KIEL – Ein Computersystem zur Visualisierung der Auswertung von funktionalen Programmen

Rudolf Berghammer

Institut für Informatik und Praktische Mathematik
Christian-Albrechts-Universität Kiel
Olshausenstraße 40, D-24098 Kiel

Zusammenfassung Wir stellen das Computersystem KIEL vor, welches die interaktive Auswertung von funktionalen Programmen erster Stufe erlaubt, die in einer einfachen Teilsprache von Standard ML geschrieben sind. KIEL wurde für den Unterricht im Grundstudium konzipiert und hier insbesondere im Hinblick auf die Visualisierung von Programmauswertungen und die Auswirkungen von Berechnungsregeln.

1 Einleitung

Seit etwa zwei Jahrzehnten findet das funktionale Programmierparadigma immer größeren Anklang und wird mittlerweile auch gerne verwendet, um an Universitäten den Informatikstudierenden die grundlegenden Konzepte des Programmierens nahezubringen, oder sogar, wie in [2] beschrieben, jene der gesamten Informatik. Ein Grund dafür ist sicher, daß funktionale Programme in der Regel einfacher, leichter zu erstellen und deshalb auch weniger fehlerhaft sind als ihre imperativen Gegenstücke. Ein weiterer Grund ist, daß bei ihnen Syntax und Semantik sehr nahe zusammenliegen, denn dies impliziert, daß Eigenschaften meist einfacher zu verifizieren sind als im imperativen Fall.

Eine bekannte operationale Art, die Semantik von funktionalen Programmen zu definieren, verwendet Berechnungssequenzen $e_1, e_2, \ldots$ zur Beschreibung der Einzelschritte einer Auswertung. Dabei sind die e_i Terme und werden, ausgehend vom auszuwertenden Originalterm e_1, nacheinander dadurch erzeugt, daß man e_{i+1} aus e_i durch einen Expansionsschritt gefolgt von einem Vereinfachungsschritt gewinnt. Der Expansionsschritt ersetzt dabei gewisse Aufrufe einer Rechenvorschrift jeweils durch den Rumpf, wobei in diesem noch die formalen Parameter durch die jeweiligen Argumente substituiert werden. Welche Aufrufe zu ersetzen sind, wird durch eine Berechnungsregel (auch Parameterübergabemechanismus genannt) bestimmt; Beispiele sind Leftmost-innermost-Ersetzung (Call-by-value), Leftmost-outermost-Ersetzung (Call-by-name) und volle Ersetzung (Herbrand-Regel). Der an die Expansion sich anschließende Vereinfachungsschritt führt die Operationen der primitiven Datentypen und Fallunterscheidungen soweit wie möglich aus. Bezüglich der Details verweisen wir auf [6].

Die eben skizzierte Semantik wertet ein funktionales Programm imgrundegenommen dadurch aus, daß bedeutungserhaltende Transformationen angewendet werden. Sie ist nicht nur leicht verständlich, sondern auch einfach zu implementieren. Die Vorteile, die sich aus einem entsprechenden Computersystem ergeben, sind vielfältig. Dies gilt insbesondere, wenn man mittels moderner GUI-Toolkits Terme geeignet graphisch darstellt und dem Benutzer die volle Kontrolle über den Auswertungsprozeß überläßt. So ein System kann etwa dazu dienen, Programmauswertungen und die Auswirkungen der Berechnungsregeln zu verdeutlichen. Man kann damit aber auch Programmabläufe schrittweise nachvollziehen, z.B. um Schwachstellen und logische Fehler zu erkennen. Eine Visualisierung der Auswertung kann schließlich noch dazu verwendet werden, die Ideen zu erkennen, die dem durch das funktionale Programm implementierten (abstrakten) Algorithmus zugrunde liegen, sowie den durch sie verursachten Aufwand.

Die Vorteile einer interaktiven Auswertung funktionaler Programme wurden auch andererorts erkannt, und mittlerweile existieren einige Computersysteme zu diesem Zweck. In [3] wird beispielsweise das System MIRACALC beschrieben, welches die schrittweise Auswertung von Miranda-Programmen erlaubt. Ein weiteres solches System ist π-RED. Es unterstützt die interaktive Auswertung von funktionalen Programmen, welche in KiR (der "Kiel reduction language") geschrieben sind; siehe [4]. Auch für viele Lisp-Systeme existieren Debugger, die ein schrittweises Arbeiten erlauben. Abgesehen von einigen nicht-publizierten ad hoc Ansätzen ist dies jedoch nicht der Fall für alle bekannten Implementierungen von modernen funktionalen Sprachen wie Haskell, Miranda und ML.

