Concurrency Theory

Winter 2025/26

Lecture 13: Timed Modelling

Thomas Noll, Peter Thiemann
Programming Languages Group
University of Freiburg

https://proglang.github.io/teaching/25ws/ct.html

Thomas Noll, Peter Thiemann

Winter 2025/26

Outline of Lecture 13

1. Real-Time Reactive Systems
2. CCS with Time Delays
3. Syntax of Timed CCS
4. Semantics of Timed CCS
5. Handling Parallel Composition

So far: “Qualitative” Modelling

- Algebraic language (CCS) for syntactic description of concurrent systems
- Meaning given by LTSs that define dynamic behaviour of process terms
- Structural operational semantics for mapping CCS processes to LTSs
- Modal logics (HML) to specify desired system properties
- Notions of behavioural equivalence (trace equivalence, bisimilarity) for comparing process behaviours
- Later: Petri Nets as model of true concurrency with partial-order semantics

So far: “Qualitative” Modelling

- Algebraic language (CCS) for syntactic description of concurrent systems
- Meaning given by LTSs that define dynamic behaviour of process terms
- Structural operational semantics for mapping CCS processes to LTSs
- Modal logics (HML) to specify desired system properties
- Notions of behavioural equivalence (trace equivalence, bisimilarity) for comparing process behaviours
- Later: Petri Nets as model of true concurrency with partial-order semantics

$\Rightarrow$ Very abstract (if any) notion of time:
logical order of computation steps (causality)

Example 13.1 (Real-time reactive systems)

- Brake systems and airbags in cars
- Plant controls
- Mobile phones
- ...

Real-Time Reactive Systems

Example 13.1 (Real-time reactive systems)

- Brake systems and airbags in cars
- Plant controls
- Mobile phones
- ...

Real-time requirements

The correct behaviour of a real-time system does not only depend on the logical order in which events are performed but also on their timing.

Real-Time Reactive Systems

Example 13.1 (Real-time reactive systems)

- Brake systems and airbags in cars
- Plant controls
- Mobile phones
- ...

Real-time requirements

The correct behaviour of a real-time system does not only depend on the logical order in which events are performed but also on their timing.

Example 13.2 (Untimed vs. timed)

- Untimed: “If the car crashes, eventually the airbag will be inflated.”
- Timed: “If the car crashes, the airbag must be inflated within 50 milliseconds.”

Extensive research work on formal methods for real-time systems:

- **Modelling**
  - extensions of CCS: Timed CCS (Yi 1990), Temporal Process Algebra (Hennessy/Regan 1995), Temporal CCS (Moller/Tofts 1990)
  - extensions of other untimed process algebras (ACP, CSP)
  - timed automata (Alur/Dill 1990)

Theory of Real-Time Systems

Extensive research work on formal methods for real-time systems:

- **Modelling**
  - extensions of CCS: Timed CCS (Yi 1990), Temporal Process Algebra (Hennessy/Regan 1995), Temporal CCS (Moller/Tofts 1990)
  - extensions of other untimed process algebras (ACP, CSP)
  - timed automata (Alur/Dill 1990)
- **Requirement specification**
  - HML with time (Laroussinie et al. 1990)
  - extensions of LTL: Timed Propositional Temporal Logic (TPTL; Alur/Henzinger 1994), Metric Temporal Logic (MTL; Koymans 1990)
  - extension of CTL: Timed Computation Tree Logic (TCTL; Alur et al. 1993)

Theory of Real-Time Systems

Extensive research work on formal methods for real-time systems:

- **Modelling**
  - extensions of CCS: Timed CCS (Yi 1990), Temporal Process Algebra (Hennessy/Regan 1995), Temporal CCS (Moller/Tofts 1990)
  - extensions of other untimed process algebras (ACP, CSP)
  - timed automata (Alur/Dill 1990)
- **Requirement specification**
  - HML with time (Laroussinie et al. 1990)
  - extensions of LTL: Timed Propositional Temporal Logic (TPTL; Alur/Henzinger 1994), Metric Temporal Logic (MTL; Koymans 1990)
  - extension of CTL: Timed Computation Tree Logic (TCTL; Alur et al. 1993)
- **Analysis**
  - timed behavioural equivalences (timed trace equivalence, timed bisimilarity)
  - abstraction of timed automata via regions and zones

Theory of Real-Time Systems

Extensive research work on formal methods for real-time systems:

- **Modelling**
  - extensions of CCS: Timed CCS (Yi 1990), Temporal Process Algebra (Hennessy/Regan 1995), Temporal CCS (Moller/Tofts 1990)
  - extensions of other untimed process algebras (ACP, CSP)
  - timed automata (Alur/Dill 1990)
- **Requirement specification**
  - HML with time (Laroussinie et al. 1990)
  - extensions of LTL: Timed Propositional Temporal Logic (TPTL; Alur/Henzinger 1994), Metric Temporal Logic (MTL; Koymans 1990)
  - extension of CTL: Timed Computation Tree Logic (TCTL; Alur et al. 1993)
- **Analysis**
  - timed behavioural equivalences (timed trace equivalence, timed bisimilarity)
  - abstraction of timed automata via regions and zones
- **Here:** Syntax and semantics of Timed CCS (TCCS)
  - Wang Yi: *Real-time behaviour of asynchronous agents*, CONCUR 1990

Outline of Lecture 13

1. Real-Time Reactive Systems
2. CCS with Time Delays
3. Syntax of Timed CCS
4. Semantics of Timed CCS
5. Handling Parallel Composition

Example 13.3 (Light switch)

(1) If the switch is off, and is pressed once, then the light will turn on.

(2) If the switch is pressed again “soon” after the light was turned on, the light becomes brighter. Otherwise, the light is turned off by the next button press.

(3) The light is also turned off by a button press when it is bright.

