Interactive Proof Trees

M. Allmann, F. Christoph, S. Langenstein,
H. Mahmoud, S. Schwenkreis

16. Februar 2022

Motivation

Semantik

- legt Typ und Wert von Ausdrücken eindeutig fest
- kombinieren der Regeln zu Beweisbäumen

Motivation

Semantik

- legt Typ und Wert von Ausdrücken eindeutig fest
- kombinieren der Regeln zu Beweisbäumen

Warum brauchen wir Bäume?

- Typ überprüfen
- falsche Ausdrücke analysieren
- Sprache und Regeln besser verstehen

Motivation

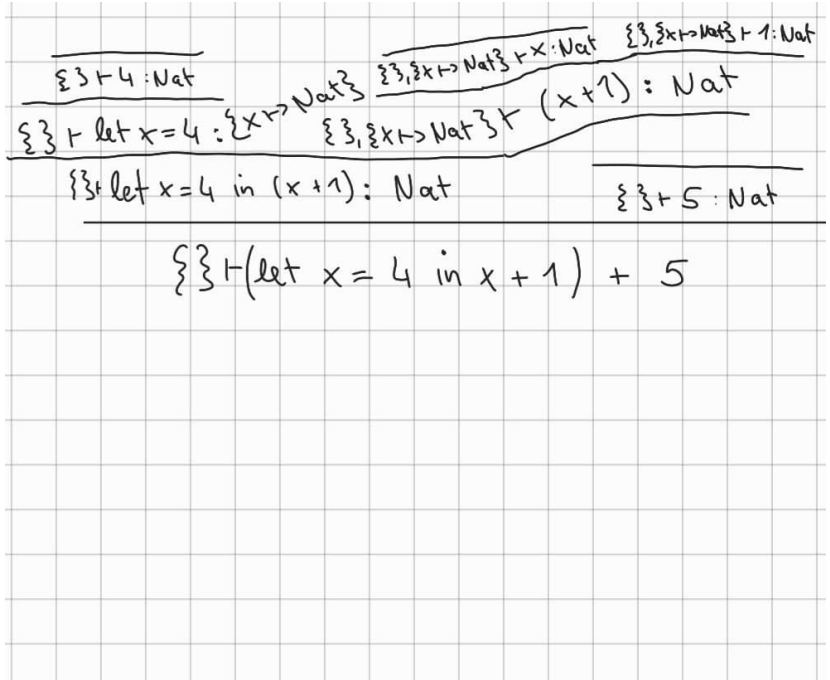
Problem

- bisher nur von Hand

Motivation

Problem

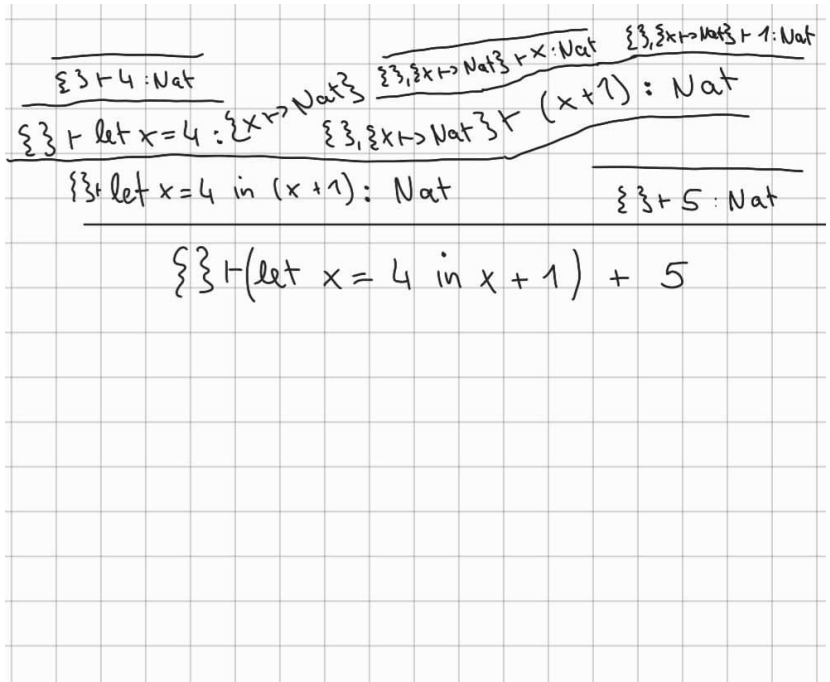
- bisher nur von Hand



Motivation

Problem

- bisher nur von Hand
- Platzprobleme
- unübersichtlich
- schlecht lesbare Bäume
- überfordernd für Anfänger:innen



Motivation

Problem

- bisher nur von Hand
- Platzprobleme
- unübersichtlich
- schlecht lesbare Bäume
- überfordernd für Anfänger:innen

