Specification-based testing for refinement

Temesghen Kahsai
University of Iowa
Specification-based testing for refinement

Temesghen Kahsai

University of Iowa

This work is done in cooperation with:

- **M. Roggenbach**, Swansea University (Wales) and
- **H-B. Schlingloff**, Humboldt University Berlin / Fraunhofer FIRST (Germany)
Motivation

Joint project by a number (mainly Swiss) financial institutes (www.eftpos2000.ch)
EP2 informal specification
(12 documents)

Vertical Development

Implementation
EP2 informal specification (12 documents)

Vertical Development

Formalize in CSP-CASL

Implementation

$Sp_0$

$Sp_1$

$Sp_n$
Overview

Processes and Data: CSP-CASL

Testing and refinement in CSP-CASL
Processes and Data: CSP-CASL
Processes and Data: CSP-CASL

Processes and Data

Processes: CSP

Communicating Sequential Processes (Hoare 1985, Roscoe 1998)

- Established formalism to describe concurrent systems.
- Applications in industry include Train Controllers, Avionics, Security Protocols.
- Tools: FDR, ProBe, Csp-Prover (Swansea University / AIST)

Data: CASL

Common Algebraic Specification Language.

- De-facto standard in algebraic specification, stable release 1.0.2 in October 2003.
- Tools: HETS – Parsing, Static analysis, Proof management and Interface with theorem provers (Isabelle, SPASS, Vampire, etc).
Processes and Data

Processes: CSP

Communicating Sequential Processes (Hoare 1985, Roscoe 1998)
- Established formalism to describe concurrent systems.
- Applications in industry include Train Controllers, Avionics, Security Protocols.
- Tools: FDR, ProBe, Csp-Prover (Swansea University / AIST)

Data: CASL

Common Algebraic Specification Language.
- De-facto standard in algebraic specification, stable release 1.0.2 in October 2003.
- Tools: HETS – Parsing, Static analysis, Proof management and Interface with theorem provers (Isabelle, SPASS, Vampire, etc).
CSP-CASL

\texttt{ccspec} \( S_p = \text{data} \ D \ \text{channel} \ Ch \ \text{process} \ P \ \text{end} \)

\textbf{Data part} \( D \) : structured \texttt{CASL} specification

\textbf{Channel part} \( Ch \) : communication channels typed by \texttt{CASL} sorts

\textbf{Process part} \( P \) : set of (mutually recursive) \texttt{CSP} processes where

- communications: \texttt{CASL} terms
- sets of communications: \texttt{CASL} sorts
- relational renamings: binary \texttt{CASL} predicates
- conditions: \texttt{CASL} formulae
\[ Sp = (D, P) \]

\[ (P'(A(\beta(M))))_{M \in \text{Mod}(D)} \rightarrow (d_M)_{M \in \text{Mod}(D)} \]
References for Csp-Casl


A specification exercise:
Binary Calculator
Setting up the interface

\textbf{ccspec} B_{\text{CALC}0} =
\textbf{data}
  \textbf{sort} \ Number
  \textbf{ops} \ 0, 1 : \ Number;
    \_ + \_ : \ Number \times \ Number \to \? \ Number
\textbf{channel}
  \ Button, \ Display : \ Number
\textbf{process}
  P_0 = (\?x : \ Button \to P_0) \sqcap (\!y : \ Display \to P_0)
\textbf{end}

\textit{Button!}0 \to \textit{Button!}0 \textit{ is left open} behaviour.
Alternating buttons and display

cspec \texttt{BCALC1} =
data
\begin{align*}
\textbf{sort} & \quad \text{Number} \\
\textbf{ops} & \quad 0, 1 : \text{Number}; \\
& \quad \_ + \_ : \text{Number} \times \text{Number} \rightarrow \? \text{Number}
\end{align*}
channel
\begin{align*}
\text{Button, Display} &: \text{Number}
\end{align*}
process
\begin{align*}
P_1 &= \text{x : Button} \rightarrow \text{!y : Display} \rightarrow P_1
\end{align*}
end

\begin{quote}
\text{Button!0} \rightarrow \text{Button!0} \text{ is a 'unwanted' behaviour.}
\end{quote}
Fixing the displayed value

ccspec \( BCALC3 = \)
data
  \textbf{sort} \ Number
ops \( 0, 1 : \ Number; \)
  \( _+ _: \ Number \times \ Number \rightarrow ? \ Number \)
channel
  \textit{Button}, \textit{Display} : \ Number
process
  \( P_2 = ?x : \textit{Button} \rightarrow \textit{Display}!x \rightarrow ?y : \textit{Button} \)
  \( \rightarrow \textit{Display}!(x + y) \rightarrow P_2 \)
end

\begin{verbatim}
Button!0 \rightarrow \textit{Display}!0 \rightarrow \textit{Button}!1 \rightarrow \textit{Display}!1
\end{verbatim}
is left open behaviour.
1-bit arithmetic

ccspec $\textsc{BCalc4} =$

data

\begin{align*}
\text{CARDINAL} \ [\text{op WordLength} = 1 : \text{Nat}] \\
\text{with sort} \ \text{CARDINAL} \mapsto \text{Number}
\end{align*}

channel

\begin{align*}
\text{Button, Display} : \text{Number}
\end{align*}

process

\begin{align*}
P_4 = ?x : \text{Button} \rightarrow \text{Display}!x \rightarrow ?y : \text{Button} \\
\rightarrow \text{Display}!(x + y) \rightarrow P_4
\end{align*}

end

monomorphic data, no internal non-determinism: 
behaviour either 'unwanted' or 'intended'.

