Compositionality, Interaction, and Abstraction in Concurrent Computing Systems: Process Calculi and Behavioral Equivalences

Marco Bernardo

University of Urbino – Italy
Department of Pure and Applied Sciences
Section of Informatics and Mathematics

Syllabus:

1. Concurrency and Communication

- Sequential computing (since 1930's):
  - A single step at a time is executed.
  - Imperative models: Turing machines and Von Neumann architecture.
  - Programming languages: Fortran, Cobol, Algol, Basic, Pascal, C, ...
  - Declarative models: Church λ-calculus and first-order logic.
  - Programming languages: Lisp, Scheme, ML, Haskell, Prolog, ...
- Concurrent and distributed computing (since 1970's):
  - Several steps can be simultaneously executed.
  - Shared-memory model vs. message-passing model.
  - Primitives for software synchronization: semaphores, monitors, ...
  - Concurrent programming languages: Ada, Occam, Erlang, Scala, ...
  - Extensions of previously developed languages.

- Global computing (since 2000's): computation over infrastructures globally accessible via personal devices and offering uniform services.
- Abstraction of a global computer that we can use *anytime anywhere*.
- Development of large-scale general-purpose computing systems that hopefully have a dependably predictable behavior for the needs of a distributed and mobile world.
- Providing support for e-government and e-commerce (web services), resource sharing (cloud), ambient intelligence (IoT), ...
- Addressing issues that go beyond concurrent and distributed systems: mobility, ubiquity, dynamicity, interactivity, ...
- *"Computing is interaction!"* (Robin Milner, 1994).

- Concurrency and communication are essential in the design and deployment of modern computing systems.
- Any such system is composed of many interconnected parts that interact by exchanging information or simply synchronizing.
- Concurrency and communication are complementary notions:
  - Diversity: each part acts concurrently with (independently of) other parts.
  - Unity: achieved through communication among the various parts.
- Computing systems featuring concurrency and communication are often required to possess a degree of reactivity to external stimuli and are usually nonterminating.
- Nondeterminism in the final result or in the computation can arise due to the different speeds of the parts, the interaction scheme among the parts, and the scheduling policies that are adopted.

- The behavior of a sequential system can be formalized as a mathematical function that associates a final state (output) with every possible initial state (input).
- This input-output transformation approach for sequential systems is no longer applicable to communicating concurrent systems.
- Cannot abstract from the *intermediate states* of the computation.
- Consider the following two sequential program fragments:

```
(1) X := 1;    (2) X := 0; X := X + 1;
```

- In the absence of interference, they have the same effect (`X` becomes `1`).
- If the two fragments are executed concurrently, the final value of `X` is not necessarily `1`, but can be either `1` or `2` (cannot be deterministically predicted).

- How to model and analyze communicating concurrent systems?
- Making no distinction of kind between systems and their parts enables uniform reasoning at different abstraction levels.
- Accomplished through the notion of process.
- A process may be decomposed into subprocesses for a certain purpose or may be viewed as being atomic for other purposes.
- A process is a series of actions/events divided into:
  - Internal actions, possibly due to subprocesses communication.
  - Interactions with neighboring processes or the external environment.

- Any computing system features a structure and a behavior.
- *A process is an abstraction of the behavior of a computing system.*
- The behavior of a complex computing system can be defined as its entire capability of communication.
- Black-box view: the behavior of a system is exactly what is observable and to observe a system is exactly to communicate with it.
- The notion of process focuses on the behavioral aspects of a system, while neglecting its structural and physical attributes.

- Consider a sequential system that:
  - either performs action a followed by action b and then terminates;
  - or performs action b followed by action a and then terminates.
- Consider another system that performs actions a and b in parallel:
  - either action a terminates first and action b terminates afterwards;
  - or action b terminates first and action a terminates afterwards.
- Structurally different, but behaviorally equivalent!
- A concurrent system behaves like a sequential one obtained by interleaving the actions executed by the parts of the former.

- Is there a calculus for processes as basic as λ-calculus for functions?
- Starting between late 1970's and early 1980's:
  - Robin Milner, Tony Hoare, Matthew Hennessy, Rocco De Nicola, Jan Bergstra, Willem Klop, Jos Baeten, Rob van Glabbeek, ...
  - CCS, CSP, ACP, π-calculus, LOTOS, ...
  - CWB, FDR, μCRL, CADP, ...
- Process: series of actions or events.
- Calculus: system or method of calculation.
- Algebra: calculus of symbols combining according to certain laws.
- Generalization of formal languages and automata theory focusing on *system behavior* rather than language generation and recognition.
- Foundations of concurrent programming semantics.
- Model-based design of modern computing systems.

- Conceived for studying communicating concurrent systems and their various aspects: nondeterminism, mobility, probability, time.
- Linguistic counterpart of computational models such as Keller transition systems, Petri nets, Winskel event structures.
- Compositional modeling by means of several behavioral operators expressing *sequential*, *alternative*, *parallel* compositions of processes.
- Process comparison through behavioral equivalences and preorders formalizing the notions of same behavior and behavior refinement.
- System dynamics described through behavioral equations.
- Abstraction from certain unnecessary details of system behavior by distinguishing between *visible* and *invisible* actions.

- **Running example**: producer-consumer system.
- General description:
  - Three components: producer, finite-capacity buffer, consumer.
  - The producer deposits items into the buffer as long as the buffer capacity is not exceeded.
  - Stored items are then withdrawn from the buffer by the consumer according to some predefined discipline (like FIFO or LIFO).
- Specific scenario:
  - The buffer has only two positions.
  - Items are identical, hence the discipline is not important.

2. Syntax of Process Calculi

- Modeling languages for communicating concurrent systems.
- Compositionality: building complex models from simpler ones.
- Abstraction: ability to neglect some details of a model.
- Their basic ingredients are *actions* and *behavioral operators*.
- Act_{v}: countable set of visible action names.
- τ: invisible (or silent or unobservable) action.
- $Act = Act_{\mathrm{v}} \cup \{\tau\}$: set of all action names.
- $Relab = \{\varphi : Act \to Act \mid \varphi^{-1}(\tau) = \{\tau\}\}$:
  set of visibility-preserving action relabeling functions.

- Syntax of process terms:

$$
\begin{array}{rclll}
P & ::= & \underline{0} & \text{terminated process} & \\
& | & a\,.\,P & \text{action prefix} & (a \in \mathit{Act}) \\
& | & P + P & \text{alternative composition} & \\
& | & P \parallel_S P & \text{parallel composition} & (S \subseteq \mathit{Act}_{\mathrm{v}}) \\
& | & P\,/\,H & \text{hiding} & (H \subseteq \mathit{Act}_{\mathrm{v}}) \\
& | & P \setminus L & \text{restriction} & (L \subseteq \mathit{Act}_{\mathrm{v}}) \\
& | & P\,[\varphi] & \text{relabeling} & (\varphi \in \mathit{Relab}) \\
& | & X & \text{process variable} & (X \in \mathit{Var}) \\
& | & \mathrm{rec}\, X : P & \text{recursion} & (X \in \mathit{Var})
\end{array}
$$

- Recursion can equivalently be expressed by means of:
  - Binder rec for a process variable X in Var that may occur in P.
  - *Behavioral equation* $B \triangleq P$ for a process constant B in Const.

- Operator precedence: unary operators $> + > \|$.
- Operator associativity: $+$ and $\|$ are left associative.
- $\underline{0}$ is a terminated process and hence cannot execute any action.
- $a \,.\, P$ can perform a and then behaves as P (action-based sequential composition).
- $P_1 + P_2$ behaves as P_1 or P_2 depending on the actions they enable.
- The choice among several actions initially enabled by P_1 and P_2 is solved nondeterministically.
- The choice is *internal* if all the initially enabled actions are identical, otherwise it can be influenced by the *external environment*.
- Action prefix and alternative composition are *dynamic operators* because they are discarded from the target process along with the executed action or the unselected subprocess.

