The Stable Revivals Model in CSP-Prover

D G Samuel (Swansea), Y Isobe (AIST, Japan), M Roggenbach (Swansea)

AVoCS 2008
A trend: Theorem proving for process algebra
Theorem proving for process algebra

CSP
- CSP in HOL (Camilleri '90)
- HOL-CSP (Tej/Wolf '97)
- CSP-T (Dutertre/Schneider '01)
- CSP-F (Wei/Heather '05)
- CSP-Prover ('05,'06,'08a,'08b)
- CSP-N (Kammueller '07)

μCRL/ACP
- van de Pol ('01)
- Badban et al ('05)

CCS
- Nesi ('92)

π-calculus
- Röckl/Hirschkoff ('03)
- Bengtson/Parrow ('07)
Outline of the talk

- The stable revivals model $\mathcal{R}$
- CSP-Prover
- Implementation of the model $\mathcal{R}$ in CSP-Prover
- Mechanical validation of algebraic laws
Outline of the talk

- The stable revivals model $\mathcal{R}$
- CSP-Prover
  - Implementation of the model $\mathcal{R}$ in CSP-Prover
  - Mechanical validation of algebraic laws
Outline of the talk

- The stable revivals model $\mathcal{R}$
- CSP-Prover
- Implementation of the model $\mathcal{R}$ in CSP-Prover
- Mechanical validation of algebraic laws
Outline of the talk

- The stable revivals model $\mathcal{R}$
- CSP-Prover
- Implementation of the model $\mathcal{R}$ in CSP-Prover
- Mechanical validation of algebraic laws
The stable revivals model
The stable revivals model

- The stable revivals model \( \mathcal{R} \) (Roscoe 2005, revised 2007) is a new semantic model of CSP.
- It is a finite observation model.
- Developed to reason about Responsiveness and Stuck-freeness.
- A process \( Q \) is responsive to process \( P \) if process \( Q \) will not cause process \( P \) to deadlock by not responding when expected by \( P \).
- A network is stuck if network of process does not terminate leaving one partner hanging.
- The model \( T \), the model \( F \), and the model \( R \) are successively more refined.
- There does not exist any model that refines the model \( R \) and is more abstract than the stable acceptance model \( A \) and the refusal testing model \( RT \).
The stable revivals model

- The stable revivals model $\mathcal{R}$ (Roscoe 2005, revised 2007) is a new semantic model of CSP.
- It is a finite observation model.
- Developed to reason about Responsiveness and Stuck-freeness.
- A process $Q$ is responsive to process $P$ if process $Q$ will not cause process $P$ to deadlock by not responding when expected by $P$.
- A network is stuck if network of process does not terminate leaving one partner hanging.
- The model $\mathcal{T}$, the model $\mathcal{F}$, and the model $\mathcal{R}$ are successively more refined.
- There does not exist any model that refines the model $\mathcal{R}$ and is more abstract than the stable acceptance model $\mathcal{A}$ and the refusal testing model $\mathcal{RT}$. 
The stable revivals model

- The stable revivals model $R$ (Roscoe 2005, revised 2007) is a new semantic model of CSP.
- It is a finite observation model.
- Developed to reason about Responsiveness and Stuck-freeness.
- A process $Q$ is *responsive* to process $P$ if process $Q$ will not cause process $P$ to deadlock by not responding when expected by $P$.
- A network is *stuck* if network of process does not terminate leaving one partner hanging.
- The model $T$, the model $F$, and the model $R$ are successively more refined.
- There does not exist any model that refines the model $R$ and is more abstract than the stable acceptance model $\mathcal{A}$ and the refusal testing model $RT$. 

Theorem Proving for Process Algebra

The stable revivals model

Semantics of the stable revivals model
The Domain of the stable revivals model
CSP Prover
Syntax of CSP-Prover
Implementing the stable revivals model
Steps in Implementing the model
Type correctness
Recursive processes
Validation of algebraic laws
Conclusion
The stable revivals model

- The stable revivals model \( \mathcal{R} \) (Roscoe 2005, revised 2007) is a new semantic model of CSP.
- It is a finite observation model.
- Developed to reason about Responsiveness and Stuck-freeness.
- A process \( Q \) is **responsive** to process \( P \) if process \( Q \) will not cause process \( P \) to deadlock by not responding when expected by \( P \).
- A network is **stuck** if network of process does not terminate leaving one partner hanging.
- The model \( \mathcal{T} \), the model \( \mathcal{F} \), and the model \( \mathcal{R} \) are successively more refined.
- There does not exist any model that refines the model \( \mathcal{R} \) and is more abstract than the stable acceptance model \( \mathcal{A} \) and the refusal testing model \( \mathcal{RT} \).
The stable revivals model

- The stable revivals model $R$ (Roscoe 2005, revised 2007) is a new semantic model of CSP.
- It is a finite observation model.
- Developed to reason about Responsiveness and Stuck-freeness.
- A process $Q$ is **responsive** to process $P$ if process $Q$ will not cause process $P$ to deadlock by not responding when expected by $P$.
- A network is **stuck** if network of process does not terminate leaving one partner hanging.
- The model $T$, the model $F$, and the model $R$ are successively more refined.
- There does not exist any model that refines the model $R$ and is more abstract than the stable acceptance model $A$ and the refusal testing model $RT$. 
The stable revivals model

- The stable revivals model \( \mathcal{R} \) (Roscoe 2005, revised 2007) is a new semantic model of CSP.
- It is a finite observation model.
- Developed to reason about Responsiveness and Stuck-freeness.
- A process \( Q \) is **responsive** to process \( P \) if process \( Q \) will not cause process \( P \) to deadlock by not responding when expected by \( P \).
- A network is **stuck** if network of process does not terminate leaving one partner hanging.
- The model \( \mathcal{T} \), the model \( \mathcal{F} \), and the model \( \mathcal{R} \) are successively more refined.
- There does not exist any model that refines the model \( \mathcal{R} \) and is more abstract than the stable acceptance model \( \mathcal{A} \) and the refusal testing model \( \mathcal{RT} \).
The stable revivals model

- The stable revivals model $\mathcal{R}$ (Roscoe 2005, revised 2007) is a new semantic model of CSP.
- It is a finite observation model.
- Developed to reason about Responsiveness and Stuck-freeness.
- A process $Q$ is responsive to process $P$ if process $Q$ will not cause process $P$ to deadlock by not responding when expected by $P$.
- A network is stuck if network of process does not terminate leaving one partner hanging.
- The model $\mathcal{T}$, the model $\mathcal{F}$, and the model $\mathcal{R}$ are successively more refined.
- There does not exist any model that refines the model $\mathcal{R}$ and is more abstract than the stable acceptance model $\mathcal{A}$ and the refusal testing model $\mathcal{RT}$. 
Semantics of the stable revivals model

Stable revivals semantics assigns meaning for each process $P$ in terms of

$$(\text{traces}(P), \text{deadlock}(P), \text{revivals}(P))$$

Given an alphabet $\Sigma$:

- $\sigma \in \text{traces}(P) \subseteq \Sigma^*$
  $P$ can perform the finite sequence $\sigma$.
- $\sigma \in \text{deadlock}(P) \subseteq \Sigma^*$
  $P$ can deadlock after $\sigma$.
- $(\sigma, X, a) \in \text{revivals}(P) \subseteq (\Sigma^* \times \mathcal{P}(\Sigma) \times \Sigma)$
  $P$ can execute $\sigma$, stably refuse $X$, and then perform $a$.  

Semantics of the stable revivals model

Stable revivals semantics assigns meaning for each process $P$ in terms of

$$(\text{traces}(P), \text{deadlock}(P), \text{revivals}(P))$$

Given an alphabet $\Sigma$:

- $\sigma \in \text{traces}(P) \subseteq \Sigma^*$
  $P$ can perform the finite sequence $\sigma$.
- $\sigma \in \text{deadlock}(P) \subseteq \Sigma^*$
  $P$ can deadlock after $\sigma$.
- $(\sigma, X, a) \in \text{revivals}(P) \subseteq (\Sigma^* \times \mathcal{P}(\Sigma) \times \Sigma)$
  $P$ can execute $\sigma$, stably refuse $X$, and then perform $a$.  

Semantics of the stable revivals model

Stable revivals semantics assigns meaning for each process $P$ in terms of

$$(\text{traces}(P), \text{deadlock}(P), \text{revivals}(P))$$

Given an alphabet $\Sigma$:

- $\sigma \in \text{traces}(P) \subseteq \Sigma^* \forall$
  $P$ can perform the finite sequence $\sigma$.
- $\sigma \in \text{deadlock}(P) \subseteq \Sigma^*$
  $P$ can deadlock after $\sigma$.
- $(\sigma, X, a) \in \text{revivals}(P) \subseteq (\Sigma^* \times \mathcal{P}(\Sigma) \times \Sigma)$
  $P$ can execute $\sigma$, stably refuse $X$, and then perform $a$. 
Semantics of the stable revivals model

Stable revivals semantics assigns meaning for each process $P$ in terms of

$$(\text{traces}(P), \text{deadlock}(P), \text{revivals}(P))$$

Given an alphabet $\Sigma$:

- $\sigma \in \text{traces}(P) \subseteq \Sigma^*\sqrt{\text{v}}$
  - $P$ can perform the finite sequence $\sigma$.

- $\sigma \in \text{deadlock}(P) \subseteq \Sigma^*$
  - $P$ can deadlock after $\sigma$.