In diesem Artikel beschreiben wir ein Computersystem, das es erlaubt, die Auswertung von funktionalen Programmen erster Stufe zu visualisieren, die in einer einfachen Teilsprache von ML geschrieben sind. Es heißt KIEL, was ein Akronym ist für "Kiel Interactive Evaluation Laboratory", und wurde speziell für den Unterricht im Informatik-Grundstudium an Universitäten konzipiert. Der Rest des Artikels ist wie folgt gegliedert: In Abschnitt 2 stellen wir die Teilsprache von ML vor, die das System unterstützt. Abschnitt 3 beschreibt zuerst KIEL überblicksartig und geht dann etwas genauer auf seine graphische Benutzeroberfläche ein. In Abschnitt 4 wird ein bekanntes Beispiel mit KIEL behandelt. Abschließende Bemerkungen findet man schließlich in Abschnitt 5.

2 Die von KIEL unterstützte Teilsprache von ML

Die von KIEL unterstützte Teilsprache von ML, im folgenden MlKiel genannt, entspricht in etwa dem Umfang, wie er in den Kapiteln 1 bis 5 von [9] beschrieben wird. Dieser soll nun skizziert werden. Wir verzichten auf eine formale Beschreibung. Die entsprechenden Syntaxdiagramme findet man in [8] und mit Ausnahme der Parameterübergabe, deren freie Wahl ja geradezu charakteristisch für KIEL ist, entspricht die Semantik aller Sprachkonstrukte der von ML.

Es gibt fünf primitive Typen in MlKiel, nämlich den einelementigen Typ `unit` und die Typen `bool`, `int`, `real` und `string` für die Wahrheitswerte, ganzen Zahlen, reellen Zahlen und Zeichenreihen. Jeder primitive Typ m ist ein "Typ mit Gleichheit" im Sinne von ML, was heißt, daß Gleichheitstest `=` : $m * m$ `->` `bool` und Ungleichheitstest `<>` : $m * m$ `->` `bool` definiert sind.

Die Darstellung aller Konstanten ist wie in ML. Der Typ `unit` besitzt also nur `()` als Konstante, die Wahrheitswerte werden mit `true` und `false` bezeichnet, die Konstanten des Typs `int` sind Ziffernfolgen, ev. mit einer vorangestellten unären Negation "~", die Konstanten des Typs `real` beginnen mit einem optionalen "~" gefolgt von einer Ziffernfolge und dann entweder dem Dezimalpunkt und einer Ziffernfolge oder dem Symbol `E`, einem optionalen "~" und wiederum einer Ziffernfolge und die Konstanten des Typs `string` sind in Anführungszeichen eingeschlossene Zeichenreihen. Im Gegensatz zu ML erlaubt MLKIEL in Zeichenreihen keine speziellen Zeichen wie "newline". Die Maximallänge der Zeichenreihen kann mittels einer Konfigurationsdatei festgelegt werden.

In KIEL werden sequentielle Konjunktion und Disjunktion als Operationen auf den Wahrheitswerten eingeführt und nicht als Schreibweisen für spezielle Fallunterscheidungen. Dies führt zu

`not : bool -> bool`	Negation
`andalso, orelse : bool * bool -> bool`	Konjunktion / Disjunktion

als den primitiven Operationen auf den Wahrheitswerten. Die primitiven Operationen auf den ganzen Zahlen, reellen Zahlen und Zeichenreihen bilden jeweils eine Teilmenge der entsprechenden ML-Operationen. Zur Behandlung von ganzen Zahlen bietet KIEL an:

`+, -, * : int * int -> int`	Arithmetik
`div, mod : int * int -> int`	Ganzzahl. Division mit Rest
`min, max : int * int -> int`	Minimum / Maximum
`abs : int -> int`	Absolutbetrag
`<, <=, >, >= : int * int -> bool`	Vergleiche

Die primitiven Operationen des KIEL-Systems für die reellen Zahlen sind:

`+, -, *, / : real * real -> real`	Arithmetik
`abs : real -> real`	Absolutbetrag
`sqrt : real -> real`	Quadratwurzel
`sin, cos : real -> real`	Sinus / Cosinus
`exp, ln : real -> real`	e-Funktion / natürlicher Log.
`real : int -> real`	Konvertierung
`truncate : real -> int`	Rundung nach unten
`<, <=, >, >= : real * real -> bool`	Vergleiche

Auf Zeichenreihen kann man schließlich die folgenden Operationen verwenden:

`^ : string * string -> string`	Konkatenation
`size : string -> int`	Länge
`ord : string -> int`	ASCII-Code Anfangsz.
`chr : int -> string`	Inversion ASCII-Code
`substring : string * int * int -> string`	Teilzeichenreihe
`<, <=, >, >= : string * string -> bool`	Vergleiche (lexikog.)

