→ motivation: $a_0 = 11/2$ $a_1 = 61/11$, $a_{n+2} = 111 - \frac{1130 - 3000/a_n}{a_{n+1}}$
- straightf. Java impl converges at $a_{20} = 99.95\ldots$, while true value $a_{20} = 6$

→ correctness: meeting design requirements
- must show that system can't fail to meet req.
- Dijkstra (1972): testing can show only the presence of bugs, not absence

(→ validation: checking if something satisfies a certain criterion)

→ validation vs. verification:
- val: "are we building the right product?" (doing what the user wants) } val. by: testing actual sys., simulation } both not exhaustive
- ver: "are we building the product right?" (conforming to specs)

→ formal methods: based on math. theories
- f. spec: unambiguous desc. of system or its properties
- f. anal./verif: verifying the sys. satisfies its spec.
- limitations: 1) working w a model, 2) can't disc. fabrication faults (e.g. chips), 3) faulty impl. of tools, 4) state explosion
- no guarantees, but increase of reliability and finding hard to find errors
- formal synthesis vs. verification → w.r.t. impl.
  defining program from specs → ϕ_{spec} → ϕ_{impl}
- verifying specs: deductive: 1) descr. spec in formal log., 2) desc. model in same syst., 3) prove $\phi_I \Rightarrow \phi_S$
  algorithmic: 1) ϕ_{spec}, 2) desc. sys. as interpr. Mimp of given logic, 3) prove that Π_{imp} is a "model" in a logical sense of ϕ_{spec}

→ classification of systems:
- architecture: 1) async. vs. sync HW, 2) analog vs. digital HW, 3) mono vs. multiproc., 4) concurrent vs. seq. SW, 5) conventional vs. RTOS, 6) embedded vs. local vs. distrib systems
- type of interaction: 1) transformational (input $\xrightarrow{\text{trans}}$ outp), should always terminate, 2) interactive, as 1) but not assumed to terminate, 3) reactive - nonterminating, react to events in the env; must be fast — RTOS

→ property taxonomy:
- functional correctness, temporal behavior, safety & liveness prop., persistence, fairness

---

→ SPL, based on Dijkstra guarded commands; subset of Promela

→ logical "language" to express properties of programs: FOPL:
- variables, relation symbols, function symbols, connectives ($\neg, \vee, \wedge, \Rightarrow, \Leftrightarrow$), 2 quantifiers ($\forall, \exists$)
- expressions: variables are atomic exp.
  $f(t_1, \ldots, t_n)$; f function, t_i terms
  when $n = 0$ → f is a constant
- atomic formulae: $T, \bot$; $P(t_1, \ldots, t_n)$ where P is relation
- boolean formulae: all atomic f.
  $\neg\varphi$ $(\varphi \vee \psi)$ $(\varphi \wedge \psi)$ $(\varphi \Rightarrow \psi)$ $(\varphi \Leftrightarrow \psi)$ where φ, ψ bool. f.
- F.O. fm: $(\exists x)\varphi$, $(\forall x)\varphi$ where x var, φ FO f.
  also F.O. fm. with connectives

→ Semantics: model $M = (D, I)$; D = domain (non-empty set), I (interpretation) mapping s.t.
$f^I : D^n \to D$ for every function f of arity n
$P^I \subseteq D^n$ for every relation P of arity n

- we assume an implicit model with D = naturals and sets of nat., it is "obvious" what function and relations should be mapped to

→ state s over $V \in \mathbb{V}$ is a mapping from V to D (i.e. $s : V \to D$)

→ value $s(t')$ of expr: $s(x') = s(x)$
$s([f(t_1, \ldots, t_n)]') = f'(s(t_1'), \ldots, s(t_n'))$

e.g. $s([(2 * x) + z]') = s([2 * x]') +' s(z') = s(2') *' s(x') +' s(z')$
$= 2 * s(x) + s(z) = 2 * 256 + 512 = 1024$