- $(\sigma, X, a) \in \text{revivals}(P) \subseteq (\Sigma^* \times \mathcal{P}(\Sigma) \times \Sigma)$
  - $P$ can execute $\sigma$, stably refuse $X$, and then perform $a$. 
Semantics of the stable revivals model

Stable revivals semantics assigns meaning for each process \( P \) in terms of

\[
\text{traces}(P), \text{deadlock}(P), \text{revivals}(P)
\]

Given an alphabet \( \Sigma \):

- \( \sigma \in \text{traces}(P) \subseteq \Sigma^* \sqrt{\text{a}} \)
  
  \( P \) can perform the finite sequence \( \sigma \).

- \( \sigma \in \text{deadlock}(P) \subseteq \Sigma^* \)
  
  \( P \) can deadlock after \( \sigma \).

- \( (\sigma, X, a) \in \text{revivals}(P) \subseteq (\Sigma^* \times \mathcal{P}(\Sigma) \times \Sigma) \)
  
  \( P \) can execute \( \sigma \), stably refuse \( X \), and then perform \( a \).
Semantics of the stable revivals model

Stable revivals semantics assigns meaning for each process $P$ in terms of

$$(\text{traces}(P), \text{deadlock}(P), \text{revivals}(P))$$

Given an alphabet $\Sigma$:

- $\sigma \in \text{traces}(P) \subseteq \Sigma^* \sqrt{}$
  $P$ can perform the finite sequence $\sigma$.

- $\sigma \in \text{deadlock}(P) \subseteq \Sigma^*$
  $P$ can deadlock after $\sigma$.

- $($σ, X, a$) \in \text{revivals}(P) \subseteq (\Sigma^* \times \mathcal{P}(\Sigma) \times \Sigma)$
  $P$ can execute $\sigma$, stably refuse $X$, and then perform $a$. 
Semantics of the stable revivals model

Stable revivals semantics assigns meaning for each process $P$ in terms of

$$(\text{traces}(P), \text{deadlock}(P), \text{revivals}(P))$$

Given an alphabet $\Sigma$:

- $\sigma \in \text{traces}(P) \subseteq \Sigma^* \sqrt{\Box}$
  
  $P$ can perform the finite sequence $\sigma$.

- $\sigma \in \text{deadlock}(P) \subseteq \Sigma^*$
  
  $P$ can deadlock after $\sigma$.

- $(\sigma, X, a) \in \text{revivals}(P) \subseteq (\Sigma^* \times \mathcal{P}(\Sigma) \times \Sigma)$
  
  $P$ can execute $\sigma$, stably refuse $X$, and then perform $a$. 
Domain of the stable revivals model

- Healthiness conditions on \((T, D, R)\):
  - \(T1\): \(T\) is nonempty and prefix-closed.
  - \(D1\): \(D \subseteq T\).
  - \(R1\): \((s, X, a) \in R \Rightarrow s \wedge \langle a \rangle \in T\).
  - \(R2\): \((s, X, a) \in R \wedge Y \subseteq X \Rightarrow (s, Y, a) \in R\).
  - \(R3\): \((s, X, a) \in R \wedge b \in \Sigma \Rightarrow ((s, X, b) \in R \vee (s, X \cup \{b\}, a) \in R)\).
  - \(RRS\): \((s, X, a) \in R \Rightarrow a \notin X\).
  - \(R3'\): \((s, X, a) \in R \wedge Y \subseteq \Sigma \wedge (\forall b \in Y. (s, X, b) \notin R) \Rightarrow (s, X \cup Y, a) \in R\).
  - \(\text{dom}R^\text{fin}_\Sigma = \{ (T, D, R) \mid T1, D1, R1, R2, R3, RRS \}\), \(\Sigma\) is finite.
  - \(\text{dom}R^\text{arb}_\Sigma = \{ (T, D, R) \mid T1, D1, R1, R2, R3, RRS \}\), \(\Sigma\) is arbitrary.
  - \(\text{dom}R^\text{m}_\Sigma = \{ (T, D, R) \mid T1, D1, R1, R2, R3', RRS \}\), \(\Sigma\) is arbitrary.
  - As \(R3'\) implies \(R3\): \(\text{dom}R^\text{m}_\Sigma \subseteq \text{dom}R^\text{m}_\Sigma\).
Domain of the stable revivals model

- Healthiness conditions on \((T, D, R)\):
  - \(T1\): \(T\) is nonempty and prefix-closed.
  - \(D1\): \(D \subseteq T\).
  - \(R1\): \((s, X, a) \in R \Rightarrow s \cap \langle a \rangle \in T\).
  - \(R2\): \((s, X, a) \in R \land Y \subseteq X \Rightarrow (s, Y, a) \in R\).
  - \(R3\): \((s, X, a) \in R \land b \in \Sigma \)
    \(\Rightarrow (s, X, b) \notin R\).
  - \(RRS\): \((s, X, a) \in R \Rightarrow a \notin X\).
  - \(R3':\) \((s, X, a) \in R \land Y \subseteq \Sigma \land (\forall b \in Y. (s, X, b) \notin R) \)
    \(\Rightarrow (s, X \cup Y, a) \in R\).
  - \(\text{domR}_{\Sigma}^{\text{fin}} = \{(T, D, R) \mid T1, D1, R1, R2, R3, RRS\},\)
    \(\Sigma\) is finite.
  - \(\text{domR}_{\Sigma}^{\text{arb}} = \{(T, D, R) \mid T1, D1, R1, R2, R3, RRS\},\)
    \(\Sigma\) is arbitrary.
  - \(\text{domR}_{\Sigma}^{m} = \{(T, D, R) \mid T1, D1, R1, R2, R3', RRS\},\)
    \(\Sigma\) is arbitrary.
  - As \(R3'\) implies \(R3\): \(\text{domR}_{\Sigma}^{'} \subseteq \text{domR}_{\Sigma}\).
Domain of the stable revivals model

- Healthiness conditions on \((T, D, R)\):
  - \(T1\): \(T\) is nonempty and prefix-closed.
  - \(D1\): \(D \subseteq T\).
  - \(R1\): \((s, X, a) \in R \Rightarrow s \triangleleft (a) \in T\).
  - \(R2\): \((s, X, a) \in R \land Y \subseteq X \Rightarrow (s, Y, a) \in R\).
  - \(R3\): \((s, X, a) \in R \land b \in \Sigma\)
    \(\Rightarrow ((s, X, b) \in R \lor (s, X \cup \{b\}, a) \in R)\).
  - \(RRS\): \((s, X, a) \in R \Rightarrow a \notin X\).

- \(R3'\): \((s, X, a) \in R \land Y \subseteq \Sigma \land (\forall b \in Y. (s, X, b) \notin R)\)
  \(\Rightarrow (s, X \cup Y, a) \in R\).

- \(domR^\text{fin}_\Sigma = \{(T, D, R) \mid T1, D1, R1, R2, R3, RRS\}\),
  \(\Sigma\) is finite.

- \(domR^\text{arb}_\Sigma = \{(T, D, R) \mid T1, D1, R1, R2, R3, RRS\}\),
  \(\Sigma\) is arbitrary.

- \(domR^m_\Sigma = \{(T, D, R) \mid T1, D1, R1, R2, R3', RRS\}\),
  \(\Sigma\) is arbitrary.

- As \(R3'\) implies \(R3\): \(domR'^\prime_\Sigma \subseteq domR_\Sigma\).
Domain of the stable revivals model

- Healthiness conditions on $(T, D, R)$:
  - $T1: T$ is nonempty and prefix-closed.
  - $D1: D \subseteq T$.
  - $R1: (s, X, a) \in R \Rightarrow s \setminus \langle a \rangle \in T$.
  - $R2: (s, X, a) \in R \land Y \subseteq X \Rightarrow (s, Y, a) \in R$.
  - $R3: (s, X, a) \in R \land b \in \Sigma$
      \[ \Rightarrow ((s, X, b) \in R \lor (s, X \cup \{b\}, a) \in R). \]
  - $RRS: (s, X, a) \in R \Rightarrow a \notin X$.

- $R3': (s, X, a) \in R \land Y \subseteq \Sigma \land (\forall b \in Y. (s, X, b) \notin R)$
  \[ \Rightarrow (s, X \cup Y, a) \in R. \]

- $\text{dom}_\Sigma^{\text{fin}} = \{(T, D, R) \mid T1, D1, R1, R2, R3, RRS\}$, $\Sigma$ is finite.

- $\text{dom}_\Sigma^{\text{arb}} = \{(T, D, R) \mid T1, D1, R1, R2, R3, RRS\}$, $\Sigma$ is arbitrary.

- $\text{dom}_\Sigma^{m} = \{(T, D, R) \mid T1, D1, R1, R2, R3', RRS\}$, $\Sigma$ is arbitrary.

- As $R3'$ implies $R3$: $\text{dom}_\Sigma'^\prime \subseteq \text{dom}_\Sigma$. 
Domain of the stable revivals model

- Healthiness conditions on \((T, D, R)\):
  - \(T1\) : \(T\) is nonempty and prefix-closed.
  - \(D1\) : \(D \subseteq T\).
  - \(R1\) : \((s, X, a) \in R \implies s \land \langle a \rangle \in T\).
  - \(R2\) : \((s, X, a) \in R \land Y \subseteq X \implies (s, Y, a) \in R\).
  - \(R3\) : \((s, X, a) \in R \land b \in \Sigma \\
  \implies ((s, X, b) \in R \lor (s, X \cup \{b\}, a) \in R)\).
  - \(RRS\) : \((s, X, a) \in R \implies a \notin X\).
  - \(R3'\) : \((s, X, a) \in R \land Y \subseteq \Sigma \land (\forall b \in Y.(s, X, b) \notin R) \\
  \implies (s, X \cup Y, a) \in R\).

