

Make Isabelle
Accessible !

Reinhard
Koutny, Klaus
Miesenberger,
Walther
Neuper,
Bernhard
Stöger

Good luck
with
"Accessible
Isabelle"

Assist.Tech.

Mod.Concept.

Use "ThEdu"

Make Isabelle Accessible !

Reinhard Koutny Klaus Miesenberger Walther
Neuper Bernhard Stöger

University of Technology
Graz, Austria

*ThEdu'21 at CADE, Pittsburgh
July 11, 2021*

Outline

- ➊ Assistive Technologies and Mathematics — make “Accessible” means: “usable for visually impaired people”
- ➋ Modular Conception of Isabelle/VSCode
- ➌ Further Use of ““ThEdu””

Outline

- 1 Assistive Technologies and Mathematics — make “Accessible” means: “usable for visually impaired people”
- 2 Modular Conception of Isabelle/VSCode
- 3 Further Use of ““ThEdu””

Outline

- ➊ Assistive Technologies and Mathematics — make “Accessible” means: “usable for visually impaired people”
- ➋ Modular Conception of Isabelle/VSCode
- ➌ Further Use of ““ThEdu””

Make Isabelle
Accessible !

Reinhard
Koutny, Klaus
Miesenberger,
Walther
Neuper,
Bernhard
Stöger

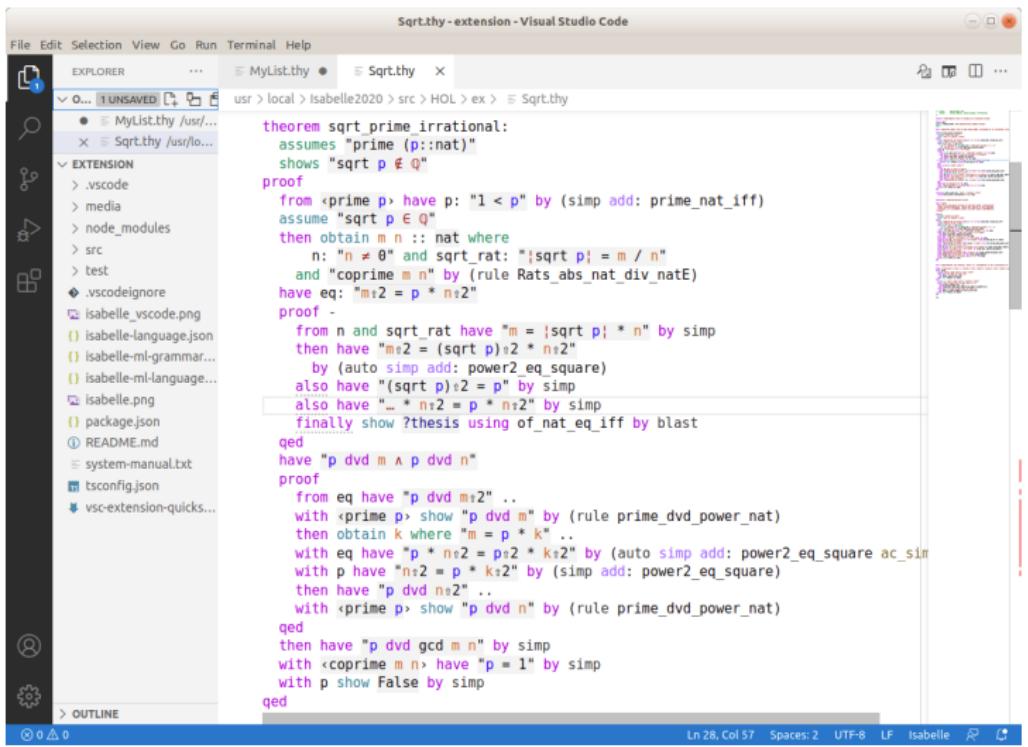
Good luck
with
"Accessible
Isabelle"

Assist.Tech.

Mod.Concept.

Use "ThEdu"

This can be read by a **blind** mathematician . . .

