TCP_{\tau}(A, \gamma): basis

TCP_{\tau}(A, \gamma) includes:
- deadlock (0)
- successful termination (1)
- action prefix (a., a \in A_{\tau})
- choice (+)
- sequential composition (-)

\begin{align*}
x + y &= y + x & \text{A1} \\
(x + y) + z &= x + (y + z) & \text{A2} \\
x + x &= x & \text{A3} \\
(x + y) \cdot z &= x \cdot (y + z) & \text{A4} \\
(x \cdot y) \cdot z &= x \cdot (y \cdot z) & \text{A5} \\
x + 0 &= x & \text{A6} \\
0 \cdot x &= 0 & \text{A7} \\
x \cdot 1 &= x & \text{A8} \\
1 \cdot x &= x & \text{A9} \\
a \cdot x \cdot y &= a \cdot (x \cdot y) & \text{A10}
\end{align*}

TCP_{\tau}(A, \gamma): abstraction

To abstract from internal activity, TCP_{\tau}(A, \gamma) includes unary operations \( \tau_I (I \subseteq A) \).

\begin{align*}
\tau_I(1) &= 1 & \text{TI1} \\
\tau_I(0) &= 0 & \text{TI2} \\
\tau_I(a \cdot x) &= a \cdot \tau_I(x) & \text{if } a \not\in I & \text{TI3} \\
\tau_I(a \cdot x) &= \tau_I(x) & \text{if } a \in I & \text{TI4} \\
\tau_I(x + y) &= \tau_I(x) + \tau_I(y) & \text{TI5}
\end{align*}

The idea that \( \tau \) is considered unobservable is reflected by the axiom for rooted branching bisimilarity:

\[ a \cdot (\tau \cdot (x + y) + x) = a \cdot (x + y) \quad \text{B} \]

TCP_{\tau}(A, \gamma): parallelism

TCP_{\tau}(A, \gamma) includes:
- merge (\|)
- left merge (\|)
- comm. merge (\|)

\begin{align*}
x \| y &= x \| y + y \| x + x \| y & \text{M} \\
0 \| x &= 0 & \text{LM1} \\
1 \| x &= 0 & \text{LM2} \\
a \cdot x \| y &= a \cdot (x \| y) & \text{LM3} \\
(x + y) \| z &= x \| z + y \| z & \text{LM4}
\end{align*}

\begin{align*}
0 | x &= 0 & \text{CM1} \\
(x + y) | z &= x | z + y | z & \text{CM2} \\
1 | 1 &= 1 & \text{CM3} \\
a \cdot x | 1 &= 0 & \text{CM4} \\
a \cdot x | b \cdot y &= c \cdot (x \| y) & \text{if } \gamma(a, b) = c & \text{CM5} \\
a \cdot x | b \cdot y &= 0 & \text{if } \gamma(a, b) \text{ is not defined} & \text{CM6}
\end{align*}
**TCP\(_\tau(A, \gamma)\): standard concurrency**

\[
x \parallel y = y \parallel x \quad \text{SC1}
\]
\[
x \parallel 1 = x \quad \text{SC2}
\]
\[
1 \parallel x + 1 = 1 \quad \text{SC3}
\]
\[
(x \parallel y) \parallel z = x \parallel (y \parallel z) \quad \text{SC4}
\]
\[
(x \parallel y) \parallel z = x \parallel (y \parallel z) \quad \text{SC5}
\]
\[
(x \parallel y) \parallel z = x \parallel (y \parallel z) \quad \text{SC6}
\]
\[
x \parallel \tau.y = x \parallel y \quad \text{SC7}
\]
\[
x \parallel \tau.y = 0 \quad \text{SC10}
\]

Exercise 8.6.4

Derive \(x \parallel (\tau.(y + z) + y) = x \parallel (y + z)\).

Exercise 8.6.3

Derive SC9 and SC10 for closed TCP\(_\tau(A, \gamma)\)-terms.

**TCP\(_\tau(A, \gamma)\): encapsulation**

**Example**

Consider the term \(\partial_{\{\tau\}}(a.\tau.1)\).

Then
\[
\partial_{\{\tau\}}(a.\tau.1) = a.\partial_{\{\tau\}}(\tau.1) = a.0
\]

and
\[
\partial_{\{\tau\}}(a.\tau.1) = \partial_{\{\tau\}}(a.1) = a.1
\]

From the above, it follows that
\[
a.1 = \partial_{\{\tau\}}(a.\tau.1) = a.0
\]

Conclusion: we must not allow \(\tau\) to be an element of \(H\) in \(\partial_H\).

**TCP\(_\tau(A, \gamma)\): recursion**

**Exercise**

Give two distinct solutions in \(P(BSP_{\tau}(A))/\sim_{rb}\) for \(X\) in \(\{X = \tau.X\}\).

Conclusion: we should not allow \(\tau\) as a guard.

