Unique Solutions of Guarded Recursive Equations

Rob van Glabbeek

University of Edinburgh, Scotland

University of New South Wales, Sydney, Australia

Stanford University

25 August 2025

Recursive specifications

A recursive specification:

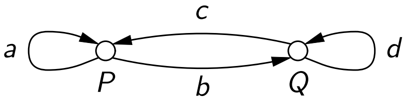
$$\begin{aligned} X &= a.X + b.Y \\ Y &= c.X + d.Y \end{aligned}$$

Recursive specifications

A recursive specification:

$$\begin{aligned} X &= a.X + b.Y \\ Y &= c.X + d.Y \end{aligned}$$

A solution:

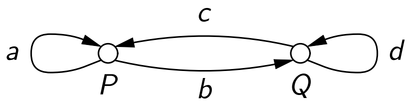


Recursive specifications

A recursive specification:

$$
\begin{aligned}
X &= a.X + b.Y \\
Y &= c.X + d.Y
\end{aligned}
$$

A solution:



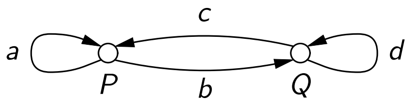
In fact, this is the only solution

Recursive specifications

A recursive specification:

$$X = a.X + b.Y$$
$$Y = c.X + d.Y$$

A solution:



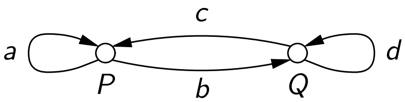
In fact, this is the only solution (up to strong bisimilarity)

Recursive specifications

A recursive specification:

$$X = a.X + b.Y$$
$$Y = c.X + d.Y$$

A solution:



In fact, this is the only solution (up to strong bisimilarity)

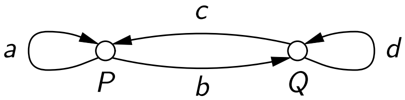
Goal: For a general class of languages,
find a general class of recursive specifications
and show that they have unique solutions

Recursive specifications

A recursive specification:

$$X = a.X + b.Y$$
$$Y = c.X + d.Y$$

A solution:



In fact, this is the only solution (up to strong bisimilarity)

Goal: For a general class of languages,
find a general class of recursive specifications
and show that they have unique solutions
up to strong bisimilarity, or some other equivalence.

Syntax

We postulate an infinite collection $Var \ni X, Y, Z, \ldots$ of *variables*.

A *signature* Σ is a collection of operator symbols f,
each with an *arity* n.

The set of *terms* or *expressions* $\mathbb{T}(\Sigma)$ is defined inductively:

- variables are terms,
- if $f \in \Sigma$ has arity n and $E_1, \ldots, E_n \in \mathbb{T}(\Sigma)$ then $f(E_1, \ldots, E_n) \in \mathbb{T}(\Sigma)$,

Syntax

We postulate an infinite collection $Var \ni X, Y, Z, \ldots$ of *variables*.

A *signature* Σ is a collection of operator symbols f,
each with an *arity* n.

The set of *terms* or *expressions* $\mathbb{T}(\Sigma)$ is defined inductively:

- variables are terms,
- if $f \in \Sigma$ has arity n and $E_1, \ldots, E_n \in \mathbb{T}(\Sigma)$ then $f(E_1, \ldots, E_n) \in \mathbb{T}(\Sigma)$,
- If then $\langle X | \mathcal{S} \rangle \in \mathbb{T}(\Sigma)$.

  $\mathcal{S}$ is a *recursive specification*,

Syntax

We postulate an infinite collection $Var \ni X, Y, Z, \ldots$ of *variables*.

A *signature* Σ is a collection of operator symbols f,
each with an *arity* n.

The set of *terms* or *expressions* $\mathbb{T}(\Sigma)$ is defined inductively:

- variables are terms,
- if $f \in \Sigma$ has arity n and $E_1, \ldots, E_n \in \mathbb{T}(\Sigma)$ then $f(E_1, \ldots, E_n) \in \mathbb{T}(\Sigma)$,
- If then $\langle X|\mathcal{S}\rangle \in \mathbb{T}(\Sigma)$.

  $\mathcal{S}$ is a *recursive specification*, such as

$$\begin{aligned} X &= a.X + b.Y \\ Y &= c.X + d.Y \end{aligned}$$

Syntax

We postulate an infinite collection $Var \ni X, Y, Z, \ldots$ of *variables*.

A *signature* Σ is a collection of operator symbols f,
each with an *arity* n.

The set of *terms* or *expressions* $\mathbb{T}(\Sigma)$ is defined inductively:

- variables are terms,
- if $f \in \Sigma$ has arity n and $E_1, \ldots, E_n \in \mathbb{T}(\Sigma)$ then $f(E_1, \ldots, E_n) \in \mathbb{T}(\Sigma)$,
- If $V_{\mathcal{S}} \subseteq Var$, $\mathcal{S} : V_{\mathcal{S}} \to \mathbb{T}(\Sigma)$ and $X \in V_{\mathcal{S}}$, then $\langle X|\mathcal{S}\rangle \in \mathbb{T}(\Sigma)$.

Here $\mathcal{S}$ is a *recursive specification*, such as

$$\begin{aligned} X &= a.X + b.Y \\ Y &= c.X + d.Y \end{aligned}$$

Syntax

We postulate an infinite collection $Var \ni X, Y, Z, \ldots$ of *variables*.

A *signature* Σ is a collection of operator symbols f,
each with an *arity* n.

The set of *terms* or *expressions* $\mathbb{T}(\Sigma)$ is defined inductively:

- variables are terms,
- if $f \in \Sigma$ has arity n and $E_1, \ldots, E_n \in \mathbb{T}(\Sigma)$ then $f(E_1, \ldots, E_n) \in \mathbb{T}(\Sigma)$,
- If $V_{\mathcal{S}} \subseteq Var$, $\mathcal{S} : V_{\mathcal{S}} \rightarrow \mathbb{T}(\Sigma)$ and $X \in V_{\mathcal{S}}$, then $\langle X|\mathcal{S}\rangle \in \mathbb{T}(\Sigma)$.

