Schlussbericht: Rationale Zahlen für IML

Christian Güdel, Peter Rudolf von Rohr

Compilerbau, HS 2015, Team GR

Abstract

Unsere Erweiterung für IML führt rationale Zahlen ein. Rationale Zahlen werden als Bruch von zwei Integern z (Zähler) und n (Nenner) dargestellt. Für den Nenner n ist die Zahl 0 nicht zulässig.

Ein Literal einer rationalen Zahl wird im Programmcode dargestellt als z/n. Zur Laufzeit werden die Brüche jeweils gekürzt, so dass z und n keine gemeinsamen Teiler grösser als 0 ausser 1 besitzen. Wobei der gemeinsame Teiler auch positiv sein muss!

Lexikalische Syntax

Die rationalen Zahlen werden als neuer Datentyp *Ratio* in IML eingebaut. Dazu wird die Aufzählung *Type* um den Wert *RATIO* erweitert.

Ein neues Literal für rationale Zahlen wird eingeführt:

Pattern	Beispiel Lexeme	Beispiel Token
`[0-9]+('*[0-9]+)*/[1-9]+('*[0-9]+)*`	1'337/42	LITERAL, RatioVal 1337 42

Durch die zusätzliche Einschränkung dass im Nenner keine führende 0 vorkommen kann, wird das Problem der Division durch 0 elegant gelöst. Als Kompromiss können Zahlen wie 0001 nicht im Nenner verwendet werden.

Als Konsequenz wird ein neues Attribut *RatioVal* eingeführt, welches zwei ganzzahlige Werte z und n akzeptiert.

Zusätzlich fügen wir folgende Schlüsselwörter ein:

Pattern	Token	Funktion
ratio	(TYPE, RatioType)	Deklaration einer Variable für eine rationale Zahl
num	(RATIOOPR, Numerator)	Zähler einer rationalen Zahl als *int* auslesen
denom	(RATIOOPR, Denominator)	Nenner einer rationalen Zahl als *int* auslesen
floor	(RATIOOPR, Floor)	Abrunden auf die grösste ganze Zahl, die kleiner oder gleich gross wie die rationale Zahl ist, als *int*
ceil	(RATIOOPR, Ceil)	Aufrunden auf die kleinste ganze Zahl, die grösser oder gleich gross wie die rationale Zahl ist, als *int*
round	(RATIOOPR, Round)	Runden auf die am nächsten liegende ganze Zahl, 0.5 wird auf 1 gerundet, als *int*.
asRatio	(TYPECOPR, RatioType)	Nachfolgende Expression wird als Ratio behandelt

Grammatikalische Syntax

Wir erweitern die Grammatik um die folgende Regeln:

```
monadicOpr ::= RATIOOPR | asRatio
```

Die Operatoren der Gruppe RATIOOPR haben die gleiche Präzedenz wie diejenigen der Gruppe der monadischen Operatoren (*monadicOpr*). Sie sind nicht assoziativ. Dies ergibt keinen Sinn, da die Operatoren einen anderen Rückgabe- als Eingabetyp aufweisen. Somit sieht die Operatorentabelle neu wie folgt aus:

	Präzedenz	Assoziativität
MONADICOPR (NOT ADDOPR **asRatio**)	4 (=höchste)	Keine
RATIOOPR **(num denom floor ceil round)**	**4**	**Keine**
MULTOPR (* divE modE)	3	Links
ADDOPR (+ -)	2	Links
RELOPR (< <= > >= = /=)	1	Keine
BOOLOPR (&? |?)	0 (= niedrigste)	Rechts

Deklaration:

```
const r1 : ratio;
const r2 : ratio;
const r3 : ratio;
const r4 : ratio;
```

Initialisierung:

```
r1 init := 4/3;
r2 init := 0/1;
r3 init := 24/24;
r4 init := 12/3;
```

Zugriff auf den Zähler:

```
debugout num r1; // Ausgabe: 4
debugout num r2; // Ausgabe: 0
debugout num r3; // Ausgabe: 1
debugout num r4; // Ausgabe: 4
```

Zugriff auf den Nenner:

```
debugout denom r1; // Ausgabe : 3
debugout denom r2; // Ausgabe : 1
debugout denom r3; // Ausgabe : 1
debugout denom r4; // Ausgabe : 1
```

Abrunden auf die nächstkleinere oder gleich grosse ganze Zahl:

```
debugout floor r1; // Ausgabe: 1
debugout floor r2; // Ausgabe: 0
debugout floor r3; // Ausgabe: 1
debugout floor r4; // Ausgabe: 4
```

Aufrunden auf die nächstgrössere oder gleich grosse ganze Zahl:

```
debugout ceil r1; // Ausgabe: 2
debugout ceil r2; // Ausgabe: 0
debugout ceil r3; // Ausgabe: 1
debugout ceil r4; // Ausgabe: 4
```

