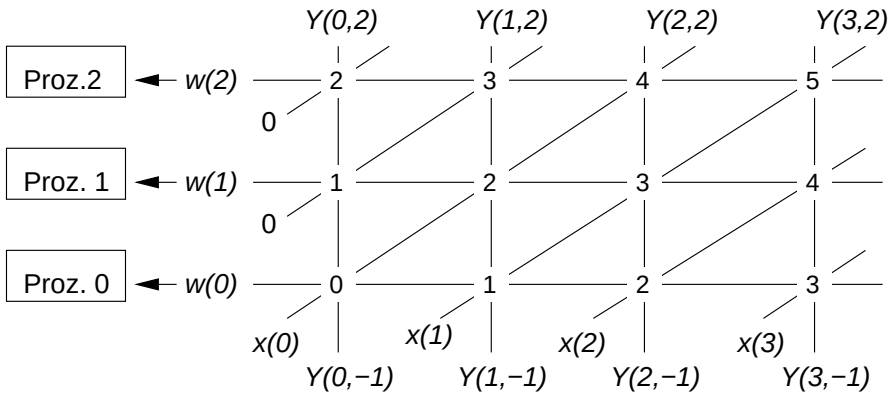

Abbildung 2.4: Projektion auf Prozessoren

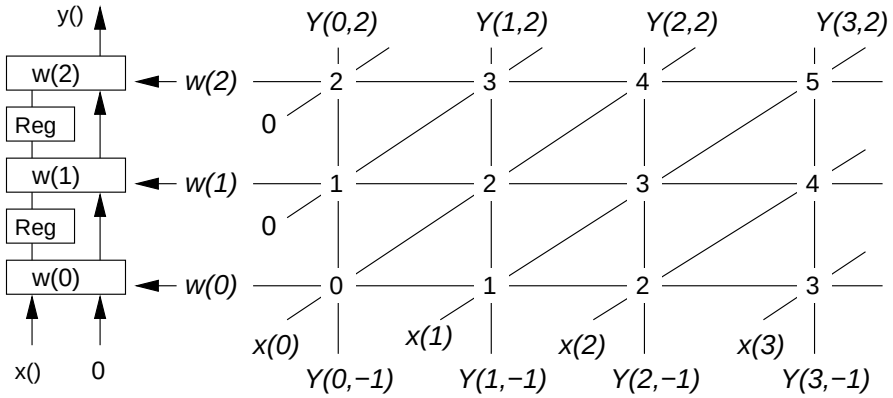

Abbildung 2.5: Eindimensionales Prozessorfeld für die Faltung

Systolische und befehlssystolische Felder wurden in den 80er Jahren intensiv untersucht. Über die hier angesprochene Art der Synthese gibt es zahlreiche Publikationen (z. B. [LM85], [Mol83], [Ull84]). Nach einer Zeit der relativen Ruhe sind sie durch die Verfügbarkeit von FPGAs wieder sehr interessant geworden.

Befehlsystolische Felder (engl. *instruction systolic arrays*) sind eine Erweiterung von systolischen Feldern, bei der Befehle durch das Prozessorfeld hindurch geschoben werden können [KLS+88]. Damit ist eine größere Flexibilität möglich.

2.2 Synthese aus imperativen Spezifikationen

2.2.1 Ziel der Synthese

Die Synthese auf der Basis **imperativer** Spezifikationen wurde zuerst an der Universität Kiel (Zimmermann et al.[Zim79, Mar79]) und der Carnegie Mellon Universität in Pittsburgh (Barbacci, Thomas, Parker[Par84]) entwickelt.

Im Folgenden beschränken wir uns zunächst auf das Rechenwerk (Steuerwerke werden später behandelt). Eine gründliche Analyse der Anforderungen ist v.a. bei berechnungsintensiven Spezifikationen erforderlich.

Berechnungen (ohne Sprünge) können mit Datenfluss-Graphen dargestellt werden.

Beispiel: Berechnung von Determinanten.

Die Determinante von

$$\begin{vmatrix} a & b & c \\ d & e & f \\ g & h & i \end{vmatrix}$$

kann durch die Formel

$$d = a * (e * i - f * h) + b * (f * g - d * i) + c * (d * h - e * g)$$

beschrieben werden. In VHDL wird die Formel meist in Form von einfachen Variablen- oder Signalzuweisungen notiert:

```
h1 := e * i;
h2 := f * h;
h3 := f * g;
h4 := d * i;
....
```

Def.: Ein **Basisblock** ist eine (maximale) Codesequenz, die eine Verzweigung höchstens an ihrem Ende und mehrere Vorgänger (im Sinne eines Verschmelzens von Kontrollflüssen) höchstens an ihrem Anfang besitzt.

Bei Beschränkung auf Basisblöcke können die Anweisungen zu **Datenfluss-Graphen** (DFGs) zusammengefasst werden.