Lösung

- graphische Benutzeroberfläche für Beweisbäume
- benutzerfreundlich und entgegenkommend
- unterstützt auch Tutor:innen

Präsentation

Typregel if-then-else

$$\frac{\Sigma \vdash e_1 : \mathit{Bool} \qquad \Sigma \vdash e_2 : t \qquad \Sigma \vdash e_3 : t}{\Sigma \vdash \textbf{\textit{if}}\ e_1\ \textbf{\textit{then}}\ e_2\ \textbf{\textit{else}}\ e_3 : t}$$

Axiome für boolsche Werte und natürliche Zahlen

$$\frac{}{\mathit{false} : \mathit{Bool}} \qquad \frac{}{\mathit{true} : \mathit{Bool}}$$

$$\frac{}{n : \mathit{Nat}}$$

Typregel Addition

$$\frac{e_1 : \mathit{Nat} \qquad e_2 : \mathit{Nat}}{e_1 + e_2 : \mathit{Nat}}$$

Präsentation

Typregel Funktionsdefinition

$$\frac{\Sigma, \{x_1 \mapsto t_1\} \vdash e_2 : t_2}{\Sigma \vdash (\textbf{let}\ f\ (x_1 : t_1) : t_2 = e_2) : \{f \mapsto t_1 \to t_2\}}$$

Datenstruktur

```
export interface Expression {
    typecheck(signatur: Signatur, ergtyp: Type): Beweisziel[]
    toString(): string
}

export interface Definition {
    typecheck(signatur: Signatur, ergsign: Signaturvariable): Beweisziel[]
    toString(): string

}
```

Idee

- Objektorientierter Ansatz $\rightarrow$ eigene Klassen für jede Art von Ausdruck/Definition

Ausdrücke

Typregel

$$\frac{\Sigma \vdash e_1 : Bool \qquad \Sigma \vdash e_2 : t \qquad \Sigma \vdash e_3 : t}{\Sigma \vdash \textit{if } e_1 \textit{ then } e_2 \textit{ else } e_3 : t}$$

Umsetzung

```
export class IfThenElse implements Expression{
    e1: Expression;
    e2: Expression;
    e3: Expression;

    typecheck(signatur: Signatur, ergtyp:Type): BeweiszielExpression[] {
        return [
            new BeweiszielExpression(signatur, this.e1, BooleanType),
            new BeweiszielExpression(signatur, this.e2, ergtyp),
            new BeweiszielExpression(signatur, this.e3, ergtyp)
        ]
    }

    //(...)
```

Definitionen

Typregel

$$\frac{\Sigma \vdash e : t}{\Sigma \vdash (\textbf{\textit{let}}\ x = e) : \{x \mapsto t\}}$$

Umsetzung

```
export class VariDef implements Definition{
    x: Identifier;
    e: Expression;

    typecheck(signatur: Signatur, ergsign: Signaturvariable): BeweiszielExpression[] {
        let t = new Typvariable()
        let map = new Map<string,Type>()
        map.set(this.x.id, t)
        ergsign.setSignature(map)
        return [new BeweiszielExpression(signatur, this.e, t)]
    }
```

in-Ausdruck

Typregel

$$\frac{\Sigma \vdash d : \Sigma' \qquad \Sigma, \Sigma' \vdash e : t}{\Sigma \vdash (d \ \textbf{\textit{in}} \ e) : t}$$

Umsetzung

```
export class inAusdruck implements Expression{
    d: Definition;
    e: Expression;

    typecheck (signatur: Signatur, ergtyp: Type): Beweisziel[] {
        let newSign = new Signaturvariable();
        const def = new BeweiszielSignatur(signatur, this.d, newSign)
        const expr = new BeweiszielExpression
            (new KommaoperatorSignatur (signatur, newSign), this.e, ergtyp)
        return [def, expr]
    }
```

Interactive

Problem

Wie bekommen wir es hin Expression.ts interactive zu machen. Sprich Beweisbäume auf dem Browser statt auf dem Terminal zu zeigen

