LTS in Haskell

Paper definition

Definition 1. *A* labelled transition system *is a 4-tuple* $\langle Q, L, T, q_0 \rangle$ *where*

- Q *is a countable, non-empty set of* states*;*
- L *is a countable set of* labels*;*
- $T \subseteq Q \times (L \cup \{\tau\}) \times Q$*, with* $\tau \notin L$*, is the* transition relation*;*
- $q_0 \in Q$ *is the* initial state.

Source: Tretmans, Jan. "Model based testing with labelled transition systems."
Formal methods and testing. Springer, Berlin, Heidelberg, 2008. 1-38.

Haskell definition

```haskell
type State = Integer
type Label = String
type LabeledTransition = (State, Label, State)
type Trace = [Label]
type LTS = ([State], [Label], [LabeledTransition], State)
```

createLTS

```haskell
coffeeImplSimple :: LTS
coffeeImplSimple = createLTS [(1, "coin", 2), (2, "coffee", 3)]
```

```
*LTS> :t coffeeImplSimple
coffeeImplSimple :: LTS
*LTS> coffeeImplSimple
([1,2,3],["coffee","coin"],[(1,"coin",2),(2,"coffee",3)],1)
```

Defining input-output relations

Definition 6. *A* labelled transition system with inputs and outputs *is a 5-tuple* $\langle Q, L_I, L_U, T, q_0 \rangle$ *where*

– $\langle Q, L_I \cup L_U, T, q_0 \rangle$ *is a labelled transition system in* $\mathcal{LTS}(L_I \cup L_U)$*;*
– L_I *and* L_U *are countable sets of* input labels *and* output labels, *respectively, which are disjoint:* $L_I \cap L_U = \emptyset$.

The class of labelled transition systems with inputs in L_I *and outputs in* L_U *is denoted by* $\mathcal{LTS}(L_I, L_U)$.

Source: Tretmans, Jan. "Model based testing with labelled transition systems." Formal methods and testing. Springer, Berlin, Heidelberg, 2008. 1-38.

Haskell definition

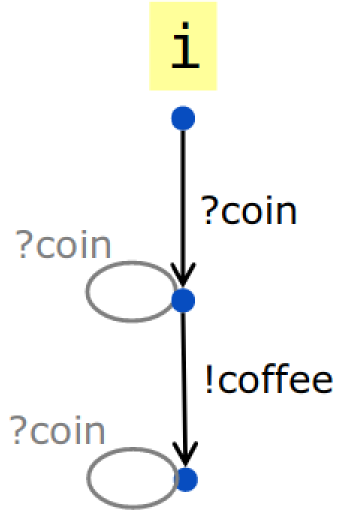
```haskell
type IOLTS = ([State], [Label], [Label], [LabeledTransition], State)
```

createIOLTS

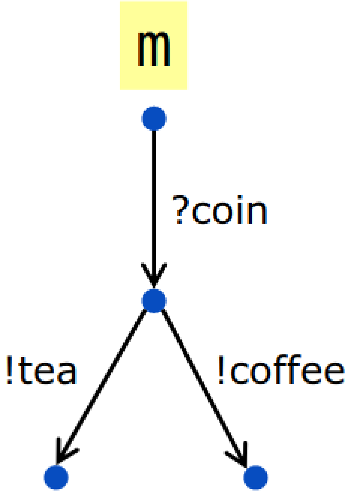
```haskell
coffeeImpl1 :: IOLTS
coffeeImpl1 = createIOLTS [(1, "?coin", 2), (2, "!coffee", 3)]
```

```
*LTS> :t coffeeImpl1
coffeeImpl1 :: IOLTS
*LTS> coffeeImpl1
([1,2,3],["coin"],["coffee"],[(1,"coin",2),(2,"coffee",3)],1)
```

i **ioco** m $=_{def}$ ∀σ ∈ **Straces**(m): out(i **after** σ) ⊆ out(m **after** σ)



