# Specification-based testing for refinement

Temesghen Kahsai

University of Iowa

# Specification-based testing for refinement

Temesghen Kahsai

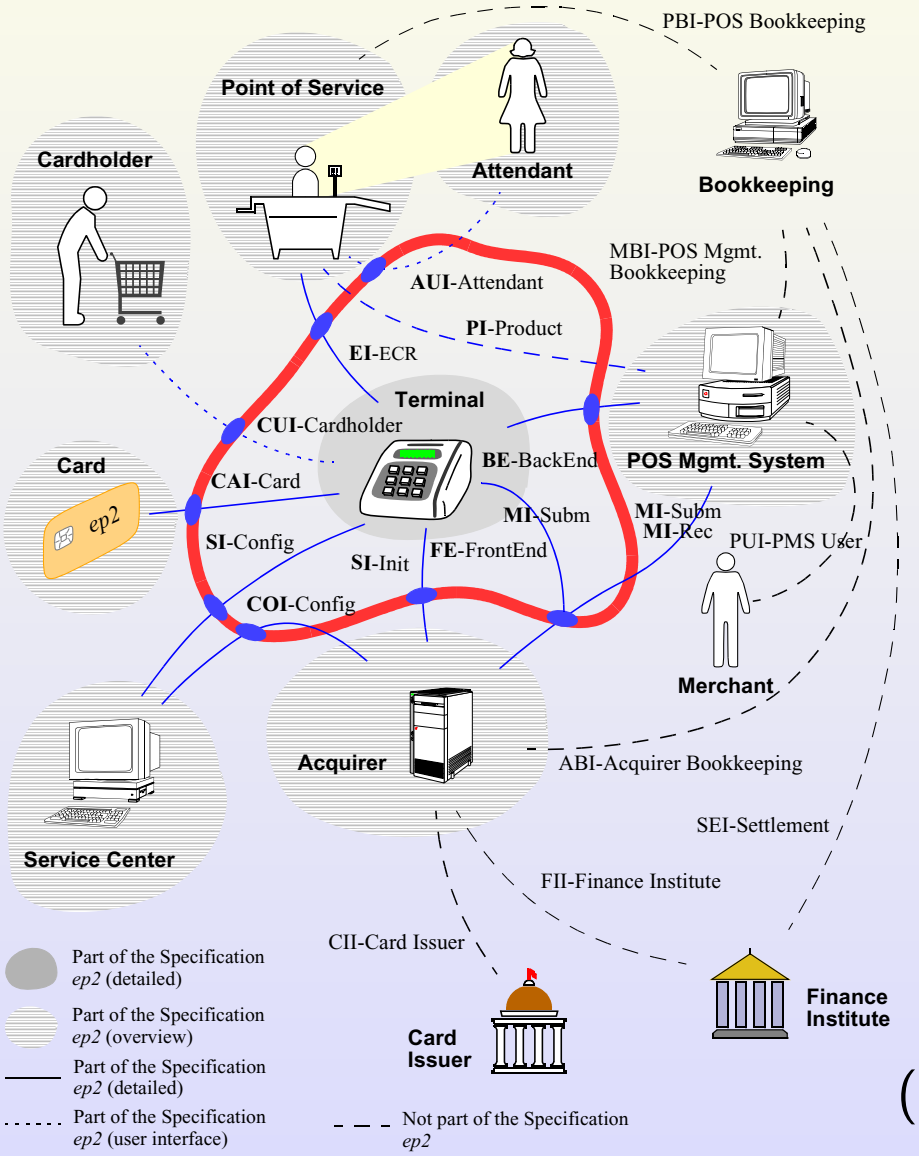
University of Iowa

This work is done in cooperation with:

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

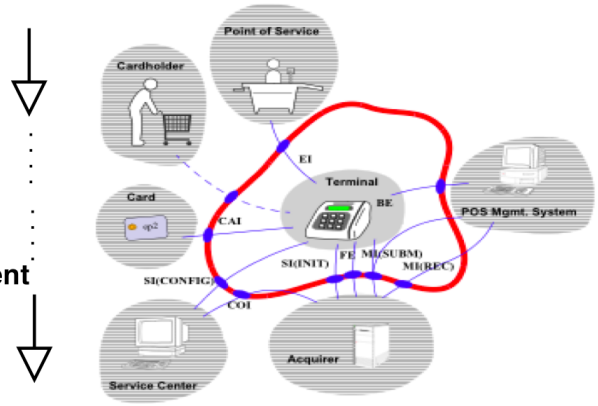
# Motivation

# EP2 or EFT/POS 2000:
## International standard for Electronic Funds Transfer Point Of Service 2000




Joint project by a number (mainly Swiss) financial institutes (`www.eftpos2000.ch`)