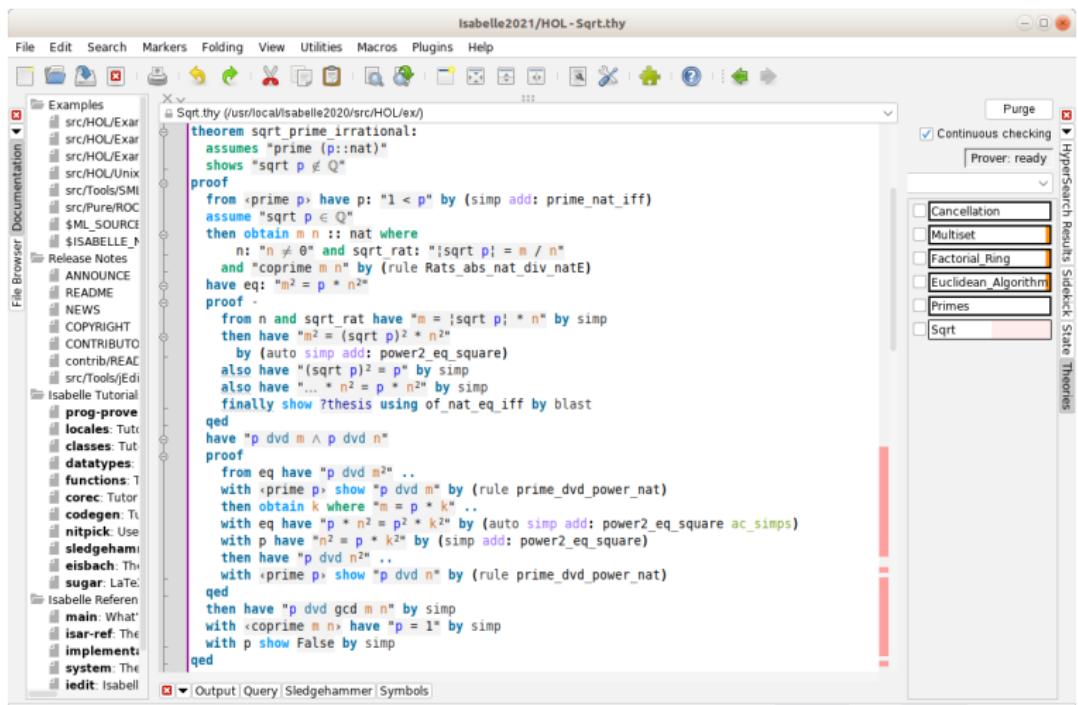


The screenshot shows a Visual Studio Code window with the title "Sqrt.thy - extension - Visual Studio Code". The left sidebar shows a file tree with files like "MyList.thy", "Sqrt.thy", ".vscode", "media", "node_modules", "src", "test", ".vscodeignore", "isabelle_vscode.png", "isabelle-language.json", "isabelle-ml-grammar...", "isabelle-ml-language...", "isabelle.png", "package.json", "README.md", "system-manual.txt", "tsconfig.json", and "vsc-extension-quicks...". The main editor area contains the following Isabelle proof script:

```
theorem sqrt_prime_irrational:
  assumes "prime (p::nat)"
  shows "sqrt p ∉ ℚ"
proof
  from <prime p> have p: "1 < p" by (simp add: prime_nat_iff)
  assume "sqrt p ∈ ℚ"
  then obtain m n :: nat where
    n: "n ≠ 0" and sqrt_rat: "|sqrt p| = m / n"
    and "coprime m n" by (rule Rats_abs_nat_div_natE)
  have eq: "m² = p * n²"
  proof -
    from n and sqrt_rat have "m = |sqrt p| * n" by simp
    then have "m² = (sqrt p)² * n²"
      by (auto simp add: power2_eq_square)
    also have "(sqrt p)² = p" by simp
    also have "... * n² = p * n²" by simp
    finally show ?thesis using of_nat_eq_iff by blast
  qed
  have "p dvd m ∧ p dvd n"
  proof
    from eq have "p dvd m²" ..
    with <prime p> show "p dvd m" by (rule prime_dvd_power_nat)
    then obtain k where "m = p * k" ..
    with eq have "p * n² = p² * k²" by (auto simp add: power2_eq_square ac_simps)
    with p have "n² = p * k²" by (simp add: power2_eq_square)
    then have "p dvd n²" ..
    with <prime p> show "p dvd n" by (rule prime_dvd_power_nat)
  qed
  then have "p dvd gcd m n" by simp
  with <coprime m n> have "p = 1" by simp
  with p show False by simp
qed
```

The status bar at the bottom shows "Ln 28, Col 57" and other file-related information.