Der Datenfluss-Graph für die Determinantenberechnung sieht wie folgt aus:

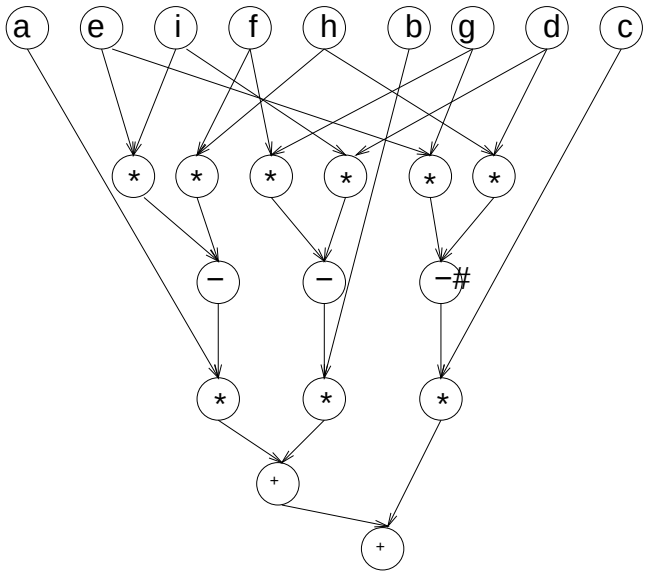

Abbildung 2.6: Datenfluss-Graph (-# = - mit vertauschten Eingängen)

Kommen in der Spezifikation auch Verzweigungen vor, so verwendet man einen separaten **Kontrollfluss-Graphen** zur Darstellung der Programm-Kontrolle. In Kombination mit Datenfluss-Graphen und Verweisen zwischen diesen kommt man zu **Kontroll/Datenfluss-Graphen** (CDFGs).

Beispiel:

Zur Spezifikation `IF bed. THEN <statements-1> ELSE <statements-1>` gehört der CDFG der Abb. 2.7.

Die Aufgabe der Mikroarchitektur-Synthese besteht darin, zu einem vorgegebenen CDFG eine Hardware-Implementierung zu erzeugen, die vorgegebene Randbedingungen (wie z.B. minimale Rechenleistung, maximaler Stromverbrauch, Ausnutzung von Bibliothekskomponenten usw.) einhält.

Man erwartet von der Mikroarchitektur-Synthese üblicherweise, dass drei Aufgaben gelöst werden:

- Das *Scheduling* (deutsch „**Ablaufplanung**“). Hierdurch wird für jede Operation festgelegt, **wann** sie ausgeführt wird.
- Die **Bereitsstellung** (engl. *allocation*) von Ressourcen (Addierern, usw.).
- Die **Zuordnung** (engl. *(resource) binding*). Hierdurch wird für jede Operation festgelegt, welche Hardware-Ressource (**wer**) die Ausführung übernimmt.

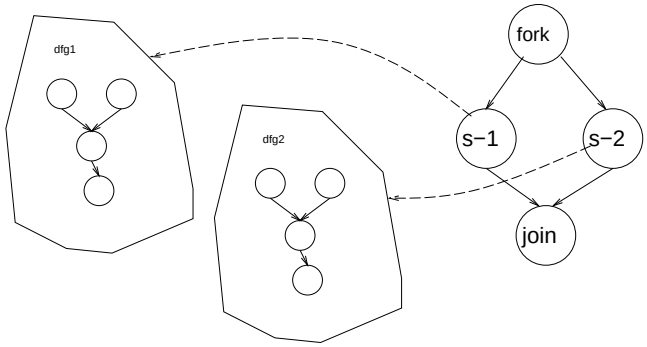

Abbildung 2.7: Kontroll/Datenflussgraph

Die Suche nach „optimalen" Architekturen erlaubt leider keine unabhängigen Verfahren zur Lösung der drei Aufgaben. Da aber bereits das Scheduling NP-hart ist, wird die Architektur-Synthese häufig dennoch in einzelne Phasen zerlegt.

Im Folgenden beschränken wir uns wieder auf DFGs. CDFGs werden häufig behandelt, indem sequenziell die einzelnen DFGs bearbeitet werden.

2.2.2 Scheduling

Die Bestimmung des Ausführungszeitpunktes von Operationen oder Tasks ist ein Standardproblem, das in verschiedenen Disziplinen (in der Logistik, der Fabrikationsplanung, in den Wirtschaftswissenschaften usw.) immer wieder vorkommt. Wir können grob zwischen dem Scheduling bei unabhängigen und bei abhängigen Aufgaben unterscheiden (siehe Abb. 2.8).

