Eine gemeinsame Basis zur formalen Beschreibung von Software und Hardware

Hermann von Issendorff
Institut für Netzwerkprogrammierung
Hauptstr. 40, D-21745 Hemmoor

Erweiterte Zusammenfassung:

Im letzten Jahr habe ich in diesem Workshop über das Thema "Über die formale Beschreibung räumlicher Netze" vorgetragen. Ein räumliches Netz, also ein Netz im dreidimensionalen Raum, erhält man, wenn man ein diskretes physikalisches System von seiner Metrik und seiner Funktionalität abstrahiert. Das übrig bleibende Skelett des Systems bildet im allgemeinen Fall ein räumliches Netz, in dem die Netzknoten die Systemkomponenten darstellen und die Netzkanten die Beziehungen zwischen den Komponenten. Ein diskretes physikalisches System kann statisch oder funktional sein. Ein funktionales System ist von vornherein gerichtet, ein statisches kann willkürlich als gerichtet angenommen werden.

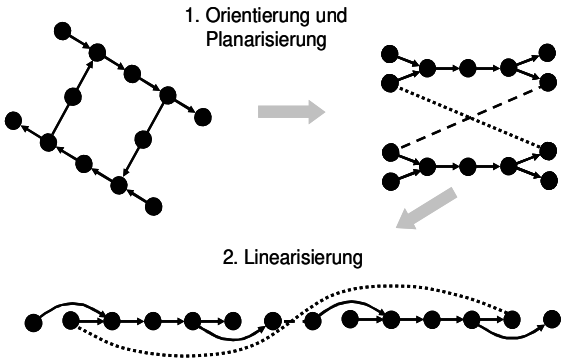

Bild 1: Homöomorphe Abbildung einer gerichteten Knotenstruktur auf eine orientierte Knotenfolge (Gummibandabbildung)

Das Skelett des diskreten physikalischen Systems kann in einem dreidimensionalen topologischen Raum beschrieben werden, der durch einen Beobachter aufgespannt wird, der zwischen links/rechts, oben/unten und vorne/hinten unterscheidet. In diesem Bezugssystem lassen sich die Knoten eines räumlichen Netzes zunächst auf eine Beobachtungsebene abbilden, die zwischen dem Netz und dem Beobachter liegt. Die Abbildung erfolgt so, dass alle Netzknoten und Knotengruppen von links nach rechts orientiert werden. Zyklische Knotennetze werden an beliebiger Stelle symbolisch aufgetrennt und die dadurch entstehenden Enden durch ein spezielles Knotenpaar markiert. Die Information über die Raumtiefe steckt dann nur noch in den Netzkanten, die sich untereinander kreuzen können. Die Kreuzungen lassen sich in ähnlicher Weise dadurch beseitigen, dass jede untere Netzkante symbolisch aufgeschnitten und die entstehenden Enden durch einen zweiten Typ von Knotenpaar markiert wird. Die orientierte planare Struktur des Knotennetzes lässt sich sodann durch weitere Streckung und Auftrennung in eine orientierte lineare Struktur umformen.

Die homöomorphen Abbildungen sind in Bild 1 gezeigt. Das Knotennetz enthält einen Zyklus, dessen Auftrenung symbolisch durch die gestrichelte Linie dargestellt wird. Mit der orientierten Planarisierung entsteht eine Kreuzung, deren untere Verbindung aufgetrennt und symbolisch als gepunktete Linie dargestellt wird. Nach der Linearisierung ergibt sich eine Knotenkette, die reale und symbolische Verbindungen enthält.

Die Abbildungen der dreidimensionalen Struktur des Knotennetzes über eine zweidimensionale auf eine eindimensionale Struktur sind Homöomorphismen, d.h. sie sind bijektiv und kontinuierlich und damit umkehrbar. Anschaulich lässt sich Homöomorphismus durch ein Gummibandmodell beschreiben, in dem ein diskretes System beliebig gestreckt und verwunden werden kann, aber ohne Einwirkung äusserer Kräfte wieder in die ursprüngliche Form zurückkehrt.