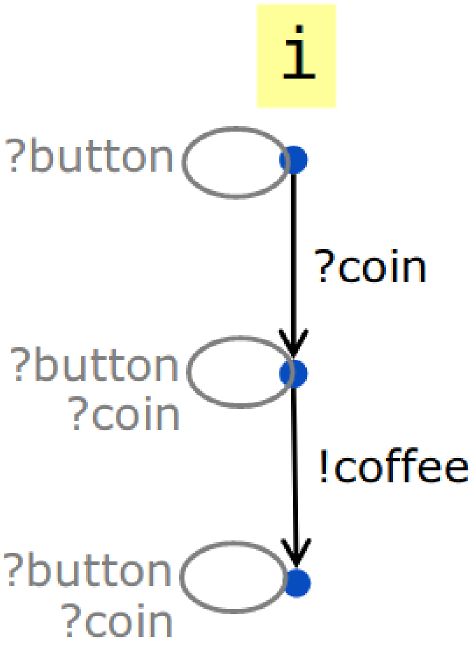
i ioco m



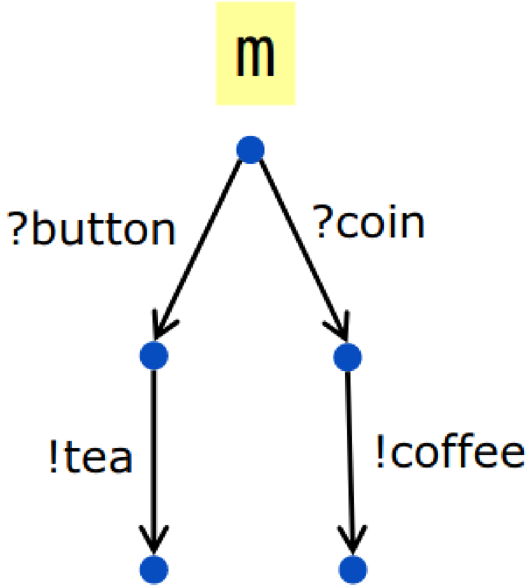
out(i **after** ?coin) = {!coffee} ⊆ out(m **after** ?coin) = {!coffee, !tea}

```
*Lab6> :t out
out :: LTS -> [State] -> [Label]
*Lab6> :t after
after :: LTS -> Trace -> [State]
*Lab6> out coffeeImpl1 (coffeeImpl1 `after` ["?coin"])
["!coffee"]
*Lab6> out coffeeModel1 (coffeeModel1 `after` ["?coin"])
["!coffee","!tea"]
```

```
*Lab6> coffeeImpl1 `ioco` coffeeModel1
True
```



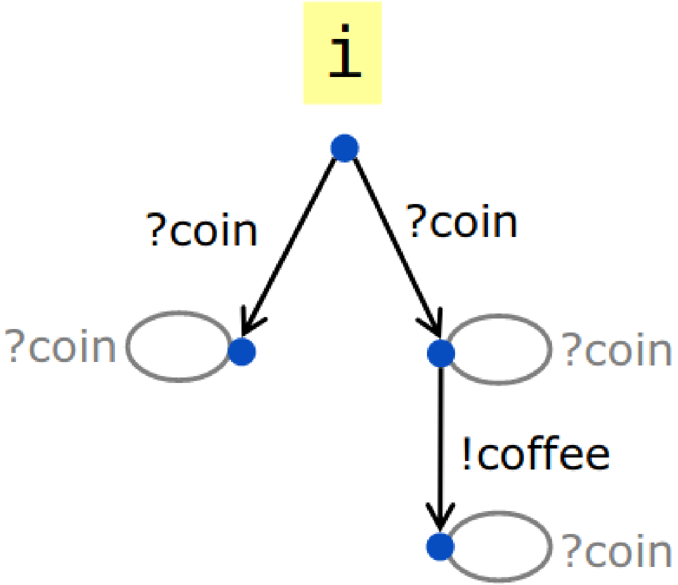
i ~~**ioco**~~ m



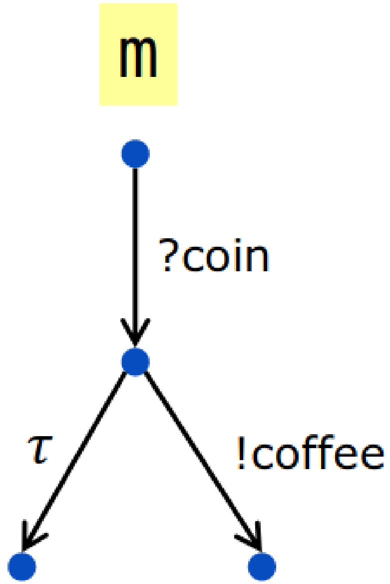
out(i **after** ?button) = {δ}

out(m **after** ?button) = {!tea}

```
*Lab6> out coffeeImpl4 (coffeeImpl4 `after` ["?button"])
["delta"]
*Lab6> out coffeeModel4 (coffeeModel4 `after` ["?button"])
["!tea"]
*Lab6> coffeeImpl4 `ioco` coffeeModel4
False
```