→ free occ. of var. : not within the scope of a quantifier, otherwise - bound
- s_2 is x-variant of s_1 (s_1, s_2 states, x var.) if
  $s_1(y) = s_2(y)$ for all $y \in V \setminus \{x\}$ (i.e. s_1, s_2 disagree only in x)

- let φ F.O. fm, x var, t exp $\varphi[x/t]$: every free occ. of x replaced by t
  [e.g. $\varphi = (\forall x) P(x) \vee P(x)$. $\varphi[x/c] = (\forall x) P(x) \vee P(c)$]

- state formula φ is true/false relative to a model $M = (D, I)$ in a state s:
  $M, s \models T$; $M, s \not\models \bot$; $M, s \models R(t_1, .., t_n)$ iff $(s(t_1), .., s(t_n)) \in R^I$
  $M, s \models \neg\varphi$ iff $M, s \not\models \varphi$ ...
  $M, s \models (\forall x)\varphi$ iff $M, t \models \varphi$ for every t that is an x-variant of s
  $M, s \models (\exists x)\varphi$ iff $M, t \models \varphi$ for some t that is an x-variant of s

- φ is true in model M if $M, s \models \varphi$ for every state s: $M \models \varphi$
- φ is valid if $M \models \varphi$ for every model M : $\models \varphi$

→ proof system: $\dfrac{\varphi_1 \dots \varphi_n}{\psi}$ → premises / → conclusion } inference rules + axioms

→ derivation from a set of fm. S : each fm. is either in S, an axiom, or can be obt. by applying inf. rules to previous fm.
→ proof : derivation from an empty set ; last fm in proof : theorem
- proof system is : sound (every thm. is true), complete (true fm. is a theorem)
- we write $S \vdash_L \varphi$ if there is deriv. from S to φ in logic L ; $\vdash_L \varphi$ when φ is thm.
- there exists a sound and complete proof syst. for FOL

→ SPL
- skip (does nothing), assignment $(x_1, .., x_k) := (t_1, .., t_k)$ (x_i vars, t_i expr.)
- await c waits until c is true ; halt ≡ await ⊥
- request r decrements r as soon as $r > 0$; release r increments r
- critical, noncritical (noncr. need not terminate)
- produce x assigns non-0 to x ; consume x → for prod/cons. programs
- conditional: if c then S_1 else S_2 - c boolean
- concatenation: $S_1 ; S_2 ; \dots ; S_k$ → sequential execution
- selection: S_1 or S_2 or ... or S_k → nondeterministic selection
- while c do S - c boolean
- cooperation: $S_1 \| \dots \| S_k$ → interleaved exec. of processes ; justice ensures that no proc. is ignored forever
- block: [local ; S]
- program: $P :: [\text{declaration} ; [P_1 :: S_1 \| \dots \| P_k :: S_k]]$; P_i - processes
→ declaration: sequence of stmts of the form
  MODE $x_1, .., x_n$: TYPE where φ
  in, local, out ← integer, bool ←
  ↳ $y_1 = t_1 \wedge \dots \wedge y_m = t_m$ where $y_i \in \{x_1 \dots x_n\}$

→ Hoare logic: $\{\varphi\}\ S\ \{\psi\}$ where φ, ψ F.O. fm., S a stmt in SPL
triple : true if whenever S starts executing and φ is true, and if S terminates ψ is true
→ partial correctness: $\varphi \supset \Box(\text{terminates}(S) \supset \psi)$ (safety prop.)
→ total corr. : $\varphi \supset \Diamond(\text{terminates}(S) \wedge \psi)$ (liveness prop., if S assumed to terminate)

→ post locations of stmts!

