PREG Axiomatizer – A Ground Bisimilarity Checker for GSOS with Predicates

Luca Aceto[1], Georgiana Caltais[1], Eugen-Ioan Goriac[1], Anna Ingólfsdóttir[1]

[1]Reykjavik University ICE-TCS, Iceland

Purpose

Check for behavioral equivalences

- between processes specified using GSOS operators
- faster than by just applying the definition

Extend the expressiveness of the GSOS framework for giving semantics to operators

- with predicates

Purpose

Check for behavioral equivalences

- between processes specified using GSOS operators
- faster than by just applying the definition

Extend the expressiveness of the GSOS framework for giving semantics to operators

- with predicates

Pre . (lude + liminaries)

$\mathcal{A}$ – finite set of actions, $\mathcal{P}$ – finite set of predicates
$f \in \Sigma$ – an l-ary operation, defined by:

- *preg* transition rules ($\mathcal{R}^{\mathcal{A}}$):

$$\frac{\{x_i \xrightarrow{a_{ij}} y_{ij} \mid i \in I^+\} \qquad \{P_{ij}x_i \mid i \in J^+\} \\ \{x_i \stackrel{b}{\nrightarrow} \mid i \in I^-, b \in \mathcal{B}_i\} \quad \{\neg Q x_i \mid i \in J^-, Q \in \mathcal{Q}_i\}}{f(x_1, \ldots, x_l) \xrightarrow{c} C[\vec{x}, \vec{y}]}$$

- *preg* predicate rules ($\mathcal{R}^{\mathcal{P}}$):

$$\frac{\{x_i \xrightarrow{a_{ij}} y_{ij} \mid i \in I^+\} \qquad \{P_{ij}x_i \mid i \in J^+\} \\ \{x_i \stackrel{b}{\nrightarrow} \mid i \in I^-, b \in \mathcal{B}_i\} \quad \{\neg Q x_i \mid i \in J^-, Q \in \mathcal{Q}_i\}}{P(f(x_1, \ldots, x_l))}$$

preg system: $G = (\Sigma, \mathcal{R}^{\mathcal{A}} \cup \mathcal{R}^{\mathcal{P}})$

Pre . (lude + liminaries)

$\mathcal{A}$ – finite set of actions, $\mathcal{P}$ – finite set of predicates
$f \in \Sigma$ – an l-ary operation, defined by:

- *preg* transition rules ($\mathcal{R}^{\mathcal{A}}$):

$$\frac{\{x_i \xrightarrow{a_{ij}} y_{ij} \mid i \in I^+\} \qquad \{P_{ij}x_i \mid i \in J^+\} \quad \{x_i \stackrel{b}{\nrightarrow} \mid i \in I^-, b \in \mathcal{B}_i\} \quad \{\neg Q x_i \mid i \in J^-, Q \in \mathcal{Q}_i\}}{f(x_1, \ldots, x_l) \xrightarrow{c} C[\vec{x}, \vec{y}]}$$

- *preg* predicate rules ($\mathcal{R}^{\mathcal{P}}$):

$$\frac{\{x_i \xrightarrow{a_{ij}} y_{ij} \mid i \in I^+\} \qquad \{P_{ij}x_i \mid i \in J^+\} \quad \{x_i \stackrel{b}{\nrightarrow} \mid i \in I^-, b \in \mathcal{B}_i\} \quad \{\neg Q x_i \mid i \in J^-, Q \in \mathcal{Q}_i\}}{P(f(x_1, \ldots, x_l))}$$

preg system: $G = (\Sigma, \mathcal{R}^{\mathcal{A}} \cup \mathcal{R}^{\mathcal{P}})$

Pre . (lude + liminaries)

$\mathcal{A}$ – finite set of actions, $\mathcal{P}$ – finite set of predicates
$f \in \Sigma$ – an l-ary operation, defined by:

- *preg* transition rules ($\mathcal{R}^{\mathcal{A}}$):

$$\frac{\{x_i \xrightarrow{a_{ij}} y_{ij} \mid i \in I^+\} \qquad \{P_{ij}x_i \mid i \in J^+\} \qquad \{x_i \overset{b}{\nrightarrow} \mid i \in I^-, b \in \mathcal{B}_i\} \quad \{\neg Q x_i \mid i \in J^-, Q \in \mathcal{Q}_i\}}{f(x_1, \ldots, x_l) \xrightarrow{c} C[\vec{x}, \vec{y}]}$$

- *preg* predicate rules ($\mathcal{R}^{\mathcal{P}}$):

$$\frac{\{x_i \xrightarrow{a_{ij}} y_{ij} \mid i \in I^+\} \qquad \{P_{ij}x_i \mid i \in J^+\} \qquad \{x_i \overset{b}{\nrightarrow} \mid i \in I^-, b \in \mathcal{B}_i\} \quad \{\neg Q x_i \mid i \in J^-, Q \in \mathcal{Q}_i\}}{P(f(x_1, \ldots, x_l))}$$

preg system: $G = (\Sigma, \mathcal{R}^{\mathcal{A}} \cup \mathcal{R}^{\mathcal{P}})$

