Collaborative Interactive Theorem Proving with Clide

Martin Ring, Christoph Lüth

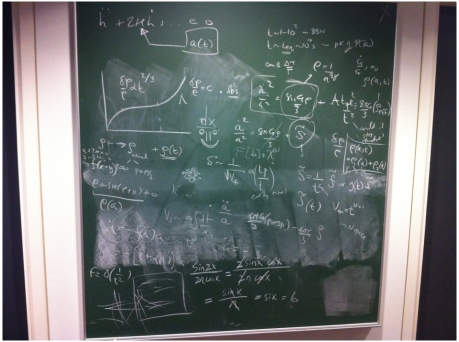
ITP 2014, 15.07.2014, Vienna

Motivation

Interactive theorem proving can be lonesome. . .

Motivation

Interactive theorem proving can be lonesome. . .

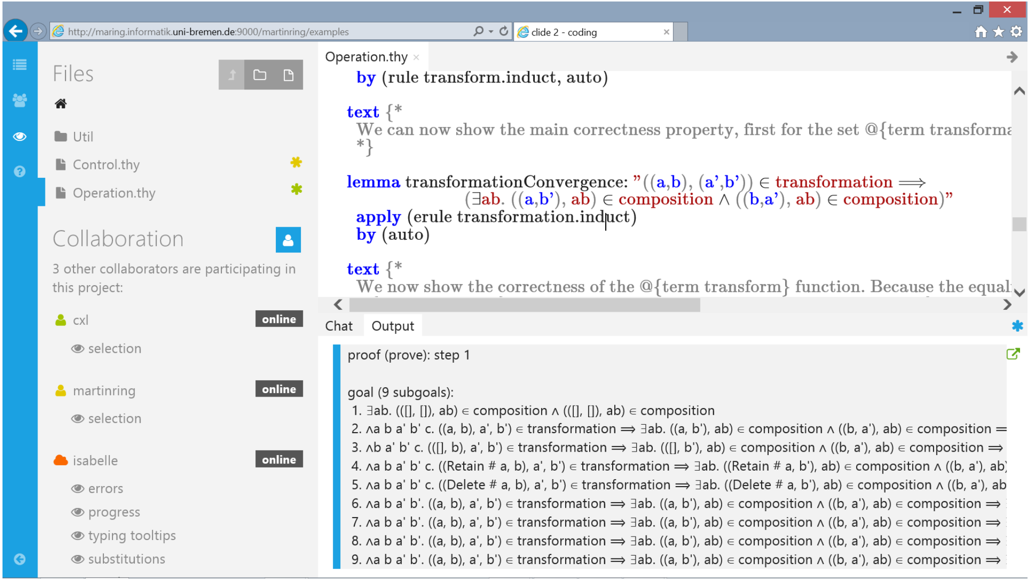


. . . but mathematics is a social activity!

Introducing Clide

- Previous work: a web interface for Isabelle
- Next step: extend this to **real-time collaborative** proof
- "Google docs for proofs"

Action!



3 other collaborators are participating in this project:

```
    by (rule transform.induct, auto)

  text {*
    We can now show the main correctness property, first for the set @{term transforma
    *}

  lemma transformationConvergence: "((a,b), (a',b')) ∈ transformation ⟹
                      (∃ab. ((a,b'), ab) ∈ composition ∧ ((b,a'), ab) ∈ composition)"
    apply (erule transformation.induct)
    by (auto)

  text {*
    We now show the correctness of the @{term transform} function. Because the equal
```

