Bachelorarbeit

Ein Computeralgebrasystem in Rust

Verfasser	Bernd Haßfurther <nerglom@posteo.de>
Matrikel-Nr.	4372280
Betreuerin	Prof. Dr. Lena Oden
Datum	29. August 2022

Inhaltsverzeichnis

Was ist ein CAS?

- Mathematische Ausdrücke mit Variablen darstellen

Was ist ein CAS?

- Mathematische Ausdrücke mit Variablen darstellen
- Resistent gegen Ungenauigkeiten

Was ist ein CAS?

- Mathematische Ausdrücke mit Variablen darstellen
- Resistent gegen Ungenauigkeiten
- Einsatz in verschiedenen Gebieten

Was ist ein CAS?

- Mathematische Ausdrücke mit Variablen darstellen
- Resistent gegen Ungenauigkeiten
- Einsatz in verschiedenen Gebieten
- SymPy als konkrete Implementierung

(vgl. [2] [5, S. 1])

Grundregeln und Annahmen

- Zahlenräume $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$

Grundregeln und Annahmen

- Zahlenräume $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$
- Jede Subtraktion ist eine Addition

Grundregeln und Annahmen

- Zahlenräume $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$
- Jede Subtraktion ist eine Addition
- Jede Division ist entweder eine rationale Zahl oder eine Multiplikation

Grundregeln und Annahmen

- Zahlenräume $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$
- Jede Subtraktion ist eine Addition
- Jede Division ist entweder eine rationale Zahl oder eine Multiplikation
- Wurzeln können als Potenzen dargestellt werden

(vgl. [3, S. 23 ff.] [8, S. 2])

Funktionsumfang der Implementierung

- Addition von Zahlen und Symbolen

Funktionsumfang der Implementierung

- Addition von Zahlen und Symbolen
- Multiplikation von Zahlen und Symbolen

Funktionsumfang der Implementierung

- Addition von Zahlen und Symbolen
- Multiplikation von Zahlen und Symbolen
- Potenzregeln in Hinblick auf Genauigkeit auswerten

Funktionsumfang der Implementierung

- Addition von Zahlen und Symbolen
- Multiplikation von Zahlen und Symbolen
- Potenzregeln in Hinblick auf Genauigkeit auswerten
- Auswertung von mathematischen Funktionen und Konstanten

Ziele von Rust

- Performance

Ziele von Rust

- Performance
- Verlässlichkeit

Ziele von Rust

- Performance
- Verlässlichkeit
- Produktivität

(vgl. [9] [7, S. 196 ff.] [6])

Hinweise zur Syntax

- Expression und Statements

```
let t = if bedingung_1 { false } else { true };
```

Hinweise zur Syntax

- Expression und Statements

```
let t = if bedingung_1 { false } else { true };
```

- `struct` und `trait` als Klassen und Interfaces

```
struct MyStruct {  my_field: i32, }
impl MyStruct { fn do_smth(&self) {} }
```

Hinweise zur Syntax

- Expression und Statements

```
let t = if bedingung_1 { false } else { true };
```

- `struct` und `trait` als Klassen und Interfaces

```
struct MyStruct {  my_field: i32, }
impl MyStruct { fn do_smth(&self) {} }
```

- Enums

```
enum MyEnum {
  Entry1(i32, i32, i32), Entry2,
}
```

(vgl. [1, S. 122 ff.])

Hinweise zur Syntax

- Generics und `trait objects`

```
enum MyEnum<T> { Entry1(T) }
```

Hinweise zur Syntax

- Generics und `trait objects`

```
enum MyEnum<T> { Entry1(T) }
```

- Operatorenüberladung

```
impl std::ops::Add<MyEnum> for MyEnum {
  fn add(self, rhs: MyEnum) -> MyEnum { ... }
}
```

Hinweise zur Syntax

- Generics und `trait objects`

```
enum MyEnum<T> { Entry1(T) }
```

- Operatorenüberladung

```
impl std::ops::Add<MyEnum> for MyEnum {
  fn add(self, rhs: MyEnum) -> MyEnum { ... }
}
```

- Referenzen

(vgl. [7, S. 196 ff.] [4] [1, S. 246 ff.])

Vertiefung des Ownership und Borrowing

- Performance

Vertiefung des Ownership und Borrowing

- Performance
- Verlässlichkeit

Vertiefung des Ownership und Borrowing

- Performance
- Verlässlichkeit
- Produktivität

(vgl. [9])

Stack, Heap, Copy und Clone

- Performance

Stack, Heap, Copy und Clone

- Performance
- Verlässlichkeit

Stack, Heap, Copy und Clone

- Performance
- Verlässlichkeit
- Produktivität

(vgl. [9])

Tokenizer

- Performance

Tokenizer

- Performance
- Verlässlichkeit

Tokenizer

- Performance
- Verlässlichkeit
- Produktivität

(vgl. [9])

Parser

- Performance

Parser

- Performance
- Verlässlichkeit

Parser

- Performance
- Verlässlichkeit
- Produktivität

(vgl. [9])