Here $\mathcal{S}$ is a *recursive specification*, such as

$$\begin{aligned} X &= a.X + b.Y \\ Y &= c.X + d.Y \end{aligned}$$

A term is *closed* if it has no *free* variables.
Closed terms denote *processes* in process algebra.

Operational semantics

The meaning of a process is a state in a labelled transition system. For a language given by a signature Σ, the set $\mathrm{T}(\Sigma)$ of closed terms or processes form the states.

Operational semantics

The meaning of a process is a state in a labelled transition system.
For a language given by a signature Σ, the set $\mathrm{T}(\Sigma)$ of closed terms or processes form the states.
The transitions are determined by transition rules like these:

$$a.x \xrightarrow{a} x$$

$0 \qquad a \in Act$

Operational semantics

The meaning of a process is a state in a labelled transition system.
For a language given by a signature Σ, the set $\mathrm{T}(\Sigma)$ of closed terms or processes form the states.
The transitions are determined by transition rules like these:

$$a.x \xrightarrow{a} x$$

$$\frac{x \xrightarrow{a} x'}{x \parallel_S y \xrightarrow{a} x' \parallel_S y} \; (a \notin S) \qquad \frac{x \xrightarrow{a} x' \quad y \xrightarrow{a} y'}{x \parallel_S y \xrightarrow{a} x' \parallel_S y'} \; (a \in S) \qquad \frac{y \xrightarrow{a} y'}{x \parallel_S y \xrightarrow{a} x \parallel_S y'} \; (a \notin S)$$

0 $\qquad a \in Act$ $\qquad S \subseteq Act$

Operational semantics

The meaning of a process is a state in a labelled transition system.
For a language given by a signature Σ, the set $\mathrm{T}(\Sigma)$ of closed terms or processes form the states.
The transitions are determined by transition rules like these:

$$a.x \xrightarrow{a} x \qquad\qquad \frac{\langle \mathcal{S}_X|\mathcal{S}\rangle \xrightarrow{a} z}{\langle X|\mathcal{S}\rangle \xrightarrow{a} z}$$

$$\frac{x \xrightarrow{a} x'}{x \parallel_S y \xrightarrow{a} x' \parallel_S y}\ {\scriptstyle (a \notin S)} \qquad \frac{x \xrightarrow{a} x' \quad y \xrightarrow{a} y'}{x \parallel_S y \xrightarrow{a} x' \parallel_S y'}\ {\scriptstyle (a \in S)} \qquad \frac{y \xrightarrow{a} y'}{x \parallel_S y \xrightarrow{a} x \parallel_S y'}\ {\scriptstyle (a \notin S)}$$

$0 \qquad a \in Act \qquad S \subseteq Act$

Operational semantics

The meaning of a process is a state in a labelled transition system.
For a language given by a signature Σ, the set $\mathrm{T}(\Sigma)$ of closed terms or processes form the states.
The transitions are determined by transition rules like these:

$$a.x \xrightarrow{a} x \qquad \begin{array}{l} X = a.X + b.Y \\ Y = c.X + d.Y \end{array} \qquad \frac{\langle \mathcal{S}_X | \mathcal{S} \rangle \xrightarrow{a} z}{\langle X | \mathcal{S} \rangle \xrightarrow{a} z} \qquad \frac{a.\langle X | \mathcal{S} \rangle + b.\langle Y | \mathcal{S} \rangle \xrightarrow{a} z}{\langle X | \mathcal{S} \rangle \xrightarrow{a} z}$$

$$\frac{x \xrightarrow{a} x'}{x \|_S y \xrightarrow{a} x' \|_S y} \ (a \notin S) \qquad \frac{x \xrightarrow{a} x' \quad y \xrightarrow{a} y'}{x \|_S y \xrightarrow{a} x' \|_S y'} \ (a \in S) \qquad \frac{y \xrightarrow{a} y'}{x \|_S y \xrightarrow{a} x \|_S y'} \ (a \notin S)$$

$0 \qquad a \in Act \qquad R, S \subseteq Act$

Operational semantics

The meaning of a process is a state in a labelled transition system. For a language given by a signature Σ, the set $\mathrm{T}(\Sigma)$ of closed terms or processes form the states.
The transitions are determined by transition rules like these:

$$a.x \xrightarrow{a} x \qquad\qquad \frac{\langle \mathcal{S}_X | \mathcal{S} \rangle \xrightarrow{a} z}{\langle X | \mathcal{S} \rangle \xrightarrow{a} z}$$

$$\frac{x \xrightarrow{a} x'}{x \parallel_S y \xrightarrow{a} x' \parallel_S y}\ (a \notin S) \qquad \frac{x \xrightarrow{a} x' \quad y \xrightarrow{a} y'}{x \parallel_S y \xrightarrow{a} x' \parallel_S y'}\ (a \in S) \qquad \frac{y \xrightarrow{a} y'}{x \parallel_S y \xrightarrow{a} x \parallel_S y'}\ (a \notin S)$$

$$\frac{x \xrightarrow{a} x'}{en_R(x) \xrightarrow{a} en_R(x')} \qquad \frac{x \stackrel{a}{\not\longrightarrow}}{en_R(x) \xrightarrow{a} en_R(x)}\ (a \in R)$$

$0 \qquad a \in Act \qquad R, S \subseteq Act$

Operational semantics

The meaning of a process is a state in a labelled transition system. For a language given by a signature Σ, the set $\mathrm{T}(\Sigma)$ of closed terms or processes form the states.
The transitions are determined by transition rules like these:

$$a.x \xrightarrow{a} x \qquad\qquad \frac{\langle \mathcal{S}_X | \mathcal{S} \rangle \xrightarrow{a} z}{\langle X | \mathcal{S} \rangle \xrightarrow{a} z}$$

$$\frac{x \xrightarrow{a} x'}{x \parallel_S y \xrightarrow{a} x' \parallel_S y}\ (a \notin S) \qquad \frac{x \xrightarrow{a} x' \quad y \xrightarrow{a} y'}{x \parallel_S y \xrightarrow{a} x' \parallel_S y'}\ (a \in S) \qquad \frac{y \xrightarrow{a} y'}{x \parallel_S y \xrightarrow{a} x \parallel_S y'}\ (a \notin S)$$

$$\frac{x \xrightarrow{a} x'}{en_R(x) \xrightarrow{a} en_R(x')} \qquad \frac{x \stackrel{a}{\not\longrightarrow}}{en_R(x) \xrightarrow{a} en_R(x)}\ (a \in R)$$

$$\frac{x \xrightarrow{a} x'}{x \triangleright y \xrightarrow{a} x' \triangleright y}\ (a \neq sw) \qquad \frac{x \xrightarrow{sw} x'}{x \triangleright y \xrightarrow{sw} y \triangleright x'}$$

