Formal Meta-Theory of Bisimulation Up To in Abella

Kaustuv Chaudhuri [1] Matteo Cimini [1] Dale Miller [1,2]

[1] INRIA Saclay, France
[2] LIX, Ecole Polytechnique, France

MSC Week 5: Concurrency, Logic, and Types

2014-02-11

Outline

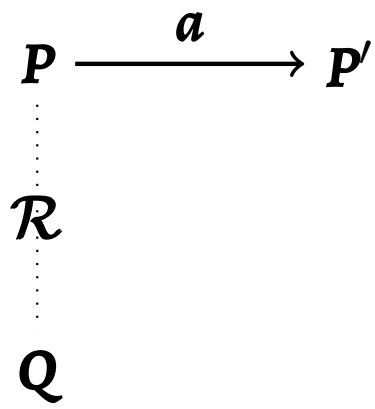
- **Formalizing Bisimulation in Abella**
- **Bisimulation Up To**
- **Major Up To Techniques (for CCS)**
- **Case of the π-Calculus**
- **Perspectives**

Bisimulation in Abella

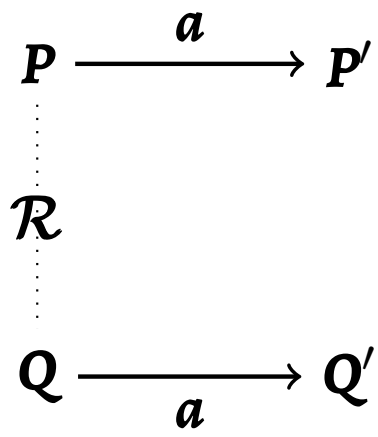
`http://abella-prover.org`

Formally: Progress

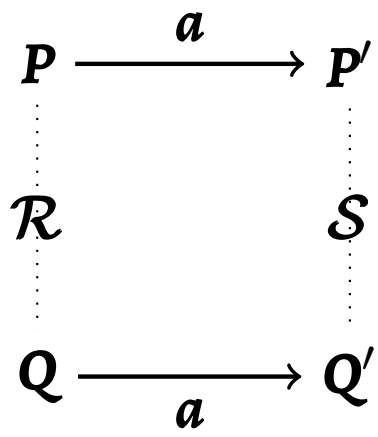
Formally: Progress



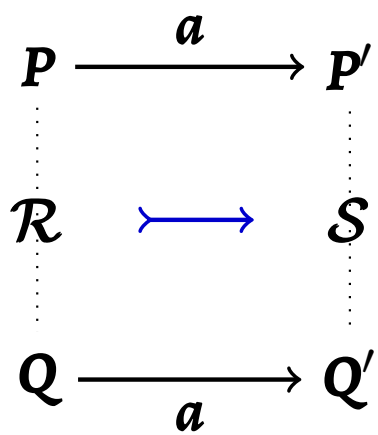
Formally: Progress



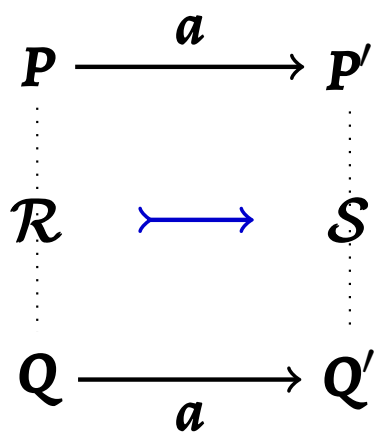
Formally: Progress



Formally: Progress

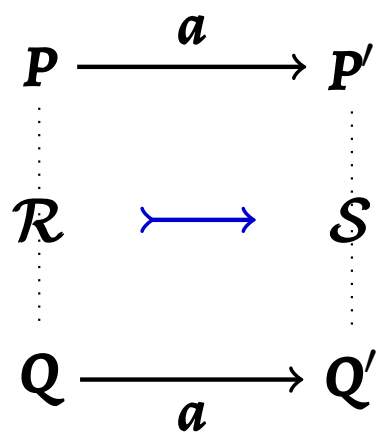


Formally: Progress



$$\mathcal{R} \rightarrowtail \mathcal{S} \triangleq$$
$$\forall P, Q.\ P\,\mathcal{R}\,Q \implies$$
$$\left(\forall a, P'.\ P \xrightarrow{a} P' \implies \exists Q'.\ Q \xrightarrow{a} Q' \wedge P'\,\mathcal{S}\,Q'\right)$$

Formally: Progress



$$\mathcal{R} \rightarrowtail \mathcal{S} \triangleq$$
$$\forall P, Q.\ P\,\mathcal{R}\,Q \implies$$
$$\left(\forall a, P'.\ P \xrightarrow{a} P' \implies \exists Q'.\ Q \xrightarrow{a} Q' \wedge P'\,\mathcal{S}\,Q'\right) \wedge$$
$$\left(\forall a, Q'.\ Q \xrightarrow{a} Q' \implies \exists P'.\ P \xrightarrow{a} P' \wedge P'\,\mathcal{S}\,Q'\right)$$

Formally: Bisimulation and Bisimilarity

A relation $\mathcal{R} \subseteq \text{Proc} \times \text{Proc}$ is a bisimulation if $\mathcal{R} \rightarrowtail \mathcal{R}$.

Formally: Bisimulation and Bisimilarity

A relation $\mathcal{R} \subseteq \text{Proc} \times \text{Proc}$ is a **bisimulation** if $\mathcal{R} \rightarrowtail \mathcal{R}$.

Bisimilarity ($\sim \subseteq \text{Proc} \times \text{Proc}$) is:

- the union of all bisimulations

Formally: Bisimulation and Bisimilarity

A relation $\mathcal{R} \subseteq \text{Proc} \times \text{Proc}$ is a **bisimulation** if $\mathcal{R} \rightarrowtail \mathcal{R}$.

Bisimilarity ($\sim \subseteq \text{Proc} \times \text{Proc}$) is:

- the union of all bisimulations, or, equivalently
- the greatest fixed point of $\rightarrowtail$.

Model Checking

The bisimulation proof method is well suited to exhaustive search using automated techniques.

Model Checking

The bisimulation proof method is well suited to exhaustive search using automated techniques.

To show P and Q are similar,

1. **Initialize the *simulation set* to empty**
2. **If (P, Q) is in the simulation set, succeed.**
   Else, add (P, Q) to the simulation set
3. **For every P' with $P \xrightarrow{a} P'$,**
   a. **Find a Q' with $Q \xrightarrow{a} Q'$** **(failure, backtracking)**
   b. **Recursively-goto step 2 with P' and Q'**

Model Checking

The bisimulation proof method is well suited to exhaustive search using automated techniques.

To show P and Q are bisimilar,

1. **Initialize the *bisimulation set* to empty**
2. **If (P, Q) is in the bisimulation set, succeed.**
   Else, add (P, Q) to the bisimulation set
3. **For every P' with $P \xrightarrow{a} P'$,**
   a. **Find a Q' with $Q \xrightarrow{a} Q'$** **(failure, backtracking)**
   b. **Recursively-goto step 2 with P' and Q'**
4. **For every Q' with $Q \xrightarrow{a} Q'$,**
   a. **Find a P' with $P \xrightarrow{a} P'$** **(failure, backtracking)**
   b. **Recursively-goto step 2 with P' and Q'**

Abella

http://abella-prover.org

- Simply typed intuitionistic logic
- Least and greatest *relational* fixed point definitions
- An interactive tactics-based proof environment

- Quantification over λ-terms and higher-order syntax
- Equality of λ-terms upto $\alpha\beta\eta$
- Binding and scope issues reduced to quantifier alternation instead of side conditions
- Generic quantification

- Built-in executable *specification logic* (subset of λProlog)

Abella

`http://abella-prover.org`

- Simply typed intuitionistic logic
- Least and greatest *relational* fixed point definitions
- An interactive tactics-based proof environment

- Quantification over λ-terms and higher-order syntax
- Equality of λ-terms upto $\alpha\beta\eta$
- Binding and scope issues reduced to quantifier alternation instead of side conditions
- Generic quantification

- Built-in executable *specification logic* (subset of λProlog)

In Abella

$$P \sim Q \triangleq_{\nu}$$
$$\left(\forall a, P'.\ P \xrightarrow{a} P' \implies \exists Q'.\ Q \xrightarrow{a} Q' \wedge P' \sim Q'\right) \wedge$$
$$\left(\forall a, Q'.\ Q \xrightarrow{a} Q' \implies \exists P'.\ P \xrightarrow{a} P' \wedge P' \sim Q'\right)$$

