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:

- 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 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 ∂_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 δ 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,*

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

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^{∞})

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

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^{∞}, 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 moreover they have the same complete traces:*

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

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

Weak Equivalences and τ-actions

Definition 2.7 (strong equivalence) *A **strong equivalence** relation treats τ 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 τ 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 bisimulation 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:

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

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 i.
  2. Creates transition $\mathsf{i} \xrightarrow{a} I_P$.

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

  - Define new initial state 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 γ 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

Congurence If an equivalence relation is a *congurence* 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 (congurence) *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 ∂_H) *This is a counterexample for showing why $=_{CT}$ is NOT a congurence 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 (congurence closure) *A **congurence 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:

$(P + Q) + R = P + (Q + R)$	(associativity)
$P + Q = Q + P$	(commutativity)
$P + P = P$	(idempotence)
$P + 0 = P$	(0 as neutral element of +)

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

$$P \coloneqq \textstyle\sum_{i \in I} a_i.P_i \quad Q \coloneqq \sum_{j \in J} b_j.Q_j$$

Then,

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

*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 congruence 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. CSP fixes this by foregoing the + operator.

Definition 4.6 (Rooted Branching Bisimilarity)

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

$=_{rBB}$ is equivalent to branching bisimulation congruence $=^c_{BB}$ over ACP.

Definition 4.7 (Rooted Weak Bisimilarity)

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

$=_{rWB}$ is equivalent to weak bisimulation congruence $=^c_{WB}$ over CCS.

Definition 4.8 (Eq. axiomisation for rBB, rWB) *$=_{rWB}$ is axiomatised as follows:*

$$a.\tau.P = a.P \tag{1}$$
$$\tau.P = \tau.P + P \tag{2}$$
$$a.(\tau.P + Q) = a.(\tau.P + Q) + a.P \tag{3}$$

$=_{rBB}$ is axiomatised as follows:

$$a.(\tau.(P+Q)+Q) = a.(P+Q)$$

Strongly/Weakly Guarded Recursions Recall that a recursive spec of the form e.g. $X = a.(b + X)$ is guarded – X exists as a subexpression of X – and are equivlent modulo strong bisimilarity (viz. RSP).

On the other hand $X = \tau.X$ has solely equivalent solutions modulo $=_B$ but not up to e.g. $=_{rBB}$. This breaks the equivalence lattice – we hence need a stronger concept of unguardedness for $=_B$.

Definition 4.9 (strong unguardedness) *A **strongly unguarded** recursive specification is one where, for*

$$X = \mathsf{Expr}(X, \ldots)$$

the recursive variable X occurs NOT in a subterm of the form:

$$a \leftarrow A \cup \{\tau\} : a.P', X \in P'$$

as in, X is not guarded by $\forall a \in A$ nor τ.

*It turns out that **RSP is sound modulo bisimulation for all non-strongly-unguarded recursive specifications**.*

Definition 4.10 (weak unguardedness) *Likewise, a **weakly unguarded** recursive specification is one where recursive variable X is NOT guarded by $\forall a \in A$ only.*

Note that strong unguardedness entails weak unguardedness.

***RSP is sound modulo weak/branching bisimulation for all non-weakly-unguarded recursive specifications**.*

5 CSP; SOS

CSP Introduce the following operations:

1. 0 or STOP (inaction).

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

2. $a.P$ or $a \to P$ (action prefix) for $\forall a \in A$.

   Likewise CCS.

3. $P \Box Q$ (external choice).

   Semantically it is similar to parallel composition without synchronization, where:

   - Prior to "choice", one of the two actions might happen between the processes.
   - After one of the action happens, only the choiced process may occur at runtime.

4. $P \sqcap Q$ (internal choice):

   $$\mathrm{CSP}[P \sqcap Q] \equiv \mathrm{CCS}[\tau.P + \tau.Q]$$

5. $P||_S Q$ (parallel composition) with enforced synchronization over $S \subseteq A$. Semantically speaking:

   - $\mathsf{States}(P||_S Q) \coloneqq \mathsf{States}(P) \times \mathsf{States}(Q)$
   - $(s,t) \xrightarrow{a} (s',t)$ if $(a \notin S) \wedge (s \xrightarrow{a} s' \in P)$
   - $(s,t) \xrightarrow{a} (s,t')$ if $(a \notin S) \wedge (t \xrightarrow{a} t' \in Q)$
   - $(s,t) \xrightarrow{a} (s',t')$ if $(a \in S) \wedge (s \xrightarrow{a} s' \in P) \wedge (t \xrightarrow{a} t' \in Q)$

6. P/a (concealment).

   Like CCS, rename a into τ.