$0 \qquad a \in Act \qquad R, S \subseteq Act$

The meaning of negative premises

$$a.x \xrightarrow{a} x \qquad \frac{\langle \mathcal{S}_X | \mathcal{S} \rangle \xrightarrow{a} z}{\langle X | \mathcal{S} \rangle \xrightarrow{a} z}$$

$$\frac{x \xrightarrow{a} x'}{x \parallel_S y \xrightarrow{a} x' \parallel_S y}\ (a \notin S) \qquad \frac{x \xrightarrow{a} x' \quad y \xrightarrow{a} y'}{x \parallel_S y \xrightarrow{a} x' \parallel_S y'}\ (a \in S) \qquad \frac{y \xrightarrow{a} y'}{x \parallel_S y \xrightarrow{a} x \parallel_S y'}\ (a \notin S)$$

$$\frac{x \xrightarrow{a} x'}{en_R(x) \xrightarrow{a} en_R(x')} \qquad \frac{x \stackrel{a}{\not\longrightarrow}}{en_R(x) \xrightarrow{a} en_R(x)}\ (a \in R)$$

$$\frac{x \xrightarrow{a} x'}{x \triangleright y \xrightarrow{a} x' \triangleright y}\ (a \neq sw) \qquad \frac{x \xrightarrow{sw} x'}{x \triangleright y \xrightarrow{sw} y \triangleright x'}$$

Let $\mathcal{S}$ be the recursive specification $X = b.0 \parallel_b en_b(X)$.

Under the assumption $\langle X | \mathcal{S} \rangle \stackrel{b}{\not\longrightarrow}$ one proves $\langle X | \mathcal{S} \rangle \xrightarrow{b} 0 \parallel_b en_b(\langle X | \mathcal{S} \rangle)$.
However, there is no outright proof of any transition $\langle X | \mathcal{S} \rangle \xrightarrow{b} Q$.

The meaning of negative premises

The well founded semantics assigns to every set $\mathcal{T}$ of transition rules a 3-valued labelled transition system (CT, PT) with $CT \subseteq PT$, such that

PT^- is the set of negative literals that do not deny any $\beta \in CT^+$ $\quad \beta \in PT^+$ iff $\mathcal{T} \vdash \frac{PT^-}{\beta}$

CT^- is the set of negative literals that do not deny any $\beta \in PT^+$ $\quad \beta \in CT^+_\lambda$ iff $\mathcal{T} \vdash \frac{CT^-}{\beta}$.

Here $P \stackrel{c}{\not\longrightarrow}$ denies $P \stackrel{c}{\longrightarrow} Q$.

In fact, it assign the unique 3-valued LTS with these properties.

Main theorem

For any language with a SOS in the ready simulation format,
guarded recursive specifications have unique solutions
up to strong bisimulation equivalence / ready simulation equivalence.

Main theorem

For any language with a SOS in the ready simulation format,
guarded recursive specifications have unique solutions
up to strong bisimulation equivalence / ready simulation equivalence.

Example: $X = a.X$ is a guarded recursive specification
in a language with an SOS in the ready simulation format.

Main theorem

For any language with a SOS in the ready simulation format,
guarded recursive specifications have unique solutions
up to strong bisimulation equivalence / ready simulation equivalence.

Example: $X = a.X$ is a guarded recursive specification
in a language with an SOS in the ready simulation format.

If P and Q and processes that satisfy

$$P =_B a.P \qquad \text{and} \qquad Q =_B a.Q$$

then they are both solutions of this rec. spec. up to $=_B$.

Main theorem

For any language with a SOS in the ready simulation format,
guarded recursive specifications have unique solutions
up to strong bisimulation equivalence / ready simulation equivalence.

Example: $X = a.X$ is a guarded recursive specification
in a language with an SOS in the ready simulation format.

If P and Q and processes that satisfy

$$P =_B a.P \qquad \text{and} \qquad Q =_B a.Q$$

then they are both solutions of this rec. spec. up to $=_B$.

The theorem now says that $P =_B Q$.

Bisimulation equivalence

A *bisimulation* $\mathscr{B}$ is a binary relation on processes such that,

- if $P \mathrel{\mathscr{B}} Q$ and $P \xrightarrow{a} P'$, then $\exists Q'$ with $Q \xrightarrow{a} Q'$ and $P' \mathrel{\mathscr{B}} Q'$,
- if $P \mathrel{\mathscr{B}} Q$ and $Q \xrightarrow{a} Q'$, then $\exists P'$ with $P \xrightarrow{a} P'$ and $P' \mathrel{\mathscr{B}} Q'$.

Write $P \equiv_B Q$ if $P \mathrel{\mathscr{B}} Q$ for some bisimulation $\mathscr{B}$.

Ready simulation equivalence

A *ready simulation* $\mathscr{B}$ is a binary relation on processes such that,

- if $P \mathrel{\mathscr{B}} Q$ and $P \xrightarrow{a} P'$, then $\exists Q'$ with $Q \xrightarrow{a} Q'$ and $P' \mathrel{\mathscr{B}} Q'$,
- if $P \mathrel{\mathscr{B}} Q$ and $Q \xrightarrow{a}$, then $P \xrightarrow{a}$

Write $P \sqsubseteq_{RS} Q$ if $P \mathrel{\mathscr{B}} Q$ for some ready simulation $\mathscr{B}$.