Wie sich zeigt, lässt sich die eigentliche Struktur eines räumlichen Knotennetzes auf drei elementare Knoten zurückführen, die eine *(1,1)-, (1,2)-* oder *(2,1)-*Verbindungsstruktur haben, wobei die erste Zahl die Eingangs- und die zweite die Ausgangsverbindungen nennt. Knoten mit mehr Eingangs- und Ausgangsverbindungen lassen sich in elementare Knoten zerlegen.

Die Abbildungen lassen sich durch eine formale Sprache beschreiben, die Akton-Algebra genannt wird. Sie ist eine Termalgebra, deren Elemente als Aktonen bezeichnet werden. Akton-Algebra hat in dieser abstrakten Form eine reine Strukturesemantik, in der die Strukturaktonen *Fork, Join, Link, Exit, Entry, Down* und *Up* die elementaren Netzknoten beschreiben. *Exit/Entry*-Paare dienen zur symbolischen Trennung von Verbindungen in der Ebene, z.B. von Zyklen, *Down/Up*-Paare zur symbolischen Trennung von Unterkreuzungen. Zwei binäre Operatoren, *Next* und *Juxta* genannt, dienen zur Beschreibung abhängiger bzw. unabhängiger Nachbarschaft.

Akton-Algebra ist zudem kompositional und konstruktiv; kompositional, weil jeder Aktonterm in einem Akton verborgen werden kann, und konstruktiv, weil die totale Ordnung eines akton-algebraischen Textes die partielle Ordnung enthält, die den Aufbau eines statischen Systems bzw. die Aktivitäten eines funktionalen Systems beschreibt.

Versieht man die Aktonen mit funktionaler Semantik, dann wird aus dem abstrakten Knotennetz ein konkretes Datenflussnetz. Die funktionale Semantik kann

digital, analog oder gemischt digital/analog sein. Digitale Funktionen lassen sich beispielsweise dadurch einführen, dass das Basiselement *Join* als Sortenbezeichner für binäre Schaltelemente wie *And, Or, Nand, Nor*, etc. erklärt wird. Akton-Algebra wird dadurch zu einer Datenflusssprache.

Versieht man die Aktonen mit Metrik, d.h. physikalischen Abmessungen der Komponenten, dann erhält man eine Layout-Sprache, die die räumliche Struktur diskreter physikalischer Systeme beschreibt. Dem Gummibandmodell entsprechend, kann diese räumliche Struktur durch Transformation in eine planare Anordnung der Komponenten überführt werden. *Down/Up*-Paare markieren in dieser Anordnung die Durchkontaktierungen, die unterhalb der Beobachtungsebene verbunden werden müssen. Da *Down/Up*-Paare topologische Schnitte unterhalb der Beobachtungsebene sind, können sie dort immer vereinigt, d.h. als lokale Punkt-zu-Punkt-Verbindung realisiert werden. Verzichtet man auf Geradlinigkeit, dann ist es der Lokalität wegen immer möglich, alle Punkt-zu-Punkt-Verbindungen gemeinsam auf einer zweiten Ebene zu realisieren. Dies beinhaltet einen wichtigen technischen Fortschritt, der das Layout entscheidend vereinfacht. Heutige Layout-Verfahren gehen immer von Paarlisten aus, die lediglich Funktionszusammenhänge, aber keine räumliche, insbesondere keine Nachbarschaftsinformation enthalten. Eine geeignete Verteilung der Komponenten muss daher durch Placement, d.h. durch Permutation der Komponenten erreicht werden. Akton-Algebra dagegen enthält bereits alle Nachbarschaftsinformation. Ein Layout mit Aktonalgebra beschränkt sich damit auf die Schachtelung der Komponenten, die lediglich Translation und Rotation erfordert.

Schwerpunkt dieses Vortrags ist der Zusammenhang zwischen Akton-Algebra auf der einen Seite und Programmiersprachen auf der anderen.

