# 1 LTS; ACP

**LTS and Process Graphs** Both specifications and implementations could be represented by *models of concurrency*, for example *labelled transition systems (LTS)* or *process graphs.*

**Definition 1.1 (Process Graph)** *A process graph is a triple* $(S, I, \twoheadrightarrow)$ *such that:*

- $S$ *a set of states;*
- $I \in S$ *an initial state;*
- $\twoheadrightarrow$ *a set of triples* $(s, a, t)$ *each describing a (named) relation* $S \to S$*:*
  - $s, t \in S$*;*
  - $a \in Act$ *– a set of actions.*

**Definition 1.2 (LTS)** *Same as process graph, except without an initial state. Sometimes used synonymously with process graphs bc. mathematicians are evil.*

Alternatively, one may use *process algebraic expressions* to formally represent spec.s and impl.s, for example using *CCS (Calculus of Communicating Systems)*, *CSP (Communicating Sequential Processes)*, and *ACP (Algebra of Communicating Processes)*. Each semantics is of different expressive power.

**ACP** Define the set of operations:

- $\varepsilon$ (successful termination – ACP$_\varepsilon$ extension).
- $\delta$ (deadlock).
- $a$ (action constant) for each action $a \in Act$.

  Each $a$ describe a **visible action** – $\tau \notin Act$;
- $P \cdot Q$ (sequential composition between processes $P, Q$)
- $P + Q$ (summation / choice / alternative composition);
- $P||Q$ (parallel composition).
- $\partial_H(P)$ (restriction / encapsulation).

  Given set of (visible) actions $H$, this removes $\forall a \in H$ in $P$.

  Practically this is often used after defining $\gamma(a, b)$ to enforce sync – via removing non-synced $a.b$ or $b.a$ behaviors;
- $\tau_I(P)$ (abstraction – ACP$_\tau$ extension).

  Given set of (visible) actions $I$, this converts $\forall a \in I$ into $\tau$ in $P$.

  A $\tau$ action is **non-observable** – this will be significant for describing traces & equivalence relations.
- $\gamma : A \times A \to A$ (partial communication function).

  For example, $\gamma(a, b)$ defines new (synchronized) visible action alongside $a, b$.

We further define the following transition rules (omitting commutative equivalents). First, transition rules for basic process algebra wrt. termination, sequential composition, and choice:

$$\frac{}{a \xrightarrow{a} \varepsilon} \qquad \frac{a \xrightarrow{a} \varepsilon}{a + b \xrightarrow{a} \varepsilon} \qquad \frac{a \xrightarrow{a} \varepsilon}{a \cdot b \xrightarrow{a} b}$$

$$\frac{a \xrightarrow{a} a'}{a + b \xrightarrow{a} a'} \qquad \frac{a \xrightarrow{a} a'}{a \cdot b \xrightarrow{a} a' \cdot b}$$

Then, for parallel processes which may or may not communicate:

$$\frac{a \xrightarrow{a} \varepsilon}{a||b \xrightarrow{a} b} \qquad \frac{a \xrightarrow{a} a'}{a||b \xrightarrow{a} a'||b}$$

$$\frac{a \xrightarrow{a} \varepsilon \quad b \xrightarrow{b} \varepsilon}{a||b \xrightarrow{\gamma(a,b)} \varepsilon} \qquad \frac{a \xrightarrow{a} a' \quad b \xrightarrow{b} \varepsilon}{a||b \xrightarrow{\gamma(a,b)} a'}$$

$$\frac{a \xrightarrow{a} \varepsilon \quad b \xrightarrow{b} b'}{a||b \xrightarrow{\gamma(a,b)} b'} \qquad \frac{a \xrightarrow{a} a' \quad b \xrightarrow{b} b'}{a||b \xrightarrow{\gamma(a,b)} a'||b'}$$

Furthermore, for encapsulation $\partial_H$:

$$\frac{a \xrightarrow{x} \varepsilon}{\partial_H(a) \xrightarrow{x} \varepsilon} x \notin H \qquad \frac{a \xrightarrow{x} a'}{\partial_H(a) \xrightarrow{x} \partial_H(a')} x \notin H$$

This is to say, $\partial_H(a)$ can execute all transitions of $a$ that are also not in $H$.

Finally, deadlocks **does not display any behavior** – that is, a $\delta$ process cannot transition to any other states no matter what (though obviously as a constituent part of e.g., a parallel process the other concurrent constituent can still run).