- $P_1 \parallel_S P_2$ behaves as P_1 running in parallel with P_2.
- Actions enabled by P_1 or P_2 whose name does not belong to S are executed autonomously by P_1 and by P_2.
- Synchronization is forced between any action enabled by P_1 and any action enabled by P_2 that have the same name belonging to S ($S = \emptyset$ means P_1 and P_2 fully independent, $S = Act_{\mathrm{v}}$ means P_1 and P_2 fully synchronized).
- $P \,/\, H$ behaves as P but every action belonging to H is turned into τ (abstraction mechanism; can be used for preventing a process from communicating).
- $P \setminus L$ behaves as P but every action belonging to L is forbidden (same effect as $P \parallel_L \underline{0}$).
- $P\,[\varphi]$ behaves as P but every action is renamed according to φ (redundance avoidance; encoding of the previous two operators if φ is non-visibility-preserving or partial).
- They are *static operators*, parallel composition is also *structural*.

- $\mathrm{rec}\, X : P$ behaves as P with every free occurrence of variable X being replaced by $\mathrm{rec}\, X : P$.
- Same as $B \triangleq P$ with B a process constant that may occur in P.
- A process variable is said to occur *free* in a process term if it is not in the scope of a rec binder for that variable, otherwise it is said to be *bound* in that process term.
- A process term is said to be *closed* if all of its occurrences of process variables are bound, otherwise it is said to be *open* (closed if all of its occurrences of process constants are defined).
- A process term is said to be *guarded* iff all of its occurrences of process variables/constants are in the scope of action prefix operators.
- $\mathbb{P}$: set of closed and guarded process terms, which are fully defined and enable finitely many actions.

- **Running example** (process syntax):
  - Action names: verbs composed of lower-case letters.
  - Process constant names: nouns starting with an upper-case letter.
  - The only observable activities are deposits and withdrawals.
  - Visible actions: $deposit$ and $withdraw$.
  - Structure-independent process algebraic description where the system state is the number of items in the buffer:

$$\begin{array}{lcl} ProdCons_{0/2} & \triangleq & deposit \,.\, ProdCons_{1/2} \\ ProdCons_{1/2} & \triangleq & deposit \,.\, ProdCons_{2/2} + withdraw \,.\, ProdCons_{0/2} \\ ProdCons_{2/2} & \triangleq & withdraw \,.\, ProdCons_{1/2} \end{array}$$

  - Specification to which every correct implementation should conform.

3. Interleaving Semantics

- Mathematical model in the form of a state transition graph that represents all computations and branching points.
- States are *global* as contain the description of the local states of the subprocesses composed in parallel.
- Computations are obtained by *interleaving* the actions executed by the subprocesses composed in parallel (all possible sequencings).
- Keller labeled transition systems (1976) instead of Kripke structures to elicit transition-labeling actions instead of state properties.
- Process term $P \in \mathbb{P}$ is mapped to the LTS $[\![P]\!] = (\mathbb{P}, Act, \longrightarrow, P)$:
  - Each state corresponds to a process term into which P can evolve.
  - The initial state corresponds to P.
  - Each transition from a source state to a target state is labeled with the action that determines the corresponding state change.

- The transition relation $\longrightarrow \subseteq \mathbb{P} \times Act \times \mathbb{P}$ is the smallest subset of $\mathbb{P} \times Act \times \mathbb{P}$ that satisfies *Plotkin-style operational semantic rules* defined by induction on the syntactical structure of process terms.
- Derivation of one single transition at a time by applying the rules to the source state of the transition under construction.
- No rule for $\underline{0}$: $[\![\underline{0}]\!]$ has a single state and no transitions.
- Basic rule for action prefix: $a \,.\, P \xrightarrow{a} P$.
- Inductive rules for all the other operators.
- Different formats for *dynamic* $(+)$ and *static* $(\| \;/\; \backslash\; [\,])$ operators.
- $[\![P]\!]$ is *finite state* if inside P there are no recursive definitions that contain static operators.

- Operational semantic rules for alternative composition:

$$\frac{P_1 \xrightarrow{a} P_1'}{P_1 + P_2 \xrightarrow{a} P_1'} \qquad \frac{P_2 \xrightarrow{a} P_2'}{P_1 + P_2 \xrightarrow{a} P_2'}$$

- Reading order: left-bottom, left-top, right-top, right-bottom.
- If several actions are initially enabled, the choice among them is solved nondeterministically due to the absence of precise criteria or quantitative information (if-then-else, priority, probability, time race).
- The choice is internal if all the initially enabled actions are identical.
- Otherwise the choice can be influenced by the external environment.
- Note that $+$ no longer occurs in the target processes P_1' and P_2'.

- Operational semantic rule for synchronization:

$$\frac{P_1 \xrightarrow{a} P_1' \qquad P_2 \xrightarrow{a} P_2' \qquad a \in S}{P_1 \parallel_S P_2 \xrightarrow{a} P_1' \parallel_S P_2'}$$

- Operational semantic rules for parallel execution:

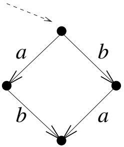
$$\frac{P_1 \xrightarrow{a} P_1' \qquad a \notin S}{P_1 \parallel_S P_2 \xrightarrow{a} P_1' \parallel_S P_2} \qquad \frac{P_2 \xrightarrow{a} P_2' \qquad a \notin S}{P_1 \parallel_S P_2 \xrightarrow{a} P_1 \parallel_S P_2'}$$

- Note that $\parallel_S$ still occurs in the target processes.
- The last two rules result in the interleaving semantics.

- The following process terms represent structurally different systems:

$$a \,.\, b \,.\, \underline{0} + b \,.\, a \,.\, \underline{0}$$
$$a \,.\, \underline{0} \parallel_{\emptyset} b \,.\, \underline{0}$$

- But they are indistinguishable by an external observer.
- The interleaving semantics yields the same labeled transition system:



- Up to processes associated with states, which are not observable:
  - Sequential: left $b \,.\, \underline{0}$, right $a \,.\, \underline{0}$, bottom $\underline{0}$.
  - Concurrent: left $\underline{0} \parallel_{\emptyset} b \,.\, \underline{0}$, right $a \,.\, \underline{0} \parallel_{\emptyset} \underline{0}$, bottom $\underline{0} \parallel_{\emptyset} \underline{0}$.

- Operational semantic rules for hiding, restriction, relabeling:

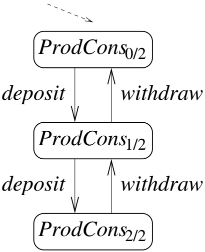
$$\frac{P \xrightarrow{a} P' \qquad a \in H}{P \,/\, H \xrightarrow{\tau} P' \,/\, H} \qquad \frac{P \xrightarrow{a} P' \qquad a \notin H}{P \,/\, H \xrightarrow{a} P' \,/\, H}$$

$$\frac{P \xrightarrow{a} P' \qquad a \notin L}{P \setminus L \xrightarrow{a} P' \setminus L} \qquad \frac{P \xrightarrow{a} P'}{P\,[\varphi] \xrightarrow{\varphi(a)} P'\,[\varphi]}$$

- Operational semantic rules for recursion (variables vs. constants):

$$\frac{P\{\mathrm{rec}\, X : P \hookrightarrow X\} \xrightarrow{a} P'}{\mathrm{rec}\, X : P \xrightarrow{a} P'} \qquad \frac{B \triangleq P \qquad P \xrightarrow{a} P'}{B \xrightarrow{a} P'}$$

- **Running example** (process semantics):
  - Labeled transition system $[\![ProdCons_{0/2}]\!]$:



  - Obtained by mechanically applying the operational semantic rules for process constant, alternative composition, and action prefix.

4. Truly Concurrent Semantics via Petri Nets

- Petri nets (1962) are truly concurrent models in the form of bipartite graphs whose vertices are respectively called *places* and *transitions*.
- The notion of state is *distributed* among places marked with *tokens*, while transitions correspond to activities/events.
- A *multiset* over Pl is a function $M : Pl \to \mathbb{N}$ (element multiplicity), which is finite iff so is the set $\{p \in Pl \mid M(p) > 0\}$.
- A labeled Petri net is a tuple $N = (Pl, Act, Tr, M_0)$ where:
  - Pl is a set of places.
  - $Tr \subseteq \mathcal{M}u_{\text{fin}}(Pl) \times Act \times \mathcal{M}u_{\text{fin}}(Pl)$ is a set of labeled transitions.
  - $M_0 \in \mathcal{M}u_{\text{fin}}(Pl)$ is the initial marking.
- Places are drawn as circles, transitions are drawn as boxes.
- $M(p)$ black dots are drawn inside $p \in Pl$ if M is the current marking.

