Mikroarchitekturen

Peter Marwedel
Informatik 12
TU Dortmund

2014/05/02

3.2.4 Gleitkomma-Operationen System-Aspekte

Verwendung von Gleitkomma-Arithmetik in höheren Programmiersprachen?

Keinerlei Aussagen im IEEE-Standard.

Grund: Im IEEE-Gremium kein Compilerbauer.

Zwischenrechnungen mit welcher Genauigkeit?

Intuitive Ansätze:

1. Für alle Zwischenrechnungen maximal verfügbare Genauigkeit:

Unerwartete Ergebnisse, Beispiel:

Sei **`q`** Variable einfacher Genauigkeit (32 Bit).

```
q = 3.0/7.0; print(q==(3.0/7.0))
```

führt zum Ausdruck von **`false`**,

Bei Zuweisung von **`3.0/7.0`** zu **`q`** gehen Mantissenstellen verloren.

Wird der Vergleich (als “Zwischenrechnung”) mit doppelter Genauigkeit ausgeführt, so müssen Mantissenstellen von **`q`** mit Nullen oder Einsen aufgefüllt werden.

Zwischenrechnungen mit welcher Genauigkeit?

2. Für alle Zwischenrechnungen das Maximum der Genauigkeiten der Argumente.

Unnötiger Verlust bekannter Information.

Beispiel: Seien **`x,y`**: Variablen einfacher Genauigkeit;

sei **`dx`** Variable doppelter Genauigkeit.

Betrachte: **`dx = x - y`**

Subtraktion: einfache Genauigkeit.

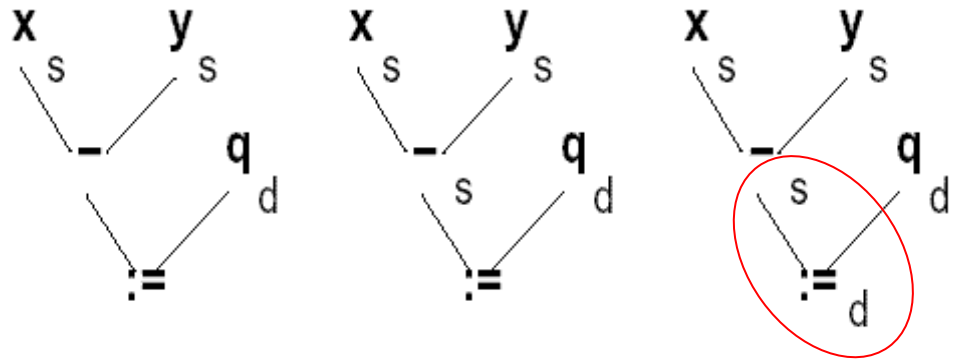
`dx` würden im Prinzip bekannte Mantissenstellen nicht zugewiesen.

Lösung des Problems (1)

Im Compiler zwei Durchläufe durch Ausdrucksbaum:

1. **Durchlauf von den Blättern zur Wurzel**
   Für jede arithmetische Operation wird das Maximum der Genauigkeiten der Argumente gebildet.
   Genauigkeit einer Zuweisung = Genauigkeit der Zielvariablen.
   Genauigkeit von Vergleichen: in der Regel das Minimum der Genauigkeit der Argumente.
   Ausdrucksbaum kann jetzt inkonsistent sein.

Beispiel: **`s`**: einfache, **`d`** doppelte Genauigkeit.



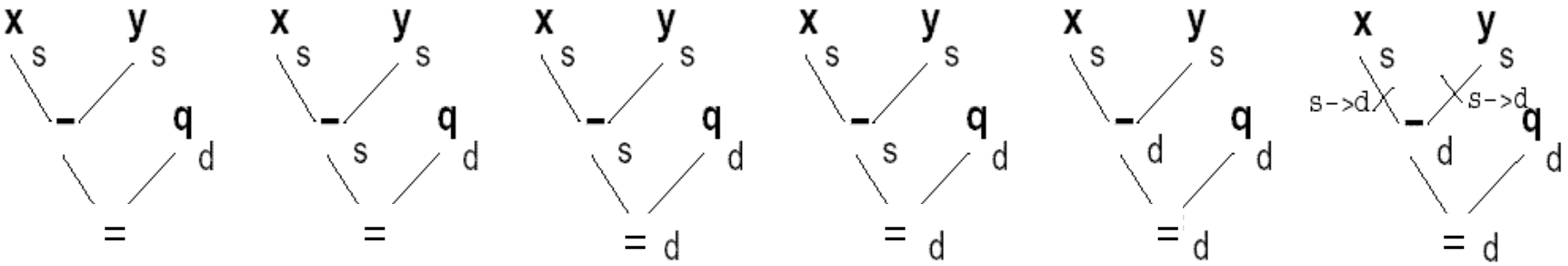
Lösung des Problems (2)

2. Durchlauf von der Wurzel zu den Blättern

Genauigkeit ↓, wenn Ergebnis nicht in der bisherigen Genauigkeit benötigt.

