Context & Motivation
Testing
Testing:

- **specification**
- **tester**
- **IUT**
Testing:

checking or measuring some quality characteristics of an executing object by performing experiments in a controlled way w.r.t. a specification
Formal Methods

Examples:
- Z
- Temporal Logic
- First order logic
- SDL (Specification and Description Language)
- LOTOS
- Promela
- Labelled Transition Systems
- Finite state machine
- Process algebra
- ....
Formal Methods

↩️ Use mathematics to model relevant parts of software

➡️ Examples:
- Z
- Temporal Logic
- First order logic
- SDL (Specification and Description Language)
- LOTOS
- Promela
- Labelled Transition Systems
- Finite state machine
- Process algebra
- ......
Formal Methods

- Use mathematics to model relevant parts of software

Examples:
- Z
- Temporal Logic
- First order logic
- SDL (Specification and Description Language)
- LOTOS
- Promela
- Labelled Transition Systems
- Finite state machine
- Process algebra
- ....
Formal Methods

- Use mathematics to model relevant parts of software

- Precise, formal semantics: no room for ambiguity or misinterpretation

Examples:
- Z
- Temporal Logic
- First order logic
- SDL (Specification and Description Language)
- LOTOS
- Promela
- Labelled Transition Systems
- Finite state machine
- Process algebra
- .......
Formal Methods

- Use mathematics to model relevant parts of software
- Precise, formal semantics: no room for ambiguity or misinterpretation

Examples:
- Z
- Temporal Logic
- First order logic
- SDL (Specification and Description Language)
- LOTOS
- Promela
- Labelled Transition Systems
- Finite state machine
- Process algebra
- ......
Formal Methods

- Use mathematics to model relevant parts of software
- Precise, formal semantics: no room for ambiguity or misinterpretation
- Allow formal validation and reasoning about systems

Examples:
- Z
- Temporal Logic
- First order logic
- SDL (Specification and Description Language)
- LOTOS
- Promela
- Labelled Transition Systems
- Finite state machine
- Process algebra
- ......
Formal Methods

- Use mathematics to model relevant parts of software

- Precise, formal semantics: no room for ambiguity or misinterpretation

- Allow formal validation and reasoning about systems

Examples:
- Z
- Temporal Logic
- First order logic
- SDL (Specification and Description Language)
- LOTOS
- Promela
- Labelled Transition Systems
- Finite state machine
- Process algebra
- ......
Formal Methods

- Use mathematics to model relevant parts of software
- Precise, formal semantics: no room for ambiguity or misinterpretation
- Allow formal validation and reasoning about systems
- Amenable to tools: automation

Examples:
- $Z$
- Temporal Logic
- First order logic
- SDL (Specification and Description Language)
- LOTOS
- Promela
- Labelled Transition Systems
- Finite state machine
- Process algebra
- ....
Software Testing
Software Testing

Testing is:

- important
- much practiced
- 30% - 50% of project effort
- expensive
- time critical
- not constructive (but sadistic?)
Software Testing

Testing is:

- important
- much practiced
- 30% - 50% of project effort
- expensive
- time critical
- not constructive (but sadistic?)

But also:

- ad-hoc, manual, error-prone
- hardly theory / research
- no attention in curricula
- not cool:
  “if you’re a bad programmer you might be a tester”
Software Testing

Testing is:
- important
- much practiced
- 30% - 50% of project effort
- expensive
- time critical
- not constructive (but sadistic?)

But also:
- ad-hoc, manual, error-prone
- hardly theory / research
- no attention in curricula
- not cool: “if you’re a bad programmer you might be a tester”

Attitude is changing:
- more awareness
- more professional
Software Testing

Testing is:

- important
- much practiced
- 30% - 50% of project effort
- expensive
- time critical
- not constructive (but sadistic?)

But also:

- ad-hoc, manual, error-prone
- hardly theory / research
- no attention in curricula
- not cool: “if you’re a bad programmer you might be a tester”

Attitude is changing:

- more awareness
- more professional

Improvements possible with formal methods!?
Types of Testing

- Level of detail
  - system
  - integration
  - module
  - unit

- Accessibility
  - portability
  - maintainability
  - efficiency
  - usability
  - reliability
  - functionality

Characteristics
Types of Testing

- **Level of detail**
  - system
  - integration
  - module
  - unit
  - portability
  - maintainability
  - efficiency
  - usability
  - reliability
  - functionality

- **Accessibility**
  - white box
  - black box

- **Characteristics**
  - usability
  - reliability
  - functionality

© Jan Tretmans and Julien Schmaltz

Embedded Systems INSTITUTE
A Model-Based Development Process

informal ideas

specification

design

code

realization
A Model-Based Development Process

- informal ideas
  - specification
  - design
  - code
- world of models
- real world
- informal world
- formalizable
A Model-Based Development Process

informal ideas

specification

design

code

realization

informal world

world of models

real world
A Model-Based Development Process

- informal ideas
  - specification
    - design
      - code
        - realization

 informal world

world of models

real world
A Model-Based Development Process

informal ideas → specification → design → code → realization

informal world

world of models

real world

validation

formal verification
A Model-Based Development Process

informal ideas → specification → design → code

validation → formal verification

world of models → testing

real world

© Jan Tretmans and Julien Schmaltz
A Model-Based Development Process

informal ideas

specification

validation

design

formal verification

code

world of models

real world

testing

realization

informal world

Jan Tretmans and Julien Schmaltz
A Model-Based Development Process

- informal ideas
  - informal world
  - validation
  - formal verification
  - world of models
- specification
- design
- code
- model-based testing
- realization
- real world
Verification and Testing
Verification and Testing

Verification:

- formal manipulation
- prove properties
- performed on model
Verification and Testing

Verification:
- formal manipulation
- prove properties
- performed on model

Testing:
- experimentation
- show error
- concrete system

formal world ———> concrete world
Verification and Testing

Verification:
- formal manipulation
- prove properties
- performed on model

Testing:
- experimentation
- show error
- concrete system

Verification is only as good as the validity of the model on which it is based.
Verification and Testing

Verification:
- formal manipulation
- prove properties
- performed on model

Testing:
- experimentation
- show error
- concrete system

Verification is only as good as the validity of the model on which it is based.

Testing can only show the presence of errors, not their absence.
Model-Based Testing
Model-Based Testing

Model based testing has potential to combine

- practice - testing
- theory - formal methods
Model-Based Testing

Model based testing has potential to combine

- practice - testing
- theory - formal methods

Model Based Testing:

- testing with respect to a (formal) model / specification
  state model, pre/post, CSP, Promela, UML, Spec#, ....

- promises better, faster, cheaper testing:
  - algorithmic generation of tests and test oracles: tools
  - formal and unambiguous basis for testing
  - measuring the completeness of tests
Testing & Formal Methods
A Perfect Couple?
Testing & Formal Methods

A Perfect Couple?

Formal methods:

- proving properties
- research
- sound theories
- “clean”
Testing & Formal Methods

A Perfect Couple?

Formal methods:
- proving properties
- research
- sound theories
- “clean”

Testing:
- trial & error
- practice
- heuristics
- “dirty hands”
Testing & Formal Methods

A Perfect Couple?

Formal methods:
- proving properties
- research
- sound theories
- “clean”

Testing:
- trial & error
- practice
- heuristics
- “dirty hands”

“Testing is not necessary after formal verification”
Testing & Formal Methods

A Perfect Couple?

Formal methods:
- proving properties
- research
- sound theories
- “clean”

Testing:
- trial & error
- practice
- heuristics
- “dirty hands”

“Testing is not necessary after formal verification”

“Testing can only detect the presence of errors, not their absence”
Testing & Formal Methods

A Perfect Couple?

Formal methods:
- proving properties
- research
- sound theories
- “clean”

Testing:
- trial & error
- practice
- heuristics
- “dirty hands”

“Testing is not necessary after formal verification”

“Testing can only detect the presence of errors, not their absence”

“Formal methods are toys for boys”
Testing & Formal Methods

A Perfect Couple?

Formal methods:
- proving properties
- research
- sound theories
- “clean”

Testing:
- trial & error
- practice
- heuristics
- “dirty hands”

“Testing is not necessary after formal verification”

“Testing can only detect the presence of errors, not their absence”

“Formal methods are toys for boys”

“Formal methods have extreme potential - but not for my project”
Testing with Formal Methods
Testing with Formal Methods

Testing with respect to a formal specification
Testing with Formal Methods

- Testing with respect to a formal specification
- Precise, formal definition of correctness:
  - good and unambiguous basis for testing
Testing with Formal Methods

- Testing with respect to a formal specification
- Precise, formal definition of correctness:
  - good and unambiguous basis for testing
- Formal validation of tests
Testing with Formal Methods

- Testing with respect to a formal specification
- Precise, formal definition of correctness:
  good and unambiguous basis for testing
- Formal validation of tests
- Algorithmic derivation of tests:
  tools for automatic test generation
Testing with Formal Methods

- Testing with respect to a formal specification
- Precise, formal definition of correctness:
  good and unambiguous basis for testing
- Formal validation of tests
- Algorithmic derivation of tests:
  tools for automatic test generation
- Allows to define measures expressing coverage
  and quality of testing
Testing & Formal Methods
Testing & Formal Methods

Claims:
Testing & Formal Methods

Claims:

- Combining the “mathematically clean” world of formal methods and the “dirty hands” world of testing
Testing & Formal Methods

Claims:

- Combining the “mathematically clean” world of formal methods and the “dirty hands” world of testing
- Testing and formal methods are both necessary in software development
Testing & Formal Methods

Claims:

- Combining the “mathematically clean” world of formal methods and the “dirty hands” world of testing
- Testing and formal methods are both necessary in software development
- Formal methods improve the testing process
Testing & Formal Methods

Claims:

- Combining the “mathematically clean” world of formal methods and the “dirty hands” world of testing
- Testing and formal methods are both necessary in software development
- Formal methods improve the testing process
- Formal testing stimulates the use of formal methods
Automated Model-Based Testing
Automated Model-Based Testing
Automated Model-Based Testing
Automated Model-Based Testing

test cases

test tool

IUT

© Jan Tretmans and Julien Schmaltz
Automated Model-Based Testing

- Test generation tool
- Model
- Test tool
- IUT
Automated Model-Based Testing

test generation tool

model

IUT

confto model

test tool

IUT
Automated Model-Based Testing

IUT \(\xrightarrow{conf\to}\) model

IUT passes tests
Automated Model-Based Testing

IUT $\rightarrow$ model

test generation tool

model

IUT $\leftarrow$ model

IUT $\rightarrow$ IUT

IUT passes tests

IUT $\rightarrow$ test tool

IUT $\leftarrow$ test tool
Automated Model-Based Testing

IUT \textit{conf} \textit{to} model

exhaustive $\uparrow \downarrow$ sound

IUT \textit{passes} tests

IUT \textit{conf} \textit{to} model

望去
Automated Model-Based Testing

IUT \( \text{conf} \) to model

exhaustive \( \uparrow \) sound

IUT \text{pass} \ \text{tests}

IUT

\text{model}

test
generation
tool

model

test
tool

IUT
Approaches to Model-Based Testing
Approaches to Model-Based Testing

Several modelling paradigms:
Approaches to Model-Based Testing

Several modelling paradigms:
Approaches to Model-Based Testing

Several modelling paradigms:

- Finite State Machine
Approaches to Model-Based Testing

Several modelling paradigms:

- Finite State Machine
Approaches to Model-Based Testing

Several modelling paradigms:

- Finite State Machine
- Pre/post-conditions
Approaches to Model-Based Testing

Several modelling paradigms:

- Finite State Machine
- Pre/post-conditions
Approaches to Model-Based Testing

Several modelling paradigms:

- Finite State Machine
- Pre/post-conditions
- Labelled Transition Systems
Approaches to Model-Based Testing

Several modelling paradigms:

- Finite State Machine
- Pre/post-conditions
- Labelled Transition Systems
Approaches to Model-Based Testing

Several modelling paradigms:

- Finite State Machine
- Pre/post-conditions
- Labelled Transition Systems
- Programs as Functions
Several modelling paradigms:

- Finite State Machine
- Pre/post-conditions
- Labelled Transition Systems
- Programs as Functions
Approaches to Model-Based Testing

Several modelling paradigms:

- Finite State Machine
- Pre/post-conditions
- Labelled Transition Systems
- Programs as Functions
- Abstract Data Type testing
Approaches to Model-Based Testing

Several modelling paradigms:

- Finite State Machine
- Pre/post-conditions
- Labelled Transition Systems
- Programs as Functions
- Abstract Data Type testing
Approaches to Model-Based Testing

Several modelling paradigms:

- Finite State Machine
- Pre/post-conditions
- Labelled Transition Systems
- Labelled Transition Systems
- Programs as Functions
- Abstract Data Type testing
Model Based Testing
Model Based Testing

IUT $\overset{\text{[model]}\to}{\overset{\text{exhaustive}}{\overset{\text{sound}}{\overset{\text{IUT passes tests}}{\overset{\text{pass fail}}{\overset{\text{test execution tool}}{\overset{\text{test generation tool}}{\overset{\text{model}}{\overset{\text{IUT}}{\text{confto}}}}}}}}} \overset{\text{IUT}}{\overset{\text{confto}}{\overset{\text{model}}}}$
Model Based Testing with Transition Systems

\( \text{gen} : \text{LTS} \rightarrow \wp(\text{TTS}) \)

\( i \parallel \text{gen}(s) \rightarrow \text{pass} \)

\( i \in \text{IOTS} \)

\( \text{Soundness} \uparrow \downarrow \text{Completeness} \)

\( i \parallel i \text{o} \)
Formal Testing
Formal Testing

specification $S$
Formal Testing

\[ S \] specification

\[ i \] implementation
Formal Testing

specification $S$

correctness criterion

implementation relation

implementation $i$
Formal Testing

correction criterion
implementation relation

specification $S$

test generation

test suite $T_S$

implementation $i$
Formal Testing

specification $S$

correctness criterion

implementation relation

implementation $i$

test generation

test suite $T_S$

test execution

pass / fail
Formal Testing

- Specification $S$
- Implementation $i$
- Correctness criterion
- Implementation relation
- Test generation
- Test suite $T_S$
- Test execution
- Pass / fail

$i$ conforms-to $S$

$i$ passes $T_S$
Formal Testing

- **specification** $S$
- **test generation**
- **test suite** $T_S$
- **test execution**
- **pass / fail**

**i conforms-to** $s$

**i passes** $T_S$
Formal Testing

- specification $S$
- implementation $i$
- test generation
- test suite $T_S$
- test execution
- pass / fail

$i$ conforms-to $s$

$i$ passes $T_S$
Model-Based Testing for LTS

Involves:
- specification model
- implementation IUT + models of IUTs
- correctness
- test cases
- test generation
- test execution
- test result analysis

pass / fail
Formal Testing with Transition Systems

\[ s \in \text{LTS} \]

\[ \text{gen} : \text{LTS} \rightarrow \mathcal{P} (\text{TTS}) \]

\[ T_s \subseteq \text{TTS} \]

\[ \text{passes} : \text{IOTS} \times \text{TTS} \rightarrow \{\text{pass}, \text{fail}\} \]

Test hypothesis:
\[ \forall \text{IUT} \in \text{IMP} . \exists i_{\text{IUT}} \in \text{IOTS} . \]
\[ \forall t \in \text{TTS} . \text{IUT passes} t \]
\[ \iff i_{\text{IUT}} \text{ passes} t \]

Proof soundness and exhaustiveness:
\[ \forall i \in \text{IOTS} . \]
\[ (\forall t \in \text{gen}(s) . i \text{ passes} t) \]
\[ \iff i \text{ ioco } s \]

\[ \text{pass / fail} \]
LTS with inputs and outputs
Models: Labelled Transition Systems

Labelled Transition System \( \langle S, L, T, s_0 \rangle \)

- **states**
- **actions**
- **transitions** \( T \subseteq S \times (L \cup \{\tau\}) \times S \)
- **initial state** \( s_0 \in S \)

Diagram:
- States: ?coin, !alarm, !coffee, ?button
- Transitions: ?coin -> !alarm, !coffee, ?button
Labelled Transition Systems
Labelled Transition Systems

L = { dub, kwart, coffee, tea, soup}
Labelled Transition Systems

\[ L = \{ \text{dub, kwart, coffee, tea, soup} \} \]
Labelled Transition Systems

\[ L = \{ \text{dub, kwart, coffee, tea, soup} \} \]
Labelled Transition Systems

\[ L = \{ \text{dub}, \text{kwart}, \text{coffee}, \text{tea}, \text{soup} \} \]
Labelled Transition Systems

$L = \{ \text{dub, kwart, coffee, tea, soup} \}$
**Labelled Transition Systems**

$L = \{ \text{dub, kwart, coffee, tea, soup} \}$

$LTS(L)$

all transition systems over $L$
Labelled Transition Systems

Diagram:

- States: S0, S1, S2, S3, S4
- Transitions:
  - S0 -> S1: dub -> coffee
  - S0 -> S2: dub -> tea
  - S1 -> S3: coffee
  - S2 -> S4: tea
Labelled Transition Systems

Sequences of observable actions:

\[
\text{traces } (s) = \{ \sigma \in L^* \mid s \xrightarrow{\sigma} \}
\]
Labelled Transition Systems

Sequences of observable actions:

\[ \text{traces}(s) = \{ \sigma \in L^* \mid s \xrightarrow{\sigma} \} \]

\[ \text{traces}(s) = \{ \varepsilon, \text{dub}, \text{dub coffee}, \text{dub tea} \} \]
Labelled Transition Systems

Sequences of observable actions:

\[ \text{traces}(s) = \{ \sigma \in L^* \mid s \xrightarrow{\sigma} \} \]

\[ \text{traces}(s) = \{ \varepsilon, \text{dub}, \text{dub coffee}, \text{dub tea} \} \]

Reachable states:

\[ s \text{ after } \sigma = \{ s' \mid s \xrightarrow{\sigma} s' \} \]
Labelled Transition Systems

Sequences of observable actions:

\[ \text{traces}(s) = \{ \sigma \in L^* \mid s \xrightarrow{\sigma} \} \]

Reachable states:

\[ s \text{ after } \sigma = \{ s' \mid s \xrightarrow{\sigma} s' \} \]

\[ s \text{ after } \text{dub} = \{ s_1, s_2 \} \]
Labelled Transition Systems

Sequences of observable actions:

$$\text{traces}(s) = \{ \sigma \in L^* \mid s \xrightarrow{\sigma} \}$$

$$\text{traces}(s) = \{ \epsilon, \text{dub}, \text{dub coffee}, \text{dub tea} \}$$

Reachable states:

$$s \after \sigma = \{ s' \mid s \xrightarrow{\sigma} s' \}$$

$$s \after \text{dub} = \{ s_1, s_2 \}$$

$$s \after \text{dub tea} = \{ s_4 \}$$
Input-Output Transition Systems
Input-Output Transition Systems

dub, kwart

coffee, tea

S0

S1

S3

dub

S2

S4

kwar

coffee

tea
Input-Output Transition Systems

dub, kwart

coffee, tea
Input-Output Transition Systems

dub, kwart  
coffee, tea

from user to machine initiative with user
machine cannot refuse

from machine to user initiative with machine
user cannot refuse
Input-Output Transition Systems

dub, kwart

coffee, tea

from user to machine initiative with user
machine cannot refuse

from machine to user initiative with machine
user cannot refuse
Input-Output Transition Systems

- dub, kwart
- coffee, tea

from user to machine
initiative with user
machine cannot refuse

from machine to user
initiative with machine
user cannot refuse

S0

S0

S1

S2

S3

S4

dub, kwart

coffee, tea

input

output

L_I

L_U

© Jan Tretmans and Julien Schmaltz
Input-Output Transition Systems

dub, kwart
coffee, tea

from user to machine
initiative with user
machine cannot refuse

from machine to user
initiative with machine
user cannot refuse

input $L_I$

output $L_U$
Input-Output Transition Systems

- **Input** $L_I$
  - dub
  - kwart
- **Output** $L_U$
  - coffee
  - tea

From user to machine:
- Initiative with user:
  - Machine cannot refuse
From machine to user:
- Initiative with machine:
  - User cannot refuse

$L_I \cap L_U = \emptyset$

$L_I \cup L_U = L$
**Input-Output Transition Systems**

- **Input** ($L_I$): $\{ ?\text{dub}, ?\text{kwart} \}$
- **Output** ($L_U$): $\{ !\text{coffee}, !\text{tea} \}$

**Relationships:**
- $L_I \cap L_U = \emptyset$
- $L_I \cup L_U = L$

- From user to machine:
  - Initiative with user
  - Machine cannot refuse

- From machine to user:
  - Initiative with machine
  - User cannot refuse
Input-Output Transition Systems

\[ \begin{align*}
L_I &= \{ \text{?dub, ?kwart} \} \\
L_U &= \{ \text{!coffee, !tea} \}
\end{align*} \]
Input-Output Transition Systems

\[ L_I = \{ \text{?dub, ?kwart} \} \]

\[ L_U = \{ \text{!coffee, !tea} \} \]

\[ \text{IOTS (} L_I, L_U \text{)} \subseteq \text{LTS (} L_I \cup L_U \text{)} \]
Input-Output Transition Systems

\[ L_I = \{ \texttt{?dub, ?kwart} \} \]
\[ L_U = \{ \texttt{!coffee, !tea} \} \]

\[ \text{IOTS} (L_I, L_U) \subseteq \text{LTS} (L_I \cup L_U) \]
Input-Output Transition Systems

