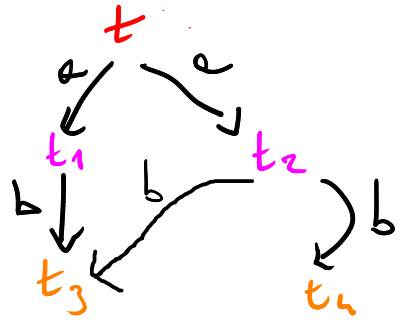
Odd cases

Bisim. ?



$B = \{ (s, t) ,$

$(s_1, t_1) , (s_1, t_2) , (t_1, t_2) ,$

$(s_2, t_3) , (s_2, t_4) , (t_3, t_4)$

$\}$

Properties of bisimulations

Bisimulations are not (necessarily) equivalence relations!

Thm: Given a transition system on S, Id_S is a bisimulation

Proof: Exercise

Thm: Given a set of indexes I, let B_i be bisimulations for all $i \in I$

(1) B_i^{-1} is a bisimulation for all $i \in I$

(2) $\bigcup_{i \in I} B_i$ is a bisimulation

(3) $B_i \circ B_j$ is a bisimulation for all $i, j \in I$

Proof (1): $(s,t) \in B_i^{-1} \iff (t,s) \in B_i$

for each $t \xrightarrow{a} t'$ there is $s \xrightarrow{a} s'$ s.t. $(s',t') \in B_i \Rightarrow (t',s') \in B_i^{-1}$

and likewise for each $s \xrightarrow{a} s'$ using the first clause of Def of bisim.

Proof (2): Exercise

Proof (3) $(r, t) \in B_i \circ B_j \iff \exists\, s : (r, s) \in B_i$ and $(s, t) \in B_j$

$\Rightarrow \forall\, r \xrightarrow{a} r' \quad \exists\, s \xrightarrow{a} s' : \; r' \; B_i \; s'$

$\Rightarrow \exists\, t \xrightarrow{a} t' \; : s' B_j t' \quad \Rightarrow \quad r' \; B_i \circ B_j \; t'$

And likewise for the other clause $\square$

Bisimilarity $\sim \; = \bigcup \{ B : B \text{ is a bisimulation} \}$

Thm $\sim$ is the largest bisimulation (!)

Thm $\sim$ is an equivalence relation

What about communication?

Let $A_\perp = A \cup \{\perp\}$ $\quad \perp \notin A$ and fix a communication function

$$-\circ- : A_\perp \times A_\perp \to A_\perp \begin{cases} \circ \text{ commutative} \\ \circ \text{ associative} \\ \forall a \in A_\perp : a \circ 1 = 1 \circ a = 1 \end{cases}$$

$$(com_1)\ \frac{x \xrightarrow{a} x' \quad y \xrightarrow{b} y' \quad a \circ b \in A}{x \parallel y \xrightarrow{a \circ b} x' \parallel y'}$$

$$(com_2)\ \frac{x \xrightarrow{a} 1 \quad y \xrightarrow{b} y' \quad a \circ b \in A}{x \parallel y \xrightarrow{a \circ b} y'}$$