→ axioms of Hoare logic

- await axiom: $\{\varphi\}$ await c $\{\varphi \wedge c\}$
- assignment axiom: $\{\varphi[x/t]\}\ x := t\ \{\varphi\}$ (ex: $\{1=1\}\ x:=1\ \{x=1\}$)
- concat. rule: $\dfrac{\{\varphi\}\ S_1\ \{\chi\} \quad \{\chi\}\ S_2\ \{\psi\}}{\{\varphi\}\ S_1; S_2\ \{\psi\}}$
- condit. rule: $\dfrac{\{\varphi \wedge c\}\ S_1\ \{\psi\} \quad \{\varphi \wedge \neg c\}\ S_2\ \{\psi\}}{\{\varphi\}\ \text{if } c \text{ then } S_1 \text{ else } S_2\ \{\psi\}}$
- skip axiom: $\{\varphi\}$ skip $\{\varphi\}$
- selection rule: $\dfrac{\{\varphi\}\ S_1\ \{\psi\} \ \dots\ \{\varphi\}\ S_k\ \{\psi\}}{\{\varphi\}\ S_1 \text{ or } \dots \text{ or } S_k\ \{\psi\}}$
- while rule: $\dfrac{\{\varphi \wedge c\}\ S\ \{\varphi\}}{\{\varphi\}\ \text{while } c \text{ do } S\ \{\varphi \wedge \neg c\}}$
- consequence rule: $\dfrac{\varphi' \supset \{\varphi\}\ S\ \{\psi\} \supset \psi'}{\{\varphi'\}\ S\ \{\psi'\}}$ with T a set of FO fm. from which we can derive arith. fms. such as $1+1=2$
  if $T \vdash \varphi' \supset \varphi$ and $T \vdash \psi \supset \psi'$

→ FAIR TRANS. SYST.

- each stmt. in prog. has a label; equiv. relation btw labels as.
  $\ell \sim_\ell \ell_1$ for $\ell: [\ell_1: S_1; \dots; \ell_k: S_k]$
  $\ell: [\text{local}; \ell_1: S_1]$
  $\ell \sim_\ell \ell_1 \sim_\ell \ell_2 \sim_\ell \dots \sim_\ell \ell_k$ for $\ell: [\ell_1: S_1 \text{ or } \ell_2: S_2 \text{ or } \dots \text{ or } \ell_k: S_k]$
  - no equiv. relation for parallel composition
- $[\ell] = \{\ell' : \ell' \sim_\ell \ell\}$ → the equiv class of labels wrt $\sim_\ell$
  ↳ location corresp. to label ℓ

→ fair. trans. syst is $P = (V, \Theta, \mathcal{T}, \mathcal{J}, \mathcal{C})$
$V \subseteq \mathcal{V}$: set of syst. var.; $\Theta \subseteq \mathcal{L}$ initial cond, $\mathcal{T}$ finite set of transitions
$\mathcal{J} \subseteq \mathcal{T}$ set of just (weakly fair), $\mathcal{C} \subseteq \mathcal{T}$ set of compassionate (strongly fair) trans.

- $V = Y \cup \{\Pi\}$ where Y are program var. ($y \in Y$ ranges over its declared domain), Π is a control var. (ranges over sets of locations of the program)
- Θ is of the form $\Pi = \{[\ell_1], \dots, [\ell_k]\} \wedge \varphi$ for $P :: [\text{decl}; P_1 :: [\ell_1: S_1; \hat{\ell}_1:] \| \dots \| P_k :: [\ell_k: S_k; \hat{\ell}_k]]$
  φ is the conj. of where constraints from the declaration
- let Σ be the set of all states. transition is a function $\tau : \Sigma \to \mathcal{P}(\Sigma)$
  $\tau(s) = \{s_1, \dots, s_n\}$ → τ-successors of s
  transition τ is enabled on state s if $\tau(s) \neq \emptyset$
- primed state: $s'(x) = s(x')$. assume that every $x \in V$ has primed version $x' \in V$. if τ is enabled on s we assume $s' \in \tau(s)$. we regard x' as referring to x in τ-succ. s' of s
- trans. relation associated to each trans τ is ρ_τ of the form $\boxed{\varphi_\tau \wedge \bigwedge_{x \in V} (x' = t_x)}$
  where t_x refers only to unprimed vars.
- τ is enabled on s iff $s \models \rho_\tau$. ρ_τ is the enabling condition
  + uniqueness: s' maps every $x \in V$ to a unig. expr. $\bigwedge_{x \in V} (x' = t_x)$
  ⇒ any state has at most one τ-succ. s': $\boxed{\text{if } s' \in \tau(s) \Rightarrow \tau(s) = \{s'\}}$

