Modelling Concurrent Systems Notes

Made by Leon :)

1 Process Algebras

Definition 1.1.1: ACP, CCS, CSP

The syntax of ACP, CCS, and CSP is defined as:

Operation	ACP	CCS	CSP
Termination	ϵ	0	STOP
Deadlock	δ		
Action	a		
Sequential Composition	$P.Q$		
Action Prefixing		$a.P$	$a \to P$
Alternative Choice	$P+Q$	$P+Q$	
External Choice			$P \,\Box\, Q$
Internal Choice			$P \sqcap Q$
Parallel Composition	$P \,\|\, Q$	$P \mid Q$	$P \,\|_A\, Q$
Restriction	$\partial_H(P)$	$P \backslash a$	
Abstraction	$\tau_I(P)$		P/a
Relabelling		$P[f]$	$P[f]$

Differences between ACP, CCS, and CSP

- **Action**: CCS and CSP require Action Prefixing, while ACP can perform sequential composition on any process. This also requires CCS and CSP processes to feature the inaction 0/STOP, while ACP is not restricted to this.
- **Choice**: ACP and CCS have an operator which can perform both External and Internal Choice. CSP Differentiates internal choices from external ones, and internal actions within $\Box$ do not count as a choice in CSP.
- **Parallel Composition**:
  - ACP actions follow a communication function to decide what to synchronise, i.e. $\gamma(a,b)$
  - CCS actions can only synchronise with its conjugate counterpart, i.e. a and $\overline{a}$
  - CSP actions can only synchronise over the same action, i.e. a and a
- **Restriction, Abstraction, Relabelling**:
  - Relabelling just doesn't exist in base ACP, lol
  - CCS combines communication and abstraction into one step - every synchronisation results in a τ.
  - CSP combines Parallel Composition and Restriction into one step, as CSP Parallel Composition doesn't feature left-over Left Merges.

Definition 1.1.2: The GSOS Format

General Structured Operational Semantics (GSOS) operations are compositional.

Rules of GSOS

- Its source has the form $f(x_1, \dots, x_{ar(f)})$ with $f \in \Sigma$ and $x_i \in V$
- The left hand sides of its premises are variables x_i with $1 \le i \le ar(f)$
- The right hand sides of its positive premises are variables that are all distinct, and that do not occur in its source
- Its target only contains variables that also occur in its source or premises

GSOS Semantics of ACP

$$(a.P) \xrightarrow{\alpha} P \qquad P+Q \xrightarrow{\alpha} P \qquad P+Q \xrightarrow{\alpha} Q$$

$$\frac{P \xrightarrow{\alpha} P'}{P \,\|\, Q \xrightarrow{\alpha} P' \,\|\, Q} \qquad \frac{P \xrightarrow{a} P' \quad Q \xrightarrow{b} Q' \quad {\scriptstyle a|b=c}}{P \,\|\, Q \xrightarrow{a} P' \,\|\, Q'}$$

$$\frac{Q \xrightarrow{\alpha} Q'}{P \,\|\, Q \xrightarrow{\alpha} P \,\|\, Q'} \qquad \frac{P \xrightarrow{\alpha} P' \quad (\alpha \notin A)}{\partial_H(P) \xrightarrow{\alpha} \partial_H(P')} \qquad \frac{\langle \mathcal{S}_X \mid \mathcal{S} \rangle \xrightarrow{a} P'}{\langle X \mid \mathcal{S} \rangle \xrightarrow{a} P'}$$

GSOS Semantics of CSP

$$(a \to P) \xrightarrow{a} P \qquad P \sqcap Q \xrightarrow{\tau} P \qquad P \sqcap Q \xrightarrow{\tau} Q$$

$$\frac{P \xrightarrow{a} P'}{P \,\Box\, Q \xrightarrow{a} P'} \qquad \frac{P \xrightarrow{\tau} P'}{P \,\Box\, Q \xrightarrow{\tau} P' \,\Box\, Q} \qquad \frac{Q \xrightarrow{a} Q'}{P \,\Box\, Q \xrightarrow{a} Q'}$$

$$\frac{Q \xrightarrow{\tau} Q'}{P \,\Box\, Q \xrightarrow{\tau} P \,\Box\, Q'} \qquad \frac{P \xrightarrow{\alpha} P'}{f(P) \xrightarrow{f(\alpha)} f(P')} \qquad \frac{P \xrightarrow{\alpha} P' \quad (\alpha \notin A)}{P \|_A Q \xrightarrow{\alpha} P' \|_A Q}$$

$$\frac{P \xrightarrow{a} P' \quad Q \xrightarrow{a} Q' \quad (a \in A)}{P \|_A Q \xrightarrow{a} P' \|_A Q'} \qquad \frac{Q \xrightarrow{\alpha} Q' \quad (\alpha \notin A)}{P \|_A Q \xrightarrow{\alpha} P \|_A Q'}$$

$$\mu p.P \xrightarrow{\tau} P[\mu p.P/p]$$

Definition 1.1.3: CSP Expansion Theorem

Let $P := \sum_{i \in I} \alpha_i P_i$ and $Q := \sum_{j \in J} \beta_j.Q_j$. Then

$$P \mid Q = \sum_{i \in I} \alpha_i (P_i \mid Q) + \sum_{j \in J} \beta_j (P \mid Q_j) + \sum_{\substack{i \in I, j \in J \\ \alpha_i = \overline{b}_j}} \tau.(P_i \mid Q_j)$$

Any guarded CCS expression can be written into a bisimulation equivalent CCS expression of the form $\sum_{i \in I} \alpha_i.P_i$. This is called **head normal form**

