colour1Red colour2Green colour3Cerulean
a4paper, margin=0.17in
3

Modelling Concurrent Systems Notes

Made by Leon :)

1 Process Algebras

[ACP, CCS, CSP]dfn: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		$P[f]$	$P[f]$

0.08ex

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.

[The GSOS Format]dfn:gsos General Structured Operational Semantics (GSOS) operations are compositional.

Rules of GSOS

Its source has the form $f(x_1, \ldots, x_{\alpha(f)})$ with $f \in \Sigma$ and $x_i \in V$

Axiomatisation of Branching Bisimularity

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

Axiomatisation of strong partial trace equivalence

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

Axiomatisation of weak partial trace equivalence

$$\tau.P = P$$

[Axioms of ACP]dfn:acp-axioms is this really necessary.. who knows

2 Semantics and shit like that

[Trace Semantice]dfn:trace-semantics

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

[Bisimulation Semantice]dfn:bisimulation

True Bisimulation ():

if sRt and sas' then $\exists t'$ s.t. tat' and $s'Rt'$

if sRt and tat' then $\exists s'$ s.t. sas' and $s'Rt'$

if sRt then $s \models p \iff t \models p$ for all $p \in P$

Branching Bisimilarity ($=_{RBB}$)

if sRt and sas' then either:

$a = \tau$ and $s'Rt$

$\exists t_1$, t' such that $t \Rightarrow t_1 a t'$, sRt_1 and $s'Rt'$

if sRt and tat' then either:

$a = \tau$ and $t'Rs$

$\exists s_1$, s' such that $s \Rightarrow s_1 a s'$, $s_1 Rt$ 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 $s_1 Rt$

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 $s_1 Rt$ dropped.

Weak Bisimilarity: The same as *delay bisimularity* except the action requirements are also relaxed:

If sRt and sas' then either:

$a = \tau$ and $s'Rt$

$\exists t_1, t_2, t'$ such that $t \Rightarrow t_1 a t_2 \Rightarrow t'$ and $s'Rt'$

If sRt and tat' then either:

$a = \tau$ and sRt'

$\exists s_1, s_2, s'$ such that $s \Rightarrow s_1 a s_2 \Rightarrow s'$ and $s'Rt'$

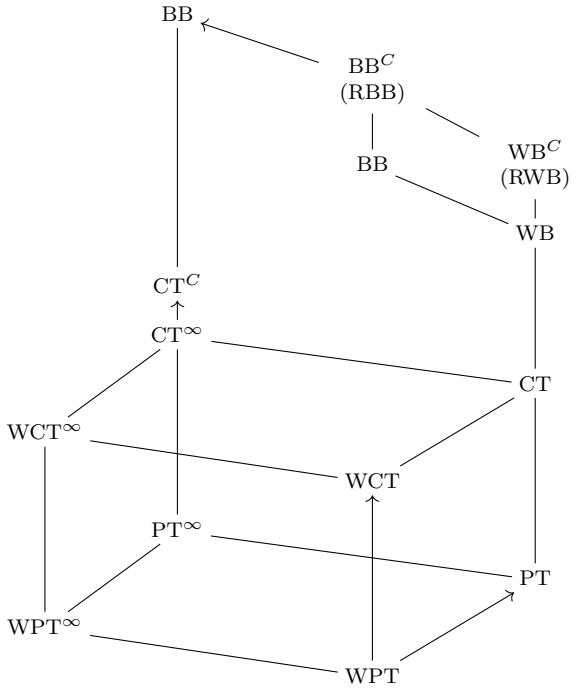
[Compositionality, Congruence]dfn:congruence An equivalence $\sim$ is a **congruence**[1] for a language L if $P \sim Q$ implies that $C[P] \sim C[Q]$ for every context $C[\ ]$, where $C[\ ]$ is an L-expression with a **hole** in it, and $C[P]$ is the result of plugging in P for the hole. An alternative definition for a congruence $\sim$ is if every n-ary operator f of L, we have

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

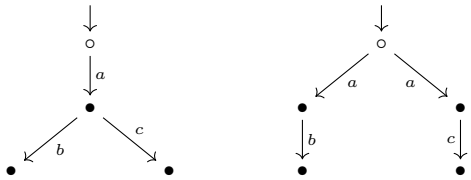
0.08ex The **Congruence closure** of a language, denoted $\sim^c$ of a language is a modification to a language that isn't compositional to turn it compositional. The *congruence closure* of Branching Bisimilarity is Rooted Branching Bisimilarity.

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

[Semantic Equivalence Spectrum]thm:spectrum



[The GOAT of Process algebra]dfn:bisim-counterexample



[Failure Semantics]dfn:failure

[Consistent Colouring]dfn:colouring
[Safety]dfn:

3 Other models of Concurrency

[Hennessy-Milner Logic]dfn:hml 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 $P \models \Phi$.

$$P \models \top$$

$$P \not\models \bot$$

$$P \models \Phi \wedge \Psi \text{ iff } P \models \Phi \text{ and } P \models \Psi$$

$$P \models \Phi \vee \Psi \text{ iff } P \models \Phi \text{ or } P \models \Psi$$

$$P \models [K]\Phi \text{ iff } \forall Q \in \{P' : PaP' \text{ and } a \in K\}.\ Q \models \Phi$$

$$P \models \langle K\rangle\Phi \text{ iff } \exists Q \in \{P' : PaP' \text{ and } a \in K\}.\ Q \models \Phi$$

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

[Preorder]dfn: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

$$Spec \sqsubseteq Impl$$

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

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

[Simulation Equivalence]dfn: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

[Kripke Structure]dfn:kripke 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** $s \in S$.