→ move: $L, \hat{L}$ sets of locations. $\text{move}(L, \hat{L}) = L \subseteq \Pi \wedge \Pi' = (\Pi \setminus L) \cup \hat{L}$
- simultaneous move of control from L to $\hat{L}$

→ preservation of vars: $\text{pres}(U) = \bigwedge_{x \in U} (x' = x)$ for $U \subseteq V$

→ idle trans: to ensure that every state has at least 1 trans. enabled on it.
- always enabled; $\tau_I(s) = \{s\}$, $\rho_I = \text{pres}(V)$

- TODO: SPL semantics
- upr.: P: skip; $\hat{\ell}$: $\rho_I = move(\ell, \hat{\ell}) \wedge pres(\gamma)$
  ℓ: await c; $\hat{\ell}$: $\rho_I = move(\ell, \hat{\ell}) \wedge c \wedge pres(\gamma)$
  ℓ: $(x_1, .., x_k) := (t_1 .. t_k)$; $\hat{\ell}$: $\rho_I = move(\ell, \hat{\ell}) \wedge \bigwedge x_i' = t_i \wedge pres(\gamma \setminus \{x_1 .. x_k\})$
  ℓ: request r; $\hat{\ell}$: $\rho_I = move(\ell, \hat{\ell}) \wedge r > 0 \wedge r' = r - 1 \wedge pres(\gamma \setminus \{r\})$

→ the following are <u>not</u> directly assoc. with trans due to label equiv.:
- concat: ℓ: [[ℓ_1: s_1; $\hat{\ell}_1$]; ..]; $\hat{\ell}$: → $\tau_\ell = \tau_{\ell_1}$
- block: ℓ: [decl; [ℓ_1: s_1;]]; $\hat{\ell}$: → $\tau_\ell = \tau_{\ell_1}$
- selection: ℓ: [[ℓ_1: s_1; $\hat{\ell}_1$] or ...]; $\hat{\ell}$: → $\tau_\ell \in \{\tau_{\ell_1}, \tau_{\ell_2} .. \tau_{\ell_k}\}$
- <u>cooperat</u>: ℓ: [[ℓ_1: s_1; $\hat{\ell}_1$] || .. [ℓ_k: s_k; $\hat{\ell}_k$]]; $\hat{\ell}$:
  - two trans: entry: $\rho_\ell^E = move(\{\ell\}, \{\ell_1, .., \ell_k\}) \wedge pres(\gamma)$
    exit: $\rho_\ell^x = move(\{\hat{\ell}_1, .., \hat{\ell}_k\}, \{\hat{\ell}\}) \wedge pres(\gamma)$
  - exit can be taken only when all s_i have terminated

→ <u>weak fairness</u>: any <u>just</u> trans. that is <u>continually</u> enabled, must eventually be taken
(only idle and noncritical are <u>not</u> just - they are allowed not to terminate)
→ <u>strong fairness</u>: any <u>compassionate</u> trans. that is enabled inf. often, must be taken inf. oft.
(only request + communication stmts. are compassionate)
- we have $C \subset J$ (all compass trans. are just)

→ let $P = (V, \Theta, T, J, C)$. inf. sequence of states $\sigma = s_0, s_1, s_2, ...$ is a <u>P-computation</u> if:
1. initiality: $s_0 \models \Theta$
2. consecution: there always exists trans. τ s.t. $s_{j+1} \in \tau(s_j)$ $\forall j \geq 0$
3. justice: any trans. in J that is cont. en. must be eventually taken:
   $(\forall \tau \in J)(\forall j \geq 0)((\forall k \geq j) E(\tau, k) \supset (\exists k \geq j) T(\tau, k))$ E=enabled, T=taken
4. compass: any trans. in C that is en. inf. often, must be taken inf. often
   $(\forall \tau \in C)[(\forall j \geq 0)(\exists k \geq j) E(\tau, k) \supset (\forall j \geq 0)(\exists k \geq j) T(\tau, k)]$