IOTS is LTS with Input-Output and always enabled inputs:

\[ L_I = \{ ?dub, ?kwart \} \]

\[ L_U = \{ !coffee, !tea \} \]
Input-Output Transition Systems

$IOTS (L_I, L_U) \subseteq LTS (L_I \cup L_U)$

$IOTS$ is LTS with Input-Output and always enabled inputs:

$L_I = \{ ?dub, ?kwart \}$

$L_U = \{ !coffee, !tea \}$
**Input-Output Transition Systems**

\[ L_I = \{ ?\text{dub}, ?\text{kwart} \} \]

\[ L_U = \{ !\text{coffee}, !\text{tea} \} \]

\[ \text{IOTS} (L_I, L_U) \subseteq \text{LTS} (L_I \cup L_U) \]

IOTS is LTS with Input-Output and *always enabled inputs*:

for all states \( s \),

for all inputs \( ?a \in L_I \):

\[ S \xrightarrow{?a} \]
Input-Output Transition Systems

$IOTS (L_I, L_U) \subseteq LTS (L_I \cup L_U)$

$IOTS$ is $LTS$ with Input-Output and always enabled inputs:

for all states $s$,

for all inputs $?a \in L_I : S \xrightarrow{?a}$

$L_I = \{ ?dub, ?kwart \}$

$L_U = \{ !coffee, !tea \}$
Input-Output Transition Systems
Input-Output Transition Systems
Input-Output Transition Systems
Formal Testing with Transition Systems

\( s \in \text{LTS} \)

\[ \text{gen} : \text{LTS} \rightarrow \wp(\text{TTS}) \]

\( T_s \subseteq \text{TTS} \)

\( i_{\text{IUT}} \in \text{IOTS} \)

\( \text{passes} : \text{IOTS} \times \text{TTS} \rightarrow \{\text{pass}, \text{fail}\} \)

Test hypothesis:
\[ \forall \text{IUT} \in \text{IMPS} . \exists i_{\text{IUT}} \in \text{IOTS} . \forall t \in \text{TTS} . \text{IUT passes } t \iff i_{\text{IUT}} \text{ passes } t \]

Proof soundness and exhaustiveness:
\[ \forall i \in \text{IOTS} . (\forall t \in \text{gen}(s) . i \text{ passes } t) \iff i \text{ ioco } s \]

\( \text{pass} / \text{fail} \)
A conformance relation
Implementation Relation \textit{ioco}

Correctness expressed by implementation relation \textit{ioco}:

\[
\text{i \textit{ioco} } s \; =_{\text{def}} \; \forall \sigma \in \text{Straces } (s) : \; \text{out } (i \; \text{after} \; \sigma) \subseteq \text{out } (s \; \text{after} \; \sigma)
\]
Implementation Relation \texttt{ioco}

Correctness expressed by implementation relation \texttt{ioco}:

\[
i \texttt{ioco} \; s \; =_{\text{def}} \; \forall \sigma \in \text{Straces} \; (s) : \; \text{out} \; (i \; \text{after} \; \sigma) \subseteq \text{out} \; (s \; \text{after} \; \sigma)
\]

Intuition:

\texttt{i ioco}-conforms to \texttt{s}, iff

\begin{itemize}
  \item if \texttt{i} produces output \texttt{x} after trace \texttt{\sigma},
  then \texttt{s} can produce \texttt{x} after \texttt{\sigma}
  \item if \texttt{i} cannot produce any output after trace \texttt{\sigma},
  then \texttt{s} cannot produce any output after \texttt{\sigma} (quiescence \texttt{\delta})
\end{itemize}
Implementation Relation \textit{ioco}

Correctness expressed by implementation relation \textit{ioco}:

\[ i \textit{ioco} s \overset{\text{def}}{=} \forall \sigma \in \text{Straces}(s) : \text{out}(i \text{ after } \sigma) \subseteq \text{out}(s \text{ after } \sigma) \]
Implementation Relation \text{ ioco }

Correctness expressed by implementation relation \text{ioco}:

$$i\ ioco\ s =_{def} \forall \sigma \in \text{Straces}\ (s):\ \text{out}\ (i\ \text{after}\ \sigma) \subseteq \text{out}\ (s\ \text{after}\ \sigma)$$

\[
p \xrightarrow{\delta} p = \forall !x \in \text{L}_U \cup \{\tau\}. \quad p \xrightarrow{!x}
\]
Implementation Relation $\text{ioco}$

Correctness expressed by implementation relation $\text{ioco}$:

$$i \text{ ioco } s \overset{\text{def}}{=} \forall \sigma \in \text{Straces (s)} : \text{out (i after } \sigma) \subseteq \text{out (s after } \sigma)$$

$$p \xrightarrow{\delta} p = \forall !x \in L \cup \{\tau\} . \ p \xrightarrow{!x}$$

$$\text{Straces (s) } = \{ \sigma \in (L \cup \{\delta\})^* | s \xrightarrow{\sigma} \}$$
Implementation Relation \( i\text{oco} \)

Correctness expressed by implementation relation \( i\text{oco} \):

\[
i\text{oco} \; s =_{\text{def}} \forall \sigma \in \text{Straces}(s) : \text{out}(i \text{ after } \sigma) \subseteq \text{out}(s \text{ after } \sigma)
\]

\[
p \xrightarrow{\delta} p = \forall !x \in L \cup \{\tau\} . \quad p \xrightarrow{!x}
\]

\[
\text{Straces}(s) = \{ \sigma \in (L \cup \{\delta\})^* | s \xrightarrow{\sigma} \}
\]

\[
p \text{ after } \sigma = \{ p' | p \xrightarrow{\sigma} p' \}
\]
Implementation Relation \textbf{ioco}

Correctness expressed by implementation relation \textbf{ioco}:

\[ \text{i ioco } s \overset{\text{def}}{=} \forall \sigma \in \text{Straces } (s) : \text{ out } (\text{i after } \sigma) \subseteq \text{ out } (s \text{ after } \sigma) \]

\begin{align*}
\text{Straces } (s) &= \{ \sigma \in (L \cup \{\delta\})^* \mid s \xrightarrow{\sigma} \} \\
\text{p after } \sigma &= \{ p' \mid p \xrightarrow{\sigma} p' \} \\
\text{out } (P) &= \{ !x \in L \cup \{\tau\} \mid p \xrightarrow{!x}, p \in P \} \cup \{ \delta \mid p \xrightarrow{\delta} p, p \in P \}
\end{align*}
Implementation Relation \textit{ioco}

\[ \text{i ioco } s \ =_{\text{def}} \ \forall \sigma \in \text{Straces}(s) : \text{out}(i \ \text{after} \ \sigma) \subseteq \text{out}(s \ \text{after} \ \sigma) \]
Implementation Relation \( \text{ioco} \)

\[
\text{i oco } s \triangleq \forall \sigma \in \text{Straces}(s) : \text{out}(\text{i after } \sigma) \subseteq \text{out}(s \text{ after } \sigma)
\]
Implementation Relation \textit{ioco}

\[ \text{i oco } s =_{\text{def}} \forall \sigma \in \text{Straces } (s) : \text{ out } (\text{i after } \sigma) \subseteq \text{ out } (s \text{ after } \sigma) \]
**Implementation Relation  \ido\co\o**

\[ \ido\co\o s \quad =_{\text{def}} \quad \forall \sigma \in \text{Straces}(s) : \text{out}(i\after \sigma) \subseteq \text{out}(s\after \sigma) \]

\[
\begin{align*}
\text{out}(i\after \epsilon) &= \\
\text{out}(i\after ?\text{dub}) &= \\
\text{out}(i\after ?\text{dub}.?\text{dub}) &= \\
\text{out}(i\after ?\text{dub}.!\text{coffee}) &= \\
\text{out}(i\after ?\text{kwart}) &= \\
\text{out}(i\after !\text{coffee}) &= \\
\text{out}(i\after ?\text{dub}.!\text{tea}) &= \\
\text{out}(i\after \delta) &=
\end{align*}
\]
Implementation Relation \textit{ioco}

\begin{align*}
i \textit{ioco } s & = \text{def } \forall \sigma \in \text{Straces }(s) : \text{out } (i \text{ after } \sigma) \subseteq \text{out } (s \text{ after } \sigma) \\
\text{out } (i \text{ after } \varepsilon ) & = \{ \delta \} \\
\text{out } (i \text{ after } ?dub ) & = \\
\text{out } (i \text{ after } ?dub.?dub ) & = \\
\text{out } (i \text{ after } ?dub.!coffee) & = \\
\text{out } (i \text{ after } ?kwart ) & = \\
\text{out } (i \text{ after } !coffee) & = \\
\text{out } (i \text{ after } ?dub.!tea ) & = \\
\text{out } (i \text{ after } \delta ) & =
\end{align*}
Implementation Relation \( \ioco \)

\[
i \ioco s \doteq \forall \sigma \in \text{Straces}(s) : \text{out}(i \text{ after } \sigma) \subseteq \text{out}(s \text{ after } \sigma)
\]

\[
\begin{align*}
\text{out}(i \text{ after } \varepsilon) &= \{ \delta \} \\
\text{out}(i \text{ after } \text{?dub}) &= \{ \text{!coffee} \} \\
\text{out}(i \text{ after } \text{?dub.?dub}) &= \\
\text{out}(i \text{ after } \text{?dub.!coffee}) &= \\
\text{out}(i \text{ after } \text{?kwart}) &= \\
\text{out}(i \text{ after } \text{!coffee}) &= \\
\text{out}(i \text{ after } \text{?dub.!tea}) &= \\
\text{out}(i \text{ after } \delta) &= 
\end{align*}
\]
Implementation Relation \textit{ioco}

\[ \textit{i ioco s} \overset{\text{def}}{=} \forall \sigma \in \text{Straces} (s) : \text{out} (i \text{ after } \sigma) \subseteq \text{out} (s \text{ after } \sigma) \]

\[ \begin{align*}
\text{out} (i \text{ after } \varepsilon) &= \{ \delta \} \\
\text{out} (i \text{ after } ?dub) &= \{ !\text{coffee} \} \\
\text{out} (i \text{ after } ?dub.?dub) &= \{ !\text{coffee} \} \\
\text{out} (i \text{ after } ?dub.!\text{coffee}) &= \\
\text{out} (i \text{ after } ?\text{kwart}) &= \\
\text{out} (i \text{ after } !\text{coffee}) &= \\
\text{out} (i \text{ after } ?dub.!\text{tea}) &= \\
\text{out} (i \text{ after } \delta) &= 
\end{align*} \]
Implementation Relation $\mathbf{ioco}$

$$\mathbf{ioco} \ s \ = \ \text{def} \ \forall \sigma \in \text{Straces} (s) : \ \text{out} \ (i \ \text{after} \ \sigma) \subseteq \ \text{out} \ (s \ \text{after} \ \sigma)$$

\[
\begin{align*}
\text{out} \ (i \ \text{after} \ \epsilon) &= \ \{ \delta \} \\
\text{out} \ (i \ \text{after} \ \text{?dub}) &= \ \{ \text{!coffee} \} \\
\text{out} \ (i \ \text{after} \ \text{?dub.?dub}) &= \ \{ \text{!coffee} \} \\
\text{out} \ (i \ \text{after} \ \text{?dub.?kwart}) &= \ \{ \delta \} \\
\text{out} \ (i \ \text{after} \ \text{?kwart}) &= \ \\
\text{out} \ (i \ \text{after} \ \text{!coffee}) &= \ \\
\text{out} \ (i \ \text{after} \ \delta) &= \ \\
\end{align*}
\]
Implementation Relation $\text{ioco}$

$$\text{i ioco } s = \text{def } \forall \sigma \in \text{Straces } (s) : \text{ out (i after } \sigma) \subseteq \text{ out (s after } \sigma)$$

