



Concurrency Theory

Winter Semester 2017/18

Lecture 3: Hennessy-Milner Logic

Joost-Pieter Katoen and Thomas Noll
Software Modeling and Verification Group
RWTH Aachen University

<http://moves.rwth-aachen.de/teaching/ws-1718/ct/>

Recap: Calculus of Communicating Systems

Syntax of CCS I

Definition (Syntax of CCS)

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

$P ::= nil$	(inaction)
$\alpha.P$	(prefixing)
$P_1 + P_2$	(choice)
$P_1 \parallel P_2$	(parallel composition)
$P \setminus L$	(restriction)
$P[f]$	(relabelling)
C	(process call)

where $\alpha \in Act$, $L \subseteq A$, $C \in Pid$, and $f : Act \rightarrow Act$ such that $f(\tau) = \tau$ and $f(\bar{a}) = \overline{f(a)}$ for each $a \in A$.

Recap: Calculus of Communicating Systems

Syntax of CCS II

Definition (continued)

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

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

where $k \geq 1$, $C_i \in \mathit{Pid}$ (pairwise distinct), and $P_i \in \mathit{Prc}$ (with identifiers from $\{C_1, \dots, C_k\}$).

Notational Conventions:

- \bar{a} means a
- $\sum_{i=1}^n P_i$ ($n \in \mathbb{N}$) means $P_1 + \dots + P_n$ (where $\sum_{i=1}^0 P_i := \mathit{nil}$)
- $P \setminus a$ abbreviates $P \setminus \{a\}$
- $[a_1 \mapsto b_1, \dots, a_n \mapsto b_n]$ stands for $f : \mathit{Act} \rightarrow \mathit{Act}$ with $f(a_i) = b_i$ ($i \in [n]$) and $f(\alpha) = \alpha$ otherwise
- restriction and relabelling bind stronger than prefixing, prefixing stronger than composition, composition stronger than choice:

$$P \setminus a + b.Q \parallel R \quad \text{means} \quad (P \setminus a) + ((b.Q) \parallel R)$$

Recap: Calculus of Communicating Systems

Labelled Transition Systems

Goal: represent behaviour of system by (infinite) graph

- nodes = system states
- edges = transitions between states

Definition (Labelled transition system)

A (*Act*-)labelled transition system (LTS) is a triple $(S, Act, \longrightarrow)$ consisting of

- a set S of states
- a set Act of (action) labels
- a transition relation $\longrightarrow \subseteq S \times Act \times S$

For $(s, \alpha, s') \in \longrightarrow$ we write $s \xrightarrow{\alpha} s'$. An LTS is called **finite** if S is so.

Remarks:

- sometimes an **initial state** $s_0 \in S$ is distinguished (“ $LTS(s_0)$ ”)
- (finite) LTSs correspond to (finite) **automata** without final states

Recap: Calculus of Communicating Systems

Semantics of CCS I

We define the assignment

$$\begin{aligned} \text{syntax} &\rightarrow \text{semantics} \\ \text{process definition} &\mapsto \text{LTS} \end{aligned}$$

by induction over the syntactic structure of process expressions. Here we employ **derivation rules** of the form

$$\text{(rule name)} \frac{\text{premise(s)}}{\text{conclusion}}$$

which are composed to form **derivation trees** (where **axioms**, i.e., rules without premises, correspond to leaves).

Recap: Calculus of Communicating Systems

Semantics of CCS II

Definition (Semantics of CCS)

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

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

$$\text{(Sum}_1\text{)} \frac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'}$$

$$\text{(Sum}_2\text{)} \frac{Q \xrightarrow{\alpha} Q'}{P + Q \xrightarrow{\alpha} Q'}$$

$$\text{(Par}_1\text{)} \frac{P \xrightarrow{\alpha} P'}{P \parallel Q \xrightarrow{\alpha} P' \parallel Q}$$

$$\text{(Par}_2\text{)} \frac{Q \xrightarrow{\alpha} Q'}{P \parallel Q \xrightarrow{\alpha} P \parallel Q'}$$

$$\text{(Com)} \frac{P \xrightarrow{\lambda} P' \quad Q \xrightarrow{\bar{\lambda}} Q'}{P \parallel Q \xrightarrow{\tau} P' \parallel Q'}$$

$$\text{(Res)} \frac{P \xrightarrow{\alpha} P' \quad (\alpha, \bar{\alpha} \notin L)}{P \setminus L \xrightarrow{\alpha} P' \setminus L}$$

$$\text{(Rel)} \frac{P \xrightarrow{\alpha} P'}{P[f] \xrightarrow{f(\alpha)} P'[f]}$$