- Each transition $t \in Tr$ can be written as ${}^\bullet t \xrightarrow{a} t^\bullet$ where:
  - ${}^\bullet t$ is the weighted *preset* of t (places where tokens are consumed).
  - $t^\bullet$ is the weighted *postset* of t (places where tokens are produced).
- An arrow-headed arc is drawn from every place in ${}^\bullet t$ to t as well as from t to every place in $t^\bullet$, each labeled with its place multiplicity.
- Transition t is *enabled* at marking $M \in \mathcal{M}u_{\text{fin}}(Pl)$ iff ${}^\bullet t \subseteq M$.
- The *firing* of t enabled at M produces marking $M' = (M \ominus {}^\bullet t) \oplus t^\bullet$, written $M \,[a\rangle\, M'$.
- The *reachability set* $RS(M)$ of marking $M \in \mathcal{M}u_{\text{fin}}(Pl)$ is the smallest subset of $\mathcal{M}u_{\text{fin}}(Pl)$ such that:
  - $M \in RS(M)$.
  - If $M_1 \in RS(M)$ and $M_1 \,[a\rangle\, M_2$, then $M_2 \in RS(M)$.
- The *reachability graph (or interleaving marking graph)* of N is the LTS $\mathcal{RG}[\![N]\!] = (RS(M_0), Act, [\rangle, M_0)$.

- The Petri net semantics $\mathcal{N}$ of Degano, De Nicola, Montanari (1988) associates a place with any state of a sequential subprocess of $P \in \mathbb{P}$.
- The syntax of places in V is the same as the one of process terms, but binary parallel composition is replaced by unary $\_ \parallel_S \mathrm{id}$ and $\mathrm{id} \parallel_S \_$.
- The decomposition $\mathit{dec} : \mathbb{P} \to \mathcal{M}u_{\mathrm{fin}}(V)$ is defined as follows:

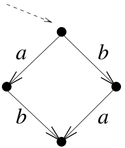
$$
\begin{array}{rcl}
\mathit{dec}(\underline{0}) & = & \{\!|\, \underline{0} \,|\!\} \\
\mathit{dec}(a \,.\, P) & = & \{\!|\, a \,.\, P \,|\!\} \\
\mathit{dec}(P_1 + P_2) & = & \{\!|\, V_1 + V_2 \mid V_1 \in \mathit{dec}(P_1), V_2 \in \mathit{dec}(P_2) \,|\!\} \\
\mathit{dec}(P_1 \parallel_S P_2) & = & \{\!|\, V \parallel_S \mathrm{id} \mid V \in \mathit{dec}(P_1) \,|\!\} \oplus \{\!|\, \mathrm{id} \parallel_S V \mid V \in \mathit{dec}(P_2) \,|\!\} \\
\mathit{dec}(P \,/\, H) & = & \{\!|\, V \,/\, H \mid V \in \mathit{dec}(P) \,|\!\} \\
\mathit{dec}(P \setminus L) & = & \{\!|\, V \setminus L \mid V \in \mathit{dec}(P) \,|\!\} \\
\mathit{dec}(P\,[\varphi]) & = & \{\!|\, V\,[\varphi] \mid V \in \mathit{dec}(P) \,|\!\} \\
\mathit{dec}(B) & = & \mathit{dec}(P) \quad \text{if } B \triangleq P
\end{array}
$$

- Transitions in T stems from semantic rules similar to those for $\mathbb{P}$.
- *Retrievability*: $\mathcal{RG}[\![\mathcal{N}[\![P]\!]]\!]$ is isomorphic to $[\![P]\!]$.

- Let us reconsider the following process terms:

$$a \,.\, b \,.\, \underline{0} + b \,.\, a \,.\, \underline{0}$$
$$a \,.\, \underline{0} \parallel_{\emptyset} b \,.\, \underline{0}$$

- They are indistinguishable according to the interleaving semantics:



- The Petri net semantics shows that they are structurally different:

5. Computational Power of Process Calculi

- Process calculi with the considered operators (including recursion) have the same computational power as Turing machines.
- A Turing machine can be simulated by two stacks together with a finite-state control mechanism.
- In a process calculus, the interaction of the control mechanism with the two stacks can be represented through parallel composition.
- The finite-state control mechanism can be represented by means of action prefix, alternative composition, and recursion, resulting in as many behavioral equations of the following form as there are states:

$$Q_i \triangleq a_{i,j_1} \,.\, Q_{j_1} + a_{i,j_2} \,.\, Q_{j_2} + \cdots + a_{i,j_{n_i}} \,.\, Q_{j_{n_i}}, \;\; 1 \leq i \leq k$$

- Consider a finite set V of values that can be placed in a stack.
- The stack content σ is an element of V^*, which is ε when empty.
- Actions $push$ and pop for modeling stack operations.
- The stack can be inductively specified as follows:

$$
\begin{aligned}
Stack_\varepsilon &\triangleq \sum_{v \in V} push_v \,.\, Stack_v + signal\_empty \,.\, Stack_\varepsilon \\
Stack_{v::\sigma} &\triangleq \sum_{w \in V} push_w \,.\, Stack_{w::v::\sigma} + pop_v \,.\, Stack_\sigma, \ \ \sigma \in V^*
\end{aligned}
$$

- *Infinitely many* behavioral equations because V^* is countable.

- *Finite implementation* based on as many cells as there are values of V in the stack, which are created and linked together as needed:

$$Cell_{v_n} \frown Cell_{v_{n-1}} \frown \ldots \frown Cell_{v_1} \frown Empty$$

- Only $|V| + 2$ equations:

$$\begin{aligned}
Empty &\triangleq \sum_{v \in V} push_v \,.\, (Cell_v \frown Empty) + signal_empty \,.\, Empty \\
Cell_v &\triangleq \sum_{w \in V} push_w \,.\, (Cell_w \frown Cell_v) + pop_v \,.\, Changing, \;\; v \in V \\
Changing &\triangleq \sum_{u \in V} prev_top_u \,.\, Cell_u + bottom_cell \,.\, Empty
\end{aligned}$$

- Definition of the left-associative linking operator $\frown$:

  $$P \frown Q \triangleq P \left[{}^{p_u}/_{prev\_top_u}, {}^{e}/_{bottom\_cell}\right] \|_{\{p_u, e\}} Q \left[{}^{p_u}/_{pop_u}, {}^{e}/_{signal\_empty}\right]$$

  where P stands for $Changing$ and Q stands for $Cell_u$ or $Empty$ while any action with subscript u is a shorthand for all such actions:
- Upon $push$ with the external environment, the leftmost cell spawns a new cell to the left for the new top value of the stack.
- Upon pop with the external environment, the leftmost cell takes on the value of the cell to the right and this behavior propagates till the rightmost cell, which becomes empty (no garbage collection).

6. Spectrum of Behavioral Equivalences

- Establishing whether two process terms are equivalent amounts to establishing whether the systems they represent behave the same.
- *Compositional reasoning*: substituting equals for equals.
- *Abstraction capabilities*: behaving the same up to certain details.
- Useful for theoretical and applicative purposes:
  - Comparing process terms that are syntactically different on the basis of the behavior they exhibit.
  - Relating process algebraic descriptions of the same system at different abstraction levels (top-down modeling).
  - Manipulating process algebraic descriptions in a way that preserves certain properties (state space reduction before model checking).

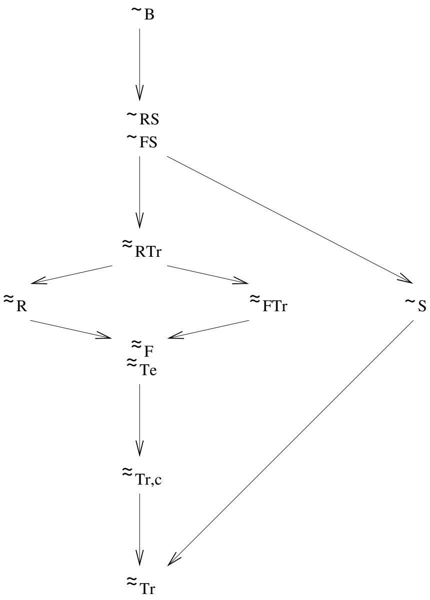
- Features of a good behavioral equivalence:
  - Being a congruence with respect to all the behavioral operators, so as to support compositional reasoning.
  - Having a sound and complete axiomatization, which elucidates the fundamental equational laws of the equivalence with respect to the behavioral operators (rewriting rules for syntactical manipulation).
  - Having a logical characterization, which shows the behavioral properties preserved by the equivalence (diagnostics for inequivalence).
  - Being equipped with an efficient verification algorithm, which runs in polynomial time in the worst case (finite-state systems – undecidable otherwise).
  - Being able to abstract from invisible actions.
- Three fundamental approaches: *trace*, *bisimulation*, *testing* (1980's).