Listen sind in funktionalen Programmen ein extrem hilfreicher Datentyp. Deshalb stellt ML einen primitiven polymorphen Datentyp für Listen mit vielen Operationen zur Verfügung. In MlKiel gibt es (derzeit noch) keinen Polymorphismus. Wegen der Bedeutung der Listen haben wir uns jedoch dazu entschlossen, für jeden der obigen fünf Typen m den Typ m `list` für Listen mit Objekten des Typs m zu implementieren. Die Konstanten werden dabei wie in ML dargestellt, was heißt, daß die leere Liste mit `nil` oder `[]` bezeichnet wird und die Elemente einer nichtleeren Liste durch Kommata getrennt und mittels "`[`" und "`]`" geklammert werden. Neben dem Gleichheits- und Ungleichheitstest sind in MlKiel noch die folgenden Listenoperationen vorhanden:

`::` : $m * m$ `list -> ` m `list`	Links anfügen
`hd` : m `list -> ` m	Anfangselement
`tl` : m `list -> ` m `list`	Entfern. Anfangselement
`@` : m `list` $*$ m `list -> ` m `list`	Konkatenation
`length` : m `list -> int`	Länge
`null` : m `list -> bool`	Test auf leere Liste
`rev` : m `list -> ` m `list`	Spiegelung

Wie bei den Zeichenreihen kann auch die Maximallänge der Listen mittels der schon erwähnten Konfigurationsdatei festgelegt werden.

Ein Programm in MlKiel besteht aus einer Deklarationsfolge, wobei jede Deklaration entweder eine einzelne Rechenvorschrift oder ein System von (möglicherweise verschränkt-rekursiven) Rechenvorschriften definiert und mit einem Semikolon abgeschlossen ist. Die Deklaration einer einzelnen Rechenvorschrift geschieht wie in ML mit der Ausnahme, daß, wegen des fehlenden Polymorphismus, eine MlKiel-Rechenvorschrift vollständig typisiert sein muß. Dabei wird der Typ m eines formalen Parameters x innerhalb der Parameterliste in der Form x `:` m angegeben und der Resultattyp wird, ebenfalls mit dem Doppelpunkt als Trennzeichen, der Parameterliste nachgestellt. Insgesamt geschieht die Deklaration einer einzelnen Rechenvorschrift F mit formalen Parametern x_i des Typs m_i $(1 \leq i \leq k)$, Resultattyp m und Rumpf E in der folgenden Form:

$$\texttt{fun}\ F\ \texttt{(}x_1\ \texttt{:}\ m_1\texttt{,}\ \ldots\texttt{,}x_k\ \texttt{:}\ m_k\texttt{)}\ \texttt{:}\ m\ \texttt{=}\ E\ \texttt{;}$$

Dabei ist der Term E in der üblichen Weise aufgebaut mittels der formalen Parameter, der obigen Konstanten und Operationen, Aufrufen von schon deklarierten Rechenvorschriften, Fallunterscheidungen der Form `if` B `then` A_1 `else` A_2 und rekursiven Aufrufen von F. Die Anwendungen der Operationen werden wie in ML notiert mit der (aus methodischen Gründen) einzigen Ausnahme, daß, wenn verwendet, MlKiel grundsätzlich die klassische mathematische Notation F`(`A`)` für die Applikation verlangt und die in ML auch zulässigen Formen $F\,A$ oder `(`$F\,A$`)` verbietet.

Es sollte ausdrücklich erwähnt werden, daß die derzeitige Version von KIEL weder höhere funktionale Konstrukte noch Deklarationen von Datentypen, Muster in Rechenvorschriften und lokale Deklarationen erlaubt.

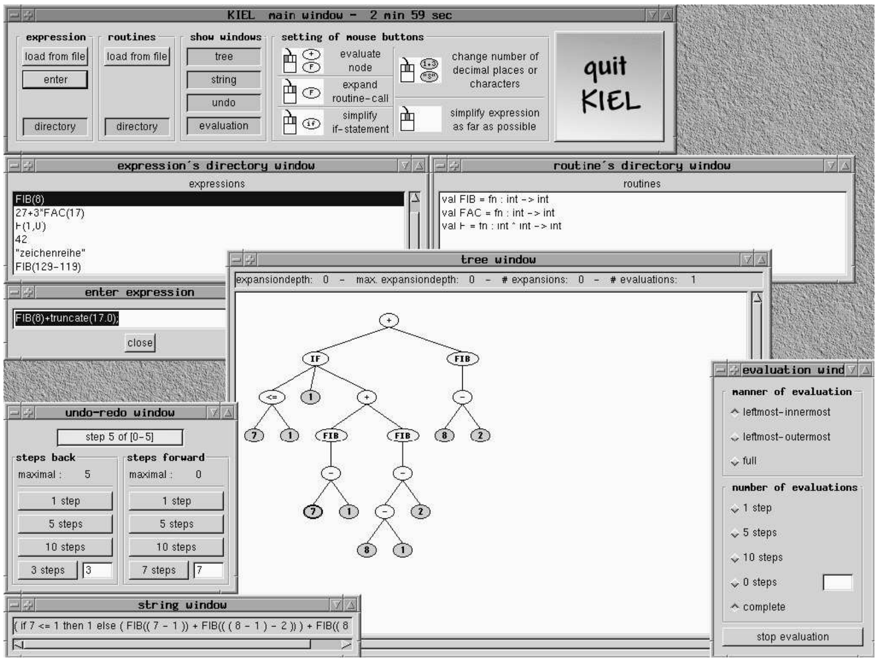
Nachfolgend ist ein einfaches Beispiel eines MLKIEL-Programms angegeben, wobei ein Aufruf von `sort` eine Liste von ganzen Zahlen sortiert. Dies geschieht unter Verwendung einer Hilfsrechenvorschrift `insort`, die ein Element in eine sortierte Liste einsortiert.

```
fun insort (y : int, w : int list) : int list =
  if null(w) then [y]
             else if y <= hd(w) then y :: w
                                else hd(w) :: insort(y,tl(w));
fun sort (s : int list) : int list =
  if null(s) then s
             else insort(hd(s),sort(tl(s)));
```

