Über die formale Beschreibung räumlicher Netze

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

Erweiterte Zusammenfassung:

Thema dieses Vortrags ist ein grundlegender Formalismus, der die räumliche Struktur und - sofern vorhanden - das Zeitverhalten diskreter Systeme beschreibt, d.h. von Systemen mit individuellen Komponenten. Genauer gesagt, ist der Formalismus eine formale Sprache, die in ihrem Kern das Skelett beschreibt, auf das sich alle diskreten Systeme durch Abstraktion reduzieren lassen. Umgekehrt kann das Skelett durch Einbeziehung der Eigenschaften diskreter Systeme erweitert werden. Das Ergebnis ist dann eine formale Sprache mit mehr oder weniger speziellen Eigenschaften.

In den vergangenen Jahren ist wiederholt über den Stand der Aktonalgebra berichtet worden, die ursprünglich als Programmiersprache für Rechnernetze dienen sollte. Seit längerer Zeit ist erkannt, dass Aktonalgebra mit verschiedenen Semantiken ausgestattet werden kann und so z.B. die Eigenschaften einer klassischen Programmiersprache, einer Programmiersprache für digitale oder analoge Schaltungen oder einer Layout-Sprache annehmen kann. Unklar war bisher, wie die gemeinsame Basis aller dieser Anwendungssprachen formal erfasst werden kann. Mit diesen Grundlagen, in denen - erstaunlicherweise - der geordnete dreidimensionale Raum eine wesentliche Rolle spielt, befasst sich dieser Vortrag.

Eine formale Sprache dient zur Erstellung von Texten. Texte sind Zeichenfolgen, die einen Anfang und ein Ende haben und Zeichen für Zeichen, d.h. zeitlich geordnet, gelesen werden. Abstrakt ausgedrückt sind Texte gerichtete eindimensionale Strukturen. Der zeitlichen Ordnung wegen kann ein Text als Anweisungsfolge sowohl zum schrittweisen und damit konstruktiven Aufbau eines Systems als auch zur schrittweisen Änderung von lokalen Zuständen (Auswertung von Funktionen) verstanden werden.

Abstrahiert man beispielsweise einen Computer von seiner Metrik, dann wird aus dem gerichteten zyklischen dreidimensionalen Netz von Komponenten ein Netz, das nur noch die Funktionen der Komponenten enthält. Abstrahiert man alternativ von der Funktionalität des Computers, bleibt ein gerichtetes zyklisches dreidimensionales Netz von Bausteinen übrig, die die Abmessungen der Komponenten haben. Abstrahiert man von Metrik und Funktionalität, dann bleibt ein gerichtetes zyklisches dreidimensionales Netz von Knoten übrig, die untereinander partiell verbunden sind. Die Verbindungen zwischen den Knoten stellen irreflexive und intransitive Relationen dar, die in diesem Beispiel gerichtet und damit auch asymmetrisch sind. Dieses Knotennetz ist das topologische Skelett, auf dem jede formale Sprache aufbaut. In den üblichen Programmiersprachen ist nur deshalb nichts von dem Skelett wahrnehmbar, weil alle Räumlichkeit in Komplexkonstrukten versteckt ist.

Um die räumlichen Eigenschaften des topologischen Skeletts zu erfassen, ist ein Bezugssystem erforderlich. Ein solches lässt sich mittels eines Beobachters definieren, der in natürlicher Weise Raum in den Relationen rechts bzw. links, oben bzw. unten und vorne bzw. hinten wahrnimmt. Als Abbildungsfläche wird eine Beobachtungsebene zwischen dem Beobachter und dem Knotennetz definiert.

Zur Beschreibung eines beliebigen Knotennetzes in der Beobachtungsebene sind 8 gerichtete Strukturelemente erforderlich, die Entry, Exit, Up, Down, Link, Fork, Join und AS (vollständiges Akton-System) genannt werden und die Basis für die mehrsortige Aktonalgebra bilden. Gleichindizierte Exit/Entry- bzw. Down/Up-Paare dienen dazu, Verbindungskreuzungen und Zyklen symbolisch zu beschreiben und so aus der expliziten Darstellung zu entfernen. Dadurch wird ein dreidimensionales Knotennetz in ein homöomorphes planares Knotennetz abgebildet, das gerichtet und in der Beobachtungsebene orientiert ist (z.B. von links nach rechts), weder Kreuzungen noch Zyklen enthält und direkt mittels zweier binärer Operatoren algebraisch beschreibbar ist. Das planare Knotennetz lässt sich in ähnlicher Weise in eine homöomorphe Zeichenfolge, d.h. einen Text abbilden.

Die abstrakte Aktonalgebra kann ohne Probleme in eine Anwendungssprache überführt werden, indem den Sorten Mengen strukturidentischer Elemente zugeordnet werden. Die Sortenbezeichner werden dadurch zu Sortenvariablen. Z.B. wird die Aktonalgebra durch Join$\in${And,Or,Nand,Nor} und Link$\in${Leitung,Inverter} zu einer Hardware-Entwurfssprache. Fügt man dieser Sprache zusätzlich Funktionsbeschreibungen der Elemente hinzu, dann wird sie zu einer Hardware-Simulationssprache. Führt man stattdessen mit Link$\in${Right,Left,Ahead} und Fork$\in${Right-Ahead,Left-Ahead,Right-Left} konkrete Strukturelemente ein und dazu einen Längenmassstab, in dessen Raster die Strukturelemente gemessen werden, dann hat man eine Layout-Programmiersprache.

Mit Hilfe eines graphischen Programms werden sowohl aktonalgebraisch definierte Digitalschaltungen als auch ihre abstrakten Netzstrukturen gezeigt.