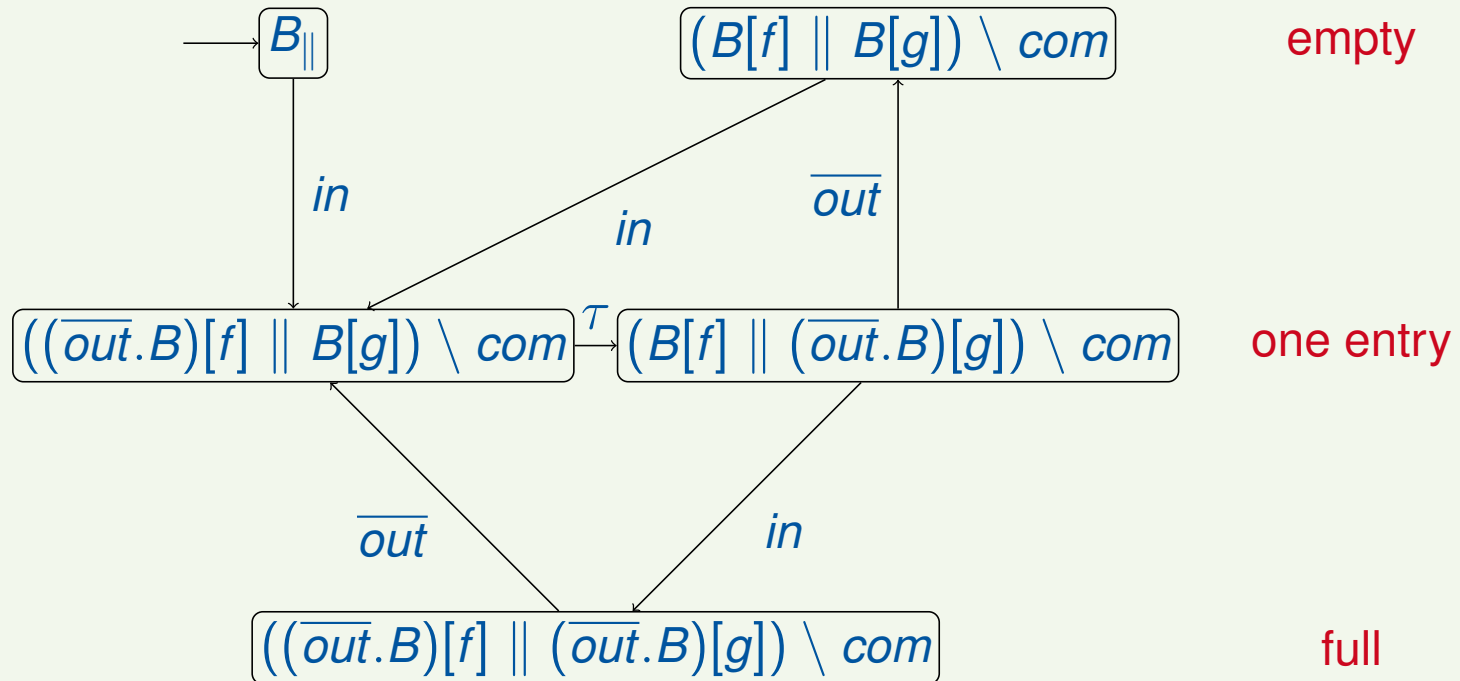
$$\text{(Call)} \frac{P \xrightarrow{\alpha} P' \quad (C = P)}{C \xrightarrow{\alpha} P'}$$

Recap: Calculus of Communicating Systems

Semantics of CCS II

Example

Parallel two-place buffer: $B_{||} = (B[f] \parallel B[g]) \setminus com$, $B = in.\overline{out}.B$
where $f := [out \mapsto com]$ and $g := [in \mapsto com]$