Die Syntax von Systemen von Rechenvorschriften stimmt mit der von ML überein. Dies heißt, daß das Schlüsselwort `fun` bei allen Rechenvorschriften mit Ausnahme der ersten weggelassen wird und `and` zur Trennung dient.

3 Struktur und Benutzerschnittstelle von KIEL

Das Computerssystem KIEL ist in C geschrieben und verwendet die Bibliotheken GTK, GDK und GLib für die graphische Benutzerschnittstelle. Obwohl der Benutzer die Oberfläche des Systems mittels einer Konfigurationsdatei in weiten Teilen selbst gestalten kann (vergl. [8] wegen der Details), zeigt das nachfolgende Bild eine typische Benutzeroberfläche mit den wichtigsten Fenstern.

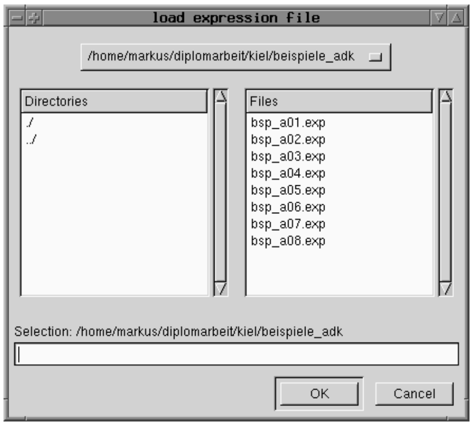


Aus der Sicht des Benutzers besteht KIEL, grob gesehen, aus zwei Komponenten. Die erste Komponente ist der Arbeitsbereich. Er enthält MlKiel-Programme und Terme über ihnen. Dabei ist angenommen, daß die Programme in Dateien mit der Endung "`.prg`" enthalten sind. Aus diesen werden sie in das System mit Hilfe des Fensters "load program file" geladen. Auch Terme können aus Dateien mit der Endung "`.exp`" in KIEL geladen werden; das entsprechende Fenster hat den Namen "load expression file". Sie werden jedoch üblicherweise direkt mit Hilfe des Fensters "enter expression" eingegeben.

Die zweite Komponente des Systems ist die Auswertungseinheit. Durch sie werden Terme des Arbeitsbereichs berechnet. Dies geschieht hauptsächlich mit Hilfe des Fensters "tree window". Wir haben diesen Namen gewählt, weil Terme graphisch als Bäume dargestellt werden. Die Auswertung kann schrittweise, semiautomatisch oder vollautomatisch erfolgen. Ersteres geschieht mittels einzelner Mausklicks im Fenster "tree window"; in den beiden anderen Fällen kommen noch zusätzliche Fenster ins Spiel, mit denen beispielsweise eine spezielle Berechnungsregel und die maximale Schrittanzahl festgelegt werden. Auf Details gehen wir später noch genauer ein.

Wir beschreiben nun die Benutzeroberfläche des KIEL-Systems etwas genauer und beginnen mit dem Hauptfenster "KIEL main window". Wie man der obigen Gesamtübersicht entnehmen kann, ist es konzeptuell in drei Teile gegliedert. Der linke Teil enthält drei Spalten von Bedienknöpfen, welche zum Laden bzw. zur Eingabe von Termen, zum Laden von Programmen und zum Öffnen bzw. Schließen der vier mit der Auswertung zu tun habenden Fenster dienen. Im mittleren Teil findet man fünf kleine Bilder, welche die verschiedenen Aktionen beschreiben, die durch das Drücken von Maustasten ausgelöst werden. Der rechte Teil des Hauptfensters besteht nur aus einem einzigen großen Knopf zum Beenden von KIEL.