**Background 1.1 (commutativity)**

$$f(a, b) = f(b, a) \iff f \text{ commutative}$$

**Background 1.2 (associativity)**

$$(a \circ b) \circ c = a \circ (b \circ c) \iff \circ \text{ associative}$$

**Background 1.3 (distributivity)**

$$f(x, a \circ b) = f(x, a) \circ f(x, b) \iff f \text{ distributes over } \circ$$

**Background 1.4 (isomorphism)** *An isomorphism describes a* ***bijective homomorphism****:*

- ***Homomorphism*** *describes a* ***structure-preserving*** *map between two algebraic* ***structures*** *of the same* ***type****:*
  - ***Algebraic structure*** *describes a set with additional properties – e.g., an additive group over* $\mathbb{N}$*, a ring of integers modulo* $x$*, etc.*
  - *Two* ***structures*** *of the same* ***type*** *refers to structures with the same name (i.e., class of property) – e.g., two groups, two rings, etc.*
  - *A* ***structure-preserving*** *map* $f$ *between two structures intuitively describes a structure such that, for properties* $p \in X$*,* $q \in Y$ *between same-type structures* $X, Y$*, any tuples* $X^n \in p$ *accepted by* $p$ *(e.g.,* $3 + 5 = 8 \implies (3, 5, 8) \in \mathbb{R}.(+)$*) satisfies* $\mathsf{map}(f, X^n) \in q$*.*
- ***Bijection*** *describes a 1-to-1 correspondence between elements of two sets – i.e., invertible.*

# 2 Semantic Equivalences

**Background 2.1 (lattice)** *A* ***lattice*** *describes a real coordinate space* $\mathbb{R}^n$ *that satisfies:*

- *Addition / subtraction between two points always produce another point in lattice – i.e., closed under addition / subtraction.*
- *Lattice points are separated by bounded distances in some range* $(0, \max]$*.*

Define a lattice over which *semantic equivalence relations* for spec. and impl. verification is defined.

**Background 2.2 (reflexivity)**

$$\forall x \in X : x \circ x \iff \circ \text{ reflexive on } X$$

**Background 2.3 (symmetry)**

$$\forall x, y \in X : \frac{x \circ y}{y \circ x} \iff \circ \text{ symmetric on } X$$

**Background 2.4 (transitivity)**

$$\forall x, y, z \in X : \frac{x \circ y \quad y \circ z}{x \circ z} \iff \circ \text{ transitive on } X$$

**Background 2.5 (equivalence relation)** ***Equivalence relation*** *on set $X$ satisfies reflexivity, symmetry, and transitivity on $X$.*

**Definition 2.1 (discrimination measure)** *One equivalence relation $\equiv$ is* ***finer*** */* ***more discriminating*** *than another $\sim$ if each $\equiv$-eq. class is a subset of a $\sim$-eq. class. In other words,*

$$p \equiv q \implies p \sim q$$
$$\iff \quad \equiv \text{ finer than } \sim$$

*In other words, $\equiv$ creates finer partitions on its domain compared to $\sim$.*

## Trace Equivalence

**Definition 2.2 (path)** *A* ***path*** *of a process $p$ is an alternating sequence of states and transitions starting from state $p$. It can be infinite or ending in a state.*

*A path is* ***complete*** *if it is either infinite or ends in a state where no further transitions are possible – a maximal path.*

**Definition 2.3 (complete trace)** *A* ***complete trace*** *of process $p$ is the sequence of labels of transitions in a complete path.*

*The set of finite complete traces of process $p$ is denoted as $CT^{fin}(p)$, while the set of all finite/infinite complete traces of $p$ is $CT^{\infty}(p)$ – aka. $CT(p)$ from now on.*

**Example 2.1** ($CT^{\infty}$)

$$CT^{\infty}(a.(\varepsilon + b.\delta)) = \{a\checkmark, ab\}$$

**Definition 2.4 (partial trace)** *Likewise, a* ***partial trace*** *of a process $p$ is the sequence of labels of transitions in any partial path.*

*We also likewise define $PT^{fin}(p)$ and $PT^{\infty}(p)$ for some process $p$. Define $PT(p)$ as $PT^{fin}(p)$.*

**Definition 2.5 ($=_{PT}$)** *Processes $p, q$ are* ***partial trace equivalent*** *($p =_{PT} q$) if they have the same partial traces:*

$$p =_{PT} q \iff PT(p) = PT(q)$$

*Mirroring the differences between $PT^{fin}$ and $PT^{\infty}$, define* ***finitary partial trace equivalence*** *($=_{PT^{fin}}$) and* ***infinitary partial trace equivalence*** *($=_{PT^{\infty}}$).*