In Abella

$$P \sim Q \triangleq_{\nu} \left(\forall a, P'.\ P \xrightarrow{a} P' \implies \exists Q'.\ Q \xrightarrow{a} Q' \wedge P' \sim Q'\right) \wedge \left(\forall a, Q'.\ Q \xrightarrow{a} Q' \implies \exists P'.\ P \xrightarrow{a} P' \wedge P' \sim Q'\right)$$

```
CoDefine bisim : proc → proc → prop by
  bisim P Q ≜
     (forall A P', one P A P' →
        exists Q', one Q A Q' ∧ bisim P' Q')
   ∧ (forall A Q', one Q A Q' →
        exists P', one P A P' ∧ bisim P' Q').
```

In Abella

$$P \sim Q \triangleq_{\nu} \left(\forall a, P'.\ P \xrightarrow{a} P' \implies \exists Q'.\ Q \xrightarrow{a} Q' \wedge P' \sim Q'\right) \wedge \left(\forall a, Q'.\ Q \xrightarrow{a} Q' \implies \exists P'.\ P \xrightarrow{a} P' \wedge P' \sim Q'\right)$$

```
Define one : proc → action → proc → prop by
  ...
```

```
CoDefine bisim : proc → proc → prop by
  bisim P Q ≜
     (forall A P', one P A P' →
        exists Q', one Q A Q'  ∧ bisim P' Q')
   ∧ (forall A Q', one Q A Q' →
        exists P', one P A P'  ∧ bisim P' Q').
```

In Abella

$$P \sim Q \triangleq_{\nu} \left(\forall a, P'.\ P \xrightarrow{a} P' \implies \exists Q'.\ Q \xrightarrow{a} Q' \wedge P' \sim Q'\right) \wedge \left(\forall a, Q'.\ Q \xrightarrow{a} Q' \implies \exists P'.\ P \xrightarrow{a} P' \wedge P' \sim Q'\right)$$

```
Kind proc, action     type.
...
```

```
Define one : proc → action → proc → prop by
  ...
```

```
CoDefine bisim : proc → proc → prop by
  bisim P Q ≜
     (forall A P', one P A P' →
        exists Q', one Q A Q'  ∧ bisim P' Q')
   ∧ (forall A Q', one Q A Q' →
        exists P', one P A P'  ∧ bisim P' Q').
```

In Abella: CCS (signature)

```
Kind name,action,proc type.

% Action constructors
Type tau          action.
Type up,dn        name → action.

% Process constructors
Type null         proc.
Type plus,par     proc   → proc → proc.
Type act          action → proc → proc.
Type repl         proc   → proc.

% Dual actions
Define dual : action → action → prop by
  dual (up A) (dn A)
; dual (dn A) (up A).
```

Digression: Definitions by Pattern Matching

```
Define dual : action → action → prop by
  dual (up X) (dn X)
; dual (dn X) (up X).

Define dual' : action → action → prop by
  dual A B ≜
    (exists X, A = up X ∧ B = dn X) ∨
    (exists X, A = dn X ∧ B = up X).
```

In Abella: CCS (steps)

```
Define one : proc → action → proc → prop by
  one (act A P) A P
; one (plus P1 P2) A Q              ≜ one P1 A Q
; one (plus P1 P2) A Q              ≜ one P2 A Q
; one (par P Q) A (par P1 Q)        ≜ one P A P1
; one (par P Q) A (par P Q1)        ≜ one Q A Q1
; one (repl P) A (par (repl P) Q)   ≜ one P A Q
; one (par P Q) tau (par P1 Q1)     ≜
    exists A B, dual A B ∧ one P A P1 ∧ one Q B Q1
; one (repl P) tau (par (repl P) (par Q R)) ≜
    exists A B, dual A B ∧ one P A Q ∧ one P B R.
```

Example Meta-Theorem: Bisimilarity is a Congruence

```
Theorem bisim_plus1_cong :
  forall P Q R,
    bisim P Q → bisim (plus P R) (plus Q R).
```

Example Meta-Theorem: Bisimilarity is a Congruence

```
Theorem bisim_plus1_cong :
  forall P Q R,
    bisim P Q → bisim (plus P R) (plus Q R).

coinduction.
intros PbiQ.    % PbiQ : bisim P Q
case PbiQ. split.
  intros PaP1. % PaP1 : one P A P1
  case PaP1.
    ...

  intros QaQ1. % QaQ1 : one Q A Q1
    ...
```

Bisimulation Up To

Infinite Bisimulation Sets

$$!(a + b) \sim !a \mid !b$$

Infinite Bisimulation Sets

$$!(a + b) \sim !a \mid !b$$

$$!(a + b) \xrightarrow{a} !(a + b) \mid 0 \xrightarrow{b} !(a + b) \mid 0 \mid 0$$

Infinite Bisimulation Sets

$$!(a + b) \sim !a \mid !b$$

$$!(a + b) \xrightarrow{a} !(a + b) \mid 0 \xrightarrow{b} !(a + b) \mid 0 \mid 0$$

$$!a \mid !b \xrightarrow{a} (!a \mid 0) \mid !b \xrightarrow{b} (!a \mid 0) \mid (!b \mid 0)$$

Infinite Bisimulation Sets

$$\boxed{!(a+b) \sim\ !a \mid !b}$$

$$!(a+b) \xrightarrow{a} !(a+b) \mid 0 \xrightarrow{b} !(a+b) \mid 0 \mid 0$$

$$!a \mid !b \xrightarrow{a} (!a \mid 0) \mid !b \xrightarrow{b} (!a \mid 0) \mid (!b \mid 0)$$

In this case we have a closed form "generalization"

$$!(a+b) \mid 0^{m+n} \quad \sim \quad (!a \mid 0^{m}) \mid (!b \mid 0^{n})$$

Infinite Bisimulation Sets

$$!!a \sim !a$$

$$!!a \xrightarrow{a} !!a \mid (!a \mid 0) \xrightarrow{a} !!a \mid (!a \mid 0) \mid (!a \mid 0) \xrightarrow{a} !!a \mid (!a \mid 0 \mid 0) \mid (!a \mid 0)$$

$$!a \xrightarrow{a} !a \mid 0 \xrightarrow{a} !a \mid 0 \mid 0 \xrightarrow{a} !a \mid 0 \mid 0 \mid 0 \mid 0$$

Infinite Bisimulation Sets

$$!!a \sim !a$$

$$!!a \xrightarrow{a} !!a \mid (!a \mid 0) \xrightarrow{a} !!a \mid (!a \mid 0) \mid (!a \mid 0) \xrightarrow{a} !!a \mid (!a \mid 0 \mid 0) \mid (!a \mid 0)$$

$$!a \xrightarrow{a} !a \mid 0 \xrightarrow{a} !a \mid 0 \mid 0 \xrightarrow{a} !a \mid 0 \mid 0 \mid 0 \mid 0$$

Closed form for the generalization is difficult!

Bisimulation Up To

[Milner, Sangiorgi, Pous, ...]

Simple idea:

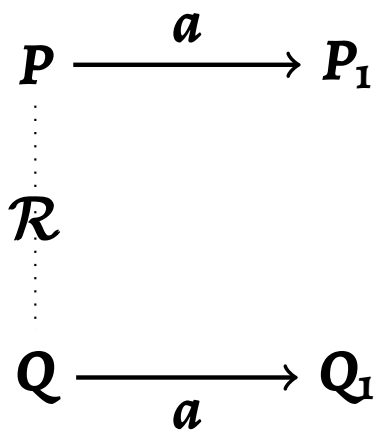
> ***Before making the recursive call to the procedure, modify the process pair to a more reasonable form.***

Example:

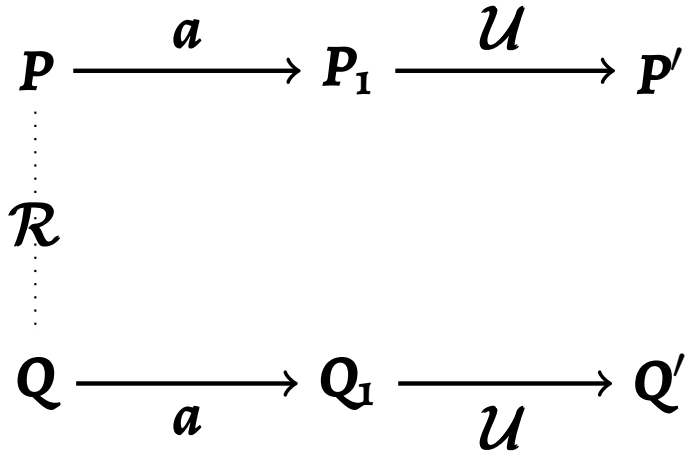
$$!!a \xrightarrow{a} !!a \mid (!a \mid 0) \rightsquigarrow !!a$$

$$!a \xrightarrow{a} !a \mid 0 \rightsquigarrow !a$$

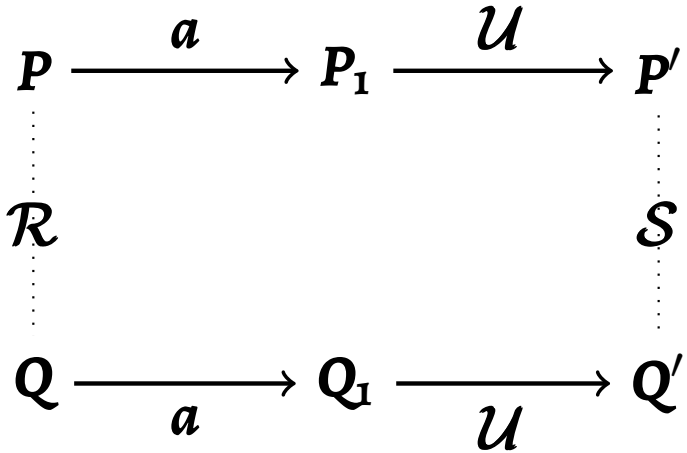
Progress Up To



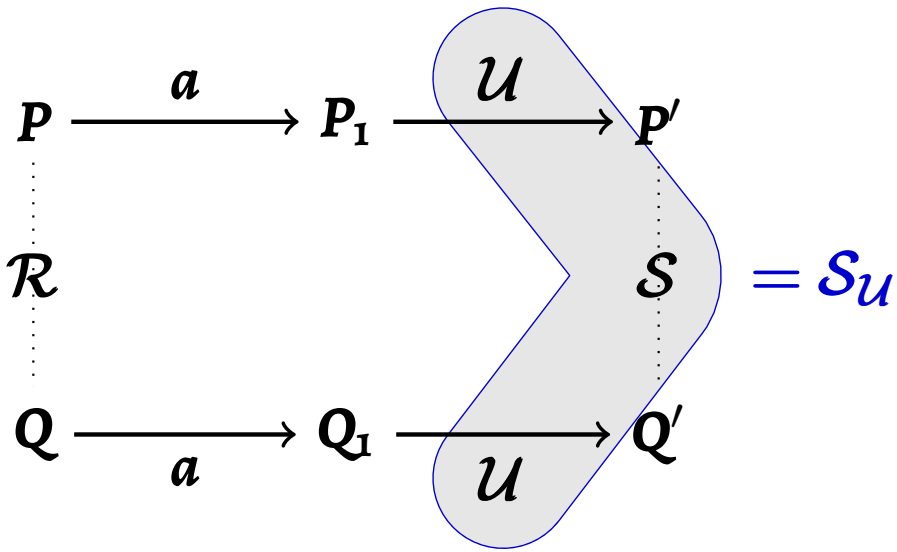
Progress Up To



Progress Up To



Progress Up To



Progress Up To

$$(\mathcal{U} \subseteq \mathbf{Proc} \times \mathbf{Proc} \times \mathbf{Proc} \times \mathbf{Proc})$$

$$P \,\mathcal{R}_{\mathcal{U}}\, Q \triangleq \exists P', Q'.\ \mathcal{U}(P, P', Q, Q') \wedge P' \,\mathcal{R}\, Q'$$

$$\begin{aligned}
&\mathcal{R} \rightarrowtail_{\mathcal{U}} \mathcal{S} \triangleq \\
&\quad \forall P, Q.\ P \,\mathcal{R}\, Q \implies \\
&\qquad \left(\forall a, P'.\ P \xrightarrow{a} P' \implies \right. \\
&\qquad\qquad \left. \exists Q'.\ Q \xrightarrow{a} Q' \wedge P' \,\mathcal{S}_{\mathcal{U}}\, Q'\right) \wedge \\
&\qquad \left(\forall a, Q'.\ Q \xrightarrow{a} Q' \implies \right. \\
&\qquad\qquad \left. \exists P'.\ P \xrightarrow{a} P' \wedge P' \,\mathcal{S}_{\mathcal{U}}\, Q'\right)
\end{aligned}$$

Bisimulation and Bisimilarity Up To

Given a $\mathcal{U} \subseteq \text{Proc} \times \text{Proc} \times \text{Proc} \times \text{Proc}$,

$\mathcal{R} \subseteq \text{Proc} \times \text{Proc}$ **is a bisimulation up to** $\mathcal{U}$ **if** $\mathcal{R} \rightarrowtail_{\mathcal{U}} \mathcal{R}$.

Bisimilarity up to $\mathcal{U}$ ($\sim_{\mathcal{U}} \subseteq \text{Proc} \times \text{Proc}$) **is:**

- **the union of all bisimulations up to** $\mathcal{U}$**, or, equivalently**
- **the greatest fixed point of** $\rightarrowtail_{\mathcal{U}}$.

Bisimulation and Bisimilarity Up To

Given a $\mathcal{U} \subseteq \text{Proc} \times \text{Proc} \times \text{Proc} \times \text{Proc}$,

$\mathcal{R} \subseteq \text{Proc} \times \text{Proc}$ **is a bisimulation up to** $\mathcal{U}$ **if** $\mathcal{R} \rightarrowtail_{\mathcal{U}} \mathcal{R}$.

Bisimilarity up to $\mathcal{U}$ ($\sim_{\mathcal{U}} \subseteq \text{Proc} \times \text{Proc}$) **is:**

- **the union of all bisimulations up to** $\mathcal{U}$**, or, equivalently**
- **the greatest fixed point of** $\rightarrowtail_{\mathcal{U}}$.

Note: $\sim \; = \; \sim_{\mathrm{Id}}$ **where:**

$$\mathrm{Id}(P, P', Q, Q') \triangleq P' = P \wedge Q' = Q$$

Up To Techniques in Abella

```
Define upto : (proc → proc → proc → proc → prop)
           → (proc → proc → prop)
           → (proc → proc → prop) by
  upto UT Rel P Q ≜
    exists P' Q', UT P P' Q Q' ∧ Rel P' Q'.

Define bisupto : (proc → proc → proc → proc → prop)
              → (proc → proc → prop) by
  bisupto UT P Q ≜
     (forall A P', one P A P' →
        exists Q', one Q A Q' ∧ upto UT (bisupto UT) P' Q')
   ∧ (forall A Q', one Q A Q' →
        exists P', one P A P' ∧ upto UT (bisupto UT) P' Q').
```

Up To Techniques in Abella

```
Define upto : (proc → proc → proc → proc → prop)
           → (proc → proc → prop)
           → (proc → proc → prop) by
  upto UT Rel P Q ≜
    exists P' Q', UT P P' Q Q' ∧ Rel P' Q'.

Define bisupto : (proc → proc → proc → proc → prop)
              → (proc → proc → prop) by
  bisupto UT P Q ≜
     (forall A P', one P A P' →
        exists Q', one Q A Q' ∧ upto UT (bisupto UT) P' Q')
   ∧ (forall A Q', one Q A Q' →
        exists P', one P A P' ∧ upto UT (bisupto UT) P' Q').
```

Up To Techniques in Abella

```
Define bisupto : (proc → proc → proc → proc → prop)
                 → (proc → proc → prop) by
  bisupto UT P Q ≜
     (forall A P1, one P A P1 →
        exists Q1, one Q A Q1 ∧
          exists P2 Q2, UT P1 P2 Q1 Q2 ∧ bisupto UT P2 Q2)
   ∧ (forall A Q1, one Q A Q1 →
        exists P1, one P A P1 ∧
          exists P2 Q2, UT P1 P2 Q1 Q2 ∧ bisupto UT P2 Q2).
```

Soundness

Not every bisimulation up to is a bisimulation.

Soundness

Not every bisimulation up to is a bisimulation.

Think: $\mathcal{U}(P, P', Q, Q') \triangleq P' = P \wedge Q' = P$.

Soundness

Not every bisimulation up to is a bisimulation.

Think: $\mathcal{U}(P, P', Q, Q') \triangleq P' = P \wedge Q' = P$.

An up to technique $\mathcal{U}$ is sound if

- **the fixed points of $\rightarrowtail_{\mathcal{U}}$ are contained in the fixed points of $\rightarrowtail$**

Soundness

Not every bisimulation up to is a bisimulation.