Wenn man den Knopf "load from file" der linken Knopfspalte des Hauptfensters betätigt, so öffnet sich das Fenster "load expression file", mit dem man Terme aus Dateien ins System laden kann. Da dieses Fenster in der obigen Gesamtübersicht nicht geöffnet ist, zeigen wir in dem nachfolgenden Bild, wie es typischerweise aussieht:



Die linke Liste dieses Fensters enthält die Unterverzeichnisse und die rechte Liste die exp-Dateien des Verzeichnisses, in dem man sich gegenwärtig befindet. Mit der Maus ist es möglich, in ein anderes Verzeichnis zu wechseln oder eine Datei auszuwählen; der Ladevorgang wird durch Betätigung von "OK" ausgelöst. Üblicherweise erfolgt jedoch die Eingabe der auszuwertenden Terme unter Verwendung des Fensters "enter expression", welches durch den Knopf "enter" geöffnet wird und sich in der obigen Gesamtübersicht in der Mitte am linken Bildrand befindet. Der Knopf "directory" der linken Knopfspalte des Hauptfensters öffnet schließlich ein Fenster, welches die derzeitigen Terme des Arbeitsbereichs zeigt. Dieses Fenster sieht man in der obigen Gesamtübersicht direkt über dem Fenster "enter expression".

Das Laden von Programmen aus prg-Dateien ins System geschieht analog zum Laden von Termen mit Hilfe des schon erwähnten Fensters "load program file", das durch den Knopf "load from file" der mittleren Knopfspalte des Hauptfensters geöffnet wird. Der Knopf "directory" dieser Spalte hat für Programme die gleiche Bedeutung wie der entsprechende Knopf der linken Spalte für Terme.

Der primäre Zweck von KIEL ist die benutzerkontrollierte interaktive Auswertung von funktionalen Programmen. Dies geschieht mit Hilfe des Fensters "tree window", das sich öffnet, wenn man den Knopf "tree" im Hauptfenster betätigt. Die Auswahl des auszuwertenden Terms aus der Liste aller Terme des Arbeitsbereichs geschieht mit der Maus und stellt diesen anschließend im Fenster "tree window" als Baum dar. In der Gesamtübersicht ist ein Baum zu sehen, der sich während der Auswertung einer Rechenvorschrift FIB für die Fibonacci-Zahlen ergeben hat. Ein Knoten eines Baums steht für eine Konstante bzw. Operation aus Abschnitt 2, eine Fallunterscheidung, eine Rechenvorschrift oder einen Auswertungsfehler. Die nächsten vier Bilder zeigen für jede von diesen vier Klassen ein Beispiel. Vielleicht fallen dem Leser die unterschiedlichen Grauwerte auf; bei einem Farbbildschirm entsprechen diese vier unterschiedlichen Farben, die vom Benutzer mittels einer Konfigurationsdatei frei wählbar sind.

Normalerweise ist die Größe der Knoten fest, und damit werden von großen Zahlen und langen Zeichenreihen und Listen nur die Anfangsstücke angezeigt. Mit Hilfe der mittleren Maustaste kann jedoch die Größe der Knoten der Größe der Objekte angepaßt werden, für die sie stehen.

Wie man der obigen Gesamtübersicht auch entnehmen kann, wird der Term des Fensters "tree window" im Fenster "string window" zusätzlich noch in der gewohnten textuellen Darstellung angegeben. Dieses letztgenannte Fenster wird mittels des Knopfs "string" der rechten Spalte des Hauptfensters geöffnet.

Die semi- oder vollautomatische Auswertung eines (Teil-)Terms geschieht, indem man mit der Maus auf den entsprechenden Wurzelknoten zeigt und anschließend die linke Maustaste drückt. Bei diesen Vorgehensweisen ist es zusätzlich möglich, eine spezielle Berechnungsregel und die maximale Anzahl der Auswertungsschritte mittels des Fensters "evaluation window" einzustellen, das durch den untersten Knopf "evaluation" der rechten Knopfspalte des Hauptfensters geöffnet wird und sich in der Gesamtübersicht in der rechten unteren Ecke befindet. Die derzeitige Version von KIEL erlaubt die Auswahl zwischen genau den in der Einleitung genannten Berechnungsregeln, also zwischen Leftmost-innermost-Ersetzung, Leftmost-outermost-Ersetzung und voller Ersetzung.

Nachdem wir die semi- oder vollautomatische Auswertung von Termen mittels KIEL beschrieben haben, wenden wir uns nun deren schrittweiser Auswertung zu. Diese ist durch drei Aktionen möglich, nämlich Expandieren von Rechenvorschriftsaufrufen, Ausführen von Operationen von Abschnitt 2 und Vereinfachen von Fallunterscheidungen. Wie diese drei Aktionen angestoßen werden, wird durch die Bilder im mittleren Teil des Hauptfensters erklärt. Beispielsweise wird eine Expansion angestoßen, indem man mit der Maus auf den entsprechenden Rechenvorschriftsknoten zeigt und dann die rechte Maustaste drückt; man vergleiche mit dem mittleren Bild der linken Bilderspalte. Wie man aus der Gesamtübersicht ersehen kann, gibt es zusätzlich zu den eben aufgeführten drei Aktionen noch die Möglichkeit, den gesamten Term des Fensters "tree window" soweit wie möglich auszuwerten. Dazu hat man, wie das untere Bild der rechten Bilderspalte angibt, mit der Maus auf eine Position außerhalb des Baums zu zeigen und dann die mittlere Maustaste zu drücken. In Verbindung mit Expansionen, die jeweils in einer speziellen Weise durch die Auswertung des Gesamtterms mit eins als maximaler Schrittanzahl realisiert sind, kann man damit unmittelbar die operationale Semantik mittels Berechnungssequenzen visualisieren, die in der Einleitung skizziert wurde.