- Trace approach (*Hoare* et al.): two processes are equivalent iff they are able to execute the same sequences of visible actions ($\approx_{\rm Tr}$).
- Abstraction from branching points leads to deadlock insensitivity: $\mathrm{rec}\, X : (a \,.\, X + a \,.\, \underline{0}) \;\approx_{\rm Tr}\; \mathrm{rec}\, X : a \,.\, X$ but the first one can deadlock.
- Deadlock-sensitive (hence finer) variants of trace equivalence:
  - Completed-trace equivalence: compares process terms also with respect to traces that lead to deadlock ($\approx_{\rm Tr,c}$).
  - Failure equivalence: takes into account the set of visible actions that can be refused after executing a trace ($\approx_{\rm F}$).
  - Readiness equivalence: takes into account the set of visible actions that are enabled after executing a trace ($\approx_{\rm R}$).
  - Failure-trace equivalence: takes into account the sets of visible actions that can be refused after all individual steps of a trace ($\approx_{\rm FTr}$).
  - Ready-trace equivalence: takes into account the sets of visible actions that are enabled after all individual steps of a trace ($\approx_{\rm RTr}$).

- Bisimulation approach (*Park*, *Milner*): two processes are equivalent iff they are able to mimic each other's behavior stepwise ($\sim_{\mathrm{B}}$).
- Faithful account of branching points leads to overdiscrimination: $a \,.\, b \,.\, c \,.\, \underline{0} + a \,.\, b \,.\, d \,.\, \underline{0} \not\sim_{\mathrm{B}} a \,.\, (b \,.\, c \,.\, \underline{0} + b \,.\, d \,.\, \underline{0})$ is hardly justifiable.
- Coarser variants of bisimulation equivalence:
    - Simulation equivalence: it is the intersection of two preorders, each considering the capability of stepwise behavior mimicking in one single direction ($\sim_{\mathrm{S}}$).
    - Failure-simulation equivalence: same as simulation equivalence, with each of the two preorders additionally checking for the equality of the sets of actions that can be stepwise refused ($\sim_{\mathrm{FS}}$).
    - Ready-simulation equivalence: same as simulation equivalence, with each of the two preorders additionally checking for the equality of the sets of actions that are stepwise enabled ($\sim_{\mathrm{RS}}$).

- Testing approach (*De Nicola & Hennessy*): two processes are equivalent iff their reaction to tests is the same ($\approx_{\mathrm{Te}}$).
- Tests formalized as processes extended with success action/state ω.
- Interaction between a process and a test formalized through their parallel composition with synchronization on any visible action.
- It holds that $a\,.\,b\,.\,c\,.\,\underline{0} + a\,.\,b\,.\,d\,.\,\underline{0} \approx_{\mathrm{Te}} a\,.\,(b\,.\,c\,.\,\underline{0} + b\,.\,d\,.\,\underline{0})$.
- Intersection of may-testing equivalence (at least one computation leads to success) and must-testing equivalence (all computations lead to success).
- May-testing equivalence coincides with trace equivalence.
- Testing equivalence coincides with failure equivalence in the case of nondiverging, finitely-branching processes.
- The diverging process $\mathrm{rec}\,X : (\tau\,.\,X + a\,.\,\underline{0})$ is not must-testing equivalent to $a\,.\,\underline{0}$ because it can fail test $a\,.\,\omega$ (they are failure equivalent).
- Checking whether every test may/must be passed is expensive.

- Linear-time/branching-time spectrum for finitely-branching processes without τ-actions and hence divergence (*De Nicola*, *Van Glabbeek*):



7. Strong Bisimilarity and Its Properties

- *Simulation game*: whenever the challenger performs a given action, then the defender has to respond with the same action and the two derivatives must be able to repeat this game.
- Bisimulation game is a simulation game in both directions.
- A binary relation $\mathcal{B}$ over $\mathbb{P}$ is a strong bisimulation iff, whenever $(P_1, P_2) \in \mathcal{B}$, then for all actions $a \in Act$:
  - Whenever $P_1 \xrightarrow{a} P_1'$, then $P_2 \xrightarrow{a} P_2'$ with $(P_1', P_2') \in \mathcal{B}$.
  - Whenever $P_2 \xrightarrow{a} P_2'$, then $P_1 \xrightarrow{a} P_1'$ with $(P_1', P_2') \in \mathcal{B}$.
- Strong bisimulation equivalence or strong bisimilarity $\sim_{\mathrm{B}}$ is defined as the union of all the strong bisimulations.

- Properties of strong bisimulations:
  - The identity relation over $\mathbb{P}$ is a strong bisimulation.
  - The inverse of a strong bisimulation is a strong bisimulation.
  - The composition of two strong bisimulations is a strong bisimulation.
  - The denumerable union of strong bisimulations is a strong bisimulation.
- *Coinductive definition*: strong bisimilarity is the *maximum fixed point* of the higher-order relation $\mathcal{F}$ such that for all binary relations $\mathcal{R}$ it holds that $(P_1, P_2) \in \mathcal{F}(\mathcal{R})$ iff for all actions $a \in Act$:
  - Whenever $P_1 \xrightarrow{a} P_1'$, then $P_2 \xrightarrow{a} P_2'$ with $(P_1', P_2') \in \mathcal{R}$.
  - Whenever $P_2 \xrightarrow{a} P_2'$, then $P_1 \xrightarrow{a} P_1'$ with $(P_1', P_2') \in \mathcal{R}$.
- $\mathcal{R}$ is a strong bisimulation iff $\mathcal{R} \subseteq \mathcal{F}(\mathcal{R})$.

- In order for $P_1 \sim_B P_2$ it is necessary they enable the same actions, i.e., for all $a \in Act$ there exist $P_1', P_2' \in \mathbb{P}$ such that:

$$P_1 \xrightarrow{a} P_1' \iff P_2 \xrightarrow{a} P_2'$$

- Focus on important pairs of processes that form a strong bisimulation.
- A binary relation $\mathcal{B}$ over $\mathbb{P}$ is a strong bisimulation up to $\sim_B$ iff, whenever $(P_1, P_2) \in \mathcal{B}$, then for all actions $a \in Act$:
  - Whenever $P_1 \xrightarrow{a} P_1'$, then $P_2 \xrightarrow{a} P_2'$ with $P_1' \sim_B Q_1 \mathcal{B} Q_2 \sim_B P_2'$.
  - Whenever $P_2 \xrightarrow{a} P_2'$, then $P_1 \xrightarrow{a} P_1'$ with $P_1' \sim_B Q_1 \mathcal{B} Q_2 \sim_B P_2'$.
- In order for $P_1 \sim_B P_2$ it is sufficient a strong bisimulation up to $\sim_B$ that contains (P_1, P_2).

- $\sim_{\rm B}$ is a congruence with respect to all dynamic and static operators as well as recursion.
- Substituting equals for equals does not alter the overall behavior in any process context.
- Let $P_1, P_2 \in \mathbb{P}$. If $P_1 \sim_{\rm B} P_2$ then:
  - $a \,.\, P_1 \ \sim_{\rm B}\ a \,.\, P_2$ for all $a \in Act$.
  - $P_1 + P \ \sim_{\rm B}\ P_2 + P$, $P + P_1 \ \sim_{\rm B}\ P + P_2$ for all $P \in \mathbb{P}$.
  - $P_1 \,\|_S\, P \ \sim_{\rm B}\ P_2 \,\|_S\, P$, $P \,\|_S\, P_1 \ \sim_{\rm B}\ P \,\|_S\, P_2$ for all $P \in \mathbb{P}, S \subseteq Act_{\rm v}$.
  - $P_1 \,/\, H \ \sim_{\rm B}\ P_2 \,/\, H$ for all $H \subseteq Act_{\rm v}$.
  - $P_1 \setminus L \ \sim_{\rm B}\ P_2 \setminus L$ for all $L \subseteq Act_{\rm v}$.
  - $P_1[\varphi] \ \sim_{\rm B}\ P_2[\varphi]$ for all $\varphi \in Relab$.