Definition 1.1.4: CCS Axioms

- Axioms of CCS

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

- Axiomatisation of Rooted Weak Bisimulation

$$\begin{aligned} \alpha.\tau.P &= \alpha.P && \text{(T1)} \\ \tau.P &= \tau.P + P && \text{(T2)} \\ \alpha.(\tau.P+Q) &= \alpha.(\tau.P+Q) + \alpha.P && \text{(T3)} \end{aligned}$$

- Axiomatisation of Branching Bisimularity

$$\alpha.(\tau.(P+Q)+Q) = \alpha.(P+Q) \qquad \text{(P)}$$

- Axiomatisation of strong partial trace equivalence

$$\alpha.(P+Q) = \alpha.P + \alpha.Q$$

- Axiomatisation of weak partial trace equivalence

$$\tau.P = P$$

Definition 1.1.5: The Recursive Specification

The **recursive definition principle** (RDP) says that a system satisfies its recursive definition. The **recursive specification principle** (RSP) says that two processes satisfying the same **guarded** recursive equation must be equal. It holds for SB and Strong Finite PT.

If S is a recursive specification, i.e. a set of equations $X_i = S_i$ for each $i \in I$ where I is some index set, then $S[P_i/X_i]_{i \in I}$ consists of the equations $P_i = S_i[P_i/X_i]_{i \in I}$ for $i \in I$. The family $(P_i)_{i \in I}$ of processes P_i constitutes a **solution** of S, up to a semantic equivalence $\sim$ iff the equations in $S[P_i/X_i]_{i \in I}$ become true when interpreting $=$ as $\sim$

RDP says that each recursive specification has a solution (up to $\sim$), and RSP says that two solutions (up to $\sim$) of the same guarded recursive specification must be $\sim$-equivalent. Thus RSP can be formulated as the following proof rule

$$\frac{P_i = S_i[P_i/X_i]_{i \in I} \quad Q_i = S_i[Q_i/X_i]_{i \in I} \quad \text{for } i \in I}{P_i = Q_i \quad \text{for } i \in I}$$

2 Semantics and shit like that

Definition 2.1.1: Trace Semantice

- **Completed Trace**: A start to finish trace of a process.
- **Partial Trace**: From the start of a process to any point, including the end. Clearly, $CT(P) \subseteq PT(Q)$
- **Strong vs Weak**: Weak PT and CT means that two processes are equivalent with all instances of τ omitted.
- **Infinite Trace Semantics**: Differs from different types of divergence. Stronger than CT and PT

Definition 2.1.2: Bisimulation Semantice

- **True Bisimulation** ($\leftrightarrows$):
  - if sRt and $s \xrightarrow{a} s'$ then $\exists t'$ s.t. $t \xrightarrow{a} t'$ and $s'Rt'$
  - if sRt and $t \xrightarrow{a} t'$ then $\exists s'$ s.t. $s \xrightarrow{a} s'$ and $s'Rt'$
  - if sRt then $s \models p \iff t \models p$ for all $p \in P$
- **Branching Bisimilarity** ($=_{RBB}$)
  - if sRt and $s \xrightarrow{a} s'$ then either:
    - $a = \tau$ and $s'Rt$
    - $\exists t_1, t'$ such that $t \Rightarrow t_1 \xrightarrow{a} t'$, sRt_1 and $s'Rt'$
  - if sRt and $t \xrightarrow{a} t'$ then either:
    - $a = \tau$ and $t'Rs$
    - $\exists s_1, s'$ such that $s \Rightarrow s_1 \xrightarrow{a} s'$, s_1Rt and $s'Rt'$
  - if sRt and $s \models p$ then $\exists t_1$ s.t. $t \Rightarrow t_1 \models p$, and sRt_1
  - if sRt and $t \models p$ then $\exists s_1$ s.t. $s \Rightarrow s_1 \models p$, and s_1Rt
- Other notions:
  - **Rooted Branching Bisimilarity**: The same as Branching Bisimilarity except the first action is Strongly bisimilar. (This makes RBB a congruence on $+$)
  - **Delay Bisimilarity**: Same as *branching bisimilarity*, but with the requirements sRt_1 and s_1Rt dropped.
  - **Weak Bisimilarity**: The same as *delay bisimularity* except the action requirements are also relaxed:
    - If sRt and $s \xrightarrow{a} s'$ then either:
      - $a = \tau$ and $s'Rt$
      - $\exists t_1, t_2, t'$ such that $t \Rightarrow t_1 \xrightarrow{a} t_2 \Rightarrow t'$ and $s'Rt'$
    - If sRt and $t \xrightarrow{a} t'$ then either:
      - $a = \tau$ and sRt'
      - $\exists s_1, s_2, s'$ such that $s \Rightarrow s_1 \xrightarrow{a} s_2 \Rightarrow s'$ and $s'Rt'$
  - **Simulation**: One process simulates the other when P can do all the same moves as Q. We write $P \sqsubseteq_S Q$ if Q can be simulated by P. Two processes are **simulation equivalent**, $P =_S Q$ if one simulates the other, and vice versa. This is two one-sided equivalences, and therefore is not the same as bisimulation, which needs both processes to be equivalent at the same time