Untimed Modelling

Example 13.3 (Light switch)

(1) If the switch is off, and is pressed once, then the light will turn on.
- in CCS: $\mathit{Off} = \mathit{press}.\mathit{Light}$

(2) If the switch is pressed again "soon" after the light was turned on, the light becomes brighter. Otherwise, the light is turned off by the next button press.

(3) The light is also turned off by a button press when it is bright.
- in CCS: $\mathit{Bright} = \mathit{press}.\mathit{Off}$

Example 13.3 (Light switch)

(1) If the switch is off, and is pressed once, then the light will turn on.
- in CCS: $\mathit{Off} = \mathit{press}.\mathit{Light}$

(2) If the switch is pressed again “soon” after the light was turned on, the light becomes brighter. Otherwise, the light is turned off by the next button press.
- in CCS: $\mathit{Light} = \mathit{press}.\mathit{Bright} + \tau.\mathit{press}.\mathit{Off}$
- but: does not properly capture the “soon” requirement
- rather: system may internally choose to switch off light after next button press (after “timeout” action τ)

(3) The light is also turned off by a button press when it is bright.
- in CCS: $\mathit{Bright} = \mathit{press}.\mathit{Off}$

Modelling with time delays

$$Light = press.Bright + \varepsilon(1.5).\tau.press.Off$$

- Passage of time viewed as "action" performed by system.
- Specified by new prefixing operator $\varepsilon(d).P$ where $d \in \mathbb{R}_{\geq 0}$ gives amount of time that needs to elapse before process P is enabled.
- Thus: "soon" interpreted as "within 1.5 time units."
- Use of τ is crucial here: must be performed when enabled (details later).

Timed Labelled Transition Systems I

The semantic model for our timed extension of CCS is provided by the following concept:

Definition 13.4 (Timed labelled transition system)

A timed labelled transition system (TLTS) is a triple $(S, Lab, \longrightarrow)$ consisting of

- a set S of states,
- a set $Lab = Act \cup \mathbb{R}_{\geq 0}$ of labels
  - actions $\alpha \in Act$
  - time delays $d \in \mathbb{R}_{\geq 0}$, and
- a transition relation $\longrightarrow \subseteq S \times Lab \times S$ (written $s \xrightarrow{\lambda} s'$).

Timed Labelled Transition Systems I

The semantic model for our timed extension of CCS is provided by the following concept:

Definition 13.4 (Timed labelled transition system)

A timed labelled transition system (TLTS) is a triple $(S, Lab, \longrightarrow)$ consisting of

- a set S of states,
- a set $Lab = Act \cup \mathbb{R}_{\geq 0}$ of labels
  - actions $\alpha \in Act$
  - time delays $d \in \mathbb{R}_{\geq 0}$, and
- a transition relation $\longrightarrow \subseteq S \times Lab \times S$ (written $s \xrightarrow{\lambda} s'$).

Additional requirements:

- Time additivity: if $s \xrightarrow{d} s'$ and $0 \leq d' \leq d$, then $s \xrightarrow{d'} s'' \xrightarrow{d-d'} s'$ for some $s'' \in S$.
- Self-reachability without delay: $s \xrightarrow{0} s$ for each $s \in S$.
- Time determinism: if $s \xrightarrow{d} s'$ and $s \xrightarrow{d} s''$, then $s' = s''$.

Timed Labelled Transition Systems II

Example 13.5 (Timed labelled transition system)

$(S, Lab, \longrightarrow)$ where

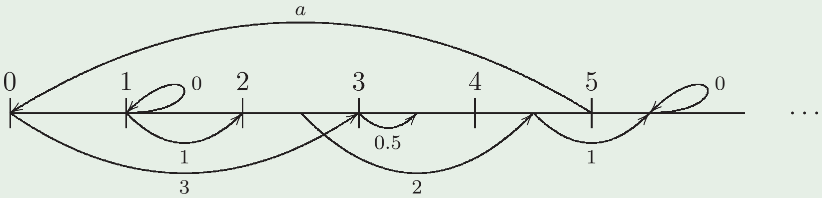
- $S = \mathbb{R}_{\geq 0}$
- $Lab = \{a\} \cup \mathbb{R}_{\geq 0}$
- $\xrightarrow{a} = \{(5, 0)\}$
- for all $d \in \mathbb{R}_{\geq 0}$: $\xrightarrow{d} = \{(s, s + d) \mid s \in \mathbb{R}_{\geq 0}\}$

Timed Labelled Transition Systems II

Example 13.5 (Timed labelled transition system)

$(S, Lab, \longrightarrow)$ where

- $S = \mathbb{R}_{\geq 0}$
- $Lab = \{a\} \cup \mathbb{R}_{\geq 0}$
- $\xrightarrow{a} = \{(5, 0)\}$
- for all $d \in \mathbb{R}_{\geq 0}$: $\xrightarrow{d} = \{(s, s + d) \mid s \in \mathbb{R}_{\geq 0}\}$



Outline of Lecture 13

1 Real-Time Reactive Systems

2 CCS with Time Delays

3 Syntax of Timed CCS

4 Semantics of Timed CCS

5 Handling Parallel Composition

Definition 13.6 (Syntax of TCCS; cf. Definition 2.1)

- Let A be a set of (action) names.

Definition 13.6 (Syntax of TCCS; cf. Definition 2.1)

- Let A be a set of (action) names.
- $\overline{A} := \{\overline{a} \mid a \in A\}$ denotes the set of co-names.

Syntax of Timed CCS I

Definition 13.6 (Syntax of TCCS; cf. Definition 2.1)

- Let A be a set of (action) names.
- $\overline{A} := \{\overline{a} \mid a \in A\}$ denotes the set of co-names.
- $Act := A \cup \overline{A} \cup \{\tau\}$ is the set of actions with the silent (or: unobservable) action τ.