**Definition 2.6 ($=_{CT}$)** *Processes $p, q$ are* ***complete trace equivalent*** *($p =_{CT} q$) if <u>moreover</u> they have the same complete traces:*

$$p =_{CT} q \iff CT(p) = CT(q)$$

*Mirroring the differences between $CT^{fin}$ and $CT^{\infty}$, define* ***finitary complete trace equivalence*** *($=_{CT^{fin}}$) and* ***infinitary complete trace equivalence*** *($=_{CT^{\infty}}$).*

## Weak Equivalences and $\tau$-actions

**Definition 2.7 (strong equivalence)** *A* ***strong equivalence*** *relation treats $\tau$ like any other (observable) action.*

*We assume above definitions for e.g., $=_{PT}$ to be assuming strong equivalence.*

**Definition 2.8 (weak equivalence)** *In its mirror case, a* ***weak equivalence*** *treats $\tau$ as if it is omitted from the input processes.*

*We additionally define weak variants of the above 4 equivalences: $=_{WPT^{fin}}$, $=_{WPT^{\infty}}$, $=_{WCT^{fin}}$, $=_{WCT^{\infty}}$.*

## Bisimulation Equivalence

**Definition 2.9 (bisimulation)** *Let $A, P$ define the actions and predicates of an LTS (in addition to states, etc.). A* ***bisimulation*** *is a binary relation $\circ \subseteq S \times S$ satisfying:*

- $s \circ t \implies (\forall p \in P : s \models p \iff t \models p)$
- $s \circ t \wedge (\exists a \in A : s \xrightarrow{a} s') \implies (\exists t' : t \xrightarrow{a} t') \wedge s' \circ t'$
- $s \circ t \wedge (\exists a \in A : t \xrightarrow{a} t') \implies (\exists s' : s \xrightarrow{a} s') \wedge s' \circ t'$

*Bisimulation (aka.* ***bisimulation equivalence****) is an equivalence relation. In general, bisimulation differentiates branching structure of processes.*

**Definition 2.10 (bisimilarity)** *Two states $s, t$ are bisimilar ($s \underline{\leftrightarrow} t$) if such a bisimulation $\circ$ exists between $s, t$.*

**Definition 2.11 (branching bisimulation)** *Given $A, P$ upon LTS, weaken <u>bisimulation</u> as follows: a* ***branching bisumlation*** *is a binary relation $\circ \subseteq S \times S$ satisfying:*

1. $s \circ t \wedge (\exists p \in P : s \models p) \implies \exists t_1 : t \rightsquigarrow t_1 \models p \wedge s \circ t_1$
2. $s \circ t \wedge (\exists p \in P : t \models p) \implies \exists s_1 : s \rightsquigarrow s_1 \models p \wedge s_1 \circ t$
3. $s \circ t \wedge (\exists a \in A_\tau : s \xrightarrow{a} s')$ $\implies \exists t_1, t_2, t' : t \rightsquigarrow t_1 \xrightarrow{(a)} t_2 = t' \wedge s \circ t_1 \wedge s' \circ t'$
4. $s \circ t \wedge (\exists a \in A_\tau : t \xrightarrow{a} t')$ $\implies \exists s_1, s_2, s' : s \rightsquigarrow s_1 \xrightarrow{(a)} s_2 = s' \wedge s_1 \circ t \wedge s' \circ t'$

*where:*

- $s \rightsquigarrow s'$ $\iff \exists n \geq 0 : \exists s_0, \ldots, s_n : s = s_0 \xrightarrow{\tau} \ldots \xrightarrow{\tau} s_n = s'$
- $A_\tau := A \cup \{\tau\}$
- $s \xrightarrow{(a)} s' := \begin{cases} s \xrightarrow{a} s' & \text{if } a \in A \\ s \xrightarrow{\tau} s' \vee s = s' & \text{if } a = \tau \end{cases}$

*Two processes $p, q$ are branching bisimilar ($p \underline{\leftrightarrow}_b t$) if such a binary relation $\circ$ exists.*

**Definition 2.12 (delay bisimulation)** *Given $\underline{\leftrightarrow}_b$, drop requirements $s \circ t_1$ and $s_1 \circ t$, thus producing $\underline{\leftrightarrow}_d$.*