Infinite State Spaces

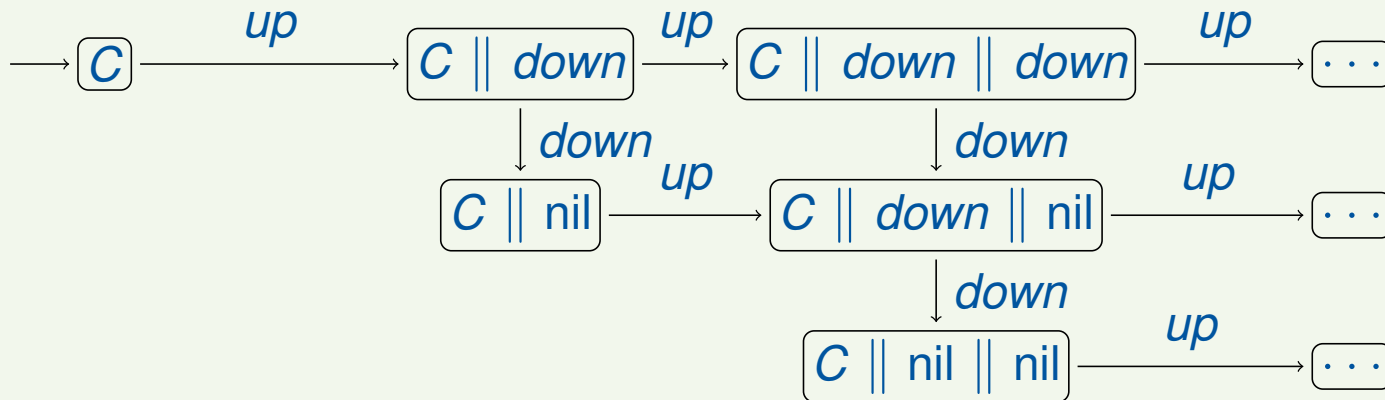
The Power of Recursive Definitions

So far: only **finite** state spaces

Example 3.1 (Counter)

$$C = up.(C \parallel down.nil)$$

gives rise to **infinite** LTS (abbreviating $down := down.nil$):



Sequential “specification”: $C_0 = up.C_1$

$$C_n = up.C_{n+1} + down.C_{n-1} \quad (n > 0)$$

Process Traces

Process Traces I

Goal: reduce processes to the action sequences they can perform

Definition 3.2 (Trace language)

For every $P \in Prc$, let

$$Tr(P) := \{w \in Act^* \mid \text{ex. } P' \in Prc \text{ such that } P \xrightarrow{w} P'\}$$

be the **trace language** of P (where $\xrightarrow{w} := \xrightarrow{a_1} \circ \dots \circ \xrightarrow{a_n}$ for $w = a_1 \dots a_n$).

$P, Q \in Prc$ are called **trace equivalent** if $Tr(P) = Tr(Q)$.

Example 3.3 (One-place buffer)

$$B = in.\overline{out}.B$$

$$\implies Tr(B) = (in \cdot \overline{out})^* \cdot (in + \varepsilon)$$

Process Traces II

Remarks:

- The trace language of $P \in \text{Prc}$ is accepted by the LTS of P , interpreted as a (finite or infinite) automaton with **initial state** P and where **every state is final**.
- Trace equivalence is obviously an **equivalence relation** (i.e., reflexive, symmetric, and transitive).
- Trace equivalence identifies processes with **identical LTSs**: the trace language of a process consists of the (finite) paths in the LTS. Thus:

$$LTS(P) = LTS(Q) \implies Tr(P) = Tr(Q)$$

- Later we will see: trace equivalence is **too coarse**, i.e., identifies too many processes
 \implies **bisimulation**

Motivation

Goal: check processes for **simple properties**

- action a is initially enabled
- action b is initially disabled
- a deadlock never occurs
- always sends a reply after receiving a request

Approach:

- Formalisation in **Hennessy-Milner Logic (HML)**
- M. Hennessy, R. Milner: *On Observing Nondeterminism and Concurrency*, ICALP 1980, Springer LNCS 85, 299–309
- Checking by **exploration of state space**

Syntax of HML

Definition 3.4 (Syntax of HML)

The set HMF of **Hennessy-Milner formulae** over a set of actions Act is defined by the following syntax:

$F ::= tt$	(true)
ff	(false)
$F_1 \wedge F_2$	(conjunction)
$F_1 \vee F_2$	(disjunction)
$\langle \alpha \rangle F$	(diamond)
$[\alpha] F$	(box)

where $\alpha \in Act$.

Meaning of HML Constructs

- All processes satisfy tt .
- No process satisfies ff .
- A process satisfies $F \wedge G$ iff it satisfies F and G .
- A process satisfies $F \vee G$ iff it satisfies either F or G or both.
- A process satisfies $\langle \alpha \rangle F$ for some $\alpha \in \text{Act}$ iff it affords an α -labelled transition to a state satisfying F (possibility).
- A process satisfies $[\alpha]F$ for some $\alpha \in \text{Act}$ iff all its α -labelled transitions lead to a state satisfying F (necessity).

Abbreviations for $L = \{\alpha_1, \dots, \alpha_n\}$ ($n \in \mathbb{N}$):

- $\langle L \rangle F := \langle \alpha_1 \rangle F \vee \dots \vee \langle \alpha_n \rangle F$
- $[L]F := [\alpha_1]F \wedge \dots \wedge [\alpha_n]F$
- In particular, $\langle \emptyset \rangle F := \text{ff}$ and $[\emptyset]F := \text{tt}$

Semantics of HML

Definition 3.5 (Semantics of HML)