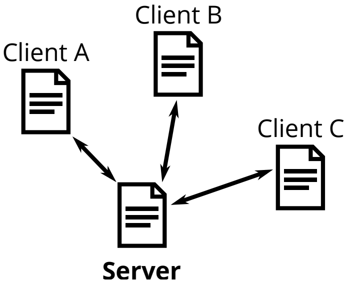
```
proof (prove): step 1

goal (9 subgoals):
 1. ∃ab. (([], []), ab) ∈ composition ∧ (([], []), ab) ∈ composition
 2. ⋀a b a' b' c. ((a, b), a', b') ∈ transformation ⟹ ∃ab. ((a, b'), ab) ∈ composition ∧ ((b, a'), ab) ∈ composition =
 3. ⋀b a' b' c. (([], b), a', b') ∈ transformation ⟹ ∃ab. (([], b'), ab) ∈ composition ∧ ((b, a'), ab) ∈ composition ⟹
 4. ⋀a b a' b' c. ((Retain # a, b), a', b') ∈ transformation ⟹ ∃ab. ((Retain # a, b'), ab) ∈ composition ∧ ((b, a'), ab)
 5. ⋀a b a' b' c. ((Delete # a, b), a', b') ∈ transformation ⟹ ∃ab. ((Delete # a, b'), ab) ∈ composition ∧ ((b, a'), ab
 6. ⋀a b a' b'. ((a, b), a', b') ∈ transformation ⟹ ∃ab. ((a, b'), ab) ∈ composition ∧ ((b, a'), ab) ∈ composition ⟹
 7. ⋀a b a' b'. ((a, b), a', b') ∈ transformation ⟹ ∃ab. ((a, b'), ab) ∈ composition ∧ ((b, a'), ab) ∈ composition ⟹
 8. ⋀a b a' b'. ((a, b), a', b') ∈ transformation ⟹ ∃ab. ((a, b'), ab) ∈ composition ∧ ((b, a'), ab) ∈ composition ⟹
 9. ⋀a b a' b'. ((a, b), a', b') ∈ transformation ⟹ ∃ab. ((a, b'), ab) ∈ composition ∧ ((b, a'), ab) ∈ composition ⟹
```

Use Cases

- **Scientific collaboration:** a small number of co-authors writing a joint proof
- **Proof review:** one user explicates content of proof to others, *e.g.* teacher to students or vice versa
- **Machine-assisted collaboration:** collaborating with a machine

Under the hood

- The basic problem: synchronisation



- Well researched solution: operational transformation

Operational Transformations

- Basic Problem:

$$\begin{tikzcd}
 & \bullet \arrow[dr, dashed, "b'"] & \\
D \arrow[ur, "a"] \arrow[dr, "b"'] & & D' \\
 & \bullet \arrow[ur, dashed, "a'"'] &
\end{tikzcd}$$

- Basic correctness:

$$\forall D.\ applyOp\ \ b'\ (applyOp\ \ a\ D) = applyOp\ \ a'\ (applyOp\ \ b\ D). \tag{1}$$

- Given by auxiliary *transform* and two equations:

$$applyOp\ \ (b \circ a)\ D = applyOp\ \ b\ (applyOp\ \ a\ D) \tag{2}$$

$$transform\ \ a\ b = \langle a', b' \rangle \Longrightarrow b' \circ a = a' \circ b \tag{3}$$

Operational Transformation: Basic Principle

Text is modified using **three** basic actions:

- *Retain* – Copy current character
- *Delete* – Drop current character
- *Insert* c – Insert c

An **operation** is a sequence of actions.

Operational Transformation: Basic Principle

Text is modified using **three** basic actions:

- *Retain* – Copy current character
- *Delete* – Drop current character
- *Insert c* – Insert c

An **operation** is a sequence of actions.

An **example**:

Input: `I P T`
Output:
Operation: [

Operational Transformation: Basic Principle

Text is modified using **three** basic actions:

- *Retain* – Copy current character
- *Delete* – Drop current character
- *Insert* c – Insert c

An **operation** is a sequence of actions.

An **example**:

Input: `P T`
Output: `I`
Operation: [*Retain*,

Operational Transformation: Basic Principle

Text is modified using **three** basic actions:

- *Retain* – Copy current character
- *Delete* – Drop current character
- *Insert c* – Insert *c*

An **operation** is a sequence of actions.

An **example**:

Input: `T`
Output: `I`
Operation: [*Retain*,
Delete,

Operational Transformation: Basic Principle

Text is modified using **three** basic actions:

- *Retain* – Copy current character
- *Delete* – Drop current character
- *Insert c* – Insert *c*

An **operation** is a sequence of actions.

An **example**:

Input:
Output: `I T`
Operation: [*Retain*,
Delete,
Retain,

Operational Transformation: Basic Principle

Text is modified using **three** basic actions:

- *Retain* – Copy current character
- *Delete* – Drop current character
- *Insert c* – Insert *c*

An **operation** is a sequence of actions.

An **example**:

Input:
Output: `I T P`
Operation: [*Retain*,
Delete,
Retain,
Insert `P`]

Operational Transformation: Basic Principle

Text is modified using **three** basic actions:

- *Retain* – Copy current character
- *Delete* – Drop current character
- *Insert* c – Insert c

An **operation** is a sequence of actions.