Pre . (lude + liminaries)

$\mathcal{A}$ – finite set of actions, $\mathcal{P}$ – finite set of predicates
$f \in \Sigma$ – an l-ary operation, defined by:

- *preg* transition rules ($\mathcal{R}^{\mathcal{A}}$):

$$\frac{\{x_i \xrightarrow{a_{ij}} y_{ij} \mid i \in I^+\} \qquad \{P_{ij}x_i \mid i \in J^+\} \qquad \{x_i \stackrel{b}{\nrightarrow} \mid i \in I^-, b \in \mathcal{B}_i\} \quad \{\neg Q x_i \mid i \in J^-, Q \in \mathcal{Q}_i\}}{f(x_1, \ldots, x_l) \xrightarrow{c} C[\vec{x}, \vec{y}]}$$

- *preg* predicate rules ($\mathcal{R}^{\mathcal{P}}$):

$$\frac{\{x_i \xrightarrow{a_{ij}} y_{ij} \mid i \in I^+\} \qquad \{P_{ij}x_i \mid i \in J^+\} \qquad \{x_i \stackrel{b}{\nrightarrow} \mid i \in I^-, b \in \mathcal{B}_i\} \quad \{\neg Q x_i \mid i \in J^-, Q \in \mathcal{Q}_i\}}{P(f(x_1, \ldots, x_l))}$$

preg system: $G = (\Sigma, \mathcal{R}^{\mathcal{A}} \cup \mathcal{R}^{\mathcal{P}})$

Pre . (lude + liminaries)

$\mathcal{A}$ – finite set of actions, $\mathcal{P}$ – finite set of predicates
$f \in \Sigma$ – an l-ary operation, defined by:

- *preg* transition rules ($\mathcal{R}^{\mathcal{A}}$):

$$\frac{\{x_i \xrightarrow{a_{ij}} y_{ij} \mid i \in I^+\} \qquad \{P_{ij}x_i \mid i \in J^+\} \qquad \{x_i \stackrel{b}{\nrightarrow} \mid i \in I^-, b \in \mathcal{B}_i\} \quad \{\neg Q x_i \mid i \in J^-, Q \in \mathcal{Q}_i\}}{f(x_1, \ldots, x_l) \xrightarrow{c} C[\vec{x}, \vec{y}]}$$

- *preg* predicate rules ($\mathcal{R}^{\mathcal{P}}$):

$$\frac{\{x_i \xrightarrow{a_{ij}} y_{ij} \mid i \in I^+\} \qquad \{P_{ij}x_i \mid i \in J^+\} \qquad \{x_i \stackrel{b}{\nrightarrow} \mid i \in I^-, b \in \mathcal{B}_i\} \quad \{\neg Q x_i \mid i \in J^-, Q \in \mathcal{Q}_i\}}{P(f(x_1, \ldots, x_l))}$$

preg system: $G = (\Sigma, \mathcal{R}^{\mathcal{A}} \cup \mathcal{R}^{\mathcal{P}})$

Finite trees Parallel composition $\_ \| \_$ Immediate termination $\downarrow$

Syntax: $t ::= \delta \mid \kappa_\downarrow \mid a.t \; (\forall a \in \mathcal{A}) \mid t + t \mid t \parallel t$

Semantics:
$$\left(\begin{array}{ccc}
\dfrac{}{a.x \xrightarrow{a} x} & \dfrac{x \xrightarrow{a} x'}{x + y \xrightarrow{a} x'} & \dfrac{y \xrightarrow{a} y'}{x + y \xrightarrow{a} y'} \\
\dfrac{}{\kappa_\downarrow \downarrow} & \dfrac{x \downarrow}{(x + y) \downarrow} & \dfrac{y \downarrow}{(x + y) \downarrow} \\
\dfrac{x \xrightarrow{a} x'}{x \parallel y \xrightarrow{a} x' \parallel y} & \dfrac{y \xrightarrow{a} y'}{x \parallel y \xrightarrow{a} x \parallel y'} & \dfrac{x \downarrow \quad y \downarrow}{(x \parallel y) \downarrow}
\end{array} \right.$$

Finite trees Parallel composition $\_\|\_$ Immediate termination $\downarrow$

Syntax: $t ::= \delta \mid \kappa_{\downarrow} \mid a.t\ (\forall a \in \mathcal{A}) \mid t + t \mid t \parallel t$

Semantics:

$$\left\{\begin{array}{ccc}
\dfrac{}{a.x \xrightarrow{a} x} & \dfrac{x \xrightarrow{a} x'}{x + y \xrightarrow{a} x'} & \dfrac{y \xrightarrow{a} y'}{x + y \xrightarrow{a} y'} \\[2ex]
\dfrac{}{\kappa_{\downarrow}\downarrow} & \dfrac{x\downarrow}{(x+y)\downarrow} & \dfrac{y\downarrow}{(x+y)\downarrow} \\[2ex]
\dfrac{x \xrightarrow{a} x'}{x \parallel y \xrightarrow{a} x' \parallel y} & \dfrac{y \xrightarrow{a} y'}{x \parallel y \xrightarrow{a} x \parallel y'} & \dfrac{x\downarrow \quad y\downarrow}{(x \parallel y)\downarrow}
\end{array}\right.$$