- $\text{out (i after } \varepsilon) = \{ \delta \}$
- $\text{out (i after ?dub)} = \{ !\text{coffee} \}$
- $\text{out (i after ?dub.?dub)} = \{ !\text{coffee} \}$
- $\text{out (i after ?dub.!coffee)} = \{ \delta \}$
- $\text{out (i after ?kwart)} = \{ \delta \}$
- $\text{out (i after !coffee)} = \{ \}$
- $\text{out (i after ?dub.!tea)} = \{ \}$
- $\text{out (i after } \delta) = \{ \}$
Implementation Relation $ioco$

$i \ io co \ s \ = \ _{def} \ \forall \sigma \in \ Straces \ (s) : \ out \ (i \ after \ \sigma) \subseteq out \ (s \ after \ \sigma)$

\[
\begin{align*}
out \ (i \ after \ \epsilon) & = \ \{\ \delta\ \} \\
out \ (i \ after \ ?dub) & = \ \{!coffee\} \\
out \ (i \ after \ ?dub.?dub) & = \ \{!coffee\} \\
out \ (i \ after \ ?dub.!coffee) & = \ \{\ \delta\ \} \\
out \ (i \ after \ ?kwart) & = \ \{\ \delta\ \} \\
out \ (i \ after \ !coffee) & = \ \emptyset \\
out \ (i \ after \ ?dub.!tea) & = \\
out \ (i \ after \ \delta) & = \\
\end{align*}
\]
Implementation Relation \( \text{ioco} \)

\[
i \; \text{ioco} \; s \; =_{\text{def}} \; \forall \sigma \in \text{Straces}(s) : \; \text{out}(i \; \text{after} \; \sigma) \subseteq \text{out}(s \; \text{after} \; \sigma)
\]

\[
\begin{align*}
\text{out}(i \; \text{after} \; \epsilon) & = \{ \delta \} \\
\text{out}(i \; \text{after} \; ?\text{dub}) & = \{ !\text{coffee} \} \\
\text{out}(i \; \text{after} \; ?\text{dub}.?\text{dub}) & = \{ !\text{coffee} \} \\
\text{out}(i \; \text{after} \; ?\text{dub}!.\text{coffee}) & = \{ \delta \} \\
\text{out}(i \; \text{after} \; ?\text{kwart}) & = \{ \delta \} \\
\text{out}(i \; \text{after} \; !\text{coffee}) & = \emptyset \\
\text{out}(i \; \text{after} \; ?\text{dub}!.\text{tea}) & = \emptyset \\
\text{out}(i \; \text{after} \; \delta) & = \emptyset
\end{align*}
\]
Implementation Relation $\text{ioco}$

$$\text{i ioco } s =_{\text{def}} \forall \sigma \in \text{Straces}(s) : \text{out}(\text{i after } \sigma) \subseteq \text{out}(s \text{ after } \sigma)$$

- $\text{out}(\text{i after } \varepsilon) = \{ \delta \}$
- $\text{out}(\text{i after } \text{?dub}) = \{ !\text{coffee} \}$
- $\text{out}(\text{i after } \text{?dub.?dub}) = \{ !\text{coffee} \}$
- $\text{out}(\text{i after } \text{?dub!coffee}) = \{ \delta \}$
- $\text{out}(\text{i after } \text{?kwart}) = \{ \delta \}$
- $\text{out}(\text{i after } !\text{coffee}) = \emptyset$
- $\text{out}(\text{i after } \text{?dub!tea}) = \emptyset$
- $\text{out}(\text{i after } \delta) = \{ \delta \}$
Implementation Relation $\mathit{ioCO}$

\[
i_{\mathit{ioCO}} s \overset{\text{def}}{=} \forall \sigma \in \text{Straces}(s) : \text{out}(i \after \sigma) \subseteq \text{out}(s \after \sigma)
\]
Implementation Relation \( \text{ioco} \)

\[
\text{i ioco } s \quad \overset{\text{def}}{=} \quad \forall \sigma \in \text{Straces} (s) : \quad \text{out } (\text{i after } \sigma) \subseteq \text{out } (s \text{ after } \sigma)
\]

\[
\text{out } (\text{i after } \varepsilon) = \{ \delta \}
\]
Implementation Relation $\text{ioco}$

\[
i \text{ioco } s \overset{\text{def}}{=} \forall \sigma \in \text{Straces}(s): \text{out}(i \text{ after } \sigma) \subseteq \text{out}(s \text{ after } \sigma)
\]

\[
\text{out}(i \text{ after } \epsilon) = \{ \delta \}
\]

\[
\text{out}(i \text{ after } ?\text{dub}) = \{ !\text{coffee} \}
\]
Implementation Relation \texttt{ioco}

\[ i \texttt{ioco} s \;=\; \text{def} \; \forall \sigma \in \text{Straces}(s) : \text{out}(i \text{ after } \sigma) \subseteq \text{out}(s \text{ after } \sigma) \]

\begin{align*}
\text{out}(i \text{ after } \varepsilon) & = \{ \delta \} \\
\text{out}(i \text{ after } ?\text{dub}) & = \{ !\text{coffee} \} \\
\text{out}(i \text{ after } ?\text{dub}.!\text{coffee}) & = \{ \delta \}
\end{align*}
**Implementation Relation ioco**

\[ i \, \text{ioco} \, s \, = \, \text{def} \, \forall \sigma \in \text{Straces}(s) : \, \text{out}(i \, \text{after} \, \sigma) \subseteq \text{out}(s \, \text{after} \, \sigma) \]

\[
\begin{align*}
\text{out}(i \, \text{after} \, \varepsilon) &= \{ \delta \} \\
\text{out}(i \, \text{after} \, \text{?dub}) &= \{ \text{!coffee} \} \\
\text{out}(i \, \text{after} \, \text{?dub!coffee}) &= \{ \delta \}
\end{align*}
\]
Implementation Relation  \( \text{ioco} \)

\[
i \text{ioco} s \quad \overset{\text{def}}{=} \quad \forall \sigma \in \text{Straces}(s) : \quad \text{out}(i \text{ after } \sigma) \subseteq \text{out}(s \text{ after } \sigma)
\]

\[
\begin{align*}
\text{out}(i \text{ after } \varepsilon) &= \{ \delta \} \\
\text{out}(i \text{ after } ?\text{dub}) &= \{ !\text{coffee} \} \\
\text{out}(i \text{ after } ?\text{dub}.!\text{coffee}) &= \{ \delta \} \\
\text{out}(s \text{ after } \varepsilon) &= \{ \delta \} \\
\text{out}(s \text{ after } ?\text{dub}) &= \{ !\text{coffee} \}
\end{align*}
\]
Implementation Relation \( \text{ioco} \)

\[
\text{i oco } s \overset{\text{def}}{=} \forall \sigma \in \text{Straces}(s) : \text{out}(i \text{ after } \sigma) \subseteq \text{out}(s \text{ after } \sigma)
\]

\[
\begin{align*}
\text{out}(i \text{ after } \varepsilon) &= \{ \delta \} \\
\text{out}(i \text{ after } \text{?dub}) &= \{ \!\text{coffee} \} \\
\text{out}(i \text{ after } \text{?dub!coffee}) &= \{ \delta \}
\end{align*}
\]

\[
\begin{align*}
\text{out}(s \text{ after } \varepsilon) &= \{ \delta \} \\
\text{out}(s \text{ after } \text{?dub}) &= \{ \!\text{coffee} \} \\
\text{out}(s \text{ after } \text{?dub!coffee}) &= \{ \delta \}
\end{align*}
\]
Implementation Relation $\text{ioco}$

$$\text{i ioco s} = \text{def } \forall \sigma \in \text{Straces (s)} : \text{out (i after } \sigma) \subseteq \text{out (s after } \sigma)$$

```
\begin{align*}
\text{out (i after } \varepsilon) &= \{ \delta \} \\
\text{out (i after } \text{?dub}) &= \{ !\text{coffee} \} \\
\text{out (i after } \text{?dub}.!\text{coffee}) &= \{ \delta \}
\end{align*}
```

```
\begin{align*}
\text{out (s after } \varepsilon) &= \{ \delta \} \\
\text{out (s after } \text{?dub}) &= \{ !\text{coffee} \} \\
\text{out (s after } \text{?dub}.!\text{coffee}) &= \{ \delta \}
\end{align*}
```
Implementation Relation \textbf{ioco}

\[ i \text{ io} \text{c} o \ s \ =_{\text{def}} \forall \sigma \in \text{Straces}(s): \text{out}(i \text{ after } \sigma) \subseteq \text{out}(s \text{ after } \sigma) \]
Implementation Relation \textit{ioco}

\[ i \ ioco \ s \quad := \quad \forall \sigma \in \text{Straces}(s) : \quad \text{out}(i \ \text{after} \ \sigma) \subseteq \text{out}(s \ \text{after} \ \sigma) \]

\[ \text{out}(i \ \text{after} \ ?\text{dub}) = \{ \text{!coffee} \} \]
Implementation Relation \( \textit{ioco} \)

\[
i \textit{ioco} s \quad =_{\text{def}} \quad \forall \sigma \in \text{Straces}(s) : \quad \text{out} (i \text{ after } \sigma) \subseteq \text{out} (s \text{ after } \sigma)
\]

\[
\text{out} (i \text{ after } ?\text{dub}) = \{ !\text{coffee} \}
\]

\[
\text{out} (s \text{ after } ?\text{dub}) = \{ !\text{coffee}, !\text{tea} \}
\]
Implementation Relation \textit{ioco}

\[
i \text{ ioco } s \ = \ \text{def} \ \forall \sigma \in \text{Straces}(s) : \ \text{out}(i \ \text{after} \ \sigma) \subseteq \ \text{out}(s \ \text{after} \ \sigma)
\]