Think: $\mathcal{U}(P, P', Q, Q') \triangleq P' = P \wedge Q' = P$.

An up to technique $\mathcal{U}$ is sound if

- **the fixed points of $\rightarrowtail_{\mathcal{U}}$ are contained in the fixed points of $\rightarrowtail$, or, equivalently**
- **if $\sim_{\mathcal{U}} \subseteq \sim$.**

Soundness in Abella

```
Define sound : (proc → proc → proc → proc → prop)
             → prop by
  sound UT ≜
    forall P Q, bisupto UT P Q → bisim P Q.
```

Up To Techniques

for CCS

Up To Bisimilarity

The up to bisimilarity technique $\mathcal{B}$ is given by:

$$\mathcal{B}(P, P', Q, Q') \triangleq P \sim P' \wedge Q \sim Q'$$

Up To Bisimilarity

The up to bisimilarity technique $\mathcal{B}$ is given by:

$$\mathcal{B}(P, P', Q, Q') \triangleq P \sim P' \wedge Q \sim Q'$$

Let $\mathcal{R}$ be a bisimulation up to bisimilarity. Then:

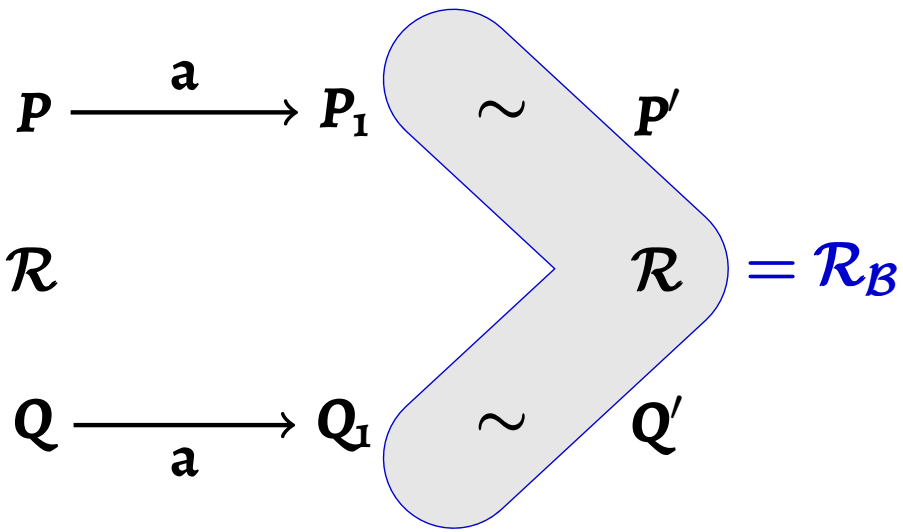
$$\begin{array}{ccc} P & \xrightarrow{a} P_1 & \sim & P' \\ \mathcal{R} & & & \mathcal{R} \\ Q & \xrightarrow[a]{} Q_1 & \sim & Q' \end{array}$$

Up To Bisimilarity

The up to bisimilarity technique $\mathcal{B}$ is given by:

$$\mathcal{B}(P, P', Q, Q') \triangleq P \sim P' \wedge Q \sim Q'$$

Let $\mathcal{R}$ be a bisimulation up to bisimilarity. Then:



Up To Bisimilarity

```
Define bisim_t : proc → proc → proc → proc → prop by
  bisim_t P1 P2 Q1 Q2 ≜ bisim P1 P2 ∧ bisim Q1 Q2.
```

Can show (in Abella):

- $!!a \sim_{\mathcal{B}} !a$
- $!(a + b) \sim_{\mathcal{B}} !a \mid !b$

Up To Bisimilarity

```
Define bisim_t : proc → proc → proc → proc → prop by
  bisim_t P1 P2 Q1 Q2 ≜ bisim P1 P2 ∧ bisim Q1 Q2.
```

Can show (in Abella):

- $!!a \sim_{\mathcal{B}} !a$
- $!(a + b) \sim_{\mathcal{B}} !a \mid !b$

```
Theorem example_2 : forall A B,
  bisupto bisim_t (repl (plus (act (dn A) null) (act (dn B) null)))
                  (par (repl (act (dn A) null)) (repl (act (dn B) null))).
coinduction. intros. unfold. split.
  % ...
```

Up To Bisimilarity

```
Define bisim_t : proc → proc → proc → proc → prop by
  bisim_t P1 P2 Q1 Q2 ≜ bisim P1 P2 ∧ bisim Q1 Q2.
```

Can show (in Abella):

- $!!a \sim_{\mathcal{B}} !a$
- $!(a + b) \sim_{\mathcal{B}} !a \mid !b$

```
Theorem example_2 : forall A B,
  bisupto bisim_t (repl (plus (act (dn A) null) (act (dn B) null)))
                  (par (repl (act (dn A) null)) (repl (act (dn B) null))).
coinduction. intros. unfold. split.
  % ...
```

Need `sound bisim_t` to conclude that they are `bisim`!

Soundness of Up To Bisimilarity

```
Define sound : (proc → proc → proc → proc → prop) → prop by
  sound UT ≜ forall P Q, bisupto UT P Q → bisim P Q.

Theorem upto_bisim_sound : sound bisim_t.
```

Soundness of Up To Bisimilarity

```
Define sound : (proc → proc → proc → proc → prop) → prop by
  sound UT ≜ forall P Q, bisupto UT P Q → bisim P Q.

Theorem upto_bisim_sound : sound bisim_t.
```


Lemma 6.2.3 *The up-to-bisimilarity technique is sound; that is, if $\mathcal{R}$ is a bisimulation up to bisimilarity, then $\mathcal{R} \subseteq \sim$.*

Proof If $\mathcal{R}$ is a bisimulation up to bisimilarity then $\sim\mathcal{R}\sim$ (the composition of the three relations) is a bisimulation and $\sim\mathcal{R}\sim \subseteq \sim$. Since $\sim$ contains the identity relation, we also have $\mathcal{R} \subseteq \sim\mathcal{R}\sim$ which allows us to conclude. □

[Pous and Sangiorgi]

Soundness of Up To Bisimilarity

```
Define sound : (proc → proc → proc → proc → prop) → prop by
  sound UT ≜ forall P Q, bisupto UT P Q → bisim P Q.

Theorem upto_bisim_sound : sound bisim_t.
```


Lemma 6.2.3 *The up-to-bisimilarity technique is sound; that is, if $\mathcal{R}$ is a bisimulation up to bisimilarity, then $\mathcal{R} \subseteq \sim$.*

Proof If $\mathcal{R}$ is a bisimulation up to bisimilarity then $\sim\mathcal{R}\sim$ (the composition of the three relations) is a bisimulation and $\sim\mathcal{R}\sim \subseteq \sim$. Since $\sim$ contains the identity relation, we also have $\mathcal{R} \subseteq \sim\mathcal{R}\sim$ which allows us to conclude. □

```
Theorem bisim_t_right : forall P Q,
  (exists P1 Q1, bisim P P1 ∧ bisupto bisim_t P1 Q1
                 ∧ bisim Q1 Q) →
    bisim P Q.

Theorem bisim_t_left : forall P Q,
  bisupto bisim_t P Q →
    (exists P1 Q1, bisim P P1 ∧ bisupto bisim_t P1 Q1
                   ∧ bisim Q1 Q).
```

Soundness of Up To Bisimilarity: Proof

```
Theorem bisim_t_right : forall P Q,
  (exists P1 Q1, bisim P P1 ∧ bisupto bisim_t P1 Q1
                 ∧ bisim Q1 Q) →
    bisim P Q.
coinduction. intros. unfold.
   % case analysis

Theorem bisim_t_left : forall P Q,
  bisupto bisim_t P Q →
    (exists P1 Q1, bisim P P1 ∧ bisupto bisim_t P1 Q1
                   ∧ bisim Q1 Q).
% by reflexivity and symmetry of bisim
%   (each proved by a straightforward co-induction)
```

Soundness of Up To Bisimilarity: An Alternative

[Cimini]

$\mathcal{B}$ **is sound if and only if** $\sim_{\mathcal{B}}$ **is an equivalence relation.**

Up To Context

Up To Context

The up to context technique $\mathcal{K}$ is given by:

$$\mathcal{K}(P, P', Q, Q') \triangleq \exists C.\, P = C[P'] \wedge Q = C[Q']$$

Up To Context

The up to context technique $\mathcal{K}$ is given by:

$$\mathcal{K}(P, P', Q, Q') \triangleq \exists C.\, P = C[P'] \wedge Q = C[Q']$$

If $\mathcal{R}$ is a bisimulation up to context, then:

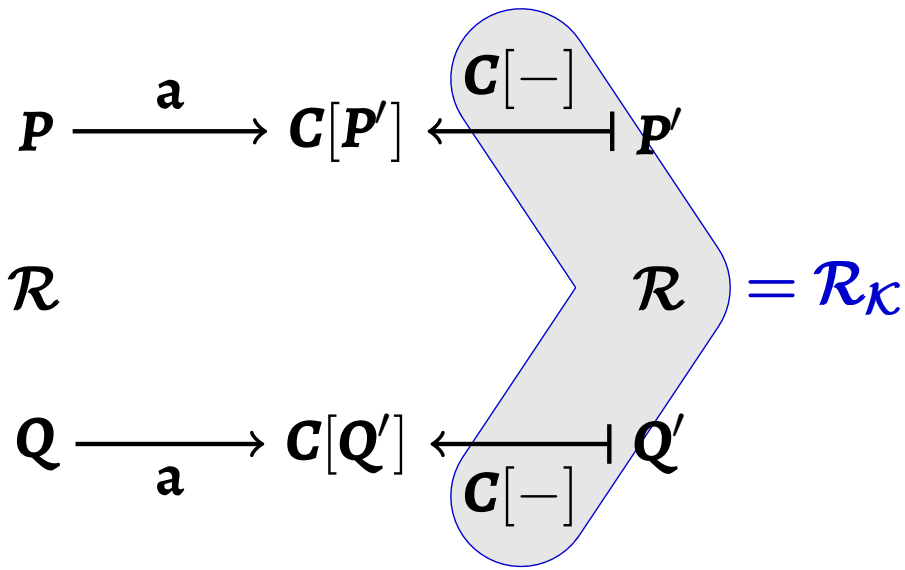
$$\begin{array}{ccccc} P & \xrightarrow{\mathbf{a}} & C[P'] & \xleftarrow{C[-]} & P' \\ \mathcal{R} & & & & \mathcal{R} \\ Q & \xrightarrow[\mathbf{a}]{} & C[Q'] & \xleftarrow[C[-]]{} & Q' \end{array}$$

Up To Context

The up to context technique $\mathcal{K}$ is given by:

$$\mathcal{K}(P, P', Q, Q') \triangleq \exists C.\, P = C[P'] \wedge Q = C[Q']$$

If $\mathcal{R}$ is a bisimulation up to context, then:



Up To Bisimilarity And Context

The up to bisimilarity and context technique $\mathcal{BK}$ is given by:

$$\mathcal{BK}(P, P', Q, Q') \triangleq \exists C.\, P \sim C[P'] \wedge Q \sim C[Q']$$

Up To Bisimilarity And Context

The up to bisimilarity and context technique $\mathcal{BK}$ is given by:

$$\mathcal{BK}(P, P', Q, Q') \triangleq \exists C.\, P \sim C[P'] \wedge Q \sim C[Q']$$

If $\mathcal{R}$ is a bisimulation up to bisimilarity and context, then:

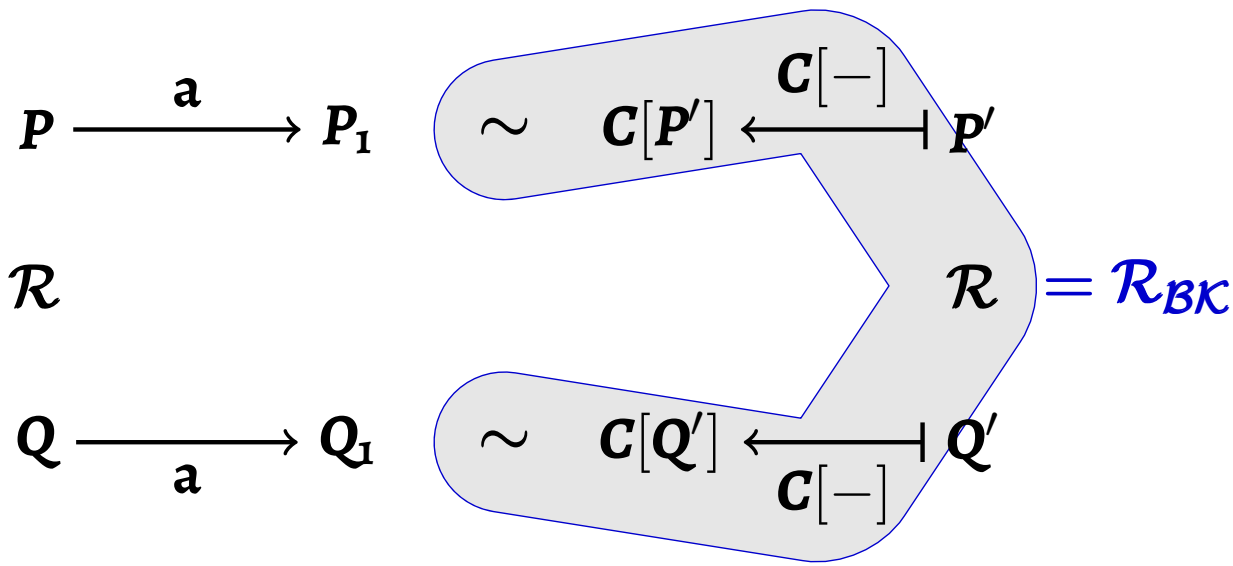
$$\begin{array}{ccccc} P & \xrightarrow{a} & P_1 & \sim & C[P'] \xleftarrow{C[-]} P' \\ \mathcal{R} & & & & \mathcal{R} \\ Q & \xrightarrow[a]{} & Q_1 & \sim & C[Q'] \xleftarrow[C[-]]{} Q' \end{array}$$

Up To Bisimilarity And Context

The **up to bisimilarity and context** technique $\mathcal{BK}$ is given by:

$$\mathcal{BK}(P, P', Q, Q') \triangleq \exists C.\, P \sim C[P'] \wedge Q \sim C[Q']$$

If $\mathcal{R}$ is a bisimulation up to bisimilarity and context, then:



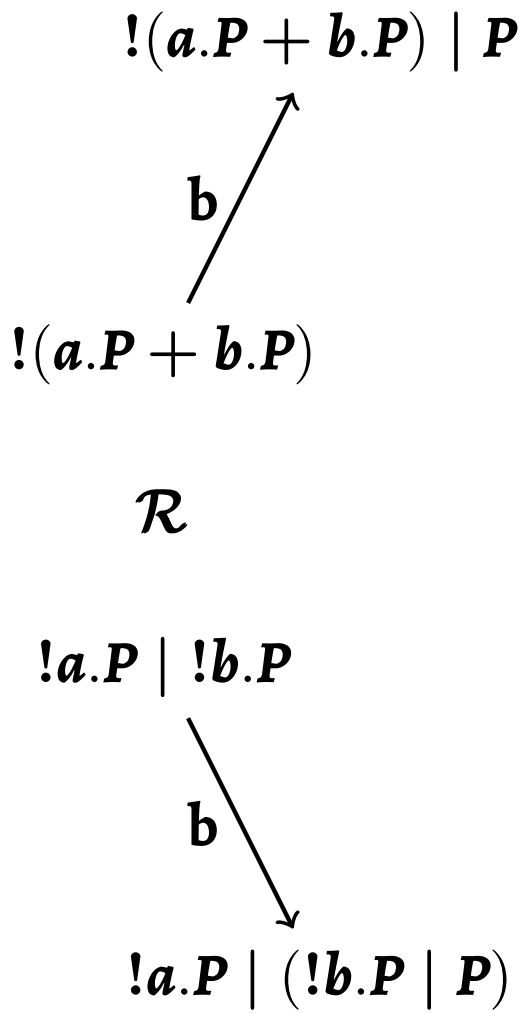
Example

$$!(a.P + b.P)$$

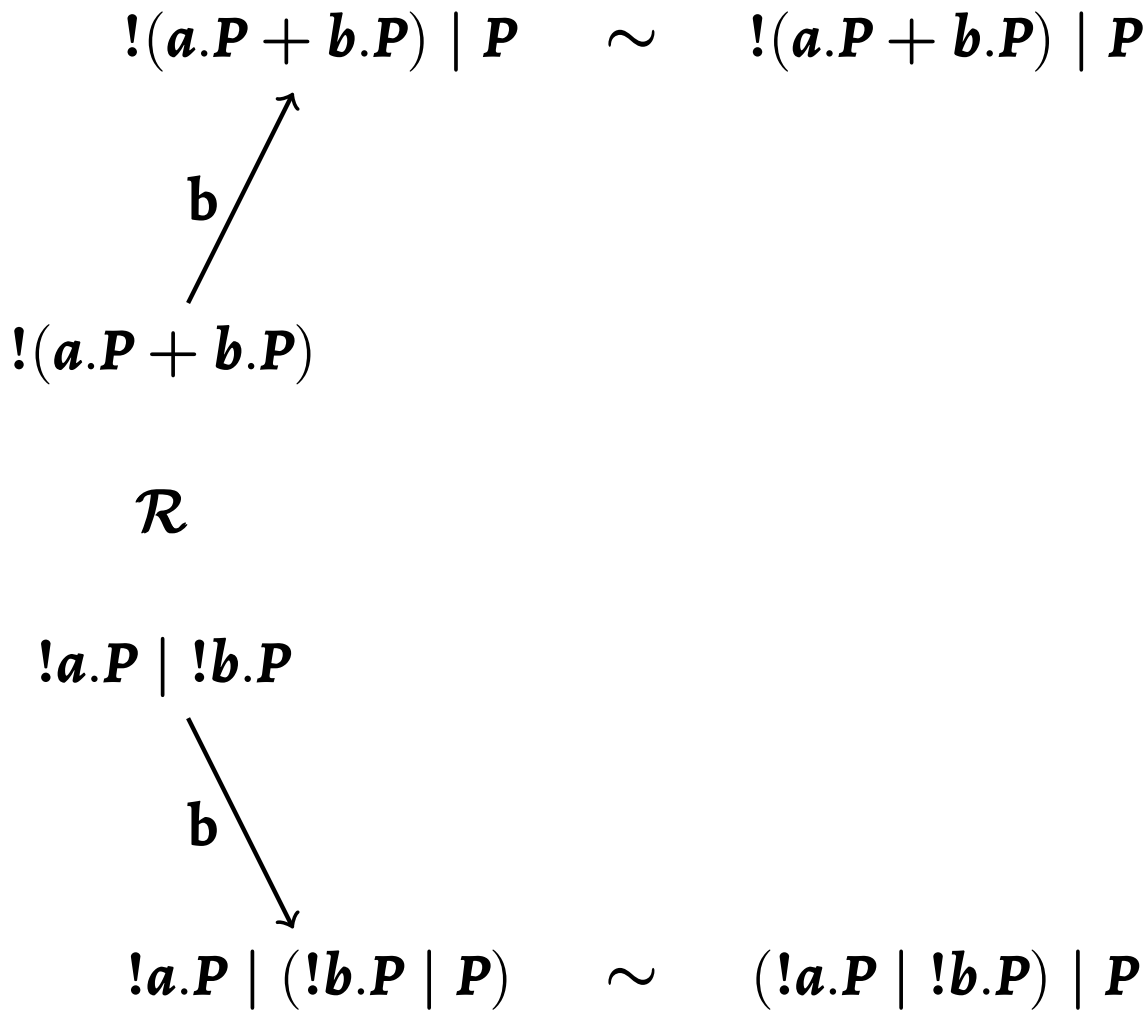
$$\mathcal{R}$$

$$!a.P \mid !b.P$$

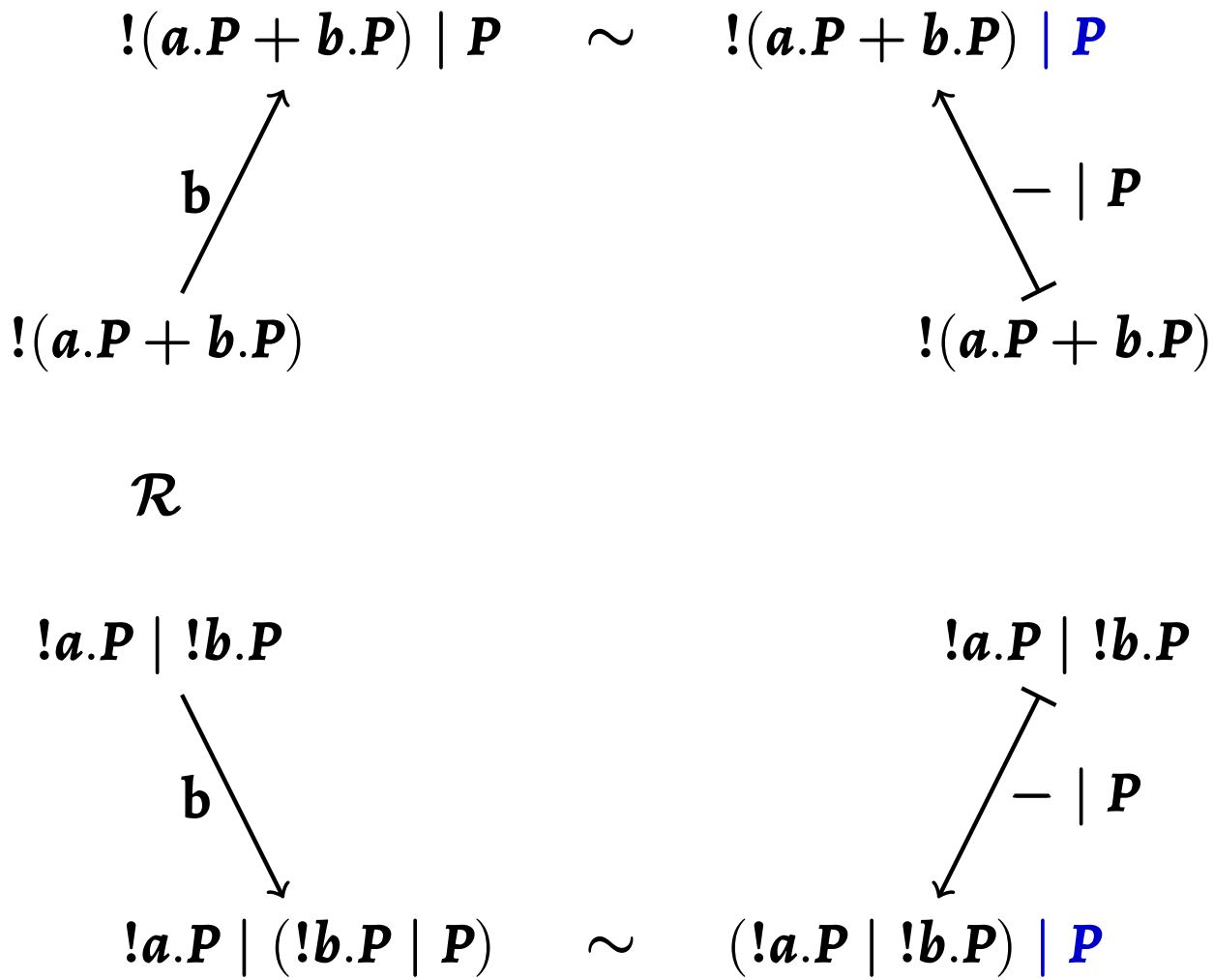
Example



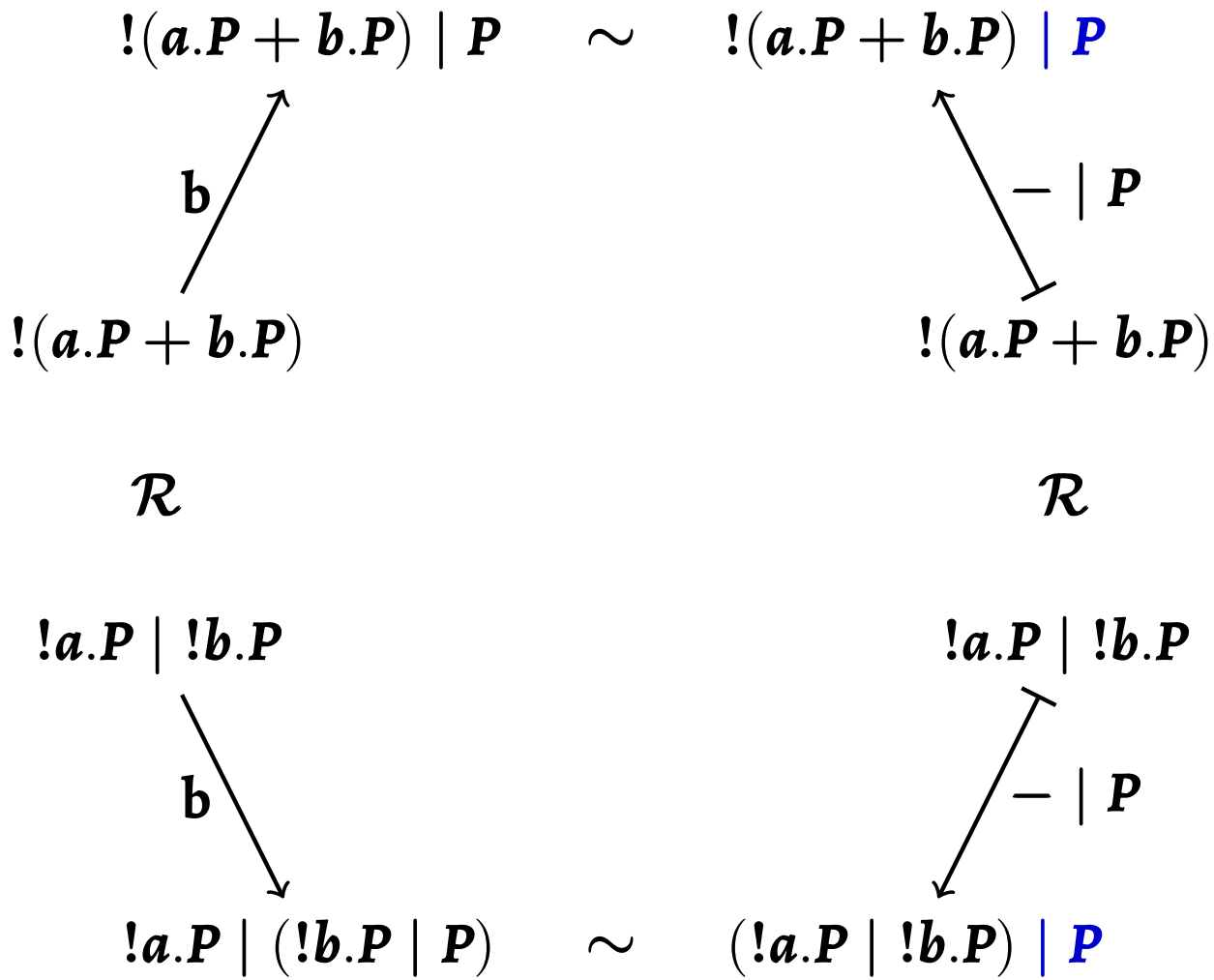
Example



Example



Example



$\mathcal{BK}$ in Abella

```
Define k_t : proc → proc → proc → proc → prop by
  k_t P1 P2 Q1 Q2 ≜ exists C,
    in_ctx C P1 P2 ∧ in_ctx C Q1 Q2.

Define bk_t : proc → proc → proc → proc → prop by
  bk_t P1 P2 Q1 Q2 ≜ exists P3 Q3,
    bisim_t P1 P3 Q1 Q3 ∧ k_t P3 P2 Q3 Q2.
```

Soundness of Up To $\mathcal{BK}$

[Cimini]

$\mathcal{BK}$ **is sound if** $\sim_{\mathcal{BK}}$ **is a congruence.**

Soundness of Up To $\mathcal{BK}$

[Cimini]

$\mathcal{BK}$ **is sound if** $\sim_{\mathcal{BK}}$ **is a congruence.**

Proof: by straightforward co-induction.

Congruence of $\sim_{\mathcal{BK}}$

[Cimini]

$\sim_{\mathcal{BK}}$ **is a congruence if contexts are faithful**

Congruence of $\sim_{\mathcal{BK}}$

[Cimini]

$\sim_{\mathcal{BK}}$ **is a congruence if contexts are faithful**