This could *not* be read (Isabelle/jEdit)

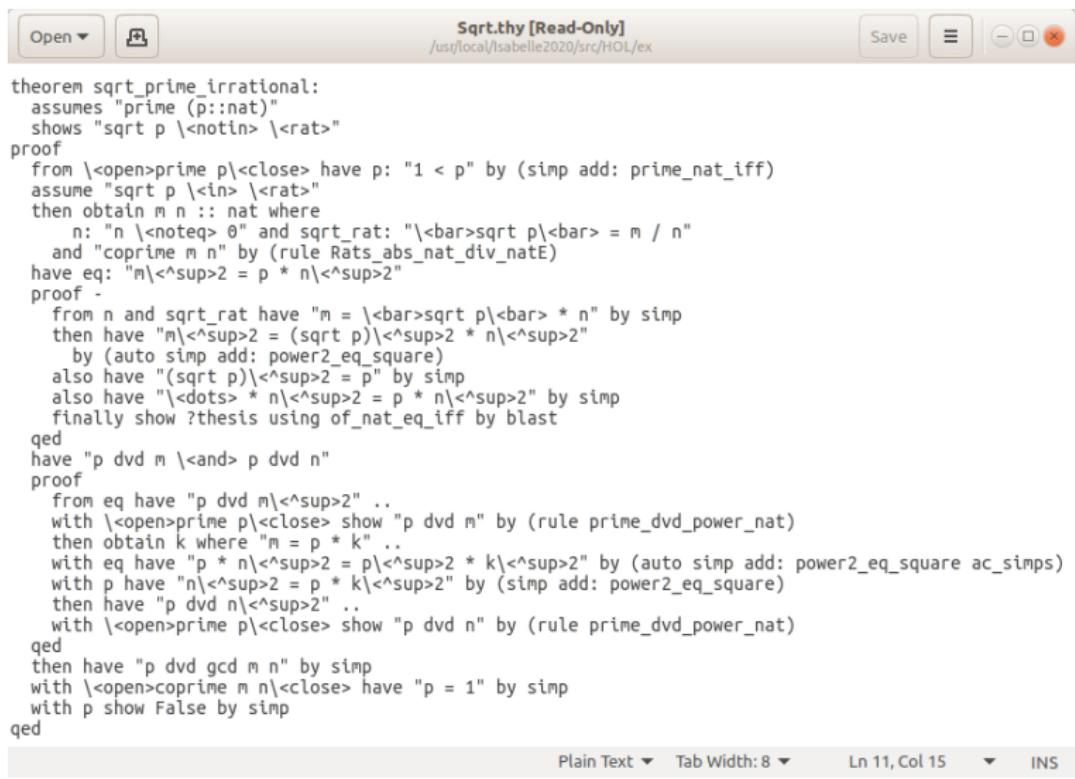


```
Isabelle2021/HOL - Sqrt.thy
File Edit Search Markers Folding View Utilities Macros Plugins Help
File Browser Documentation
Sqrt.thy (/usr/local/Isabelle2021/src/HOL/ex)
theorem sqrt_prime_irrational:
  assumes "prime (p::nat)"
  shows "sqrt p ∉ ℚ"
proof
  from ‹prime p› have p: "1 < p" by (simp add: prime_nat_iff)
  assume "sqrt p ∈ ℚ"
  then obtain m n :: nat where
    n: "n ≠ 0" and sqrt_rat: "|sqrt p| = m / n"
    and "coprime m n" by (rule Rats_abs_nat_div_natE)
  have eq: "m² = p * n²"
  proof -
    from m n sqrt_rat have "m = |sqrt p| * n" by simp
    then have "m² = (sqrt p)² * n²"
      by (auto simp add: power2_eq_square)
    also have "(sqrt p)² = p" by simp
    also have "... * n² = p * n²" by simp
    finally show ?thesis using of_nat_eq_iff by blast
  qed
  have "p dvd m ∧ p dvd n"
  proof
    from eq have "p dvd m²" ..
    with ‹prime p› show "p dvd m" by (rule prime_dvd_power_nat)
    then obtain k where "m = p * k" ..
    with eq have "p * n² = p² * k²" by (auto simp add: power2_eq_square ac_simps)
    with p have "n² = p * k²" by (simp add: power2_eq_square)
    then have "p dvd n²" ..
    with ‹prime p› show "p dvd n" by (rule prime_dvd_power_nat)
  qed
  then have "p dvd gcd m n" by simp
  with ‹coprime m n› have "p = 1" by simp
  with p show False by simp
qed
```

