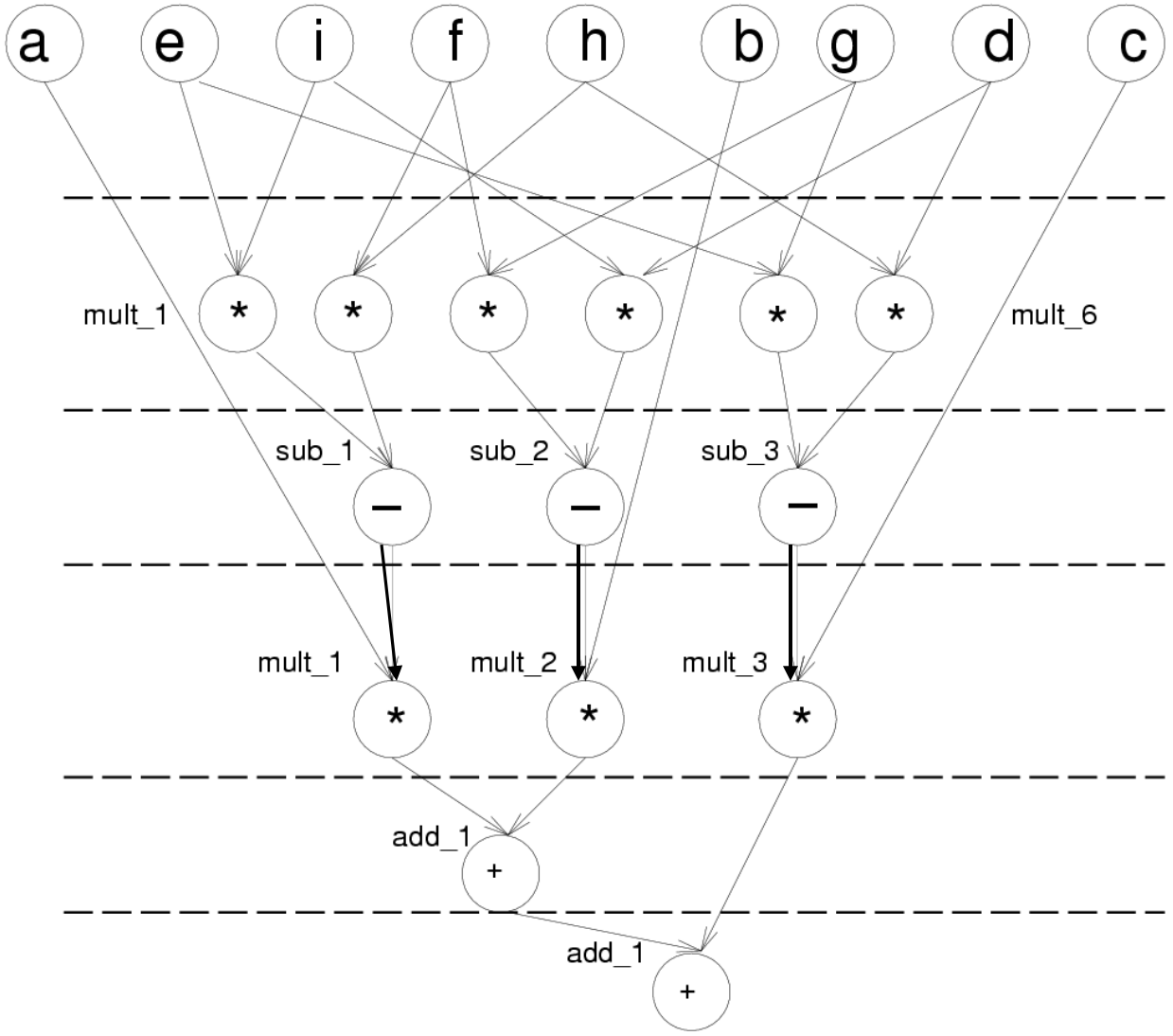
- Bereitstellung (*Allocation*)
  - Gleichartige Ressourcen
  - Färbe- und Cliquenprobleme
  - Verschiedenartige Ressourcen

Zuordnung
(*Assignment, Resource Binding*)