Proof: by straightforward *induction* on the structure of contexts.

Faithful Contexts

CCS contexts are faithful.

Faithful Contexts

CCS contexts are faithful.

Fairly well known result, but now formalized in Abella.

Faithful Contexts

CCS contexts are faithful.

Fairly well known result, but now formalized in Abella.

Laborious, but doable, proof.

Sketch of Soundness of $\mathcal{BK}$

CCS contexts are faithful

$\sim_{\mathcal{BK}}$ is a congruence

$\mathcal{BK}$ is sound

Ongoing Work

Several other up to techniques in the literature

Ongoing Work

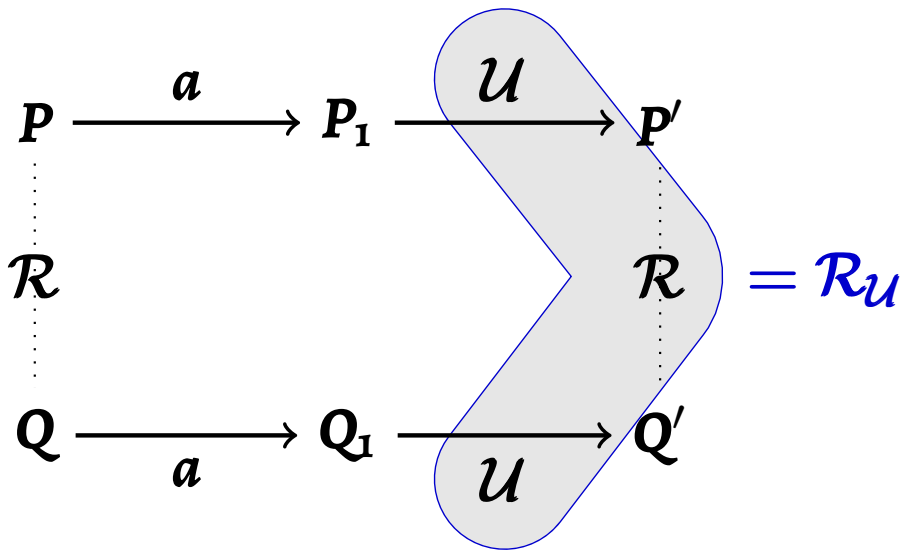
Several other up to techniques in the literature

Examples:

- **Transitivity:** $\mathcal{R} \rightarrowtail \mathcal{R}^*$
- **Union Bisimilarity:** $\mathcal{R} \rightarrowtail \mathcal{R} \cup \sim$

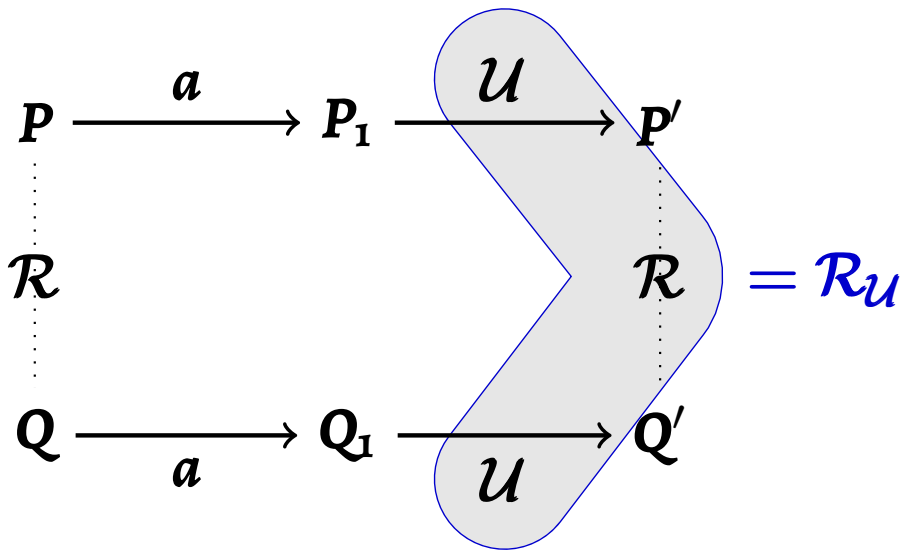
Ongoing Work: Up To Transitivity

Problem: Is there a $\mathcal{U}$ such that $\mathcal{R}_{\mathcal{U}} = \mathcal{R}^*$?



Ongoing Work: Up To Transitivity

Problem: Is there a $\mathcal{U}$ such that $\mathcal{R}_{\mathcal{U}} = \mathcal{R}^*$?