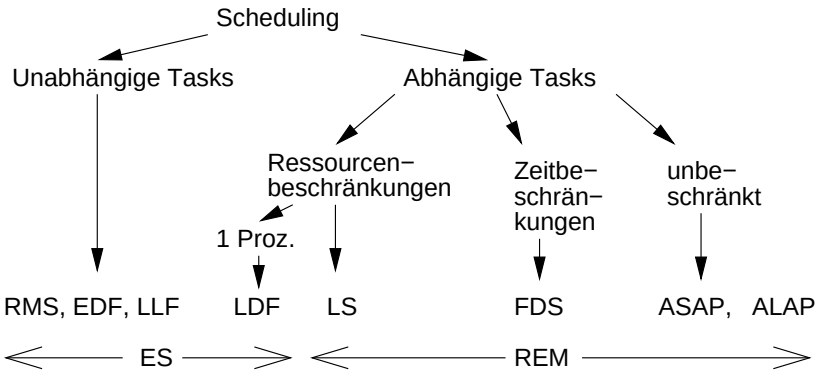

Abbildung 2.8: Eine Klassifikation von Scheduling-Verfahren

Zum Scheduling von unabhängigen Aufgaben sei hier auf die Vorlesung „Eingebettete Systeme" verwiesen [Mar07]. Scheduling-Verfahren für abhängige Operationen werden hier vorgestellt.

2.2.2.1 ALAP- und ASAP-Scheduling

Die einfachsten Scheduling-Verfahren für abhängige Tasks sind *as soon as possible* (ASAP) und *as late as possible* (ALAP) Scheduling. Beim ASAP-Verfahren werden alle Operationen so früh wie aufgrund der Abhängigkeiten möglich ausgeführt. Beim ALAP-Verfahren werden alle Operationen so spät wie aufgrund der Abhängigkeiten möglich eingeplant. Für das Beispiel der Determinantenberechnung führen diese Verfahren den Ergebnissen gemäß Abb. 2.9 und 2.10.

Das Beispiel beruht auf der Annahme gleicher Ausführungsgeschwindigkeiten aller Operationen. Üblicherweise wird eine synchrone Hardware vorausgesetzt. Die einzelnen „Zeiten" kennzeichnen Intervalle zwischen zwei Taktimpulsen. Man nennt diese Intervalle auch **Kontrollschritte**, (engl. *control steps*) da zu jedem Intervall ein Zustand des Steuerwerks gehört, das für eine geeignete Ansteuerung aller Bausteine des Rechenwerks sorgen muss (dies entspricht den Zuständen des Mikroprogramm-Steuerwerks, wie es für die

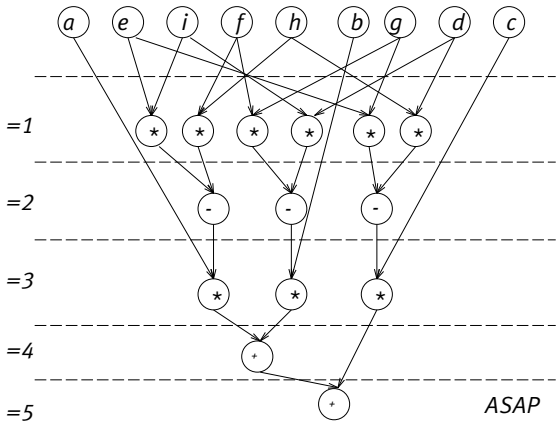

Abbildung 2.9: As-soon-as-possible Scheduling

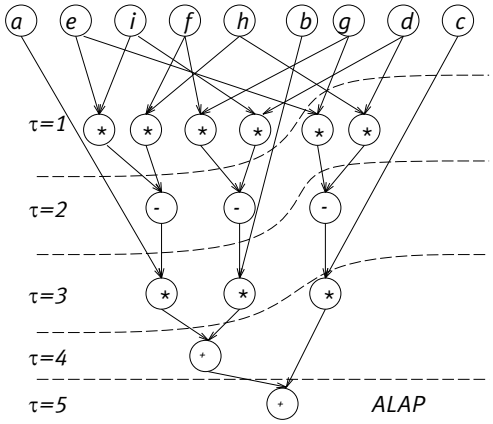

Abbildung 2.10: As-late-as-possible Scheduling

mikroprogrammierte Version der MIPS-Maschine in der Vorlesung „Rechnerstrukturen“ besprochen wurde).

2.2.2.2 Bezeichnungen

ASAP- und ALAP-Verfahren berücksichtigen keine Ressourcenbeschränkungen. Vor der Vorstellung von Verfahren, die Ressourcenbeschränkungen berücksichtigen, müssen wir zunächst einmal einige Bezeichnungen einführen.

Wir nehmen an, dass das Verhalten des zu entwerfenden Systems durch einen Datenflussgraphen gegeben ist. Die Knoten dieses Graphen bezeichnen Operationen wie Additionen und Multiplikationen. Genauer gesagt: sie bezeichnen Instanzen von Operationen (Operationstypen). Die Knoten dieses Graphen seien eindeutig durchnummeriert mit *integern* aus dem Bereich $J = \{1..j_{max}\}$. Als Variable für derartige Integer werden wir j benutzen.

Für die benutzten Operationstypen werden wir die Indexmenge G verwenden.

Sei *optype* eine Funktion, welche den Knoten des DFG die jeweiligen Operationstypen zuordnet (siehe Abb. 2.11).