Schließlich sollte noch erwähnt werden, daß KIEL erlaubt, Auswertungen zu stoppen und Schritte rückgängig zu machen. Letzteres kann insbesondere benutzt werden, um durch eine Berechnungssequenz "zu wandern", beispielsweise zu Zwecken der Visualisierung oder um alternative Berechnungen zu testen. Das entsprechende "undo-redo window" wird durch die Betätigung des dritten Knopfs "undo" der rechten Knopfspalte des Hauptfensters geöffnet.

4 Ein Beispiel zur Anwendung von KIEL

In diesem Abschnitt verwenden wir eine sehr bekannte funktionale Rechenvorschrift, um zu demonstrieren, wie KIEL im Grundstudium eingesetzt werden kann, um Programmauswertungen zu visualisieren und die Unterschiede zwischen den Berechnungsregeln hervorzuheben.

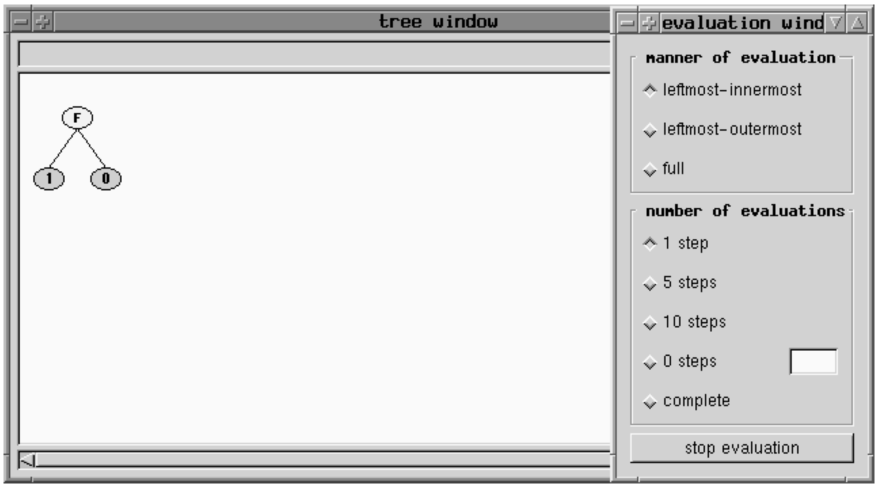
Die folgende geschachtelt-rekursive Rechenvorschrift wurde erstmals im Jahre 1968 von J.H. Morris im Rahmen seiner Dissertation [7] untersucht:

```
fun F (x : int, y : int) : int =
  if x <= 0 then 1
               else F(x-1,F(x-y,y));
```

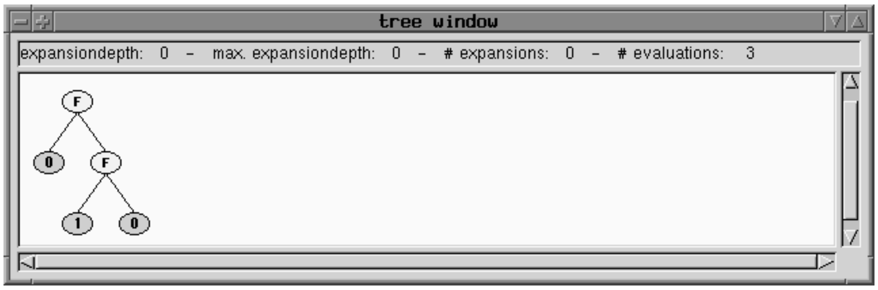
Seither ist bekannt, daß verschiedene Berechnungsregeln zu verschiedenen Resultaten führen können. Beispielsweise führt der Term `F(1,0)` zu einer unendlichen Berechnungssequenz, liefert also "undefiniert" als Resultat, wenn Leftmost-innermost-Ersetzung angewendet wird. Ist durch die Berechnungsregel hingegen

Leftmost-outermost-Ersetzung oder volle Ersetzung vorgeschrieben, so wertet sich `F(1,0)` zu `1` aus. Nachfolgend zeigen wir, wie man KIEL verwenden kann, um diesen Unterschied zu demonstrieren.