**Definition 2.13 (weak bisimulation)** *Given $\underline{\leftrightarrow}_b$,*

- *Drop requirements $s \circ t_1$, $s_1 \circ t$;*
- *Relax $t_2 = t'$ and $s_2 = s'$ to $t_2 \rightsquigarrow t'$ and $s_2 \rightsquigarrow s'$, respectively.*

*Thus producing $\underline{\leftrightarrow}_w$.*

**Language Equivalence** This paragraph is moved here for ergonomics.

**Definition 2.14 (language equivalence)** *Processes $p, q$ are* ***language equivalent*** *if they have the same traces leading to terminating states – i.e., equal subset of terminating partial traces.*

*Intuitively (and indeed) this is coarser than partial trace equivalence.*

**Overview: The Hasse Diagram** ...

# 3 CCS; SOS

**CCS** Define the set of operations and semantics:

- 0 (inaction):

  0 represents a graph with 1 (initial) state, 0 transitions.

- $a, \bar{a}$ (complementary actions):

  Complementary actions are assumed to communicate / synchronize with one another.

- $a.P$ (action prefix) for each action $a$, process $P$, which:

  1. Define new initial state $\mathsf{i}$.
  2. Creates transition $\mathsf{i} \xrightarrow{a} I_P$.

- $P + Q$ (summation / choice / alternative composition), where:

  – Define new initial state $\mathsf{root}$.

  – $\text{States}(P+Q) \coloneqq \text{States}(P) \cup \text{States}(Q) \cup \{\mathsf{root}\}$

  – Replace all $I_P \xrightarrow{a} s$ with $\mathsf{root} \xrightarrow{a} s$.

  – Replace all $I_Q \xrightarrow{a} s$ with $\mathsf{root} \xrightarrow{a} s$.

- $P|Q$ (parallel composition).

  This takes the cartesian product of the states of $P, Q$, such that:

  – $s \xrightarrow{a} s' \in P \implies \forall t \in Q : (s,t) \xrightarrow{a} (s',t)$

  – $t \xrightarrow{a} t' \in Q \implies \forall s \in P : (s,t) \xrightarrow{a} (s,t')$

  – $(s \xrightarrow{a} s' \in P) \wedge (t \xrightarrow{\bar{a}} t' \in Q) \implies (s,t) \xrightarrow{\tau} (s',t')$

  Note that **CCS adheres strictly to a handshaking communication format** – this differs from ACP which gives greater leeway to implementation, via the use of $\gamma$ operator.

- $P \backslash a$ (restriction) for each action $a$.

  This produces copy of $P$ such that all actions $a, \bar{a}$ are omitted. This is useful to remove unsuccessful communication.

- $P[f]$ (relabelling) for each function $f : A \to A$.

  This replaces each label $a, \bar{a}$ by $f(a), \overline{f(a)}$.

**Recursion**

**Definition 3.1 (process names and expressions)** *Suppose we bind names $X, Y, Z, \ldots$ to some expression in the CCS language:*

$$X = P_X$$

*Here, $P_X$ represents ANY expression in the language, possibly including $X$.*

*It is trivial to see this can cause recursive definitions:*

$$X = a.X$$

**Definition 3.2 (recursive specification)** *Define **recursive specification** as partial function $s : X \to E$:*

- *$X$: **recursion variables**.*
- *$E$: **recursion equations** of form $x = P_x$.*

*In general, recursive spec.s are written as follows:*

$$\langle x | s \rangle$$

*which reads as "process $x$ satisfying equation $s$".*

**Definition 3.3 (guarded recursion)** *A recursion is **guarded** if each occurrence of a process name in $P_X$ occurs within the scope of a subexpression $a.P'_X$.*

*Think of it as being unwind-able such that progress is guaranteed.*

**Structural Operational Semantics (CCS)**

$$\frac{}{a.E \xrightarrow{a} E} \qquad \frac{E_j \xrightarrow{a} E'_j}{\sum_{i \in I} E_i \xrightarrow{a} E'_j} \; (j \in I)$$

$$\frac{E \xrightarrow{a} E'}{E|F \xrightarrow{a} E'|F} \qquad \frac{E \xrightarrow{a} E' \quad F \xrightarrow{\bar{a}} F'}{E|F \xrightarrow{\tau} E'|F'}$$

$$\frac{E \xrightarrow{a} E' \quad a \notin L \cup \overline{L}}{E \backslash L \xrightarrow{a} E' \backslash L} \qquad \frac{E \xrightarrow{a} E'}{E[f] \xrightarrow{f(a)} E'[f]}$$