Continuous checking: Prover: ready

- Cancellation
- Multiset
- Factorial_Ring
- Euclidean_Algorithm
- Primes
- Sqrt

A glimpse at “Good Luck”: the same proof in a simple editor



The screenshot shows a simple text editor window with a toolbar at the top and a status bar at the bottom. The editor contains a HOL proof script for the theorem `sqrt_prime_irrational`.

```
theorem sqrt_prime_irrational:
  assumes "prime (p::nat)"
  shows "sqrt p <notin> <rat>
proof
  from <open>prime p<<close> have p: "1 < p" by (simp add: prime_nat_iff)
  assume "sqrt p <in> <rat>" then obtain m n :: nat where
    n: "n <noteq> 0" and sqrt_rat: "<bar>sqrt p <bar> = m / n"
    and "coprime m n" by (rule Rats_abs_nat_div_natE)
  have eq: "m<^sup>2 = p * n<^sup>2" by simp
  proof -
    from n and sqrt_rat have "m = <bar>sqrt p <bar> * n" by simp
    then have "m<^sup>2 = (sqrt p)<^sup>2 * n<^sup>2"
      by (auto simp add: power2_eq_square)
    also have "(sqrt p)<^sup>2 = p" by simp
    also have "<dots> * n<^sup>2 = p * n<^sup>2" by simp
    finally show ?thesis using of_nat_eq_iff by blast
  qed
  have "p dvd m & p dvd n" by simp
  proof
    from eq have "p dvd m<^sup>2" ..
    with <open>prime p<<close> show "p dvd m" by (rule prime_dvd_power_nat)
    then obtain k where "m = p * k" ..
    with eq have "p * n<^sup>2 = p<^sup>2 * k<^sup>2" by (auto simp add: power2_eq_square ac_simps)
    with p have "n<^sup>2 = p * k<^sup>2" by (simp add: power2_eq_square)
    then have "p dvd n<^sup>2" ..
    with <open>prime p<<close> show "p dvd n" by (rule prime_dvd_power_nat)
  qed
  then have "p dvd gcd m n" by simp
  with <open>coprime m n<<close> have "p = 1" by simp
  with p show False by simp
qed
```

The status bar at the bottom shows "Plain Text" and "Tab Width: 8". The bottom right corner has icons for navigating between tabs and windows.

A closer look:

Compare

- accessible representation in Isabelle/VSCode ...

```
proof
  from <prime p> have p: "1 < p" by (simp add: prime_nat_iff)
  assume "sqrt p ∈ ℚ"
  then obtain m n :: nat where
    n: "n ≠ 0" and sqrt_rat: "|sqrt p| = m / n"
    and "coprime m n" by (rule Rats_abs_nat_div_natE)
  have eq: "m² = p * n²"
```

- with plain text editor

```
proof
  from <open>prime p</close> have p: "1 < p" by (simp add: prime_nat_iff)
  assume "sqrt p <in> <rat>"
  then obtain m n :: nat where
    n: "n <noteq> 0" and sqrt_rat: "|sqrt p| = m / n"
    and "coprime m n" by (rule Rats_abs_nat_div_natE)
  have eq: "m<^sup>2 = p * n<^sup>2"
```

Good luck with "Accessible Isabelle"

How comes?

- Isabelle's *standard* front-end Isabelle/jEdit is **not** accessible
- the *experimental* front-end Isabelle/**VSCODE** **is** accessible because
 - special math symbols are encoded in Isabelle fonts $\in \dots \backslash \text{<in>} \quad \mathbb{Q} \dots \backslash \text{<rat>} \quad \neq \dots \backslash \text{<noteq>}$
 - VSCode is tweaked to present \neq for $\backslash \text{<noteq>}$ etc
 - VSCode uses the Webbrowser Chromium
 - Chromium feeds screen-readers with $\backslash \text{<noteq>}$ etc
 - and screen readers transfer $\backslash \text{<noteq>}$ to the braille — which **can be read by everybody**.
- while jEdit's good old JavaSwing is
 - more tweaked by Isabelle
 - not as accessible as most recent Chromium

Good luck with "Accessible Isabelle"

How comes?

- Isabelle's *standard* front-end Isabelle/jEdit is **not** accessible
- the *experimental* front-end Isabelle/**VSCODE** **is** accessible because
 - special math symbols are encoded in Isabelle fonts $\in \dots \backslash \text{in} \quad \mathbb{Q} \dots \backslash \text{rat} \quad \neq \dots \backslash \text{noteq}$
 - VSCODE is tweaked to present \neq for $\backslash \text{noteq}$ etc
 - VSCODE uses the Webbrowser Chromium
 - Chromium feeds screen-readers with $\backslash \text{noteq}$ etc
 - and screen readers transfer $\backslash \text{noteq}$ to the braille — which **can be read by everybody**.
- while jEdit's good old JavaSwing is
 - more tweaked by Isabelle
 - not as accessible as most recent Chromium

Good luck with "Accessible Isabelle"

How comes?