An **example**:

Input:
Output: `I T P`
Operation: [*Retain*,
Delete,
Retain,
Insert `P`]

Operational Transformation: Basic Principle

Text is modified using **three** basic actions:

- *Retain* – Copy current character
- *Delete* – Drop current character
- *Insert c* – Insert *c*

An **operation** is a sequence of actions.

An **example**:

Input:
Output: I T P
Operation: [*Retain*,
Delete,
Retain,
Insert P]

- Note: operations are **partial**.

- Need to consider: **composition** and **transformation**

Composing Operations

- Composing operations: case distinction on the **action**
    - Note: not simple concatenation!
- Example:

$$\begin{aligned} p &= [\mathit{Delete}, \mathit{Insert}\ \mathtt{X}, \mathit{Retain}] \\ q &= [\mathit{Retain}, \mathit{Insert}\ \mathtt{Y}, \mathit{Delete}] \\ \mathit{compose}\ a\ b &= \end{aligned}$$

- *compose* is partial.

Composing Operations

- Composing operations: case distinction on the **action**
  - Note: not simple concatenation!
- Example:

$$\begin{aligned} p &= [\mathit{Insert}\ \mathtt{X}, \mathit{Retain}] \\ q &= [\mathit{Retain}, \mathit{Insert}\ \mathtt{Y}, \mathit{Delete}] \\ \mathit{compose}\ a\ b &= [\mathit{Delete}, \end{aligned}$$

- *compose* is partial.

Composing Operations

- Composing operations: case distinction on the **action**
  - Note: not simple concatenation!
- Example:

$$\begin{aligned} p &= [\mathit{Retain}] \\ q &= [\mathit{Insert}\ \mathtt{Y}, \mathit{Delete}] \\ \mathit{compose}\ a\ b &= [\mathit{Delete}, \mathit{Insert}\ \mathtt{X}, \end{aligned}$$

- *compose* is partial.

Composing Operations

- Composing operations: case distinction on the **action**
  - Note: not simple concatenation!
- Example:

$$\begin{aligned} p &= [\mathit{Retain}] \\ q &= [\mathit{Delete}] \\ \mathit{compose}\ a\ b &= [\mathit{Delete}, \mathit{Insert}\ \mathtt{X}, \mathit{Insert}\ \mathtt{Y}, \end{aligned}$$

- *compose* is partial.

Composing Operations

- Composing operations: case distinction on the **action**
  - Note: not simple concatenation!
- Example:

$$\begin{aligned} p &= [] \\ q &= [] \\ \mathit{compose}\ a\ b &= [\mathit{Delete}, \mathit{Insert}\ \mathtt{X}, \mathit{Insert}\ \mathtt{Y}, \mathit{Delete}] \end{aligned}$$

- *compose* is partial.

Composing Operations

- Composing operations: case distinction on the **action**
  - Note: not simple concatenation!
- Example:

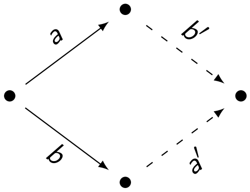
$$\begin{aligned} p &= [\mathit{Delete}, \mathit{Insert}\ \mathtt{X}, \mathit{Retain}] \\ q &= [\mathit{Retain}, \mathit{Insert}\ \mathtt{Y}, \mathit{Delete}] \\ \mathit{compose}\ a\ b &= [\mathit{Delete}, \mathit{Insert}\ \mathtt{X}, \mathit{Insert}\ \mathtt{Y}, \mathit{Delete}] \end{aligned}$$

- *compose* is partial.
- Extensional **equivalence** of operations:

$$\mathit{compose}\ a\ b \cong [\mathit{Delete}, \mathit{Delete}, \mathit{Insert}\ \mathtt{X}, \mathit{Insert}\ \mathtt{Y}]$$

Transforming Operations

- Transforming operations: pointwise completion



- Example:

$$\begin{aligned} a &= [\mathit{Insert}\ \mathtt{X}, \mathit{Retain}, \mathit{Delete}] \\ b &= [\mathit{Delete}, \mathit{Retain}, \mathit{Insert}\ \mathtt{Y}] \\ \mathit{transform}\ a\ b &= \ ([\\ & \quad\ ,[\\ & \quad\ ) \end{aligned}$$