- \(\text{domR}_{\Sigma}^{\text{fin}} = \{(T, D, R) \mid T1, D1, R1, R2, R3, RRS\}\), \(\Sigma\) is finite.
- \(\text{domR}_{\Sigma}^{\text{arb}} = \{(T, D, R) \mid T1, D1, R1, R2, R3, RRS\}\), \(\Sigma\) is arbitrary.
- \(\text{domR}_{\Sigma}^{m} = \{(T, D, R) \mid T1, D1, R1, R2, R3', RRS\}\), \(\Sigma\) is arbitrary.
- As \(R3'\) implies \(R3\): \(\text{domR}_{\Sigma}^{'} \subseteq \text{domR}_{\Sigma}\).
Domain of the stable revivals model

- Healthiness conditions on \((T, D, R)\):
  - **T1**: \(T\) is nonempty and prefix-closed.
  - **D1**: \(D \subseteq T\).
  - **R1**: \((s, X, a) \in R \Rightarrow s \cap \langle a \rangle \in T\).
  - **R2**: \((s, X, a) \in R \land Y \subseteq X \Rightarrow (s, Y, a) \in R\).
  - **R3**: \((s, X, a) \in R \land b \in \Sigma \Rightarrow ((s, X, b) \in R \lor (s, X \cup \{b\}, a) \in R)\).
  - **RRS**: \((s, X, a) \in R \Rightarrow a \notin X\).

- **R3'**: \((s, X, a) \in R \land Y \subseteq \Sigma \land (\forall b \in Y. (s, X, b) \notin R) \Rightarrow (s, X \cup Y, a) \in R\).

- \(\text{dom}R_{\Sigma}^{\text{fin}} = \{(T, D, R) \mid T1, D1, R1, R2, R3, RRS\}, \Sigma\) is finite.

- \(\text{dom}R_{\Sigma}^{\text{arb}} = \{(T, D, R) \mid T1, D1, R1, R2, R3, RRS\}, \Sigma\) is arbitrary.

- \(\text{dom}R_{\Sigma}^{m} = \{(T, D, R) \mid T1, D1, R1, R2, R3', RRS\}, \Sigma\) is arbitrary.

- As \(R3'\) implies \(R3\): \(\text{dom}R_{\Sigma}^{'} \subseteq \text{dom}R_{\Sigma}\).
Domain of the stable revivals model

- Healthiness conditions on \((T, D, R)\):
  - \(T1\): \(T\) is nonempty and prefix-closed.
  - \(D1\): \(D \subseteq T\).
  - \(R1\): \((s, X, a) \in R \Rightarrow s \cap \langle a \rangle \in T\).
  - \(R2\): \((s, X, a) \in R \land Y \subseteq X \Rightarrow (s, Y, a) \in R\).
  - \(R3\): \((s, X, a) \in R \land b \in \Sigma \Rightarrow ((s, X, b) \in R \lor (s, X \cup \{b\}, a) \in R)\).
  - \(RRS\): \((s, X, a) \in R \Rightarrow a \notin X\).

- \(R3'\): \((s, X, a) \in R \land Y \subseteq \Sigma \land (\forall b \in Y. (s, X, b) \notin R) \Rightarrow (s, X \cup Y, a) \in R\).

- \(\text{dom}\text{R}_{\Sigma}^{\text{fin}} = \{(T, D, R) \mid T1, D1, R1, R2, R3, RRS\}, \Sigma\) is finite.

- \(\text{dom}\text{R}_{\Sigma}^{\text{arb}} = \{(T, D, R) \mid T1, D1, R1, R2, R3, RRS\}, \Sigma\) is arbitrary.

- \(\text{dom}\text{R}_{\Sigma}^{m} = \{(T, D, R) \mid T1, D1, R1, R2, R3', RRS\}, \Sigma\) is arbitrary.

- As \(R3'\) implies \(R3\): \(\text{dom}\text{R}_{\Sigma}^{'} \subseteq \text{dom}\text{R}_{\Sigma}\).
Domain of the stable revivals model

- Healthiness conditions on \((T, D, R)\):
  - **T1**: \(T\) is nonempty and prefix-closed.
  - **D1**: \(D \subseteq T\).
  - **R1**: \((s, X, a) \in R \Rightarrow s \cap \langle a \rangle \in T\).
  - **R2**: \((s, X, a) \in R \land Y \subseteq X \Rightarrow (s, Y, a) \in R\).
  - **R3**: \((s, X, a) \in R \land b \in \Sigma \Rightarrow ((s, X, b) \in R \lor (s, X \cup \{b\}, a) \in R)\).
  - **RRS**: \((s, X, a) \in R \Rightarrow a \notin X\).

- **R3'**: \((s, X, a) \in R \land Y \subseteq \Sigma \land (\forall b \in Y. (s, X, b) \notin R) \Rightarrow (s, X \cup Y, a) \in R\).

- \(\text{dom}R_{\Sigma}^{\text{fin}} = \{(T, D, R) \mid T1, D1, R1, R2, R3, RRS\}, \Sigma\) is finite.

- \(\text{dom}R_{\Sigma}^{\text{arb}} = \{(T, D, R) \mid T1, D1, R1, R2, R3, RRS\}, \Sigma\) is arbitrary.

- \(\text{dom}R_{\Sigma}^{m} = \{(T, D, R) \mid T1, D1, R1, R2, R3', RRS\}, \Sigma\) is arbitrary.

- As **R3'** implies **R3**: \(\text{dom}R_{\Sigma}^{'} \subseteq \text{dom}R_{\Sigma}\).
Domain of the stable revivals model

- Healthiness conditions on \((T, D, R)\):
  - **T1**: \(T\) is nonempty and prefix-closed.
  - **D1**: \(D \subseteq T\).
  - **R1**: \((s, X, a) \in R \Rightarrow s \cap \langle a \rangle \in T\).
  - **R2**: \((s, X, a) \in R \land Y \subseteq X \Rightarrow (s, Y, a) \in R\).
  - **R3**: \((s, X, a) \in R \land b \in \Sigma \Rightarrow ((s, X, b) \in R \lor (s, X \cup \{b\}, a) \in R)\).
  - **RRS**: \((s, X, a) \in R \Rightarrow a \notin X\).
  - **R3′**: \((s, X, a) \in R \land Y \subseteq \Sigma \land (\forall b \in Y. (s, X, b) \notin R) \Rightarrow (s, X \cup Y, a) \in R\).

- **\(\text{domR}_\Sigma^{\text{fin}}\)**: \(\{(T, D, R) \mid T1, D1, R1, R2, R3, RRS\}\), \(\Sigma\) is finite.

- **\(\text{domR}_\Sigma^{\text{arb}}\)**: \(\{(T, D, R) \mid T1, D1, R1, R2, R3, RRS\}\), \(\Sigma\) is arbitrary.

- **\(\text{domR}_\Sigma^m\)**: \(\{(T, D, R) \mid T1, D1, R1, R2, R3′, RRS\}\), \(\Sigma\) is arbitrary.

- As \(R3′\) implies \(R3\): \(\text{domR}_\Sigma^{r} \subseteq \text{domR}_\Sigma\).
Domain of the stable revivals model

- Healthiness conditions on \((T, D, R)\):
  - \(T_1\): \(T\) is nonempty and prefix-closed.
  - \(D_1\): \(D \subseteq T\).
  - \(R_1\): \((s, X, a) \in R \Rightarrow s \preceq \langle a \rangle \in T\).
  - \(R_2\): \((s, X, a) \in R \land Y \subseteq X \Rightarrow (s, Y, a) \in R\).
  - \(R_3\): \((s, X, a) \in R \land b \in \Sigma \\
    \Rightarrow ((s, X, b) \in R \lor (s, X \cup \{b\}, a) \in R)\).
  - \(RRS\): \((s, X, a) \in R \Rightarrow a \notin X\).

- \(R3'\): \((s, X, a) \in R \land Y \subseteq \Sigma \land (\forall b \in Y. (s, X, b) \notin R) \\
  \Rightarrow (s, X \cup Y, a) \in R\).

- \(\text{domR}_{\Sigma}^{\text{fin}} = \{(T, D, R) \mid T_1, D_1, R_1, R_2, R_3, RRS\}, \Sigma \) is finite.

- \(\text{domR}_{\Sigma}^{\text{arb}} = \{(T, D, R) \mid T_1, D_1, R_1, R_2, R_3, RRS\}, \Sigma \) is arbitrary.

- \(\text{domR}_{\Sigma}^{m} = \{(T, D, R) \mid T_1, D_1, R_1, R_2, R_3', RRS\}, \Sigma \) is arbitrary.