\[
\text{out}(i \ \text{after} \ ?dub) \ = \ \{!\text{coffee}\} \quad \quad \quad \quad \quad \quad \text{out}(s \ \text{after} \ ?dub) \ = \ \{!\text{coffee}, !\text{tea}\}
\]
Implementation Relation \textit{ioco}

\[ i \text{ \textit{ioco}} s \overset{\text{def}}{=} \forall \sigma \in \text{Straces}(s): \text{out}(i \text{ after } \sigma) \subseteq \text{out}(s \text{ after } \sigma) \]
Implementation Relation \( \text{ioco} \)

\[
i \text{ioco} s \quad = \quad \text{def} \quad \forall \sigma \in \text{Straces} (s) : \quad \text{out} (i \text{ after } \sigma) \subseteq \text{out} (s \text{ after } \sigma)
\]

\[
\text{out} (i \text{ after } ?dub) = \{ !\text{coffee}, !\text{tea} \} \quad \not\subset \quad \text{out} (s \text{ after } ?dub) = \{ !\text{coffee} \}
\]
Implementation Relation $\text{ioco}$

$$i \text{ioco} \ s \ =_{\text{def}} \ \forall \sigma \in \text{Straces} (s) : \ \text{out} (i \ \text{after} \ \sigma) \subseteq \text{out} (s \ \text{after} \ \sigma)$$

$$\text{out} (i \ \text{after} \ ?dub) = \{ !\text{coffee}, !\text{tea} \}$$

$$\nsubseteq$$

$$\text{out} (s \ \text{after} \ ?dub) = \{ !\text{coffee} \}$$
Implementation Relation $\text{ioco}$

\[
i \text{ioco } s \triangleq \forall \sigma \in \text{Straces}(s) : \text{out}(i \text{ after } \sigma) \subseteq \text{out}(s \text{ after } \sigma)
\]

Diagram:

- **i**
  - Question mark (dub)
  - Exclamation mark (tea)
  - Exclamation mark (coffee)

- **s**
  - Question mark (dub)
  - Exclamation mark (tea)
  - Exclamation mark (coffee)
Implementation Relation $\text{ioco}$

$$\text{i oco } s \overset{\text{def}}{=} \forall \sigma \in \text{Straces}(s) : \text{out } (\text{i after } \sigma) \subseteq \text{out } (s \text{ after } \sigma)$$

$$\text{out } (\text{i after } ?\text{dub}) = \{ !\text{coffee}, !\text{tea} \} \quad \text{out } (s \text{ after } ?\text{dub}) = \{ !\text{coffee}, !\text{tea} \}$$
Implementation Relation \texttt{ioco}

\[
i \texttt{ioco } s \ =_{\text{def}} \ \forall \sigma \in \text{Straces } (s) : \ out \ (i \text{ after } \sigma) \subseteq out \ (s \text{ after } \sigma)
\]

\[
\text{out } (i \text{ after } ?dub) = \{ !\text{coffee}, !\text{tea} \} \quad \text{out } (s \text{ after } ?dub) = \{ !\text{coffee}, !\text{tea} \}
\]
Implementation Relation $\text{ioco}$

$$i \text{ ioco } s \, = \, \text{def} \, \forall \sigma \in \text{Straces}(s) : \text{out}(i \text{ after } \sigma) \subseteq \text{out}(s \text{ after } \sigma)$$
Implementation Relation $ioco$

$$i \ xoco \ s \ \overset{def}{=} \forall \sigma \in \text{Straces} (s) : \text{out} (i \ \text{after} \ \sigma) \subseteq \text{out} (s \ \text{after} \ \sigma)$$

out ($i$ after ?dub) = \{ !coffee \}

out ($i$ after ?kwart) = \{ !tea \}
Implementation Relation \texttt{ioco}

\[ i \text{ ioco } s = \text{def } \forall \sigma \in \text{Straces}(s): \text{out (i after } \sigma) \subseteq \text{out (s after } \sigma) \]

\textbf{i}

\[ \text{out (i after } \texttt{dub}) = \{ \texttt{!coffee} \} \]
\[ \text{out (i after } \texttt{kwart}) = \{ \texttt{!tea} \} \]

\textbf{s}

\[ \text{out (s after } \texttt{dub}) = \{ \texttt{!coffee} \} \]
\[ \text{out (s after } \texttt{kwart}) = \emptyset \]
Implementation Relation $\mathcal{IOCO}$

$$\mathcal{i} \mathcal{IOCO} \mathcal{s} \overset{\text{def}}{=} \forall \sigma \in \text{Straces}(s) : \text{out}(i \text{ after } \sigma) \subseteq \text{out}(s \text{ after } \sigma)$$

\[
\begin{align*}
\text{out}(i \text{ after } ?\text{dub}) & = \{ !\text{coffee} \} \\
\text{out}(i \text{ after } ?\text{kwart}) & = \{ !\text{tea} \}
\end{align*}
\]

But $?\text{kwart} \notin \text{Straces}(s)$
**Implementation Relation \textsc{ioco}**

\[
i \text{ioco} s = \text{def} \quad \forall \sigma \in \text{Straces}(s) : \quad \text{out (i after } \sigma) \subseteq \text{out (s after } \sigma)\]

\[
\begin{align*}
\text{out (i after } ?\text{dub}) &= \{ \lnot \text{coffee} \} \\
\text{out (i after } ?\text{kwart}) &= \{ \lnot \text{tea} \} \\
\text{out (s after } ?\text{dub}) &= \{ \lnot \text{coffee} \} \\
\text{out (s after } ?\text{kwart}) &= \emptyset
\end{align*}
\]

But \( ?\text{kwart} \notin \text{Straces}(s) \)
**Implementation Relation**  \( \text{ioco} \)

\[
i \text{ioco} \ s \ = \begin{array}{c}
\text{def} & \forall \sigma \in \text{Straces} \ (s) : \text{out} \ (i_{\text{after}} \ \sigma) \subseteq \text{out} \ (s_{\text{after}} \ \sigma)
\end{array}
\]

\[
i \quad \quad \quad \quad \text{S}
\]

\[
?\text{dub} \quad \quad \quad \quad ?\text{kwart}
\]

\[
?\text{dub} \quad \quad \quad \quad ?\text{kwart}
\]

\[
?\text{dub} \quad \quad \quad \quad ?\text{kwart}
\]

\[
!\text{coffee} \quad \quad \quad \quad !\text{tea}
\]

\[
!\text{coffee} \quad \quad \quad \quad !\text{tea}
\]
Implementation Relation \textit{ioco}

\[ \text{i \ ioco \ s} = \text{def} \ \forall \sigma \in \text{Straces} (s) : \ \text{out (i after } \sigma) \subseteq \text{out (s after } \sigma) \]

\text{out (i after ?dub)} = \{ !\text{coffee} \}
\text{out (i after ?kwart)} = \{ !\text{tea} \}
\text{out (s after ?dub)} = \{ !\text{coffee} \}
\text{out (s after ?kwart)} = \{ !\text{tea} \}
Implementation Relation \textit{ioco}

\[
i\text{ioco} s \overset{\text{def}}{=} \forall \sigma \in \text{Straces} (s) : \text{out} (i \text{ after } \sigma) \subseteq \text{out} (s \text{ after } \sigma)
\]

\[
\begin{align*}
\text{out} (i \text{ after } ?dub) & = \{ !\text{coffee} \} \\
\text{out} (i \text{ after } ?kwart) & = \{ !\text{tea} \}
\end{align*}
\]

\[
\begin{align*}
\text{out} (s \text{ after } ?dub) & = \{ !\text{coffee} \} \\
\text{out} (s \text{ after } ?kwart) & = \{ !\text{tea} \}
\end{align*}
\]
Implementation Relation \textit{ioco}

\[ \text{i \textit{ioco} s \overset{\text{def}}{=} \forall \sigma \in \text{Straces} (s) : \text{out} (\text{i \text{after} } \sigma) \subseteq \text{out} (s \text{after} \sigma) \]
Implementation Relation $\text{ioco}$

\[
i\text{ ioco } s \overset{\text{def}}{=} \forall \sigma \in \text{Straces}(s) : \text{out}(i\text{ after }\sigma) \subseteq \text{out}(s\text{ after }\sigma)
\]

\[
\text{out}(i\text{ after }\text{?kwart}) = \{\delta\}
\]

\[
\text{out}(s\text{ after }\text{?kwart}) = \{!\text{tea}\}
\]
Implementation Relation \( \text{ioco} \)

\[
\text{i ioco } s = \text{def } \forall \sigma \in \text{Straces}(s) : \text{out } (i \text{ after } \sigma) \subseteq \text{out } (s \text{ after } \sigma)
\]

\[
\text{out } (i \text{ after } ?\text{kwart}) = \{ \delta \} \quad \text{out } (s \text{ after } ?\text{kwart}) = \{ !\text{tea} \}
\]
Implementation Relation $\text{ioco}$

$$\text{ioco } s \overset{\text{def}}{=} \forall \sigma \in \text{Straces}(s) : \text{out}(\text{i after } \sigma) \subseteq \text{out}(s \text{ after } \sigma)$$
Implementation Relation \texttt{ioco}

\[
i \texttt{ioco } s \overset{\text{def}}{=} \forall \sigma \in \text{Straces}(s) : \text{out}(i \texttt{ after } \sigma) \subseteq \text{out}(s \texttt{ after } \sigma)
\]

\[
\text{out}(i \texttt{ after } ?\texttt{dub}) = \{ \delta, !\texttt{coffee} \}
\]
Implementation Relation $\mathrm{ioco}$

\[ \mathrm{i\ ioco\ s} \overset{\text{def}}{=} \forall \sigma \in \text{Straces}(s) : \text{out}(\mathrm{i\ after}\ \sigma) \subseteq \text{out}(\mathrm{s\ after}\ \sigma) \]

\[ \text{out}(\mathrm{i\ after}\ ?\text{dub}) = \{\delta, \text{!coffee}\} \quad \text{out}(\mathrm{s\ after}\ ?\text{dub}) = \{\text{!coffee}\} \]
Implementation Relation $\text{ioco}$

$$i \text{ ioco } s \ = \ _{\text{def}} \ \forall \sigma \in \text{Straces } (s) : \ \text{out } (i \ \text{after } \sigma) \subseteq \ \text{out } (s \ \text{after } \sigma)$$

$$\text{out } (i \ \text{after } ?\text{dub}) = \{ \delta, !\text{coffee} \} \quad \text{out } (s \ \text{after } ?\text{dub}) = \{ !\text{coffee} \}$$
Implementation Relation \texttt{ioco}

\[
i \texttt{ioco } s = \text{def } \forall \sigma \in \text{Straces}(s): \text{out}(i \text{ after } \sigma) \subseteq \text{out}(s \text{ after } \sigma)
\]
Implementation Relation \textbf{ioco}

\[ \text{i \text{ ioco s} ~ = \text{def} \quad \forall \sigma \in \text{Straces}(s) : \text{out}(\text{i after } \sigma) \subseteq \text{out}(s \text{ after } \sigma) } \]

\[
\begin{align*}
\text{out}(\text{i after } ?\text{dub}) & = \{ \delta, !\text{coffee} \}
\end{align*}
\]
Implementation Relation \[ \text{ioco} \]

\[
i \text{ioco} \ s \ = \ \text{def} \ \ \forall \sigma \in \text{Straces} (s) : \ \text{out} (i \ \text{after} \ \sigma) \subseteq \text{out} (s \ \text{after} \ \sigma)
\]

out \(i \ \text{after} \ ?\text{dub}) = \{ \delta, !\text{coffee} \} \quad \text{out} (s \ \text{after} \ ?\text{dub}) = \{ \delta, !\text{coffee} \}
Implementation Relation \textit{ioco}

\[ i \text{ ioco } s \overset{\text{def}}{=} \forall \sigma \in \text{Straces}(s) : \text{out}(i \text{ after } \sigma) \subseteq \text{out}(s \text{ after } \sigma) \]

\[ \text{out}(i \text{ after } ?dub) = \{ \delta, !\text{coffee} \} \]

\[ \text{out}(s \text{ after } ?dub) = \{ \delta, !\text{coffee} \} \]
Implementation Relation \( \text{ioCO} \)

\[
i \text{ioCO} s \quad =_{\text{def}} \quad \forall \sigma \in \text{Straces}(s) : \text{out}(i \text{ after } \sigma) \subseteq \text{out}(s \text{ after } \sigma)
\]