- Recursion: extend $\sim_{\mathrm{B}}$ to *open* process terms by replacing all variables freely occurring outside rec binders with every closed process term.
- Let P_1, P_2 be guarded process terms having free occurrences of at most $k \in \mathbb{N}$ process variables $X_1, \ldots, X_k \in \mathit{Var}$.
- We define $P_1 \sim_{\mathrm{B}} P_2$ iff for all $Q_1, \ldots, Q_k \in \mathbb{P}$:

$$P_1\{Q_i \hookrightarrow X_i \mid 1 \leq i \leq k\} \ \sim_{\mathrm{B}} \ P_2\{Q_i \hookrightarrow X_i \mid 1 \leq i \leq k\}$$

- If $P_1 \sim_{\mathrm{B}} P_2$ then $\mathrm{rec}\, X : P_1 \ \sim_{\mathrm{B}} \ \mathrm{rec}\, X : P_2$.

- $\sim_{\mathrm{B}}$ has a sound and complete axiomatization over the set $\mathbb{P}_{\mathrm{nrec}}$ of nonrecursive process terms of $\mathbb{P}$.
- Basic laws (associativity, commutativity, and neutral element of $+$):

$$
\begin{array}{lrcl}
(\mathcal{A}_{\mathrm{B},1}) & (P_1 + P_2) + P_3 & = & P_1 + (P_2 + P_3) \\
(\mathcal{A}_{\mathrm{B},2}) & P_1 + P_2 & = & P_2 + P_1 \\
(\mathcal{A}_{\mathrm{B},3}) & P + \underline{0} & = & P
\end{array}
$$

- Characterizing laws (idempotency of $+$):

$$
\begin{array}{lrcl}
(\mathcal{A}_{\mathrm{B},4}) & P + P & = & P
\end{array}
$$

- We can reduce any such process in *sum normal form* $\sum\limits_{i \in I} a_i \,.\, P_i$.

- Expansion law (interleaving view of concurrency; I and J nonempty and finite):

$$
\begin{aligned}
(\mathcal{A}_{\mathrm{B},5}) \quad \sum_{i\in I} a_i \,.\, P_i \parallel_S \sum_{j\in J} b_j \,.\, Q_j \;&=\; \sum_{k\in I, a_k\notin S} a_k \,.\, \Big(P_k \parallel_S \sum_{j\in J} b_j \,.\, Q_j\Big) \;+\\
&\quad\; \sum_{h\in J, b_h\notin S} b_h \,.\, \Big(\sum_{i\in I} a_i \,.\, P_i \parallel_S Q_h\Big) \;+\\
&\quad\; \sum_{k\in I, a_k\in S} \; \sum_{h\in J, b_h = a_k} a_k \,.\, (P_k \parallel_S Q_h)\\
(\mathcal{A}_{\mathrm{B},6}) \quad \sum_{i\in I} a_i \,.\, P_i \parallel_S \underline{0} \;&=\; \sum_{k\in I, a_k\notin S} a_k \,.\, P_k\\
(\mathcal{A}_{\mathrm{B},7}) \quad \underline{0} \parallel_S \sum_{j\in J} b_j \,.\, Q_j \;&=\; \sum_{h\in J, b_h\notin S} b_h \,.\, Q_h\\
(\mathcal{A}_{\mathrm{B},8}) \quad \underline{0} \parallel_S \underline{0} \;&=\; \underline{0}
\end{aligned}
$$

- Distribution laws (for unary static operators):

$$
\begin{array}{lrcll}
(\mathcal{A}_{\mathrm{B},9}) & \underline{0} \,/\, H & = & \underline{0} & \\
(\mathcal{A}_{\mathrm{B},10}) & (a \,.\, P) \,/\, H & = & \tau \,.\, (P \,/\, H) & \text{if } a \in H \\
(\mathcal{A}_{\mathrm{B},11}) & (a \,.\, P) \,/\, H & = & a \,.\, (P \,/\, H) & \text{if } a \notin H \\
(\mathcal{A}_{\mathrm{B},12}) & (P_1 + P_2) \,/\, H & = & P_1 \,/\, H + P_2 \,/\, H & \\
\\
(\mathcal{A}_{\mathrm{B},13}) & \underline{0} \setminus L & = & \underline{0} & \\
(\mathcal{A}_{\mathrm{B},14}) & (a \,.\, P) \setminus L & = & \underline{0} & \text{if } a \in L \\
(\mathcal{A}_{\mathrm{B},15}) & (a \,.\, P) \setminus L & = & a \,.\, (P \setminus L) & \text{if } a \notin L \\
(\mathcal{A}_{\mathrm{B},16}) & (P_1 + P_2) \setminus L & = & P_1 \setminus L + P_2 \setminus L & \\
\\
(\mathcal{A}_{\mathrm{B},17}) & \underline{0}\,[\varphi] & = & \underline{0} & \\
(\mathcal{A}_{\mathrm{B},18}) & (a \,.\, P)\,[\varphi] & = & \varphi(a) \,.\, (P\,[\varphi]) & \\
(\mathcal{A}_{\mathrm{B},19}) & (P_1 + P_2)\,[\varphi] & = & P_1\,[\varphi] + P_2\,[\varphi] &
\end{array}
$$

- $Ded(\mathcal{A}_{\mathrm{B}})$ is a deduction system based on all the previous axioms plus:
  - Reflexivity: $\mathcal{A}_{\mathrm{B}} \vdash P = P$.
  - Symmetry: $\mathcal{A}_{\mathrm{B}} \vdash P_1 = P_2 \implies \mathcal{A}_{\mathrm{B}} \vdash P_2 = P_1$.
  - Transitivity: $\mathcal{A}_{\mathrm{B}} \vdash P_1 = P_2 \wedge \mathcal{A}_{\mathrm{B}} \vdash P_2 = P_3 \implies \mathcal{A}_{\mathrm{B}} \vdash P_1 = P_3$.
  - Substitutivity: $\mathcal{A}_{\mathrm{B}} \vdash P_1 = P_2 \implies \mathcal{A}_{\mathrm{B}} \vdash a\,.\,P_1 = a\,.\,P_2 \wedge \ldots$.
- Remember that $\sim_{\mathrm{B}}$ is an equivalence relation and a congruence.
- The deduction system $Ded(\mathcal{A}_{\mathrm{B}})$ is sound and complete for $\sim_{\mathrm{B}}$ over $\mathbb{P}_{\mathrm{nrec}}$, i.e., $\mathcal{A}_{\mathrm{B}} \vdash P_1 = P_2 \iff P_1 \sim_{\mathrm{B}} P_2$ for all $P_1, P_2 \in \mathbb{P}_{\mathrm{nrec}}$.

- $\sim_{\mathrm{B}}$ has a modal logic characterization based on HML, i.e., Hennessy-Milner logic.
- Basic truth values and logical connectives, plus modal operators expressing how to behave after executing certain actions.
- Syntax of HML $(a \in Act)$:

$$\begin{array}{rcll} \phi & ::= & \mathrm{true} & \text{basic truth value} \\ & | & \neg\phi & \text{negation} \\ & | & \phi \wedge \phi & \text{conjunction} \\ & | & \langle a \rangle \phi & \text{possibility} \end{array}$$

plus derived logical operators:

$$\begin{array}{rcll} \mathrm{false} & \equiv & \neg\mathrm{true} & \text{basic truth value} \\ \phi_1 \vee \phi_2 & \equiv & \neg(\neg\phi_1 \wedge \neg\phi_2) & \text{disjunction} \\ [a]\phi & \equiv & \neg\langle a \rangle \neg\phi & \text{necessity} \end{array}$$

- Interpretation of HML over $\mathbb{P}$:

$$
\begin{array}{lcll}
P & \models & \mathrm{true} & \\
P & \models & \neg\phi & \text{if } P \not\models \phi \\
P & \models & \phi_1 \wedge \phi_2 & \text{if } P \models \phi_1 \text{ and } P \models \phi_2 \\
P & \models & \langle a \rangle \phi & \text{if there exists } P' \in \mathbb{P} \text{ such that } P \xrightarrow{a} P' \text{ and } P' \models \phi
\end{array}
$$

plus derived logical operators:

$$
\begin{array}{lcll}
P & \not\models & \mathrm{false} & \\
P & \models & \phi_1 \vee \phi_2 & \text{if } P \models \phi_1 \text{ or } P \models \phi_2 \\
P & \models & [a]\phi & \text{if for all } P' \in \mathbb{P}\text{, whenever } P \xrightarrow{a} P'\text{, then } P' \models \phi
\end{array}
$$

- $P_1 \sim_{\mathrm{B}} P_2 \iff (\forall \phi \in \mathrm{HML}.\, P_1 \models \phi \iff P_2 \models \phi)$ for all $P_1, P_2 \in \mathbb{P}$.