Peter Marwedel
TU Dortmund
Informatik 12

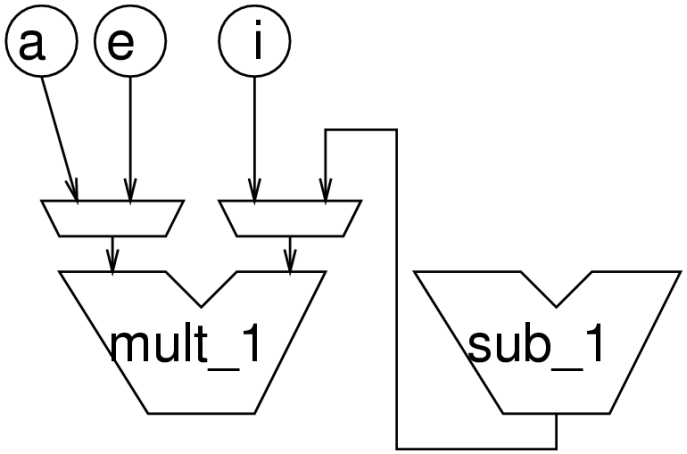
1. Durchzählen

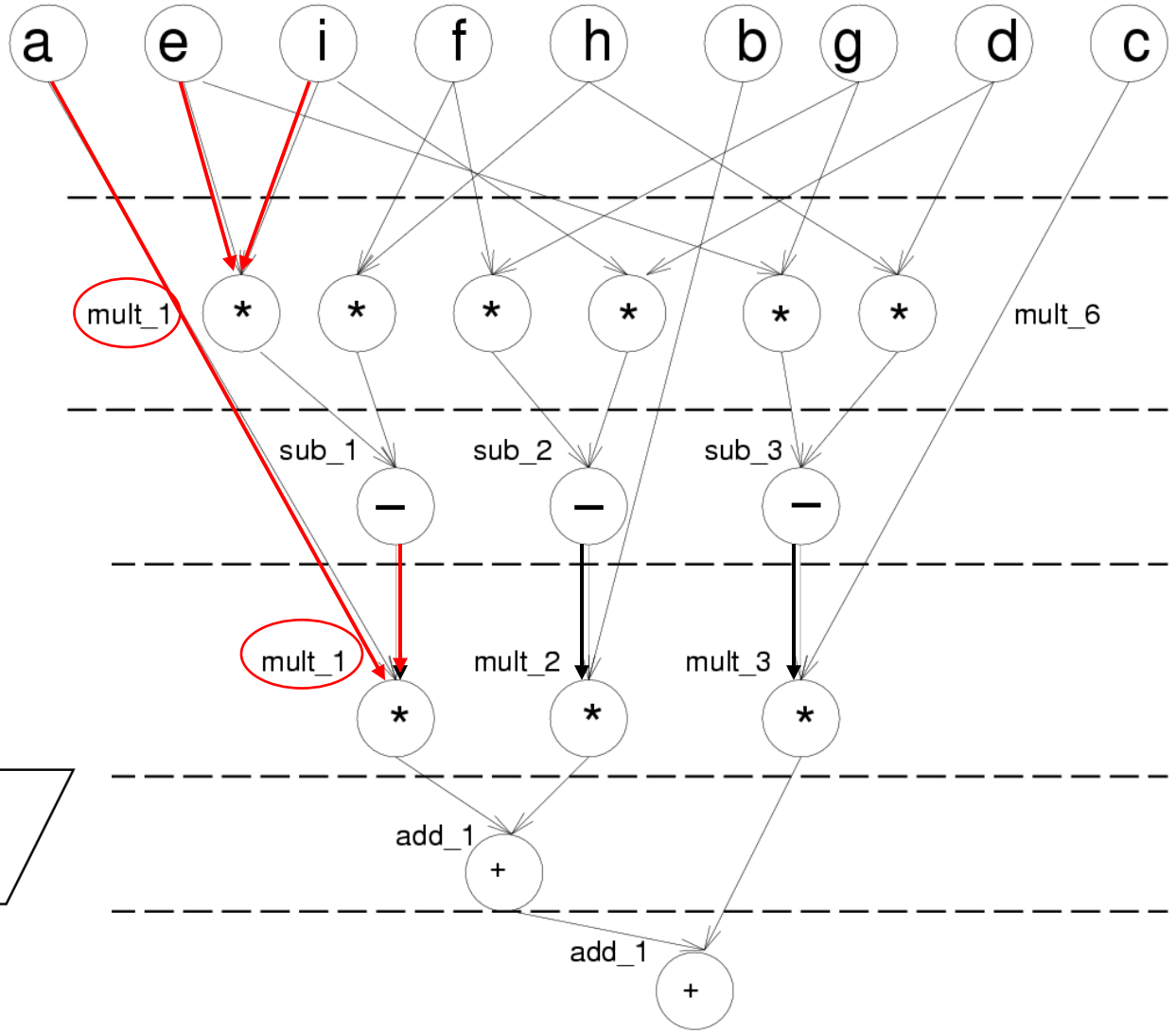
1. Die einfachste Zuordnung besteht in dem simplen Durchzählen innerhalb der Kontroll-schritte