7. $P[f]$ (renaming) for $f \in (A \to A)$

   Likewise CCS.

Weak and branching bisimulation are congurences for CSP.

GSOS As a general form over languages, GSOS describes a transition rule of the following form: define

- Σ be the collection of function symbols wrt. a language.
- $\mathsf{arity} : \Sigma \to \mathbb{N}$ a function exposing the arity of the function symbol in question.

then, under GSOS semantics, the language could be expressed as follows:

$$\frac{x_i \xrightarrow{a} y_i, \ldots \quad (f \in \Sigma, i \in [1, \mathsf{arity}(f)], y_i \notin \mathsf{args}(f))}{f(x_1, \ldots, x_{\mathsf{arity}(f)}) \xrightarrow{a} \mathsf{Expr}(x_1, \ldots, x_{\mathsf{arity}(f)}, y_i, \ldots)}$$

It is the generalization of SOS rules we have covered earlier.

6 Hennessy-Miller Logic

HML Let φ, ψ be HML expressions:

$$\varphi, \psi ::= \top \mid \bot \mid \neg\varphi \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid \langle A \rangle \Phi \mid [A]\Phi$$

where Φ denotes predicates.

We omit explanation of familiar syntactic elements from FOL. Besides them:

$$P \models [A]\Phi \iff \forall Q : (\exists a \in A : P \xrightarrow{a} Q) \implies (Q \models \Phi)$$
$$P \models \langle A \rangle \Phi \iff \exists Q : \exists a \in A : P \xrightarrow{a} Q \wedge Q \models \Phi$$

Example 6.1 (deadlock in HML) *Given set of all actions A, a process P deadlocks if this holds:*

$$P \models [A]\bot$$

7 Preorder and Simulation

Preorder Remember equivalence? Meet its lesser sibling, preorder:

Definition 7.1 (Preorder) *A **preorder** ($\sqsubseteq$) denotes a transitive, reflexive relation on a set.*

Crucially, preorder is NOT symmetrical compared to equivalence.

For example, $\leq$ is a preorder over $\mathbb{R}$ (in fact, a partial-order).

We define preorders in relation of equivalences already defined in this course. For example, partial trace preorder $\sqsubseteq_{PT}$:

$$P \sqsubseteq_{PT} Q \iff PT(P) \supseteq PT(Q)$$

where Q becomes a **refinement** of P – all properties of P must hold for Q, while Q can hold more properties than P.

In general, we want to prove:

$$\text{Spec} \sqsubseteq_{\sim} \text{Impl}$$

Definition 7.2 (Kernel) *For each preorder $\sqsubseteq_{\sim}$ there exists an associated equivalence relation $\equiv_{\sim}$:*

$$P \equiv Q \iff P \sqsubseteq Q \wedge Q \sqsubseteq P$$

*If this holds, P, Q are **kernels** of each other.*

Simulation A **simulation** relation expresses a preorder between two processes P, Q such that:

$$P \sqsubseteq_S Q \iff \forall (P \xrightarrow{a} P') : \exists (Q \xrightarrow{a} Q') : P' \sqsubseteq_S Q'$$

Using definitions for general preorders, define **simulation equivalence** as follows:

$$P \equiv_S Q \iff P \sqsubseteq_S Q \wedge Q \sqsubseteq_S P$$

Example 7.1 ($\equiv_S$ vs. $=_B$) *Simulation equivalence is NOT equivalent to bisimulation. Case in point:*

$$P := a.b + a.(b + c)$$
$$Q := a.(b + c)$$

8 LTL; CTL; Kripke Structure

LTL Let ϕ, ψ be LTL expressions, $p \in AP$ an atomic predicate, $\rightarrow^*$ indicating 0 or more steps (like regex):

$$\begin{aligned}\phi, \psi ::= & \ p \mid \top \mid \bot \mid \phi \wedge \psi \mid \phi \vee \psi \mid \neg\phi \mid \phi \implies \psi \\ & \mid X\phi : \pi \models X\phi \iff \exists \pi' : \pi \rightarrow \pi' \models \phi \\ & \mid F\phi : \pi \models F\phi \iff \exists \pi' : \pi \rightarrow^* \pi' \models \phi \\ & \mid G\phi : \pi \models G\phi \iff \forall \pi' : (\pi \rightarrow^* \pi') \implies (\pi' \models \phi) \\ & \mid \phi U \psi : \pi_0 \models \phi U \psi \\ & \quad \iff \exists \pi_0 \rightarrow^* P \models \psi : \forall \pi_i \in \pi_0 \rightarrow^* \pi_N \rightarrow \Pi : \pi_i \models \phi \\ & \mid \phi W \psi : \pi \models \phi W \psi \iff \pi \models (\phi U \psi) \vee (G\phi)\end{aligned}$$