Weiterhin nehmen wir an, dass die Struktur durch eine Netzliste mit den Baustein-Instanzen $k \in K = \{1..k_{max}\}$ beschrieben wird. Jede Baustein-Instanz wird aus einem zugehörigen Bibliothekstyp abgeleitet. Die Variablen $m \in M$ mögen die Typen von Bibliothekselementen bezeichnen. Die Funktion *type* bildet Baustein-Instanzen auf Baustein-Typen ab:

$$type : K \rightarrow M$$

Die Funktionalität eines Bausteins m wird durch die Funktion $\beta : M \rightarrow \wp(G)$ beschrieben:

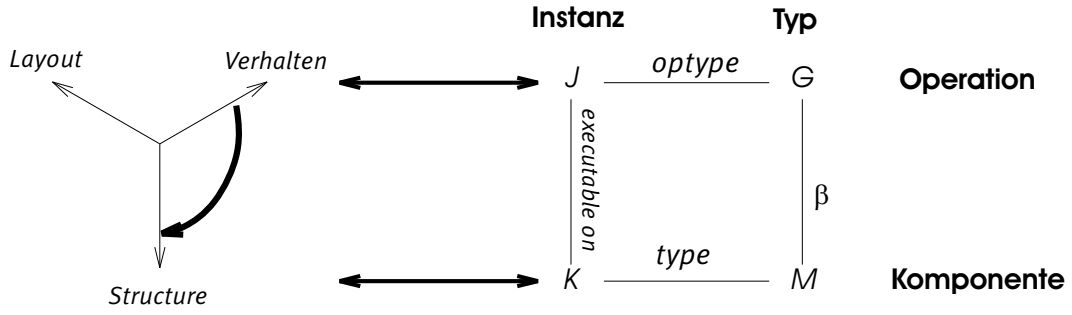

Abbildung 2.11: Notation in Synthese-Modellen

$\forall m \in M, g \in G : g \in \beta(m) \iff$ Bausteintyp m kann g ausführen.

Aus dieser Funktion leiten wir die folgende Relation ab:

Def.: $j \in J$ *executable on* $k \in K \iff optype(j) \in \beta(\wp(type(k)))$.

Die meisten Werkzeuge generieren nicht nur Struktur. Sie generieren auch eine Bindung zwischen Operationen und Kontrollschritten, in denen sie gestartet werden. Dies trifft auf alle uns bekannten Mikroarchitektur-Synthesewerkzeuge zu. Wir werden Indizes $i \in I$ benutzen, um Kontrollschritte zu bezeichnen. Modelle lassen in der Regel zu, sog. *multi-cycle operations* zu verwenden, welche mehrere Kontrollschritte benötigen. Der Einfachheit halber nehmen wir an, dass $\ell(j)$ für jede Operation die benötigte Anzahl an Kontrollschritten bezeichnet (d.h., vereinfachend nehmen wir an, dass alle Bausteine j gleich schnell ausführen können). Ausgereifte Synthesesysteme erlauben allerdings in der Regel eine Ausführungszeit, die von der Operation und vom Bausteintyp abhängt (d.h. $\ell = \ell(j, m)$).

2.2.2.3 *List Scheduling*

Eine Ressourcen-beschränkte Berechnung von Startzeitpunkten kann mit der listenorientierten Ablaufplanung (engl. *list scheduling*) durchgeführt werden. Wir setzen dabei voraus, dass für jeden Bausteintyp m ein Anzahl vorhandener Instanzen B_m festgelegt ist.

List scheduling beginnt zunächst einem topologischen Sortieren der Konten des Datenflussgraphens. Anschließend wird für jeden Knoten eine **Dringlichkeitsfunktion** (Priorität) p heuristisch bestimmt (s.u.).

Im Scheduling-Algorithmus sind einige Rechnungen wiederholt auszuführen:

- die Berechnung der in einem Kontrollschritt i auf einem Bausteinstyp m ausführungsbereiten Operationen, deren Vorgänger alle bekannt sind:

$$A_{i,m} = \{v_j \in V : type(v_j) \in \beta(m) \wedge \forall j' : (v_{j'}, v_j) \in E : i > \tau(v_{j'}) + \ell(j) - 1\}$$

- die Menge der Operationen des Typs m, die im Kontrollschritt i noch ausgeführt werden:

$$H_{i,m} = \{v_j \in V : type(v_j) \in \beta(m) \wedge \tau(v_j) + \ell(j) - 1 \geq i\} \tag{2.12}$$

- Bestimmung einer Menge S_i von zu startenden Operationen mit

$$|S_i| + |H_{i,m}| \leq B_m \tag{2.13}$$

Der eigentliche Algorithmus betrachtet dann in einem linearen Durchlauf durch die Menge notwendiger Kontrollschritte die jeweils einplanbaren Operationen [Tei97]:

```
List(G(V,E), β, B, p, ℓ){
  i:=0;
  repeat {
    for (m=1) to Anzahl Modultypen {
```

```
        Bestimme Kandidatenmenge A_{i,m};
        Bestimme Menge nicht beendeter Operationen G_{i,m} ;
        Wähle eine Menge S_i maximaler Priorität mit
          |S_i| + |H_{i,m}| ≤ B_m
        foreach (v_j ∈ S_i) : τ(v_j) := i; (*setze schedule fest*)
      }i := i + 1;
    }
    until (alle Knoten von V geplant);
    return (τ);
  }
```

Es gibt unterschiedliche Heuristiken zur Berechnung der Dringlichkeitsfunktionen (zitiert nach [MR92]):

- die Mobilitäts- oder Beweglichkeits-Heuristik: die Mobilität (engl. *mobility*) ist definiert als Differenz zwischen den aus ASAP- und ALAP-Knoten errechneten Zeitwerten. Knoten mit einer kleinen Beweglichkeit erhalten danach eine hohe Priorität.