Zuordnung bestimmt implizit die Verbindungsstruktur

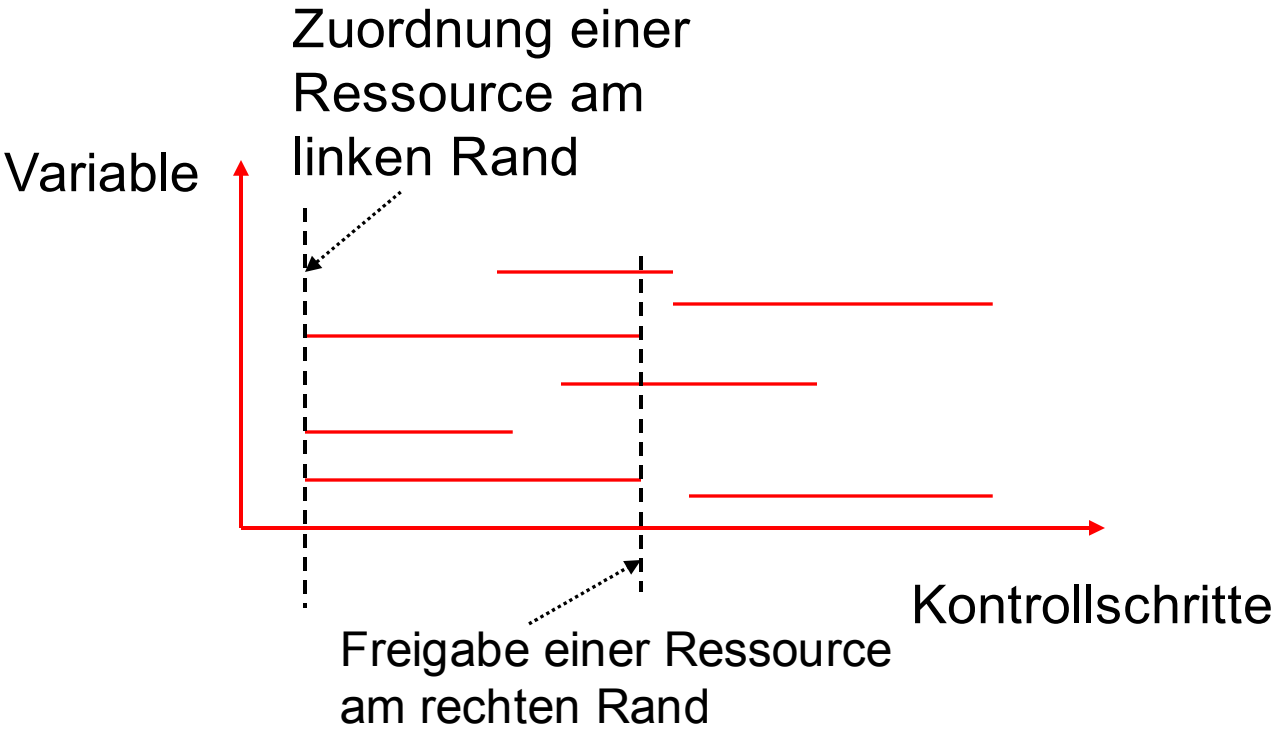
Beispiel: mult_1 muss verbunden sein mit den „Registern“ e und i sowie mit sub_1 und a.





