



FAKULTÄT FÜR INFORMATIK

DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

**Multiple Precision Floating Point
Arithmetic in Isabelle/HOL**

Fabian Hellauer

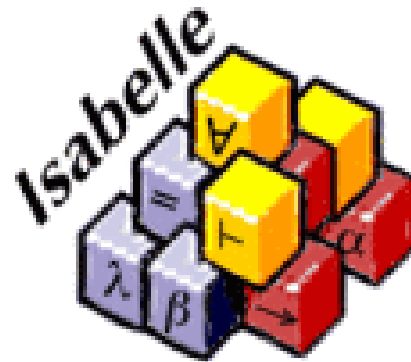
Supervisor: Prof. Tobias Nipkow

Advisor: Fabian Immler

Motivation

Numerik für:

- Dynamische Systeme
- Geometrie

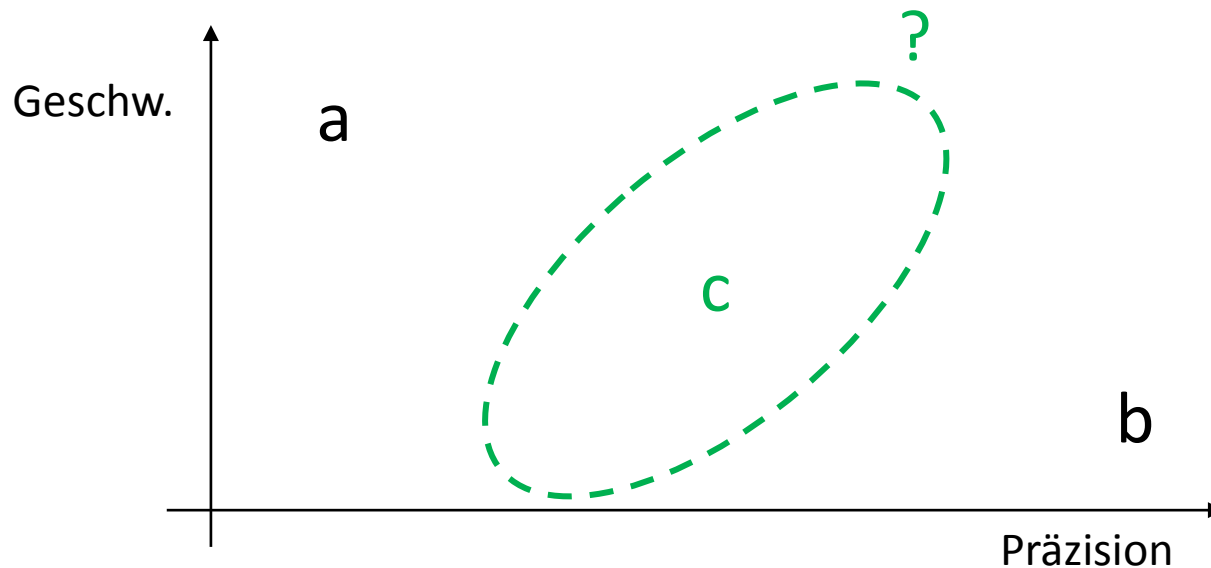


Verifikation!

Grundlagen

Umgang mit Rundungsfehlern

- a. Numerisch Analysieren/Entscheiden
- b. Vermeiden
- c. „Floating Point Expansions“



Aufgabenstellung

- Bereitstellung einer neuen Arithmetik
„Multiple Precision Float Arithmetic“
- Benutzung des Ansatzes „Floating Point Expansion“
nach Shewchuk, Joldes und Priest
- 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  $\Rightarrow$  float  $\Rightarrow$  float" (infixl " $\oplus$ " 65) where  
  "round_affected_plus a b  $\equiv$  a + b"
```

„Floating Point Expansion“

- Schritt 1: Rundungsfreie Version von \oplus und \ominus
 - Berechnung des Rundungsfehlers
- Schritt 2: Verwaltung einer Liste akkumulierter Fehler
- Schritt 3: Weiterrechnen unter Berücksichtigung dieser Fehler
 - Neue Operationen sind dann auch rundungsfrei möglich

Schritt 1a: Berechnung des Fehlers (Addition)

- schon bekannt (Ole Møller 1965)