Definition 2.1.3: Compositionality, Congruence

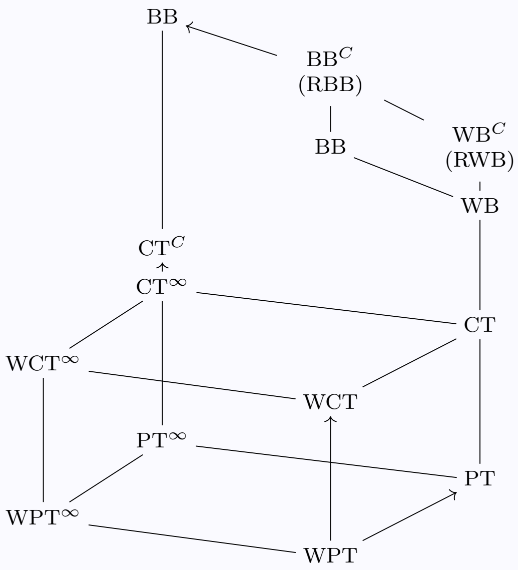
An equivalence $\sim$ is a **congruence**[a] for a language L if $P \sim Q$ implies that $C[P] \sim C[Q]$ for every context $C[\,]$, where $C[\,]$ is an L-expression with a **hole** in it, and $C[P]$ is the result of plugging in P for the hole. An alternative definition for a congruence $\sim$ is if every n-ary operator f of L, we have

$$P_i \sim Q_i \text{ for } i = 1, \dots, n \text{ implies } f(P_1, \dots, P_n) \sim f(Q_1, \dots, Q_n)$$

The **Congruence closure** of a language, denoted $\sim^c$ of a language is a modification to a language that isn't compositional to turn it compositional. The *congruence closure* of Branching Bisimility is Rooted Branching Bisimilarity.

[a] We can also say **the language is compositional for the equivalence**

Theorem 2.1.4: Semantic Equivalence Spectrum



Simulation is finer than PT, coarser than B, incomparable to CT.

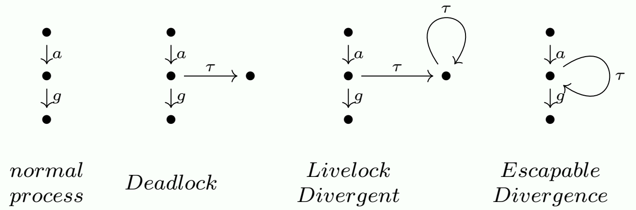
Definition 2.1.5: Safety

A process p satisfies the **canonical safety property**, $P \models \text{safety}(b)$ if no trace of P contains B

A **safety property** of processes in an LTS is given by a set $B \subseteq A^*$. A process p satisfies this safety property, $P \models \text{safety}(B)$ when $\text{WPT}(P) \cap B = \emptyset$.

If $P =_{\text{WPT}} Q$ and $P \models \text{safety}(B)$ for some $B \subseteq A^*$ then $Q \models \text{safety}(B)$.

Definition 2.1.6: Deadlock Types



NP: $a.g.0$, ED: $a.\triangle g.0$, LL: $a.(g.0 + \tau.\triangle 0)$, DL: $\alpha.(g.0 + \tau.0)$

Definition 2.1.7: Distinguishing Interleaving

When using Petri nets we take into account **interleaving semantics**, which doesn't distinguish $a \,||\, b = ab + ba$. This is rejected in causal semantics because in $a \,||\, b$, a and b are causally independent, whereas in $ab + ba$, either a depends on b or vice versa.

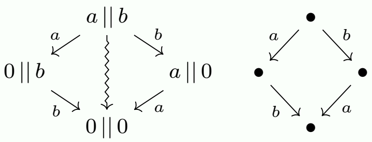
Interval Semantics lie between step semantics and causal semantics. It takes causality into account only to the extent that it manifests itself by durational actions overlapping in time.

Definition 2.1.8: Step Bisimulation

A **step bisimulation** is a binary relation B on the markings of two petri nets, such that

- the initial markings of the nets are related
- If M_1BM_2 and $M_1 \xrightarrow{L} M_1'$ then there is a marking M_2' such that $M_2 \xrightarrow{L} M_2'$ and $M_1'BM_2'$
- If M_1BM_2 and $M_2 \xrightarrow{L} M_2'$ then there is a marking M_1' such that $M_1 \xrightarrow{L} M_1'$ and $M_1'BM_2'$

Two nets are **step bisimilar** if there exists a step bisimulation between them.



Definition 2.1.9: Pomset

Causal trace semantics can be represented in terms of **pomsets**, partially ordered multisets. The process $a(b \,||\, (c + de))$ has two completed pomsets, $b \leftarrow a \rightarrow c$ and $b \leftarrow a \rightarrow d \rightarrow e$.

3 Other models of Concurrency

Definition 3.1.1: Hennessy-Milner Logic

The syntax of HML is given by:

$$\phi, \psi ::= \top \mid \bot \mid \phi \wedge \psi \mid \phi \vee \psi \mid \neg\phi \mid \langle\alpha\rangle\phi \mid [\alpha]\phi$$

Infinitary HML (HML$^\infty$) has an infinitary conjunction: $\bigwedge_{i \in I} \phi_i$ HML in set form. If a process P has a property Φ, we write $P \models \Phi$.

- $P \models \top$
- $P \not\models \bot$
- $P \models \Phi \wedge \Psi$ iff $P \models \Phi$ and $P \models \Psi$
- $P \models \Phi \vee \Psi$ iff $P \models \Phi$ or $P \models \Psi$
- $P \models [K]\Phi$ iff $\forall Q \in \{P' : P \xrightarrow{a} P'$ and $a \in K\}$. $Q \models \Phi$
- $P \models \langle K\rangle\Phi$ iff $\exists Q \in \{P' : P \xrightarrow{a} P'$ and $a \in K\}$. $Q \models \Phi$

Deadlock can be represented as $P \models [\mathrm{Act}]\bot$, where Act is the set of all actions.