Zuordnung von Ressourcen mit denselben Fähigkeiten

Erweiterung des Verfahrens zur Allokation:
„*sweep*“ von links nach rechts, Zuordnung & Freigabe jeweils am Rand eines belegten Intervalls



Präziser: für jeden Kontroll-schritt Mengen der linken & rechten Grenzen in *first*[] bzw. *last*[] eingetragen.

☞

Abb.: left-edge-Algorithmus

Der „*left-edge*“-Algorithmus

```
Free:={1..kmax}; (* Alle Ressourcen (Zellen) sind frei*)
for i:=1 to imax do (* Schleife über Kontrollschritte*)
  begin
    for each j ∈ first[i] do
      begin
        k:= kleinstes Element aus Free;
        assign[j]:= k; (*gesuchte Zuordnung*)
        Free:=Free – {k};
      end;
    for each j ∈ last[i] do Free:=Free∪{assign[j]};
end;
```

Dieser Algorithmus wird als linear bezeichnet
(setzt Implementierung innerer Schleifen in O(1) voraus).

Anwendung in der Synthese von Fadi Kurdahi vorgeschlagen

Intervallgraphen (1)

Left-edge-Algorithmus löst das Färbungsproblem eines aus Abb. „*left-edge*-Algorithmus“ erzeugten Konfliktgraphen.

Effiziente Lösung des Färbungsproblems gelingt aufgrund der speziellen Struktur des Graphen.

Def.: Ein Graph heißt **Intervallgraph**, falls es eine umkehrbar eindeutige Abbildung seiner Knoten auf eine linear geordnete Menge von Intervallen existiert, so dass zwei Knoten genau dann mit einer Kante verbunden sind, wenn sich die Intervalle schneiden.

Intervallgraphen (2)

Satz [s. Golombic]:

Intervallgraphen lassen sich in linearer Zeit färben.

(Geht mit dem „*left edge* Algorithmus“)

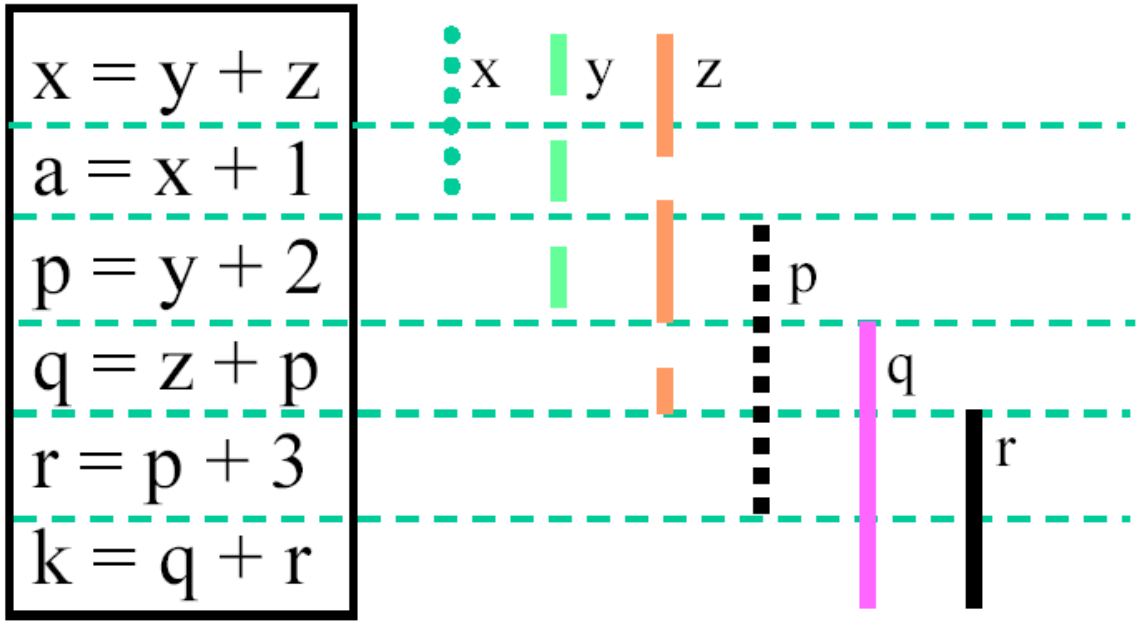
Die aus der Abbildung erzeugten Konfliktgraphen sind per Konstruktion Intervallgraphen. Die Intervalle sind gerade diejenigen der Abbildung „*left-edge*-Algorithmus“.