Wir starten mit dem auszuwertenden Term `F(1,0)` und wählen zuerst Leftmost-innermost als Ersetzungsstrategie und, da die Berechnung gemäß der in der Einleitung skizzierten Weise erfolgen soll, eins als maximale Schrittanzahl bei einer Auswertung. Das nachfolgende Bild zeigt die Fenster "tree window" und "evaluation window" von KIEL am Beginn der Auswertung:



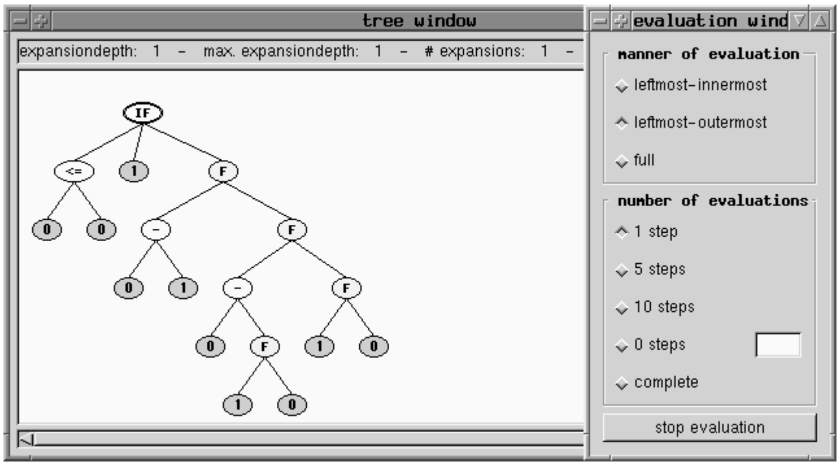
Die ersten zwei Schritte des Auswertungsprozesses bestehen aus einer Expansion des Aufrufs `F(1,0)` und einer nachfolgenden Vereinfachung des durch sie entstehenden Terms soweit wie möglich. Das Ergebnis ist `F(0,F(1,0))` und dieser Term wird durch KIEL wie folgt im Fenster "expression window" dargestellt:



Leftmost-innermost-Ersetzung bedeutet, daß von allen Aufrufen einer Rechenvorschrift, deren Argumente keine rekursiven Aufrufe mehr enthalten, der linkeste (im textuellen Aufschreibungssinne) gewählt wird. Im vorliegenden Fall haben wir also den unteren der beiden Aufrufe zu expandieren. Führen wir dies durch und vereinfachen wir anschließend wiederum soweit wie möglich, so präsentiert uns KIEL den folgenden Baum:

Schon nach diesen wenigen Schritten ist klar, daß eine Leftmost-innermost-Ersetzung zu einer unendlichen Berechnungssequenz führt.

Nun werten wir `F(1,0)` nach der Leftmost-outermost-Strategie aus. Dazu stellen wir zuerst diese Ersetzungsart ein und gehen dann an den Anfang der bisher aufgebauten Berechnungssequenz zurück. Von dort ausgehend sind die erste Expansion und Vereinfachung genau wie bei der Leftmost-innermost-Ersetzung. Dann verändert sich jedoch die Sachlage und wir zweigen im sogenannten Berechnungsdiagramm (siehe [6]) von der ersten Berechnungssequenz ab. Expandieren wir nämlich in `F(0,F(1,0))` den äußeren Aufruf von `F`, wählen also den oberen `F`-Knoten, so sehen wir in dem Fenster "tree window" den folgenden Baum (wobei, wegen der Einstellung, auch das "evaluation window" mit angeben ist):



Aufgrund der nichtstrikten Semantik der Fallunterscheidung liefert eine vollständige Vereinfachung die Konstante `1` und diese wird auch von KIEL berechnet:

Eine Besonderheit von KIEL ist, daß das System während einer semi- oder vollautomatischen Auswertung die Anzahl der Expansionen und Ausführungen von Operationen bzw. Fallunterscheidungen mitzählt. Dies erlaubt es insbesondere den Studierenden, die Effizienz von Berechnungsregeln zu bewerten. Intuitiv scheint klar zu sein, daß Leftmost-innermost-Ersetzung effizienter ist als Leftmost-outermost-Ersetzung, da sie viele redundante Berechnungen vermeidet. Jedoch weist schon J.H. Morris in [7] darauf hin, daß Leftmost-innermost-Ersetzung nicht immer überlegen sein muß. Und tatsächlich ist die auf ihn zurückgehende Rechenvorschrift `F` ein Gegenbeispiel. Die nachfolgende Tabelle zeigt einige Testergebnisse, die mittels KIEL erzielt wurden. In der zweiten und vierten Spalte sind dabei die zur Berechnung des Terms der ersten Spalte durchgeführten Expansionen unter Verwendung von Leftmost-innermost- bzw. Leftmost-outermost-Ersetzung angegeben und in der dritten und vierten Spalte die entsprechenden Ausführungen von Operationen auf den Typen.

	LI-Exp.	LI-Ausf.	LO-Exp.	LO-Ausf.
`F(5,1)`	63	125	6	21
`F(5,3)`	37	73	6	21
`F(5,5)`	33	65	6	21
`F(10,1)`	2047	4093	11	66
`F(10,5)`	1057	2113	11	66
`F(10,10)`	1025	2049	11	66

Wir haben auch einige Tests mit voller Ersetzung durchgeführt. Dabei wurden die Terme aber bald so riesig, daß beispielsweise auf einer modernen SUN ULTRA Workstation die Auswertung von `F(3,0)` mehr als 20 Minuten dauerte.