Genauigkeit ↑, wenn Ergebnis in größerer Genauigkeit benötigt wird.

Beispiel: **s**: einfache, **d** doppelte Genauigkeit.



2 Konvertierungen.

Verbleibende Probleme

- Die Genauigkeit ist nur im Kontext zu ermitteln.
- Der mögliche Fehler eines Ergebnisses ist unbekannt.

Alternative (Kulisch): **Intervallarithmetik**

Für alle Berechnungen werden die Intervallgrenzen der möglichen Werte betrachtet.
In verschiedenen Paketen angeboten.
☞ sehr umfangreicher Artikel bei Wikipedia.

Unzulässige Optimierungen

Viele zunächst korrekt erscheinende Compiler-„Optimierungen” sind falsch.

Beispiele:

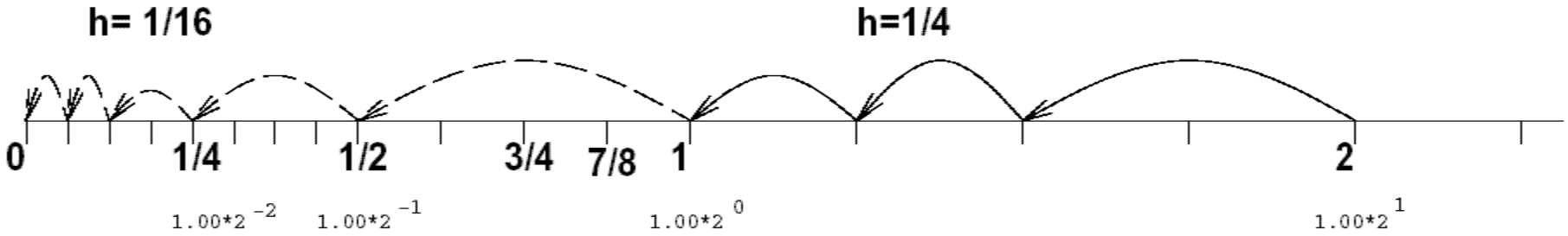
Ausdruck	unzulässige Optim.	Problem
x / 10.0	0.1 * x	x nicht exakt darstellbar
$x*y-x*z$	$x*(y-z)$	Falls $y \approx z$
$x+(y+z)$	$(x+y)+z$	Rundungsfehler
konstanter Ausdruck	Result. Konstante	Flags werden nicht gesetzt
gemeinsamer Ausdruck	Ref. auf 1. Berechnung	Rundungsmodus geändert?

Mögliches Verhalten bei Subtraktion von konstanten Werten bei Vergleichsoperatoren

```
eps = 1;
while (eps>0)
{h = eps;
eps = eps*0.5;
}
```

```
eps = 1;
while ((eps+1)>1)
{h = eps;
eps = eps*0.5;
}
```

Zwei Versionen der WHILE-Schleifen:



Die linke Schleife liefert bei 32 Bit-Gleitkommaarithmetik mit Realisierung nicht-normalisierter Zahlen den Wert $2^{-126}*2^{-23} \sim 1{,}4*10^{-45}$, bei Beschränkung auf normalisierte Zahlen den Wert $2^{-126} \sim 1{,}1*10^{-38}$.

Unproblematische Optimierungen

Lediglich einige sehr einfache Optimierungen sind unproblematisch:

Ausdruck	Optimierte Fassung
$x+y$	$y+x$
$2*x$	$x+x$
$1*x$	x
$x/2.0$	$x*0.5$

Zusammenfassung

- Vorzeichenbehaftete Zahlen (integer)
  - Überläufe
  - Multiplikation mit dem *Booth*-Algorithmus
- Gleitkomma-Zahlen
  - Bestimmung der Genauigkeit von Zwischenrechnungen
  - Zulässige und unzulässige Optimierungen