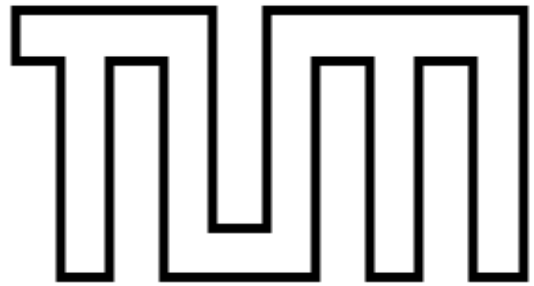


FAKULTÄT FÜR INFORMATIK

DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Multiple Precision Floating Point Arithmetic in Isabelle/HOL

Fabian Hellauer

Supervisor: Prof. Tobias Nipkow
Advisor: Fabian Immler

Generelles Ziel

Bibliotheken vergrößern!

Grundlagen

Umgang mit Rundungsfehlern

a) Analysieren/Entscheiden
b) Vermeiden
c) „Floating Point Expansions“

Aufgabenstellung

- Bereitstellung einer neuen Arithmetik

  „Multiple Precision Float Arithmetic“
- Benutzung des Ansatzes „Floating Point Expansion“
- Aufbau auf *IEEE_Floating_Point* aus dem AFP
- Code-Generierung anpassen und prüfen

Notation

- In *IEEE_Floating_Point*:

  Verwendung von +, -,… als Symbol

- Neue Notation:

  Verwendung von $\oplus$, $\ominus$,… für IEEE-Operationen

```
abbreviation round_affected_plus :: "float ⇒ float ⇒ float" (infixl "⊕" 65) where
  "round_affected_plus a b ≡ a + b"
```

„Floating Point Expansion“

- Schritt 1: Rundungsfreie Version von $\oplus$ und $\ominus$

  $\longrightarrow$ Speicherung des Rundungsfehlers
- Schritt 2: Verwaltung einer Liste akkumulierter Fehler
- Schritt 3: Weiterrechnen unter Berücksichtigung dieser Fehler

  $\longrightarrow$ Neue Operationen sind dann auch rundungsfrei möglich

Schritt 1a: Berechnung des Fehlers (Addition)

- schon bekannt (Ole Møller 1965)

```
definition TwoSum :: "float ⇒ float ⇒ float × float" where
  "TwoSum a b = (let
    x = a ⊕ b;
    bᵥ = x ⊖ a;
    aᵥ = x ⊖ bᵥ;
    bᵣ = b ⊖ bᵥ;
    aᵣ = a ⊖ aᵥ;
    y = aᵣ ⊕ bᵣ
    in (x, y))"
```

⟶ Berechnung von y, sodass a + b = x + y und x = a $\bigoplus$ b

Schritt 1b: Formalisierung der Aussagen

```
lemma TwoSum_correct1: "TwoSum a b = (x, y) ⟹ x = a ⊕ b"
  by (auto simp: TwoSum_def Let_def)

lemma TwoSum_correct2:
  fixes a b x y :: float
  assumes "Finite a"
  assumes "Finite b"
  assumes "Finite (a ⊕ b)"
  assumes out: "(x, y) = TwoSum a b"
  shows "Val a + Val b = Val x + Val y"
  sorry
```

Schritt 2: Speicherung der Fehler in einer Liste

- Darstellung des exakten Werts durch mehrere *floats*
- *float list* als Datenformat prinzipiell geeignet
- Verschiedene Optimierungen möglich
- Vorgehen nach Shewchuk:

  Approximation in der ersten Komponente
- Problem: Liste könnte leer werden

  ⟶ Festlegung als nicht-leere Liste

```
type_synonym mpf = "float × float list"
```

Schritt 3a: Algorithmen zum Weiterrechnen

- Notwendigkeit: Rundungsfreies Hinzufügen eines IEEE-*float*s

  ⟶ grow-mpf-Algorithmus

- Mehrfache Ausführung dieser Hinzufüge-Operation für alle Komponenten eines zweiten *mpf*s