- Isabelle's *standard* front-end Isabelle/jEdit is **not** accessible
- the *experimental* front-end Isabelle/**VSCODE** **is** accessible because
 - special math symbols are encoded in Isabelle fonts $\in \dots \backslash \langle in \rangle \quad \mathbb{Q} \dots \backslash \langle rat \rangle \quad \neq \dots \backslash \langle noteq \rangle$
 - VSCODE is tweaked to present \neq for $\backslash \langle noteq \rangle$ etc
 - VSCODE uses the Webbrowser Chromium
 - Chromium feeds screen-readers with $\backslash \langle noteq \rangle$ etc
 - and screen readers transfer $\backslash \langle noteq \rangle$ to the braille — which **can be read by everybody**.
- while jEdit's good old JavaSwing is
 - more tweaked by Isabelle
 - not as accessible as most recent Chromium

Good luck with "Accessible Isabelle"

How comes?

- Isabelle's *standard* front-end Isabelle/jEdit is **not** accessible
- the *experimental* front-end Isabelle/**VSCODE** **is** accessible because
 - special math symbols are encoded in Isabelle fonts $\in \dots \backslash \langle in \rangle \quad \mathbb{Q} \dots \backslash \langle rat \rangle \quad \neq \dots \backslash \langle noteq \rangle$
 - VSCODE is tweaked to present \neq for $\backslash \langle noteq \rangle$ etc
 - VSCODE uses the Webbrowser Chromium
 - Chromium feeds screen-readers with $\backslash \langle noteq \rangle$ etc
 - and screen readers transfer $\backslash \langle noteq \rangle$ to the braille — which **can be read by everybody**.
- while jEdit's good old JavaSwing is
 - more tweaked by Isabelle
 - not as accessible as most recent Chromium

Good luck with "Accessible Isabelle"

How comes?

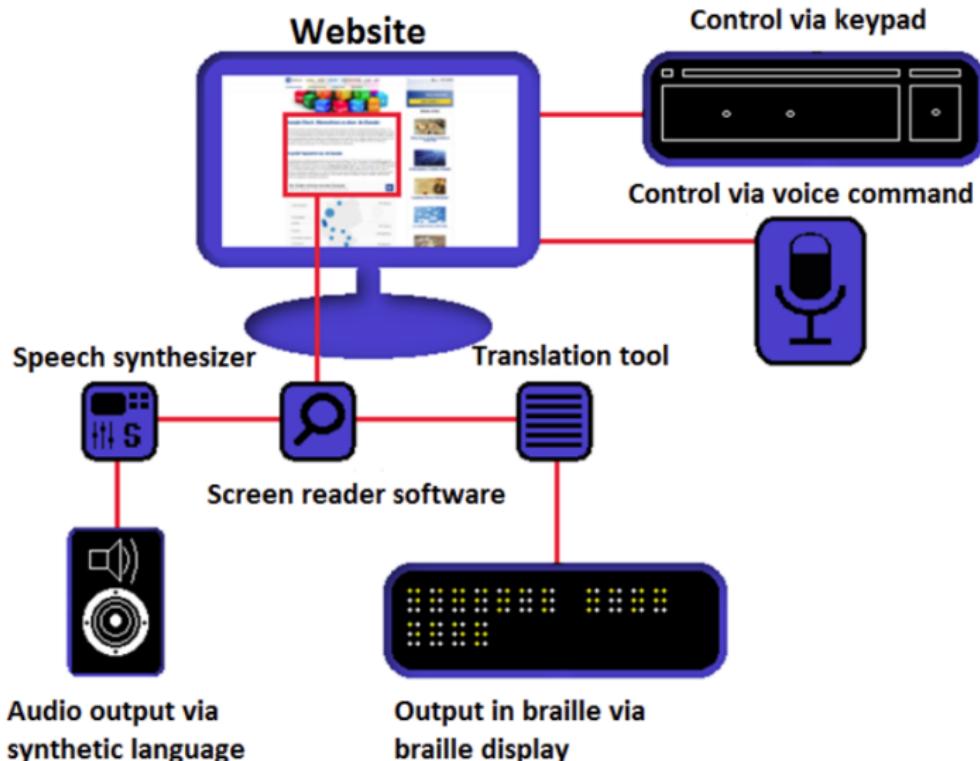
- Isabelle's *standard* front-end Isabelle/jEdit is **not** accessible
- the *experimental* front-end Isabelle/**VSCODE** **is** accessible because
 - special math symbols are encoded in Isabelle fonts $\in \dots \backslash \text{in} \quad \mathbb{Q} \dots \backslash \text{rat} \quad \neq \dots \backslash \text{noteq}$
 - VSCODE is tweaked to present \neq for $\backslash \text{noteq}$ etc
 - VSCODE uses the Webbrowser Chromium
 - Chromium feeds screen-readers with $\backslash \text{noteq}$ etc
 - and screen readers transfer $\backslash \text{noteq}$ to the braille — which **can be read by everybody**.
- while jEdit's good old JavaSwing is
 - more tweaked by Isabelle
 - not as accessible as most recent Chromium

Good luck with "Accessible Isabelle"

How comes?