Testing and refinement in CSP-CASL
CSP-CASL test case

Given:

- \( Sp = (D, P) \) CSP-CASL specification

A test case \( T \) is any CSP-CASL process in the signature of \( D \).
**CSP-CASL test case**

Given:

- \( Sp = (D, P) \) CSP-CASL specification

A **test case** \( T \) is any CSP-CASL process in the signature of \( D \).

E.g. \( Button!0 \rightarrow Display!0 \rightarrow Button!1 \rightarrow Display!1 \)

**Remark:** A CSP-CASL test case can also have variables.
The colour of test $T$ with respect to $(Sp, P)$ is a value in \{\textit{red}, \textit{yellow}, \textit{green}\}. 
Colouring test processes

Given a CSP-CASL specification $Sp = (D, P)$ and a test case $T$:
Colouring test processes

Given a \text{CSP-CASL} specification \( Sp = (D, P) \) and a test case \( T \):

- \text{colour}(T) = \text{green} \iff \forall M \in \text{Mod}(D) \text{ and all variable evaluations } \nu : X \rightarrow M \text{ it holds that:}
  1. \( \text{traces}([T]_\nu) \subseteq \text{traces}([P]_{\emptyset : \emptyset \rightarrow \beta(M)}) \)
  and
  2. for all \( tr = \langle t_1, \ldots, t_n \rangle \in \text{traces}([T]_\nu) \text{ and for all } 1 \leq i \leq n \) it holds that:
     \( \langle \langle t_1, \ldots, t_{i-1} \rangle, \{t_i\} \rangle \not\in \text{failures}([P]_{\emptyset : \emptyset \rightarrow \beta(M)}) \)

- \text{colour}(T) = \text{red} \iff \text{for all models } M \in \text{Mod}(D) \text{ and all variable evaluations } \nu : X \rightarrow M \text{ it holds that:}
  \( \text{traces}([T]_\nu) \not\subseteq \text{traces}([P]_{\emptyset : \emptyset \rightarrow \beta(M)}) \)

- \text{colour}(T) = \text{yellow} \text{ otherwise.}
Colouring test process

Syntactic characterization based on full abstraction proofs for \( \text{Csp} \).

**E.g., traces condition**

\[
\text{Check}_T = ((P \parallel T)[[R_{s_1}]] \ldots [[R_{s_h}]] \\
[[A]] \text{count}(n)) \backslash s_1 \backslash \cdots \backslash s_n
\]

\[
\text{count}(n : \text{Nat}) = \begin{cases} 
\text{OK} \rightarrow \text{Stop} & \text{if } n = 0 \\
\text{count}(n - 1) & \text{else}
\end{cases}
\]

Coloring of test cases is done in \( \text{CSP-CASL-Prover} \).
Execution of test cases

... execute a test case w.r.t. particular SUT.

Point of Control and Observation

A PCO \( \mathcal{P} = (\mathcal{A}, \|...\|, \mathcal{D}) \) of an SUT consists of:

- an alphabet \( \mathcal{A} \) of primitive events
- a mapping \( \|...\| : \mathcal{A} \rightarrow T_\Sigma \)
- a direction \( \mathcal{D} : \mathcal{A} \rightarrow \{ts2sut, sut2ts\} \).

Deriving the test verdict

- The verdict of test \( T \) w.r.t. \( Sp = (\mathcal{D}, \mathcal{P}) \) and a particular SUT is a value in \( \{\text{pass, fail, inconclusive}\} \).
- An algorithm to derive the test verdict on the fly.
- \( T_{EV} \): Hardware in the loop testing framework.
Support for vertical development
Well-behaved refinement

Let $Sp = (D, P) \leadsto Sp' = (D', P')$ be a refinement with certain properties

$\leq$ is called well-behaved (w-b), iff:

1. $\text{colour}(T) = \text{green}$ with respect to $Sp = (D, P)$ implies $\text{colour}(T) = \text{green}$ with respect to $Sp = (D', P')$, and

2. $\text{colour}(T) = \text{red}$ with respect to $Sp = (D, P)$ implies $\text{colour}(T) = \text{red}$ with respect to $Sp' = (D', P')$. 
Well-behaved refinement relations

CSP-CASL refinement

\[ Sp = (D, P) \sim_{T,\mathcal{F},\mathcal{N},\mathcal{R}} Sp' = (D', P') \]

- Data refinement: 'less algebraic models'
- Process refinement: 'less internal non-determinism'
Well-behaved refinement relations

**CSP-CASL refinement**

\[ Sp = (D, P) \leadsto_{\mathcal{T,F,N,R}} Sp' = (D', P') \]

- Data refinement: 'less algebraic models'
- Process refinement: 'less internal non-determinism'

<table>
<thead>
<tr>
<th>Refinement relation</th>
<th>Well behaved</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data refinement(^+)</td>
<td>√</td>
</tr>
<tr>
<td>Process refinement over (\mathcal{T}) model</td>
<td>X</td>
</tr>
<tr>
<td>*Process refinement over (\mathcal{F}) model</td>
<td>√</td>
</tr>
<tr>
<td>*Process refinement over (\mathcal{N}) model</td>
<td>√</td>
</tr>
<tr>
<td>*Process refinement over (\mathcal{R}) model</td>
<td>√</td>
</tr>
</tbody>
</table>

\(^+\) it works with \(\mathcal{F}, \mathcal{N}, \mathcal{R}\)

\(*\) requires divergence-free processes.


Summary

Specification — Test Cases — Implementation

T1 ..
T2 ..
......
...

CSP-CASL Prover

EP2 Test Environment

PASS - INCONCLUSIVE - FAIL
Thanks!