  ⟶ Addition innerhalb der *mpf*s

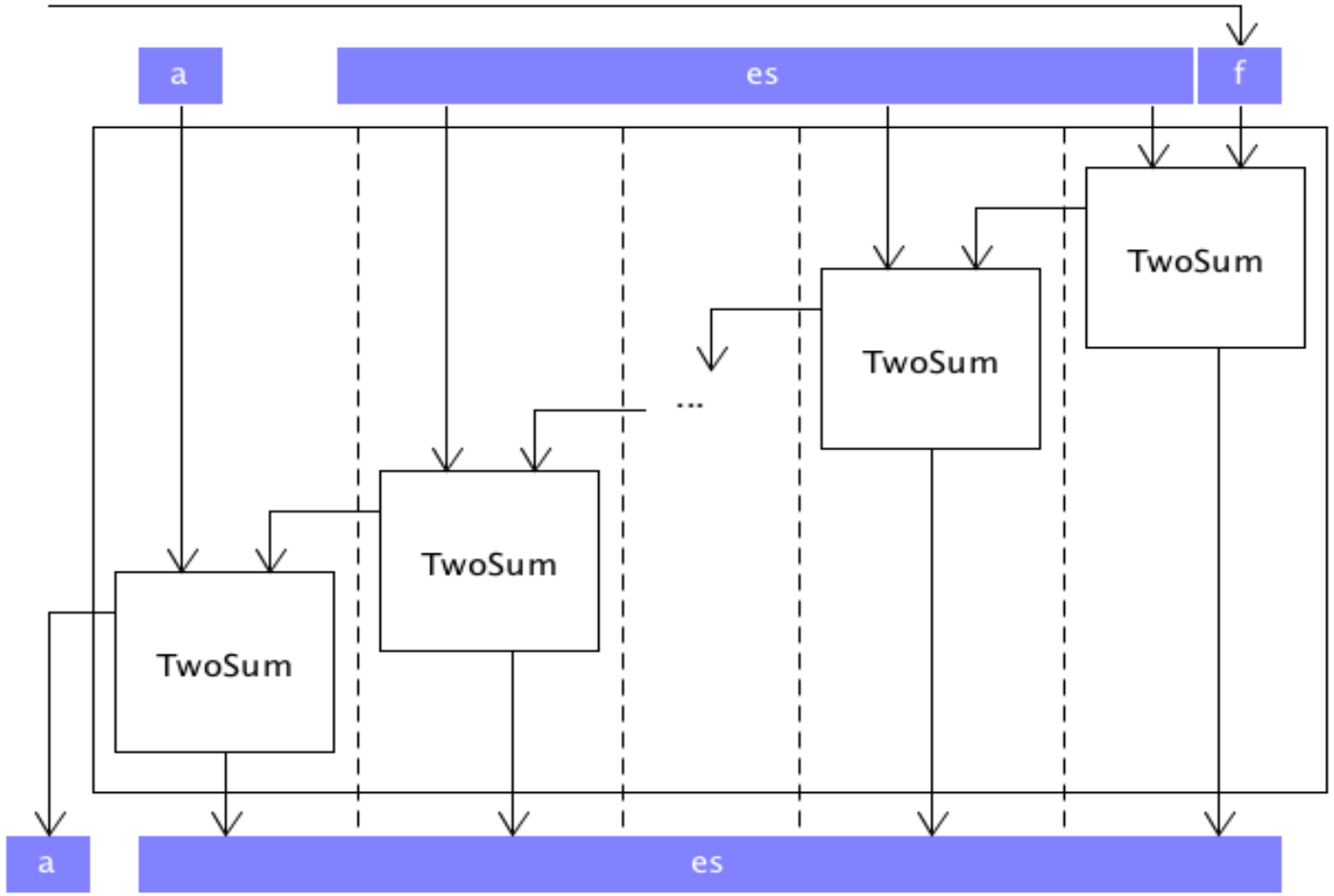
Schritt 3b: grow-mpf

- Erste Idee: Neuen *float*-Wert einfach an Liste anhängen
- Problem: Liste nach jeder *mpf*-Operation länger

  $\longrightarrow$ Aussage über *mpf*-Wert schlecht möglich
- Also: Zusammenführen/Auslöschen von Fehlern mittels *TwoSum* – rundungsfrei

  (Berechnung *TwoSum: float* y, sodass a + b = x + y und x = a $\oplus$ b)

Schritt 3b: grow-mpf



float f wird zum *mpf* (a, es) addiert.