Write $P \equiv_{RS} Q$ if $P \sqsubseteq_{RS} Q$ and $Q \sqsubseteq_{RS} P$.

simulation equivalence

A *simulation* $\mathscr{B}$ is a binary relation on processes such that,

- if $P \mathrel{\mathscr{B}} Q$ and $P \xrightarrow{a} P'$, then $\exists Q'$ with $Q \xrightarrow{a} Q'$ and $P' \mathrel{\mathscr{B}} Q'$,

Write $P \sqsubseteq_S Q$ if $P \mathrel{\mathscr{B}} Q$ for some simulation $\mathscr{B}$.

Write $P \equiv_S Q$ if $P \sqsubseteq_S Q$ and $Q \sqsubseteq_S P$.

Ready simulation format

Recursion is handled by the recursion rule.
All other rules are recursion-free and satisfy:

- The source contains at most one operator symbol, and all variables are different.

Ready simulation format

Recursion is handled by the recursion rule.
All other rules are recursion-free and satisfy:

- The source contains at most one operator symbol, and all variables are different.
- The right-hand sides of positive premises are variables that are all distinct and do not occur in the source.

Ready simulation format

Recursion is handled by the recursion rule.
All other rules are recursion-free and satisfy:

- The source contains at most one operator symbol, and all variables are different.
- The right-hand sides of positive premises are variables that are all distinct and do not occur in the source.
  The above defines the ntyft/ntxft format

Ready simulation format

Recursion is handled by the recursion rule.
All other rules are recursion-free and satisfy:

- The source contains at most one operator symbol, and all variables are different.
- The right-hand sides of positive premises are variables that are all distinct and do not occur in the source.
  The above defines the ntyft/ntxft format
- No *lookahead*.

A language in ready simulation format

$$a.x \xrightarrow{a} x \qquad \frac{\langle \mathcal{S}_X | \mathcal{S} \rangle \xrightarrow{a} z}{\langle X | \mathcal{S} \rangle \xrightarrow{a} z}$$

$$\frac{x \xrightarrow{a} x'}{x \parallel_S y \xrightarrow{a} x' \parallel_S y}\ (a \notin S) \qquad \frac{x \xrightarrow{a} x' \quad y \xrightarrow{a} y'}{x \parallel_S y \xrightarrow{a} x' \parallel_S y'}\ (a \in S) \qquad \frac{y \xrightarrow{a} y'}{x \parallel_S y \xrightarrow{a} x \parallel_S y'}\ (a \notin S)$$

$$\frac{x \xrightarrow{a} x'}{en_R(x) \xrightarrow{a} en_R(x')} \qquad \frac{x \stackrel{a}{\not\longrightarrow}}{en_R(x) \xrightarrow{a} en_R(x)}\ (a \in R)$$

$$\frac{x \xrightarrow{a} x'}{x \rhd y \xrightarrow{a} x' \rhd y}\ (a \neq sw) \qquad \frac{x \xrightarrow{sw} x'}{x \rhd y \xrightarrow{sw} y \rhd x'}$$

$$\frac{en_b(x \rhd y) \xrightarrow{a} x'}{h(x,y,z) \xrightarrow{a} h(z,x',y)}$$

$\mathbf{0}$ $\qquad a \in Act \qquad R, S \subseteq Act$

Guards, and guarded occurrences of variables

An occurrence of a variable x in a term E is *guarded* if it occurs in a subterm $f(E_1, \ldots, E_n)$ of E, such that, for some $1 \leq i \leq n$,

- the occurrence lays in E_i,
- and f guards its i^{th} argument.

f *guards* its i^{th} argument if for each transition rule

$$\frac{H}{f(x_1, \ldots, x_n) \xrightarrow{a} E}$$

there is no premise $G \xrightarrow{b} y$ or $G \stackrel{b}{\not\longrightarrow}$ in H,
containing an occurrence of the variable x_i.

Guards, and guarded occurrences of variables

An occurrence of a variable x in a term E is *guarded* if it occurs in a subterm $f(E_1, \ldots, E_n)$ of E, such that, for some $1 \leq i \leq n$,

- the occurrence lays in E_i,
- and f guards its i^{th} argument.

f *guards* its i^{th} argument if for each transition rule

$$\frac{H}{f(x_1, \ldots, x_n) \xrightarrow{a} E}$$

there is no premise $G \xrightarrow{b} y$ or $G \stackrel{b}{\not\longrightarrow}$ in H,
containing an occurrence of the variable x_i.

Example: the variable y occurs guarded in the term $en_b(x \triangleright y)$.
The variable x does not.

Guards, and guarded occurrences of variables

An occurrence of a variable x in a term E is *guarded* if it occurs in a subterm $f(E_1, \ldots, E_n)$ of E, such that, for some $1 \leq i \leq n$,

- the occurrence lays in E_i,
- and f guards its i^{th} argument.

f *guards* its i^{th} argument if for each transition rule

$$\frac{H}{f(x_1, \ldots, x_n) \xrightarrow{a} E}$$

there is no premise $G \xrightarrow{b} y$ or $G \stackrel{b}{\not\longrightarrow}$ in H,
containing an occurrence of the variable x_i.

Example: the variable y occurs guarded in the term $en_b(x \triangleright y)$.
The variable x does not.
Example: the variable z occurs guarded in the term $h(x, y, z)$.
The variable x does not.

Guards, and guarded occurrences of variables

An occurrence of a variable x in a term E is *guarded* if it occurs in a subterm $f(E_1, \ldots, E_n)$ of E, such that, for some $1 \leq i \leq n$,

- the occurrence lays in E_i,
- and f guards its i^{th} argument.

f *guards* its i^{th} argument if for each transition rule

$$\frac{H}{f(x_1, \ldots, x_n) \xrightarrow{a} E}$$

there is no premise $G \xrightarrow{b} y$ or $G \overset{b}{\not\longrightarrow}$ in H,
containing an unguarded occurrence of the variable x_i.

Example: the variable y occurs guarded in the term $en_b(x \triangleright y)$.
The variable x does not.
Example: the variable z occurs guarded in the term $h(x, y, z)$.
The variable x does not.