Definition 3.1.2: Preorder

A **preorder** is a relation that is *transitive* and *reflexive*, but not *symmetric*. Preorders are denoted with $\sqsubseteq$.
Preorders are used just like equivalence relations to compare specifications and implementations. We write

$$\mathit{Spec} \sqsubseteq \mathit{Impl}$$

For each preorder $\sqsubseteq$, there exists an associated equivalence relation $\equiv$ called its **kernel**, defined by

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

Definition 3.1.3: Petri Nets

Petri Nets are defined with actions as squares. Initial states have a number of markings that can transfer to other states.
Parallel Composition adds a new state, and replaces both actions with the new one.

Definition 3.1.4: Kripke Structure

Kripke Structures are defined on states rather than actions, called **atomic predicates**.
Let AP be a set of **atomic predicates**. A **Kripke structure** over AP is a tuple $(S, \rightarrow, \models)$ with S a set of states, $\rightarrow \subseteq S \times S$, the **transition relation**, and $\models \subseteq S \times AP$. The relation $s \models p$ says that predicate $p \in AP$ **holds in state** $s \in S$.

A **path** in a Kriple structure is a nonempty finite or infinite sequence $s_0, s_1, \ldots$ of states, such as $(s_i, s_{i+1}) \in \rightarrow$ for each adjacent pair of states s_i, s_{i+1} in the sequence. A path is **complete** if it is either infinite or ends in deadlock (a state without outgoing transitions)

Definition 3.1.5: CTL

Computational Tree Logic is defined on

$$\phi, \psi ::= p \mid \top \mid \bot \mid \neg\phi \mid \phi \wedge \psi \mid \phi \vee \psi \mid \neg\phi \mid \phi \rightarrow \psi \mid$$
$$\mathbf{EX}\phi \mid \mathbf{AX}\phi \mid \mathbf{EF}\phi \mid \mathbf{AF}\phi \mid \mathbf{EG}\phi \mid \mathbf{AG}\phi \mid \mathbf{E}\psi\mathbf{U}\phi \mid \mathbf{A}\psi\mathbf{U}\phi$$

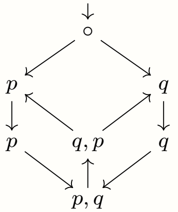
$p \in AP$ is an atomic predicate. CTL is defined on states, the relation $\models$ between states s in a Kripke structure and CTL formulae ϕ is inductively defined by

- $s \models p$, $p \in AP$ iff $(s, p) \in \models$
- $s \models \top$ always holds, and $s \models \bot$ never
- $s \models \neg\phi$ iff $s \not\models \phi$
- $s \models \phi \wedge \psi$ iff $s \models \phi$ and $s \models \psi$
- $s \models \phi \vee \psi$ iff $s \models \phi$ or $s \models \psi$
- $s \models \phi \rightarrow \psi$ iff $s \not\models \phi$ or $s \models \psi$. aka ϕ implies ψ
- $s \models \mathbf{EX}\phi$ iff there is a state s' with $s \rightarrow s'$ and $s' \models \phi$
- $s \models \mathbf{AX}\phi$ iff for each state s' with $s \rightarrow s'$ one has $s' \models \phi$
- $s \models \mathbf{EF}\phi$ iff some complete path starting in s contains a s' with $s \models \phi$
- $s \models \mathbf{AF}\phi$ iff each complete path starting in s contains an s' with $s; \models \phi$
- $s \models \mathbf{EG}\phi$ iff all states s' on some complete path starting in s satisfy $s' \models \phi$
- $s \models \mathbf{AG}\phi$ iff all states s' on all complete path starting in s satisfy $s' \models \phi$
- $s \models \mathbf{E}\psi\mathbf{U}\phi$ iff some complete path π starting in s contains an s' with $s' \models \phi$, and each state s'' on π prior to s' satisfies $s'' \models \phi$
- $s \models \mathbf{A}\psi\mathbf{U}\phi$ iff each complete path π starting in s contains an s' with $s' \models \phi$, and each state s'' on π prior to s' satisfies $s'' \models \phi$

Example 2

Examples of CTL formulae that hold in the initial state:

1. $\mathrm{AF}(p \wedge q)$
2. $\mathrm{AF}\,\mathrm{AG}(p \vee q)$
3. $\mathrm{AF}(\mathrm{EG}p \wedge \mathrm{EG}q)$
4. $\mathrm{AG}(q \rightarrow \mathrm{A}(q\mathrm{U}p))$
5. $\mathrm{A}(\mathrm{E}(\neg q\mathrm{U}p)\mathrm{U}q)$
6. $\mathrm{AG}((p \wedge q) \rightarrow \mathrm{AX}\,\mathrm{E}(q\mathrm{U}p))$



Lemma 3.1.6: Comparing LTL to CTL

LTL and CTL can be shown to be incomparable by proving that there cannot exist an LTL formula that is equivalent to the CTL formula $\mathbf{AGEF}a$, and by showing that there cannot exist a CTL formula equivalent to the LTL formula $\mathbf{FG}a$

Definition 3.1.7: LTL

Linear-Time Temporal Logic is defined on

$$\phi, \psi ::= p \mid \top \mid \bot \mid \neg\phi \mid \phi \wedge \psi \mid \phi \vee \psi \mid \neg\phi \mid \phi \rightarrow \psi \mid$$
$$\mathbf{X}\phi \mid \mathbf{F}\phi \mid \mathbf{G}\phi \mid \psi\mathbf{U}\phi \mid$$