```
definition TwoSum :: "float  $\Rightarrow$  float  $\Rightarrow$  float  $\times$  float" where
  "TwoSum a b = (let
    x = a  $\oplus$  b;
    bv = x  $\ominus$  a;
    av = x  $\ominus$  bv;
    br = b  $\ominus$  bv;
    ar = a  $\ominus$  av;
    y = ar  $\oplus$  br
  in (x, y))"
```

→ Berechnung von **y**, sodass **a** + **b** = **x** + **y** und **x** = **a** \oplus **b**

Schritt 1b: Formalisierung der Aussagen

```
lemma TwoSum_correct1: "TwoSum a b = (x, y)  $\implies$  x = a  $\oplus$  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  $\oplus$  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

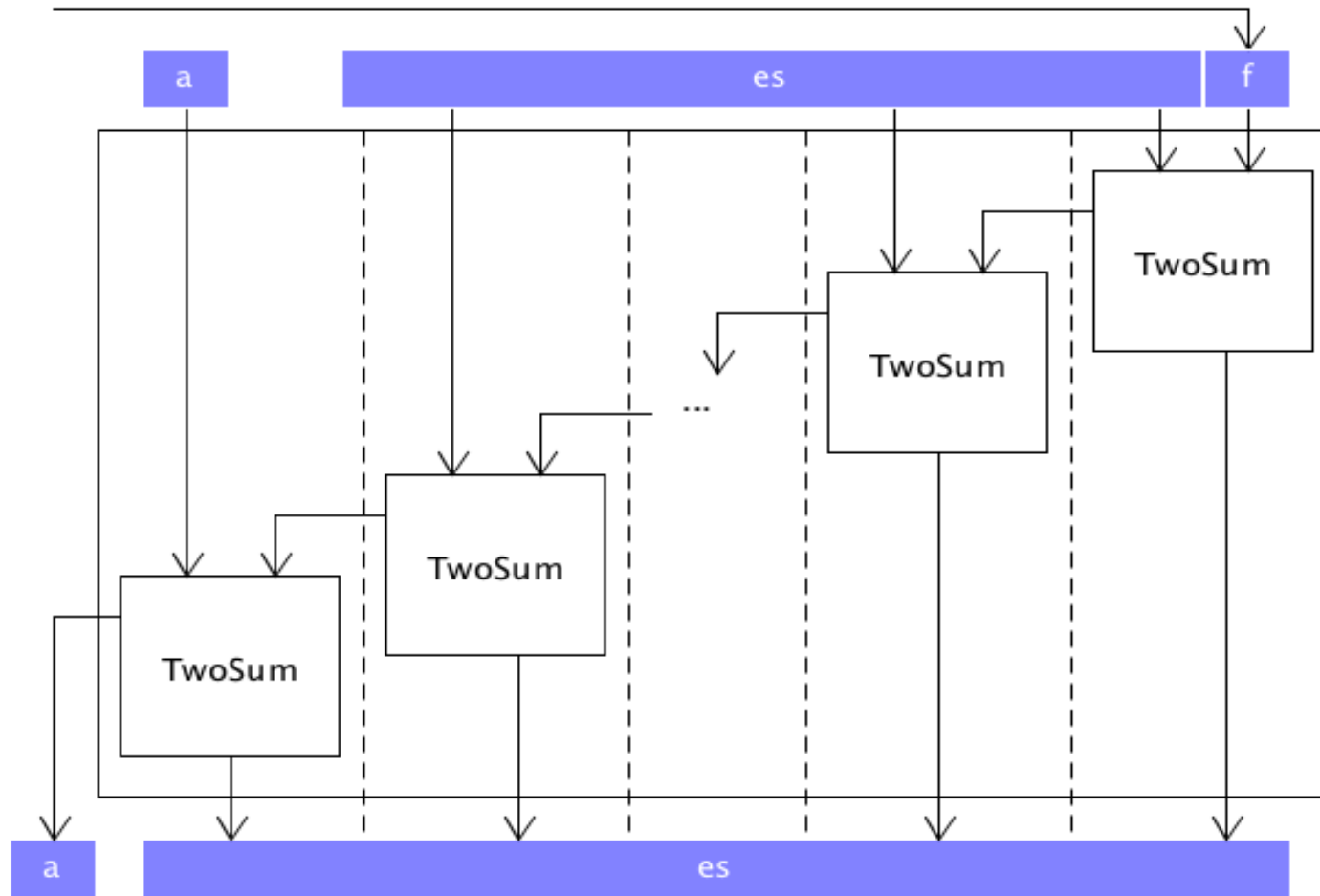
- Darstellung des exakten Werts als Summe von *floats*
- *float list* als Datenformat prinzipiell geeignet
- Verschiedene Einschränkungen nötig
 - Approximation in der ersten Komponente (J.R. Shewchuk)
 - nicht-leere Liste