**Example**

Suppose that \(i \in I\), and consider the recursive specification
\[
X = \tau_I(i.X)
\]

Then, for distinct actions \(a\) and \(b\), \([\tau.a.1] \sim_{rb}\) and \([\tau.b.1] \sim_{rb}\) are both solutions for \(X\) in \(P(BSP_{\tau}(A))/\sim_{rb}\).

Conclusion: we should revise the definition of guardedness.
**TCP\(_\tau(A, \gamma)\): recursion**

**Definition**
An occurrence of a variable \( X \) is guarded if and only if it is not in the scope of an abstraction operator and occurs in a subterm of the form \( a.t \) for some \( a \in A \) (so \( a \neq \tau \)) and term \( t \).

**Theorem**
RDP and RSP are valid in \( \mathbb{P}(TCP_{\tau, rec}(A, \gamma)) \leftrightarrow rb; AIP \) is valid in the extension of \( \mathbb{P}(TCP_{\tau, rec}(A, \gamma)) \leftrightarrow rb \) with projection operators.

**TCP\(_\tau(A, \gamma)\): definability and abstraction**

We cannot use \( \tau \) in guarded recursive specifications. This does not, however, entirely preclude the use of abstraction.

**Definition**
A process is (finitely) definable with abstraction over \( TCP_{\tau}(A, \gamma) \) if it is obtained by applying an abstraction operator to a process that is (finitely) definable over \( TCP_{\tau}(A, \gamma) \).

If time permits, at the end of this lecture we will give evidence for the following theorem:

**Theorem**
Every executable process is definable with abstraction over \( TCP_{\tau}(A, \gamma) \).

[See the book for details]

---

**Queue**

Consider the behaviour of a queue over a (finite) set of data \( D \):

\[
\begin{align*}
\text{Queue} & = Q_e, \\
Q_e & = 1 + \sum_{d \in D} i?d.Q_d, \\
Q_{\sigma d} & = o!d.Q_{\sigma} + \sum_{e \in D} i?e.Q_{\sigma e d} \quad (d \in D, \ \sigma \in D^*) .
\end{align*}
\]

The behaviour of a queue is not finitely definable in BSP\((A)\), TSP\((A)\) and BCP\((A, \gamma)\).

**Theorem**
The behaviour of a queue is
1. finitely definable in \( TCP\(A, \gamma\)\) using an unbounded communication function;
2. finitely definable with abstraction in \( TCP_{\tau}(A, \gamma)\).

---

**Alternating-Bit Protocol**

High-level requirements
- Sender \( S \) receives data along its input port \( i \), and is supposed to transmit these data through an unreliable channel to receiver \( R \).
- The channel may corrupt data, but the assumption is that this can be recognised (e.g., by means of a checksum).
- If data is not correctly transmitted, it should be sent again.
Alternating-Bit Protocol

Implementation details

- To inform $S$ of correct transmission of a datum, $R$ sends acknowledgement.
- If $S$ receives a correct acknowledgement, it sends the next datum; if $S$ receives a corrupted acknowledgement, it sends the current datum.
- But how does $R$ know whether it receives a new datum or a datum that is re-sent?
- Solution: transmit data with appended alternating bit.

The unreliable channels

$$K = 1 + \sum_{x \in F} sk?x.(t.kr!x.K + t.kr!⊥.K)$$
$$L = 1 + \sum_{n \in \{0,1\}} rl?n.(t.ls!n.L + t.ls!⊥.L)$$

The sender

$$S = 1 + S_0 \cdot S_1 \cdot S$$
$$S_n = 1 + \sum_{d \in D} i?d.Sdn \quad (n \in \{0,1\})$$
$$S_{dn} = sk!dn.T_{dn} \quad (d \in D, \ n \in \{0,1\})$$
$$T_{dn} = ls?(1-n).S_{dn} + ls?⊥.S_{dn} + ls?n.1 \quad (d \in D, \ n \in \{0,1\})$$

The receiver

$$R = 1 + R_1 \cdot R_0 \cdot R$$
$$R_n = 1 + kr?⊥.r!n.R_n + \sum_{d \in D} kr?dn.r!n.R_n +$$
$$\sum_{d \in D} kr?d(1-n).o!d.r!(1-n).1 \quad (n \in \{0,1\})$$
Alternating-Bit Protocol

The behaviour of the protocol is now specified as

\[ \partial_H(S \parallel K \parallel L \parallel R) , \]

with

\[ H = \{ p?x, p!x \mid x \in F \cup \{0, 1, \perp\}, \ p \in \{ sk, kr, rl, ls \} \} \]

and

\[ \gamma(p?x, x!) = \gamma(p!x, p?x) = p\overline{x} \]

for all \( x \in F \cup \{0, 1, \perp\} \) and \( p \in \{ sk, kr, rl, ls \} \), and undefined otherwise.