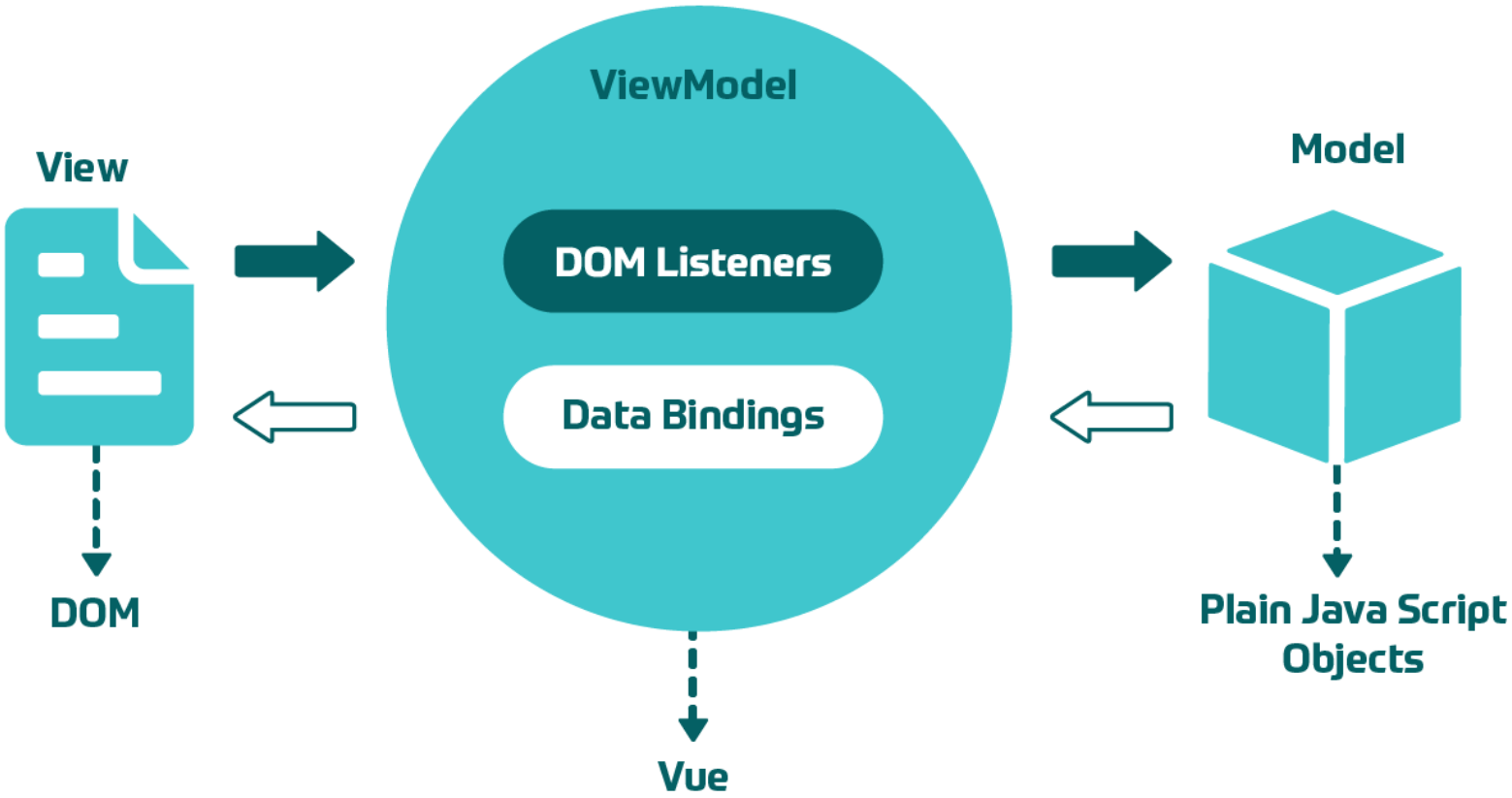
Lösung

Wir verwenden den Webframework Vue.JS(TypeScript, HTML, CSS).