Finite trees Parallel composition $\_\|\_$ Immediate termination $\downarrow$

Syntax: $t ::= \delta \mid \kappa_{\downarrow} \mid a.t \ (\forall a \in \mathcal{A}) \mid t + t \mid t \parallel t$

Semantics:
$$\left\{ \begin{array}{ccc}
\dfrac{}{a.x \xrightarrow{a} x} & \dfrac{x \xrightarrow{a} x'}{x + y \xrightarrow{a} x'} & \dfrac{y \xrightarrow{a} y'}{x + y \xrightarrow{a} y'} \\[2ex]
\dfrac{}{\kappa_{\downarrow} \downarrow} & \dfrac{x \downarrow}{(x + y) \downarrow} & \dfrac{y \downarrow}{(x + y) \downarrow} \\[2ex]
\dfrac{x \xrightarrow{a} x'}{x \parallel y \xrightarrow{a} x' \parallel y} & \dfrac{y \xrightarrow{a} y'}{x \parallel y \xrightarrow{a} x \parallel y'} & \dfrac{x \downarrow \quad y \downarrow}{(x \parallel y) \downarrow}
\end{array} \right.$$

Finite trees   Parallel composition $\_ \| \_$ Immediate termination $\downarrow$

Syntax: $t ::= \delta \mid \kappa_{\downarrow} \mid a.t \ (\forall a \in \mathcal{A}) \mid t + t \mid t \parallel t$

Semantics:

$$\left(\begin{array}{ccc}
\dfrac{}{a.x \xrightarrow{a} x} & \dfrac{x \xrightarrow{a} x'}{x + y \xrightarrow{a} x'} & \dfrac{y \xrightarrow{a} y'}{x + y \xrightarrow{a} y'} \\[2ex]
\dfrac{}{\kappa_{\downarrow} \downarrow} & \dfrac{x \downarrow}{(x + y) \downarrow} & \dfrac{y \downarrow}{(x + y) \downarrow} \\[2ex]
\dfrac{x \xrightarrow{a} x'}{x \parallel y \xrightarrow{a} x' \parallel y} & \dfrac{y \xrightarrow{a} y'}{x \parallel y \xrightarrow{a} x \parallel y'} & \dfrac{x \downarrow \quad y \downarrow}{(x \parallel y) \downarrow}
\end{array}\right.$$

Finite trees | Parallel composition _||_ | Immediate termination ↓

Syntax: $t ::= \delta \mid \kappa_{\downarrow} \mid a.t \ (\forall a \in \mathcal{A}) \mid t + t \mid t \parallel t$

Semantics:

$$\left\{ \begin{array}{ccc} \dfrac{}{a.x \xrightarrow{a} x} & \dfrac{x \xrightarrow{a} x'}{x + y \xrightarrow{a} x'} & \dfrac{y \xrightarrow{a} y'}{x + y \xrightarrow{a} y'} \\ \dfrac{}{\kappa_{\downarrow} \downarrow} & \dfrac{x \downarrow}{(x + y) \downarrow} & \dfrac{y \downarrow}{(x + y) \downarrow} \\ \dfrac{x \xrightarrow{a} x'}{x \parallel y \xrightarrow{a} x' \parallel y} & \dfrac{y \xrightarrow{a} y'}{x \parallel y \xrightarrow{a} x \parallel y'} & \dfrac{x \downarrow \quad y \downarrow}{(x \parallel y) \downarrow} \end{array} \right.$$

Finite trees   Parallel composition _||_   Immediate termination ↓

Syntax: $t ::= \delta \mid \kappa_{\downarrow} \mid a.t \; (\forall a \in \mathcal{A}) \mid t + t \mid t \parallel t$

Semantics:

$$\left(\begin{array}{ccc} \dfrac{}{a.x \xrightarrow{a} x} & \dfrac{x \xrightarrow{a} x'}{x + y \xrightarrow{a} x'} & \dfrac{y \xrightarrow{a} y'}{x + y \xrightarrow{a} y'} \\[2ex] \dfrac{}{\kappa_{\downarrow}\downarrow} & \dfrac{x\downarrow}{(x+y)\downarrow} & \dfrac{y\downarrow}{(x+y)\downarrow} \\[2ex] \dfrac{x \xrightarrow{a} x'}{x \parallel y \xrightarrow{a} x' \parallel y} & \dfrac{y \xrightarrow{a} y'}{x \parallel y \xrightarrow{a} x \parallel y'} & \dfrac{x\downarrow \quad y\downarrow}{(x \parallel y)\downarrow} \end{array}\right.$$