- Isabelle's *standard* front-end Isabelle/jEdit is **not** accessible
- the *experimental* front-end Isabelle/**VSCODE** **is** accessible because
 - special math symbols are encoded in Isabelle fonts $\in \dots \backslash \text{in} \quad \mathbb{Q} \dots \backslash \text{rat} \quad \neq \dots \backslash \text{noteq}$
 - VSCODE is tweaked to present \neq for $\backslash \text{noteq}$ etc
 - VSCODE uses the Webbrowser Chromium
 - Chromium feeds screen-readers with $\backslash \text{noteq}$ etc
 - and screen readers transfer $\backslash \text{noteq}$ to the braille — which **can be read by everybody**.
- while jEdit's good old JavaSwing is
 - more tweaked by Isabelle
 - not as accessible as most recent Chromium

Assistive technologies: screen-reader is central



Make Isabelle
Accessible !

Reinhard
Koutny, Klaus
Miesenberger,
Walther
Neuper,
Bernhard
Stöger

Good luck
with
"Accessible
Isabelle"

Assist.Tech.

Mod.Concept.

Use "ThEdu"

Assistive technologies: braille-display !?!



Assistive Technologies and Mathematics

In brief:

- Assistive technologies are
 - **sequential**: audio output from screen-readers
 - **linear**: output from screen-readers to braille
- while mathematical structures are mainly **not linear**:
 - formulas: $\int \frac{x^2+1}{(x+2)(x-1)(x^2+x+1)} = \int \frac{a}{x+2} + \frac{b}{x-1} + \frac{cx+d}{x^2+x+1}$
 - matrices, binomial coefficients, etc
 - proofs: have enclosed sub-proofs etc
 - calculations in engineering math: subproblems etc
 - diagrams, graphs, etc
- A blind person with an acad.degree in math — **unthinkable**.

Assistive Technologies and Mathematics

In brief:

- Assistive technologies are
 - **sequential**: audio output from screen-readers
 - **linear**: output from screen-readers to braille
- while mathematical structures are mainly **not linear**:
 - formulas: $\int \frac{x^2+1}{(x+2)(x-1)(x^2+x+1)} = \int \frac{a}{x+2} + \frac{b}{x-1} + \frac{cx+d}{x^2+x+1}$
 - matrices, binomial coefficients, etc
 - proofs: have enclosed sub-proofs etc
 - calculations in engineering math: subproblems etc
 - diagrams, graphs, etc
- A blind person with an acad.degree in math — **unthinkable**.

Assistive Technologies and Mathematics

In brief:

- Assistive technologies are
 - **sequential**: audio output from screen-readers
 - **linear**: output from screen-readers to braille
- while mathematical structures are mainly **not linear**:
 - formulas: $\int \frac{x^2+1}{(x+2)(x-1)(x^2+x+1)} = \int \frac{a}{x+2} + \frac{b}{x-1} + \frac{cx+d}{x^2+x+1}$
 - matrices, binomial coefficients, etc
 - proofs: have enclosed sub-proofs etc
 - calculations in engineering math: subproblems etc
 - diagrams, graphs, etc
- A blind person with an acad.degree in math — **unthinkable**.

Modular conception of Isabelle/**VSCODE**

- VSCODE renders content using **Chromium** (this is accessible)
- **Electron** combines Chromium and the Node.js runtime; this allows to develop in parallel
 - web applications
 - desktop applications
- **Node.js** executes JavaScript code outside a web browser.
 - *one language on server-side and on client-side*
 - event-driven architecture capable of asynchronous I/O
 - *technology to develop collaborative learning tools.*
- VSCODE communicates with Isabelle using Microsoft’s **LSP** – language server protocol
 -
 -
 -

Modular conception of Isabelle/**VSCode**

- VSCode renders content using **Chromium** (this is accessible)
- **Electron** combines Chromium and the Node.js runtime; this allows to develop in parallel
 - web applications
 - desktop applications
- **Node.js** executes JavaScript code outside a web browser.
 - *one language on server-side and on client-side*
 - event-driven architecture capable of asynchronous I/O
 - *technology to develop collaborative learning tools.*
- VSCode communicates with Isabelle using Microsoft's **LSP** – language server protocol
 -
 -
 -