Transforming Operations

- Transforming operations: pointwise completion

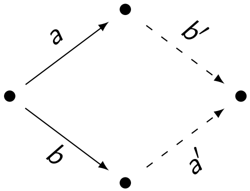
$$\begin{tikzcd}
& \bullet \arrow[dr, dashed, "b'"] & \\
\bullet \arrow[ur, "a"] \arrow[dr, "b"'] & & \bullet \\
& \bullet \arrow[ur, dashed, "a'"'] &
\end{tikzcd}$$

- Example:

$$\begin{aligned}
a &= [\mathit{Retain}, \mathit{Delete}] \\
b &= [\mathit{Delete}, \mathit{Retain}, \mathit{Insert}\ \mathtt{Y}] \\
\mathit{transform}\ a\ b &= \ ([\mathit{Insert}\ \mathtt{X}, \\
&\quad\ , [\mathit{Retain}, \\
&\quad\ )
\end{aligned}$$

Transforming Operations

- Transforming operations: pointwise completion



- Example:

$$\begin{aligned} a &= [\mathit{Delete}] \\ b &= [\mathit{Retain}, \mathit{Insert}\ \texttt{Y}] \\ \mathit{transform}\ a\ b &= \ ([\mathit{Insert}\ \texttt{X}, \mathit{Delete}, \\ &\ , [\mathit{Retain}, \\ &\ ) \end{aligned}$$

Transforming Operations

- Transforming operations: pointwise completion

$$\begin{tikzcd}
 & \bullet \arrow[dr, dashed, "b'"] & \\
\bullet \arrow[ur, "a"] \arrow[dr, "b"'] & & \bullet \\
 & \bullet \arrow[ur, dashed, "a'"'] &
\end{tikzcd}$$

- Example:

$$\begin{aligned}
a &= [] \\
b &= [\mathit{Insert}\ \texttt{Y}] \\
\mathit{transform}\ a\ b &= ([\mathit{Insert}\ \texttt{X}, \mathit{Delete}, \\
&\quad , [\mathit{Retain}, \mathit{Delete}, \\
&\quad)
\end{aligned}$$

Transforming Operations

- Transforming operations: pointwise completion

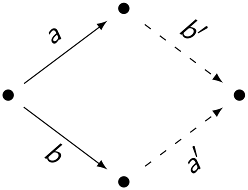
$$\begin{tikzcd}
 & \bullet \arrow[dr, dashed, "b'"] & \\
\bullet \arrow[ur, "a"] \arrow[dr, "b"'] & & \bullet \\
 & \bullet \arrow[ur, dashed, "a'"'] &
\end{tikzcd}$$

- Example:

$$\begin{aligned}
a &= [] \\
b &= [] \\
transform\ a\ b &= ([Insert\ \mathtt{X}, Delete, Retain] \\
&\quad , [Retain, Delete, Insert\ \mathtt{Y}] \\
&\quad)
\end{aligned}$$

Transforming Operations

- Transforming operations: pointwise completion



- Example:

$$\begin{aligned}
a &= [\mathit{Insert}\ \texttt{X}, \mathit{Retain}, \mathit{Delete}] \\
b &= [\mathit{Delete}, \mathit{Retain}, \mathit{Insert}\ \texttt{Y}] \\
\mathit{transform}\ a\ b &= \ ([\mathit{Insert}\ \texttt{X}, \mathit{Delete}, \mathit{Retain}] \\
&\ \ , [\mathit{Retain}, \mathit{Delete}, \mathit{Insert}\ \texttt{Y}] \\
&\ \ )
\end{aligned}$$

Formalisation: Correctness

- Correctness of *compose* (**??**):

theorem *composeCorrect*:

$$\llbracket\ \mathit{compose}\ a\ b = \mathit{Some}\ \mathit{ab};\ \mathit{applyOp}\ a\ d = \mathit{Some}\ d';\ \mathit{applyOp}\ b\ d' = \mathit{Some}\ d''\ \rrbracket$$
$$\Longrightarrow \mathit{applyOp}\ \mathit{ab}\ d = \mathit{Some}\ d''$$

- Correctness of *transform* (**??**):

theorem *transformCorrect*: $\mathit{transform}\ a\ b = \mathit{Some}\ (a',b')$

$$\Longrightarrow \mathit{compose}\ a\ b' \neq \mathit{None} \wedge \mathit{compose}\ a\ b' = \mathit{compose}\ b\ a'$$

- To show previous lemmas, need to construct **graphs** of the partial functions.