**EP2 informal specification**
**(12 documents)**

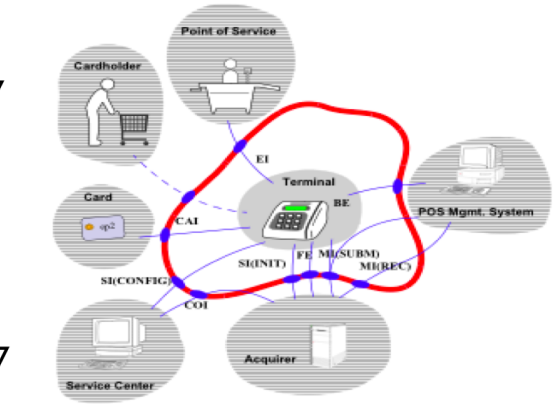



**Vertical Development**

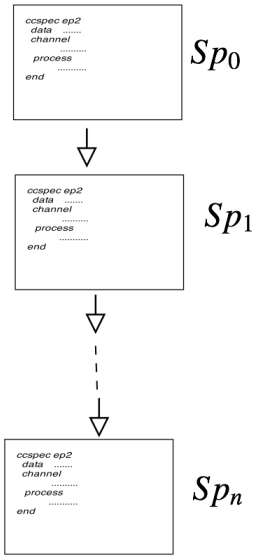
**Implementation**

**EP2 informal specification**
**(12 documents)**

Formalize in
CSP-CASL

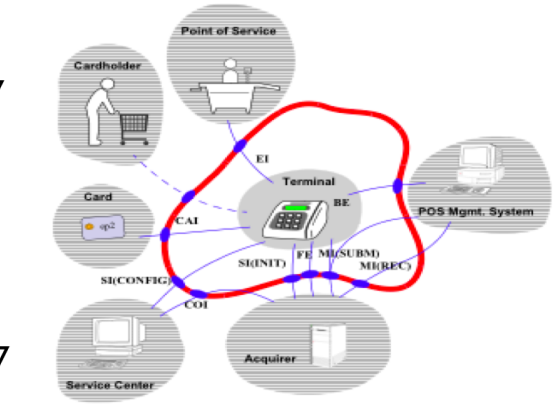



**Vertical Development**




**Implementation**

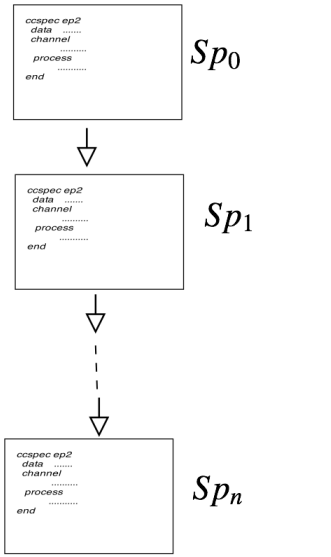
**EP2 informal specification**
**(12 documents)**