  Die Mobilitätsheuristik will das Auswahlproblem des Knotens dadurch lösen, dass die einzuplanenden bereiten Operationen nach ihrer Mobilität sortiert werden. Da aufgrund von Ressourcen-Restriktionen nicht alle bereiten Operationen eingeplant werden können, versucht die Mobilitätsheuristik, wenigstens die Operationen auf dem kritischen Pfad einzuplanen, deren Mobilität aufgrund der Definition 0 ergibt. Bei knappen Ressourcen führt dieser Algorithmus allerdings nicht zu besonders guten Lösungen, weil dann die Mobilität als Auswahlkriterium keine gute Entscheidungshilfe ist.

- Lebenszeit-Heuristik:

  Die Lebenszeit-Heuristik hat das Ziel, eine gute Lösung für das Auswahlproblem zu finden, indem die Anzahl der benötigten Speicherzellen minimiert wird. Mit einer Lebenszeit-Analyse der Eingangswerte einer Operation kann festgestellt werden, ob durch das Einplanen der Operation die Speicherzellen frei werden, in denen die Eingangswerte dieser Operation gespeichert sind. Das Auswahlproblem wird also bevorzugt dadurch gelöst, dass die Operationen dann eingeplant werden, wenn die Lebenszeiten der Eingangswerte dieser Operation in dem vorliegenden Kontrollschritt enden.

- Vorausschau-Heuristik (*look ahead*):

  Während die bisher beschriebenen Heuristiken für die Lösung des Auswahlproblems im wesentlichen „greedy"-Strategien verfolgen und insbesondere keine Möglichkeit bieten, nachteilige Folgen einer einmal getroffenen Auswahl für spätere Kontrollschritte zu berücksichtigen, sollen die vorausschauenden Verfahren die Konsequenzen einer Auswahl für die Folgeschritte vorher bestimmen und berücksichtigen. Diese Verfahren sind zwar aufwendiger, führen aber in der Praxis zu wesentlich besseren Ergebnissen. Beispiele für solche Vorausschau-Heuristiken sind die Konflikt-Reduktions-Heuristik oder die Verzögerungs-Reduktions-Heuristik. Konflikte sind definiert als die Differenz zwischen der Anzahl der bereiten Operationen und der Anzahl der zur Verfügung stehenden Ressourcen. Die KonfliktReduktions-Heuristik versucht nun abzuschätzen, wieviele Konflikte sich in späteren Kontrollschritten unter der Annahme einer bestimmten Auswahl ergeben. Die Folgekonflikte können durch ein ASAP-Scheduling ohne Ressourcen-Beschränkung sehr effizient abgeschätzt werden.

Listenorientierte Ablaufplanung ist im Prinzip auf die Vorgabe von Ressourcen und die selbständige Wahl der Schedulelänge ausgelegt. Sie kann aber auch ohne Ressourcen-Beschränkungen durchgeführt werden.

2.2.2.4 *Force-directed scheduling*

Das nachfolgend beschriebene kräftebasierte Verfahren geht im Gegensatz zu listenorientierten Verfahren davon aus, dass die Schedulelänge fest ist und eine kostengünstige Ressourcenmenge gesucht wird. *Die zugrunde liegende Modellvorstellung, die auch bei Plazierungsverfahren verwendet wird, greift auf eine physikalische Analogie zurück: Zwischen Kontrollschritten werden, bezogen auf eine bestimmte Verarbeitungseinheit, „Kräfte" eingeführt, deren Stärke von der unterschiedlichen Auslastung der Operationseinheiten in den verschiedenen Kontrollschritten abhängt. Die auf diesem Kräfteverfahren basierende Ablaufplanung versucht nun, durch Verschieben von Operationen zwischen Kontrollschritten diese Kräfte zu minimieren. Dadurch wird eine möglichst gleichmäßige Auslastung aller Verarbeitungseinheiten erreicht und eine Minimierung der Verarbeitungseinheiten erzielt. Es wird davon ausgegangen, dass Operationen für jeden Zeitpunkt zwischen dem frühest- und dem spätestmöglichen mit gleicher Wahrscheinlichkeit eingeplant werden*

können. Aus dieser Annahme ergeben sich Wahrscheinlichkeiten für die Einplanung zu einem bestimmten Zeitpunkt. Das Kräfteverfahren geht nun im einzelnen wie folgt vor: Zunächst werden die Operationen eingeplant, deren frühestmöglicher Zeitpunkt mit dem spätestmöglichen zusammenfällt. Diese Operationen liegen also offensichtlich auf dem kritischen Pfad, während für alle anderen Operationen eine Verschiebung innerhalb dieser Zeitspanne möglich ist, ohne den gesamten Ablauf zu verzögern. In einem iterativen Verfahren zur Festlegung der Einplanung dieser freien Operationen wird versucht, die vorhandenen Komponenten möglichst gut auszunutzen, was einer Reduktion der Anzahl der benötigten Komponenten entspricht [MR92].