Life-time of Variables

Beispiel

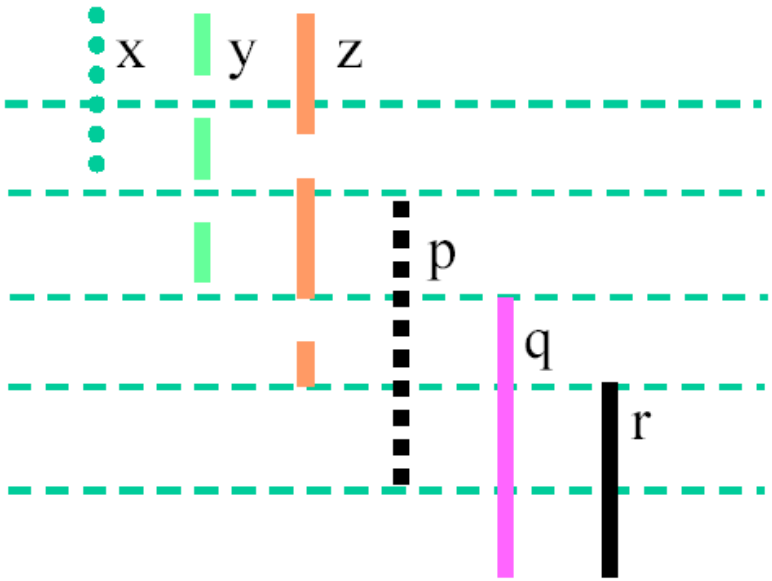
Register optimization technique:

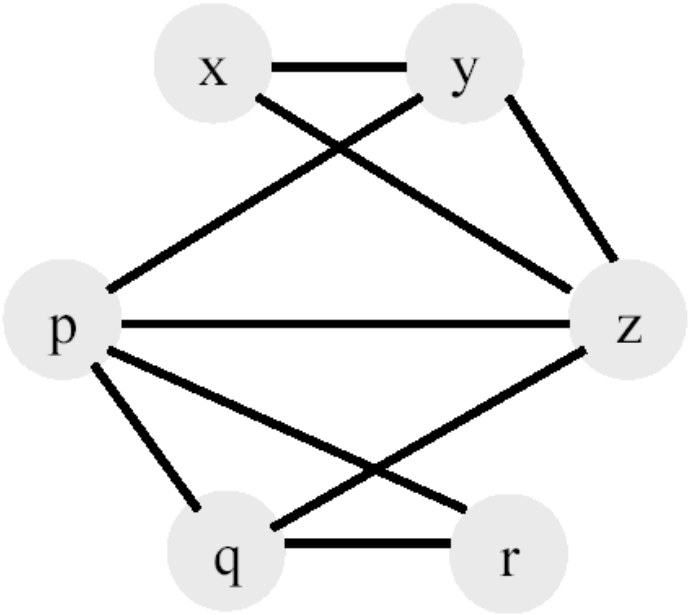
Life-time: definition to last use of variable