Syntax of Timed CCS I

Definition 13.6 (Syntax of TCCS; cf. Definition 2.1)

- Let A be a set of (action) names.
- $\overline{A} := \{\overline{a} \mid a \in A\}$ denotes the set of co-names.
- $Act := A \cup \overline{A} \cup \{\tau\}$ is the set of actions with the silent (or: unobservable) action τ.
- Let Pid be a set of process identifiers.

Syntax of Timed CCS I

Definition 13.6 (Syntax of TCCS; cf. Definition 2.1)

- Let A be a set of (action) names.
- $\overline{A} := \{\overline{a} \mid a \in A\}$ denotes the set of co-names.
- $Act := A \cup \overline{A} \cup \{\tau\}$ is the set of actions with the silent (or: unobservable) action τ.
- Let Pid be a set of process identifiers.
- The set Prc^* of timed process expressions is defined by the following syntax:

$$
\begin{array}{rlll}
P ::= & \mathsf{nil} & & \text{(inaction)} \\
| & \alpha.P & & \text{(prefixing)} \\
| & \varepsilon(d).P & & \text{(time delay)} \\
| & P_1 + P_2 & & \text{(choice)} \\
| & P_1 \parallel P_2 & & \text{(parallel composition)} \\
| & P \setminus L & & \text{(restriction)} \\
| & P[f] & & \text{(relabelling)} \\
| & C & & \text{(process call)}
\end{array}
$$

where $\alpha \in Act$, $d \in \mathbb{R}_{\geq 0}$, $\emptyset \neq L \subseteq A$, $C \in Pid$, and $f : Act \to Act$ such that $f(\tau) = \tau$ and $f(\overline{a}) = \overline{f(a)}$ for each $a \in A$.

Syntax of Timed CCS II

Definition 13.6 (continued)

- A (recursive) timed process definition is an equation system of the form

$$(C_i = P_i \mid 1 \leq i \leq k)$$

  where $k \geq 1$, $C_i \in Pid$ (pairwise distinct), and $P_i \in Prc^*$ (with identifiers from $\{C_1, \ldots, C_k\}$).
- An occurrence of a process identifier $C \in Pid$ in an expression $P \in Prc^*$ is guarded if it occurs within a subexpression of P of the form $\lambda.Q$ where $\lambda \in Act$ or $\lambda = \varepsilon(d)$ for some $d > 0$.
- A process expression/definition is guarded if all occurrences of process identifiers are guarded.

Definition 13.6 (continued)

- A (recursive) timed process definition is an equation system of the form

$$(C_i = P_i \mid 1 \leq i \leq k)$$

  where $k \geq 1$, $C_i \in Pid$ (pairwise distinct), and $P_i \in Prc^*$ (with identifiers from $\{C_1, \ldots, C_k\}$).
- An occurrence of a process identifier $C \in Pid$ in an expression $P \in Prc^*$ is guarded if it occurs within a subexpression of P of the form $\lambda.Q$ where $\lambda \in Act$ or $\lambda = \varepsilon(d)$ for some $d > 0$.
- A process expression/definition is guarded if all occurrences of process identifiers are guarded.

Conventions:

- Processes P and $\varepsilon(0).P$ will not be distinguished.
- All process definitions have to be guarded, which avoids some semantic intricacies (for instance, the "self-reachability without delay" property of Definition 13.4 would otherwise not hold for TLTSs derived from TCCS processes).

Syntax of Timed CCS III

Example 13.7

(1) For $P = (a.C_1 + (C_2 \parallel b.C_3) + C_1) \parallel (\varepsilon(4.2).(C_4 \parallel \mathrm{nil}) + \varepsilon(1.2).C_3) \in Prc^*$:

- First occurrence of C_1 is guarded, second unguarded.
- Occurrence of C_2 is unguarded.
- Both occurrences of C_3 are guarded.
- Occurrence of C_4 is guarded.
- Overall expression is unguarded.

Example 13.7

(1) For $P = (a.C_1 + (C_2 \parallel b.C_3) + C_1) \parallel (\varepsilon(4.2).(C_4 \parallel \mathrm{nil}) + \varepsilon(1.2).C_3) \in Prc^*$:

- First occurrence of C_1 is guarded, second unguarded.
- Occurrence of C_2 is unguarded.
- Both occurrences of C_3 are guarded.
- Occurrence of C_4 is guarded.
- Overall expression is unguarded.

(2) The process definition

$$\begin{aligned} Off &= press.Light \\ Light &= press.Bright + \varepsilon(1.5).\tau.press.Off \\ Bright &= press.Off \end{aligned}$$

is guarded.

Outline of Lecture 13

1. Real-Time Reactive Systems
2. CCS with Time Delays
3. Syntax of Timed CCS
4. Semantics of Timed CCS
5. Handling Parallel Composition

Definition 13.8 (Semantics of TCCS – action transitions; cf. Definition 2.4)

A guarded process definition $(C_i = P_i \mid 1 \leq i \leq k)$ determines the TLTS $(Prc^*, Lab, \longrightarrow)$ whose transitions can be inferred from the following rules ($P, P', Q, Q' \in Prc^*$, $\alpha \in Act$, $\lambda \in A \cup \overline{A}$):

$$(\text{Act})\,\frac{}{\alpha.P \xrightarrow{\alpha} P}$$

$$(\text{Del})\,\frac{P \xrightarrow{\alpha} P'}{\varepsilon(0).P \xrightarrow{\alpha} P'} \qquad (\text{Sum}_1)\,\frac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'} \qquad (\text{Sum}_2)\,\frac{Q \xrightarrow{\alpha} Q'}{P + Q \xrightarrow{\alpha} Q'}$$

$$(\text{Par}_1)\,\frac{P \xrightarrow{\alpha} P'}{P \parallel Q \xrightarrow{\alpha} P' \parallel Q} \qquad (\text{Par}_2)\,\frac{Q \xrightarrow{\alpha} Q'}{P \parallel Q \xrightarrow{\alpha} P \parallel Q'} \qquad (\text{Com})\,\frac{P \xrightarrow{\lambda} P' \quad Q \xrightarrow{\overline{\lambda}} Q'}{P \parallel Q \xrightarrow{\tau} P' \parallel Q'}$$

$$(\text{Res})\,\frac{P \xrightarrow{\alpha} P' \quad (\alpha, \overline{\alpha} \notin L)}{P \setminus L \xrightarrow{\alpha} P' \setminus L} \qquad (\text{Rel})\,\frac{P \xrightarrow{\alpha} P'}{P[f] \xrightarrow{f(\alpha)} P'[f]} \qquad (\text{Call})\,\frac{P \xrightarrow{\alpha} P' \quad (C = P)}{C \xrightarrow{\alpha} P'}$$