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

Schritt 3: Algorithmen zum Weiterrechnen

- Notwendigkeit: Rundungsfreies Hinzufügen eines IEEE-*floats*
→ grow-mpf-Algorithmus
- Mehrfache Ausführung dieser Hinzufüge-Operation für alle Komponenten eines zweiten *mpfs*
→ Addition innerhalb der *mpfs*

Schritt 3: grow-mpf

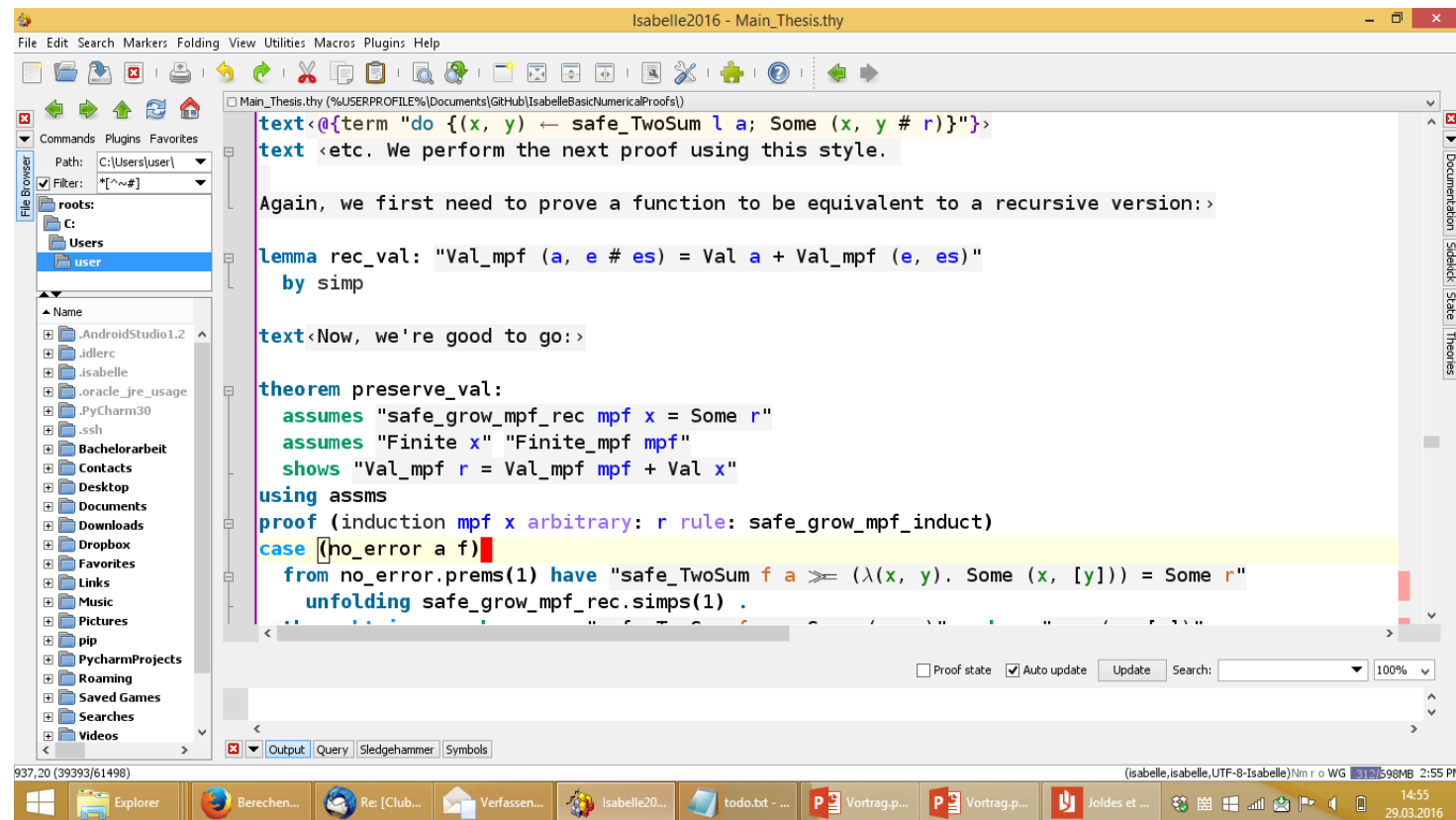


float **f** wird zum *mpf* (**a**, **es**) addiert.

grow-mpf in Isabelle

Korrektheitsbeweise: Aussagen über *grow-mpf*

- *grow-mpf* führt nur *TwoSum*-Operationen auf *floats* im *mpf* aus
→ nur fehlerfreie Transformationen
- Korrektheit:



The screenshot shows the Isabelle2016 IDE with a file named 'Main_Thesis.thy'. The left sidebar displays a file browser with a tree view of the user's file system. The main editor window contains the following text and code:

```
text<@term "do {(x, y) ← safe_TwoSum l a; Some (x, y # r)}">
text <etc. We perform the next proof using this style.>

Again, we first need to prove a function to be equivalent to a recursive version:>

lemma rec_val: "Val_mpf (a, e # es) = Val a + Val_mpf (e, es)"
  by simp

text<Now, we're good to go:>

theorem preserve_val:
  assumes "safe_grow_mpf_rec mpf x = Some r"
  assumes "Finite x" "Finite_mpf mpf"
  shows "Val_mpf r = Val_mpf mpf + Val x"
  using assms
  proof (induction mpf x arbitrary: r rule: safe_grow_mpf_induct)
  case (ho_error a f)