Finite trees Parallel composition $\_\|\_$ Immediate termination $\downarrow$

Syntax: $t ::= \delta \mid \kappa_{\downarrow} \mid a.t \ (\forall a \in \mathcal{A}) \mid t + t \mid t \parallel t$

Semantics:
$$\left(\begin{array}{c}
\dfrac{}{a.x \xrightarrow{a} x} \quad \dfrac{x \xrightarrow{a} x'}{x + y \xrightarrow{a} x'} \quad \dfrac{y \xrightarrow{a} y'}{x + y \xrightarrow{a} y'} \\[2ex]
\dfrac{}{\kappa_{\downarrow} \downarrow} \quad \dfrac{x \downarrow}{(x + y) \downarrow} \quad \dfrac{y \downarrow}{(x + y) \downarrow} \\[2ex]
\dfrac{x \xrightarrow{a} x'}{x \parallel y \xrightarrow{a} x' \parallel y} \quad \dfrac{y \xrightarrow{a} y'}{x \parallel y \xrightarrow{a} x \parallel y'} \quad \dfrac{x \downarrow \quad y \downarrow}{(x \parallel y) \downarrow}
\end{array}\right.$$

Question

Is

$$s = a.\kappa_\downarrow \parallel a.\kappa_\downarrow \parallel b.\kappa_\downarrow \parallel b.\kappa_\downarrow$$

strongly bisimilar to

$$\begin{aligned} \mathrm{t} = a.(a.b.b.\kappa_\downarrow + b.(a.b.\kappa_\downarrow + b.a.\kappa_\downarrow)) + \\ b.(a.(a.b.\kappa_\downarrow + b.a.\kappa_\downarrow) + b.a.a.\kappa_\downarrow) \end{aligned}$$

?

Answer by using $\begin{cases} \text{1) the definition of strong bisimilarity} \\ \text{2) an axiomatization modulo bisimilarity} \end{cases}$

Question

Is

$$s = a.\kappa_{\downarrow} \parallel a.\kappa_{\downarrow} \parallel b.\kappa_{\downarrow} \parallel b.\kappa_{\downarrow}$$

strongly bisimilar to

$$\mathrm{t} = a.(a.b.b.\kappa_{\downarrow} + b.(a.b.\kappa_{\downarrow} + b.a.\kappa_{\downarrow})) + b.(a.(a.b.\kappa_{\downarrow} + b.a.\kappa_{\downarrow}) + b.a.a.\kappa_{\downarrow})$$

?

$$\text{Answer by using} \begin{cases} \text{1) the definition of strong bisimilarity} \\ \text{2) an axiomatization modulo bisimilarity} \end{cases}$$

1) By the definition of strong bisimilarity

Definition (Bisimilarity "$\leftrightarrow$")

A symmetric relation R is a bisimulation iff:

- if $s\ R\ t$, $a \in \mathcal{A}$ and $s \xrightarrow{a} s'$ then $t \xrightarrow{a} t'$ and $s'\ R\ t'$;
- if $s\ R\ t$, and $s\downarrow$ then $t\downarrow$.

Terms s and t are bisimilar ($s \leftrightarrow t$) iff $s\ R\ t$ and R is a bisimulation.

Assume $\_\|\_$ is associative, commutative, with $\kappa_\downarrow$ as the identity.

Does $a.\kappa_\downarrow \parallel a.\kappa_\downarrow \parallel b.\kappa_\downarrow \parallel b.\kappa_\downarrow \xrightarrow{a} s'$ hold ?

Instantiate $\dfrac{x \xrightarrow{a} x'}{x \parallel y \xrightarrow{a} x' \parallel y}$ as $\dfrac{a.\kappa_\downarrow \parallel a.\kappa_\downarrow \parallel b.\kappa_\downarrow \xrightarrow{a} s^{ii}}{(a.\kappa_\downarrow \parallel a.\kappa_\downarrow \parallel b.\kappa_\downarrow) \parallel b.\kappa_\downarrow \xrightarrow{a} s^{ii} \parallel b.\kappa_\downarrow}$.

1) By the definition of strong bisimilarity

Definition (Bisimilarity "$\leftrightarrow$")

A symmetric relation R is a bisimulation iff:

- if $s\ R\ t$, $a \in \mathcal{A}$ and $s \xrightarrow{a} s'$ then $t \xrightarrow{a} t'$ and $s'\ R\ t'$;
- if $s\ R\ t$, and $s\downarrow$ then $t\downarrow$.

Terms s and t are bisimilar ($s \leftrightarrow t$) iff $s\ R\ t$ and R is a bisimulation.

Assume $\_\|\_$ is associative, commutative, with $\kappa_\downarrow$ as the identity.

Does $a.\kappa_\downarrow \parallel a.\kappa_\downarrow \parallel b.\kappa_\downarrow \parallel b.\kappa_\downarrow \xrightarrow{a} s'$ hold ?