Korrektheitsbeweise: Aussagen über *grow-mpf*

- *grow-mpf* führt nur *TwoSum*-Operationen auf *float*s im *mpf* aus

  $\longrightarrow$ nur fehlerfreie Transformationen

- Ausweitung der Rundungsfreiheit auf gesamte *grow-mpf*-Prozedur auch in Isabelle möglich

Korrektheitsbeweise: Behandlung von Spezialfällen

- Auftreten von Überlauf bei IEEE *float*s
- Nach Überlauf ($\pm\infty$): Ergebnis von Addition/Subtraktion keine endliche Zahl mehr

  $\longrightarrow$ Möglichkeit zur Aussage: Wenn Ergebnis-mpf endlich, dann…
- Fakt aber in *IEEE_Floating_Point* nicht gegeben
- Unbekannte Schwierigkeit eines Beweises

  $\longrightarrow$ Stattdessen: Endlichkeit der Zwischenergebnisse aktiv sicherstellen

Korrektheitsbeweise: Benutzung von *option*

- Also: *safe_TwoSum*

```
definition "safe_TwoSum a b =
  (let r = TwoSum a b in
    if Finite (fst r) ∧ Finite (snd r)
    then Some r
    else None)"

lemma safe_TwoSum_correct2:
  assumes "Finite a" "Finite b" "Finite (a ⊕ b)"
  assumes out: "safe_TwoSum a b = Some (x, y)"
  shows "Val a + Val b = Val x + Val y"
using assms
by (auto intro!: TwoSum_correct2 simp: safe_TwoSum_def Let_def split: split_if_asm)
```

grow-mpf in Isabelle

Korrektheitsbeweis: *grow-mpf*

```
lemma preserve_finite:
  assumes "safe_grow_mpf_rec mpf x = Some r"
  assumes "Finite x" "Finite_mpf mpf"
  shows "Finite_mpf r"
using assms
proof (induction mpf x arbitrary: r rule: safe_grow_mpf_induct)
--‹The base case is the case where the mpf is a single float with an empty error-list:›
case (no_error a f)
--‹We apply the definition of @{const safe_grow_mpf_rec}:›
from no_error.prems(1) have "do {(x, y) ← safe_TwoSum f a; Some (x, [y])} = Some r"
    unfolding safe_grow_mpf_rec.simps(1) .
--‹Since we required the result to be some value, we can give it a name:›
  then obtain x y where xy: "safe_TwoSum f a = Some (x, y)" and r: "r = (x, [y])"
    by (auto simp: bind_eq_Some_conv)
--‹and then delegate to the corresponding property of @{const safe_TwoSum}:›
  moreover from safe_TwoSum_finite[OF xy]
    have "Finite x" "Finite y".
  ultimately show ?case
    by simp
next
```

Mögliche Optimierungen

- Nutzung von *FastTwoSum*
- Endrekursion
- Nutzung von fold
- Generalität
- Verschiedene Möglichkeiten bei der Addition

Schwierigkeiten beim Testen

- Fehlende Übersetzung: SML-floats in HOL-Terme und zurück
- SML-Berechnungen in polyML fehlerhaft
- In vielen Sprachen: Anzeige von floats als gerundete Dezimalzahl

  $\longrightarrow$ unpräzise Darstellung

Lösung: Nutzung von *Float.float*

Ergebnisse

- Praktisch-orientierte Analyse von Shewchuks Algorithmen
- Umsetzung: Shewchuks Algorithmen in Isabelle/HOL
- Ansätze/Lösungen für formale Verifikationen
- Spezifikation eines Datenformats
- Neue Möglichkeit für rundungsfreie Addition/Subtraktion in Isabelle
- Anregung einer Korrektur von polyMLs IEEE-Berechnungen
- Unklarheit in der Code Generierung aus *Float.float* transparent gemacht

Ausblick - Fortführung

- Korrektheitsbeweis von *TwoSum* in Isabelle
- Weitere Aussagen über *mpfs*
  - „nonoverlapping“-Eigenschaft (Shewchuk)
  - Maximale Länge der Fehlerliste
- Mehr Zielsprachen für float-Code

Vielen Dank

Vielen Dank!