Conflict Graph of Life-times

Beispiel



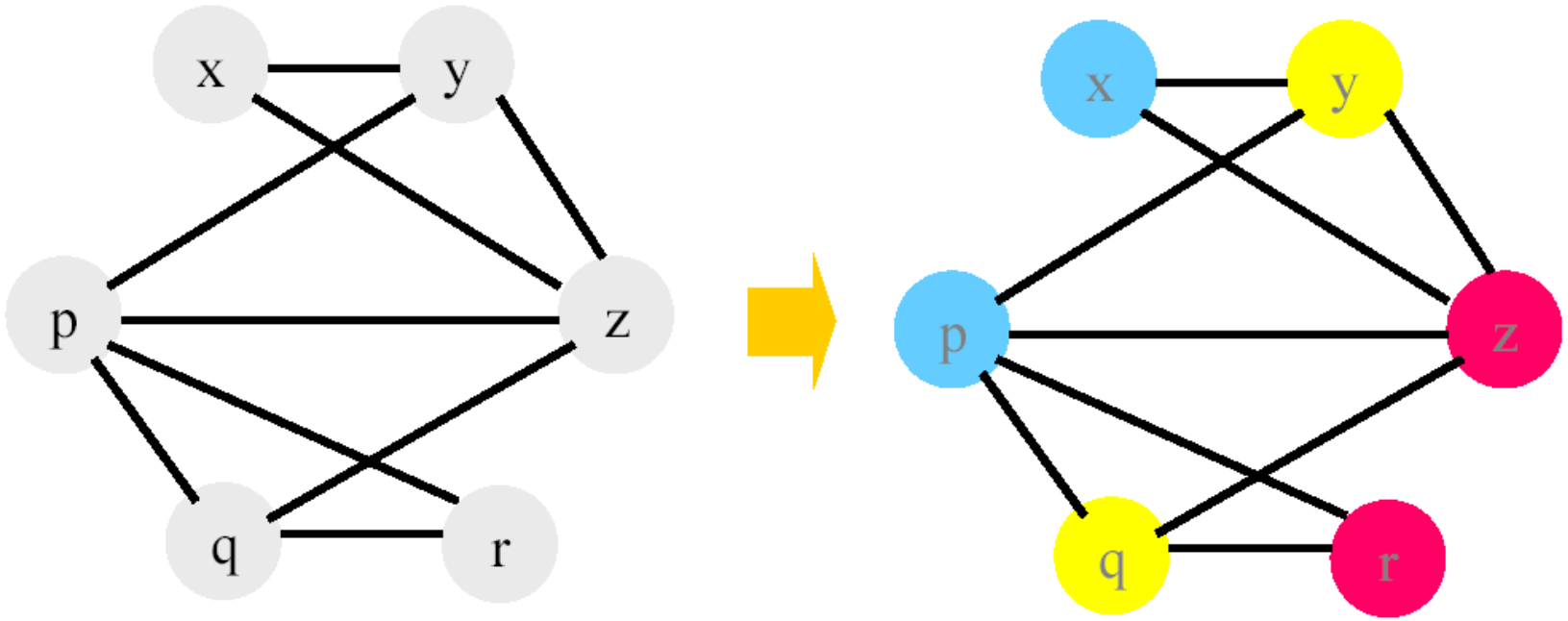


24

Coloring the Conflict Graph