Guards, and guarded occurrences of variables

An occurrence of a variable x in a term E is *guarded* if it occurs in a subterm $f(E_1, \ldots, E_n)$ of E, such that, for some $1 \leq i \leq n$,

- the occurrence lays in E_i,
- and f guards its i^{th} argument.

f *guards* its i^{th} argument if for each transition rule

$$\frac{H}{f(x_1, \ldots, x_n) \xrightarrow{a} E}$$

there is no premise $G \xrightarrow{b} y$ or $G \stackrel{b}{\not\longrightarrow}$ in H,
containing an unguarded occurrence of the variable x_i.

Example: the variable y occurs guarded in the term $en_b(x \triangleright y)$.
The variable x does not.
Example: the variable z occurs guarded in the term $h(x, y, z)$.
The variable x does not. The occurrence of y is guarded.

Guarded recursive specifications

A recursive specification $\mathcal{S}$ is *manifestly guarded* if $\forall X, Y \in V_{\mathcal{S}}$ all occurrences of the variable Y in the term $\mathcal{S}_X$ are guarded.

Guarded recursive specifications

A recursive specification $\mathcal{S}$ is *manifestly guarded* if $\forall X, Y \in V_{\mathcal{S}}$ all occurrences of the variable Y in the term $\mathcal{S}_X$ are guarded.

Example: The recursive specification $\{X = h(a.b.sw.X, X, X)\}$ is manifestly guarded.

Guarded recursive specifications

A recursive specification $\mathcal{S}$ is *manifestly guarded* if $\forall X, Y \in V_{\mathcal{S}}$ all occurrences of the variable Y in the term $\mathcal{S}_X$ are guarded.

It is *guarded* if it can be converted into a manifestly guarded recursive specification by repeated substitution of expressions $\mathcal{S}_Y$ for variables $Y \in V_{\mathcal{S}}$ occurring in expressions $\mathcal{S}_Z$ for $Z \in V_{\mathcal{S}}$.

Example: The recursive specification $\{X = h(a.b.sw.X, X, X)\}$ is manifestly guarded.

Guarded recursive specifications

A recursive specification $\mathcal{S}$ is *manifestly guarded* if $\forall X, Y \in V_{\mathcal{S}}$ all occurrences of the variable Y in the term $\mathcal{S}_X$ are guarded.

It is *guarded* if it can be converted into a manifestly guarded recursive specification by repeated substitution of expressions $\mathcal{S}_Y$ for variables $Y \in V_{\mathcal{S}}$ occurring in expressions $\mathcal{S}_Z$ for $Z \in V_{\mathcal{S}}$.

Example: The recursive specification $\{X = h(a.b.sw.X, X, X)\}$ is manifestly guarded.

Example: $\{X = Y \triangleright X,\ Y = (a.b.sw.X) \triangleright Y\}$ is guarded.

Main theorem

For any language with a SOS in the ready simulation format,
guarded recursive specifications have unique solutions
up to strong bisimulation equivalence / ready simulation equivalence.

Main theorem

For any language with a SOS in the positive ready simulation format, guarded recursive specifications have unique solutions up to simulation equivalence.

The recursive specification principle (RSP)

Guarded recursive specifications have at most one solution.

$$\frac{\vec{P} \equiv \mathcal{S}[\vec{P}] \qquad \vec{Q} \equiv \mathcal{S}[\vec{Q}]}{\vec{P} \equiv \vec{Q}}$$

The recursive specification principle (RSP)

Guarded recursive specifications have at most one solution.

$$\frac{\vec{P} \equiv \mathcal{S}[\vec{P}] \qquad \vec{Q} \equiv \mathcal{S}[\vec{Q}]}{\vec{P} \equiv \vec{Q}}$$

Let $\mathcal{S}$ be

$$\begin{aligned} X &= a.X + b.Y \\ Y &= c.X + d.Y \end{aligned}$$

Let $\vec{P} = (P_X, P_Y)$ be a vector (pair) of two processes, such that

$$\begin{aligned} P_X &\equiv a.P_X + b.P_Y \\ P_Y &\equiv c.P_X + d.P_Y \end{aligned}$$

Then I write $\vec{P} \equiv \mathcal{S}[\vec{P}]$. In this case $\vec{P}$ is a *solution* of $\mathcal{S}$.

The recursive specification principle (RSP)

Guarded recursive specifications have at most one solution.

$$\frac{\vec{P} \equiv \mathcal{S}[\vec{P}] \qquad \vec{Q} \equiv \mathcal{S}[\vec{Q}]}{\vec{P} \equiv \vec{Q}} \qquad \frac{\vec{P} \sqsubseteq \mathcal{S}[\vec{P}] \qquad \vec{Q} \sqsubseteq \mathcal{S}[\vec{Q}]}{\vec{P} \quad ?? \quad \vec{Q}}.$$

Let $\mathcal{S}$ be

$$\begin{aligned} X &= a.X + b.Y \\ Y &= c.X + d.Y \end{aligned}$$

Let $\vec{P} = (P_X, P_Y)$ be a vector (pair) of two processes, such that

$$\begin{aligned} P_X &\sqsubseteq a.P_X + b.P_Y \\ P_Y &\sqsubseteq c.P_X + d.P_Y \end{aligned}$$

Then I write $\vec{P} \sqsubseteq \mathcal{S}[\vec{P}]$. In this case $\vec{P}$ is a *presolution* of $\mathcal{S}$.

The recursive specification principle (RSP)

Guarded recursive specifications have at most one solution.

$$\frac{\vec{P} \equiv \mathcal{S}[\vec{P}] \qquad \vec{Q} \equiv \mathcal{S}[\vec{Q}]}{\vec{P} \equiv \vec{Q}} \qquad \frac{\vec{P} \sqsupseteq \mathcal{S}[\vec{P}] \qquad \vec{Q} \sqsupseteq \mathcal{S}[\vec{Q}]}{\vec{P} \quad ?? \quad \vec{Q}}.$$

Let $\mathcal{S}$ be

$$\begin{aligned} X &= a.X + b.Y \\ Y &= c.X + d.Y \end{aligned}$$

Let $\vec{P} = (P_X, P_Y)$ be a vector (pair) of two processes, such that

$$\begin{aligned} P_X &\sqsupseteq a.P_X + b.P_Y \\ P_Y &\sqsupseteq c.P_X + d.P_Y \end{aligned}$$

Then I write $\vec{P} \sqsupseteq \mathcal{S}[\vec{P}]$. In this case $\vec{P}$ is a *postsolution* of $\mathcal{S}$.