Definition 13.8 (Semantics of TCCS – timed transitions)

Additionally for $d, d' \in \mathbb{R}_{\geq 0}$:

$$\text{(tAdd)}\frac{P \xrightarrow{d'} P'}{\varepsilon(d).P \xrightarrow{d+d'} P'} \qquad \text{(tSub)}\frac{(d' \leq d)}{\varepsilon(d).P \xrightarrow{d'} \varepsilon(d-d').P} \qquad \text{(tAct)}\frac{(\alpha \neq \tau)}{\alpha.P \xrightarrow{d} \alpha.P}$$

$$\text{(tTau)}\frac{}{\tau.P \xrightarrow{0} \tau.P} \qquad \text{(tSum)}\frac{P \xrightarrow{d} P' \quad Q \xrightarrow{d} Q'}{P + Q \xrightarrow{d} P' + Q'} \qquad \text{(tRes)}\frac{P \xrightarrow{d} P'}{P \setminus L \xrightarrow{d} P' \setminus L}$$

$$\text{(tRel)}\frac{P \xrightarrow{d} P'}{P[f] \xrightarrow{d} P'[f]} \qquad \text{(tCall)}\frac{P \xrightarrow{d} P' \quad (C = P)}{C \xrightarrow{d} P'}$$

Semantics of Timed CCS II

Definition 13.8 (Semantics of TCCS – timed transitions)

Additionally for $d, d' \in \mathbb{R}_{\geq 0}$:

$$\text{(tAdd)}\ \frac{P \xrightarrow{d'} P'}{\varepsilon(d).P \xrightarrow{d+d'} P'} \qquad \text{(tSub)}\ \frac{(d' \leq d)}{\varepsilon(d).P \xrightarrow{d'} \varepsilon(d-d').P} \qquad \text{(tAct)}\ \frac{(\alpha \neq \tau)}{\alpha.P \xrightarrow{d} \alpha.P}$$

$$\text{(tTau)}\ \frac{}{\tau.P \xrightarrow{0} \tau.P} \qquad \text{(tSum)}\ \frac{P \xrightarrow{d} P' \quad Q \xrightarrow{d} Q'}{P+Q \xrightarrow{d} P'+Q'} \qquad \text{(tRes)}\ \frac{P \xrightarrow{d} P'}{P \setminus L \xrightarrow{d} P' \setminus L}$$

$$\text{(tRel)}\ \frac{P \xrightarrow{d} P'}{P[f] \xrightarrow{d} P'[f]} \qquad \text{(tCall)}\ \frac{P \xrightarrow{d} P' \quad (C = P)}{C \xrightarrow{d} P'}$$

Remarks:

- Delay transitions do *not* resolve non-deterministic choices (Rule (tSum)) (according to time-determinism property of Definition 13.4).
- Rules (tAct) and (tTau) ensure that τ cannot be delayed if enabled.
- Parallel composition will be considered later.

Semantics of Timed CCS III

Example 13.9 (Light switch)

$$
\begin{aligned}
\mathit{Off} &= \mathit{press}.\mathit{Light} \\
\mathit{Light} &= \mathit{press}.\mathit{Bright} + \varepsilon(1.5).\tau.\mathit{press}.\mathit{Off} \\
\mathit{Bright} &= \mathit{press}.\mathit{Off}
\end{aligned}
$$

Example 13.9 (Light switch)

$$\begin{aligned} \mathit{Off} &= \mathit{press}.\mathit{Light} \\ \mathit{Light} &= \mathit{press}.\mathit{Bright} + \varepsilon(1.5).\tau.\mathit{press}.\mathit{Off} \\ \mathit{Bright} &= \mathit{press}.\mathit{Off} \end{aligned}$$

(1) $\mathit{Light} \xrightarrow{\mathit{press}} \mathit{Bright} \qquad ((\mathrm{Call}), (\mathrm{Sum}_1), (\mathrm{Act}))$

Semantics of Timed CCS III

Example 13.9 (Light switch)

$$\begin{aligned} Off &= press.Light \\ Light &= press.Bright + \varepsilon(1.5).\tau.press.Off \\ Bright &= press.Off \end{aligned}$$

(1) $Light \xrightarrow{press} Bright$ $\qquad$ ((Call), (Sum$_1$), (Act))

(2) For $0 \leq d \leq 1.5$:

$$Light \xrightarrow{d} press.Bright + \varepsilon(1.5 - d).\tau.press.Off \qquad ((\text{Call}), (\text{tSum}), (\text{tAct}), (\text{tSub}))$$

Semantics of Timed CCS III

Example 13.9 (Light switch)

$$
\begin{aligned}
\mathit{Off} &= \mathit{press}.\mathit{Light} \\
\mathit{Light} &= \mathit{press}.\mathit{Bright} + \varepsilon(1.5).\tau.\mathit{press}.\mathit{Off} \\
\mathit{Bright} &= \mathit{press}.\mathit{Off}
\end{aligned}
$$

(1) $\mathit{Light} \xrightarrow{\mathit{press}} \mathit{Bright}$ ((Call), (Sum$_1$), (Act))