Diagram:

- For \( i \):
  - Input nodes: \(?\text{dub}\), \(?\text{dub}\), \(?\text{dub}\)
  - Output nodes: \(!\text{tea}\), \(!\text{coffee}\), \(?\text{dub}\)

- For \( s \):
  - Input nodes: \(?\text{dub}\), \(?\text{dub}\), \(?\text{dub}\)
  - Output nodes: \(!\text{tea}\), \(!\text{coffee}\), \(?\text{dub}\)

© Jan Tretmans and Julien Schmaltz
Implementation Relation $\text{ioco}$

\[
i \text{ioco } s \quad = \quad \text{def} \quad \forall \sigma \in \text{Straces } (s) : \quad \text{out } (i \text{ after } \sigma) \subseteq \text{out } (s \text{ after } \sigma)
\]

\[
\text{out } (i \text{ after } ?\text{dub}.?\text{dub}) = \text{out } (s \text{ after } ?\text{dub}.?\text{dub}) = \{ \text{!tea, !coffee} \}
\]
Implementation Relation \textit{ioco}

\[ i \text{ ioco } s \overset{\text{def}}{=} \forall \sigma \in \text{Straces}(s): \text{out}(i \text{ after } \sigma) \subseteq \text{out}(s \text{ after } \sigma) \]

\[ \text{out}(i \text{ after } ?dub. ?dub) = \text{out}(s \text{ after } ?dub. ?dub) = \{ !\text{tea}, !\text{coffee} \} \]

\[ \text{out}(i \text{ after } ?dub. \delta. ?dub) = \{ !\text{coffee} \} \neq \text{out}(s \text{ after } ?dub. \delta. ?dub) = \{ !\text{tea}, !\text{coffee} \} \]
Implementation Relation \textit{ioco}

\[ \text{i ioco s} \overset{\text{def}}{=} \forall \sigma \in \text{Straces (s)}: \text{out (i after } \sigma) \subseteq \text{out (s after } \sigma) \]

\[ \text{out (i after ?dub.?dub)} = \text{out (s after ?dub.?dub)} = \{ !\text{tea, !coffee} \} \]

\[ \text{out (i after ?dub.δ.?dub)} = \{ !\text{coffee} \} \neq \text{out (s after ?dub.δ.?dub)} = \{ !\text{tea, !coffee} \} \]
Implementation Relation \textit{ioco}

\[
i \text{ ioco } s \overset{\text{def}}{=} \forall \sigma \in \text{Straces } (s) : \text{out } (i \text{ after } \sigma) \subseteq \text{out } (s \text{ after } \sigma)
\]

out \text{ (i after ?dub.?dub)} = \text{out } (s \text{ after ?dub.?dub}) = \{ !\text{tea, !coffee} \}
out \text{ (i after ?dub.δ.?dub)} = \{ !\text{coffee} \} \neq \text{out } (s \text{ after ?dub.δ.?dub}) = \{ !\text{tea, !coffee} \}
Test cases
Test Case \( t \in \text{TTS} \)

TTS - Test Transition System:
Test Cases

Test case $t \in \text{TTS}$

TTS - Test Transition System:

[Diagram showing a Test Transition System with states and transitions labeled with symbols like `?coffee`, `!dub`, `?tea`, `θ`, `pass`, `fail`.]
Test Cases

Test case $t \in TTS$

TTS - Test Transition System:

- labels in $L \cup \{\theta\}$
Test Cases

Test case $t \in TTS$

TTS - Test Transition System:

- labels in $L \cup \{\theta\}$
- tree-structured
Test Cases

Test case $t \in TTS$

TTS - Test Transition System:

- labels in $L \cup \{\theta\}$
- tree-structured
- finite, deterministic

Diagram:

- Top level: ![Diagram](attachment:image.png)
- root node: ![Root Node](attachment:image.png)
- Nodes: ![Nodes](attachment:image.png)
- Edges: ![Edges](attachment:image.png)
- Labels: ![Labels](attachment:image.png)
- States: ![States](attachment:image.png)
- Transitions: ![Transitions](attachment:image.png)
- Pass/Fail: ![Pass/Fail](attachment:image.png)
Test Cases

Test case $t \in TTS$

TTS - Test Transition System:

- labels in $L \cup \{\theta\}$
- tree-structured
- finite, deterministic
- final states pass and fail

[Diagram of a test transition system with labels for coffee, tea, and conditions for pass and fail.]
Test Cases

Test case \( t \in \text{TTS} \)

\text{TTS - Test Transition System :}

- labels in \( L \cup \{\theta\} \)
- tree-structured
- finite, deterministic
- final states \text{pass} and \text{fail}
- from each state \( \neq \) \text{pass, fail}:
  - either one input \( !a \)
  - or all outputs \( ?x \) and \( \theta \)
### Test Cases

#### test case t

<table>
<thead>
<tr>
<th>Action</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>!dub !kwart ; Start timer1</td>
<td>fail</td>
</tr>
<tr>
<td>!dub ; Start timer2</td>
<td></td>
</tr>
<tr>
<td>!dub</td>
<td></td>
</tr>
<tr>
<td>!kwart ; Start timer1</td>
<td>fail</td>
</tr>
<tr>
<td>?tea</td>
<td>pass</td>
</tr>
<tr>
<td>?timer1</td>
<td>fail</td>
</tr>
<tr>
<td>?coffee</td>
<td></td>
</tr>
<tr>
<td>?dub</td>
<td></td>
</tr>
<tr>
<td>?kwart ?tea</td>
<td>pass</td>
</tr>
<tr>
<td>?timer2</td>
<td>pass</td>
</tr>
<tr>
<td>?coffee</td>
<td>fail</td>
</tr>
<tr>
<td>pass</td>
<td></td>
</tr>
<tr>
<td>pass</td>
<td></td>
</tr>
<tr>
<td>fail</td>
<td></td>
</tr>
</tbody>
</table>

© Jan Tretmans and Julien Schmaltz
Observations and Verdicts
Observations and Verdicts

\[ \text{OBS} \rightarrow (L \cup \{\emptyset\})^* \]
Observations and Verdicts

\[
\text{OBS} \rightarrow (L \cup \{\theta\})^*
\]

\[
\text{obs} \rightarrow \text{obs}: \text{TTS} \times \text{IOTS} \rightarrow \wp ( (L \cup \{\theta\})^* )
\]

\[
\text{obs} (t, i) = \{ \sigma \in (L \cup \{\theta\})^* \mid t \parallel i \xrightarrow{\sigma} \text{pass} \parallel i' \text{ or } t \parallel i \xrightarrow{\sigma} \text{fail} \parallel i' \}
\]
Observations and Verdicts

\[ \text{OBS} \rightarrow (L \cup \{\theta\})^* \]

\[ \text{obs} \rightarrow \text{obs} : \text{TTS} \times \text{IOTS} \rightarrow \wp((L \cup \{\theta\})^*) \]

\[ \text{obs}(t,i) = \{ \sigma \in (L \cup \{\theta\})^* \mid t || i \xrightarrow{\sigma} \text{pass} \} \cup \{ \sigma \in (L \cup \{\theta\})^* \mid t || i \xrightarrow{\sigma} \text{fail} \} \]

\[ \nu_t \rightarrow \nu_t : \wp((L \cup \{\theta\})^*) \rightarrow \{ \text{fail, pass} \} \]

\[ \nu_t(O) = \text{pass} \quad \text{if } \forall \sigma \in O. \quad t || i \xrightarrow{\sigma} \text{pass} \]

\[ = \text{fail} \quad \text{otherwise} \]
Test Generation
Test Generation

Look for test derivation algorithm \( \text{der} : \text{LTS} \rightarrow \wp(\text{TTS}) \)

such that \( \text{der} \) is sound and exhaustive:

\[
\forall i \in \text{IOTS}. \ (\forall t \in \text{der}(s). \ \nu_t(\text{obs}(t,i)) = \text{pass}) \iff i \in \text{ioco s}
\]
Look for test derivation algorithm \( \text{der} : \text{LTS} \rightarrow \mathcal{P}(\text{TTS}) \) such that \( \text{der} \) is sound and exhaustive:

\[
\forall i \in \text{IOTS}. \quad (\forall t \in \text{der}(s). \nu_t(\text{obs}(t, i)) = \text{pass}) \iff i \text{ ioco } s
\]

\[
i \text{ ioco } s =_{\text{def}} \forall \sigma \in \text{Straces}(s) : \text{out}(i \text{ after } \sigma) \subseteq \text{out}(s \text{ after } \sigma)
\]

\[
\text{obs}(t, i) = \{ \sigma \in (L \cup \{\theta\})^* \mid t \parallel i \xrightarrow{\sigma} \text{pass} \parallel i' \text{ or } t \parallel i \xrightarrow{\sigma} \text{fail} \parallel i' \}
\]

\[
\nu_t(O) = \text{pass} \text{ if } \forall \sigma \in O . \ t \xrightarrow{\sigma} \text{pass}
\]

\[
= \text{fail} \text{ otherwise}
\]
Test Generation
Test Generation

\[ \text{i ioco s} \overset{\text{def}}{=} \forall \sigma \in \text{Straces (s)}: \text{out (i after } \sigma) \subseteq \text{out (s after } \sigma) \]
Test Generation

\[ i \text{ ioco} s \overset{\text{def}}{=} \forall \sigma \in \text{Straces}(s) : \text{out}(i \text{ after } \sigma) \subseteq \text{out}(s \text{ after } \sigma) \]

\[ \text{out}(s \text{ after } \sigma) = \{ \!x, \!y \} \]
Test Generation

\[ i \text{ioco} s =_{\text{def}} \forall \sigma \in \text{Straces}(s) : \text{out}(i\text{ after } \sigma) \subseteq \text{out}(s\text{ after } \sigma) \]

\[
\begin{align*}
s & \quad \sigma \\
!x & \quad !y \quad \sigma \\
!x & \quad !z \quad \sigma
\end{align*}
\]

\[
\begin{align*}
\text{out}(s\text{ after } \sigma) &= \{!x, ly\} \\
\text{out}(i\text{ after } \sigma) &= \{!x, lz\}
\end{align*}
\]
Test Generation

\[ \text{i ioco } s \overset{\text{def}}{=} \forall \sigma \in \text{Straces } (s) \colon \text{out } (i \text{ after } \sigma) \subseteq \text{out } (s \text{ after } \sigma) \]

\[ \text{out } (s \text{ after } \sigma) = \{ !x, !y \} \]

\[ \text{out } (i \text{ after } \sigma) = \{ !x, !z \} \]

\[ \text{out } (\text{test after } \sigma) = L_U \]