- As \(R3'\) implies \(R3\): \(\text{domR}_{\Sigma}' \subseteq \text{domR}_{\Sigma}\).
Domain of the stable revivals model

- Healthiness conditions on \((T, D, R)\):
  - **T1**: \(T\) is nonempty and prefix-closed.
  - **D1**: \(D \subseteq T\).
  - **R1**: \((s, X, a) \in R \Rightarrow s \smallsetminus \langle a \rangle \in T\).
  - **R2**: \((s, X, a) \in R \land Y \subseteq X \Rightarrow (s, Y, a) \in R\).
  - **R3**: \((s, X, a) \in R \land b \in \Sigma \Rightarrow ((s, X, b) \in R \lor (s, X \cup \{b\}, a) \in R)\).
  - **RRS**: \((s, X, a) \in R \Rightarrow a \notin X\).

- **R3'**: \((s, X, a) \in R \land Y \subseteq \Sigma \land (\forall b \in Y. (s, X, b) \notin R) \Rightarrow (s, X \cup Y, a) \in R\).

- \(\text{dom}R^\text{fin}_\Sigma = \{(T, D, R) \mid T1, D1, R1, R2, R3, RRS\}, \Sigma \text{ is finite.}\)

- \(\text{dom}R^\text{arb}_\Sigma = \{(T, D, R) \mid T1, D1, R1, R2, R3, RRS\}, \Sigma \text{ is arbitrary.}\)

- \(\text{dom}R^m_\Sigma = \{(T, D, R) \mid T1, D1, R1, R2, R3', RRS\}, \Sigma \text{ is arbitrary.}\)

- As **R3'** implies **R3**: \(\text{dom}R'_\Sigma \subseteq \text{dom}R_\Sigma\).
Domain of the stable revivals model

- Healthiness conditions on \((T, D, R)\):
  - \(T1\) : \(T\) is nonempty and prefix-closed.
  - \(D1\) : \(D \subseteq T\).
  - \(R1\) : \((s, X, a) \in R \Rightarrow s \sqcup \langle a \rangle \in T\).
  - \(R2\) : \((s, X, a) \in R \land Y \subseteq X \Rightarrow (s, Y, a) \in R\).
  - \(R3\) : \((s, X, a) \in R \land b \in \Sigma \Rightarrow ((s, X, b) \in R \lor (s, X \cup \{b\}, a) \in R)\).
  - \(RRS\) : \((s, X, a) \in R \Rightarrow a \notin X\).

- \(R3'\) : \((s, X, a) \in R \land Y \subseteq \Sigma \land (\forall b \in Y. (s, X, b) \notin R) \Rightarrow (s, X \cup Y, a) \in R\).

- \(\text{dom}R_{\Sigma}^{\text{fin}} = \{(T, D, R) \mid T1, D1, R1, R2, R3, RRS\}, \Sigma\) is finite.

- \(\text{dom}R_{\Sigma}^{\text{arb}} = \{(T, D, R) \mid T1, D1, R1, R2, R3, RRS\}, \Sigma\) is arbitrary.

- \(\text{dom}R_{\Sigma}^{m} = \{(T, D, R) \mid T1, D1, R1, R2, R3', RRS\}, \Sigma\) is arbitrary.

- As \(R3'\) implies \(R3\): \(\text{dom}R_{\Sigma}^{\text{fin}} \subseteq \text{dom}R_{\Sigma}\).
The Stable Revivals Model in CSP-Prover

D G Samuel (Swansea), Y Isobe (AIST, Japan), M Roggenbach (Swansea)

Theorem Proving for Process Algebra

The stable revivals model
  Semantics of the stable revivals model
  The Domain of the stable revivals model

CSP Prover
  Syntax of CSP-Prover

Implementing the stable revivals model
  Steps in Implementing the model
  Type correctness
  Recursive processes

Validation of algebraic laws

Conclusion

CSP-Prover
CSP-Prover is a proof infrastructure to prove CSP process refinement and equality proofs using Isabelle/HOL.

- Developed by Yoshinao Isobe (AIST, Japan) and Markus Roggenbach (Swansea University, UK)
- Proofs on infinite state systems, which may also have infinite non-determinism.
- Currently fully supports the traces model and the stable failures model.
CSP-Prover

- CSP-Prover is a proof infrastructure to prove CSP process refinement and equality proofs using Isabelle/HOL.
- Developed by Yoshinao Isobe (AIST, Japan) and Markus Roggenbach (Swansea University, UK)
- Proofs on infinite state systems, which may also have infinite non-determinism.
- Currently fully supports the traces model and the stable failures model.
CSP-Prover

- CSP-Prover is a proof infrastructure to prove CSP process refinement and equality proofs using Isabelle/HOL.
- Developed by Yoshinao Isobe (AIST, Japan) and Markus Roggenbach (Swansea University, UK)
- Proofs on infinite state systems, which may also have infinite non-determinism.
- Currently fully supports the traces model and the stable failures model.
CSP-Prover

- CSP-Prover is a proof infrastructure to prove CSP process refinement and equality proofs using Isabelle/HOL.
- Developed by Yoshinao Isobe (AIST, Japan) and Markus Roggenbach (Swansea University, UK)
- Proofs on infinite state systems, which may also have infinite non-determinism.
- Currently fully supports the traces model and the stable failures model.
CSP-Prover with the stable revivals model

Two major parts: Reusable parts and Instantiated parts.

- The reusable part is independent of specific CSP models.
- Instantiated parts are built on the top of reusable part.
CSP-Prover with the stable revivals model

- Two major parts: Reusable parts and Instantiated parts.
- The reusable part is independent of specific CSP models.
- Instantiated parts are built on the top of reusable part.
Two major parts: Reusable parts and Instantiated parts.

- The reusable part is independent of specific CSP models.
- Instantiated parts are built on the top of reusable part.
### Syntax of CSP-Prover: $CSP_{TP}$

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P ::= Skip$</td>
<td>%% terminating process</td>
</tr>
<tr>
<td>$</td>
<td>Stop$</td>
</tr>
<tr>
<td>$</td>
<td>Div$</td>
</tr>
<tr>
<td>$</td>
<td>a \rightarrow P$</td>
</tr>
<tr>
<td>$</td>
<td>? x : A \rightarrow P(x)$</td>
</tr>
<tr>
<td>$</td>
<td>P \sqcap P$</td>
</tr>
<tr>
<td>$</td>
<td>P \sqcap P$</td>
</tr>
<tr>
<td>$</td>
<td>!! c : C \bullet P(c)$</td>
</tr>
<tr>
<td>$</td>
<td>if b \ then \ P \ else \ P$</td>
</tr>
<tr>
<td>$</td>
<td>P \parallel [X] \parallel P$</td>
</tr>
<tr>
<td>$</td>
<td>P \setminus X$</td>
</tr>
<tr>
<td>$</td>
<td>P[[r]]$</td>
</tr>
<tr>
<td>$</td>
<td>P \odot P$</td>
</tr>
<tr>
<td>$</td>
<td>P \downarrow n$</td>
</tr>
<tr>
<td>$</td>
<td>$P$</td>
</tr>
</tbody>
</table>

where $A, X \subseteq \Sigma$, $C \subseteq \mathcal{P}(\mathcal{P}(\Sigma)) \cup \mathcal{P}(\text{Nat})$, $\cup$ is a disjoint union of two sets, $b$ is a condition, $r \in \mathcal{P}(\Sigma \times \Sigma)$, and $n \in \text{Nat}$. 
Implementing the stable revivals model
Implementation Steps

- Create a new type to represent the domain of the model.
- Prove that this domain is a complete partial order.
- Encode the semantic functions of the model.
- Prove type correctness (well definedness) of these semantic functions.
- Prove that these semantic functions are continuous.
- Provide proof infrastructure for recursive processes.
Implementation Steps

- Create a new type to represent the domain of the model.
- Prove that this domain is a complete partial order.
- Encode the semantic functions of the model.
- Prove type correctness (well definedness) of these semantic functions.
- Prove that these semantic functions are continuous.
- Provide proof infrastructure for recursive processes.
Implementation Steps

- Create a new type to represent the domain of the model.
- Prove that this domain is a complete partial order.
- Encode the semantic functions of the model.
- Prove type correctness (well definedness) of these semantic functions.
- Prove that these semantic functions are continuous.
- Provide proof infrastructure for recursive processes.
Implementation Steps

- Create a new type to represent the domain of the model.
- Prove that this domain is a complete partial order.
- Encode the semantic functions of the model.
- Prove type correctness (well definedness) of these semantic functions.
- Prove that these semantic functions are continuous.
- Provide proof infrastructure for recursive processes.
Implementation Steps

- Create a new type to represent the domain of the model.
- Prove that this domain is a complete partial order.
- Encode the semantic functions of the model.
- Prove type correctness (well definedness) of these semantic functions.
- Prove that these semantic functions are continuous.
- Provide proof infrastructure for recursive processes.
Implementation Steps

- Create a new type to represent the domain of the model.
- Prove that this domain is a complete partial order.
- Encode the semantic functions of the model.
- Prove type correctness (well definedness) of these semantic functions.
- Prove that these semantic functions are continuous.
- Provide proof infrastructure for recursive processes.
Create a new type for the domain of the model

- Create a type to represent the revivals components:
  ```
  types
  'a revival = "'a trace * 'a event set * 'a event"
  typedef 'a setR = "{ R :: ('a revivals set). HC_RT(R) & HC_RRS(R) & HC_R2(R) & HC_R3(R) }"
  ```

- Define the Cartesian product:
  ```
  types 'a domTsetDsetR =
  "('a domT * 'a setD * 'a setR)"
  ```