$p \in AP$ is an atomic predicate. The modalities X, F, G, U are called **next-state**, **eventually**, **globally**, and **until**. The relation $\models$ between paths and LTL formulae, with $\pi \models \phi$ saying that the path π satisfies the formula ϕ, or that ϕ is valid on π, or **holds** on π, is inductively defined by

- $\pi \models p$, $p \in AP$ iff $(s, p) \in \models$
- $\pi \models \top$ always holds, and $\pi \models \bot$ never
- $\pi \models \neg\phi$ iff $s \not\models \phi$
- $\pi \models \phi \wedge \psi$ iff $\pi \models \phi$ and $\pi \models \psi$
- $\pi \models \phi \vee \psi$ iff $\pi \models \phi$ or $\pi \models \psi$
- $\pi \models \phi \rightarrow \psi$ iff $s \not\models \phi$ or $\pi \models \psi$
- $\pi \models \mathbf{X}\phi$ iff $\pi' \models \phi$, where π' is the suffix of π obtained by omitting the first state
- $\pi \models \mathbf{F}\phi$ iff $\pi' \models \phi$ for some suffix π'
- $\pi \models \mathbf{G}\phi$ iff $\pi' \models \phi$ for each suffix π'
- $\pi \models \psi\mathbf{U}\phi$ iff $\pi' \models \phi$ for some suffix π' of π, and $\pi'' \models \phi$ for each path $\pi'' \neq \pi'$ with $\pi \Rightarrow \pi'' \Rightarrow \pi'$

Here a path is seen as a state, namely the first state on that path, together with a future that has been chosen already when evaluating an LTL formula on that state. When applying to finite paths, we have to make one adaptation, namely $\pi \models \mathbf{X}\phi$ never holds if π has only one state. So $\mathbf{X}\phi$ says that there is a next state, and that ϕ holds in it.

$s \models \phi$ iff $\pi \models \phi$ for all complete paths π starting in state s. Here a path is **complete** if it is either infinite or ends in deadlock.

Definition 3.1.8: X variants

The variants CTL$_{-X}$ and LTL$_{-X}$ are the same thing but without the X operators.

Theorem 3.1.9: Satisfaction of Strong Bisimilarity

- Two processes are strongly bisimlar iff they satisfy the same infinitary HML formulas. Therefore, to show that two processes are not strongly bisimilar, it suffices to find an infinitary HML formula that is satisfied by one, but not the other.
- Two processes P and Q satisfy the same CTL formulas if they are strongly bisimilar. For finitely branching processes, we have "iff". For general processes, we have "iff" if we use CTL with infinite conjunctions.
- Two divergence-free processes satisfy the same CTL$_{-X}$ formulas if they are branching bisimulation equivalent. We have "iff" if we use CTL$_{-X}$ with infinite conjunctions.

4 Other stuff and Algorithms

Definition 4.1.1: Completeness Criteria

A completeness criterion D is **stronger** than a criterion C if it rules out more paths as incomplete, i.e. if the set of complete paths according to D is a subset of the set of complete paths according to C. If D is stronger than C, then $P \models^c \phi$ implies $P \models^D \phi$ but not necessarily vice versa. For to check $P \models^D \phi$ we have to check that ϕ holds for all complete paths, and under D there are fewer complete paths than under C, so there is less to check.

Theorem 4.1.2: The Completeness Hierarchy

The strongest completeness criteron says that no path is complete. We have $P \models^\infty \phi$ for all processes P and properties ϕ.
The weakest completeness criterion says that all paths are complete, which we call $\emptyset$.

We define a **task** as a set of transitions in a Kripke structure. A task T is **enabled** in a state $s \in S$ if s has an outgoing transition that belongs to the task, **perpetually enabled** on a path π if it is enabled in every state of π, **occurs** in π if π contains a transition that belongs to T, and **relentlessly enabled** on π if each suffix of π contains a state in which it is enabled. This is the case if the task is enabled in infinitely many states of π, in a state that occurs infinitely often in π, or in the last state of a finite π.

A **finite prefix** of a path π is the part of π from its beginning until a given state. A **suffix** of a path π is the path that remains after you remove a finite prefix of it. Note that if π is infinite, then all its suffixes are infinite as well.

1. **Progress**: A path is complete if it is either infinite, or ends in a state in deadlock. Used for CTL/LTL formulae, and for CT semantics.
2. **Justness**: A path is complete
3. **Weak Fairness**: A path is **weakly fair** if, for each suffix π' of π, each task that is perpetually enabled on π', occurs in π'. Equivalently, if each task that from some state onwards is perpetually enabled on π', occurs infinitely often in π.

   Weak fairness can be expressed by the LTL formula
   $$\mathbf{G}(\mathbf{G}(\text{enabled}(T)) \implies \mathbf{F}(\text{occurs}(T)))$$
   for each task T.
4. **Strong Fairness**: A path π is **strongly fair** if for every suffix π' of π, each task that is relentlessly enabled on π' occurs on π'. Equivalently, a path π is **strongly fair** if each task that is relentless enabled on π occurs infinitely often in π.

   Strong fairness can be expressed in LTL as
   $$\mathbf{G}(\mathbf{GF}(\text{enabled}(T)) \implies \mathbf{F}(\text{occurs}(T)))$$
5. **Full Fairness**: Rooted BB..? Doesn't count as a completeness criterion though.