(2) For $0 \leq d \leq 1.5$:

$$\mathit{Light} \xrightarrow{d} \mathit{press}.\mathit{Bright} + \varepsilon(1.5 - d).\tau.\mathit{press}.\mathit{Off} \qquad \text{((Call), (tSum), (tAct), (tSub))}$$

(3) Especially for $d = 1.5$ (and all $d' \in \mathbb{R}_{\geq 0}$):

$$
\begin{aligned}
\mathit{Light} &\xrightarrow{1.5} \mathit{press}.\mathit{Bright} + \varepsilon(0).\tau.\mathit{press}.\mathit{Off} \quad (*) && \text{((Call), (tSum), (tAct), (tSub))} \\
&\xrightarrow{\tau} \mathit{press}.\mathit{Off} && \text{((Sum}_2\text{), (Del), (Act))} \\
&\xrightarrow{d'} \mathit{press}.\mathit{Off} && \text{((tAct))} \\
&\xrightarrow{\mathit{press}} \mathit{Off} && \text{((Act))}
\end{aligned}
$$

Semantics of Timed CCS III

Example 13.9 (Light switch)

$$\begin{aligned} \mathit{Off} &= \mathit{press}.\mathit{Light} \\ \mathit{Light} &= \mathit{press}.\mathit{Bright} + \varepsilon(1.5).\tau.\mathit{press}.\mathit{Off} \\ \mathit{Bright} &= \mathit{press}.\mathit{Off} \end{aligned}$$

(1) $\mathit{Light} \xrightarrow{\mathit{press}} \mathit{Bright}$ ((Call), (Sum$_1$), (Act))

(2) For $0 \leq d \leq 1.5$:

$\mathit{Light} \xrightarrow{d} \mathit{press}.\mathit{Bright} + \varepsilon(1.5 - d).\tau.\mathit{press}.\mathit{Off}$ ((Call), (tSum), (tAct), (tSub))

(3) Especially for $d = 1.5$ (and all $d' \in \mathbb{R}_{\geq 0}$):

$\mathit{Light} \xrightarrow{1.5} \mathit{press}.\mathit{Bright} + \varepsilon(0).\tau.\mathit{press}.\mathit{Off}$	$(*)$	((Call), (tSum), (tAct), (tSub))
$\xrightarrow{\tau} \mathit{press}.\mathit{Off}$		((Sum$_2$), (Del), (Act))
$\xrightarrow{d'} \mathit{press}.\mathit{Off}$		((tAct))
$\xrightarrow{\mathit{press}} \mathit{Off}$		((Act))

(4) Moreover in $(*)$: $\mathit{press}.\mathit{Bright} + \varepsilon(0).\tau.\mathit{press}.\mathit{Off} \stackrel{d}{\not\longrightarrow}$ (for any $d > 0$)

$\Rightarrow$ First alternative only enabled up to time point 1.5.

Properties of the Semantics

Lemma 13.10 (cf. Definition 13.4)

(1) *Time additivity: if* $P \xrightarrow{d} P'$ *and* $0 \leq d' \leq d$, *then* $P \xrightarrow{d'} P'' \xrightarrow{d-d'} P'$ *for some* $P'' \in Prc^*$.

(2) *Self-reachability without delay:* $P \xrightarrow{0} P$ *for each* $P \in Prc^*$.

(3) *Time determinism: if* $P \xrightarrow{d} P'$ *and* $P \xrightarrow{d} P''$, *then* $P' = P''$.

(4) *Persistency of action transitions: for all* $P, Q \in Prc^*$, $\alpha \in Act$ *and* $d \in \mathbb{R}_{\geq 0}$, *if* $P \xrightarrow{\alpha}$ *and* $P \xrightarrow{d} Q$, *then* $Q \xrightarrow{\alpha}$.

(1)–(3) implies that the semantics of a TCCS process is indeed a TLTS (Def. 13.4).

Properties of the Semantics

Lemma 13.10 (cf. Definition 13.4)

(1) *Time additivity: if* $P \xrightarrow{d} P'$ *and* $0 \leq d' \leq d$, *then* $P \xrightarrow{d'} P'' \xrightarrow{d-d'} P'$ *for some* $P'' \in Prc^*$.

(2) *Self-reachability without delay:* $P \xrightarrow{0} P$ *for each* $P \in Prc^*$.

(3) *Time determinism: if* $P \xrightarrow{d} P'$ *and* $P \xrightarrow{d} P''$, *then* $P' = P''$.

(4) *Persistency of action transitions: for all* $P, Q \in Prc^*$, $\alpha \in Act$ *and* $d \in \mathbb{R}_{\geq 0}$, *if* $P \xrightarrow{\alpha}$ *and* $P \xrightarrow{d} Q$, *then* $Q \xrightarrow{\alpha}$.

(1)–(3) implies that the semantics of a TCCS process is indeed a TLTS (Def. 13.4).

Proof.

$$\text{(tAdd)}\frac{P \xrightarrow{d'} P'}{\varepsilon(d).P \xrightarrow{d+d'} P'} \quad \text{(tSub)}\frac{(d' \leq d)}{\varepsilon(d).P \xrightarrow{d'} \varepsilon(d-d').P} \quad \text{(tAct)}\frac{(\alpha \neq \tau)}{\alpha.P \xrightarrow{d} \alpha.P} \quad \text{(tTau)}\frac{}{\tau.P \xrightarrow{0} \tau.P} \quad \text{(tSum)}\frac{P \xrightarrow{d} P' \quad Q \xrightarrow{d} Q'}{P+Q \xrightarrow{d} P'+Q'}$$

By induction on derivation tree. Essential rules:

(1) (tAdd) and (tSub)

(2) (tSub), (tAct) and (tTau) (note that every P is guarded)

(3) (tSum) (and later (tPar) in Definition 13.12)

(4) all

Outline of Lecture 13

1. Real-Time Reactive Systems
2. CCS with Time Delays
3. Syntax of Timed CCS
4. Semantics of Timed CCS
5. Handling Parallel Composition