CTL Orthogonally (in terms of expressibility), CTL prefixes all $X|F|G|U$ operators with temporal signifiers:

$$A \sim \forall : \text{ for all paths}$$
$$E \sim \exists : \text{ exists some path}$$

Example 8.1 (incomparability of CTL, LTL) *No LTL equivalent exists for CTL formula $AG(EFa)$. Likewise, no CTL equivalent exists for LTL formula $F(Ga)$.*

Kripke Structure is an alternative means of representing a transition system. Let AP be a set of atomic predicates. A Kripke structure over AP is a tuple $(S, \rightarrowtriangle, \models)$ where:

- S: set of state.
- $\rightarrowtriangle \subseteq S \times S$: transition relation.
- $\models \subseteq S \times AP$: state-predicate mapping.

LTL and CTL operate upon Kripke structures.

A state $s \models \phi$ if for all paths incident from s, each path satisfies ϕ.

LTS-Kripke Translation A translation system η maps states in LTSs to states in Kripke structures. This allows us to perform LTL/CTL validation on LTS:

$$P \models_{\eta} \phi \iff \eta(P) \models \phi$$

Definition 8.1 (De Nicola-Vaandrager Translation) *The **DV-translation** translates process graphs into Kripke structures. Let the process graph be defineed as $(S, I, \rightarrowtriangle)$, as was the case for LTS.*

The associated Kripke structure is $((S', I), \rightarrowtriangle', \models)$, where:

$$\begin{aligned}S' &:= S \cup \{[s, a, t] \in \rightarrowtriangle \mid a \neq \tau\} \\ \rightarrowtriangle' &:= \{(s, [s, a, t]), ([s, a, t], t) \mid [s, a, t] \in S'\} \\ &\quad \cup \{(s, t) \mid (s, \tau, t) \in \rightarrowtriangle\}\end{aligned}$$

In short, we create new states out of transitions due to visible actions, and create linkages accordingly.

Theorem 8.1 *Processes P, Q satisfy the same LTL formulas if:*

$$P =_{CT}^{\infty} Q$$

*Moreover, finitely-branching processes P, Q satisfies **iff**.*

Theorem 8.2 *Processes P, Q satisfy the same LTL_{-X} formulas (i.e., LTL without X) if:*

$$P =_{WCT}^{\infty} Q$$

Theorem 8.3 *Processes P, Q satisfy the same CTL formulas if:*

$$P =_B Q$$

*Moreover, finitely-branching processes satisfies **iff**.*

*On the otherhand, when CTL is expanded to CTL^{∞} (i.e., CTL with infinite $\wedge$), all processes also satisfy **iff**.*

Theorem 8.4 *A **divergence-free** process is one where no reachable state p has $p \xrightarrow{\tau^{\infty}}!$ – infinite τs.*

*Two **divergence-free** processes P, Q satisfy the same CTL_{-X} formulas if:*

$$P =_{BB} Q$$

*Moreover, when CTL is adjusted to CTL_{-X}^{∞}, we have **iff**. Same goes if P, Q are additionall finitely branching.*

9 Petri Net; Concurrency Semantics

Concurrency Semantics Define the following semantics of interest wrt. concurrency theory:

1. **Interleaving Semantics**: concurrent actions a, b occur in one of the following orders:
   - $a; b$
   - $b; a$
2. **Step Semantics**: concurrent actions a, b occur in one of the following orders:

- $a; b$
- $b; a$
- $a||b$ (in parallel)

3. **Interval Semantics**: concurrent actions a, b occur in continuous time, such that a, b may occur in parallel for a subset of total runtime.

4. **Partial-order (aka. Causal) Semantics**: concurrent actions a, b not only can occur in parallel for some continuous time interval, but can also intersperse as unspecified segments (e.g., OS scheduling).