Definition 4.1.3: Partition Refining

Partition refining is an algorithm to turn a process into its minimal state, making it easier to compare bisimularity. Works with

$$\texttt{split}(B, a, P)$$

where B is an equivalence class, a is an action, and P is the process. `split` splits an equivalence class into two, ones with the action and ones without it.

$$[\{s \xrightarrow{a} \bullet\}]_P$$

means "state s does action a outside the equivalence class in process P"

Listing 1: Pseudocode for `split`

```
split(B, a, P) → ({$B_i$} a set of blocks)
  choose $s \in B$
  $B_1 = \emptyset$ ($B_1$ contains states equivalent to $s*$)
  $B_2 = \emptyset$ ($B_2$ contains states inequivalent to $s*$)
  for each $s' \in B$ do
  begin
    if $[\{s \xrightarrow{a} \bullet\}]_P = [\{s' \xrightarrow{a} \bullet\}]_P$ then
      $B_1 = B_1 \cup \{s'\}$
    else
      $B_2 = B_2 \cup \{s'\}$
  end
  if $B_2 = \emptyset$ then
    return $\{B_1\}$
  else
    return $\{B_1, B_2\}$
```

Methodology:

- Start with one equivalence class for all states
- Run `split` on the outermost states, this now splits into R_1 and R_2, where R_2 are outside states
- Run `split` on states with outgoing actions to states in R_2, this now splits R_1 into R_1 and R_3, where R_3 are second-most outer states
- Keep on running until root state is partitioned
- If needed, the minimal graph will have exactly one of each equivalence class

---

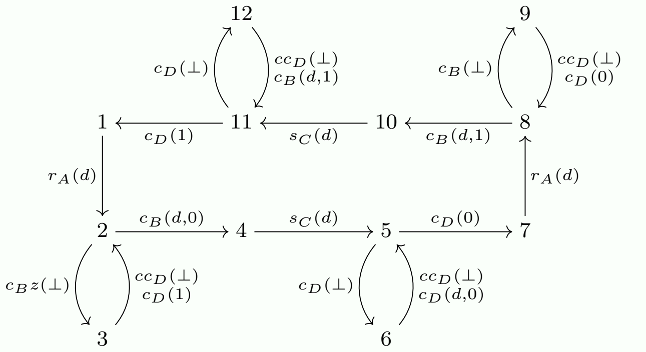
Running partition refinement for Branching Bisimularity

- When checking whether a state in block B has an α transition to block C, it is okay if we can move through block B by doing only τ-transitions, and then reach a state with an α transition to block C
- We never check τ-transitions from block B to block B (τ-transitions to another block are fair game though)

This implies running the rule on

$$[\{s \Rightarrow\xrightarrow{a} \bullet\}]_P$$

Definition 4.1.4: Alternating Bit Protocol



Specification for the Sender

$$S_b = \sum_{d \in \Delta} r_A(d) \cdot T_{db}$$

$$T_{db} = (s_B(d,b) + s_B(\perp)) \cdot U_{db}$$
$$U_{db} = r_D(b) \cdot S_{1-b} + (r_D(1-b) + r_D(\perp)) \cdot T_{db}$$

In state S_b, the Sender reads a datum d from channel A. Then it proceeds to state T_{db}, in which it sends datum d into channel B, with the bit b attached to it. However, the pair (d,b) may be distorted by the channel, so that it becomes the error message $\perp$. Next, the system proceeds to state U_{db}, in which it expects to receive the acknowledgement b through channel D, ensuring that the pair (d,b) has reached the Receiver unscathed. If the correct acknowledgement b is received, then the system proceeds to state S_{1-b}, in which it is going to send out a datum with the bit $1-b$ attached to it. If the acknowledgement is either the wrong bit, $1-b$ or the error message $\perp$, then the system proceeds to state T_{db}, to send the pair (d,b) into channel B once more.

Specification for the Reciever

$$R_b = \sum_{d' \in \Delta} \{r_B(d',b) \cdot s_C(d') \cdot Q_b + r_B(d', 1-b) \cdot Q_{1-b}\}$$
$$+ r_B(\perp) \cdot Q_{1-b}$$
$$Q_b = (s_D(b) + s_D(\perp) \cdot R_{1-b})$$

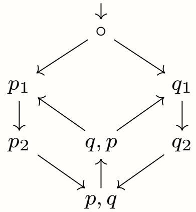
1. If in Rb the Receiver reads a pair (d',b) from channel B, then this constitutes new information, so the datum d' is sent into channel C. Then the Receiver proceeds to state Q_b, in which it sends acknowledgement b to the Sender via channel D. However, this acknowledgement may be distorted by the channel, so that it becomes the error message $\perp$. Next, the Receiver proceeds to state R_{1-b}, in which it is expecting to receive a datum with the bit $1-b$ attached to it.
2. If in R_b the Receiver reads a pair $(d', 1-b)$ or an error message $\perp$ from channel B, then this does not constitute new information. So then the Receiver proceeds to state Q_{1-b} straight away, to send acknowledgement $1-b$ to the Sender via channel D. However, this acknowledgement may be distorted by the channel, so that it becomes the error message $\perp$. Next, the Receiver proceeds to state R_b again.