Instantiate $\dfrac{x \xrightarrow{a} x'}{x \parallel y \xrightarrow{a} x' \parallel y}$ as $\dfrac{a.\kappa_\downarrow \parallel a.\kappa_\downarrow \parallel b.\kappa_\downarrow \xrightarrow{a} s^{ii}\ \ ?}{(a.\kappa_\downarrow \parallel a.\kappa_\downarrow \parallel b.\kappa_\downarrow) \parallel b.\kappa_\downarrow \xrightarrow{a} s^{ii} \parallel b.\kappa_\downarrow}$.

1) By the definition of strong bisimilarity

Definition (Bisimilarity "$\leftrightarroweq$")

A symmetric relation R is a bisimulation iff:

- if $s\ R\ t$, $a \in \mathcal{A}$ and $s \xrightarrow{a} s'$ then $t \xrightarrow{a} t'$ and $s'\ R\ t'$;
- if $s\ R\ t$, and $s \downarrow$ then $t \downarrow$.

Terms s and t are bisimilar ($s \leftrightarroweq t$) iff $s\ R\ t$ and R is a bisimulation.

Assume $\_\|\_$ is associative, commutative, with $\kappa_\downarrow$ as the identity.

So, does $a.\kappa_\downarrow \parallel a.\kappa_\downarrow \parallel b.\kappa_\downarrow \xrightarrow{a} s^{ii}$ hold ?

Instantiate $\dfrac{x \xrightarrow{a} x'}{x \parallel y \xrightarrow{a} x' \parallel y}$ as $\dfrac{a.\kappa_\downarrow \parallel a.\kappa_\downarrow \xrightarrow{a} s^{iii}}{(a.\kappa_\downarrow \parallel a.\kappa_\downarrow) \parallel b.\kappa_\downarrow \xrightarrow{a} s^{iii} \parallel b.\kappa_\downarrow}$.

1) By the definition of strong bisimilarity

Definition (Bisimilarity "$\leftrightarrow$")

A symmetric relation R is a bisimulation iff:

- if $s \; R \; t$, $a \in \mathcal{A}$ and $s \xrightarrow{a} s'$ then $t \xrightarrow{a} t'$ and $s' \; R \; t'$;
- if $s \; R \; t$, and $s \downarrow$ then $t \downarrow$.

Terms s and t are bisimilar ($s \leftrightarrow t$) iff $s \; R \; t$ and R is a bisimulation.

Assume $\_\|\_$ is associative, commutative, with $\kappa_\downarrow$ as the identity.

So, does $a.\kappa_\downarrow \| a.\kappa_\downarrow \| b.\kappa_\downarrow \xrightarrow{a} s^{ii}$ hold ?

Instantiate $\dfrac{x \xrightarrow{a} x'}{x \| y \xrightarrow{a} x' \| y}$ as $\dfrac{a.\kappa_\downarrow \| a.\kappa_\downarrow \xrightarrow{a} s^{iii} \quad ?}{(a.\kappa_\downarrow \| a.\kappa_\downarrow) \| b.\kappa_\downarrow \xrightarrow{a} s^{iii} \| b.\kappa_\downarrow}$.

1) By the definition of strong bisimilarity

Definition (Bisimilarity "$\leftrightarrow$")

A symmetric relation R is a bisimulation iff:

- if $s \; R \; t$, $a \in \mathcal{A}$ and $s \xrightarrow{a} s'$ then $t \xrightarrow{a} t'$ and $s' \; R \; t'$;
- if $s \; R \; t$, and $s \downarrow$ then $t \downarrow$.

Terms s and t are bisimilar ($s \leftrightarrow t$) iff $s \; R \; t$ and R is a bisimulation.

Assume $\_\|\_$ is associative, commutative, with $\kappa_\downarrow$ as the identity.

So, does $a.\kappa_\downarrow \parallel a.\kappa_\downarrow \xrightarrow{a} s^{iii}$ hold ?

Instantiate $\dfrac{x \xrightarrow{a} x'}{x \parallel y \xrightarrow{a} x' \parallel y}$ as $\dfrac{a.\kappa_\downarrow \xrightarrow{a} s^{iv}}{a.\kappa_\downarrow \parallel a.\kappa_\downarrow \xrightarrow{a} s^{iv} \parallel a.\kappa_\downarrow}$.

1) By the definition of strong bisimilarity

Definition (Bisimilarity "$\leftrightarrow$")

A symmetric relation R is a bisimulation iff:

- if $s\ R\ t$, $a \in \mathcal{A}$ and $s \xrightarrow{a} s'$ then $t \xrightarrow{a} t'$ and $s'\ R\ t'$;
- if $s\ R\ t$, and $s\downarrow$ then $t\downarrow$.

Terms s and t are bisimilar ($s \leftrightarrow t$) iff $s\ R\ t$ and R is a bisimulation.

Assume $\_\|\_$ is associative, commutative, with $\kappa_\downarrow$ as the identity.

So, does $a.\kappa_\downarrow \parallel a.\kappa_\downarrow \xrightarrow{a} s^{iii}$ hold ?