Let $(S, Act, \longrightarrow)$ be an LTS and $F \in HMF$. The set of processes in S that **satisfy** F , $\llbracket F \rrbracket \subseteq S$, is defined by:

$$\begin{aligned} \llbracket tt \rrbracket &:= S & \llbracket ff \rrbracket &:= \emptyset \\ \llbracket F_1 \wedge F_2 \rrbracket &:= \llbracket F_1 \rrbracket \cap \llbracket F_2 \rrbracket & \llbracket F_1 \vee F_2 \rrbracket &:= \llbracket F_1 \rrbracket \cup \llbracket F_2 \rrbracket \\ \llbracket \langle \alpha \rangle F \rrbracket &:= \langle \cdot \alpha \cdot \rangle (\llbracket F \rrbracket) & \llbracket [\alpha] F \rrbracket &:= [\cdot \alpha \cdot] (\llbracket F \rrbracket) \end{aligned}$$

where $\langle \cdot \alpha \cdot \rangle, [\cdot \alpha \cdot] : 2^S \rightarrow 2^S$ are given by

$$\begin{aligned} \langle \cdot \alpha \cdot \rangle (T) &:= \{s \in S \mid \exists s' \in T : s \xrightarrow{\alpha} s'\} \\ [\cdot \alpha \cdot] (T) &:= \{s \in S \mid \forall s' \in S : s \xrightarrow{\alpha} s' \implies s' \in T\} \end{aligned}$$

We write $s \models F$ iff $s \in \llbracket F \rrbracket$. Two HML formulae are **equivalent** (written $F \equiv G$) iff they are satisfied by the same processes in every LTS.

Example 3.6 ($\langle \cdot \alpha \cdot \rangle, [\cdot \alpha \cdot]$ operators)

on the board

Simple Properties Revisited

Example 3.7

1. Action a is initially enabled: $\langle a \rangle \text{tt}$

$$\begin{aligned} \llbracket \langle a \rangle \text{tt} \rrbracket &= \langle \cdot a \cdot \rrbracket \llbracket \text{tt} \rrbracket = \langle \cdot a \cdot \rrbracket (S) \\ &= \{s \in S \mid \exists s' \in S : s \xrightarrow{a} s'\} =: \{s \in S \mid s \xrightarrow{a}\} \end{aligned}$$

2. Action b is initially disabled: $[b] \text{ff}$

$$\begin{aligned} \llbracket [b] \text{ff} \rrbracket &= [\cdot b \cdot] \llbracket \text{ff} \rrbracket = [\cdot b \cdot] (\emptyset) \\ &= \{s \in S \mid \forall s' \in S : s \xrightarrow{b} s' \implies s' \in \emptyset\} \\ &= \{s \in S \mid \nexists s' \in S : s \xrightarrow{b} s'\} =: \{s \in S \mid s \not\xrightarrow{b}\} \end{aligned}$$

3. Absence of deadlock:

- initially: $\langle \text{Act} \rangle \text{tt}$
- always: later (requires recursion)

4. Responsiveness:

- initially: $[request] \langle \overline{reply} \rangle \text{tt}$
- always: later (requires recursion)

Closure under Negation

Closure under Negation

Observation: **negation** is *not* one of the HML constructs

Reason: HML is **closed under negation**

Lemma 3.8

For every $F \in \text{HMF}$ there exists $F^c \in \text{HMF}$ such that $\llbracket F^c \rrbracket = S \setminus \llbracket F \rrbracket$ for every LTS $(S, \text{Act}, \longrightarrow)$.

Proof.

Definition of F^c :

$$\begin{array}{ll} \text{tt}^c := \text{ff} & \text{ff}^c := \text{tt} \\ (F_1 \wedge F_2)^c := F_1^c \vee F_2^c & (F_1 \vee F_2)^c := F_1^c \wedge F_2^c \\ (\langle \alpha \rangle F)^c := [\alpha] F^c & ([\alpha] F)^c := \langle \alpha \rangle F^c \end{array}$$

$\llbracket F^c \rrbracket = S \setminus \llbracket F \rrbracket$: on the board □

HML and Process Traces

HML and Process Traces

Lemma 3.9

Let $(Prc, Act, \longrightarrow)$ be an LTS, and let $P, Q \in Prc$ satisfy the same HMF (i.e., $\forall F \in HMF : P \models F \iff Q \models F$). Then $Tr(P) = Tr(Q)$.

Proof.

on the board □

Remark: the converse does *not* hold.

Example 3.10

- Let $P := a.(b.nil + c.nil) \in Prc$, $Q := a.b.nil + a.c.nil \in Prc$
- Then $Tr(P) = Tr(Q) = \{\varepsilon, a, ab, ac\}$
- Let $F := [a](\langle b \rangle tt \wedge \langle c \rangle tt) \in HMF$
- Then $P \models F$ but $Q \not\models F$
- [Later: $P, Q \in Prc$ HML-equivalent iff bismilar]