**Vertical Development**

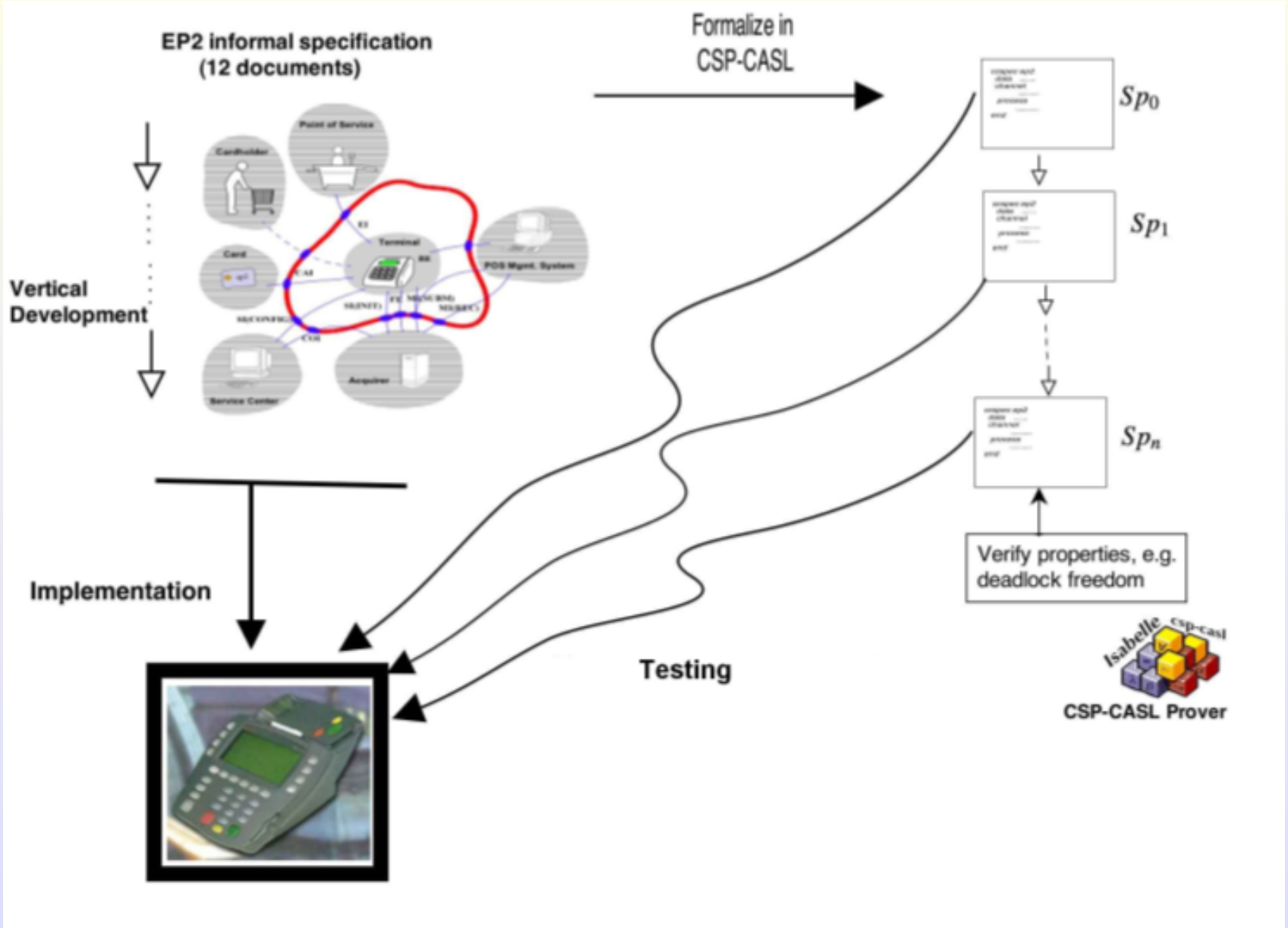
**Implementation**

Formalize in CSP-CASL




Verify properties, e.g. deadlock freedom


**CSP-CASL Prover**


EP2 informal specification
(12 documents)
Formalize in
CSP-CASL
Vertical
Development
Point of Service
Cardholder
Card
Terminal
POS Mgmt. System
Acquirer
Service Center
$Sp_0$
$Sp_1$
$Sp_n$
Verify properties, e.g.
deadlock freedom
Isabelle csp-casl
CSP-CASL Prover
Implementation
Testing

# Overview

Processes and Data: CSP-CASL

Testing and refinement in 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)