Instantiate $\dfrac{x \xrightarrow{a} x'}{x \parallel y \xrightarrow{a} x' \parallel y}$ as $\dfrac{a.\kappa_\downarrow \xrightarrow{a} s^{iv} \quad ?}{a.\kappa_\downarrow \parallel a.\kappa_\downarrow \xrightarrow{a} s^{iv} \parallel a.\kappa_\downarrow}$.

1) By the definition of strong bisimilarity

Definition (Bisimilarity "$\leftrightarrow$")

A symmetric relation R is a bisimulation iff:

- if $s\ R\ t$, $a \in \mathcal{A}$ and $s \xrightarrow{a} s'$ then $t \xrightarrow{a} t'$ and $s'\ R\ t'$;
- if $s\ R\ t$, and $s\downarrow$ then $t\downarrow$.

Terms s and t are bisimilar ($s \leftrightarrow t$) iff $s\ R\ t$ and R is a bisimulation.

Assume $\_\|\_$ is associative, commutative, with $\kappa_\downarrow$ as the identity.

So, does $a.\kappa_\downarrow \xrightarrow{a} s^{iv}$ hold ?

Instantiate $\dfrac{}{a.x \xrightarrow{a} x}$ as $\dfrac{}{a.\kappa_\downarrow \xrightarrow{a} \kappa_\downarrow}$.

1) By the definition of strong bisimilarity

Definition (Bisimilarity "$\leftrightarrow$")

A symmetric relation R is a bisimulation iff:

- if $s\ R\ t$, $a \in \mathcal{A}$ and $s \xrightarrow{a} s'$ then $t \xrightarrow{a} t'$ and $s'\ R\ t'$;
- if $s\ R\ t$, and $s \downarrow$ then $t \downarrow$.

Terms s and t are bisimilar ($s \leftrightarrow t$) iff $s\ R\ t$ and R is a bisimulation.

Assume $\_\|\_$ is associative, commutative, with $\kappa_\downarrow$ as the identity.

So, does $a.\kappa_\downarrow \xrightarrow{a} s^{iv}$ hold ?

Instantiate $\dfrac{}{a.x \xrightarrow{a} x}$ as $\dfrac{}{a.\kappa_\downarrow \xrightarrow{a} \kappa_\downarrow}$. ✓

1) By the definition of strong bisimilarity

Definition (Bisimilarity "$\leftrightarrow$")

A symmetric relation R is a bisimulation iff:

- if $s\ R\ t$, $a \in \mathcal{A}$ and $s \xrightarrow{a} s'$ then $t \xrightarrow{a} t'$ and $s'\ R\ t'$;
- if $s\ R\ t$, and $s \downarrow$ then $t \downarrow$.

Terms s and t are bisimilar ($s \leftrightarrow t$) iff $s\ R\ t$ and R is a bisimulation.

Assume $\_\|\_$ is associative, commutative, with $\kappa_\downarrow$ as the identity.

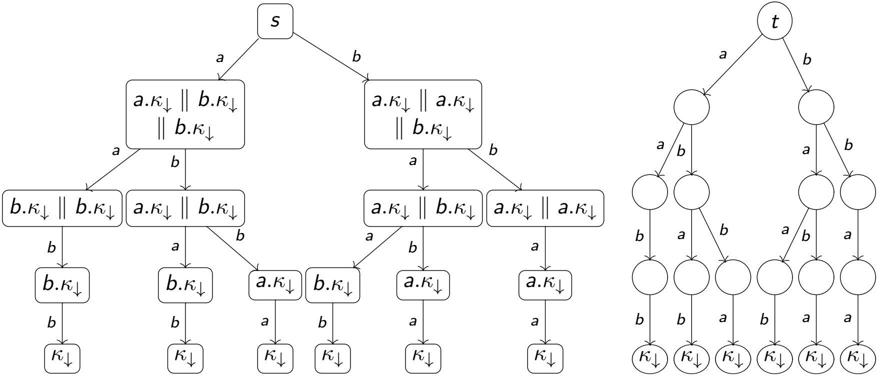
Therefore, at the end of the day, it holds that:

$$a.\kappa_\downarrow \parallel a.\kappa_\downarrow \parallel b.\kappa_\downarrow \parallel b.\kappa_\downarrow \xrightarrow{a} s' = a.\kappa_\downarrow \parallel b.\kappa_\downarrow \parallel b.\kappa_\downarrow$$

1) By the definition of strong bisimilarity

$s = a.\kappa_{\downarrow} \parallel a.\kappa_{\downarrow} \parallel b.\kappa_{\downarrow} \parallel b.\kappa_{\downarrow}$

$$t = a.(a.b.b.\kappa_{\downarrow} + b.(a.b.\kappa_{\downarrow} + b.a.\kappa_{\downarrow})) + b.(a.(a.b.\kappa_{\downarrow} + b.a.\kappa_{\downarrow}) + b.a.a.\kappa_{\downarrow})$$