Runden auf die am nächsten liegende ganze Zahl:

```
debugout round r1; // Ausgabe: 1
debugout round r2; // Ausgabe: 0
debugout round r3; // Ausgabe: 1
debugout round r4; // Ausgabe: 4
```

Ausgabe einer relationalen Zahl mittels debugout:

```
debugout 51/10; // Ausgabe: 5.1
debugout 2/3    // Ausgabe: 0.(6) // ()-Klammern um den periodischen Teil
```

Ausgabe von rationalen Zahlen

Anhand des Beispiels mit der rationalen Zahl -281/280 wird gezeigt wie der Output dargestellt wird:

```
debugout -281/280 // Ausgabe: -1.003(571428)
```

Vorzeichen	Ganzzahlteil	Trennzeichen	Nichtperiodische Nachkommastellen	(	Perodische Nachkomastellen	)
-	1	.	003	(	571428	)

Vorzeichen

Zuerst werden Zähler und Nenner überprüft, ob sie negativ sind und falls ja jeweils negiert. Es wird immer mit positiven rationalen Zahlen bei der Ausgabe gerechnet. Am Schluss wird dann ein negatives Vorzeichen ausgegeben, sofern erforderlich. Sollten Zähler und Nenner negativ sein, wird der Bruch als positive Zahl ausgegeben. Im Beispiel hat der Zähler ein negatives Vorzeichen, also wird dieses dargestellt.

Ganzzahlteil

Der Ganzzahlteil der rationalen Zahl wird mit einer Division des Zählers durch den Nenner berechnet und dann dargestellt. Im Beispiel wird 281/280 gerechnet und ergibt 1. Weiter wird nur noch mit dem Bruch 1/280 gerechnet, da wir für die Nachkomastellenberechnung den Ganzzahlteil nicht mehr benötigen.

Trennzeichen

Das Trennzeichen «.», dass den Ganzzahlteil vom Nachkommastellenteil trennt, wird in jedem Fall ausgegeben. Auch für unser Beispiel wird dieses Trennzeichen dargestellt.

Nichtperiodische Nachkommastellen

Die nichtperiodischen Nachkommastellen werden nur dargestellt, sofern sie vorhanden sind. Um zu prüfen, ob eine rationale Zahl nichtperiodische Nachkommastellen hat, werden zwei Teilbrüche berechnet: Ein Teilbruch für den nichtperiodischen Nachkommastellenteil und ein Teilbruch für den periodischen Teil. Hat der nichtperiodische Nachkommastellenbruch den Wert 0, existiert kein nichtperiodischer Nachkommastellenteil.
Existiert überhaupt kein Nachkommastellenteil (werder periodisch noch nichtperiodisch), wird stadessen eine «0» ausgegeben. In unserem Beispiel ist der nichtperiodische Nachkommastellenteil «003».

Klammer «(»

Diese Klammer wird nur dargestellt, sofern ein periodischer Nachkommastellenteil existiert und kennzeichnet dessen Anfang. Existiert nur ein periodischer Nachkomastellenteil, folgt diese Klammer direkt nach dem Trennzeichen. In unserem Beispiel existiert ein periodischer Nachkommastellenteil, also wird diese Klammer dargestellt.

Periodische Nachkommastellen

Diese Nachkommastellen werden nur ausgegeben, sofern sie existieren. Zur Berechnung der periodischen Nachkommastellen wird der zweite Teilbruch hinzugezogen. Dieser periodische Teil der Ausgabe haben wir auf maximal 50 Stellen begrenzt. Hat der periodische Teil der rationalen Zahl mehr als 50 Stellen, wird dies mit drei Punkten nach der 50isten Stelle signalisiert. Dazu muss vor dem Berechnen der Zahlenfolge die Anzahl Nachkommastellen bestimmt werden.

Beispiel einer Ausgabe mit mehr als 50 periodischer Nachkommastellen:

```
debugout 378171782/1177818189
// Ausgabe: 0.(321078232219421090974508630211007889266006232478O4...)
```

In unserem Beispiel ist dieser Teil kleiner als 50 Stellen, also wird er vollständig als «571428» dargestellt.

Klammer «)»

Diese Klammer wird nur dargestellt, wenn ein periodischer Nachkommastellenteil existiert und schliesst diesen ab.

Kontext- und Typeinschränkungen

Die Operatoren der Gruppe *RATIOOPR* unterstützen als Argument nur solche vom Typ *RATIO*. Für die Operatoren der Gruppen *RELOPR*, *ADDOPR* und *MULTOPR* werden gemischte Argumenttypen aus *RATIO* und *INT* unterstützt. Dabei wird jeweils die *INT*-Seite in eine *RATIO*-Zahl konvertiert. Der Rückgabetyp von *ADDOPR* und *MULTOPR* bei gemischten Argumenttypen ist dann ebenfalls *RATIO*.

Die Zuweisung eines *RATIO*-Ausdrucks an eine *INT*-Variable führt zu einem Type-Error.