Beispiel

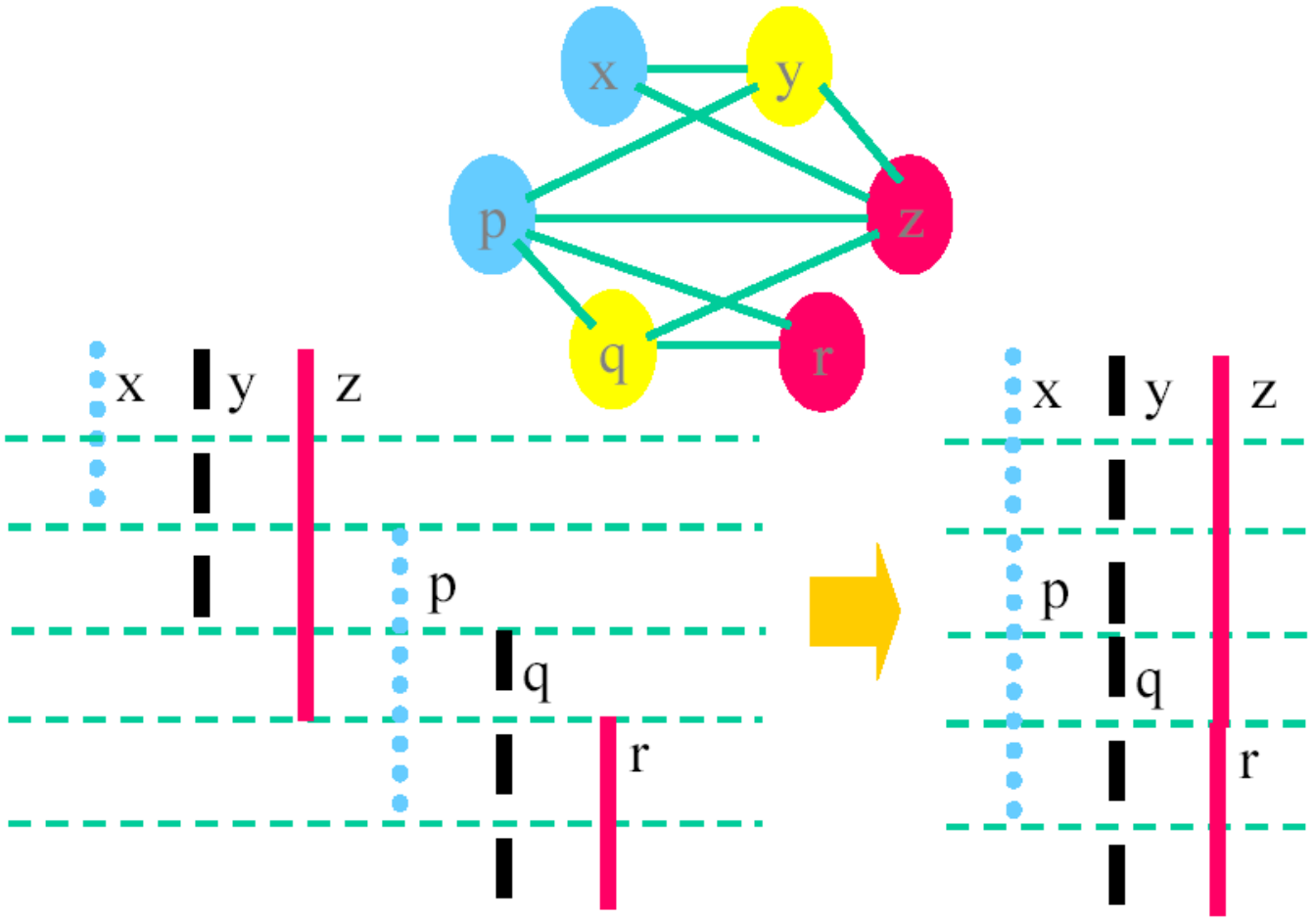
Minimum number of registers = Chromatic number of conflict graph