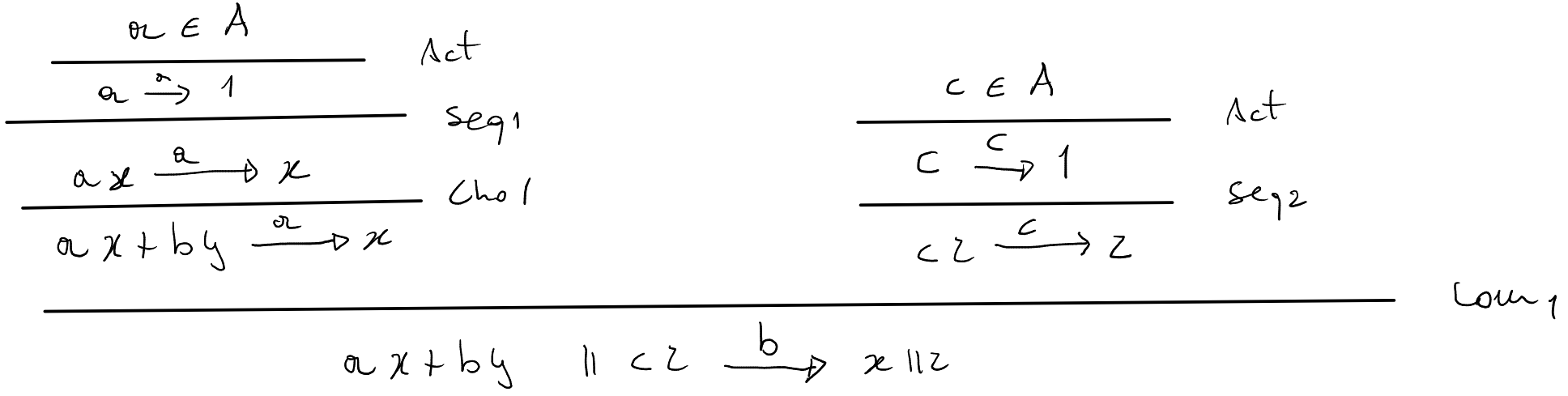
$$(com_3)\ \frac{x \xrightarrow{a} x' \quad y \xrightarrow{b} 1 \quad a \circ b \in A}{x \parallel y \xrightarrow{a \circ b} x'}$$

$$(com_4)\ \frac{x \xrightarrow{a} 1 \quad y \xrightarrow{b} 1 \quad a \circ b \in A}{x \parallel y \xrightarrow{a \circ b} 1}$$

An instance of the above framework is CCS where synchronisation happens between a "sender" and a "receiver" thread on a port 'a': the alphabet is partitioned in a set In of "input ports" and a set $Out = \{\bar{a} \mid a \in I\}$ of "output ports" and $\bar{a} \circ a = \tau = a \circ \bar{a}$ for all $a \in In$

Example

Show that $a\,x + b\,y \;\|\; c\,z \xrightarrow{b} x \;\|\; z$ if $a \circ c = b$, $x \neq 1$, and $y \neq 1$



Exercise 12
Is the inference tree the same if x = 1 or y = 1? Justify your answer.
Suppose that $a \circ b = a, x \neq 1, \text{ and } y \neq 1$. Give an inference tree for each possible transition of $a\,x + b\,y$

Summary

- FM : what for & basic (fundamental questions)
- Brief overview of concurrency
  - Problems
  - Shared-memory vs communication
- Operational semantics
  - Transition Systems
  - Structural operational semantics
  - Reg Exp
  - BPAs
- Bisimulations & bisimilarity

https://martin.kleppmann.com/2025/12/08/ai-formal-verification.html

Prediction: AI will make formal verification go mainstream

Published by Martin Kleppmann on 08 Dec 2025.

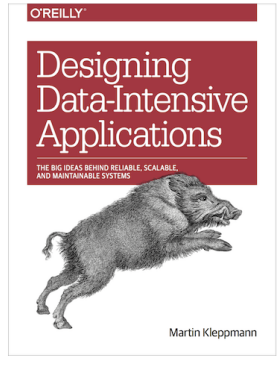
Much has been said about the effects that AI will have on software development, but there is an angle I haven't seen talked about: I believe that AI will bring formal verification, which for decades has been a bit of a fringe pursuit, into the software engineering mainstream.

Proof assistants and proof-oriented programming languages such as Rocq, Isabelle, Lean, F*, and Agda have been around for a long time. They make it possible to write a formal specification that some piece of code is supposed to satisfy, and then mathematically prove that the code *always* satisfies that spec (even on weird edge cases that you didn't think of testing). These tools have been used to develop some large formally verified software systems, such as an operating system kernel, a C compiler, and a cryptographic protocol stack.

At present, formal verification is mostly used by research projects, and it is uncommon for industrial software engineers to use formal methods (even those working on classic high-assurance software such as medical devices and aircraft). The reason is that writing those proofs is both very difficult (requiring PhD-level training) and very laborious.

For example, as of 2009, the formally verified seL4 microkernel consisted of 8,700 lines of C code, but proving it correct required 20 person-years and 200,000 lines of Isabelle code – or 23 lines of proof and half a person-day for every single line of implementation. Moreover, there are maybe a few hundred people in the world (wild guess) who know

My book



My book, *Designing Data-Intensive Applications*, has received thousands of five-star reviews.