- Application: **generate** Scala code from Isabelle

Annotations

- **Two** types of annotation actions
  - *Plain n* – Retain *n* characters
  - *Annotate n c* – Annotate *n* characters with annotation *c*

- Annotations ≈ identity operations with side-effects
- No interference with operations – can be handled separately

lemma *transformIdL*:
$$\mathit{transform}\ (\mathit{ident}\ (\mathit{inputLength}\ b))\ b = \mathit{Some}\ (\mathit{ident}\ (\mathit{outputLength}\ b),\ b)$$

- Multiple named annotations per collaborator
- Selections, syntax coloring, substitutions, tooltips, completion, etc.

The Control Algorithm - Server

- Purpose:
  - sequentialise concurrent operations
  - distribute transformed operations

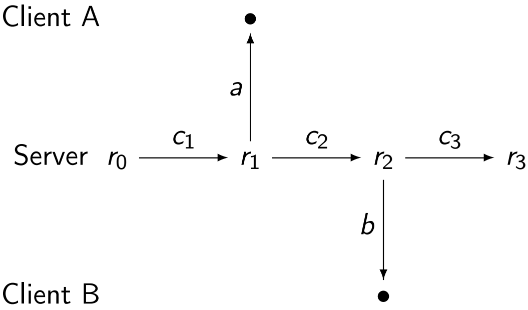
Client A

Server $$\begin{tikzcd} r_0 \arrow[r, "c_1"] & r_1 \arrow[r, "c_2"] & r_2 \arrow[r, "c_3"] & r_3 \end{tikzcd}$$

Client B

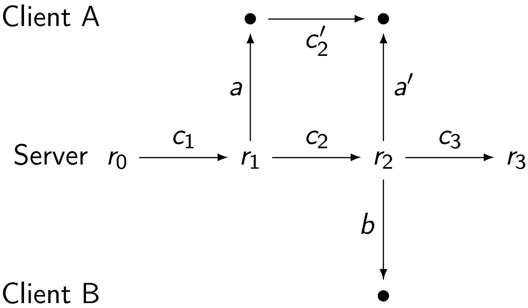
The Control Algorithm - Server

- Purpose:
  - sequentialise concurrent operations
  - distribute transformed operations



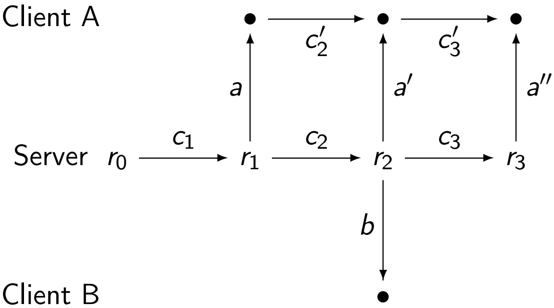
The Control Algorithm - Server

- Purpose:
  - sequentialise concurrent operations
  - distribute transformed operations



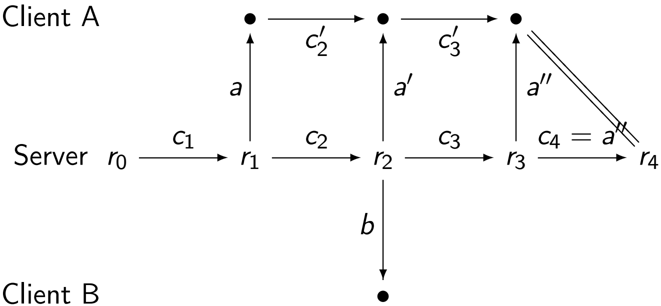
The Control Algorithm - Server

- Purpose:
  - sequentialise concurrent operations
  - distribute transformed operations



The Control Algorithm - Server

- Purpose:
  - sequentialise concurrent operations
  - distribute transformed operations