Almost! $\quad \mathcal{U}(P, P', Q, Q') \triangleq P \mathcal{R}^* P' \wedge Q \mathcal{R}^* Q'.$

Bisimulation Up To for the π-Calculus

Higher-Order Encoding

```
Type null          proc.
Type taup          proc → proc.
Type plus, par     proc → proc → proc.
Type match, out    name → name → proc → proc.
Type repl          proc → proc.

Type in            name → (name → proc) → proc.
Type nu            (name → proc) → proc.
```

Higher-Order Encoding

```
Type null          proc.
Type taup          proc → proc.
Type plus, par     proc → proc → proc.
Type match, out    name → name → proc → proc.
Type repl          proc → proc.

Type in            name → (name → proc) → proc.
Type nu            (name → proc) → proc.
```

$(\nu x)\ a(u).\ \overline{u}\langle x\rangle.\ 0$ $\qquad$ $\texttt{nu}\ (\lambda x.\ \texttt{in}\ a\ (\lambda u.\ \texttt{out}\ u\ x\ \texttt{null}))$

Digression: Representational Adequacy

Higher-Order Abstract Syntax; λ-Tree Syntax

π-calculus process $\cong$ term of type `proc`

Digression: Representational Adequacy

Higher-Order Abstract Syntax; λ-Tree Syntax

π-calculus process $\cong$ term of type `proc`

- The only term of type $A \to B$ is of the form $\lambda(x{:}A).\, M$
- Therefore, can access the "body" of a λ-term (by higher-order unification)
- Substitution is by application and normalization
- There are no "exotic terms", *i.e.*, terms of type `proc` that do not denote a π-calculus process

Bound Actions

```
Define
 one  : proc → action           → proc           → prop,
 oneb : proc → (name → action) → (name → proc) → prop
by
   ... % cases of tau, plus, par, repl, symmetry
; one (out X Y P) (up X Y) P
; one (nu λx. P x) A (nu λx. Q x) ≜
    nabla x, one  (P x) A (Q x)
; one (par P Q) tau (nu λy. par (M y) (N y)) ≜
    exists X, oneb P (dn X) M ∧ oneb Q (up X) N

; oneb (in X M) (dn X) M
; oneb (nu λx. P x) A (λy. nu λx. Q x y) ≜
    nabla x, oneb (P x) A (λy.Q x y)
; oneb (nu λx. M x) (up X) N ≜
    nabla y, one (M y) (up X y) (N y)
; ... % cases of plus, par, repl
```

Bisimilarity for the π-Calculus

```
CoDefine bisim : proc → proc → prop by
  bisim P Q ≜
    (∀A P1, one P A P1 → ∃Q1, one Q A Q1 ∧ bisim P1 Q1)
∧   (∀X M, oneb P (dn X) M → ∃N, oneb Q (dn X) N ∧
      (∀w, bisim (M w) (N w)) )
∧   (∀X M, oneb P (up X) M → ∃N, oneb Q (up X) N ∧
       nabla w, bisim (M w) (N w))
∧   (∀A Q1, one Q A Q1 → ∃P1, one P A P1 ∧ bisim Q1 P1)
∧   (∀X N, oneb Q (dn X) N → ∃M, oneb P (dn X) M ∧
      (∀w, bisim (N w) (M w)))
∧   (∀X N, oneb Q (up X) N → ∃M,  oneb P (up X) M ∧
       nabla w, bisim (N w) (M w)).
```

Bisimilarity Up To for the π-Calculus

```
CoDefine bisupto : (proc → proc → proc → proc → prop)
                 → (proc → proc → prop) by
  bisupto UT P Q ≜
   (∀A P1, one P A P1 → ∃Q1, one Q A Q1 ∧
      upto UT (bisupto UT) P1 Q1)
∧  (∀X M, oneb P (dn X) M → ∃N, oneb Q (dn X) N ∧
     (∀w, upto UT (bisupto UT) (M w) (N w)) )
∧  (∀X M, oneb P (up X) M → ∃N, oneb Q (up X) N ∧
     nabla w, upto UT (bisupto UT) (M w) (N w))
∧  (∀A Q1, one Q A Q1 → ∃P1, one P A P1 ∧
     upto UT (bisupto UT) Q1 P1)
∧  (∀X N, oneb Q (dn X) N → ∃M, oneb P (dn X) M ∧
     (∀w, upto UT (bisupto UT) (N w) (M w)))
∧  (∀X N, oneb Q (up X) N → ∃M,  oneb P (up X) M ∧
     nabla w, upto UT (bisupto UT) (N w) (M w)).
```

Modularity in Definitions

We can reuse the definitions already seen earlier.

Modularity in Definitions

We can reuse the definitions already seen earlier.

```
CoDefine
 bisupto_bisim : proc → proc → prop,
 bisupto_k     : proc → proc → prop,
 bisupto_bk    : proc → proc → prop
by
  bisupto_bisim P Q ≜ bisupto bisim_t P Q
; bisupto_k P Q     ≜ bisupto k_t P Q
; bisupto_bk P Q    ≜ bisupto bk_t P Q.la
```

Status of Meta-Proofs for the π-Calculus

- **Soundness of $\mathcal{B}$: DONE**
  - **Factoring: DONE**
  - **Characterization as equivalence of $\sim_{\mathcal{B}}$: DONE**
- **Soundness of $\mathcal{BK}$: in progress**
  - **Generalization of faithful contexts for the π-calculus: DONE**
  - **Faithful contexts implies Congruence of $\sim_{\mathcal{BK}}$: in progress**
  - **Congruence of $\sim_{\mathcal{BK}}$ implies soundness: DONE**

Perspectives

Related Work: Compatibility

[Sangiorgi, Pous, …]

- When are two up to techniques composable?
- Mathematics has been thoroughly worked out
- Soundness of a technique is a corollary of compatibility
- Formalized in Coq by Pous
  - Builds up from first principles:
    — lattices, functions, fixed points, *etc.*
  - Encodes the mathematics directly
  - Does not use `cofix`!

Related Work: Compatibility

[Sangiorgi, Pous, ...]

- When are two up to techniques composable?
- Mathematics has been thoroughly worked out
- Soundness of a technique is a corollary of compatibility
- Formalized in Coq by Pous
  - Builds up from first principles:
    — lattices, functions, fixed points, *etc.*
  - Encodes the mathematics directly
  - Does not use `cofix`!
- In Abella?

Related Work: Co-Induction Modulo Rewriting in Coq

- **Paco: A Coq Library for Parameterized Coinduction'**
  [Hur, Neis, Dreyer, Vafeiadis]

  > *The Paco library provides a tactic called "**pcofix**", replacing Coq's primitive **cofix** and avoiding its syntactic guardedness checking of proof terms. We have found that **pcofix** yields clear [...] usability benefits, even on simple examples.*

- **Our Abella formalization repeated in Paco/Coq?**
  - **CCS case is a great *stage* project!** **(talk to me or Dale)**

Related Work: Co-Induction Modulo Rewriting in Coq

- **Paco: A Coq Library for Parameterized Coinduction'**
  [Hur, Neis, Dreyer, Vafeiadis]

  > *The Paco library provides a tactic called "**pcofix**", replacing Coq's primitive **cofix** and avoiding its syntactic guardedness checking of proof terms. We have found that **pcofix** yields clear [...] usability benefits, even on simple examples.*

- **Our Abella formalization repeated in Paco/Coq?**
  - **CCS case is a great *stage* project!** **(talk to me or Dale)**
  - **Less clear for the π-calculus**

Summary

Bisimulation Up To is a great test case for logical frameworks.

Summary

Bisimulation Up To is a great test case for logical frameworks.

The standard definitions can be formalized in a straightforward manner using the (co-)inductive and higher-order features available in Abella.

Summary

Bisimulation Up To is a great test case for logical frameworks.

The standard definitions can be formalized in a straightforward manner using the (co-)inductive and higher-order features available in Abella.

Our formalization is designed to make interactive proofs of bisimulations up to easy, using standard tactics familiar from the formalization of the ordinary bisimulation.

Summary

Bisimulation Up To is a great test case for logical frameworks.

The standard definitions can be formalized in a straightforward manner using the (co-)inductive and higher-order features available in Abella.

Our formalization is designed to make interactive proofs of bisimulations up to easy, using standard tactics familiar from the formalization of the ordinary bisimulation.

Full soundness of $\mathcal{BK}$ for the π-calculus is in progress.

Summary

Bisimulation Up To is a great test case for logical frameworks.

The standard definitions can be formalized in a straightforward manner using the (co-)inductive and higher-order features available in Abella.

Our formalization is designed to make interactive proofs of bisimulations up to easy, using standard tactics familiar from the formalization of the ordinary bisimulation.

Full soundness of $\mathcal{BK}$ for the π-calculus is in progress.

(Higher-order syntax makes reasoning about binding structures and bound actions systematic and natural in Abella, while this is problematic for first-order syntax or rich function types.)