Überlauf und Ungenauigkeit

- Performance

Überlauf und Ungenauigkeit

- Performance
- Verlässlichkeit

Überlauf und Ungenauigkeit

- Performance
- Verlässlichkeit
- Produktivität

(vgl. [9])

Überlegungen zur Datenstruktur

- Performance

Überlegungen zur Datenstruktur

- Performance
- Verlässlichkeit

Überlegungen zur Datenstruktur

- Performance
- Verlässlichkeit
- Produktivität

(vgl. [9])

Vergleich zu existierenden Lösungen

- Performance

Vergleich zu existierenden Lösungen

- Performance
- Verlässlichkeit

Vergleich zu existierenden Lösungen

- Performance
- Verlässlichkeit
- Produktivität

(vgl. [9])

Datenstruktur in Rust

- Performance

Datenstruktur in Rust

- Performance
- Verlässlichkeit

Datenstruktur in Rust

- Performance
- Verlässlichkeit
- Produktivität

(vgl. [9])

Grundfunktionalitäten

- Performance

Grundfunktionalitäten

- Performance
- Verlässlichkeit

Grundfunktionalitäten

- Performance
- Verlässlichkeit
- Produktivität

(vgl. [9])

Erweiterung des CAS mit EvalFn

- Performance

Erweiterung des CAS mit EvalFn

- Performance
- Verlässlichkeit

Erweiterung des CAS mit EvalFn

- Performance
- Verlässlichkeit
- Produktivität

(vgl. [9])

Konkrete Erweiterungen des CAS

- Performance

Konkrete Erweiterungen des CAS

- Performance
- Verlässlichkeit

Konkrete Erweiterungen des CAS

- Performance
- Verlässlichkeit
- Produktivität

(vgl. [9])

Implementierung von mathematischen Funktionen

- Performance

Implementierung von mathematischen Funktionen

- Performance
- Verlässlichkeit

Implementierung von mathematischen Funktionen

- Performance
- Verlässlichkeit
- Produktivität

(vgl. [9])

Parsen von Termen

- Performance

Parsen von Termen

- Performance
- Verlässlichkeit

Parsen von Termen

- Performance
- Verlässlichkeit
- Produktivität

(vgl. [9])

Performance

- Performance

Performance

- Performance
- Verlässlichkeit

Performance

- Performance
- Verlässlichkeit
- Produktivität

(vgl. [9])

Verbesserungsideen und deren Ansätze

- Performance

Verbesserungsideen und deren Ansätze

- Performance
- Verlässlichkeit

Verbesserungsideen und deren Ansätze

- Performance
- Verlässlichkeit
- Produktivität

(vgl. [9])

Vor- und Nachteile Rust

- Performance

Vor- und Nachteile Rust

- Performance
- Verlässlichkeit

Vor- und Nachteile Rust

- Performance
- Verlässlichkeit
- Produktivität

(vgl. [9])

[1] Jim Blandy und Jason Orendorff: *Programming Rust: Fast, Safe Systems Development*.
O'Reilly Media, Inc, 1. Auflage, 2018,
ISBN 978-1-491-92728-1.

[2] Fachgruppe Computeralgebra: *Was ist Computeralgebra?*
`https://fachgruppe-computeralgebra.de/computeralgebra/`.
[abgerufen am 18.07.2022].

[3] Keith O. Geddes, Stephen R. Czapor und George Labahn: *Algorithms for Computer Algebra* -, 2007,
ISBN 978-0-585-33247-5.

[4] Steve Klabnik und Carol Nichols: *Using Trait Objects That Allow for Values of Different Types*, Kapitel Using Trait Objects That Allow for Values of Different Types.
No Starch Press, 2019, ISBN 9781718500440.

[5] Aaron Meurer, Christopher P. Smith, Mateusz Paprocki, Ondřej Čertík, Sergey B. Kirpichev, Matthew Rocklin, AMiT Kumar, Sergiu Ivanov, Jason K. Moore, Sartaj Singh, Thilina Rathnayake, Sean Vig, Brian E. Granger, Richard P. Muller, Francesco Bonazzi, Harsh Gupta, Shivam Vats, Fredrik Johansson, Fabian Pedregosa, Matthew J. Curry, Andy R. Terrel, Štěpán Roučka, Ashutosh Saboo, Isuru Fernando, Sumith Kulal, Robert Cimrman und Anthony Scopatz: *SymPy: symbolic computing in Python*.
PeerJ Computer Science, 3:e103, Januar 2017,
ISSN 2376-5992.
https://doi.org/10.7717/peerj-cs.103.

[6] Oracle: *Type Erasure*.
`https://docs.oracle.com/javase/tutorial/java/generics/erasure.html`.
[abgerufen am 18.07.2022].

[7] Rahul Sharma und Vesa Kaihlavirta: *Mastering RUST*.
Packt Publishing Ltd., second edition Auflage, 2019,
ISBN 978-1-78934-657-2.

[8] Kiat Shi Tan, Willi Hans Steeb und Yorick Hardy:
SymbolicC++: An Introduction to Computer Algebra using Object-Oriented Programming, 2000, ISBN 978-1-852-33260-0.

[9] Rust Website: *Why Rust?*
`https://www.rust-lang.org/`.
[abgerufen am 29.08.2022].