Von mathematischen über algorithmische zu physikalischen Strukturen

Hermann von Issendorff
Hauptstr. 40, 21745 Hemmoor
hviss@issendorff.de

Erweiterte Zusammenfassung. In der Vergangenheit wurde auf diesem Workshop schon mehrfach darüber vorgetragen, dass eine Programmiersprache, die Aktonalgebra, definiert werden kann, mit der sich nicht nur wie mit klassischen Programmiersprachen Datenverarbeitung beschreiben lässt, sondern auch die Struktur und das Layout der Hardware-Systeme, auf denen die Datenverarbeitung ablaufen soll. Dies führt unmittelbar auf die Frage, wie sinnvoll es ist, Software, Hardware und Layout von Rechnersystemen gemeinsam in einem einheitlichen Formalismus zu beschreiben. Die Antwort lässt sich in drei Schlagworten zusammenfassen: Man gewinnt an formaler Korrektheit, Automatisierbarkeit und Wirtschaftlichkeit. Zwischen der Ablaufbeschreibung, die mit Programmiersprachen geschieht, und den Schaltungsstrukturen, auf denen die Datenverarbeitung stattfindet, klafft eine semantische Lücke, die mit Compilern, Hardware-Entwurfssystemen und Layoutverfahren bisher nur inselartig abgedeckt wird. Aktonalgebra schliesst diese Lücken. Sie garantiert damit durch-gängige formale Korrektheit, ermöglicht maschinelle Bearbeitung und bietet eine Basis für die automatische Übertragung von Software in Chips.

Dieser Vortrag befasst sich mit der Frage, was mathematische Algebren, Programmiersprachen und Aktonalgebra verbindet und unterscheidet. Wie sich zeigt, beschreibt diese Reihenfolge einen Weg zunehmender Konkretisierung

Die Strukturen klassischer mathematischer Formalismen sind abstrakt, genauer, abstrakt von den Raumzeiteigenschaften der Physik. Die Zeichenfolgen, mit denen algebraische Ausdrücke beschrieben werden, sind, sieht man von der ihnen aufgeprägten Semantik ab, nichts anderes als physikalisch-konkrete lineare Strukturen. Setzt man zwei verschiedene algebraische Ausdrücke gleich, wie das in den Axiomen geschieht, dann verzichtet man auf die konkrete strukturelle Information. Die Wirkung einer solchen Abstraktion lässt sich besonders klar am Kommutativaxiom erkennen. Das Kommutativaxiom entspricht physikalisch der Gleichsetzung einer Struktur mit seinem Spiegelbild. Dabei geht die in der Struktur enthaltene Ordnungsinformation verloren, was bedeutet, dass nicht mehr zwischen links und rechts unterschieden werden kann, und bei zeitlich interpretierten Strukturen nicht mehr zwischen früher und später. Es ist dann keine Ordnung mehr festlegbar, in der die Ausdrücke ausgewertet werden sollen.

Konventionelle Programmiersprachen dagegen sind spezielle Algebren, die eine explizite oder implizite Auswertungsordnung haben, in denen also kein Kommutativaxiom gilt. Andererseits beschreiben sie nur die Verarbeitung von Daten, aber nicht die Strukturen, auf denen die Verarbeitung stattfindet. Sie können deshalb als raumabstrakte aber zeitkonkrete Algebren bezeichnet werden.

Aktonalgebra geht noch einen Schritt weiter: Sie ist raumzeitkonkret. Mit dieser Raumzeitsemantik lassen sich diskrete Systeme, d.h. Maschinen, strukturell und

operational vollständig beschreiben. Zu jeder Maschine gibt es genau eine aktonalgebraische Beschreibung und umgekehrt.

Aktonalgebra hat einerseits die gleiche Berechnungsmächtigkeit wie alle universelle Programmiersprachen, zusätzlich aber auch die Fähigkeit, die räumlichen Strukturen beschreiben zu können, auf der die Datenverarbeitung stattfindet. Abstrahiert man von der Raumstruktur, dann bleiben nur die Eigenschaften der universellen Programmiersprachen erhalten Abstrahiert man vom Zeitverhalten der als Aktonen bezeichneten Komponenten, dann wird Aktonalgebra zu einer Programmiersprache für räumliche Strukturen, darunter insbesondere multiplanare Layouts.

Die formale Beschreibbarkeit von DV-Strukturen wurde in der Vergangenheit vielfach untersucht. Die zu diesem Zweck entworfenen Algebren oder Programmiersprachen sind jedoch alle raumabstrakt, d.h. alle Konstrukte, die nur in einer räumlichen Struktur realisierbar sind, werden verkapselt und sind damit einer analytischen Behandlung entzogen. Die Network Algebra [1] z.B. beschreibt nur planare Strukturen und verwendet dazu verkapselte Verbindungselemente wie Mehrfach-Kreuzung, -Verzweigung, -Vereinigung, und -Schleife. Einen ähnlichen Ansatz macht Möller [4], der von einer rein funktionalen Beschreibung ausgeht, in der die räumlichen Eigenschaften der verschiedenen Hardware-Elemente in speziellen Moduln versteckt werden. Ruby [2] beschreibt die Beziehungen zwischen Digitalschaltungen, aber ohne Bezug auf die konkrete Struktur. Einzig CADIC [3] ist als Layout-Sprache entwickelt, die konkrete planare Strukturen beschreibt. CADIC besitzt aber keine Funktionalität.

Die heute allgemein verwendeten Hardware-Programmiersprachen, wie z.B. VHDL oder Verilog, beschreiben nur die Funktionen einer Schaltung, nicht aber ihre Struktur. Die funktionale Beschreibung ist zudem auf die Auflistung Boolescher Funktionen zwischen Speichertakten, d.h. auf die Registertransfer-Ebene beschränkt. Da für das Layout keine Strukturinformation zur Verfügung steht, muss eine geeignete Schaltung durch Vertauschen der Komponenten sowie deren Verbindungen gesucht werden. Dies Verfahren ist bei der heute üblichen grossen Menge der Komponenten aufwändig und liefert üblicherweise nur suboptimale Lösungen.

In [5] wurde bereits gezeigt, dass Aktonalgebra so allgemein und elementar ist, dass konventionelle Programmiersprachen und mathematische Algebren in ihr ausgedrückt werden können. Mit einfachen Konversionsregeln, die eine Raumstruktur per Default hinzufügen oder von ihr abstrahieren, lassen sich Programmiersprachen oder mathematische Algebren in Aktonalgebra konvertieren oder umgekehrt.

Literatur

1. Stefanescu, Gh.: Network Algebra. Theoretical Springer-Verlag (2000)
2. Jones, G., Sheeran, M.: Circuit Design in Ruby. In: Staunstrup, J. (ed.): Formal Methods of VLSI Design. North Holland (1990) 13-70
3. Kolla, R., Molitor, P., Osthof, H.G.: Einführung in den VLSI-Entwurf. Teubner-Verlag, Stuttgart (1989)
4. Möller, B.: Deductive Hardware Design: A Functional Approach. In: Möller B., Tucker J.V. (eds): Prospects of Hardware Foundations, LNCS, Vol. 1546. Springer-Verlag (1998)
5. von Issendorff, H.: Algebraic Description of Physical Systems. In: Moreno-Diaz R., Buchberger B., Freire Nistel J.L (eds.): Computer Aided Systems Theory - EUROCAST'01, LNCS, Vol. 2178. Springer-Verlag (2001)