- $\sim_{\mathrm{B}}$ has a temporal logic characterization based on CTL*.
- *State formulae* with atomic propositions and logical connectives, plus *path formulae* including temporal operators about the future.
- A path π is a sequence of states $s_0 s_1 s_2 \ldots$ such that $s_i \longrightarrow s_{i+1}$.
- CTL* is interpreted over *Kripke structures*, hence we redefine $\sim_{\mathrm{B}}$ (propositions labeling states instead of actions labeling transitions).
- A binary relation $\mathcal{B}$ over a Kripke structure $(S, \mathcal{L}, \longrightarrow)$ is a strong bisimulation iff, whenever $(s_1, s_2) \in \mathcal{B}$, then $\mathcal{L}(s_1) = \mathcal{L}(s_2)$ and:
    - Whenever $s_1 \longrightarrow s_1'$, then $s_2 \longrightarrow s_2'$ with $(s_1', s_2') \in \mathcal{B}$.
    - Whenever $s_2 \longrightarrow s_2'$, then $s_1 \longrightarrow s_1'$ with $(s_1', s_2') \in \mathcal{B}$.
- Strong bisimilarity $\sim_{\mathrm{B}}$ is the union of all the strong bisimulations.

- Syntax of CTL*:

$$
\begin{array}{rcll}
\phi & ::= & p & \text{atomic proposition} \\
& \mid & \neg\phi & \text{state formula negation} \\
& \mid & \phi \wedge \phi & \text{state formula conjunction} \\
& \mid & \mathbf{E}\,\psi & \text{existential path quantifier} \\
\\
\psi & ::= & \phi & \text{state formula} \\
& \mid & \neg\psi & \text{path formula negation} \\
& \mid & \psi \wedge \psi & \text{path formula conjunction} \\
& \mid & \mathbf{X}\,\psi & \text{next operator} \\
& \mid & \psi\,\mathbf{U}\,\psi & \text{until operator}
\end{array}
$$

plus derived logical operators:

$$
\begin{array}{rcll}
\mathbf{A}\,\psi & \equiv & \neg\mathbf{E}\,\neg\psi & \text{universal path quantifier} \\
\mathbf{F}\,\psi & \equiv & \text{true}\,\mathbf{U}\,\psi & \text{eventually operator} \\
\mathbf{G}\,\psi & \equiv & \neg\mathbf{F}\,\neg\psi & \text{globally operator}
\end{array}
$$

- Interpretation of CTL* over Kripke structure $(S, \mathcal{L}, \longrightarrow)$:

$$\begin{array}{lcll}
s & \models & p & \text{if } p \in \mathcal{L}(s) \\
s & \models & \neg\phi & \text{if } s \not\models \phi \\
s & \models & \phi_1 \wedge \phi_2 & \text{if } s \models \phi_1 \text{ and } s \models \phi_2 \\
s & \models & \mathbf{E}\,\psi & \text{if there exists a path } \pi \text{ starting from } s \text{ such that } \pi \models \psi \\
\\
\pi & \models & \phi & \text{if } \pi[0] \models \phi \\
\pi & \models & \neg\psi & \text{if } \pi \not\models \psi \\
\pi & \models & \psi_1 \wedge \psi_2 & \text{if } \pi \models \psi_1 \text{ and } \pi \models \psi_2 \\
\pi & \models & \mathbf{X}\,\psi & \text{if } \pi^1 \models \psi \\
\pi & \models & \psi_1 \,\mathbf{U}\, \psi_2 & \text{if } \pi^k \models \psi_2 \text{ for some } k \geq 0 \text{ and } \pi^i \models \psi_1 \text{ for all } 0 \leq i < k
\end{array}$$

  where, given path $\pi = s_0 s_1 s_2 ...$, $\pi[0]$ is the initial state s_0 of π while π^k is the subpath starting from state s_k.

- $s_1 \sim_{\mathrm{B}} s_2 \iff (\forall \phi \in \mathrm{CTL}^*.\, s_1 \models \phi \iff s_2 \models \phi)$ for all $s_1, s_2 \in S$.

- $\sim_{\mathrm{B}}$ is decidable in polynomial time over the set $\mathbb{P}_{\mathrm{fin}}$ of finite-state terms of $\mathbb{P}$ through Kanellakis-Smolka partition refinement algorithm.
- Based on the fact that $\sim_{\mathrm{B}}$ can be characterized as the limit of a sequence of successively finer equivalence relations:

$$\sim_{\mathrm{B}} \; = \; \bigcap_{i \in \mathbb{N}} \sim_{\mathrm{B},i}$$

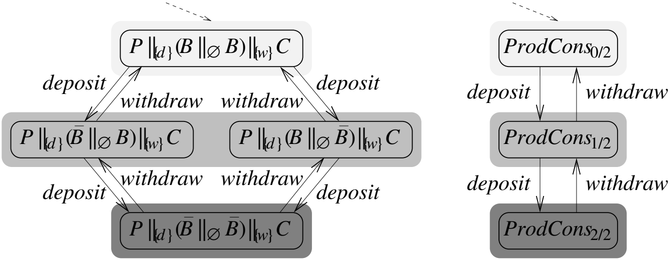
- $\sim_{\mathrm{B},0} = \; \mathbb{P} \times \mathbb{P}$ thus inducing the trivial partition $\{\mathbb{P}\}$.
- $\sim_{\mathrm{B},1}$ refines $\{\mathbb{P}\}$ by creating an equivalence class for each set of process terms that satisfy the necessary condition for $\sim_{\mathrm{B}}$.
- Whenever $P_1 \sim_{\mathrm{B},i} P_2$, $i \in \mathbb{N}_{\geq 1}$, then for all actions $a \in Act$:
    - Whenever $P_1 \xrightarrow{a} P_1'$, then $P_2 \xrightarrow{a} P_2'$ with $P_1' \sim_{\mathrm{B},i-1} P_2'$.
    - Whenever $P_2 \xrightarrow{a} P_2'$, then $P_1 \xrightarrow{a} P_1'$ with $P_1' \sim_{\mathrm{B},i-1} P_2'$.

- Steps of the algorithm for checking whether $P_1 \sim_{\mathrm{B}} P_2$ $(P_1, P_2 \in \mathbb{P}_{\mathrm{fin}})$:
  1. Build an initial partition with a single class including all the states of $[\![P_1]\!]$ and all the states of $[\![P_2]\!]$.
  2. Initialize a list of splitters with the above class as its only element.
  3. While the list of splitters is not empty, select a splitter and remove it from the list after refining the current partition for all $a \in Act_{P_1,P_2}$:
     a. Split each class of the current partition by comparing its states when executing actions of name a that lead to the selected splitter.
     b. For each class that has been split, insert its smallest subclass into the list of splitters ("process the smaller half").
  4. Return yes/no depending on whether the initial states of $[\![P_1]\!]$ and $[\![P_2]\!]$ belong to the same class of the final partition or not.
- The time complexity is $O(m \cdot \log n)$, where n is the number of states and m is the number of transitions of $[\![P_1]\!]$ and $[\![P_2]\!]$ as a whole.
- Also useful for state space minimization before model checking.