Example 13.11 (cf. Example 13.9)

$$\begin{aligned}
\mathit{Off} &= \mathit{press}.\mathit{Light} \\
\mathit{Light} &= \mathit{press}.\mathit{Bright} + \varepsilon(1.5).\tau.\mathit{press}.\mathit{Off} \\
\mathit{Bright} &= \mathit{press}.\mathit{Off} \\
\mathit{FastUser} &= \overline{\mathit{press}}.\varepsilon(0.3).\overline{\mathit{press}}.\mathrm{nil}
\end{aligned}$$

The Light Switch Example Revisited

Example 13.11 (cf. Example 13.9)

$$\begin{aligned} \mathit{Off} &= \mathit{press}.\mathit{Light} \\ \mathit{Light} &= \mathit{press}.\mathit{Bright} + \varepsilon(1.5).\tau.\mathit{press}.\mathit{Off} \\ \mathit{Bright} &= \mathit{press}.\mathit{Off} \\ \mathit{FastUser} &= \overline{\mathit{press}}.\varepsilon(0.3).\overline{\mathit{press}}.\mathrm{nil} \end{aligned}$$

- Expect immediate synchronisation between *FastUser* and *Off*:

$$(\mathit{FastUser} \parallel \mathit{Off}) \setminus \mathit{press} \xrightarrow{\tau} (\varepsilon(0.3).\overline{\mathit{press}}.\mathrm{nil} \parallel \mathit{Light}) \setminus \mathit{press}$$

Example 13.11 (cf. Example 13.9)

$$\begin{aligned} \mathit{Off} &= \mathit{press}.\mathit{Light} \\ \mathit{Light} &= \mathit{press}.\mathit{Bright} + \varepsilon(1.5).\tau.\mathit{press}.\mathit{Off} \\ \mathit{Bright} &= \mathit{press}.\mathit{Off} \\ \mathit{FastUser} &= \overline{\mathit{press}}.\varepsilon(0.3).\overline{\mathit{press}}.\mathrm{nil} \end{aligned}$$

- Expect immediate synchronisation between *FastUser* and *Off*:

$$(\mathit{FastUser} \parallel \mathit{Off}) \setminus \mathit{press} \xrightarrow{\tau} (\varepsilon(0.3).\overline{\mathit{press}}.\mathrm{nil} \parallel \mathit{Light}) \setminus \mathit{press}$$

- Now $\overline{\mathit{press}}$-transition only enabled after 0.3 time units – also a possible delay for *Light*:

$$\mathit{Light} \xrightarrow{0.3} \mathit{press}.\mathit{Bright} + \varepsilon(1.2).\tau.\mathit{press}.\mathit{Off}$$

The Light Switch Example Revisited

Example 13.11 (cf. Example 13.9)

$$\begin{aligned} \mathit{Off} &= \mathit{press}.\mathit{Light} \\ \mathit{Light} &= \mathit{press}.\mathit{Bright} + \varepsilon(1.5).\tau.\mathit{press}.\mathit{Off} \\ \mathit{Bright} &= \mathit{press}.\mathit{Off} \\ \mathit{FastUser} &= \overline{\mathit{press}}.\varepsilon(0.3).\overline{\mathit{press}}.\mathrm{nil} \end{aligned}$$

- Expect immediate synchronisation between *FastUser* and *Off*:

$$(\mathit{FastUser} \parallel \mathit{Off}) \setminus \mathit{press} \xrightarrow{\tau} (\varepsilon(0.3).\overline{\mathit{press}}.\mathrm{nil} \parallel \mathit{Light}) \setminus \mathit{press}$$

- Now $\overline{\mathit{press}}$-transition only enabled after 0.3 time units – also a possible delay for *Light*:

$$\mathit{Light} \xrightarrow{0.3} \mathit{press}.\mathit{Bright} + \varepsilon(1.2).\tau.\mathit{press}.\mathit{Off}$$

- Therefore expected that whole system can delay:

$$\begin{aligned} &(\varepsilon(0.3).\overline{\mathit{press}}.\mathrm{nil} \parallel \mathit{Light}) \setminus \mathit{press} \\ \xrightarrow{0.3}\ &(\overline{\mathit{press}}.\mathrm{nil} \parallel (\mathit{press}.\mathit{Bright} + \varepsilon(1.2).\tau.\mathit{press}.\mathit{Off})) \setminus \mathit{press} \end{aligned}$$

The Light Switch Example Revisited

Example 13.11 (cf. Example 13.9)

$$\begin{aligned} \mathit{Off} &= \mathit{press}.\mathit{Light} \\ \mathit{Light} &= \mathit{press}.\mathit{Bright} + \varepsilon(1.5).\tau.\mathit{press}.\mathit{Off} \\ \mathit{Bright} &= \mathit{press}.\mathit{Off} \\ \mathit{FastUser} &= \overline{\mathit{press}}.\varepsilon(0.3).\overline{\mathit{press}}.\mathrm{nil} \end{aligned}$$

- Now $\overline{\mathit{press}}$-transition only enabled after 0.3 time units – also a possible delay for Light:

$$\mathit{Light} \xrightarrow{0.3} \mathit{press}.\mathit{Bright} + \varepsilon(1.2).\tau.\mathit{press}.\mathit{Off}$$

- Therefore expected that whole system can delay:

$$\begin{aligned} &(\varepsilon(0.3).\overline{\mathit{press}}.\mathrm{nil} \parallel \mathit{Light}) \setminus \mathit{press} \\ \xrightarrow{0.3}\ &(\overline{\mathit{press}}.\mathrm{nil} \parallel (\mathit{press}.\mathit{Bright} + \varepsilon(1.2).\tau.\mathit{press}.\mathit{Off})) \setminus \mathit{press} \end{aligned}$$