0.08ex 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)

[Petri Nets]dfn:petris they exist

[CTL]dfn:ctl Computational Tree Logic is defined on

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

$p \in AP$ 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 $s \models \psi$

$s \models \phi \to \psi$ iff $s \not\models \phi$ or $s \models \psi$

$s \models EX\phi$ iff there is a state s' with $s \to s'$ and $s' \models \psi$

$s \models AX\phi$ iff for each state s' with $s \to s'$ one has $s' \models \psi$

$s \models EF\phi$ iff some complete path starting in s contains a state s' with $s \models \phi$

$s \models AF\phi$ iff each complete path starting in s contains a state s' with $s \models \phi$

$s \models EG\phi$ iff all states s' on some complete path starting in s satisfy $s' \models \phi$

$s \models AG\phi$ iff all states s' on all complete path starting in s satisfy $s' \models \phi$

$s \models E\psi U\phi$ iff some complete path π starting in s contains a state s' with $s' \models \phi$, and each state s'' on π prior to s' satisfies $s'' \models \phi$

$s \models A\psi U\phi$ iff each complete path π starting in s contains a state s' with $s' \models \phi$, and each state s'' on π prior to s' satisfies $s'' \models \phi$

[LTL]dfn:ltl Linear-Time Temporal Logic is defined on

$$\phi, \psi ::= p \mid \top \mid \bot \mid \neg\phi \mid \phi \wedge \psi \mid \phi \vee \psi \mid \neg\phi \mid \phi \to \psi \mid$$
$$X\phi \mid F\phi \mid G\phi \mid \psi U\phi \mid$$

$p \in AP$ 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 $\pi \models \psi$

$\pi \models X\phi$ iff $\pi' \models \phi$, where π' is the suffix of π obtained by omitting the first state

$\pi \models F\phi$ iff $\pi' \models \phi$ for some suffix π'

$\pi \models G\phi$ iff $\pi' \models \phi$ for each suffix π'

$\pi \models \psi U \phi$ 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 staet on that path, together with a future that has been chosen already when evaluating an LTL formula on that state. Traditionally, these judgements $\pi \models \phi$ were defined only for infinite paths π. When also applying them to finite paths, we only 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.

[Comparing LTL to CTL]lma: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 $\mathbf{FG}a$

[Satisfaction of Strong Bisimilarity]thm:hml-sb

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.

[De Nicola-Vaandrager]thm: Translates Krikpe structures :D

[Partition Refining]dfn: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.

$$[\{sa\bullet\}]_P$$

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

[escapeinside=(**), caption=Pseudocode for `split`] split(B, a, P) (*$\rightarrow$*) ((*$\{B_i\}$*) 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 (*$[\{sa\bullet\}]_P = [\{s'a\bullet\}]_P$*) 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

0.08ex 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

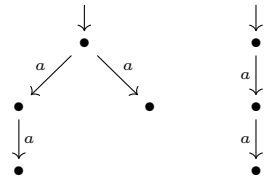
- We never check τ-transitions from block B to block B (τ-transitions to another block are fair game though)

This implies running the rule on

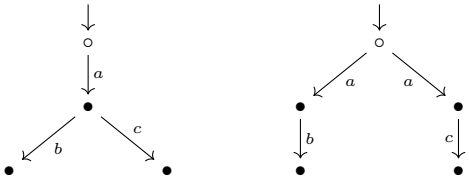
$$[\{s \Rightarrow a\bullet\}]_P$$

4 Example Catalogue

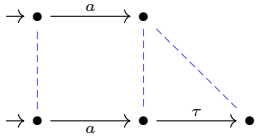
[Trace Semantics]xmp:trace-semantics A process that is PT equivalent but not CT equivalent



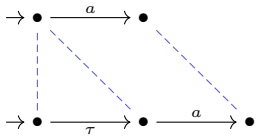
A process that is CT equivalent but not Bisimulation equivalent



[Bisimulation Semantics]xmp:bisim-semantics
Two processes that are Branching Bisimulation equivalent

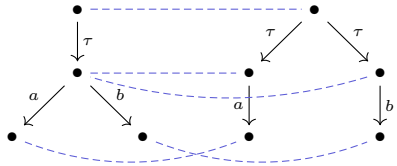


Two processes that are Branching Bisimulation equivalent but not Rooted BB equivalent

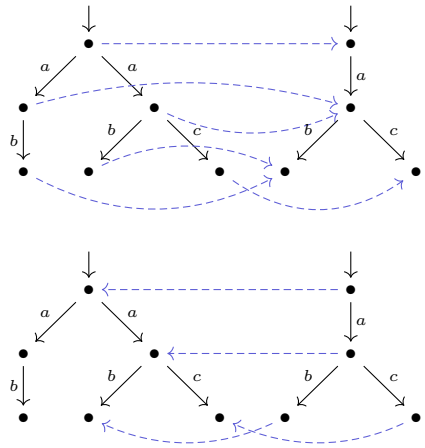


Two processes that are Weak Bisimulation equivalent but not Branching Bisimulation

equivalent?



Two processes that are Simulation equivalent but not Bisimulation equivalent



[1-12]