# 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

**ccspec** $Sp =$ **data** $D$ **channel** $Ch$ **process** $P$ **end**

Data part $D$ : structured CASL specification

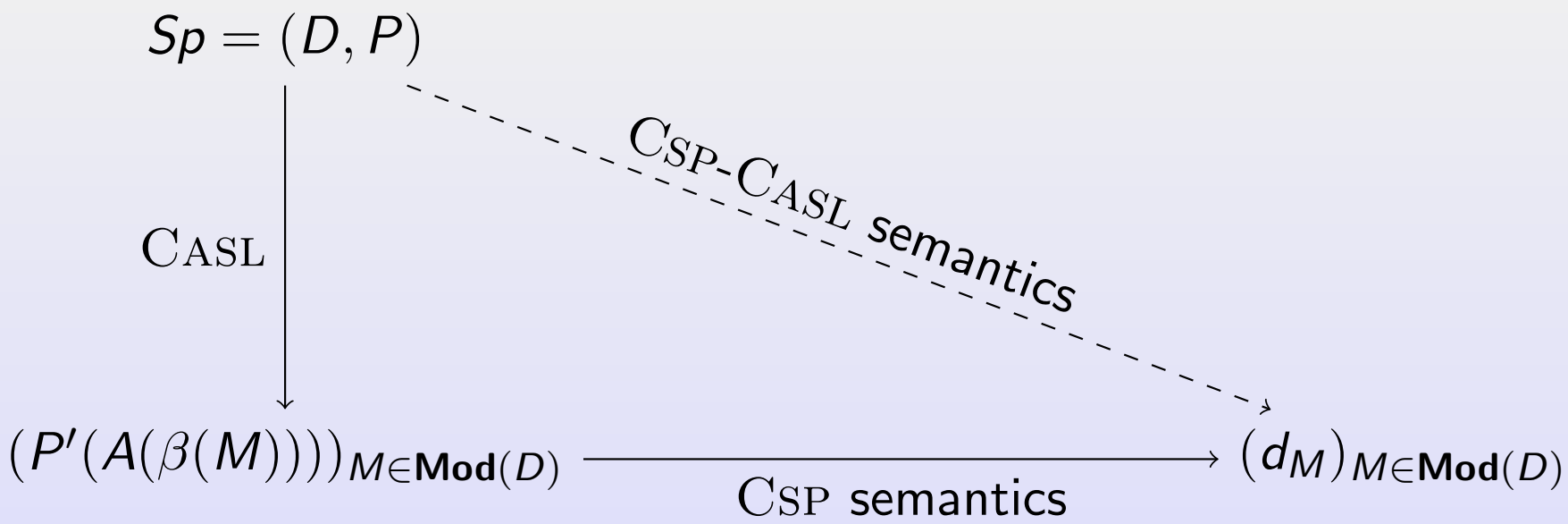
Channel part $Ch$: communication channels typed by CASL sorts

Process part $P$ : set of (mutually recursive) CSP processes where

- communications: CASL terms
- sets of communications: CASL sorts
- relational renamings: binary CASL predicates
- conditions: CASL formulae

# CSP-CASL

## CSP-CASL 2-step semantics



# References for CSP-CASL


- T. Kahsai: **Property preserving development and testing for Csp-Casl**. Ph.D. thesis, Swansea University, June 2010.
- T. Kahsai, A. Gimblett, L. O'Reilly, M. Roggenbach: **On the whereabouts of Csp-Casl- A Survey**. In STN, Festschrift dedicated to Bernd Krieg-Brückner, Bremen, 2009.
- T. Kahsai, M. Roggenbach: **Property preserving refinement notions for Csp-Casl**. In WADT 2008. Springer, 2009.
- M. Roggenbach: **Csp-Casl - A new integration of process algebra and algebraic specification**, In TCS 2006, Elsevier, 2006.

# A specification exercise: Binary Calculator

# Setting up the interface

**ccspec** $\mathrm{BCALC0} =$
**data**
**sort** *Number*
**ops** $0, 1 : Number;$
$\_\_ + \_\_ : Number \times Number \to ? Number$
**channel**
$Button, Display : Number$
**process**
$P_0 = (?x : Button \to P_0) \sqcap (!y : Display \to P_0)$
**end**

$Button!0 \to Button!0$ is left open behaviour.

# Alternating buttons and display

**ccspec** $\mathrm{BCALC1} =$
**data**

**sort** *Number*
**ops** $0, 1 : Number;$
$\_\_ + \_\_ : Number \times Number \to ? Number$

**channel**

$Button, Display : Number$

**process**

$P_1 = ?x : Button \to !y : Display \to P_1$

**end**

*Button*!0 → *Button*!0 is a 'unwanted' behaviour.