- **Running example** (strong bisimilarity):
  - Concurrent implementation with two independent one-position buffers:

$$\begin{aligned}
PC_{\mathrm{conc},2} &\triangleq Prod \parallel_{\{deposit\}} (Buff \parallel_{\emptyset} Buff) \parallel_{\{withdraw\}} Cons \\
Prod &\triangleq deposit \,.\, Prod \\
Buff &\triangleq deposit \,.\, withdraw \,.\, Buff \\
Cons &\triangleq withdraw \,.\, Cons
\end{aligned}$$

  - Strong bisimulation proving $PC_{\mathrm{conc},2} \sim_{\mathrm{B}} ProdCons_{0/2}$:



8. Weak Bisimilarity and Its Properties

- $\sim_{\mathrm{B}}$ does not abstract from invisible actions: $a\,.\,b\,.\,\underline{0} \not\sim_{\mathrm{B}} a\,.\,\tau\,.\,b\,.\,\underline{0}$.
- Two processes should be deemed equivalent in the bisimulation game if they are able to mimic each other's visible behavior stepwise.
- Need to extend the transition relation $\longrightarrow$ to action sequences.
- $P \xRightarrow{a_1 \dots a_n} P'$ iff:
  - either $n = 0$ and $P \equiv P'$, meaning that P stays idle;
  - or $n \geq 1$ and there exist $P_0, P_1, \dots, P_n \in \mathbb{P}$ such that:
    - $P \equiv P_0$;
    - $P_{i-1} \xrightarrow{a_i} P_i$ for all $1 \leq i \leq n$;
    - $P_n \equiv P'$.
- τ^* denotes a possibly empty, finite sequence of τ-actions.

- A binary relation $\mathcal{B}$ over $\mathbb{P}$ is a weak bisimulation iff, whenever $(P_1, P_2) \in \mathcal{B}$, then:
  - Whenever $P_1 \xrightarrow{\tau} P_1'$, then $P_2 \overset{\tau^*}{\Longrightarrow} P_2'$ with $(P_1', P_2') \in \mathcal{B}$.
  - Whenever $P_2 \xrightarrow{\tau} P_2'$, then $P_1 \overset{\tau^*}{\Longrightarrow} P_1'$ with $(P_1', P_2') \in \mathcal{B}$.
  - For all visible actions $a \in Act_{\mathrm{v}}$:
    - Whenever $P_1 \xrightarrow{a} P_1'$, then $P_2 \overset{\tau^* a \tau^*}{\Longrightarrow} P_2'$ with $(P_1', P_2') \in \mathcal{B}$.
    - Whenever $P_2 \xrightarrow{a} P_2'$, then $P_1 \overset{\tau^* a \tau^*}{\Longrightarrow} P_1'$ with $(P_1', P_2') \in \mathcal{B}$.
- Weak bisimulation equivalence or weak bisimilarity $\approx_{\mathrm{B}}$ is the union of all the weak bisimulations.
- The necessary condition for $\sim_{\mathrm{B}}$ is too restrictive for $\approx_{\mathrm{B}}$.
- The sufficient condition for $\sim_{\mathrm{B}}$ generalizes to $\approx_{\mathrm{B}}$ provided that weak bisimulation up to $\approx_{\mathrm{B}}$ uses $\overset{\tau^* a \tau^*}{\Longrightarrow}$ on the challenger side too.

- $\approx_{\mathrm{B}}$ is a congruence with respect to all the behavioral operators except for alternative composition (not a problem in practice).
- Additional τ-laws highlighting its abstraction capabilities:

$$\begin{aligned} \tau\,.\,P &= P \\ a\,.\,\tau\,.\,P &= a\,.\,P \\ P + \tau\,.\,P &= \tau\,.\,P \\ a\,.\,(P_1 + \tau\,.\,P_2) + a\,.\,P_2 &= a\,.\,(P_1 + \tau\,.\,P_2) \end{aligned}$$

- Weak modal operators replacing those of HML ($a \in Act_{\mathrm{v}}$):

$$\begin{array}{lcll} P & \models & \langle\!\langle \tau \rangle\!\rangle \phi & \text{if there exists } P' \in \mathbb{P} \text{ such that } P \xRightarrow{\tau^*} P' \text{ and } P' \models \phi \\ P & \models & \langle\!\langle a \rangle\!\rangle \phi & \text{if there exists } P' \in \mathbb{P} \text{ such that } P \xRightarrow{\tau^* a \tau^*} P' \text{ and } P' \models \phi \end{array}$$

- Temporal logic characterization based on CTL* without $\mathbf{X}$.

- $P_1 \approx_{\mathrm{B}} P_2$ can be decided in $O(n^2 \cdot m \cdot \log n)$ time with the verification algorithm for $\sim_{\mathrm{B}}$ preceded by the following preprocessing step:

   0. Build the reflexive and transitive closure of $\xrightarrow{\tau}$ in $[\![P_i]\!]$ for $i = 1, 2$:

      a. Add a looping τ-transition to each state.

      b. Add a τ-transition between the initial state and the final state of any sequence of at least two τ-transitions, if the two states are distinct and all the transitions in the sequence are distinct and nonlooping.

      c. Add an a-transition, $a \in Act_{\mathrm{v}}$, between the initial state and the final state of any sequence of at least two transitions in which one is labeled with a, if all the other transitions in the sequence are labeled with τ, distinct, and nonlooping.

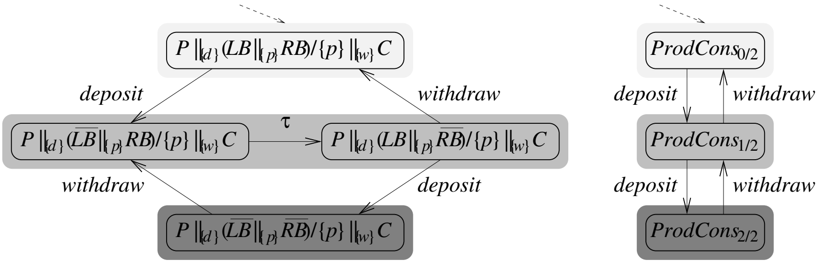
- The fact that $\approx_{\mathrm{B}}$ is not a congruence with respect to the alternative composition operator stems from $\tau \,.\, P = P$ (abstraction from initial τ-actions).
- This τ-law cannot be freely used when P does not enable τ-actions: $\tau \,.\, a \,.\, \underline{0} \approx_{\mathrm{B}} a \,.\, \underline{0}$ but $\tau \,.\, a \,.\, \underline{0} + b \,.\, \underline{0} \not\approx_{\mathrm{B}} a \,.\, \underline{0} + b \,.\, \underline{0}$.
- Congruence w.r.t. the alternative composition operator can be restored by enforcing a matching on *initial* τ-actions in the game.
- $P_1 \in \mathbb{P}$ is weakly bisimulation congruent to $P_2 \in \mathbb{P}$, written $P_1 \approx_{\mathrm{B}}^{\mathrm{c}} P_2$, iff for all actions $a \in Act$ (hence including τ):
  - Whenever $P_1 \xrightarrow{a} P_1'$, then $P_2 \xRightarrow{\tau^* a \tau^*} P_2'$ with $P_1' \approx_{\mathrm{B}} P_2'$.
  - Whenever $P_2 \xrightarrow{a} P_2'$, then $P_1 \xRightarrow{\tau^* a \tau^*} P_1'$ with $P_1' \approx_{\mathrm{B}} P_2'$.
- $\approx_{\mathrm{B}}^{\mathrm{c}}$ is strictly finer than $\approx_{\mathrm{B}}$: $\tau \,.\, a \,.\, \underline{0} \not\approx_{\mathrm{B}}^{\mathrm{c}} a \,.\, \underline{0}$.
- $\approx_{\mathrm{B}}^{\mathrm{c}}$ is the largest congruence w.r.t. $+$ contained in $\approx_{\mathrm{B}}$.