5 Example Catalogue

Example 1: LTL

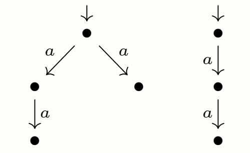
Examples of LTL that hold in the initial state:

1. $\mathbf{G}(p \vee q)$ false!
2. $\mathbf{FG}(p \vee q)$
3. $\mathbf{F}(\mathbf{G}p \vee \mathbf{G}q)$ false!
4. $\mathbf{G}(q \rightarrow q\mathbf{U}p)$
5. $\mathbf{G}((p \wedge \neg q) \rightarrow \mathbf{X}q)$ false!
6. $\mathbf{G}((p \wedge q) \rightarrow \mathbf{X}(q\mathbf{U}p))$

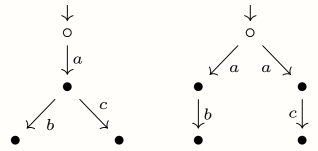


Example 5.0.1: Trace and Bisimulation Semantics

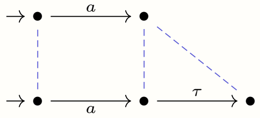
PT but not CT



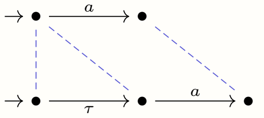
CT but not Bisimulation



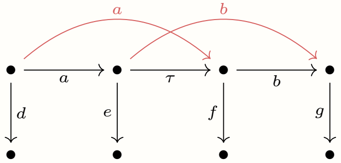
BB and RBB



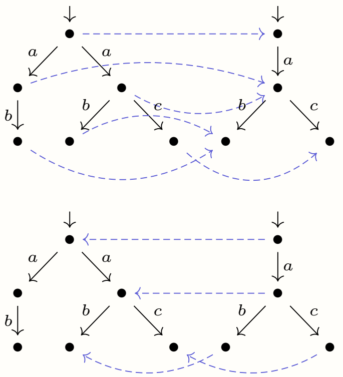
BB but not RBB



Weak Bisimulation and Branching Bisimulation Counterexample

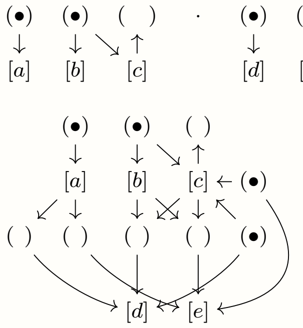


Simulation equivalent but not Bisimulation equivalent

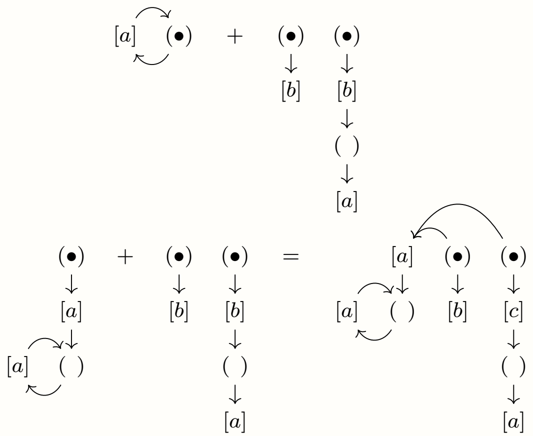


Example 5.0.2: ACP Represented in Petri Nets

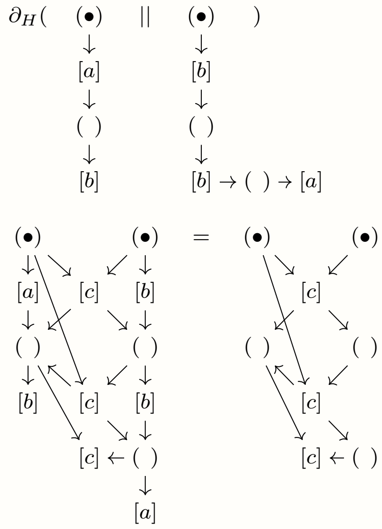
Sequential Composition



Alternative Composition



Parallel Composition



Example 5.0.3: HML

Consider the HML formula

$$\phi = [a]([b]\langle c\rangle\top \wedge \langle b\rangle[a]\bot)$$

- An example process P that satisfies ϕ and an example process Q that does not is shown

$$P \qquad \circ \xrightarrow{a} \bullet \xrightarrow{b} \bullet \xrightarrow{c} \bullet \qquad\qquad Q \qquad \circ \xrightarrow{a} \bullet$$

- Another HML formula without the negation symbol, that holds for all processes for which ϕ does not hold:

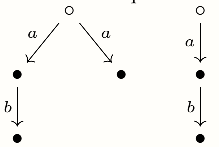
$$\phi' = [a](\langle b\rangle\langle c\rangle\bot \vee [b]\langle a\rangle\top)$$

i.e. "For all a traces, either there exists no b action followed by a c action, or all b actions aren't followed by an a action"

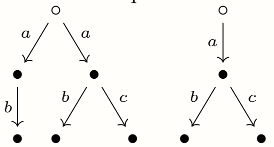
Example 5.0.4: More HML

The following counterexamples work in the LHS but not the RHS.

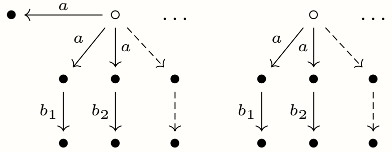
Counterexample 2



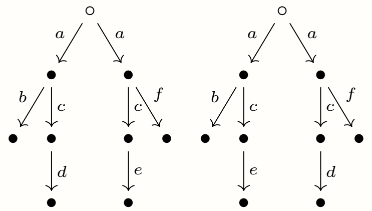
Counterexample 3



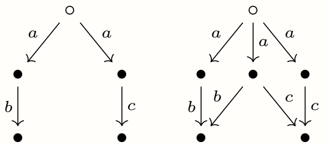
Counterexample 2: $\langle a\rangle[b]\bot$
Counterexample 3: $\langle a\rangle[c]\bot$