$s \leftrightarrow t$

1) By the definition of strong bisimilarity

Demo

2) By an axiomatization modulo bisimilarity

$$
\begin{aligned}
x + x &= x \\
x + y &= y + x \\
(x + y) + z &= x + (y + z) \\
x + \delta &= x \\
x \parallel y &= x \parallel^1 y + x \parallel^2 y + x \parallel^3 y \\
x \parallel^1 (y + z) &= x \parallel^1 y + x \parallel^1 z \\
(x + y) \parallel^1 z &= x \parallel^1 z + y \parallel^1 z \\
(x + y) \parallel^2 z &= x \parallel^2 z + y \parallel^2 z \\
x \parallel^3 (y + z) &= x \parallel^3 y + x \parallel^3 z \\
k_\downarrow \parallel^1 k_\downarrow &= k_\downarrow \\
a.x' \parallel^2 y &= a.(x' \parallel^2 y) \\
x \parallel^3 a.y' &= a.(x \parallel^3 y') \\
x \parallel^{1/2/3} y &= \delta, \text{ otherwise}
\end{aligned}
$$

Using this axiomatization seems to be less intuitive, however, it is

- much faster, and
- derived for free.

2) By an axiomatization modulo bisimilarity

$$
\begin{aligned}
x + x &= x \\
x + y &= y + x \\
(x + y) + z &= x + (y + z) \\
x + \delta &= x \\
x \parallel y &= x \parallel^1 y + x \parallel^2 y + x \parallel^3 y \\
x \parallel^1 (y + z) &= x \parallel^1 y + x \parallel^1 z \\
(x + y) \parallel^1 z &= x \parallel^1 z + y \parallel^1 z \\
(x + y) \parallel^2 z &= x \parallel^2 z + y \parallel^2 z \\
x \parallel^3 (y + z) &= x \parallel^3 y + x \parallel^3 z \\
k_\downarrow \parallel^1 k_\downarrow &= k_\downarrow \\
a.x' \parallel^2 y &= a.(x' \parallel^2 y) \\
x \parallel^3 a.y' &= a.(x \parallel^3 y') \\
x \parallel^{1/2/3} y &= \delta, \text{ otherwise}
\end{aligned}
$$

Using this axiomatization seems to be less intuitive, however, it is

- much faster, and
- derived for free.

2) By an axiomatization modulo bisimilarity

Demo

PREG Axiomatizer

- the first public tool for automatically deriving sound and ground-complete axiomatizations modulo bisimilarity for GSOS-like languages (to our knowledge)
- downloadable from `http://goriac.info/tools/preg-axiomatizer/`
- implemented using
  - Maude for the theory (∼2000 lines)
  - Python for the graphic user interface (∼300 lines)

$\_;\_$ and $\mathtt{while\_do\_}$

$$\frac{x \xrightarrow{a} x'}{x;y \xrightarrow{a} x';y}$$:

```
X -(a)-> X'
===
X ; Y -(a)-> (X' ; Y)
```

$$\frac{x\downarrow \quad y \xrightarrow{a} y'}{x;y \xrightarrow{a} y'}$$:

```
P(X) , Y -(a)-> Y'
===
X ; Y -(a)-> Y'
```

$$\frac{x\downarrow \quad y\downarrow}{(x;y)\downarrow}$$:

```
P(X) , P(Y)
===
P(X ; Y)
```

$$\frac{x\downarrow}{(\mathtt{while}\ x\ \mathtt{do}\ y)\downarrow}$$:

```
P(X)
===
P(while X do Y)
```

$$\frac{x \xrightarrow{a} x'}{\mathtt{while}\ x\ \mathtt{do}\ y \xrightarrow{a} y;\ \mathtt{while}\ x'\ \mathtt{do}\ y}$$:

```
X -(a)-> X'
===
(while X do Y) -(a)-> (Y ;(while X' do Y))
```

The following holds:

$$a.(a.a.\kappa_\downarrow;\ b.(a.a.\kappa_\downarrow;b.a.a.\kappa_\downarrow)) \Leftrightarrow \mathtt{while}\ a.b.b.\kappa_\downarrow\ \mathtt{do}\ a.a.\kappa_\downarrow\ .$$

_;_ and while_do_

Demo

_ || _

$$\frac{x\downarrow \quad y\downarrow}{(x \parallel y)\downarrow} \quad : \quad$$

```
P(X) , P(Y)
===
P(X || Y)
```

$$\frac{x \xrightarrow{act} x'}{x \parallel y \xrightarrow{act} x' \parallel y} \quad :$$

```
X -(act)-> X'
===
X || Y -(act)-> X' || Y
```

$$\frac{y \xrightarrow{act} y'}{x \parallel y \xrightarrow{act} x \parallel y'} \quad :$$

```
Y -(act)-> Y'
===
X || Y -(act)-> X' || Y
```

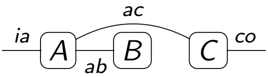
$$\frac{x \xrightarrow{p!d} x' \quad y \xrightarrow{p?d} y'}{x \parallel y \xrightarrow{p\#d} x' \parallel y'} \quad :$$

```
X -(p!d)-> X' , Y -(p?d)-> Y'
===
X || Y -(p#d)-> X' || Y'
```

$$\frac{x \xrightarrow{p?d} x' \quad y \xrightarrow{p!d} y'}{x \parallel y \xrightarrow{p\#d} x' \parallel y'} \quad :$$

```
X -(p?d)-> X' , Y -(p!d)-> Y'
===
X || Y -(p#d)-> X' || Y'
```

$\_ \parallel \_$

Consider the process network 

, where

- A, B, C are the communicating processes,
- ia, ab, ac, co are the ports, and
- the actions of sending and receiving the datum d over the port p are denoted by, respectively, $p!d$ and $p?d$.