- $\approx_{\mathrm{B}}$ and $\approx^{\mathrm{c}}_{\mathrm{B}}$ do not fully retain the property possessed by $\sim_{\mathrm{B}}$ of *respecting the branching structure* of process terms.
- Given $P_1 \approx_{\mathrm{B}} P_2$, when $P_1 \xrightarrow{a} P'_1$ for $a \in Act_{\mathrm{v}}$ then $P_2 \stackrel{\tau^*}{\Longrightarrow} Q \xrightarrow{a} Q' \stackrel{\tau^*}{\Longrightarrow} P'_2$ with $P'_1 \approx_{\mathrm{B}} P'_2$ but we do not know whether any relation exists between P_1 and Q as well as P'_1 and Q'.
- Branching structure preservation can be restored by requiring that P_1 / P_2 be equivalent to Q and P'_1 / P'_2 be equivalent to Q': branching bisimilarity $\approx_{\mathrm{B,b}}$ (*Van Glabbeek & Weijland*).
- $\approx_{\mathrm{B,b}}$ is strictly finer than $\approx_{\mathrm{B}}$, same compositionality issue w.r.t. $+$.
- $\approx_{\mathrm{B,b}}$ coincides with $\approx_{\mathrm{B}}$ on any pair of process terms such that at most one of them reaches a process term enabling τ-actions.
- A single τ-law for $\approx_{\mathrm{B,b}}$: $a \,.\, (\tau \,.\, (P_1 + P_2) + P_1) = a \,.\, (P_1 + P_2)$.
- Can be decided more efficiently: $O(m \cdot \log n)$ time (*Groote* et al.).

- **Running example** (weak bisimilarity):
  - Pipeline implementation with two communicating one-position buffers:

$$
\begin{aligned}
PC_{\text{pipe},2} &\triangleq Prod \parallel_{\{deposit\}} (LBuff \parallel_{\{pass\}} RBuff)/\{pass\} \parallel_{\{withdraw\}} Cons \\
Prod &\triangleq deposit\,.\,Prod \\
LBuff &\triangleq deposit\,.\,pass\,.\,LBuff \\
RBuff &\triangleq pass\,.\,withdraw\,.\,RBuff \\
Cons &\triangleq withdraw\,.\,Cons
\end{aligned}
$$

  - Weak bisimulation proving $PC_{\text{pipe},2} \approx_{\text{B}} ProdCons_{0/2}$:



9. Bibliographic References

[1] S. Abramsky,
"Observational Equivalence as a Testing Equivalence",
Theoretical Computer Science 53:225–241, 1987.

[2] A. Aldini, M. Bernardo, F. Corradini,
"A Process Algebraic Approach to Software Architecture Design",
Springer, 2010.

[3] J.C.M. Baeten, W.P. Weijland,
"Process Algebra",
Cambridge University Press, 1990.

[4] H. Bekic,
"Towards a Mathematical Theory of Processes",
Tech. Rep. TR 25.125, IBM Laboratory Vienna (Austria), 1971
(published in *"Programming Languages and Their Definition"*, Springer, LNCS 177:168–206, 1984).

[5] J.A. Bergstra, J.W. Klop,
"Process Algebra for Synchronous Communication",
Information and Control 60:109–137, 1984.

[6] J.A. Bergstra, A. Ponse, S.A. Smolka (eds.),
"Handbook of Process Algebra",
Elsevier, 2001.

[7] B. Bloom, S. Istrail, A.R. Meyer,
"Bisimulation Can't Be Traced",
Journal of the ACM 42:232–268, 1995.

[8] T. Bolognesi, E. Brinksma,
"Introduction to the ISO Specification Language LOTOS",
Computer Networks and ISDN Systems 14:25–59, 1987.

[9] S.D. Brookes, C.A.R. Hoare, A.W. Roscoe,
"A Theory of Communicating Sequential Processes",
Journal of the ACM 31:560–599, 1984.

[10] M.C. Browne, E.M. Clarke, O. Grümberg,
"Characterizing Finite Kripke Structures in Propositional Temporal Logic",
Theoretical Computer Science 59:115–131, 1988.

[11] R. Cleaveland, M. Hennessy,
"Testing Equivalence as a Bisimulation Equivalence",
Formal Aspects of Computing 5:1–20, 1993.

[12] R. Cleaveland, J. Parrow, B. Steffen,
"The Concurrency Workbench: A Semantics-Based Tool for the Verification of Concurrent Systems",
ACM Trans. on Programming Languages and Systems 15:36–72, 1993.

[13] R. Cleaveland, O. Sokolsky,
"Equivalence and Preorder Checking for Finite-State Systems",
in [6]:391–424.

[14] P. Degano, R. De Nicola, U. Montanari,
"A Distributed Operational Semantics for CCS Based on Condition/Event Systems",
Acta Informatica 26:59–91, 1988.

[15] R. De Nicola,
"Extensional Equivalences for Transition Systems",
Acta Informatica 24:211–237, 1987.

[16] R. De Nicola, M. Hennessy,
"Testing Equivalences for Processes",
Theoretical Computer Science 34:83–133, 1984.

[17] H. Garavel, R. Mateescu, F. Lang, W. Serwe,
"CADP 2006: A Toolbox for the Construction and Analysis of Distributed Processes",
in Proc. of the *19th Int. Conf. on Computer Aided Verification (CAV 2007)*,
Springer, LNCS 4590:158–163, Berlin (Germany), 2007.

[18] R.J. van Glabbeek,
"The Linear Time – Branching Time Spectrum I",
in [6]:3–99.

[19] R.J. van Glabbeek, W.P. Weijland,
"Branching Time and Abstraction in Bisimulation Semantics",
Journal of the ACM 43:555–600, 1996.

[20] M. Hennessy,
"Acceptance Trees",
Journal of the ACM 32:896–928, 1985.

[21] M. Hennessy,
"Algebraic Theory of Processes",
MIT Press, 1988.

[22] M. Hennessy, R. Milner,
"Algebraic Laws for Nondeterminism and Concurrency",
Journal of the ACM 32:137–162, 1985.

[23] C.A.R. Hoare,
"Communicating Sequential Processes",
Prentice Hall, 1985.

[24] D.N. Jansen, J.F. Groote, J.J.A. Keiren, A. Wijs,
"An $O(m \log n)$ Algorithm for Branching Bisimilarity on Labelled Transition Systems",
in Proc. of the *26th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2020)*,
Springer, LNCS 12079:3–20, Dublin (Ireland), 2020.

[25] P.C. Kanellakis, S.A. Smolka,
"CCS Expressions, Finite State Processes, and Three Problems of Equivalence",
Information and Computation 86:43–68, 1990.

[26] R.M. Keller,
"Formal Verification of Parallel Programs",
Communications of the ACM 19:371–384, 1976.

[27] R. Milner,
"A Complete Inference System for a Class of Regular Behaviours",
Journal of Computer and System Sciences 28:439–466, 1984.

[28] R. Milner,
"Communication and Concurrency",
Prentice Hall, 1989.

[29] R. Milner,
"Communicating and Mobile Systems: The π-Calculus",
Cambridge University Press, 1999.

[30] R. Milner, J. Parrow, D. Walker,
"A Calculus of Mobile Processes",
Information and Computation 100:1–77, 1992.

[31] E.-R. Olderog, C.A.R. Hoare,
"Specification-Oriented Semantics for Communicating Processes",
Acta Informatica 23:9–66, 1986.

[32] R. Paige, R.E. Tarjan,
"Three Partition Refinement Algorithms",
SIAM Journal on Computing 16:973–989, 1987.

[33] D. Park,
"Concurrency and Automata on Infinite Sequences",
in Proc. of the *5th GI Conf. on Theoretical Computer Science*,
Springer, LNCS 104:167–183, Karlsruhe (Germany), 1981.

[34] C.A. Petri,
"Kommunikation mit Automaten",
Ph.D. Thesis, Technical University of Darmstadt (Germany), 1962.

[35] G.D. Plotkin,
"A Structural Approach to Operational Semantics",
Tech. Rep. DAIMI-FN-19, Aarhus University (Denmark), 1981
(published in Journal of Logic and Algebraic Programming 60/61:17–139, 2004).

[36] D. Sangiorgi, R. Milner,
"The Problem of "Weak Bisimulation up to"",
in Proc. of the *3rd Int. Conf. on Concurrency Theory (CONCUR 1992)*,
Springer, LNCS 630:32–46, Stony Brook (USA), 1992.

[37] D. Sangiorgi, D. Walker,
"The π-Calculus: A Theory of Mobile Processes",
Cambridge University Press, 2001.

[38] G. Winskel,
"Events in Computation",
Ph.D. Thesis, University of Edinburgh (UK), 1980.