Die Einplanung der freien Operationen basiert auf der Berechnung einer „Wahrscheinlichkeitsverteilung“ der Operationen über die Kontrollschritte.

Deren Berechnung sei am Beispiel der Berechnung der Lösung einer Differentialgleichung (einem Benchmark der Synthese-*Community*) gezeigt. Lösungen der Differentialgleichung $y'' + 3zy' + 3y = 0$ werden danach durch das folgende kleine Programm bestimmt:

```
while (z < a) do
  begin
    zl:= z+dz
    ul:=u-(3 *z * u * dz)-(3 * y * dz);
    yl:=y+(u*dz);
    z:=zl;
    u:=ul;
    y:=yL;
  end;
```

Abb. 2.12 zeigt die dazugeörigen ASAP- und ALAP-Schedules.

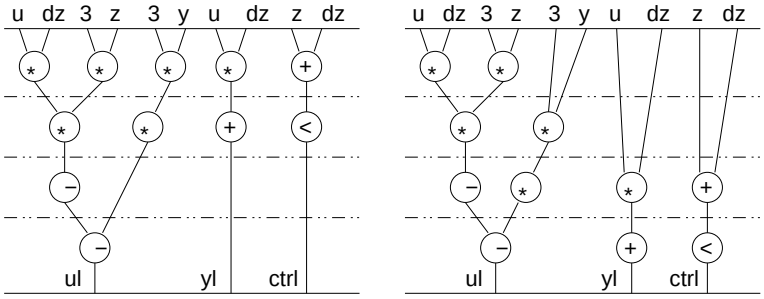

Abbildung 2.12: Schedules für das *diffeq*-Beispiel

Wir betrachten jetzt jeweils eine Menge von Operationstypen $H \subseteq \wp(G)$, für die dieselben Hardwarebausteine zur Realisierung in Frage kommen, also typischerweise $H = \{+, -\}$ oder $H = \{*\}$. Aus den Schedules bestimmen wir für die Operationen mit einem Typ aus H die **Zeitrahmen**, in denen eine Operation j ausgeführt werden kann. Wir bezeichnen die Menge der für j möglichen Kontrollschritte mit $R(j)$, ihre Anzahl mit $|R(j)|$.

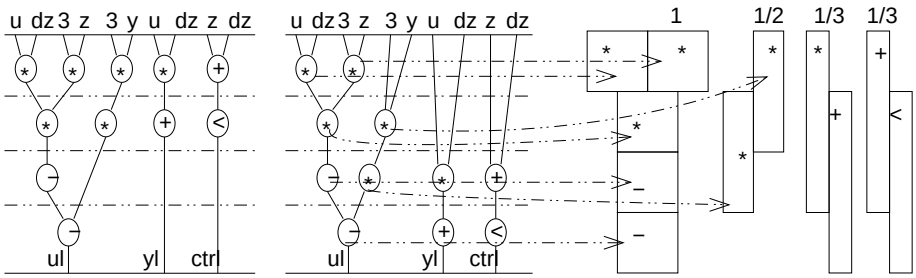

Abbildung 2.13: Zeitrahmen für das *diffeq*-Beispiel

Aus den Zeitrahmen bestimmen wir die „Wahrscheinlichkeit“ $P(j, i)$ für die Zuordnung von j zu i mittels der Gleichung [MR92]:

$$P(j,i) = \begin{cases} \frac{1}{|R(j)|} \text{ falls } i \in R(j) \\ 0 \text{ sonst} \end{cases}$$

Aus den „Wahrscheinlichkeiten“ für einzelne Operationen bestimmen wir „Verteilungen“ (engl. *distributions*) $D(i)$ mittels Betrachtung aller Operationen aus H:

$$D(i) = \sum_{j,type(j)\in H} P(j,i) \tag{2.14}$$

Abb. 2.14 zeigt die resultierenden „Wahrscheinlichkeiten“ und „Verteilungen“ für Multiplikationen im *diffeq*-Beispiel.

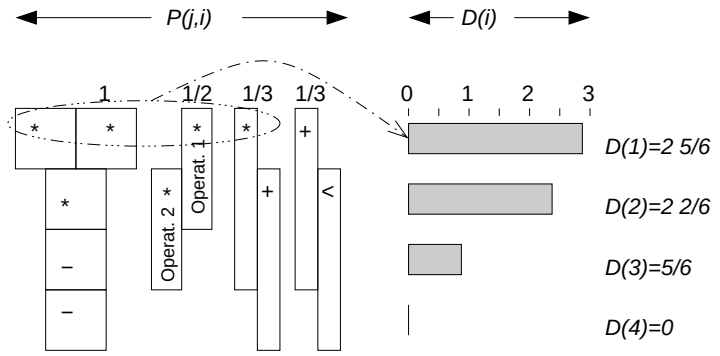

Abbildung 2.14: „Wahrscheinlichkeiten“ und „Verteilungen“ für Multiplikationen im *diffeq*-Beispiel

In Kontrollschritten, in denen die Verteilungsfunktion für einen gegebenen Operationstyp einen hohen Wert hat, sind demnach parallele Komponenten erforderlich. Um diese Parallelität zu reduzieren, müssen Operationen zu Kontrollschritten mit niedrigerem Wert der Verteilungsfunktion verschoben werden [MR92].