Counterexample 4: $\langle a\rangle[\mathrm{Act}]\bot$



Counterexample 5: $\langle a\rangle(\langle b\rangle\top \wedge \langle c\rangle\langle d\rangle\top)$



Counterexample 6: $[a]([b]\bot \vee [c]\bot)$

Example 5.0.5: Specification of 2-bit buffer

The 2-bit buffer is defined as so:

$$Q_\epsilon = \sum_{d \in D} r(d).Q_d$$

$$Q_e = \sum_{d \in D} r(d).Q_{de} + \overline{s}(d).Q_\epsilon$$

$$Q_{de} = \overline{s}(e).Q_d$$

The 1-bit buffer is defined as so:

$$B_\epsilon = \sum_{d \in D} r(d).B_d$$

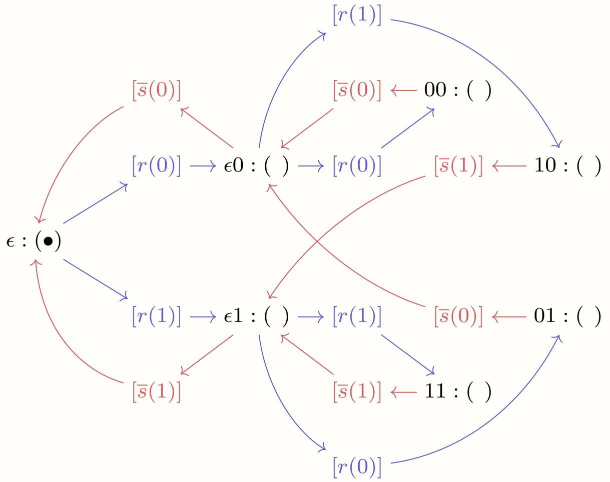
$$B_d = \overline{s}(d).B_\epsilon$$

An implementation of Q_ϵ in term of a one bit bufffer is

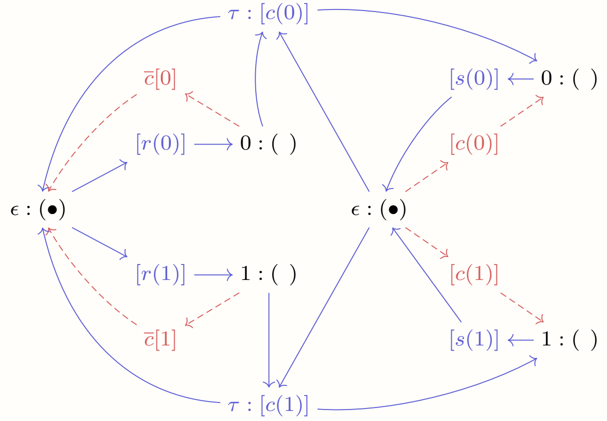
$$Q_\epsilon := (B_\epsilon[c/s] \mid B_\epsilon[c/r]) \backslash c$$

where $[c/s]$ is all actions of s replaced with c.

Petri Net of 2-bit Protocol



Petri Net of 1-bit Protocol



Example 5.0.6: Congruence

The **congruence closure** $\sim_L$ is defined by

$P \sim_L Q$ if and only if $C[P] \sim C[Q]$ for any L-context $C[\ ]$

Prove that $\sim_L$ is the coarsest congruence finer than $\sim$.

We need to show three things:

1. $\sim_L$ is a congruence
2. $\sim_L$ is finer than (or equal to) $\sim$
3. Any other congruence that is finer than $\sim$ is also finer than (or equal to) $\sim_L$.

Proof.

1. We need to show that if $P \sim_L Q$ and is a context, then $D[P] \sim_L D[Q]$. So suppose $P \sim_L Q$. This means that for any L-context $C[\ ]$ we have $C[P] \sim C[Q]$. What we have to show is that for any L-context $C[\ ]$ we have $C[D[P]] \sim C[D[Q]]$, because that's the definition of $D[P] \sim_L Q[D]$. But this holds, because a context is just a context.

2. Suppose $P \sim_L Q$. This means that for any L-context $C[\ ]$ we have $C[P] \sim C[Q]$. This holds in particular for the trivial context, consisting of one single hole. If $T[\ ]$ is the trivial context, then $T[P] = P$. Thus $P \sim_L Q$ implies $P \sim Q$, which was to be shown.

3. Let be any other congruence that is finer then $\sim$. Suppose $P \approx Q$ Then, for any context $C[\ ]$ we have $C[P] \approx C[Q]$, using that $\approx$ is a congruence, and thus $C[P] \sim C[Q]$, using that $\approx$ is finer than $\sim$. By definition this means that $P \sim_L Q$, which had to be shown.

□

Example 5.0.7: GSOS

GSOS for sequencing

$$\frac{P \xrightarrow{a} P' \quad (P' \neq 0)}{P;Q \xrightarrow{a} P';Q} \qquad \frac{P \xrightarrow{a} 0}{P;Q \xrightarrow{a} Q}$$

Is this in GSOS format?

Proof. No, it is not in GSOS format. The second rule fails this format because the right-hand side of its premise features a constant. (The first rule also fails this format, because it has a side-condition that cannot be expanded away.)
To show that strong bisimulation is not a congruence, consider the processes $P = a.0$, $Q = a.(0+0)$ and $R = b.0$. Now P and Q are strongly bisimilar, but $P;R$ and $Q;R$ are not. In fact $P;R$ has a trace ab, whereas $Q;R$ can never do a b. Here it is important that to match the second rule for ;, being equal to 0 is taken literally; not up to bisimilarity or so. □