    from no_error.premis(1) have "safe_TwoSum f a ≈ (λ(x, y). Some (x, [y])) = Some r"
    unfolding safe_grow_mpf_rec.simps(1) .
```

The bottom status bar shows the file path '(isabelle,isabelle,UTF-8-Isabelle)Nm r o WG' and the date '29.03.2016'.

Korrektheitsbeweise: Behandlung von Spezialfällen

- Auftreten von Überlauf bei IEEE *floats*
- Nach Überlauf ($\pm\infty$): Ergebnis von Addition/Subtraktion keine endliche Zahl mehr
 - Möglichkeit zur Aussage: Wenn Ergebnis-mpf endlich, dann...
- Fakt aber in *IEEE_Floating_Point* nicht gegeben
 - 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)  $\wedge$  Finite (snd r)  
    then Some r  
    else None) "
```

```
lemma safe_TwoSum_correct2:
```

```
  assumes "Finite a" "Finite b" "Finite (a  $\oplus$  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)
```

Schwierigkeiten beim Testen

- Fehlende Übersetzung: SML-floats in HOL-Terme und zurück
- SML-Berechnungen in polyML fehlerhaft
 - Genauigkeit der Berechnung nicht vorhersehbar
- In vielen Sprachen: Anzeige von floats als gerundete Dezimalzahl
 - unpräzise Darstellung

Lösung: Nutzung von *sw_float* (aus `Library/Float.thy`)
als Referenzimplementierung

Ergebnisse

- 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
 - Klare 64bit-Semantik

Ausblick - Fortführung

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

Vielen Dank



Vielen Dank!