Zum diesem Zweck rechnen wir Kräfte aus. Wenn eine Operation j in einem festen Kontrollschritt i eingeplant wurde, gibt die Kraft $SF(j,i)$ die Änderung dieser Festlegung auf die Verteilung wieder:

$$SF(j,i) = \sum_{i' \in R(j)} D(i') \Delta P_i(j,i') \tag{2.15}$$

Dabei bezeichnet $\Delta P_i(j,i')$ die Änderung der Wahrscheinlichkeit im Kontrollschritt i', wenn die Operation j im Kontrollschritt i eingeplant wird. Die neue Wahrscheinlichkeit für die Einplanung von j in i ist 1. Vorher betrug diese $P(j,i)$, also ist die Differenz $1-P(j,i)$. Für Kontrollschritte $i' \neq i$ betrug die Wahrscheinlichkeit $P(j,i')$. Nach der Einplanung von j in i ist die Wahrscheinlichkeit 0, also ist die Differenz $-P(j,i')$. Daraus folgt:

$$\Delta P_i(j,i') = \begin{cases} 1 - P(j,i) \text{ falls } i = i' \\ -P(j,i') \text{ sonst} \end{cases}$$

Abb. 2.15 zeigt die Berechnung für unser Beispiel. Der positive Wert zeigt, dass es nicht ratsam ist, die Multiplikation im Schritt $i = 1$ einzuplanen.

Die Einplanung kann sich allerdings auch auf die vorhergehenden und nachfolgenden Kontrollschritte auswirken, was sich mit sog. Vorgänger bzw. Nachfolgerkräften ausdrücken läßt. Die Vorgänger- bzw. Nachfolgerkräfte VF bzw. NF sind definiert als:

$$VF(j,i) = \sum_{j' \in \text{ Vorgänger von } j} \sum_{i' \in I} D(i') \Delta P_{j,i}(j',i') \tag{2.16}$$

$$NF(j,i) = \sum_{j' \in \text{ Nachfolger von } j} \sum_{i' \in I} D(i') \Delta P_{j,i}(j',i') \tag{2.17}$$

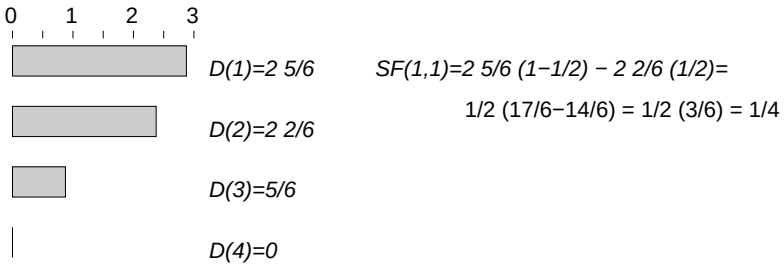

Abbildung 2.15: Berechnung von (direkten) Kräften für das *diffeq*-Beispiel

Dabei ist jetzt $\Delta P_{j,i}(j', i')$ eine Verallgemeinerung von $\Delta P_i(j', i')$. $\Delta P_{j,i}(j', i')$ ist die Änderung der Wahrscheinlichkeit der Einplanung von j' im Kontrollschritt i', wenn die Operation j im Kontrollschritt i eingeplant wird. Die Gesamtkraft $F(j, i)$ ergibt sich als Summe über die direkten, die Vorgänger- und die Nachfolgerkräfte:

$$F(j,i) = SF(j,i) + VF(j,i) + NF(j,i) \tag{2.18}$$

Betrachten wir beispielsweise die Einplanung der Multiplikation 1 (siehe Abb. 2.14) im Kontrollschritt 2. Die direkte Kraft bei einer solchen Entscheidung ergibt sich zu

$$\begin{aligned} SF(1,2) &= D(1) * \Delta P_2(1,1) + D(2) * \Delta P_2(1,2) \\ &= 2\frac{5}{6} * (-0,5) + 2\frac{2}{6} * 0.5 \\ &= -\frac{17}{12} + \frac{14}{12} \\ &= -\frac{3}{12} = -\frac{1}{4} \end{aligned} \tag{2.19}$$

Die zweite Multiplikation (siehe Abb. 2.14) wird aufgrund einer solchen Entscheidung dem dritten Kontrollschritt zugeordnet. Daher ergibt sich folgende Kraft für den Nachfolger:

$$\begin{aligned} NF(1,2) &= D(2) * \Delta P_{1,2}(2,2) + D(3) * \Delta P_{1,2}(2,3) \\ &= 2\frac{2}{6} * (-0,5) + \frac{5}{6} * 0.5 \\ &= -\frac{14}{12} + \frac{5}{12} \\ &= -\frac{9}{12} = -\frac{3}{4} \end{aligned} \tag{2.20}$$

Für die Gesamtkraft ergibt sich

$$F(1,2) = SF(1,2) + NF(1,2) = -\frac{1}{4} + (-\frac{3}{4}) = -1 \tag{2.21}$$

Unter Berücksichtigung von indirekten Kräften ergibt sich der niedrige Wert von -1. Dieser zeigt an, dass eine Einplanung der Multiplikation im Kontrollschritt 2 zu empfehlen ist.