25

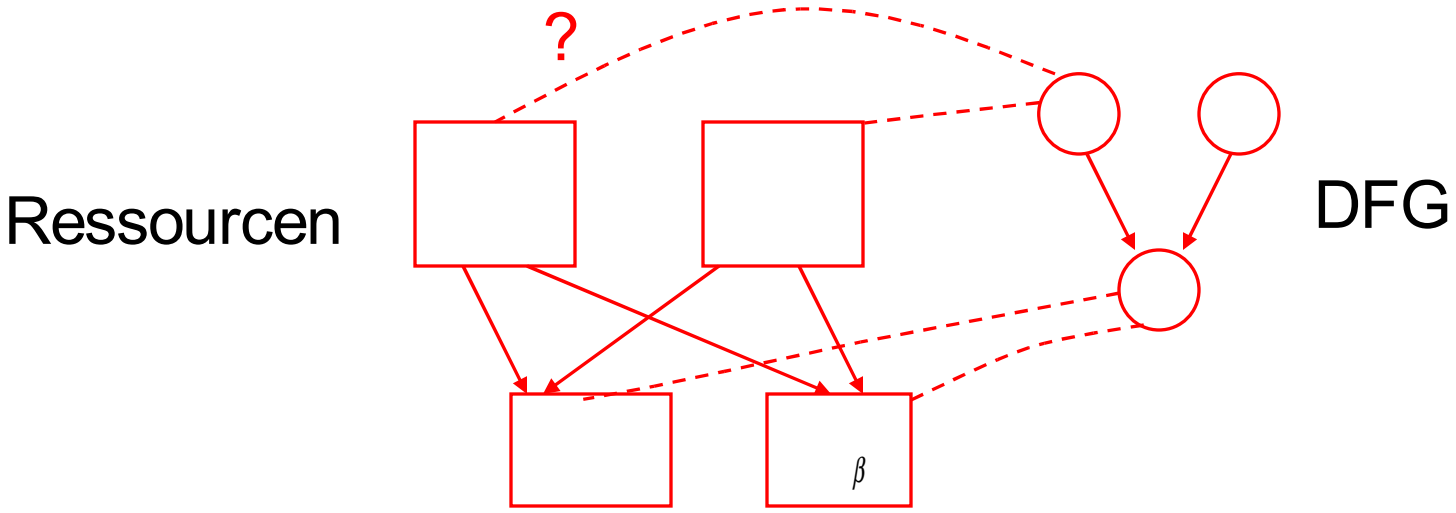
Coloring determines Register Allocation

Beispiel



26

3. Modellierung von Verbindungskosten in der Zuordnung



Sei

K: Menge der Ressourcen

J: Menge der Operationen

$c_{i,j}$: Verbindungskosten von Ressource *i* nach Ressource *j*

$x_{j,k}$=1 falls die Operation *j* Ressource *k* zugeordnet wird,

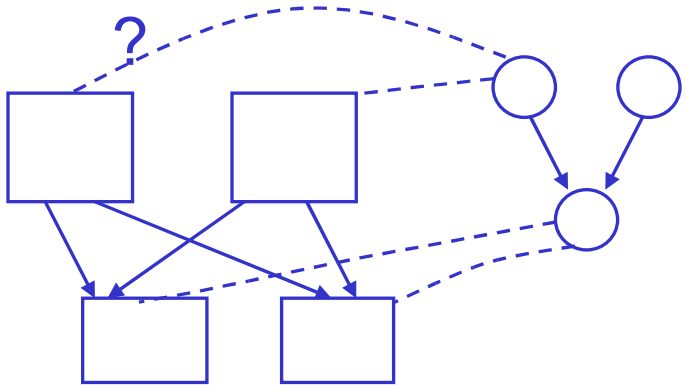
=0 sonst

$\beta_{j,k}$=1, falls Ressource *k* Operation *j* ausführen kann, 0 sonst.

Modellierung als Quadratisches Zuordnungsproblem

Verbindungskosten:

$$C = \sum_{\substack{j,j' \in J \\ k,k' \in K}} a_{j,j'} c_{k,k'} x_{j,k} x_{j'k'}$$



Quadratisches Zuordnungsproblem (QAP)

Sei $\forall j: p(j)=k \Leftrightarrow x_{j,k}=1$ (Permutation).

$\rightarrow$

$$C = \sum_{j,j' \in J} a_{j,j'} c_{p(j),p(j')}$$

Randbedingungen:

$$\forall j \in J : \sum_{k \in K} \beta_{j,k} x_{j,k} = 1$$

Innerhalb eines Kontrollschritts:

$$\forall k \in K : \sum_{j \in J} x_{j,k} \leq 1$$

Quadratisches Zuordnungsproblem (QAP)

- QAP modelliert viele Zuweisungsprobleme, z.B. die Zuordnung von Fertigungsstraßen bei Abhängigkeiten der Fertigungsstraßen voneinander.
- Viele Optimierungspakete für die mathematische Optimierung enthalten Lösungsverfahren für das QAP.
- Grundsätzlich lässt sich das QAP linearisieren: Definiere $y_{j,j',k,k'} = x_{j,k}\, x_{j'k'}\ \forall j,j',k,k'$. Allerdings wächst dadurch die Anzahl der Variablen.
- Das QAP ist NP-hart (folgt aus den entsprechenden Aussagen für die Ganzzahlige Programmierung.
- Für die Mikroarchitektursynthese i.d.R. zu komplex
- Quellen: z.B: http://www.opt.math.tu-graz.ac.at/qaplib/

Zusammenfassung

- Zuordnung (*Assignment, Resource Binding*)
  - *Left-edge* Algorithmus
  - Intervallgraphen
  - Quadratisches Zuordnungsproblem (QAP)