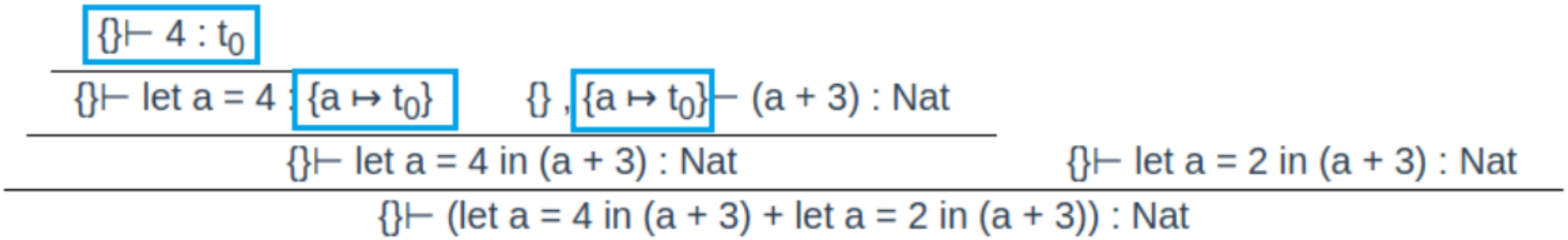
Vue – Infrastructure

Vorteile

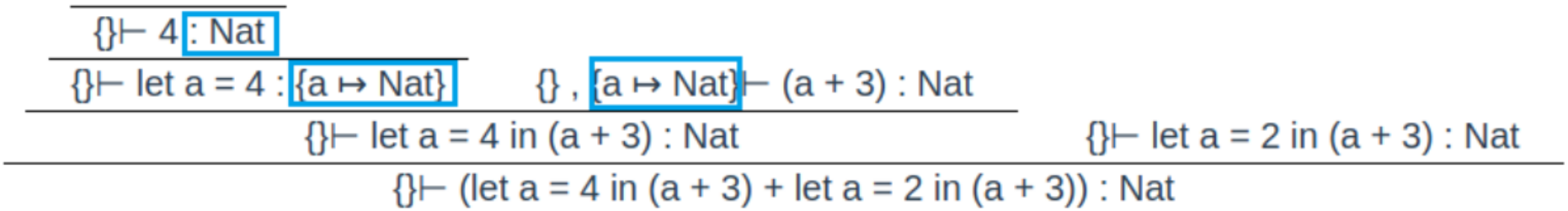
Für dieses Projekt sind:
1) Data Binding v-bind 2) Logik in HTML{v-if ; v-for} relevant
Stichwort: *Reactive Programming*



Vue – Data Binding



Vue – Data Binding



Vue – Components

Beweisbaum.vue

```
1   <template>
2   <table>
3   <tr v-if ="voraussetzungen != null">
4    <td class="center" v-for="proof in voraussetzungen" :key="proof">
5     <Beweisbaum v-if="isEmpty" v-bind:ziel="[proof]"/>
6    </td>
7   </tr>
8   <tr>
9     <div class="error" @click="fadeerror" v-if="hasError1"> ... <br></div>
10    <div class="error" @click="fadeerror" v-if="hasError2"> ... <br></div>
11    <div class="error" @click="fadeerror" v-if="hasError3"> ... <br></div>
12    <div class="center">
13    <td class="clickable" @click = "generateProofs" v-for="proofe in generateStrings()"
14    <div class="center" v-html = "proofs"></div></td></div>
15  </tr>
16  </table>
17  </template>
```

Parser

Problem

- umständliche Eingabe im Sourcecode

Lösung

- Parser um Baumstruktur zu erstellen
- verwenden die Library: TypeScript Parsec

Lexer

Verarbeitung

- Tokens definiert
- Input durch RegEx tokenisiert
- Output: Tokenstream
- Tokens speichern Informationen

```
4   enum Tok {
5       LParen, RParen,
6       Plus, Minus, Star, Div,
7       Eq, Lt, Gt, Leq, Geq,
8       Colon, Arrow,
9       If, Then, Else,
10      Let, Rec, In,
11      Fun,
12      True, False, NumberLiteral,
13      Identifier,
14      Space,
15  }
```

Lexer

```
17  const lexer = buildLexer([
18      [true, /^\(/g, Tok.LParen],
19      [true, /^\)/g, Tok.RParen],
20      [true, /^\+/g, Tok.Plus],
21      [true, /^\-/g, Tok.Minus],
22      [true, /^\*/g, Tok.Star],
23      [true, /^\//g, Tok.Div],
24      [true, /^=/g, Tok.Eq],
25      [true, /^</g, Tok.Lt],
26      [true, /^>/g, Tok.Gt],
27      [true, /^<=/g, Tok.Leq],
28      [true, /^>=/g, Tok.Geq],
29      [true, /^:/g, Tok.Colon],
30      [true, /^->/g, Tok.Arrow],
31      [true, /^if/g, Tok.If],
32      [true, /^then/g, Tok.Then],
33      [true, /^else/g, Tok.Else],
34      [true, /^let/g, Tok.Let],
35      [true, /^rec/g, Tok.Rec],
36      [true, /^in/g, Tok.In],
37      [true, /^fun/g, Tok.Fun],
38      [true, /^true/g, Tok.True],
39      [true, /^false/g, Tok.False],
40      [true, /^\d+/g, Tok.NumberLiteral],
41      [true, /^[a-zA-Z_][A-Za-z0-9_]*/g, Tok.Identifier], // must come below all keywords
42      [false, /^\s+/g, Tok.Space],
43  ]);
```

Type Parser

Unser rechtsassoziativer Typeparser...

```
168  AtomType.setPattern(alt(
169      apply(tok(Tok.Identifier), id => {
170          switch(id.text) {
171              case "Nat": return E.NaturalType;
172              case "Bool": return E.BooleanType;
173              default: throw new Error(`Unbekannter Typ: ${id.text}`);
174          }
175      }),
176      kmid(tok(Tok.LParen), Type, tok(Tok.RParen))
177  ));
178
179  Type.setPattern(alt(
180      AtomType,
181      apply(seq(AtomType, kright(tok(Tok.Arrow), Type)), ([t1, t2]) => new E.FunctionType(t1, t2)),
182  ));
```