Codeerzeugung

Für die Codeerzeugung wurde der Datentyp der rationalen Zahlen durch eine Erweiterung der statischen Data-Klasse durch die Klasse RatioData eingeführt. Um mit diesem Datentyp umzugehen, wurden folgende neue Befehle für die VM eingebaut:

Befehl	Zweck
AddRatio	Addition von rationalen Zahlen
SubRatio	Subtraktion von rationalen Zahlen
MultRatio	Multiplikation von rationalen Zahlen
DivRatio	Division von rationalen Zahlen
CeilRatio	Aufrunden einer rationalen Zahl auf die kleinste grössere ganze Zahl
FloorRatio	Abrunden einer rationalen Zahl auf die grösste kleinere ganze Zahl
RoundRatio	Runden einer rationalen Zahl auf die nächstgelegene ganze Zahl
NumRatio	Auslesen des Zählers einer rationalen Zahl
DenomRatio	Auslesen des Nenners einer rationalen Zahl
EqRatio NeRatio GeRatio GtRatio LeRatio LtRatio	Relationale Operationen mit rationalen Zahlen
NegRatio	Umkehren des Vorzeichens einer rationalen Zahl

Befehl	Zweck
InputRatio OutputRatio	Input/Output-Operationen für rationale Zahlen
LoadImRatio	Ratio-Literal laden

Der Code wird von unserem in Haskell programmierten Compiler erzeugt und in eine Textdatei geschrieben. Die modifizierte Java IML-VM kann dieses Textfile einlesen und die entsprechenden Instruktionen ausführen.

Vergleich mit anderen Programmiersprachen

Common Lisp unterstützt den Datentyp *RATIO* und bietet ebenfalls Funktionen zur Bestimmung des Zählers (*numerator*) bzw. des Nenners (*denominator*).

In Haskell gibt es einen *Rational* Datentyp welcher ähnlich funktioniert.

Java unterstützt rationale Zahlen nicht direkt. Es gibt aber Bibliotheken, mit denen die Funktionalität nachgebildet werden kann[1,2].

Warum wurde die Erweiterung so entworfen und nicht anders?

Beim Entwurf der Erweiterung wurde darauf geachtet, dass der Datentyp *RATIO* möglichst konsistent mit den anderen Datentypen verwendet werden kann. Wir haben uns bei der Syntax an bestehenden Implementationen in Lisp[3] sowie Python[4] orientiert.

Durch die automatische Konvertierung bei gemischten Ausdrücken wird der Umgang mit rationalen Zahlen weiter vereinfacht.

Typenkonvertierungen bei denen Präzision verloren gehen könnte, müssen explizit mit *floor*, *ceil* oder *round* durchgeführt werden. Damit wird vom Programmierer eine explizite Entscheidung gefordert und implizite Annahmen werden verhindert.

Das Literal hat eine gewisse Ähnlichkeit mit einem Bruchstrich, so können rationale Zahlen schnell als solche erkannt werden.

[1] http://jscience.org/api/org/jscience/mathematics/number/Rational.html
[2] http://commons.apache.org/proper/commons-math/userguide/fraction.html
[3] https://en.wikipedia.org/wiki/Rational_data_type
[4] https://docs.python.org/3.1/library/fractions.html

Quellen

[1] http://www.arndt-bruenner.de/mathe/scripts/periodenlaenge.htm - Berechnung Periodenlänge

Zusammenarbeit und Arbeitsaufteilung

Wir haben nicht mit anderen Gruppen zusammengearbeitet. Die vorliegende Arbeit wurde jeweils gemeinsam von beiden Studierenden zusammen erarbeitet. Mit Ausnahme von Codeteilen, die während dem Unterricht abgegeben oder behandelt wurden, wurde kein fremder Code verwendet.

Ehrlichkeitserklärung

Hiermit bestätigen wir, dass die vorliegende Arbeit von uns selbstständig und ohne Benutzung anderer als der im Literaturverzeichnis angegebenen Quellen und Hilfsmittel erstellt wurde.

Ort, Datum, Unterschrift

Christian Güdel

Ort, Datum, Unterschrift

Peter Rudolf von Rohr

Anhang: IML Testprogramm

Liest die erreichten Punkte und die maximale Punktzahl ein und gibt die erreichte Note berechnet nach dem linearen Notenmassstab sowohl ungerundet als auch gerundet auf Zehntel aus:

```
program notenberechnung
global
   fun calculate(inppkt:int,inpmax:int) returns grade:ratio
   do
   grade := (round (inppkt * 5/1 divE inpmax + 1) * 10) divE (asRatio 10)
   endfun;

   fun calculateUnrounded(inppkt:int,inpmax:int) returns grade:ratio
   do
   grade := ((inppkt * 5/1 divE inpmax + 1) * 10) divE (asRatio 10)
   endfun;

   pkt:int;
   max:int

do
   debugin pkt;
   debugin max;

   debugout calculateUnrounded(pkt, max);
   debugout calculate(pkt, max)
endprogram
```