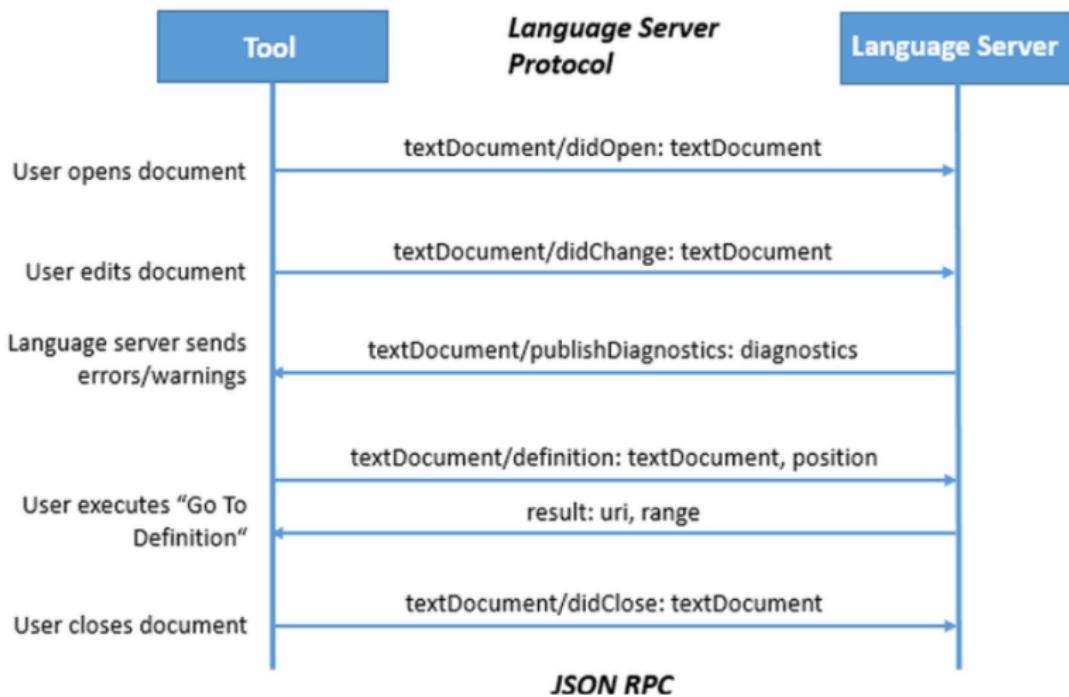
Modular conception of Isabelle/**VSCode**

- VSCode renders content using **Chromium** (this is accessible)
- **Electron** combines Chromium and the Node.js runtime; this allows to develop in parallel
 - web applications
 - desktop applications
- **Node.js** executes JavaScript code outside a web browser.
 - *one language on server-side and on client-side*
 - event-driven architecture capable of asynchronous I/O
 - *technology to develop collaborative learning tools.*
- VSCode communicates with Isabelle using Microsoft's **LSP** – language server protocol
 -
 -
 -

Modular conception of Isabelle/**VSCode**

- VSCode renders content using **Chromium** (this is accessible)
- **Electron** combines Chromium and the Node.js runtime; this allows to develop in parallel
 - web applications
 - desktop applications
- **Node.js** executes JavaScript code outside a web browser.
 - *one language on server-side and on client-side*
 - event-driven architecture capable of asynchronous I/O
 - *technology to develop collaborative learning tools.*
- VSCode communicates with Isabelle using Microsoft's **LSP** – language server protocol
 -
 -
 -

LSP language server protocol



Modular conception of Isabelle/**VSCode**

- VSCode renders content using **Chromium** (this is accessible)
- **Electron** combines Chromium and the Node.js runtime; this allows to develop in parallel
 - web applications
 - desktop applications
- **Node.js** executes JavaScript code outside a web browser.
 - *one language on server-side and on client-side*
 - event-driven architecture capable of asynchronous I/O
 - *technology to develop collaborative learning tools.*
- VSCode communicates with Isabelle using Microsoft's **LSP** – language server protocol
 - separates programming language support from IDE
 - uses remote procedure call protocol JSON-RPC
- **Isabelle/PIDE** uses an LSP variant for connection to jEdit.