- Now another synchronisation should be possible:

$$\begin{aligned} &(\overline{\mathit{press}}.\mathrm{nil} \parallel (\mathit{press}.\mathit{Bright} + \varepsilon(1.2).\tau.\mathit{press}.\mathit{Off})) \setminus \mathit{press} \qquad (*) \\ \xrightarrow{\tau}\ &(\mathrm{nil} \parallel \mathit{Bright}) \setminus \mathit{press} \end{aligned}$$

Example 13.11 (cf. Example 13.9)

$$\begin{aligned}
\mathit{Off} &= \mathit{press}.\mathit{Light}\\
\mathit{Light} &= \mathit{press}.\mathit{Bright} + \varepsilon(1.5).\tau.\mathit{press}.\mathit{Off}\\
\mathit{Bright} &= \mathit{press}.\mathit{Off}\\
\mathit{FastUser} &= \overline{\mathit{press}}.\varepsilon(0.3).\overline{\mathit{press}}.\mathrm{nil}
\end{aligned}$$

- Therefore expected that whole system can delay:

$$\begin{aligned}
&(\varepsilon(0.3).\overline{\mathit{press}}.\mathrm{nil} \parallel \mathit{Light}) \setminus \mathit{press}\\
\xrightarrow{0.3}\ &(\overline{\mathit{press}}.\mathrm{nil} \parallel (\mathit{press}.\mathit{Bright} + \varepsilon(1.2).\tau.\mathit{press}.\mathit{Off})) \setminus \mathit{press}
\end{aligned}$$

- Now another synchronisation should be possible:

$$\begin{aligned}
&(\overline{\mathit{press}}.\mathrm{nil} \parallel (\mathit{press}.\mathit{Bright} + \varepsilon(1.2).\tau.\mathit{press}.\mathit{Off})) \setminus \mathit{press} \qquad (*)\\
\xrightarrow{\tau}\ &(\mathrm{nil} \parallel \mathit{Bright}) \setminus \mathit{press}
\end{aligned}$$

- But: both parallel components of $(*)$ can delay for 1.2 time units, giving rise to

$$(*) \xrightarrow{1.2}\xrightarrow{\tau}\xrightarrow{\tau} (\mathrm{nil} \parallel \mathit{Off}) \setminus \mathit{press}$$

The Light Switch Example Revisited

Example 13.11 (cf. Example 13.9)

$$\begin{aligned} \mathit{Off} &= \mathit{press}.\mathit{Light} \\ \mathit{Light} &= \mathit{press}.\mathit{Bright} + \varepsilon(1.5).\tau.\mathit{press}.\mathit{Off} \\ \mathit{Bright} &= \mathit{press}.\mathit{Off} \\ \mathit{FastUser} &= \overline{\mathit{press}}.\varepsilon(0.3).\overline{\mathit{press}}.\mathrm{nil} \end{aligned}$$

- Now another synchronisation should be possible:

$$\begin{aligned} &(\overline{\mathit{press}}.\mathrm{nil} \parallel (\mathit{press}.\mathit{Bright} + \varepsilon(1.2).\tau.\mathit{press}.\mathit{Off})) \setminus \mathit{press} \qquad (*) \\ &\xrightarrow{\tau} (\mathrm{nil} \parallel \mathit{Bright}) \setminus \mathit{press} \end{aligned}$$

- But: both parallel components of $(*)$ can delay for 1.2 time units, giving rise to

$$(*) \xrightarrow{1.2} \xrightarrow{\tau} \xrightarrow{\tau} (\mathrm{nil} \parallel \mathit{Off}) \setminus \mathit{press}$$

- How to enforce that intended synchronisation occurs immediately?

The Maximal-Progress Assumption

Maximal-progress assumption

If a process is ready to perform an action that is entirely under its control, then it will immediately do so without further delay.

The Maximal-Progress Assumption

Maximal-progress assumption

If a process is ready to perform an action that is entirely under its control, then it will immediately do so without further delay.

In the setting of timed CCS, the only action that is entirely under the control of a process is the τ-action. Therefore:

Maximal-progress assumption for Timed CCS

For each TCCS process $P \in Prc^*$, if $P \xrightarrow{\tau}$ then $P \stackrel{d}{\not\longrightarrow}$ for any $d > 0$.

Operational Semantics with Maximal Progress

Definition 13.12 (Semantics of TCCS – timed parallel transitions)

Additionally for $P, P', Q, Q' \in Prc^*$ and $d \in \mathbb{R}_{\geq 0}$:

$$(\mathrm{tPar})\ \frac{P \xrightarrow{d} P' \quad Q \xrightarrow{d} Q' \quad NoSync(P, Q, d)}{P \parallel Q \xrightarrow{d} P' \parallel Q'}$$

where predicate $NoSync(P, Q, d)$ expresses that no synchronisation between P and Q becomes enabled by delaying less than d time units:

For each $0 \leq d' < d$ and $P', Q' \in Prc^*$,
if $P \xrightarrow{d'} P'$ and $Q \xrightarrow{d'} Q'$, then $P' \parallel Q' \stackrel{\tau}{\not\longrightarrow}$.

Definition 13.12 (Semantics of TCCS – timed parallel transitions)

Additionally for $P, P', Q, Q' \in Prc^*$ and $d \in \mathbb{R}_{\geq 0}$:

$$(\text{tPar})\,\frac{P \xrightarrow{d} P' \quad Q \xrightarrow{d} Q' \quad NoSync(P, Q, d)}{P \parallel Q \xrightarrow{d} P' \parallel Q'}$$

where predicate $NoSync(P, Q, d)$ expresses that no synchronisation between P and Q becomes enabled by delaying less than d time units:

For each $0 \leq d' < d$ and $P', Q' \in Prc^*$,
if $P \xrightarrow{d'} P'$ and $Q \xrightarrow{d'} Q'$, then $P' \parallel Q' \overset{\tau}{\not\longrightarrow}$.

Example 13.13 (cf. Example 13.11)

(1) $(\varepsilon(0.3).\overline{press}.\text{nil} \parallel Light) \setminus press \overset{d}{\not\longrightarrow}$ for any $d > 0.3$.