The Control Algorithm - Server

- Purpose:
  - sequentialise concurrent operations
  - distribute transformed operations

$$\begin{tikzcd}
\text{Client A} & & \bullet \arrow[r, "c_2'"'] & \bullet \arrow[r, "c_3'"'] & \bullet \arrow[dr, equal] & \\
\text{Server} & r_0 \arrow[r, "c_1"] & r_1 \arrow[u, "a"] \arrow[r, "c_2"] & r_2 \arrow[u, "a'"'] \arrow[r, "c_3"] \arrow[d, "b"'] & r_3 \arrow[u, "a''"'] \arrow[r, "c_4 = a''"] \arrow[d, "b'"] & r_4 \\
\text{Client B} & & & \bullet \arrow[r, "c_3''"'] & \bullet &
\end{tikzcd}$$

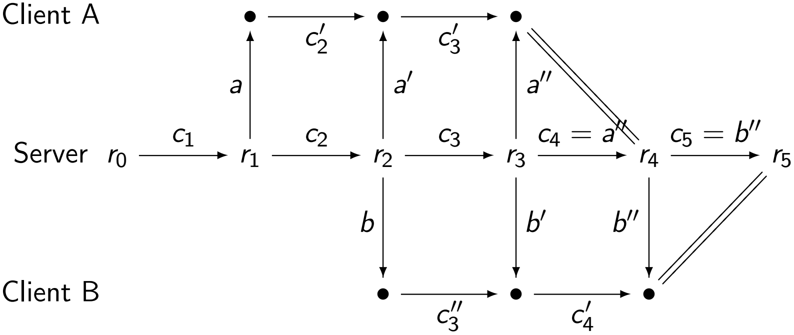
The Control Algorithm - Server

- Purpose:
  - sequentialise concurrent operations
  - distribute transformed operations

$$\begin{tikzcd}
\text{Client A} & & \bullet \arrow[r, "c_2'"'] & \bullet \arrow[r, "c_3'"'] & \bullet \arrow[dr, equal] & \\
\text{Server} & r_0 \arrow[r, "c_1"] & r_1 \arrow[r, "c_2"] \arrow[u, "a"] & r_2 \arrow[r, "c_3"] \arrow[u, "a'"'] \arrow[d, "b"'] & r_3 \arrow[r, "c_4 = a''"] \arrow[u, "a''"'] \arrow[d, "b'"'] & r_4 \arrow[d, "b''"'] \\
\text{Client B} & & & \bullet \arrow[r, "c_3''"'] & \bullet \arrow[r, "c_4'"'] & \bullet
\end{tikzcd}$$

The Control Algorithm - Server

- Purpose:
  - sequentialise concurrent operations
  - distribute transformed operations



The Control Algorithm - Server

- Purpose:
  - sequentialise concurrent operations
  - distribute transformed operations

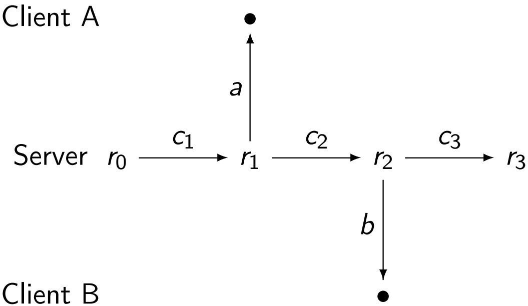
Client A

Server $r_0 \xrightarrow{c_1} r_1 \xrightarrow{c_2} r_2 \xrightarrow{c_3} r_3$

Client B

The Control Algorithm - Server

- Purpose:
  - sequentialise concurrent operations
  - distribute transformed operations



The Control Algorithm - Server

- Purpose:
  - sequentialise concurrent operations
  - distribute transformed operations

$$\begin{tikzcd}
\text{Client A} & & \bullet & & \\
\text{Server} & r_0 \arrow[r, "c_1"] & r_1 \arrow[u, "a"] \arrow[r, "c_2"] & r_2 \arrow[r, "c_3"] \arrow[d, "b"'] & r_3 \arrow[d, "b'"] \\
\text{Client B} & & & \bullet \arrow[r, "c_3''"'] & \bullet
\end{tikzcd}$$

The Control Algorithm - Server

- Purpose:
    - sequentialise concurrent operations
    - distribute transformed operations

$$\begin{tikzcd}
\text{Client A} & & \bullet & & & \\
\text{Server} & r_0 \arrow[r, "c_1"] & r_1 \arrow[u, "a"] \arrow[r, "c_2"] & r_2 \arrow[r, "c_3"] \arrow[d, "b"'] & r_3 \arrow[r, "c_4 = b'"] \arrow[d, "b'"] & r_4 \arrow[dl, equal] \\
\text{Client B} & & & \bullet \arrow[r, "c_3''"'] & \bullet &
\end{tikzcd}$$