- Restrict it to 'healthy' elements:
  ```
  typedef 'a domR = "{ X :: ('a domTsetDsetR). HC_D1(X) & HC_R1(X) }"
  ```

- Prove this restriction to be non-empty:
  ```
  apply (rule_tac x = "({<>}t , {}d, {}r)" in exI)
  ```
Create a new type for the domain of the model

- Create a type to represent the revivals components:
  types
  'a revival = "('a trace * 'a event set * 'a event)"
  typedef 'a setR = "{ R ::('a revivals set). HC_RT(R) & HC_RRS(R) & HC_R2(R) & HC_R3(R) }"

- Define the Cartesian product:
  types 'a domTsetDsetR = "('a domT * 'a setD * 'a setR)"

- Restrict it to ‘healthy’ elements:
  typedef 'a domR = "{ X ::('a domTsetDsetR). HC_D1(X) & HC_R1(X) }"

- Prove this restriction to be non-empty:
  apply (rule_tac x = "({<>})t , {}d , {}r" in exI)
Create a new type for the domain of the model

- Create a type to represent the revivals components:
  types
  'a revival = "(’a trace * ’a event set * ’a event)"
  typedef 'a setR = "{ R ::(’a revivals set). HC_RT(R) & HC_RRS(R) & HC_R2(R) & HC_R3(R) }"

- Define the Cartesian product:
  types 'a domTsetDsetR =
  "(’a domT * ’a setD * ’a setR)"

- Restrict it to ‘healthy’ elements:
  typedef 'a domR = "{ X ::(’a domTsetDsetR). HC_D1(X) & HC_R1(X) }"

- Prove this restriction to be non-empty:
  apply (rule_tac x = "({<>}t , {}d, {}r)" in exI)
Create a new type for the domain of the model

- Create a type to represent the revivals components:
  ```
  types
  'a revival = "('a trace * 'a event set * 'a event)"
  typedef 'a setR = "{ R ::('a revivals set). HC_RT(R) & HC_RRS(R) & HC_R2(R) & HC_R3(R) }"
  ```

- Define the Cartesian product:
  ```
  types 'a domTsetDsetR = "('a domT * 'a setD * 'a setR)"
  ```

- Restrict it to ‘healthy’ elements:
  ```
  typedef 'a domR = "{ X ::('a domTsetDsetR). HC_D1(X) & HC_R1(X) }"
  ```

- Prove this restriction to be non-empty:
  ```
  apply (rule_tac x = "({<>}t , {}d, {}r)" in exI)
  ```
Create a new type for the domain of the model

- Create a type to represent the revivals components:

  ```ml
  'a revival = "('a trace * 'a event set * 'a event)"
  typedef 'a setR = "{ R ::('a revivals set). HC_RT(R) & HC_RRS(R) & HC_R2(R) & HC_R3(R) }"
  ```

- Define the Cartesian product:

  ```ml
  types 'a domTsetDsetR = "('a domT * 'a setD * 'a setR)"
  ```

- Restrict it to ‘healthy’ elements:

  ```ml
  typedef 'a domR = "{ X ::('a domTsetDsetR). HC_D1(X) & HC_R1(X) }"
  ```

- Prove this restriction to be non-empty:

  ```ml
  apply (rule_tac x = "({<>}t , {}d, {}r)" in exI)
  ```
Create a new type for the domain of the model

- Create a type to represent the revivals components:
  ```
  types
  'a revival = "('a trace * 'a event set * 'a event)"
  typedef 'a setR = "{ R ::('a revivals set). HC_RT(R) & HC_RRS(R) & HC_R2(R) & HC_R3(R) }"
  ```

- Define the Cartesian product:
  ```
  types 'a domTsetDsetR =
  "('a domT * 'a setD * 'a setR)"
  ```

- Restrict it to ‘healthy’ elements:
  ```
  typedef 'a domR = "{ X ::('a domTsetDsetR). HC_D1(X) & HC_R1(X) }"
  ```

- Prove this restriction to be non-empty:
  ```
  apply (rule_tac x = "({<>}t , {}d, {}r)" in exI)
  ```
Prove the domain to be a CPO

- cpo – axiomatic class provided by CSP-Prover
  
  ```
  instance domR :: (type) cpo
  apply (intro_classes)
  ```

- we prove
  - UnionR Rs is an upper bound of set Rs.
    ```
    lemma UnionR_isUB : "(UnionR Fs) isUB Fs"
    ```
  - UnionR Rs is the least upper bound of set Rs.
    ```
    lemma UnionR_isLUB : "UnionR Fs isLUB Fs"
    ```
  - The least upper bound of Fs is UnionR Rs.
    ```
    lemma isLUB_UnionR_only_if: 
    "F isLUB Fs ==> F = UnionR Fs"
    ```

- Encode the semantic function of the model.
The Stable Revivals Model in CSP-Prover

D G Samuel (Swansea), Y Isobe (AIST, Japan), M Roggenbach (Swansea)

Theorem Proving for Process Algebra

The stable revivals model
Semantics of the stable revivals model
The Domain of the stable revivals model
CSP Prover
Syntax of CSP-Prover
Implementing the stable revivals model
Steps in Implementing the model
Type correctness
Recursive processes
Validation of algebraic laws
Conclusion

Prove the domain to be a CPO

- cpo – axiomatic class provided by CSP-Prover
  instance domR :: (type) cpo
  apply (intro_classes)

- we prove
  - UnionR Rs is an upper bound of set Rs.
    lemma UnionR_isUB : "(UnionR Fs) isUB Fs"
  - UnionR Rs is the least upper bound of set Rs.
    lemma UnionR_isLUB : "UnionR Fs isLUB Fs"
  - The least upper bound of Fs is UnionR Rs.
    lemma isLUB_UnionR_only_if:
      "F isLUB Fs ==> F = UnionR Fs"

- Encode the semantic function of the model.
Prove the domain to be a CPO

- cpo – axiomatic class provided by CSP-Prover
  
  instance domR :: (type) cpo
  apply (intro_classes)

- we prove
  
  - UnionR Rs is an upper bound of set Rs.
    lemma UnionR_isUB : "(UnionR Fs) isUB Fs"
  
  - UnionR Rs is the least upper bound of set Rs.
    lemma UnionR_isLUB : "UnionR Fs isLUB Fs"

  - The least upper bound of Fs is UnionR Rs.
    lemma isLUB_UnionR_only_if:
      "F isLUB Fs ==> F = UnionR Fs"

- Encode the semantic function of the model.
Prove the domain to be a CPO

- cpo – axiomatic class provided by Csp-Prover
  
  instance domR :: (type) cpo
  apply (intro_classes)

- we prove
  - UnionR Rs is an upper bound of set Rs.
    lemma UnionR_isUB : "(UnionR Fs) isUB Fs"
  - UnionR Rs is the least upper bound of set Rs.
    lemma UnionR_isLUB : "UnionR Fs isLUB Fs"
  - The least upper bound of Fs is UnionR Rs.
    lemma isLUB_UnionR_only_if:
      "F isLUB Fs ==> F = UnionR Fs"

- Encode the semantic function of the model.
Prove the domain to be a CPO

- cpo – axiomatic class provided by Csp-Prover
- instance domR :: (type) cpo
  apply (intro_classes)
- we prove
  - UnionR Rs is an upper bound of set Rs.
    lemma UnionR_isUB : "(UnionR Fs) isUB Fs"
  - UnionR Rs is the least upper bound of set Rs.
    lemma UnionR_isLUB : "UnionR Fs isLUB Fs"
  - The least upper bound of Fs is UnionR Rs.
    lemma isLUB_UnionR_only_if: "F isLUB Fs ==> F = UnionR Fs"
- Encode the semantic function of the model.
Type correctness

- Type correctness for the prefix choice operator

- Type correctness for the prefix choice operator

\[ T(a \rightarrow P) = \{\langle \rangle\} \cup \{\langle a \rangle \triangle s \mid s \in T(P)\} \]

\[ D(a \rightarrow P) = \{\langle a \rangle \triangle s \mid s \in D(P)\} \]

\[ R(a \rightarrow P) = \{\langle \langle\rangle, X, a \rangle \mid a \notin X\} \]

\[ \cup \{\langle a \rangle \triangle s, X, b \rangle \mid (s, X, b) \in R(P)\} \]
Type correctness

- Type correctness for the prefix choice operator
  
  \[ T(a \to P) = \{\langle \rangle\} \cup \{\langle a \rangle \triangleleft s \mid s \in T(P)\} \]
  
  \[ D(a \to P) = \{\langle a \rangle \triangleleft s \mid s \in D(P)\} \]
  
  \[ R(a \to P) = \{(\langle \rangle, X, a) \mid a \notin X\} \]
  
  \[ \cup \{(\langle a \rangle \triangleleft s, X, b) \mid (s, X, b) \in R(P)\} \]}
Type correctness

- Type correctness for the prefix choice operator
  \[ T(a \rightarrow P) = \{\langle\rangle\} \cup \{\langle a \rangle \triangleright s \mid s \in T(P)\} \]
  \[ D(a \rightarrow P) = \{\langle a \rangle \triangleright s \mid s \in D(P)\} \]
  \[ R(a \rightarrow P) = \{(\langle\rangle, X, a) \mid a \not\in X\} \]
  \[ \cup \{(\langle a \rangle \triangleright s, X, b) \mid (s, X, b) \in R(P)\} \]
Type correctness