The recursive specification principle (RSP)

Guarded recursive specifications have at most one solution.

$$\frac{\vec{P} \equiv \mathcal{S}[\vec{P}] \qquad \mathcal{S}[\vec{Q}] \equiv \vec{Q}}{\vec{P} \equiv \vec{Q}}$$

The recursive specification principle (RSP)

Guarded recursive specifications have at most one solution.

$$\frac{\vec{P} \equiv \mathcal{S}[\vec{P}] \qquad \mathcal{S}[\vec{Q}] \equiv \vec{Q}}{\vec{P} \equiv \vec{Q}} \qquad \frac{\vec{P} \sqsubseteq \mathcal{S}[\vec{P}] \qquad \mathcal{S}[\vec{Q}] \sqsubseteq \vec{Q}}{\vec{P} \sqsubseteq \vec{Q}}.$$

The recursive specification principle (RSP)

Guarded recursive specifications have at most one solution.

$$\frac{\vec{P} \equiv \mathcal{S}[\vec{P}] \qquad \mathcal{S}[\vec{Q}] \equiv \vec{Q}}{\vec{P} \equiv \vec{Q}} \qquad \frac{\vec{P} \sqsubseteq \mathcal{S}[\vec{P}] \qquad \mathcal{S}[\vec{Q}] \sqsubseteq \vec{Q}}{\vec{P} \sqsubseteq \vec{Q}}.$$

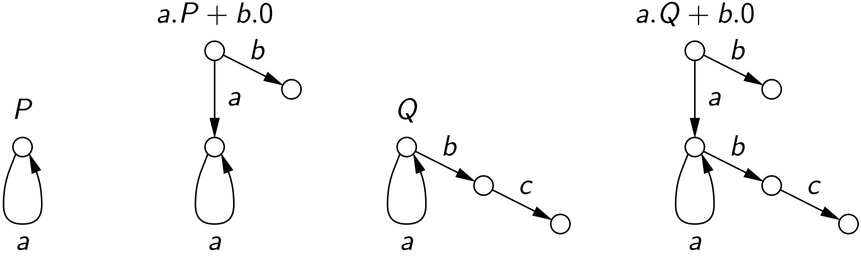
If $\equiv$ is the kernel of $\sqsubseteq$ (meaning $P \equiv Q \Leftrightarrow P \sqsubseteq Q \wedge Q \sqsubseteq P$) then RSP for $\sqsubseteq$ implies RSP for $\equiv$.

RSP for the simulation preorder

Let $\mathcal{S} := \{X = a.X + b.0\}$.
Take $P := \langle Y | \{Y = a.Y\} \rangle$ and $Q := \langle Z | \{Z = a.Z \; + a.b.0\} \rangle$.

RSP for the simulation preorder

Let $\mathcal{S} := \{X = a.X + b.0\}$.
Take $P := \langle Y | \{Y = a.Y\} \rangle$ and $Q := \langle Z | \{Z = a.Z \ + a.b.0\} \rangle$.



Now $P \sqsubseteq_S \mathcal{S}[P] = a.P + b.0$ and $\mathcal{S}[Q] = a.Q + b.0 \sqsubseteq_S Q$.
Hence, by RSP, $P \sqsubseteq_S Q$.

The recursive definition principle (RDP)

Guarded recursive specifications have at least one solution.

$$\langle V_{\mathcal{S}} | \mathcal{S} \rangle \equiv \mathcal{S}[\langle V_{\mathcal{S}} | \mathcal{S} \rangle]$$

Here $\langle V_{\mathcal{S}} | \mathcal{S} \rangle$ is the vector with components $\langle X | \mathcal{S} \rangle$ for each bound variable X of $\mathcal{S}$. It is the default solution of $\mathcal{S}$.

The recursive definition principle (RDP)

Guarded recursive specifications have at least one solution.

$$\langle V_{\mathcal{S}} | \mathcal{S} \rangle \equiv \mathcal{S}[\langle V_{\mathcal{S}} | \mathcal{S} \rangle]$$

Here $\langle V_{\mathcal{S}} | \mathcal{S} \rangle$ is the vector with components $\langle X | \mathcal{S} \rangle$ for each bound variable X of $\mathcal{S}$. It is the default solution of $\mathcal{S}$.

RSP:

$$\frac{\vec{P} \equiv \mathcal{S}[\vec{P}] \qquad \mathcal{S}[\vec{Q}] \equiv \vec{Q}}{\vec{P} \equiv \vec{Q}} \qquad \frac{\vec{P} \sqsubseteq \mathcal{S}[\vec{P}] \qquad \mathcal{S}[\vec{Q}] \sqsubseteq \vec{Q}}{\vec{P} \sqsubseteq \vec{Q}}.$$

The recursive definition principle (RDP)

Guarded recursive specifications have at least one solution.

$$\langle V_{\mathcal{S}} | \mathcal{S} \rangle \equiv \mathcal{S}[\langle V_{\mathcal{S}} | \mathcal{S} \rangle]$$

Here $\langle V_{\mathcal{S}} | \mathcal{S} \rangle$ is the vector with components $\langle X | \mathcal{S} \rangle$ for each bound variable X of $\mathcal{S}$. It is the default solution of $\mathcal{S}$.