   Nevertheless, causal relationships between $a, b, \ldots$ are preserved – a calling e.g. `fork()` will posit a partial ordering before the spawned task b, though the exact concurrency behaviors leave much leeway to the OS scheduler.

Definition 9.1 (Pomset) *A **pomset (partial-ordered multiset)** defines a $(E, <, l)$-tuple where:*

- *E a set of "events" – corresponding to each occurrence of action.*
- *$<$ a partial-order of E such that e_i happens before e_j, or incomparable, etc.*
- *$l : E \to A$ a mapping between events to their actions.*

A normal trace thus becomes a totally ordered multiset of actions, compared to a pomset representation.

Petri Net captures the dynamism within parallel systems. It defines a (S, T, F, I)-tuple where:

- S, T define places and actions grouped in bipartite form.
- $F \subseteq (S \times T) \cup (T \times S)$ set of transitions.
- $I : S \to \mathbb{N}$ (initial marking) defines the initial state of control, via allocating tokens to initial states. Subsequent states of control is referred to as **marking** in general.

Definition 9.2 (Control & Tokens) *Petri nets encode control at runtime. A **control** simply refers to the state at which the system is currently at. It is represented as a assignment of **tokens** to each place in net i.e. a **marking**.*

Progress, Justness, Fairness We define several properties on concurrent systems to make claim they display good behaviors.

Definition 9.3 (Safety) *A **safety property** defines that a "bad" predicate ϕ would never hold – e.g., $P \models G(\neg\phi)$.*

Definition 9.4 (Liveness) *A **liveness property** defines that a "good" predicate ϕ would eventually hold – e.g., $P \models F(\phi)$.*

It's easy to see that safety and liveness properties are duo/convertible to each other – $G(\neg\phi) \iff \neg F(\phi)$, after all. Hence we can speak of them in common contexts.

Whether safety/liveness properties hold in a system depends often on whether we make appropriate progress and fairness assumptions (in increasing hierarchy):

Background 9.1 (Completeness Criteria) *LTL is a **linear-time logic** that specifies **linear-time properties** on paths in a Kripke structure. For example, $F(\phi)$ really means that, within each/a path, a state marked with proposition ϕ eventually occurs within the path string.*

*We hence expand the satisfaction idea such that a process P satisfies a LTL property φ under a **completeness criterion** C:*

$$P \models^C \varphi \iff \forall \text{ path } \pi \in P : \pi \models C \implies \pi \models \varphi$$

*For example, C might refer to the assumption that a path is infinite – **deadlock-free**, as a finite path occurs only if we reach a state without outgoing transitions. This is often the **default completeness criterion** when unspecified – in which case we simply use $\models$ without superscript.*

*A completeness criterion D is **stronger** than C if:*

$$\{\pi \mid \pi \in \mathsf{path}(P), \pi \models D\} \subset \{\pi \mid \pi \in \mathsf{path}(P), \pi \models C\}$$

We hence define the following completeness criteria in increasing strength (or reverse implication chain):

Definition 9.5 (Progress)

Definition 9.6 (Justness)

Definition 9.7 (Weak Fairness) *Define a **task** to be a set of transitions in a Kripke structure (you can think of it as a subprocedure in a program). Append the Kripke structure $(S, \twoheadrightarrow, I)$ with a novel structure τ:*

$$\tau \coloneqq \textit{Collection of tasks}$$

*A task T is **enabled** in state $s \in S$ if there exists an outgoing transition from s that is also in T. T is **perpetually enabled** on path π if it is enabled in every state of π (i.e., every state in π contains outgoing transition in T).*

*Orthogonally, if a T-transition exists in π then T **occurs** in π.*

*A path π is hence **weakly fair** if it satisfies the following LTL formula:*

$$\begin{aligned}\mathsf{WF}(T) \coloneqq G(\ & G(\mathsf{enabled}(T)) \implies F(\mathsf{occurs}(\mathsf{T}))\ ) \\ \Leftrightarrow\ & F(G(\mathsf{enabled}(T))) \implies G(F(\mathsf{occurs}(T)))\end{aligned}$$

where predicates enabled, occurs *follow above definition – a state s is marked with* $\mathsf{enabled}(T)$ *iff T is enabled at state s i.e. exists outgoing transition from s that is also in T.*

Hence, define $\models^{WF}$ as:

$$P \models^{WF} \varphi \iff P \models [\wedge_{T \in \tau}(\mathsf{WF}(T))] \implies \varphi$$

Definition 9.8 (Strong Fairness)