# Fixing the displayed value

**ccspec** BCalc3 =
**data**
**sort** *Number*
**ops** $0, 1 : Number$;
$\_\_ + \_\_ : Number \times Number \to ? Number$
**channel**
$Button, Display : Number$
**process**
$P_2 = ?x : Button \to Display!x \to ?y : Button \to Display!(x + y) \to P_2$
**end**

$Button!0 \to Display!0 \to Button!1 \to Display!1$
is left open behaviour.

# 1-bit arithmetic

**ccspec** BCalc4 =
**data**
CARDINAL [**op** *WordLength* = 1 : *Nat*]
**with sort** *CARDINAL* ↦ *Number*
**channel**
*Button*, *Display* : *Number*
**process**
$P_4 = ?x : Button \rightarrow Display!x \rightarrow ?y : Button \rightarrow Display!(x + y) \rightarrow P_4$
**end**

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

# 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

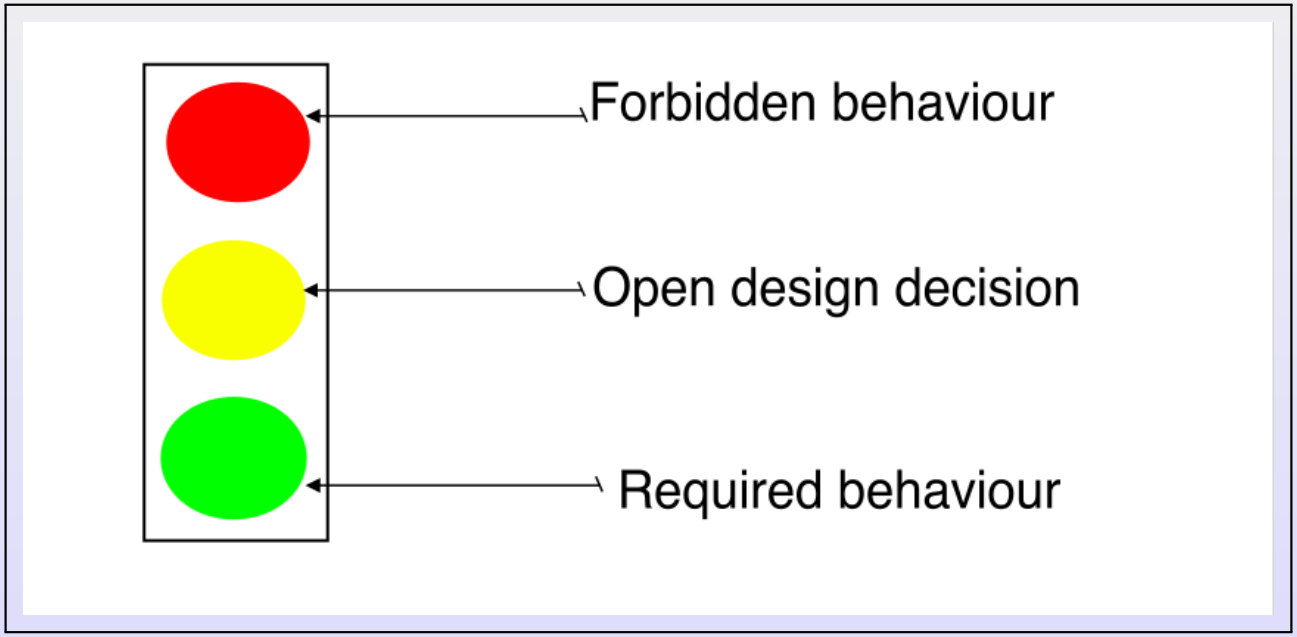
A test case $T$ is any CSP-CASL process in the signature of $D$.

e.g. $Button!0 \rightarrow Display!0 \rightarrow Button!1 \rightarrow Display!1$

Remark: A CSP-CASL test case can also have variables.

# Colouring test processes

The colour of test $T$ with respect to $(Sp, P)$ is a value in $\{red, yellow, green\}$.



# Colouring test processes

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

# Colouring test processes

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

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

# Colouring test process

Syntactic characterization based on full abstraction proofs for CSP.

## E.g., traces condition

$$
\begin{array}{lcl}
Check_T & = & ((P \parallel T)[[R_{s_1}]] \ldots [[R_{s_h}]] \\
 & & |[\, A \,]|\, count(n)) \setminus s_1 \setminus \cdots \setminus s_n \\
count(n : Nat) & = & \textbf{if } n = 0 \textbf{ then } OK \rightarrow Stop \textbf{ else } count(n-1)
\end{array}
$$

Coloring of test cases is done in 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 $\|...\| : \mathcal{A} \longrightarrow T_\Sigma$
- a direction $D : \mathcal{A} \longrightarrow \{ts2sut, sut2ts\}$.