Mit diesen Kräften lässt sich der vollständige Ablauf des kräftebasierten Verfahrens angeben [MR92]:

```
procedure kräfteverfahren;
  begin
    ASAP-Scheduling;
```

```
    ALAP-Scheduling;
    while nicht alle Operationen eingeplant do
      begin
        wähle Operation mit niedrigster Gesamtkraft aus;
        plane Operation in dem Kontrollschritt mit niedrigster Kraft ein;
        berechne Ausführungsintervalle neu;
        berechne D(i) neu;
      end
end
```

Das Verfahren muss für jede zu betrachtende Menge H wiederholt werden, also typischerweise für Multiplikationen und Addtionen getrennt durchgeführt werden. Überlappen sich die Funktionalitäten aber (können also etwa Multiplizierer auch addieren), so gerät das ursprüngliche Verfahren an seine Grenzen. Würde man Additionen und Multiplikationen zu einem H zusammenfassen, so könnte man die unterschiedlichen Kosten von Addierern und Multiplizierer nicht mehr berücksichtigen.

Force-directed scheduling ist ein sehr populäres Verfahren, von dem verschiedene Varianten in Benutzung sind, auch außerhalb des Mikroelektronikentwurfs.

Komplexere Scheduling-Verfahren müssen u.a. auch unterschiedliche Ausführungsgeschwindigkeiten der verschiedenen Operationen berücksichtigen und zu möglichst günstigen Allokationen führen.

2.2.3 Bereitstellung (*allocation*)

Dieser Schritt legt fest, welche wesentlichen Hardware-Ressourcen zur Lösung des Syntheseproblems zur Verfügung gestellt werden sollen. Als Ressourcen kommen u.a. arithmetisch/logische Einheiten, Speicherzellen und Verbindungen in Frage. Im Folgenden ist die Erklärung[3] allgemein an „Ressourcen“ ausgerichtet. Im konkreten Fall ist dafür eine der erwähnten Ressourcen-Klassen einzusetzen.

Während der Bereitstellung wird noch keine Zuordnung zwischen Ressourcen und ausgeführten Operationen vorgenommen. Die Methode der Kopplung der verschiedenen Synthesephasen hängt dabei vom Synthesesystem ab. Zum Teil sind die Phasen weitgehend unabhängig voneinander, zum Teil ist aber eine gewisse Vorausschau auf die späteren Synthesephasen vorhanden und zum Teil werden Synthesephasen zusammengefasst.

Es gibt mehrere Methoden, Ressourcen zur Verfügung zu stellen. Dabei unterscheiden sich die einzelnen sowohl hinsichtlich der benutzten Teilalgorithmen als auch in der Reihenfolge von deren Anwendung.

2.2.3.1 Vorgabe durch den Benutzer

Bei dieser Form erklärt der Benutzer bereits, welche Ressourcen einer bestimmten Klasse in der späteren Struktur vorhanden sein sein sollen. Dieses ist die einfachste Form, Ressourcen auszuwählen. Die Möglichkeit der Vorgabe von Ressourcen bereits in der Spezifikation sollte vorhanden sein. Hierzu werden Spezifikationssprachen benötigt, die eine Beschreibung einer Ressourcen-Bibliothek ermöglichen. Über diese Möglichkeit hinaus kann auch eine selbstständige Auswahl der Ressourcen durch das Synthesesystem erfolgen.

2.2.3.2 Direkte Compilation

Eine noch recht einfache Möglichkeit der Bereitstellung besteht darin, für jedes Vorkommen eines Operators eine eigene Ressource zu reservieren und keinerlei Mehrfachausnutzung vorzusehen. Wie in gewöhnlichen Compilern wird für jede Operation „Code“ erzeugt. Bei direkter Compilation werden Bereitstellung und Zuordnung generell in derselben Synthesphase durchgeführt.

Dieser Ansatz ist nur für kleine Verhaltsbeschreibungen geeignet, denn für eine Verhaltensbeschreibung mit (beispielsweise) 1000 Additionen würden ja 1000 Addierer erzeugt. Dennoch wird dieser Ansatz in einfachen Systemen benutzt. Vielfach wird einer Variablen der Verhaltensbeschreibung genau ein Register exklusiv zugeordnet. Dies ist dann ein Fall von direkter Compilation.

[3] Die Darstellung basiert auf dem Buch [Mar93] des Autors.