- Type correctness for the prefix choice operator
  \[ T(a \rightarrow P) = \{ \langle \rangle \} \cup \{ \langle a \rangle \triangleright s \mid s \in T(P) \} \]
  \[ D(a \rightarrow P) = \{ \langle a \rangle \triangleright s \mid s \in D(P) \} \]
  \[ R(a \rightarrow P) = \{ \langle \langle \rangle, X, a \rangle \mid a \notin X \} \]
  \[ \cup \{ \langle a \rangle \triangleright s, X, b \mid (s, X, b) \in R(P) \} \]
Type correctness

- Type correctness for the prefix choice operator
  
  \[
  T(a \rightarrow P) = \{\langle \rangle \} \cup \{\langle a \rangle \uparrow s \mid s \in T(P)\} \\
  D(a \rightarrow P) = \{\langle a \rangle \uparrow s \mid s \in D(P)\} \setminus \{\langle a \rangle \uparrow s \mid s \in R(P)\} \\
  R(a \rightarrow P) = \{((\langle \rangle, X, a) \mid a \not\in X\} \\
  \cup \{(\langle a \rangle \uparrow s, X, b) \mid (s, X, b) \in R(P)\}
  \]

D G Samuel (Swansea),
Y Isobe (AIST, Japan),
M Roggenbach (Swansea)
Type correctness of renaming

- $domR_{\Sigma}^{\text{fin}}$: renaming is type correct (Roscoe’s setting!).
- $domR_{\Sigma}^{\text{arb}}$: renaming fails to be type correct.

Counter Example:

- $\Sigma = \mathcal{N} \cup \{a, b\}$.
- $C = (\{\langle\rangle, \langle a\rangle, \langle b\rangle\},$
  
- $\{\},
  
- $\{((\langle\rangle, X, a), (\langle\rangle, X, b) | X \in \mathcal{P}_{\text{fin}}(\mathcal{N})\})$
  
where $\mathcal{P}_{\text{fin}}(\mathcal{N})$ is a set of all finite sets of $\mathcal{N}$.
- $Rel = \{(a, a)\} \cup \{(n, b) | n \in \mathcal{N}\}$

- $domR_{\Sigma}^{m}$: renaming is type correct.
Type correctness of renaming

- $domR_{\Sigma}^{\text{fin}}$: renaming is type correct (Roscoe’s setting!).
- $domR_{\Sigma}^{\text{arb}}$: renaming fails to be type correct.

Counter Example:

- $\Sigma = \mathcal{N} \cup \{a, b\}$.
- $C = (\{\langle \rangle, \langle a \rangle, \langle b \rangle\},$
  
  $\{\},$
  
  $\{(\langle \rangle, X, a), (\langle \rangle, X, b) \mid X \in P_{\text{fin}}(\mathcal{N})\})$

  where $P_{\text{fin}}(\mathcal{N})$ is a set of all finite sets of $\mathcal{N}$.
- $Rel = \{(a, a)\} \cup \{(n, b) \mid n \in N\}$

- $domR_{\Sigma}^{m}$: renaming is type correct.
Type correctness of renaming

- $\text{domR}^\text{fin}_\Sigma$: renaming is type correct (Roscoe’s setting!).

- $\text{domR}^\text{arb}_\Sigma$: renaming fails to be type correct.

Counter Example:

- $\Sigma = \mathcal{N} \cup \{a, b\}$.
- $C = (\{\langle \rangle, \langle a \rangle, \langle b \rangle\},$
  \{\},
  \{\langle \langle \rangle \rangle, X, a\}, \langle \langle \rangle \rangle, X, b\mid X \in P_{\text{fin}}(\mathcal{N})\})$
  where $P_{\text{fin}}(\mathcal{N})$ is a set of all finite sets of $\mathcal{N}$.

- $\text{Rel} = \{(a, a)\} \cup \{(n, b) \mid n \in N\}$

- $\text{domR}^m_\Sigma$: renaming is type correct.
Type correctness of renaming

- \( \text{domR}_{\Sigma}^{\text{fin}} \): renaming is type correct (Roscoe’s setting!).
- \( \text{domR}_{\Sigma}^{\text{arb}} \): renaming fails to be type correct.

**Counter Example:**

- \( \Sigma = \mathcal{N} \cup \{a, b\} \).
- \( C = \{\mathcal{O}, \{a\}, \{b\}\}, \),
  \( \{\}, \),
  \( \{((\mathcal{O}, X, a), (\mathcal{O}, X, b) \mid X \in \mathcal{P}_{\text{fin}}(\mathcal{N})\}\} \)

where \( \mathcal{P}_{\text{fin}}(\mathcal{N}) \) is a set of all finite sets of \( \mathcal{N} \).
- \( \text{Rel} = \{(a, a)\} \cup \{(n, b) \mid n \in \mathbb{N}\} \)

- \( \text{domR}_{\Sigma}^m \): renaming is type correct.
Type correctness of renaming

- \( \text{dom}R^\text{fin}_\Sigma \): renaming is type correct (Roscoe’s setting!).
- \( \text{dom}R^\text{arb}_\Sigma \): renaming fails to be type correct.

Counter Example:
- \( \Sigma = \mathcal{N} \cup \{a, b\} \).
- \( C = \{\langle \rangle, \langle a \rangle, \langle b \rangle\}, \{\}, \{((\langle \rangle, X, a), (\langle \rangle, X, b) | X \in P_{\text{fin}}(\mathcal{N})\}\} \)
  where \( P_{\text{fin}}(\mathcal{N}) \) is a set of all finite sets of \( \mathcal{N} \).
- \( \text{Rel} = \{(a, a)\} \cup \{(n, b) | n \in N\} \)
- \( \text{dom}R^m_\Sigma \): renaming is type correct.
The Stable Revivals Model in CSP-Prover
D G Samuel (Swansea), Y Isobe (AIST, Japan), M Roggenbach (Swansea)

Theorem Proving for Process Algebra
The stable revivals model
Semantics of the stable revivals model
The Domain of the stable revivals model
CSP Prover
Syntax of CSP-Prover
Implementing the stable revivals model
Steps in Implementing the model
Type correctness
Recursive processes
Validation of algebraic laws
Conclusion

Recursive processes: Continuity

- For each process name \( N \in \Pi \) (a set of all process names), a process equation is defined:
  \[ N(x_1, x_2, \ldots x_n) = P \]
  where \( P \in \text{Proc}(\Pi, \Sigma) \), the process name \( N \) behaves like the process \( P \).

- A special function \( \text{PNfun}_\Pi : \Pi \rightarrow \text{Proc}(\Pi, \Sigma) \), which is called a \textit{process-name function}, in order to describe the right hand sides of defining equations.

- \( \left[ [\text{PNfun}_\Pi]_{\text{fun}} \right]_{\mathcal{R}} \) is continuous.

- Finally, the semantics \( [P]_{\mathcal{R}} \) of each process \( P \) is defined as follows: \( [P]_{\mathcal{R}} = [P]_{\mathcal{R}(\text{MR}_\Pi)} \). Consequently,
  \[ [N]_{\mathcal{R}} = [N]_{\mathcal{R}(\text{MR}_\Pi)} \]

- the ideal interpretation, written \( \text{MR}_\Pi \), is given as follows:
  \[ \text{MR}_\Pi = \text{LFP}([\text{PNfun}_\Pi]_{\text{fun}}) \]
  where LFP represents the least fixed point.
Recursive processes: Continuity

- For each process name $N \in \Pi$ (a set of all process names), a process equation is defined:
  
  $$N(x_1, x_2, \ldots, x_n) = P$$

  where $P \in Proc(\Pi, \Sigma)$, the process name $N$ behaves like the process $P$.

- A special function $PN_{\Pi} : \Pi \rightarrow Proc(\Pi, \Sigma)$, which is called a *process-name function*, in order to describe the right hand sides of defining equations.

- $(PN_{\Pi})_{fun}$ is continuous.

- Finally, the semantics $[P]_R$ of each process $P$ is defined as follows: $[P]_R = [P]_{R(MR_\Pi)}$. Consequently,

  $$[N]_R = [N]_{R(MR_\Pi)}$$

- the ideal interpretation, written $MR_\Pi$, is given as follows:

  $$MR_\Pi = \text{LFP}(PN_{\Pi})_{fun}$$

  where LFP represents the least fixed point.
Recursive processes: Continuity

- For each process name $N \in \Pi$ (a set of all process names), a process equation is defined:

$$N(x_1, x_2, \ldots x_n) = P$$

where $P \in \text{Proc}_{(\Pi, \Sigma)}$, the process name $N$ behaves like the process $P$.

- A special function $\text{PNfun}_\Pi : \Pi \rightarrow \text{Proc}_{(\Pi, \Sigma)}$, which is called a *process-name function*, in order to describe the right hand sides of defining equations.

- $(\llbracket \text{PNfun}_\Pi \rrbracket_{f_{\text{un}}})$ is continuous.