The Control Algorithm - Server

- Purpose:
  - sequentialise concurrent operations
  - distribute transformed operations

$$\begin{tikzcd}
\text{Client A} & & \bullet \arrow[r, swap, "c_2'"] & \bullet & & \\
\text{Server} & r_0 \arrow[r, "c_1"] & r_1 \arrow[r, "c_2"] \arrow[u, "a"] & r_2 \arrow[r, "c_3"] \arrow[u, swap, "a'"] \arrow[d, swap, "b"] & r_3 \arrow[r, "c_4 = b'"] \arrow[d, "b'"] & r_4 \arrow[dl, equal] \\
\text{Client B} & & & \bullet \arrow[r, swap, "c_3''"] & \bullet &
\end{tikzcd}$$

The Control Algorithm - Server

- Purpose:
  - sequentialise concurrent operations
  - distribute transformed operations

$$\begin{tikzcd}
\text{Client A} & & \bullet \arrow[r, "c_2'"'] & \bullet \arrow[r, "c_3'"'] & \bullet & \\
\text{Server} & r_0 \arrow[r, "c_1"] & r_1 \arrow[u, "a"] \arrow[r, "c_2"] & r_2 \arrow[u, "a'"'] \arrow[r, "c_3"] \arrow[d, "b"'] & r_3 \arrow[u, "a''"'] \arrow[r, "c_4 = b'"] \arrow[d, "b'"] & r_4 \arrow[dl, equal] \\
\text{Client B} & & & \bullet \arrow[r, "c_3''"'] & \bullet &
\end{tikzcd}$$

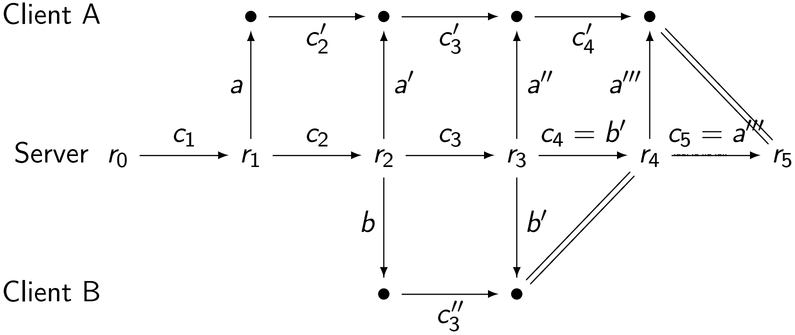
The Control Algorithm - Server

- Purpose:
  - sequentialise concurrent operations
  - distribute transformed operations

$$\begin{tikzcd}
\text{Client A} & & \bullet \arrow[r, swap, "c_2'"] & \bullet \arrow[r, swap, "c_3'"] & \bullet \arrow[r, swap, "c_4'"] & \bullet \\
\text{Server} & r_0 \arrow[r, "c_1"] & r_1 \arrow[r, "c_2"] \arrow[u, "a"] & r_2 \arrow[r, "c_3"] \arrow[u, swap, "a'"] \arrow[d, swap, "b"] & r_3 \arrow[r, "c_4 = b'"] \arrow[u, swap, "a''"] \arrow[d, "b'"] & r_4 \arrow[u, "a'''"] \arrow[dl, equal] \\
\text{Client B} & & & \bullet \arrow[r, swap, "c_3''"] & \bullet &
\end{tikzcd}$$

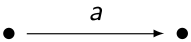
The Control Algorithm - Server

- Purpose:
  - sequentialise concurrent operations
  - distribute transformed operations



The Control Algorithm - Client

- Purpose: buffer operations while waiting for acknowledgment



Revision r

The Control Algorithm - Client

- Purpose: buffer operations while waiting for acknowledgment

The Control Algorithm - Client

- Purpose: buffer operations while waiting for acknowledgment

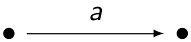
$$\begin{tikzcd}
\bullet \arrow[r, "a"] \arrow[d, "c"'] & \bullet \arrow[d, "c'"'] \\
\bullet \arrow[r, "a'"'] & \bullet
\end{tikzcd}$$