Wie vorstehend behandelt, ist jede diskrete räumliche Struktur in Aktonalgebra darstellbar, darunter insbesondere Zyklen und Kreuzungen. Als elementare Funktionskomponenten stehen z.B. digitale Gatter und Inverter zur Verfügung. Dies ermöglicht den Ausbau von Zyklen zu positiven und negativen Rückkopplungen. Positive Rückkopplung ist die Grundlage von Speichern, negative die Grundlage von bedingten Schleifen. Kreuzungen treten in allen Schaltungen auf, in denen verschiedene Information gemeinsam verarbeitet wird.

Mit Akton-Algebra ist damit jede digitale Funktionsstruktur beschreibbar. Aus der Beschreibung einfacher Strukturen lässt sich durch Komposition die Beschreibung komplexerer Strukturen gewinnen. Auf diese Weise ist es möglich, die Menge aller Maschineninstruktionen eines Rechners zu beschreiben. Auf der Basis der Maschinenbefehle lässt sich ein Assembler aufbauen, der wiederum als Grundlage für die Compiler höherer Programmiersprachen dient.

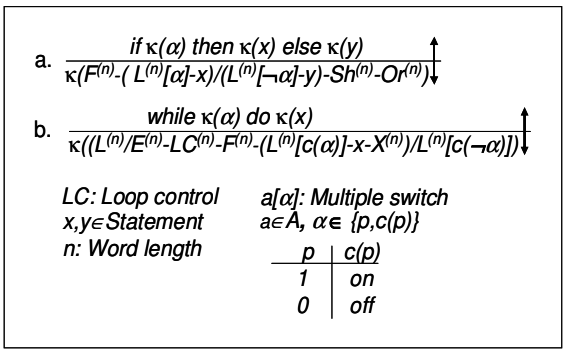

Bild 2: Konversionsregeln für eine if-then-else-Anweisung und eine while-Schleife

Akton-Algebra kann aber auch direkt zur Beschreibung der Funktionen höherer Programmiersprachen verwendet werden. Im Vortrag wird das am Beispiel einer *if-then-else*-Anweisung und einer *while*-Schleife demonstriert. Bild 2 zeigt die beiden Konversionsregeln. Darin ist κ eine Konversionsfunktion und α eine Boolesche Bedingung. Die akton-algebraischen Notationen sind folgenderweise zu lesen: Der Operator *Next* wird durch einen Bindestrich dargestellt, der Operator *Juxta* durch einen Schrägstrich. Das Attribut *(n)* bezeichnet die Wortlänge. *F* bedeutet *Fork*, *L Link, L(*α*)* ein bedingtes *Link*, also einen multiplen Schalter, der die Werte *0* oder *1* hat. *Sh* bedeutet *Shuffle*, d.h. die bitstellengerechte Zusammenführung von zwei Wörtern der Länge *n*. Die akton-algebraischen Notationen in der Konvertierungsregel zur *while*-Schleife sind folgenderweise zu lesen: *E* bedeutet *Entry*, *X* bedeutet *Exit*. *LC* ist eine komplexe Struktur, die die Schleife kontrolliert, Zwischenergebnisse speichert und die Schaltbedingung α setzt. Im Gegensatz zur *if-then-else*-Anweisung sperrt α in der *while*-Anweisung entweder die Schleife oder den Ausgang. In taktgesteuerten Rechnern sperrt α den Takt.

In der gleichen Weise lassen sich alle Elemente einer Programmiersprache in Akton-Algebra ausdrücken. Zusammengefasst ergibt sich für jede Programmiersprache eine Konversionstabelle, die eine Konversion von Programmen in Akton-Algebra oder umgekehrt ermöglicht. Während Programmiersprachen häufig nur unter Schwierigkeiten und unter Effizienzverlusten ineinander konvertierbar sind, insbesondere wenn sie sich, wie z.B. imperative und funktionale Sprachen, strukturell unterscheiden, ist die Konversion zu und von Akton-Algebra immer einfach und direkt möglich. Das prädestiniert Akton-Algebra als gemeinsame Zwischensprache zur Konvertierung einer Programmiersprache in eine andere.

So mächtig Akton-Algebra ist, so sicher ist aber auch, dass sie sich ihrer strengen Form wegen nicht als Programmiersprache eignet.