- Finally, the semantics $\llbracket P \rrbracket_{\mathcal{R}}$ of each process $P$ is defined as follows: $\llbracket P \rrbracket_{\mathcal{R}} = \llbracket P \rrbracket_{\mathcal{R}(\text{MR}_\Pi)}$. Consequently,

$$\llbracket N \rrbracket_{\mathcal{R}} = \llbracket N \rrbracket_{\mathcal{R}(\text{MR}_\Pi)}$$

- the ideal interpretation, written $\text{MR}_\Pi$, is given as follows:

$$\text{MR}_\Pi = \text{LFP}(\llbracket \text{PNfun}_\Pi \rrbracket_{f_{\text{un}}})$$

where LFP represents the least fixed point.
Recursive processes: Continuity

- For each process name $N \in \Pi$ (a set of all process names), a process equation is defined:

$$N(x_1, x_2, \ldots x_n) = P$$

where $P \in Proc(\Pi, \Sigma)$, the process name $N$ behaves like the process $P$.

- A special function $PNfun_\Pi : \Pi \rightarrow Proc(\Pi, \Sigma)$, which is called a *process-name function*, in order to describe the right hand sides of defining equations.

- $(\llbracket PNfun_\Pi \rrbracket_{\mathcal{R}})$ is continuous.

- Finally, the semantics $\llbracket P \rrbracket_{\mathcal{R}}$ of each process $P$ is defined as follows: $\llbracket P \rrbracket_{\mathcal{R}} = \llbracket P \rrbracket_{\mathcal{R}(MR_\Pi)}$. Consequently,

$$\llbracket N \rrbracket_{\mathcal{R}} = \llbracket N \rrbracket_{\mathcal{R}(MR_\Pi)}$$

- The ideal interpretation, written $MR_\Pi$, is given as follows:

$$MR_\Pi = \text{LFP}(\llbracket PNfun_\Pi \rrbracket_{\mathcal{R}})$$

where LFP represents the least fixed point.
Recursive processes: Continuity

- For each process name $N \in \Pi$ (a set of all process names), a process equation is defined:
  $$N(x_1, x_2, \ldots x_n) = P$$
  where $P \in \text{Proc}(\Pi, \Sigma)$, the process name $N$ behaves like the process $P$.

- A special function $\text{PNfun}_\Pi : \Pi \rightarrow \text{Proc}(\Pi, \Sigma)$, which is called a process-name function, in order to describe the right hand sides of defining equations.

- $(\text{PNfun}_\Pi)^{\text{fun}}$ is continuous.

- Finally, the semantics $\llbracket P \rrbracket_R$ of each process $P$ is defined as follows: $\llbracket P \rrbracket_R = \llbracket P \rrbracket_R^{\text{MR}_\Pi}$. Consequently,
  $$\llbracket \$N \rrbracket_R = \llbracket \$N \rrbracket_R^{\text{MR}_\Pi}$$

- the ideal interpretation, written $\text{MR}_\Pi$, is given as follows:
  $$\text{MR}_\Pi = \text{LFP}(\text{PNfun}_\Pi)^{\text{fun}}$$
  where LFP represents the least fixed point.
Validation of algebraic laws
Some algebraic laws

- As expected: Internal choice distributes over external choices fails.

\[(P \square Q) \sqcap R \neq_R (P \sqcap R) \square (Q \sqcap R)\]

- Distributive law of prefixing holds:

\[
\begin{align*}
\forall a : A \rightarrow (P(a) \sqcap Q(a)) \\
\rightarrow_R (\forall a : A \rightarrow P(a)) \sqcap (\forall a : A \rightarrow Q(a))
\end{align*}
\]

- Over \(domR_{\Sigma}^{arb}\) and \(domR_{\Sigma}^{m}\) the following laws have been proved: (\(\sqcap\)-idem), (\(\square\)-sym), (\(\sqcap\)-sym), (\(||X||\)-sym), (\(\square\)-assoc), (\(\sqcap\)-assoc), (\(\square\)-\(\sqcap\)-dist), (\(Stop\)-\(||X||\)), (\(\circ\)-step), (prefix-step), and (\(\downarrow\)-step).
Some algebraic laws

- As expected: Internal choice distributes over external choices fails.

\[ (P \square Q) \sqcap R \neq R (P \sqcap R) \square (Q \sqcap R) \]

- Distributive law of prefixing holds:

\[ ?a : A \rightarrow (P(a) \sqcap Q(a)) = R (?a : A \rightarrow P(a)) \sqcap (?a : A \rightarrow Q(a)) \]

- Over \( \text{dom}R_{\Sigma}^{arb} \) and \( \text{dom}R_{\Sigma}^{m} \) the following laws have been proved: (\( \sqcap \)-idem), (\( \square \)-sym), (\( \sqcap \)-sym), (\( ||[X]|| \)-sym), (\( \square \)-assoc), (\( \sqcap \)-assoc), (\( \square \)-\( \sqcap \)-dist), (\( \text{Stop} \)-\( ||[X]|| \)), (\( g \)-step), (prefix-step), and (\( \downarrow \)-step).
Some algebraic laws

- As expected: Internal choice distributes over external choices fails.

\[(P \ Diamond Q) \sqcap R \neq_R (P \sqcap R) \ Diamond (Q \sqcap R)\]

- Distributive law of prefixing holds:

\[\text{?}a : A \rightarrow (P(a) \sqcap Q(a)) =_R (\text{?}a : A \rightarrow P(a)) \sqcap (\text{?}a : A \rightarrow Q(a))\]

- Over \(\text{dom} R^{arb}_\Sigma\) and \(\text{dom} R^m_\Sigma\) the following laws have been proved: (\(\sqcap\)-idem), (\(\Box\)-sym), (\(\sqcap\)-sym), (\(\parallel X \parallel\)-sym), (\(\Box\)-assoc), (\(\sqcap\)-assoc), (\(\Box\)-\(\sqcap\)-dist), (\(\text{Stop-} \parallel X \parallel\)), (\(\searrow\)-step), (prefix-step), and (\(\downarrow\)-step).
Modification of $\text{deadlock}(\textstyle?x : A \rightarrow P)$

Step law of STOP:

\[
\text{Stop} = \textstyle?x : \{\emptyset\} \rightarrow Q
\]

- $\text{deadlocks}(\text{Stop}) = \{\langle \rangle \}$
- $\text{deadlock}(\textstyle?x : A \rightarrow P) =$

Original Version:

\[
\{ s \mid s = \langle a \rangle \uplus t, a \in A, t \in \text{deadlock}(P) \}
\]

Our variant:

\[
\{ s \mid s = \langle a \rangle \uplus t, a \in A, t \in \text{deadlock}(P) \}
\]

\[
\lor (s = \langle \rangle \uplus A = \emptyset) \}
\]

Consequence of this change:

modification of $\text{deadlock}(P[R])$. 


The Stable Revivals Model in CSP-Prover

D G Samuel (Swansea), Y Isobe (AIST, Japan), M Roggenbach (Swansea)

Theorem Proving for Process Algebra

The stable revivals model
Semantics of the stable revivals model
The Domain of the stable revivals model
CSP-Prover
Syntax of CSP-Prover
Implementing the stable revivals model
Steps in Implementing the model
Type correctness
Recursive processes
Validation of algebraic laws
Conclusion
Modification of $\text{deadlock}(?x : A \rightarrow P)$

Step law of STOP:

\[
\text{Stop} = ?x : \{\emptyset\} \rightarrow Q
\]

- $\text{deadlocks}(\text{Stop}) = \{ \langle \rangle \}$
- $\text{deadlock}(?x : A \rightarrow P) =$

Original Version:

\[\{ s \mid s = \langle a \rangle \uparrow t, a \in A, t \in \text{deadlock}(P) \}\]

Our variant:

\[\{ s \mid s = \langle a \rangle \uparrow t, a \in A, t \in \text{deadlock}(P) \}
\quad \vee \quad (s = \langle \rangle \uparrow A = \emptyset)\}

Consequence of this change:

modification of $\text{deadlock}(P[R])$. 
Modification of \( \textit{deadlock}(\ ?x : A \rightarrow P) \)

Step law of STOP:

\[
\text{Stop} = \ ?x : \{\emptyset\} \rightarrow Q
\]

\begin{itemize}
  \item \( \text{deadlocks}(\text{Stop}) = \{\langle\rangle\} \)
  \item \( \text{deadlock}(\ ?x : A \rightarrow P) = \)
\end{itemize}

Original Version:

\[
\{ s \mid s = \langle a \rangle \bigtriangleup t, a \in A, t \in \text{deadlock}(P) \}
\]

Our variant:

\[
\{ s \mid s = \langle a \rangle \bigtriangleup t, a \in A, t \in \text{deadlock}(P) \\
\quad \quad \quad \bigvee (s = \langle\rangle \land A = \emptyset) \}
\]

Consequence of this change:

modification of \( \text{deadlock}(P[R]) \).
Modification of \textit{deadlock}(?x : A \rightarrow P)

Step law of STOP:

\[
\text{Stop} = ?x : \{\emptyset\} \rightarrow Q
\]

- \textit{deadlocks}(\text{Stop}) = \{\langle \rangle \}\]
- \textit{deadlock}(?x : A \rightarrow P) =

Original Version:

\{s \mid s = \langle a \rangle \blacktriangledown t, a \in A, t \in \text{deadlock}(P)\}

Our variant:

\{s \mid s = \langle a \rangle \blacktriangledown t, a \in A, t \in \text{deadlock}(P) \land (s = \langle \rangle \land A = \emptyset)\}

Consequence of this change:

modification of \textit{deadlock}(P[R]).
Modification of \textit{deadlock}(\(?x : A \rightarrow P\))

Step law of STOP:

\[
\text{Stop} = ?x : \{\emptyset\} \rightarrow Q \]