# 4 Equational Axiomisation

**Congruence** If an equivalence relation is a *congruence* for an operator – i.e., an operator is *compositional* for the equivalence – then there exists a sort of isomorphism detailed as follows:

**Definition 4.1 (congruence)** *An equivalence $\sim$ is a **congruence** for a language $\mathcal{L}$ if:*

$$\forall C[\,] \in \mathcal{L} : P \sim Q \implies C[P] \sim C[Q]$$

*where:*

- *$C[\,]$ (context) represents a $\mathcal{L}$-expression with a hole in it, plugged (e.g., with $P$) as $C[P]$.*

*For example, let $P = a.[\,]$:*

$$\frac{P = Q}{a.P = a.Q}$$

*Equivalently, we can say that $CCP.(.)$ is compositional under equality ($=$).*

**Example 4.1 ($=_{CT}$ and $\partial_H$)** *This is a counterexample for showing why $=_{CT}$ is NOT a congruence over ACP. Obviously:*

$$a.b + a.c =_{CT} a.(b+c)$$

*However:*

$$\partial_{\{c\}}(a.b + a.c) \neq_{CT} \partial_{\{c\}}(a.(b+c))$$

**Definition 4.2 (congruence closure)** *A **congruence closure** $\sim^c$ of $\sim$ wrt. language $\mathcal{L}$ is defined by:*

$$P \sim^c Q \iff \forall C[\,] \in \mathcal{L} : C[P] \sim C[Q]$$

**Equational Axiomisation** In terms of e.g., real addition we describe the operator as possessing e.g., associativity and commutativity, which in turn allows us to do some transformation during analysis, etc.

Same goes for operators in e.g, CCS:

$$\begin{aligned} (P+Q)+R &= P+(Q+R) && \text{(associativity)} \\ P+Q &= Q+P && \text{(commutativity)} \\ P+P &= P && \text{(idempotence)} \\ P+0 &= P && \text{(0 as neutral element of +)} \end{aligned}$$

**Definition 4.3 (CCS: expansion theorem)** *Suppose:*

$$P \coloneqq \sum_{i \in I} a_i.P_i$$

$$Q \coloneqq \sum_{j \in J} b_i.Q_j$$

*Then,*

$$P|Q = \sum_{i \in I} a_i(P_i|Q) + \sum_{i \in I, j \in J} \tau(P_i|Q_j) \text{ (given } a_i = \overline{b_j}) + \sum_{j \in J} b_i(P|Q_j)$$

*Expressions of the form* $\sum a.P$ *are aka.* ***head normal form****.*

**Definition 4.4 (Recursive Definition Principle)**

$$i \in [1,n] : \langle X_i|E\rangle \in \mathsf{Expr}(X_1 \coloneqq \langle X_1|E\rangle, \ldots, X_n \coloneqq \langle X_n|E\rangle)$$

*Basically, some series of expressions for* $X_1, \ldots, X_n$ *exists as solution for* $E$*.*

**Definition 4.5 (Recursive Specification Principle)** *If there* *exists*

$$i \in [1,n] : y_i \leftarrow \mathsf{Expr}(y_1, \ldots, y_n)$$

*then:*

$$i \in [1,n] : y_i = \langle X_i|E\rangle$$

*In other words, any* $y_{1\ldots n}$ *that exists is the sole solution for* $E$ *modulo bisimulation equivalence.*

**Rooted Bisimilarity** We note that depending on semantics of $\mathcal{L}$, equivalences may (and in fact likely) fail to be a congurence over $\mathcal{L}$. This also is the case for e.g., branching bisimilarity: $\tau.a =_{BB} a$ but $\tau.a + b \neq_{BB} a + b$.

ACP and CCS fixes this by changing the equivalence operator.

**Definition 4.6 (Rooted Branching Bisimilarity)**

$$P =_{rBB} Q \iff (P \xrightarrow{a} P' \implies Q \xrightarrow{a} Q' \wedge P' =_{BB} Q') \wedge (Q \xrightarrow{a} Q' \implies P \xrightarrow{a} P' \wedge P' =_{BB} Q')$$

**Definition 4.7 (Rooted Weak Bisimilarity)**

$$P =_{rWB} Q \iff (P \xrightarrow{a} P' \implies Q \xrightarrow{a} Q' \wedge P' =_{WB} Q') \wedge (Q \xrightarrow{a} Q' \implies P \xrightarrow{a} P' \wedge P' =_{WB} Q')$$