Modular conception of Isabelle/**VSCode**

- VSCode renders content using **Chromium** (this is accessible)
- **Electron** combines Chromium and the Node.js runtime; this allows to develop in parallel
 - web applications
 - desktop applications
- **Node.js** executes JavaScript code outside a web browser.
 - *one language on server-side and on client-side*
 - event-driven architecture capable of asynchronous I/O
 - *technology to develop collaborative learning tools.*
- VSCode communicates with Isabelle using Microsoft's **LSP** – language server protocol
 - separates programming language support from IDE
 - uses remote procedure call protocol JSON-RPC
- **Isabelle/PIDE** uses an LSP variant for connection to jEdit.

Modular conception of Isabelle/**VSCODE**

- VSCODE renders content using **Chromium** (this is accessible)
- **Electron** combines Chromium and the Node.js runtime; this allows to develop in parallel
 - web applications
 - desktop applications
- **Node.js** executes JavaScript code outside a web browser.
 - *one language on server-side and on client-side*
 - event-driven architecture capable of asynchronous I/O
 - *technology to develop collaborative learning tools.*
- VSCODE communicates with Isabelle using Microsoft's **LSP** – language server protocol
 - separates programming language support from IDE
 - uses remote procedure call protocol JSON-RPC
- **Isabelle/PIDE** uses an LSP variant for connection to jEdit.

Modular conception of Isabelle/**VSCode**

- VSCode renders content using **Chromium** (this is accessible)
- **Electron** combines Chromium and the Node.js runtime; this allows to develop in parallel
 - web applications
 - desktop applications
- **Node.js** executes JavaScript code outside a web browser.
 - *one language on server-side and on client-side*
 - event-driven architecture capable of asynchronous I/O
 - *technology to develop collaborative learning tools.*
- VSCode communicates with Isabelle using Microsoft's **LSP** – language server protocol
 - separates programming language support from IDE
 - uses remote procedure call protocol JSON-RPC
- **Isabelle/PIDE** uses an LSP variant for connection to jEdit.

All components are **open source** and cross-platform.

Use of "ThEdu"

Once upon a time "ThEdu" abbreviated

"Theorem Proving Components for Educational Software"
and was a design principle of the *ISAC*-project¹; now this project
proceeds with

① step: *Make Isabelle/VSCode Accessible*

- adapt LSP to accessibility requirements (term *structure*)
- translate Isabelle-symbols to Braille codes
- improve functionality of Isabelle/VSCode (needs to catch up several years of development)

② step: *Integrate ISAC into Isabelle/PIDE*, i.e. re-use as many components of Isabelle as possible.

- adopt features to define formal languages of *ISAC*
- add to interactivity by "Lucas-Interpretation"
- add a dialog-component for "next-step guidance"
- **get accessibility for free from Isabelle/VSCode.**

③ step: ???

¹<https://isac.miraheze.org/wiki/History> 

Use of "ThEdu"

Once upon a time "ThEdu" abbreviated

"Theorem Proving Components for Educational Software"
and was a design principle of the *ISAC*-project¹; now this project
proceeds with

① step: **Make Isabelle/VSCode Accessible**

- adapt LSP to accessibility requirements (term *structure*)
- translate Isabelle-symbols to Braille codes
- improve functionality of Isabelle/VSCode (needs to catch up several years of development)

② step: *Integrate ISAC into Isabelle/PIDE*, i.e. re-use as many components of Isabelle as possible.

- adopt features to define formal languages of *ISAC*
- add to interactivity by "Lucas-Interpretation"
- add a dialog-component for "next-step guidance"
- **get accessibility for free from Isabelle/VSCode.**

③ step: ???

¹<https://isac.miraheze.org/wiki/History>

Use of "ThEdu"

Once upon a time "ThEdu" abbreviated

"Theorem Proving Components for Educational Software"
and was a design principle of the *ISAC*-project¹; now this project
proceeds with

① step: **Make Isabelle/VSCode Accessible**

- adapt LSP to accessibility requirements (term *structure*)
- translate Isabelle-symbols to Braille codes
- improve functionality of Isabelle/VSCode (needs to catch up several years of development)

② step: **Integrate ISAC into Isabelle/PIDE**, i.e. re-use as many components of Isabelle as possible.

- adopt features to define formal languages of *ISAC*
- add to interactivity by "Lucas-Interpretation"
- add a dialog-component for "next-step guidance"
- **get accessibility for free from Isabelle/VSCode.**

③ step: ???

¹<https://isac.miraheze.org/wiki/History>

Make Isabelle
Accessible !

Reinhard
Koutny, Klaus
Miesenberger,
Walther
Neuper,
Bernhard
Stöger

Good luck
with
"Accessible
Isabelle"

Assist.Tech.

Mod.Concept.

Use "ThEdu"

Thank you for Attention!