© Jan Tretmans and Julien Schmaltz
Test Generation
Test Generation

\[ i \text{ ioco } s := \text{def } \forall \sigma \in \text{Straces}(s) : \text{out}(i \text{ after } \sigma) \subseteq \text{out}(s \text{ after } \sigma) \]
Test Generation

\[ \text{i ioco } s \overset{\text{def}}{=} \forall \sigma \in \text{Straces}(s) : \text{out}(\text{i after } \sigma) \subseteq \text{out}(s \text{ after } \sigma) \]
Test Generation

\[ \text{i ioco } s \ =_{\text{def}} \ \forall \sigma \in \text{Straces}(s) : \ \text{out (i after } \sigma) \subseteq \text{out (s after } \sigma) \]

\[
\begin{align*}
\text{out (s after } \sigma) &= \{ !x, !y, \delta \} \\
\text{out (i after } \sigma) &= \{ !x, !z, \delta \}
\end{align*}
\]
i ioco s = \text{def } \forall \sigma \in \text{Straces}(s) : \text{out (i after } \sigma \text{) } \subseteq \text{out (s after } \sigma \text{)}

\text{out (s after } \sigma \text{) } = \{ !x, !y, \delta \} \\
\text{out (i after } \sigma \text{) } = \{ !x, !z, \delta \} \\
\text{out (test after } \sigma \text{) } = L \cup \{ \theta \}
Test Generation Algorithm
Test Generation Algorithm

Algorithm

To generate a test case $t(S)$ from a transition system specification with $S$, with $S$ set of states (initially $S = s_0$ after $\varepsilon$)
Test Generation Algorithm

Algorithm

To generate a test case \( t(S) \) from a transition system specification with \( S \), with \( S \) set of states (initially \( S = s_0 \) after \( \varepsilon \))

Apply the following steps recursively, non-deterministically:
Test Generation Algorithm

Algorithm

To generate a test case $t(S)$ from a transition system specification with $S$, with $S$ set of states (initially $S = s_0$ after $\varepsilon$)

Apply the following steps recursively, non-deterministically:

1. end test case

   pass
Test Generation Algorithm

Algorithm

To generate a test case \( t(S) \) from a transition system specification with \( S \), with \( S \) set of states (initially \( S = s_0 \) after \( \varepsilon \))

Apply the following steps recursively, non-deterministically:

1. end test case
2. supply input

\[ \vdash (S \text{ after } ?a \neq \emptyset) \]
Algorithm

To generate a test case \( t(S) \) from a transition system specification with \( S \), with \( S \) set of states (initially \( S = s_0 \) after \( \varepsilon \))

Apply the following steps recursively, non-deterministically:

1. end test case
   - pass

2. supply input
   \[ \downarrow !a \]
   \[ t(S \text{ after } ?a \neq \emptyset) \]

3. observe output
   \[ \downarrow ?y \]
   \[ \text{fail} \]
   \[ \text{fail} \]
   \[ \text{allowed outputs or } \delta: \quad !x \in \text{out}(S) \]
   \[ \text{forbidden outputs or } \delta: \quad !y \notin \text{out}(S) \]
Test Generation Example
Test Generation Example
Test Generation Example

s

?dub

!tea

!coffee

test
Test Generation Example
Test Generation Example

s

?dub

!tea

!coffee

test

!dub
Test Generation Example
Test Generation Example

\[ s \]

\[ \text{test} \]

\[ \text{?dub} \]

\[ \text{!tea} \]

\[ \text{!coffee} \]

\[ \text{?coffee} \]

\[ \theta \]

\[ ?\text{tea} \]

\[ \text{pass} \]

\[ \text{fail} \]

\[ \text{fail} \]
Test Generation Example

s

?dub

!tea !coffee

!dub

?coffee

θ

?tea

pass fail fail

© Jan Tretmans and Julien Schmaltz
Test Generation Example

\[ s \]

\[ \begin{align*}
?dub \\
\_ \\
!tea \\
!coffee
\end{align*} \]

\[ \begin{align*}
\theta \\
?tea \\
?coffee
\end{align*} \]

\[ \begin{align*}
\theta \\
?tea \\
?choc
\end{align*} \]

\[ \begin{align*}
\theta \\
?coffee \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ \\
\_ 
\end{align*} \]
Test Generation Example

s

?dub

?dub

!coffee

test
Test Generation Example

double

double

!coffee

s

test
Test Generation Example

s
δ
?dub
?dub
!coffee
test
Test Generation Example

d sutest

?dub

?dub !coffee

!dub
Test Generation Example
Test Generation Example
Test Generation Example

s

θ

?dub

?dub

!coffee

δ

δ

θ

test

!dub

?coffee

?tea

fail

fail

pass
Test Execution Example

!dub

?dub

!choc

!tea

?coffee

θ

?tea

?choc

fail

pass

fail

fail
Test Execution Example

Two test runs:

\[
\begin{align*}
t \parallel i & \rightarrow \text{dub tea} \\
pass \parallel i' &
\end{align*}
\]
Test Execution Example

Two test runs:

1. \( t \parallel i \rightarrow \text{dub tea} \rightarrow \text{pass} \parallel i' \)
2. \( t \parallel i \rightarrow \text{dub choc} \rightarrow \text{fail} \parallel i'' \)
Two test runs:

\[
\begin{align*}
t \parallel \ i & \quad \text{dub tea} \quad \text{pass} \parallel \ i' \\
t \parallel \ i & \quad \text{dub choc} \quad \text{fail} \parallel \ i''
\end{align*}
\]
Test Execution Example
Test Execution Example

Two test runs:

<table>
<thead>
<tr>
<th>Test</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>\textit{t} \parallel i</td>
<td>\textit{dub} \parallel \textit{θ}</td>
</tr>
<tr>
<td>2</td>
<td>\textit{t} \parallel i</td>
<td>\textit{dub} \parallel \textit{coffee} \parallel \textit{θ}</td>
</tr>
</tbody>
</table>

© Jan Tretmans and Julien Schmaltz
Test Execution Example

Two test runs:

\[
\begin{align*}
t \parallel i \quad \text{dub} & \quad \text{θ} & \quad \text{pass} \parallel i' \\
t \parallel i \quad \text{dub coffee} & \quad \text{θ} & \quad \text{pass} \parallel i'
\end{align*}
\]
Test Generation Example
Test Generation Example

Equation solver for \( y^2 = x \)
Test Generation Example

Equation solver for $y^2 = x$

specification

$\begin{cases} ? \ x \ (x < 0) \\ ? \ x \ (x \geq 0) \end{cases}$

$! \sqrt{x} \quad ! \ -\sqrt{x}$
Test Generation Example

Equation solver for $y^2 = x$

specification

$\begin{array}{l}
? x (x < 0) \\
? x (x \geq 0) \\
! \sqrt{x} \\
! -\sqrt{x}
\end{array}$

test
Test Generation Example

Equation solver for $y^2 = x$

specification

$? \ x \ (x < 0)$

$? \ x \ (x \geq 0)$

$\sqrt{x}$

$! \sqrt{x}$

$! -\sqrt{x}$

test

$!9$
Test Generation Example

Equation solver for $y^2 = x$

specification

? $x$ ($x < 0$)

$\sqrt{x}$

? $x$ ($x \geq 0$)

$-\sqrt{x}$

pass

fail

otherwise

otherwise

$9$ otherwise

$-3$ otherwise

$3$ otherwise

$9$

$3$

pass

fail
Test Generation Example

Equation solver for $y^2 = x$

specification

\[ \begin{align*}
? x & \ (x < 0) \\
? x & \ (x \geq 0) \\
! \sqrt{x} & \\
! -\sqrt{x}
\end{align*} \]

test

\[ \begin{align*}
!9 & \\
\text{otherwise} & \\
?3 & \\
\text{fail} & \\
\text{pass} & \\
!4
\end{align*} \]
Test Generation Example

Equation solver for $y^2 = x$

specification

- $\sqrt{x}$ if $x \geq 0$
- $-\sqrt{x}$ if $x < 0$

Test

- $9$ fail otherwise
- $3$ pass otherwise
- $2$ fail otherwise
- $-2$ pass otherwise
- $-3$ fail otherwise
- $-4$ pass
Test Generation Example

Equation solver for $y^2 = x$

specification

```
? x (x < 0)

? x (x >= 0)

! √x

! -√x
```

test

```
!9
otherwise

?3
fail

pass

!4
otherwise

?2
fail

pass

pass
```

To cope with non-deterministic behaviour, tests are not linear traces, but trees.
Test Execution Example

implementation

\[ ? \times (x \geq 0) \]
\[ ! \sqrt{x} \]
\[ ? \times (x < 0) \]
\[ ! -\sqrt{x} \]
\[ ? y \]

\[ !9 \]
\[ !4 \]

\[ test \]

\[ otherwise \]
\[ ?-3 \]
\[ fail \]
\[ pass \]
\[ ?-2 \]
\[ fail \]
\[ pass \]
\[ pass \]
Test Execution Example

\[ x \begin{cases} \sqrt{x} & (x < 0) \\ -\sqrt{x} & (x \geq 0) \end{cases} \]

Implementation

\[ y \begin{cases} ? & \text{otherwise} \\ \text{fail} & (x < 0) \\ \text{pass} & (x \geq 0) \end{cases} \]

