Implementation of the stable revivals model in CSP-Prover

D. Gift Samuel
University of Wales Swansea

April 4, 2007

Supported by EPSRC grant EP/D037212/1
Implementation of the stable revivals model in CSP-Prover

D. Gift Samuel
University of Wales Swansea

Outline

CSP Prover
CSP Models
CSP Prover
Proof Infrastructure

The stable revivals model
Motivation to the stable revivals model
Introduction to the stable revivals model

Implementation of the stable revivals model
Implementation of Domain
Domain of the stable revivals Model is a CPO

Future Work

- CSP-Prover
- Motivation and Introduction to the stable revivals model
- Implementation of the stable revivals model
CSP Models

- CSP is a language to describe processes in concurrent systems.
- A collection of mathematical models and reasoning methods.
- Traces model $\mathcal{T}$, failure-divergences model $\mathcal{N}$ and the stable failures model $\mathcal{F}$.
- The stable revivals model $\mathcal{R}$ is a recently developed model (2005).
CSP-Prover is a proof infrastructure to prove refinement and equality proofs using the interactive theorem prover Isabelle. Uses the logic HOL-Complex.

Developed by Yoshinao Isobe (AIST, Japan) and Markus Roggenbach (University of Wales Swansea)

Proofs on infinite state systems, which may also have infinite nondeterminism.
Proof Infrastructure

- Verifying Process Equivalence and Process Refinement
  1. Semantical proof - by semantics function
  2. Syntactical proof - by algebraic CSP laws
  3. Semi-Automatically proof - by tactics

- Currently focuses on the stable failures model $\mathcal{F}$ and traces model $\mathcal{T}$. 
Motivations to the stable revivals model

- The stable revivals model (Roscoe 2005) was developed to reason about responsiveness and stuck-freeness.
- Gives assurance that individual processes don’t behave in an undesirable manner.
- 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 not stuck-free if network of process doesn’t terminate leaving one partner hanging.
An example

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.

\[ P = (\text{rep} \rightarrow \text{Skip}) \sqcap \text{Skip} \]
\[ Q = \text{rep} \rightarrow \text{Skip} \]

- Q RespondsTo P as Q is ready to engage in rep,
- but P RespondsTo Q is not
Introduction to the stable revivals model

▶ The stable revivals semantics assigns meaning for each process P in terms of \((\text{traces}(P), \text{deadlock}(P), \text{revivals}(P))\).

▶ \text{traces}(P) \subseteq \Sigma^* √, where \(\Sigma\) is the communication alphabet

▶ \text{deadlock}(P) \subseteq \Sigma^*

P can deadlock after execution of \(\sigma \in \text{deadlock}(P)\)

▶ \text{revivals}(P) \subseteq (\Sigma^* \times P(\Sigma) \times \Sigma √)

\(P\) can execute \(\sigma\), stably refuse \(X\) and then perform \(a\) for \((\sigma, X, a) \in \text{revivals}(P)\).
Domain of the stable revivals model

- Domain $\mathcal{R}$ is the set of all $(T, D, R)$ such that
  - $T_1$ $T$ is nonempty and prefix-closed.
  - $D_1$ $D \subseteq T$.
  - $R_1$ $(s, X, a) \in R \Rightarrow s ^\langle a \rangle \in T$.
  - $R_2$ $(s, X, a) \in R \land Y \subseteq X \Rightarrow (s, Y, a) \in R$.
  - $R_3$ $(s, X, a) \in R \land b \in \Sigma$
    $\Rightarrow ((s, X, b) \in R \lor (s, X \cup \{b\}, a) \in R)$.
  - $R_4$ $s ^\langle \sqrt{\ } \rangle \in T \Rightarrow (s, \Sigma, \sqrt{\ }) \in R$. 

Implementation of the stable revivals model in CSP-Prover

D. Gift Samuel
University of Wales Swansea

Outline
- CSP Prover
- CSP Models
- CSP Prover
- Proof Infrastructure
- The stable revivals model
  - Motivation to the stable revivals model
  - Introduction to the stable revivals model
- Implementation of the stable revivals model
  - Implementation of Domain
  - Domain of the stable revivals Model is a CPO
- Future Work

Isabelle/HOL-Complex
Implementation of the stable revivals model in CSP-Prover

D. Gift Samuel
University of Wales Swansea

Outline
CSP Prover
CSP Models
Proof Infrastructure
The stable revivals model
Motivation to the stable revivals model
Introduction to the stable revivals model
Implementation of the stable revivals model
Implementation of Domain
Domain of the stable revivals Model is a CPO
Future Work
Creating Revivals Components

- Each revival is triple ($\sigma, X, a$)
  - $\sigma$ (trace) doesn't have $\sqrt{}$
  - $X$ (refusal set) doesn't have $\sqrt{}$

- types 'a revival = 
  "('a trace * 'a event set * 'a event)"
  "HC_RT F == (ALL f . (f: F) ? Tick $\sim$: sett(FstR(f)))"
  "HC_RF F == (ALL f . (f: F) ? Tick $\sim$: SndR(f))"

typedef 'a setR = 
"{ R :: ('a revival set) . HC_RT(R) & HC_RF(R) }"

- Creates the type 'a revival to represent a trace which doesn't contain $\sqrt{}$, a set of events which does not contain $\sqrt{}$ and an event which may be $\sqrt{}$. 
Creating the domain of the stable revivals model.

- types 'a domTsetDsetR="('a domT*'a setD*'a setR)"
- typedef 'a domR "{ T ::('a domTsetDsetR).HC_D1(T)&HC_R1(T)&HC_R2(T)&HC_R3(T) & HC_R4(T)}"

Creates the type 'a domR to represent \((T, D, R)\) such that it satisfies all the healthiness conditions.
Revival component is a CPO

- Overload the definition of $\leq$ of the axiomatic class order.
  
  ```
  defs (overloaded)
  subsetR_def : "F <= E == Rep_setR (F) <= Rep_setR (E)"
  psubsetR_def : "F < E == Rep_setR (F) < Rep_setR (E)"
  instance setR :: (type) order
  ```

- The least upper bound of any directed set is simply the componentwise union.

- UnionR_def : "UnionR Rs == Abs_setR(Union(Rep_setR ' Rs))"

- lemma Union_setR_in_setR : "Union (Rep_setR ' Rs) : setR"

- UnionR Rs is an upper bound of Rs and The least upper bound of Rs is UnionR Rs.

- instance setR :: (type) cpo

Diagram:

```
'a setR
\arrow{Rep_setR}
\arrow{Abs_setR}
('a trace !=' a event set !=' a event)
```
Summary & Future works

Summary

- Implemented the domain of $\mathcal{R}$ in Csp-Prover.
- Proved in Csp-Prover that this domain is a CPO.
- Defined the semantics functions of $\mathcal{R}$ in Csp-Prover.

Future work

- Proving type correctness of these semantics functions.
- Proving these semantics functions to be continuous.
- Defining and proving step laws for $\mathcal{R}$.
- Develop proof infrastructure for $\mathcal{R}$.