## Deriving the test verdict

- The verdict of test $T$ w.r.t. $Sp = (D, P)$ and a particular SUT is a value in $\{pass, fail, inconclusive\}$.
- An algorithm to derive the test verdict on the fly.
- TEV: Hardware in the loop testing framework.

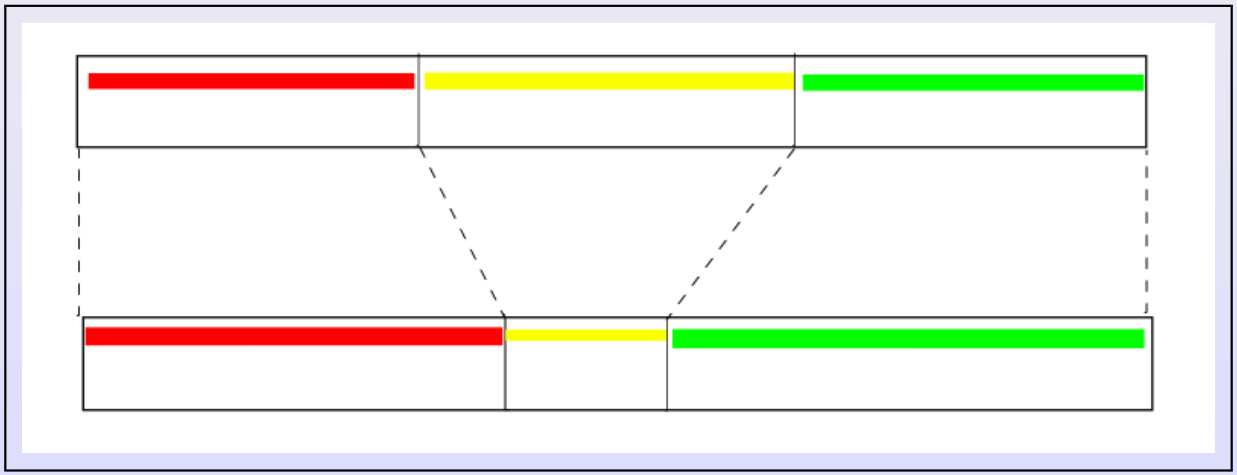
# Support for vertical development

# Well-behaved refinement

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

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

1. $colour(T) = green$ with respect to $Sp = (D, P)$ implies $colour(T) = green$ with respect to $Sp = (D', P')$, and
2. $colour(T) = red$ with respect to $Sp = (D, P)$ implies $colour(T) = red$ with respect to $Sp' = (D', P')$.



# Well-behaved refinement relations

## CSP-CASL refinement

$Sp = (D, P) \rightsquigarrow_{\mathcal{T},\mathcal{F},\mathcal{N},\mathcal{R}} Sp' = (D', P')$

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

# Well-behaved refinement relations

## CSP-CASL refinement

$Sp = (D, P) \rightsquigarrow_{\mathcal{T},\mathcal{F},\mathcal{N},\mathcal{R}} Sp' = (D', P')$

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

| Refinement relation | Well behaved |
|---|---|
| Data refinement$^{+}$ | $\surd$ |
| Process refinement over $\mathcal{T}$ model | $X$ |
| $^{*}$Process refinement over $\mathcal{F}$ model | $\surd$ |
| $^{*}$Process refinement over $\mathcal{N}$ model | $\surd$ |
| $^{*}$Process refinement over $\mathcal{R}$ model | $\surd$ |