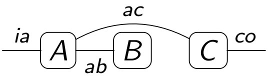
The whole protocol is specified as the term

$$T = ia?d.(ab!d.\kappa_\downarrow \parallel ac!d.\kappa_\downarrow) \parallel ab?d.\kappa_\downarrow \parallel ac?d.co!d.\kappa_\downarrow.$$

In order to enforce the communication over the ports ab and ac, one uses the encapsulation operator:

$$T' = \partial_{\{p!d, p?d \,|\, p \in \{ab, ac\}\}, \emptyset}(T).$$

$\_ \parallel \_$

Consider the process network 

, where

- A, B, C are the communicating processes,
- ia, ab, ac, co are the ports, and
- the actions of sending and receiving the datum d over the port p are denoted by, respectively, $p!d$ and $p?d$.

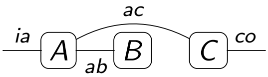
The whole protocol is specified as the term

$$T = ia?d.(ab!d.\kappa_{\downarrow} \parallel ac!d.\kappa_{\downarrow}) \parallel ab?d.\kappa_{\downarrow} \parallel ac?d.co!d.\kappa_{\downarrow}.$$

In order to enforce the communication over the ports ab and ac, one uses the encapsulation operator:

$$T' = \partial_{\{p!d, p?d \mid p \in \{ab, ac\}\}, \emptyset}(T).$$

$\_ \parallel \_$

Consider the process network , where

- A, B, C are the communicating processes,
- ia, ab, ac, co are the ports, and
- the actions of sending and receiving the datum d over the port p are denoted by, respectively, $p!d$ and $p?d$.

The whole protocol is specified as the term

$$T = ia?d.(ab!d.\kappa_{\downarrow} \parallel ac!d.\kappa_{\downarrow}) \parallel ab?d.\kappa_{\downarrow} \parallel ac?d.co!d.\kappa_{\downarrow}.$$

In order to enforce the communication over the ports ab and ac, one uses the encapsulation operator:

$$T' = \partial_{\{p!d, p?d \mid p \in \{ab, ac\}\}, \emptyset}(T).$$

Other case studies

_||_

Demo

! _

The *reentrant server* operation !_ is defined by $\dfrac{x \xrightarrow{a} x'}{!x \xrightarrow{a} x' \parallel !x}$.

In this case a pair of infinite rewriting axioms is derived:

$$!x = !'(x, x)$$
$$!'(a.x', x) = a.(x' \parallel !x).$$

This problem occurs only in the case of operations for which a positive variable appears in the target.

Facts & Other features

PREG Axiomatizer:

- works for operations given in a restricted format, extending the finite trees with predicates system
  - however, it covers most of the operators in the literature
- generates confluent axiomatizations, but only weakly normalizing
  - however, there is a class of systems (linear and syntactically well-founded) for which it is strongly normalizing

PREG Axiomatizer handles:

- format checking,
- implicit predicates for trees ($a.t$ terminates if t terminates).

Facts & Other features

PREG Axiomatizer:

- works for operations given in a restricted format, extending the finite trees with predicates system
  - however, it covers most of the operators in the literature
- generates confluent axiomatizations, but only weakly normalizing
  - however, there is a class of systems (linear and syntactically well-founded) for which it is strongly normalizing

PREG Axiomatizer handles:

- format checking,
- implicit predicates for trees ($a.t$ terminates if t terminates).

Facts & Other features

PREG Axiomatizer:

- works for operations given in a restricted format, extending the finite trees with predicates system
  - however, it covers most of the operators in the literature
- generates confluent axiomatizations, but only weakly normalizing
  - however, there is a class of systems (linear and syntactically well-founded) for which it is strongly normalizing

PREG Axiomatizer handles:

- format checking,
- implicit predicates for trees ($a.t$ terminates if t terminates).

Facts & Other features

PREG Axiomatizer:

- works for operations given in a restricted format, extending the finite trees with predicates system
  - however, it covers most of the operators in the literature
- generates confluent axiomatizations, but only weakly normalizing
  - however, there is a class of systems (linear and syntactically well-founded) for which it is strongly normalizing

PREG Axiomatizer handles:

- format checking,
- implicit predicates for trees ($a.t$ terminates if t terminates).

Future work

Ways to extend and improve the prototype:

- integration with external provers and checkers,
- format checking (operator properties),
- recursively defined terms, open terms,
- universal predicates,
- detect infinite rewriting axiomatizations,
- better user interface,
- ...