Specification vs. Implementation *Specification* describes what a system ought to satisfy and perform. **A *formal specification***, in particular, **is a specification derived using formal methods that ensure the required properties of some problem at hand**. A formal specification of a distributed system often comes in (at least) 2 parts:

1. ***Requirements* imposed on the system – i.e., a list of properties that the system should satisfy (e.g., safety / liveness properties).**

2. ***Operations* of the system, which describes the behavior (i.e., satisfiability of predicates) given interactions (e.g., effects of communication).**

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 0.1 (Process Graph) *A process graph is a triple* $(S, I, \multimap)$ *such that:*

- S *a set of states;*
- $I \in S$ *an initial state;*
- $\multimap$ *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 0.2 (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:

- ε (successful termination – ACP_ε extension).
- δ (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_τ extension).
  Given set of (visible) actions I, this converts $\forall a \in I$ into τ in P.
  A τ 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 reflexive 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,

Background 0.1 (commutativity)

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

Background 0.2 (associativity)

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

Background 0.3