Test
Test Execution Example

\[ \text{implementation} \implies \text{test} \]

\[ ? \ x (x < 0) \]
\[ ? \ x (x \geq 0) \]
\[ ! \sqrt{x} \]
\[ ! \ -\sqrt{x} \]

\[ ? \ y \]

\[ !9 \]
\[ !4 \]

\[ \text{otherwise} \]
\[ ?-3 \]

\[ \text{fail} \]
\[ \text{pass} \]

\[ \text{otherwise} \]
\[ ?2 \]

\[ \text{fail} \]
\[ \text{pass} \]
\[ \text{pass} \]
Test Execution Example

implementation

\[ ? \times (x < 0) \]

\[ ? \times (x \geq 0) \]

\[ ! \sqrt{x} \]

\[ ! -\sqrt{x} \]

\[ ? y \]

test

\[ ?-3 \]

\[ ?3 \]

\[ ?4 \]

\[ !9 \]

otherwise

fail

pass

otherwise

\[ ?-2 \]

fail

pass

pass
Test Execution Example

implementation

\[ ? \times (x < 0) \]
\[ ? \times (x \geq 0) \]
\[ ! \sqrt{x} \]
\[ ! -\sqrt{x} \]
\[ ? y \]

---

test

\[ !9 \]

otherwise

\[ ?-3 \]

fail

pass

\[ !4 \]

otherwise

\[ ?2 \]

fail

pass

pass
Test Execution Example

\[ \text{implementation} \quad \parallel \quad \text{test} \]

- \( x (x < 0) \) with result \( \sqrt{x} \)
- \( x (x \geq 0) \) with result \( -\sqrt{x} \)
- \( y \) with result \( ? \)

- 1 fail
- 2 pass
- 3 pass
- 4 pass
- 9 pass
- 3 otherwise
- 2 otherwise
- 2 otherwise
- 3 otherwise
- 3 otherwise
- 2 otherwise
Test Execution Example

implementation

\[ \begin{align*}
? x & (x < 0) \\
? x & (x \geq 0) \\
? y
\end{align*} \]

\[ ? \sqrt{x} \]

\[ ? -\sqrt{x} \]

---

test

\[ \begin{align*}
? -3 & \text{ otherwise } \\
? 3 & \text{ otherwise } \\
? 2 & \text{ otherwise }
\end{align*} \]

\[ \begin{align*}
!9 & \text{ pass } \\
!4 & \text{ pass } \\
\text{fail} & \text{ fail } \\
\text{pass} & \text{ pass }
\end{align*} \]
Test Execution Example

Implementation

\[
\text{if } (x < 0) \text{ then } \sqrt{x} \text{ else } -\sqrt{x}\]

\[
\text{if } (x \geq 0) \text{ then } ? y \text{ else } ? x
\]

Test

\[
\text{if } y \text{ then } \text{pass} \text{ else } \text{fail}
\]

\[
\text{if } x \text{ then } \text{pass} \text{ else } \text{fail}
\]

\[
\text{if } x \text{ then } \text{pass} \text{ else } \text{pass}
\]
Test Execution Example

implementation

\[ ? \times (x < 0) \]
\[ \sqrt{x} \]
\[ ! \sqrt{x} \]
\[ ! -\sqrt{x} \]

\[ ? \times (x \geq 0) \]
\[ ? y \]

\[ ! \sqrt{x} \]

\[ ? \times (x \geq 0) \]
\[ ? y \]

\[ ! \sqrt{x} \]

\[ ! -\sqrt{x} \]

\[ ? \times (x < 0) \]
\[ \sqrt{x} \]
\[ ! \sqrt{x} \]
\[ ! -\sqrt{x} \]

\[ ? \times (x < 0) \]
\[ \sqrt{x} \]
\[ ! \sqrt{x} \]
\[ ! -\sqrt{x} \]

\[ ? \times (x \geq 0) \]
\[ ? y \]

\[ ! \sqrt{x} \]

\[ ! -\sqrt{x} \]

\[ ? \times (x < 0) \]
\[ \sqrt{x} \]
\[ ! \sqrt{x} \]
\[ ! -\sqrt{x} \]

\[ ? \times (x \geq 0) \]
\[ ? y \]

\[ ! \sqrt{x} \]

\[ ! -\sqrt{x} \]

Test Execution Example

\[ ? \times (x < 0) \]
\[ \sqrt{x} \]
\[ ! \sqrt{x} \]
\[ ! -\sqrt{x} \]

\[ ? \times (x \geq 0) \]
\[ ? y \]

\[ ! \sqrt{x} \]

\[ ! -\sqrt{x} \]

\[ ? \times (x < 0) \]
\[ \sqrt{x} \]
\[ ! \sqrt{x} \]
\[ ! -\sqrt{x} \]

\[ ? \times (x \geq 0) \]
\[ ? y \]

\[ ! \sqrt{x} \]

\[ ! -\sqrt{x} \]

\[ ? \times (x < 0) \]
\[ \sqrt{x} \]
\[ ! \sqrt{x} \]
\[ ! -\sqrt{x} \]

\[ ? \times (x \geq 0) \]
\[ ? y \]

\[ ! \sqrt{x} \]

\[ ! -\sqrt{x} \]
Test Execution Example

implementation

\[ ? \times (x >= 0) \]
\[ ? \times (x < 0) \]
\[ ! \sqrt{x} \]
\[ ! -\sqrt{x} \]
\[ ? y \]

\[ ! \sqrt{x} \]
\[ ! -\sqrt{x} \]
\[ ? y \]

\[ x (x >= 0) \]
\[ x (x < 0) \]
\[ !9 \]
\[ !4 \]
\[ !9 \]
\[ !3 \]
\[ !4 \]
\[ ? -3 \]
\[ ? 2 \]
\[ ? -2 \]
\[ otherwise \]

\[ fail \]
\[ pass \]
\[ fail \]
\[ pass \]
\[ pass \]

\[ pass \]
\[ fail \]
\[ pass \]
\[ pass \]
\[ pass \]
Test Execution Example

implementation

\[ ? \ x \ (x < 0) \]

\[ ? \ x \ (x \geq 0) \]

\[ ! \sqrt{x} \]

\[ ! \ -\sqrt{x} \]

\[ ? \ y \]

test

\[ \text{fail} \]

\[ \text{pass} \]

\[ \text{pass} \]

\[ \text{otherwise} \]

\[ ?-3 \]

\[ ?3 \]

\[ !9 \]

\[ !4 \]

\[ \text{otherwise} \]

\[ ?2 \]

\[ \text{fail} \]

\[ \text{pass} \]

\[ \text{pass} \]
Test Execution Example
Test Execution Example

implementation

\[ ? \times (x < 0) \]
\[ ! \sqrt{x} \]
\[ ? \times (x >= 0) \]
\[ ! -\sqrt{x} \]
\[ ? y \]

\[ ! x (x >= 0) \]

\[ ! x (x < 0) \]

\[ ! \sqrt{x} \]

\[ ? y \]

\[ ! -\sqrt{x} \]

\[ ? x (x >= 0) \]

\[ ! \sqrt{x} \]

\[ ? x (x < 0) \]

\[ ! -\sqrt{x} \]

\[ ? y \]

\[ ! \sqrt{x} \]

\[ ? y \]

\[ ! -\sqrt{x} \]
Test Execution Example

implementation

\[ \begin{align*}
? & \times (x < 0) \\
? & \times (x \geq 0) \\
! & \sqrt{x} \\
? & y
\end{align*} \]

\[ \begin{align*}
! & \sqrt{x} \\
! & -\sqrt{x}
\end{align*} \]

test

\[ \begin{align*}
! & 9 \\
\text{otherwise} & ! 3 \\
\text{otherwise} & ! 4
\end{align*} \]

fail

pass

fail

pass

pass
Validity of Test Generation
Validity of Test Generation

For every test $t$ generated with algorithm we have:
Validity of Test Generation

For every test $t$ generated with algorithm we have:
Validity of Test Generation

For every test \( t \) generated with algorithm we have:

- **Soundness**: 
  \( t \) will never fail with correct implementation

  \[ \text{i ioco s implies i passes t} \]
Validity of Test Generation

For every test \( t \) generated with algorithm we have:

\[ \text{Soundness:} \]

\[ t \text{ will never fail with correct implementation} \]

\[ \text{i ioco s implies i passes } t \]
Validity of Test Generation
For every test \( t \) generated with algorithm we have:

\[ \text{Soundness :} \quad \forall i \quad i \text{ oco s implies } i \text{ passes } t \]

\[ \text{Exhaustiveness :} \quad \forall i \quad i \text{ oco s implies } \exists t : i \text{ fails } t \]
Validity of Test Generation
Validity of Test Generation

Theorem:
Validity of Test Generation

Theorem:

Soundness:

\[ i \mathsf{ioco} s \quad \text{implies} \quad \forall t \in \mathsf{der}(s). \quad \nu_t(\mathsf{obs}(t, i)) = \mathsf{pass} \]
Validity of Test Generation

Theorem:

Soundness:

\[ i \ ioco\ s \quad \text{implies} \quad \forall t \in \text{der}(s). \ \nu_t(\text{obs}(t, i)) = \text{pass} \]
Validity of Test Generation

Theorem:

Soundness:

\[ i \ ioco \ s \implies \forall t \in \text{der}(s). \nu_t(\text{obs}(t,i)) = \text{pass} \]

Exhaustiveness:

\[ \forall t \in \text{der}(s). \nu_t(\text{obs}(t,i)) = \text{pass} \implies i \ ioco \ s \]
Soundness and Exhaustiveness

\[ s \xrightarrow{\delta} ?dub \xrightarrow{\delta} ?dub \xrightarrow{\delta} !coffee \xrightarrow{\delta} ?dub \]

\[ \text{test} \]

\[ !dub \xrightarrow{\theta} ?coffee \]

\[ \theta \xrightarrow{\theta} ?tea \]

\[ \theta \xrightarrow{\theta} \text{fail} \]

\[ \theta \xrightarrow{\theta} \text{fail} \]

\[ \theta \xrightarrow{\theta} \text{pass} \]
Soundness and Exhaustiveness

The diagram illustrates the interactions between different states and variables. The states and their transitions are labeled with symbols such as $\delta$, $\theta$, and $\delta$. The variables include $\delta$, $\text{dub}$, $\text{coffee}$, $\text{tea}$, $\text{pass}$, and $\text{fail}$. The diagram shows how these elements interact to demonstrate the concepts of soundness and exhaustiveness in a system.
Soundness and Exhaustiveness

Still sound!
Soundness and Exhaustiveness

\[
\delta(s) = \text{?dub} \\
\delta(\text{?dub}) = \text{!coffee} \\
\delta(\text{!coffee}) = \text{?dub} \\
\delta(\text{?dub}) = \text{!coffee} \\
\delta(\text{!coffee}) = \text{?tea} \\
\delta(\text{?tea}) = \text{fail} \\
\delta(\text{fail}) = \text{pass} \\
\delta(\text{pass}) = \Theta \\
\delta(\Theta) = \text{fail} \\
\delta(\text{fail}) = \text{pass} \\
\delta(\text{pass}) = \Theta
\]
Soundness and Exhaustiveness
Soundness and Exhaustiveness

Not sound anymore!