Alternating-Bit Protocol (verification)

Theorem

The Alternating-Bit Protocol is a correct communication protocol, i.e.,

\[ (\text{TCP}_r + \text{HA})_{\text{rec}} + \text{CFAR}^b + \text{RSP}(A, \gamma) \vdash \tau_I(\partial_H(S \parallel K \parallel L \parallel R)) = \text{Buf1}_{io} . \]

where \( \gamma(p?x, x!) = \gamma(p!x, p?x) = p\overline{x} \) for all \( x \in F \cup \{0, 1, \perp\} \) and \( p \in \{ sk, kr, rl, ls \} \), and undefined otherwise, and

\[ H = \{ p?x, p!x \mid x \in F \cup \{0, 1, \perp\}, \ p \in \{ sk, kr, rl, ls \} \} , \quad \text{and} \]

\[ I = \{ p\overline{x} \mid x \in F \cup \{0, 1, \perp\}, \ p \in \{ sk, kr, rl, ls \} \} . \]

[Only CFAR^b has not been discussed in this course.]

Reactive Turing Machines (definition)

A: set of actions (intuitively, \( a \in A \) denotes an observable event);
\( \tau \): a special action (\( \not\in A \)) denoting unobservable event.

Definition

A reactive Turing machine (RTM) is a classical Turing machine with an additional action from \( A \cup \{\tau\} \) associated with every transition.

So, RTMs have two types of transitions:

\[ s \xrightarrow{a[d/e]M} t : \text{externally observable as the execution of action } a; \]

\[ s \xrightarrow{\tau[d/e]M} t : \text{not externally observable (internal computation step).} \]

RTMs (formal definition)

\( A_r \): a finite set of actions, including \( \tau \)
\( D_\square \): a finite set of tape symbols, including \( \square \)

Definition

An RTM is a quadruple \( M = (S, \rightarrow, \uparrow, \downarrow) \), with \( S \) a finite set of states,
\( \uparrow \in S \) a distinguished initial state, \( \downarrow \subseteq S \) a set of distinguished final states and

\[ \rightarrow \subseteq S \times D_\square \times A_r \times D_\square \times \{ L, R \} \times S . \]
Operational Semantics of RTMs

A configuration of an RTM $M$ is a pair of a state of $M$ and a description of tape contents.

Definition

With every RTM $M$ we associate a labeled transition system $T(M)$ that represents its behaviour:

- the states of $T(M)$ are the configurations of $M$;
- the initial state of $T(M)$ is the configuration of $M$ consisting of the initial state of $M$ and the empty tape;
- the transitions of $T(M)$ are determined by the transitions of $M$ and are labeled with actions in $A \cup \{\tau\}$;
- the terminating states of $T(M)$ are those configurations of $M$ involving a terminating state of $M$.

Example: transition system of RTM

Executability and Behavioural Equivalence

A transition system is called executable if it is behaviourally equivalent to the transition system of an RTM.

As behavioural equivalence we shall use

**rooted branching bisimilarity**

(the finest equivalence in van Glabbeek's linear time - branching time spectrum of behavioural equivalences.)

Expressiveness result

A transition system is computable if there exists computable function that associates with every state its set of outgoing transitions, and determines if the state is final or not.

**Theorem**

Every computable transition system is executable (i.e., rooted branching bisimilar to a transition system associated with an RTM).

For details see:

[BLT13] Jos Baeten, Bas Luttik, and Paul van Tilburg. Reactive Turing Machines. Information and Computation 231:244–166, 2013. (The paper is not part of the course material.)
TCP_\tau(A, \gamma) and executability

To prove that every executable transition system is finitely definable with abstraction in TCP_\tau(A, \gamma), it suffices to show that RTMs can be simulated in TCP_\tau(A, \gamma) up to rooted branching bisimilarity.

Let M = (S, \to, \uparrow, \downarrow).

We will specify the behaviour of the finite control and the behaviour of the tape memory separately, abstracting from the (enforced) communication between Control and Tape:

Goal: define Control and Tape such that

\[ T(M) \cong_{rb} \tau_1(\partial_H(Control \parallel Tape)) \]

Tape

A infinite specification of Tape:

\[ Tape = T_\parallel \]
\[ T_{\lambda\rho} = r!d.T_{\lambda\rho} + \sum_{e \in D_\square} w?e.T_{\lambda\rho} + m?L.T_{\lambda\rho} + m?R.T_{\lambda\rho} \]

To get a finite specification of Tape, we use the (finitely definable) process Queue as follows:

\[ Q_{\rho\bot\lambda} \]

(See [BLT13] for the specification.)

Control

Tape interface

\begin{align*}
\text{w!e} & : \text{write e at the position of the tape head} \\
\text{m!M} & : \text{instruct the tape head to move in direction M} \\
\text{r?f} & : \text{read the symbol at the position of the tape head}
\end{align*}

Finite control

\[ Control = C_{\uparrow, \square} \]
\[ C_{s,d} = \sum_{s \in [d,e]_M} \left( a.w!e.m!M. \sum_{f \in D_\square} r?f.C_{t,f} \right) [+1]_{s\downarrow} \]

Homework for Thursday

Read Section 7.7, 7.8, 8.6, 8.8, 8.9

Do Exercises 7.8.1, 8.6.1, 8.6.2, 8.6.3, 8.6.4, 8.8.1, 8.8.2