5 Abschließende Bemerkungen

In diesem Artikel wurde das Computersystem KIEL vorgestellt, das es erlaubt, funktionale Programme erster Stufe in ML-Syntax interaktiv auszuwerten, und es wurde seine Anwendung anhand eines bekannten Beispiels demonstriert. KIEL wurde für den Gebrauch im Informatik-Grundstudium entwickelt und ist primär vorgesehen für die Visualisierung von Programmauswertungen und die Auswirkungen von Berechnungsregeln. Zu einem gewissen Grad kann das System aber auch im Hauptstudium oder zur allgemeinen Programmierung verwendet werden, etwa um ML-Programme zu testen und/oder debuggen, sofern sie nach MlKiel reduzierbar sind.

KIEL kann auf allen SUN Workstations mit SUN-OS 5.x installiert werden, sowie auf allen PCs mit INTEL-Prozessoren und LINUX, Kernel 2.2.13. Es ist via FTP frei erhältlich vom Server ftp.informatik.uni-kiel.de, wo man es im Verzeichnis pub/kiel/progdev/KIEL findet. Zusätzliche Information wird mittels der Web-Seite http://www.informatik.uni-kiel.de/~progsys/kiel.html verbreitet. Durch diese Web-Seite hat man auch Zugang zu einer Online-Version von KIEL. Diese steht allen Benutzern des Internets als Java-Applet zur Verfügung und wurde, als beispielhafte Anwendung einer allgemeinen Methode, die in der Diplomarbeit [5] beschrieben ist, aus der in diesem Artikel beschriebenen C-Version entwickelt.

Seit der Fertigstellung von KIEL wurden mit dem System viele Fallstudien durchgeführt, u.a. auch bei einer Fortbildungsveranstaltung für Gymnasiallehrer. Dabei zeigte sich, daß das System eine große Hilfe ist bei einer Einführung in funktionale Programmierung (oder sogar in Informatik, wenn man den Ideen von [2] folgt). Aufgrund der gemachten positiven Erfahrungen glauben wir, daß eine Visualisierung von Programmauswertungen die Studierenden effizienter in die teils doch etwas verzwickte Materie der operationalen Semantik und Parameterübergabe einführt als Erklärungen an der Tafel oder in Lehrbüchern. Gleiches gilt etwa auch für Algorithmenentwurfsprinzipien (eine Visualisierung von Quicksort macht z. B. jedem sofort klar, wie das Prinzip "Teile und herrsche" funktioniert) und die Komplexität von Berechnungen (hierzu werte man etwa die Fibonacci-Rekursion mittels KIEL schrittweise aus).

Die Erfahrungen haben aber auch gezeigt, daß die Programmiersprache von KIEL zu restriktiv ist für fortgeschrittenere funktionale Programmiertechniken wie partielle Auswertung, Tabellierung, Supercompilation und Deforestation. Für die Zukunft ist deshalb geplant, MlKiel um Sprachkonstrukte wie Funktionsabstraktion, Deklaration von (rekursiven) Datentypen, Muster in Rechenvorschriften und lokale Objekte zu erweitern. Als ein erster Schritt in diese Richtung werden derzeit im Rahmen der Diplomarbeit [1] Deklarationen von (rekursiven) Datentypen und Muster in Rechenvorschriften in MlKiel integriert.

Literatur

1. Ernst H.: Erweiterung des KIEL-Systems um benutzerdefinierte Typen und Rechenvorschriften mit Mustern. Diplomarbeit, Inst. für Informatik und Praktische Mathematik, Universität Kiel (im Erscheinen)
2. Giegerich R., Hinze R., Kurtz S.: Straight to the heart of computer science via functional programming. In: Felleisen M., Hanus M., Thompson S. (eds.): Proc. FDPE '99, Report Rice COMP TR99-346, Dept. of Computer Science, Rice University, 1-5 (1999)
3. Goldson D.: A symbolic calculator for non-strict functional programs. The Computer Journal 37, 177-187 (1994)
4. Kluge W.E., Rathsack C., Scholz S.-B.: Using π-RED as a teaching tool for functional programming and program execution. In: Hartel P.H., Plasmeijer M.J. (eds.): Proc. FPLE '95, LNCS 1022, Springer, 231-250 (1995)
5. Lorenz O.: Vom C Programm zum Applet am Beispiel von KIEL. Diplomarbeit, Inst. für Informatik und Praktische Mathematik, Universität Kiel (2000)
6. Manna Z.: Mathematical theory of computation. McGraw-Hill (1974)
7. Morris J.H.: Lambda-calculus models of programming. Ph.D. Thesis, Mass. Inst. of Technology (1968)
8. Tiedt M.: Kiel interactive evaluation laboratory: Eine Arbeitsumgebung zur Visualisierung von Termersetzungssemantik. Diplomarbeit, Inst. für Informatik und Praktische Mathematik, Universität Kiel (1999)
9. Ullman J.D.: Elements of ML programming. Prentice Hall (1994)