Operational Semantics with Maximal Progress

Definition 13.12 (Semantics of TCCS – timed parallel transitions)

Additionally for $P, P', Q, Q' \in Prc^*$ and $d \in \mathbb{R}_{\geq 0}$:

$$(\text{tPar})\ \frac{P \xrightarrow{d} P' \quad Q \xrightarrow{d} Q' \quad NoSync(P, Q, d)}{P \parallel Q \xrightarrow{d} P' \parallel Q'}$$

where predicate $NoSync(P, Q, d)$ expresses that no synchronisation between P and Q becomes enabled by delaying less than d time units:

For each $0 \leq d' < d$ and $P', Q' \in Prc^*$,
if $P \xrightarrow{d'} P'$ and $Q \xrightarrow{d'} Q'$, then $P' \parallel Q' \not\xrightarrow{\tau}$.

Example 13.13 (cf. Example 13.11)

(1) $(\varepsilon(0.3).\overline{press}.\text{nil} \parallel Light) \setminus press \not\xrightarrow{d}$ for any $d > 0.3$.

(2) $(\overline{press}.\text{nil} \parallel (press.Bright + \varepsilon(1.2).\tau.press.Off)) \setminus press \not\xrightarrow{d}$ for any $d > 0$.

Modelling a Slow User

Example 13.14 (cf. Example 13.11)

$$\begin{aligned}
Off &= press.Light \\
Light &= press.Bright + \varepsilon(1.5).\tau.press.Off \\
Bright &= press.Off \\
SlowUser &= \overline{press}.\varepsilon(1.7).\overline{press}.\mathrm{nil}
\end{aligned}$$

Modelling a Slow User

Example 13.14 (cf. Example 13.11)

$$\begin{aligned} \mathit{Off} &= \mathit{press}.\mathit{Light} \\ \mathit{Light} &= \mathit{press}.\mathit{Bright} + \varepsilon(1.5).\tau.\mathit{press}.\mathit{Off} \\ \mathit{Bright} &= \mathit{press}.\mathit{Off} \\ \mathit{SlowUser} &= \overline{\mathit{press}}.\varepsilon(1.7).\overline{\mathit{press}}.\mathrm{nil} \end{aligned}$$

- As before:

$$(\mathit{SlowUser} \parallel \mathit{Off}) \setminus \mathit{press} \xrightarrow{\tau} (\varepsilon(1.7).\overline{\mathit{press}}.\mathrm{nil} \parallel \mathit{Light}) \setminus \mathit{press}$$

Modelling a Slow User

Example 13.14 (cf. Example 13.11)

$$\begin{aligned} \mathit{Off} &= \mathit{press}.\mathit{Light} \\ \mathit{Light} &= \mathit{press}.\mathit{Bright} + \varepsilon(1.5).\tau.\mathit{press}.\mathit{Off} \\ \mathit{Bright} &= \mathit{press}.\mathit{Off} \\ \mathit{SlowUser} &= \overline{\mathit{press}}.\varepsilon(1.7).\overline{\mathit{press}}.\mathrm{nil} \end{aligned}$$

- As before:

$$(\mathit{SlowUser} \parallel \mathit{Off}) \setminus \mathit{press} \xrightarrow{\tau} (\varepsilon(1.7).\overline{\mathit{press}}.\mathrm{nil} \parallel \mathit{Light}) \setminus \mathit{press}$$

- Now $\overline{\mathit{press}}$-transition only enabled after 1.7 time units, but Light can only delay for at most 1.5 units:

$$\begin{aligned} &(\varepsilon(1.7).\overline{\mathit{press}}.\mathrm{nil} \parallel \mathit{Light}) \setminus \mathit{press} \xrightarrow{1.5} \\ &(\varepsilon(0.2).\overline{\mathit{press}}.\mathrm{nil} \parallel (\mathit{press}.\mathit{Bright} + \varepsilon(0).\tau.\mathit{press}.\mathit{Off})) \setminus \mathit{press} \qquad (*) \end{aligned}$$

Modelling a Slow User

Example 13.14 (cf. Example 13.11)

$$\begin{aligned} \mathit{Off} &= \mathit{press}.\mathit{Light} \\ \mathit{Light} &= \mathit{press}.\mathit{Bright} + \varepsilon(1.5).\tau.\mathit{press}.\mathit{Off} \\ \mathit{Bright} &= \mathit{press}.\mathit{Off} \\ \mathit{SlowUser} &= \overline{\mathit{press}}.\varepsilon(1.7).\overline{\mathit{press}}.\mathrm{nil} \end{aligned}$$

- As before:

$$(\mathit{SlowUser} \parallel \mathit{Off}) \setminus \mathit{press} \xrightarrow{\tau} (\varepsilon(1.7).\overline{\mathit{press}}.\mathrm{nil} \parallel \mathit{Light}) \setminus \mathit{press}$$

- Now $\overline{\mathit{press}}$-transition only enabled after 1.7 time units, but *Light* can only delay for at most 1.5 units:

$$\begin{aligned} &(\varepsilon(1.7).\overline{\mathit{press}}.\mathrm{nil} \parallel \mathit{Light}) \setminus \mathit{press} \xrightarrow{1.5} \\ &(\varepsilon(0.2).\overline{\mathit{press}}.\mathrm{nil} \parallel (\mathit{press}.\mathit{Bright} + \varepsilon(0).\tau.\mathit{press}.\mathit{Off})) \setminus \mathit{press} \qquad (*) \end{aligned}$$

- Here the right-hand process of $(*)$ can do a τ-action, disabling further delays and thus avoiding the *Bright* state:

$$(*) \xrightarrow{\tau} (\varepsilon(0.2).\overline{\mathit{press}}.\mathrm{nil} \parallel \mathit{press}.\mathit{Off}) \setminus \mathit{press}$$