- \(\text{deadlocks}(\text{Stop}) = \{\langle \rangle \}\)
- \(\text{deadlock}(?x : A \rightarrow P) = \) 

Original Version:
\[
\{ s \mid s = \langle a \rangle \bowtie t, a \in A, t \in \text{deadlock}(P) \} 
\]

Our variant: \[
\{ s \mid s = \langle a \rangle \bowtie t, a \in A, t \in \text{deadlock}(P) \\
\quad \quad \lor (s = \langle \rangle \land A = \emptyset) \} 
\]

Consequence of this change:
modification of \textit{deadlock}(P[R]).
Modification of $\text{deadlock}(\forall x : A \rightarrow P)$

Step law of STOP:

\[
\text{Stop} = \forall x : \{\emptyset\} \rightarrow Q
\]

- $\text{deadlocks} (\text{Stop}) = \{\langle \rangle\}$
- $\text{deadlock}(\forall x : A \rightarrow P) = \quad$

Original Version:

\[
\{ s \mid s = \langle a \rangle \upharpoonright t, a \in A, t \in \text{deadlock}(P) \}\]

Our variant: \[
\{ s \mid s = \langle a \rangle \upharpoonright t, a \in A, t \in \text{deadlock}(P) \\
\quad \lor (s = \langle \rangle \upharpoonright A = \emptyset) \}\}

Consequence of this change:

modification of $\text{deadlock}(P[R])$. 
Conclusion
Summary

Summary:

- **Mechanical Verification of the model \( \mathcal{R} \):**
  - \( \text{dom}R_{\Sigma}^{\text{fin}}, \text{dom}R_{\Sigma}^{\text{arb}} \) and \( \text{dom}R_{\Sigma}^{m} \) are CPOs.
  - Semantic function are type correct and continuous.
  - Most algebraic laws hold.

- **Working Implementation of the model \( \mathcal{R} \).**

- **Suggestions to improve the model \( \mathcal{R} \):**
  - \( \text{dom}R_{\Sigma}^{m} \) for infinite alphabets.
  - Modified semantical clauses for multiple prefix and renaming.

Future work:

- Validate more algebraic laws.
- Case study on the implementation: a good example would be on-line shopping example given in Roscoe’s original paper.
Summary:

- Mechanical Verification of the model \( \mathcal{R} \):
  - \( \text{dom} R^\text{fin} \), \( \text{dom} R^\text{arb} \), and \( \text{dom} R^m \) are CPOs.
  - Semantic function are type correct and continuous.
  - Most algebraic laws hold.

- Working Implementation of the model \( \mathcal{R} \).

- Suggestions to improve the model \( \mathcal{R} \):
  - \( \text{dom} R^m \) for infinite alphabets.
  - Modified semantical clauses for multiple prefix and renaming.

Future work:

- Validate more algebraic laws.

- Case study on the implementation: a good example would be on-line shopping example given in Roscoe’s original paper.
Summary:

- Mechanical Verification of the model $\mathcal{R}$:
  - $\text{dom} R^\text{fin}_\Sigma$, $\text{dom} R^\text{arb}_\Sigma$ and $\text{dom} R^m_\Sigma$ are CPOs.
  - Semantic function are type correct and continuous.
  - Most algebraic laws hold.

- Working Implementation of the model $\mathcal{R}$.

- Suggestions to improve the model $\mathcal{R}$:
  - $\text{dom} R^m_\Sigma$ for infinite alphabets.
  - Modified semantical clauses for multiple prefix and renaming.

Future work:-

- Validate more algebraic laws.
- Case study on the implementation: a good example would be on-line shopping example given in Roscoe’s original paper.
Summary

Summary:-

- Mechanical Verification of the model $R$:
  - $\text{dom}R^\text{fin}_{\Sigma}$, $\text{dom}R^\text{arb}_{\Sigma}$ and $\text{dom}R^m_{\Sigma}$ are CPOs.
  - Semantic function are type correct and continuous.
  - Most algebraic laws hold.

- Working Implementation of the model $R$.

- Suggestions to improve the model $R$:
  - $\text{dom}R^m_{\Sigma}$ for infinite alphabets.
  - Modified semantical clauses for multiple prefix and renaming.

Future work:-

- Validate more algebraic laws.

- Case study on the implementation: a good example would be on-line shopping example given in Roscoe’s original paper.
Summary:

- **Mechanical Verification of the model $\mathcal{R}$:**
  - $\text{dom}R^\text{fin}_\Sigma$, $\text{dom}R^\text{arb}_\Sigma$ and $\text{dom}R^m_\Sigma$ are CPOs.
  - Semantic function are type correct and continuous.
  - Most algebraic laws hold.

- **Working Implementation of the model $\mathcal{R}$**.

- **Suggestions to improve the model $\mathcal{R}$:**
  - $\text{dom}R^m_\Sigma$ for infinite alphabets.
  - Modified semantical clauses for multiple prefix and renaming.

Future work:-

- Validate more algebraic laws.
- Case study on the implementation: a good example would be on-line shopping example given in Roscoe’s original paper.
Summary

Summary:-

- **Mechanical Verification of the model $\mathcal{R}$:**
  - $domR^\text{fin}_\Sigma$, $domR^arb_\Sigma$ and $domR^m_\Sigma$ are CPOs.
  - Semantic function are type correct and continuous.
  - Most algebraic laws hold.

- **Working Implementation of the model $\mathcal{R}$**.

- **Suggestions to improve the model $\mathcal{R}$:**
  - $domR^m_\Sigma$ for infinite alphabets.
  - Modified semantical clauses for multiple prefix and renaming.

Future work:-

- Validate more algebraic laws.

- Case study on the implementation: a good example would be on-line shopping example given in Roscoe’s original paper.
Summary:

- **Mechanical Verification of the model \( \mathcal{R} \):**
  - \( \text{dom}\mathcal{R}_{\Sigma}^{\text{fin}} \), \( \text{dom}\mathcal{R}_{\Sigma}^{\text{arb}} \), and \( \text{dom}\mathcal{R}_{\Sigma}^{m} \) are CPOs.
  - Semantic function are type correct and continuous.
  - Most algebraic laws hold.

- **Working Implementation of the model \( \mathcal{R} \).**

- **Suggestions to improve the model \( \mathcal{R} \):**
  - \( \text{dom}\mathcal{R}_{\Sigma}^{m} \) for infinite alphabets.
  - Modified semantical clauses for multiple prefix and renaming.

Future work:

- Validate more algebraic laws.
- Case study on the implementation: a good example would be on-line shopping example given in Roscoe’s original paper.
Summary

Summary:-

- Mechanical Verification of the model $\mathcal{R}$:
  - $\text{dom}R^\text{fin}_\Sigma$, $\text{dom}R^\text{arb}_\Sigma$ and $\text{dom}R^m_\Sigma$ are CPOs.
  - Semantic function are type correct and continuous.
  - Most algebraic laws hold.

- Working Implementation of the model $\mathcal{R}$.

- Suggestions to improve the model $\mathcal{R}$:
  - $\text{dom}R^m_\Sigma$ for infinite alphabets.
  - Modified semantical clauses for multiple prefix and renaming.

Future work:-

- Validate more algebraic laws.

- Case study on the implementation: a good example would be on-line shopping example given in Roscoe’s original paper.
Summary:

- Mechanical Verification of the model $\mathcal{R}$:
  - $\text{dom}R^\text{fin}_\Sigma$, $\text{dom}R^\text{arb}_\Sigma$ and $\text{dom}R^m_\Sigma$ are CPOs.
  - Semantic function are type correct and continuous.
  - Most algebraic laws hold.

- Working Implementation of the model $\mathcal{R}$.

- Suggestions to improve the model $\mathcal{R}$:
  - $\text{dom}R^m_\Sigma$ for infinite alphabets.
  - Modified semantical clauses for multiple prefix and renaming.

Future work:

- Validate more algebraic laws.

- Case study on the implementation: a good example would be on-line shopping example given in Roscoe’s original paper.
Summary:

- Mechanical Verification of the model $\mathcal{R}$:
  - $domR_{\Sigma}^{\text{fin}}$, $domR_{\Sigma}^{\text{arb}}$ and $domR_{\Sigma}^{m}$ are CPOs.
  - Semantic function are type correct and continuous.
  - Most algebraic laws hold.
- Working Implementation of the model $\mathcal{R}$.
- Suggestions to improve the model $\mathcal{R}$:
  - $domR_{\Sigma}^{m}$ for infinite alphabets.
  - Modified semantical clauses for multiple prefix and renaming.

Future work:-

- Validate more algebraic laws.
- Case study on the implementation: a good example would be on-line shopping example given in Roscoe’s original paper.
Summary:-

- Mechanical Verification of the model $\mathcal{R}$:
  - $\text{dom}R^\text{fin}_\Sigma$, $\text{dom}R^\text{arb}_\Sigma$ and $\text{dom}R^m_\Sigma$ are CPOs.
  - Semantic function are type correct and continuous.
  - Most algebraic laws hold.

- Working Implementation of the model $\mathcal{R}$.

- Suggestions to improve the model $\mathcal{R}$:
  - $\text{dom}R^m_\Sigma$ for infinite alphabets.
  - Modified semantical clauses for multiple prefix and renaming.

Future work:-

- Validate more algebraic laws.

- Case study on the implementation: a good example would be on-line shopping example given in Roscoe’s original paper.