Revision r

Revision $r+1$

The Control Algorithm - Client

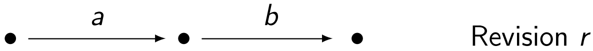
- Purpose: buffer operations while waiting for acknowledgment



Revision r

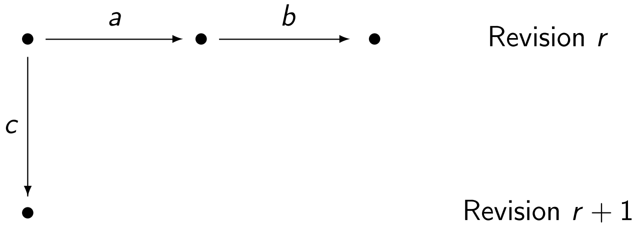
The Control Algorithm - Client

- Purpose: buffer operations while waiting for acknowledgment



The Control Algorithm - Client

- Purpose: buffer operations while waiting for acknowledgment



The Control Algorithm - Client

- Purpose: buffer operations while waiting for acknowledgment

$$\begin{tikzcd}
\bullet \arrow[r, "a"] \arrow[d, "c"'] & \bullet \arrow[r, "b"] \arrow[d, "c'"'] & \bullet & \text{Revision } r \\
\bullet \arrow[r, "a'"'] & \bullet & & \text{Revision } r+1
\end{tikzcd}$$

The Control Algorithm - Client

- Purpose: buffer operations while waiting for acknowledgment

$$\begin{tikzcd}
\bullet \arrow[r, "a"] \arrow[d, "c"'] & \bullet \arrow[r, "b"] \arrow[d, "c'"'] & \bullet \arrow[d, "c''"] & \text{Revision } r \\
\bullet \arrow[r, "a'"'] & \bullet \arrow[r, "b'"'] & \bullet & \text{Revision } r+1
\end{tikzcd}$$

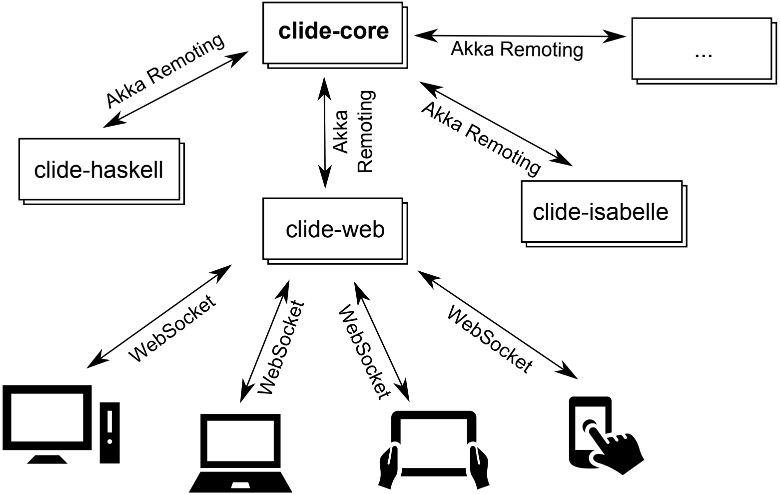
The Control Algorithm - Client

- Purpose: buffer operations while waiting for acknowledgment

$$\begin{tikzcd}
\bullet \arrow[r, "a"] \arrow[d, "c"'] & \bullet \arrow[r, "b"] \arrow[d, "c'"'] & \bullet \arrow[d, "c''"] & \text{Revision } r \\
\bullet \arrow[r, "a'"'] & \bullet \arrow[r, "b'"'] & \bullet & \text{Revision } r+1
\end{tikzcd}$$

- Problem: web client must be implemented in JavaScript

System Architecture: Components



Universal Collaboration

- Clide is generic: Isabelle is just one particular collaborator based on the great **PIDE framework**.
- Paradigm of **universal collaboration**: document-centered collaborative development.
- Allows easy development of new assistants: just define interaction with document, synchronisation and integration provided by Clide.
- Examples: prototypical Haskell and Scala IDE

Concluding Remarks

- Clide: Interactive Collaborative Real-Time Theorem Proving
  - Based on formalisation of Operational Transformations in Isabelle
  - Compares well to Isabelle/jEdit or ProofGeneral
  - Flexible system architecture built on Scala, Akka
- Clide is **generic**
  - Prototypical Haskell and Scala instantiations
  - Novel concept of **universal collaboration**