i ioco m



out(i **after** ?coin) = {δ, !coffee}

out(m **after** ?coin) = {δ, !coffee }

```
*Lab6> out coffeeImpl6 (coffeeImpl6 `after` ["?coin"])
["!coffee","delta"]
*Lab6> out coffeeModel6 (coffeeModel6 `after` ["?coin"])
["!coffee","delta","tau"]
*Lab6> coffeeImpl6 `ioco` coffeeModel6
True
```

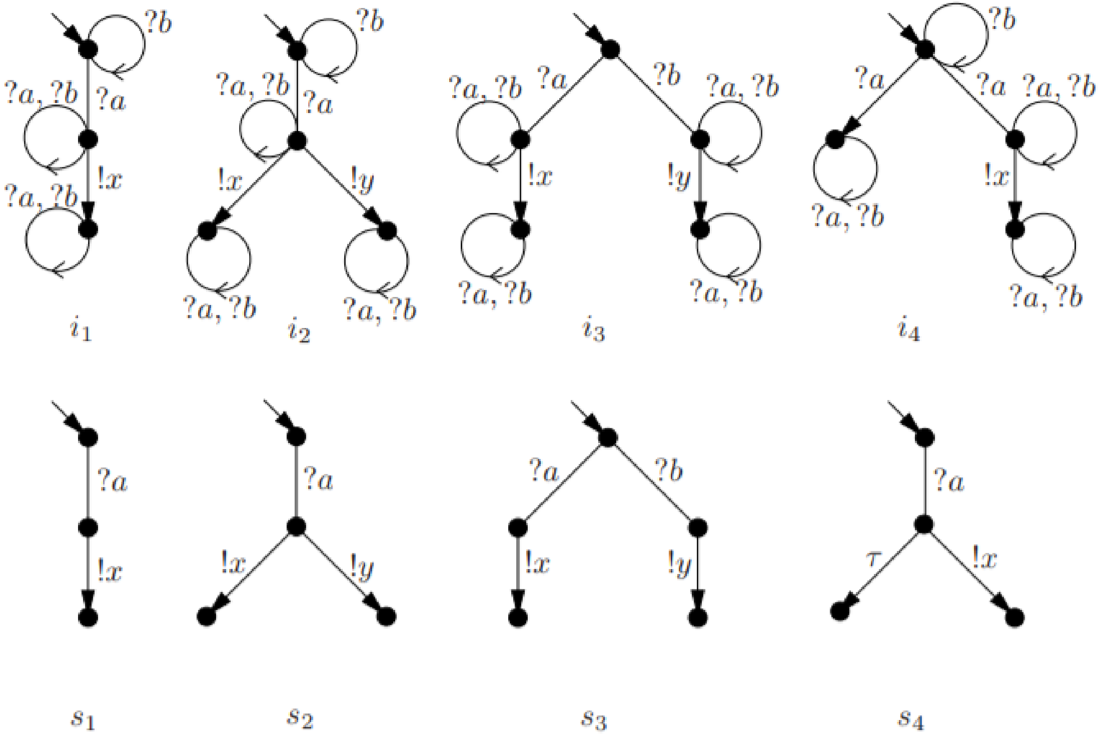


Fig. 8. Implementations and specifications with **ioco**.

“Model based testing with labelled transition systems” by Jan Tretmans
Page 22

Example 9. Figure 8 gives some implementations and specifications with $L_I = \{?a, ?b\}$ and $L_U = \{!x, !y\}$:

i_m **ioco** s_n	s_1	s_2	s_3	s_4
i_1	**ioco**	**ioco**	$\mathbf{io\not{c}o}$	**ioco**
i_2	$\mathbf{io\not{c}o}$	**ioco**	$\mathbf{io\not{c}o}$	$\mathbf{io\not{c}o}$
i_3	**ioco**	**ioco**	**ioco**	**ioco**
i_4	$\mathbf{io\not{c}o}$	$\mathbf{io\not{c}o}$	$\mathbf{io\not{c}o}$	**ioco**

That’s it (not really)