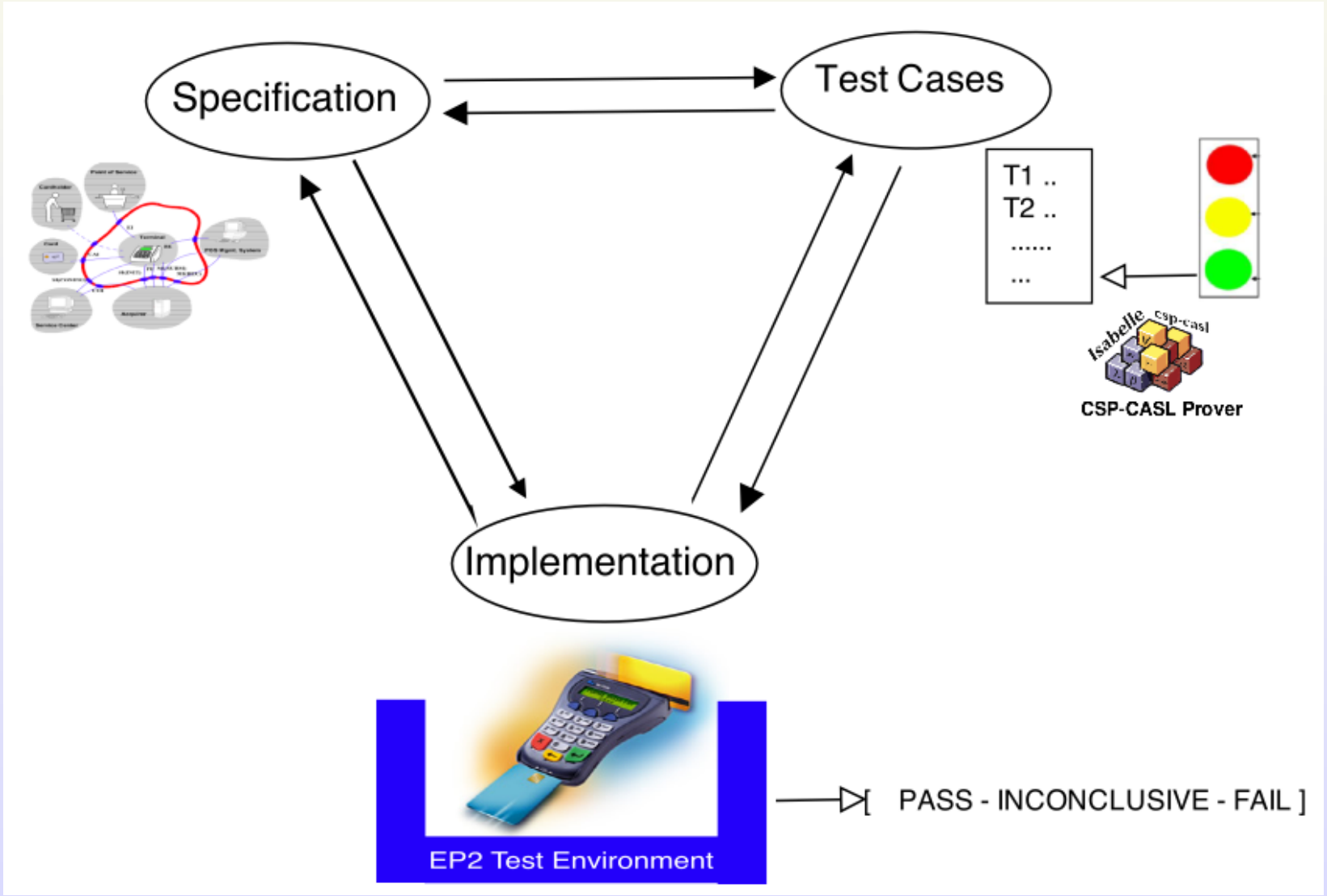
$^{+}$ it works with $\mathcal{F}$, $\mathcal{N}$, $\mathcal{R}$

$^{*}$ requires divergence-free processes.

# ... more on testing from CSP-CASL

- T. Kahsai, G. Holland, M. Roggenbach, B.-H. Schlinglof: **Towards formal testing of BR725 Rolls-Royce jet engine**. In CS&P 2009.
- T. Kahsai, M. Roggenbach, B.-H. Schlingloff: **Specification-based testing for software product lines**. In SEFM 2008. IEEE, 2008.
- T. Kahsai, M. Roggenbach, B.-H. Schlingloff: **Specification-based testing for refinement**. In SEFM 2007. IEEE, 2007.

# Summary


Specification
Test Cases
Implementation
T1 ..
T2 ..
.....
...
Isabelle csp-casl
CSP-CASL Prover
EP2 Test Environment
PASS - INCONCLUSIVE - FAIL ]

# Thanks!