RSP:

$$\frac{\vec{P} \equiv \mathcal{S}[\vec{P}] \qquad \mathcal{S}[\vec{Q}] \equiv \vec{Q}}{\vec{P} \equiv \vec{Q}} \qquad \frac{\vec{P} \sqsubseteq \mathcal{S}[\vec{P}] \qquad \mathcal{S}[\vec{Q}] \sqsubseteq \vec{Q}}{\vec{P} \sqsubseteq \vec{Q}}.$$

In combination with RDP, RSP can be restated as:

$$\frac{\vec{P} \equiv \mathcal{S}[\vec{P}]}{\vec{P} \equiv \langle V_{\mathcal{S}} | \mathcal{S} \rangle} \qquad \frac{\vec{P} \sqsubseteq \mathcal{S}[\vec{P}]}{\vec{P} \sqsubseteq \langle V_{\mathcal{S}} | \mathcal{S} \rangle} \qquad \frac{\mathcal{S}[\vec{Q}] \sqsubseteq \vec{Q}}{\langle V_{\mathcal{S}} | \mathcal{S} \rangle \sqsubseteq \vec{Q}}\ . \tag{1}$$

Lookahead

$$\frac{x \xrightarrow{a} y \quad y \xrightarrow{b} z}{f(x) \xrightarrow{b} z}$$

Lookahead

$$\frac{x \xrightarrow{a} y \quad y \xrightarrow{b} z}{f(x) \xrightarrow{b} z}$$

The recursive equation $X = f(a.X)$ does not have a unique solution.

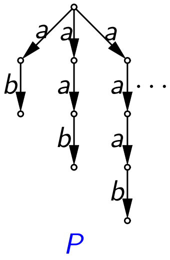
Lookahead

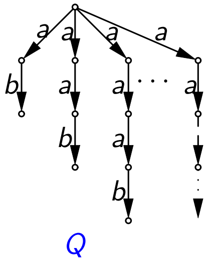
$$\frac{x \xrightarrow{a} y \quad y \xrightarrow{b} z}{f(x) \xrightarrow{b} z}$$

The recursive equation $X = f(a.X)$ does not have a unique solution.

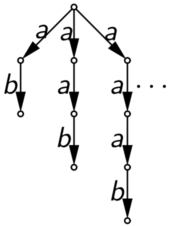
This is why I have no unique-solution results for systems with lookahead.

Linear time semantics

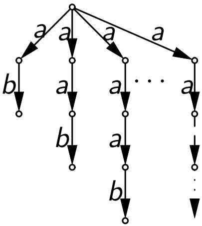




Linear time semantics



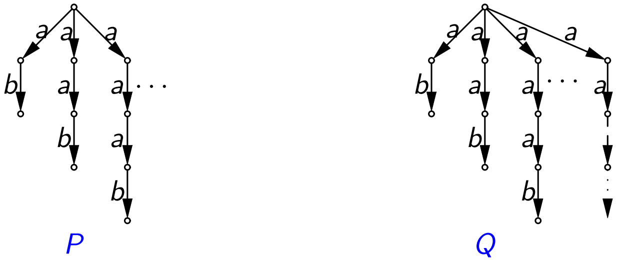
P



Q

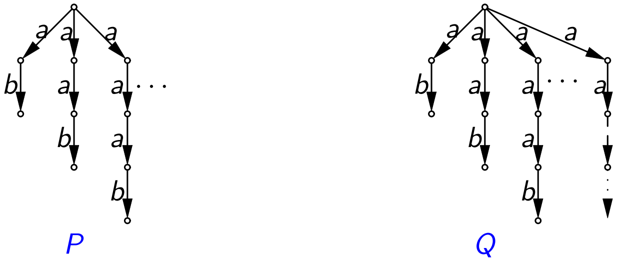
$P \not\equiv Q$ in any semantics $\equiv$ that respects liveness properties.

Linear time semantics



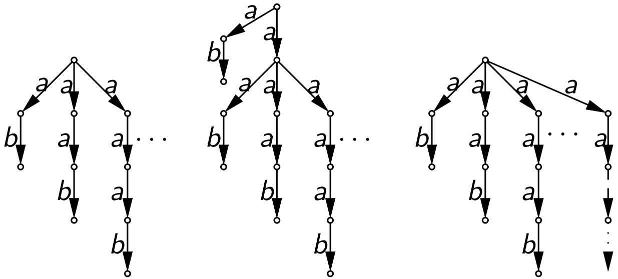
$P \not\equiv Q$ in any semantics $\equiv$ that respects liveness properties.
For instance, completed trace semantics with infinite traces.

Linear time semantics



$P \not\equiv Q$ in any semantics $\equiv$ that respects liveness properties.
For instance, completed trace semantics with infinite traces.
Or ready trace semantics with infinite ready traces.

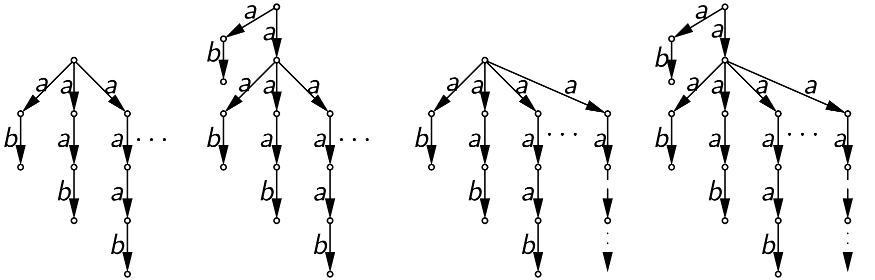
Linear time semantics



P $\qquad P' := a.b + a.P$ $\qquad Q$

$P \not\equiv Q$ in any semantics $\equiv$ that respects liveness properties.
For instance, completed trace semantics with infinite traces.
Or ready trace semantics with infinite ready traces.

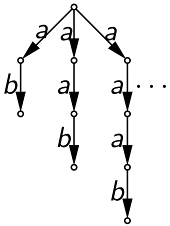
Linear time semantics



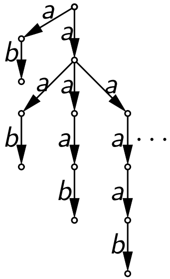
P $\quad$ $P' := a.b + a.P$ $\quad$ Q $\quad$ $Q' := a.b + a.Q$

$P \not\equiv Q$ in any semantics $\equiv$ that respects liveness properties.
For instance, completed trace semantics with infinite traces.
Or ready trace semantics with infinite ready traces.

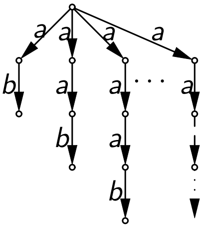
Linear time semantics



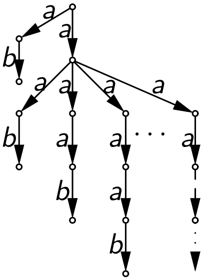
P



$P' := a.b + a.P$



Q

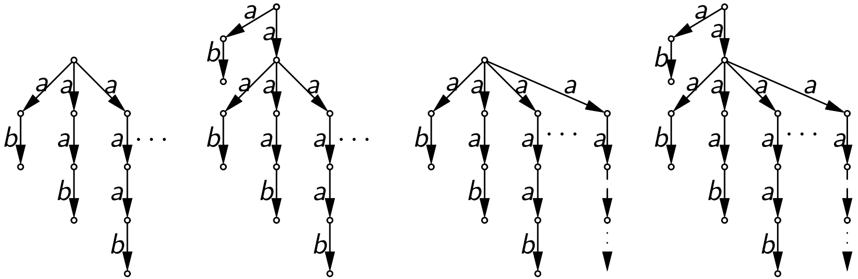


$Q' := a.b + a.Q$

$P \not\equiv Q$ in any semantics $\equiv$ that respects liveness properties.
For instance, completed trace semantics with infinite traces.
Or ready trace semantics with infinite ready traces.

For such a linear time equivalence $\equiv$, we have $P \equiv P'$ and $Q \equiv Q'$.

Linear time semantics



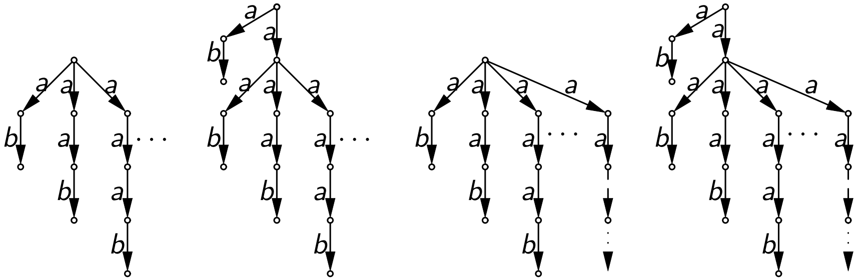
P $\qquad$ $P' := a.b + a.P$ $\qquad$ Q $\qquad$ $Q' := a.b + a.Q$

$P \not\equiv Q$ in any semantics $\equiv$ that respects liveness properties.
For instance, completed trace semantics with infinite traces.
Or ready trace semantics with infinite ready traces.

For such a linear time equivalence $\equiv$, we have $P \equiv P'$ and $Q \equiv Q'$.
So both P and Q are solutions of the guarded recursive specification

$$X = a.b + a.X$$

Linear time semantics



P $\qquad$ $P' := a.b + a.P$ $\qquad$ Q $\qquad$ $Q' := a.b + a.Q$

$P \not\equiv Q$ in any semantics $\equiv$ that respects liveness properties.
For instance, completed trace semantics with infinite traces.
Or ready trace semantics with infinite ready traces.

For such a linear time equivalence $\equiv$, we have $P \equiv P'$ and $Q \equiv Q'$.
So both P and Q are solutions of the guarded recursive specification

$$X = a.b + a.X$$

For such semantics, guarded recursive specifications do not have unique solutions.

Full congruence for guarded recursion

$$\mathcal{S} \equiv \mathcal{S}' \Rightarrow \langle V_{\mathcal{S}} | \mathcal{S} \rangle \equiv \langle V_{\mathcal{S}} | \mathcal{S}' \rangle$$

i.e.

if $\mathcal{S}_Y \equiv \mathcal{S}'_Y$ for all $Y \in V_{\mathcal{S}}$ then $\langle X | \mathcal{S} \rangle \equiv \langle X | \mathcal{S}' \rangle$ for all $X \in \mathcal{S}$.

Full congruence for guarded recursion

$$\mathcal{S} \equiv \mathcal{S}' \Rightarrow \langle V_{\mathcal{S}} | \mathcal{S} \rangle \equiv \langle V_{\mathcal{S}} | \mathcal{S}' \rangle$$

i.e.

if $\mathcal{S}_Y \equiv \mathcal{S}'_Y$ for all $Y \in V_{\mathcal{S}}$ then $\langle X | \mathcal{S} \rangle \equiv \langle X | \mathcal{S}' \rangle$ for all $X \in \mathcal{S}$.

This holds if either $\mathcal{S}$ or $\mathcal{S}'$ is guarded.

Ground-complete axiomatisations

$$P \equiv Q \quad \Leftrightarrow \quad Ax \vdash P = Q$$

Ground-complete axiomatisations

$$P \equiv Q \quad \Leftrightarrow \quad Ax \vdash P = Q$$

Languages like ACP and CCS have sound and ground-complete equational axiomatisations for recursion-free terms, modelling finite loop-free processes.

Ground-complete axiomatisations

$$P \equiv Q \quad \Leftrightarrow \quad Ax \vdash P = Q$$

Languages like ACP and CCS have sound and ground-complete equational axiomatisations for recursion-free terms, modelling finite loop-free processes.

Adding RDP and RSP makes these into sound and ground-complete conditional equational axiomatisations for guarded expressions, modelling arbitrary infinite-state finitely-branching processes.

Ground-complete axiomatisations

$$P \equiv Q \quad \Leftrightarrow \quad Ax \vdash P = Q$$

Languages like ACP and CCS have sound and ground-complete equational axiomatisations for recursion-free terms, modelling finite loop-free processes.

Adding RDP and RSP makes these into sound and ground-complete conditional equational axiomatisations for guarded expressions, modelling arbitrary infinite-state finitely-branching processes.

The same works for any GSOS language!

Future work

It would be great to extend this work to branching bisimilarity and/or other weak equivalences.