→ TEMPORAL LOGIC

- modeling time: linear vs. branching; time instances vs. intervals; discrete vs. cont.; past & future vs. future only
- FOL: express properties of states (e.g. $\langle x:21, y:49 \rangle \models x < y$)
- computation = a <u>sequence of states</u> - temporal logic: we deal only with the <u>future</u>
- we extend FO language $\mathcal{L}$ to temporal language $\mathcal{L}_T$ by adding temporal operators:
  $\Box\varphi$: φ will always hold $\Diamond\varphi$: φ will eventually hold $\bigcirc\varphi$: φ holds at the next point
  $\varphi \mathcal{U} \psi$: ψ evt. holds, <u>until</u> that φ holds $\varphi \mathcal{W} \psi$: φ holds until ψ (weak until: ψ might never hold)
  $\varphi \mathcal{R} \psi$: ψ holds until (incl.) point where φ holds

→ <u>LTL fm</u>: FO fm. is also LTL fm
- if φ, ψ are LTL fm, so are $\Box\varphi, \Diamond\varphi, \bigcirc\varphi, \neg\varphi, \varphi \mathcal{U} \psi, \varphi \mathcal{R} \psi, \varphi \mathcal{W} \psi, \varphi \vee \psi, \varphi \wedge \psi, \varphi \supset \psi, \varphi \equiv \psi$

→ <u>path</u> is an infinite sequence $\sigma = s_0 s_1 s_2 ...$
- σ^k denotes the <u>path</u> $s_k s_{k+1} ...$
- σ_k denotes the <u>state</u> s_k

} all computations are paths, but not vice-versa (because of justice/compassion)

→ LTL fm. is true/false <u>relative to path</u> σ:

$\sigma \models \varphi$ iff $\sigma_0 \models \varphi$ if $\varphi \in \mathcal{L}$ (ie. FO)
$\sigma \models \neg\varphi$ iff $\sigma \not\models \varphi$
$\sigma \models \varphi \vee \psi$ iff $\sigma \models \varphi$ or $\sigma \models \psi$

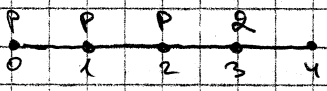
$\sigma \models \Box\varphi$ iff $\sigma^k \models \varphi$ $\forall k \geq 0$
$\sigma \models \Diamond\varphi$ iff $\exists k \geq 0$ s.t. $\sigma^k \models \varphi$
$\sigma \models \bigcirc\varphi$ iff $\sigma^1 \models \varphi$

$\sigma \models \varphi \mathcal{U} \psi$ iff $\sigma^k \models \psi$ for some $k \geq 0$ and $\sigma^i \models \varphi$ $\forall i$ $0 \leq i < k$
$\sigma \models \varphi \mathcal{R} \psi$ iff $\forall j \geq 0$, if $\sigma^i \not\models \varphi$ $\forall i < j$ then $\sigma^j \models \psi$
$\sigma \models \varphi \mathcal{W} \psi$ iff $\sigma \models \varphi \mathcal{U} \psi$ or $\sigma \models \Box\varphi$

→ if $\sigma \models \varphi$ for all paths σ, we say that φ is temporally valid: $\models \varphi$
→ if $\models \varphi \equiv \psi$ we say that φ and ψ are equivalent and write $\varphi \sim \psi$

$$\boxed{\begin{array}{l}\Box(\varphi \wedge \psi) \sim \Box\varphi \wedge \Box\psi \\ \Diamond(\varphi \vee \psi) \sim \Diamond\varphi \vee \Diamond\psi\end{array}}$$ ⇒ distribution of $\Box, \Diamond$ over $\wedge, \vee$

→ $\sigma \models p \,U\, q$



(there need not be a p, i.e. q at 0 is also accepted)
seq. of p is finite

$\sigma \models p \,R\, q$

sequence of q may be infinite

→ formulas with past operators can always be transformed into future formulas

→ duals: for a binary bool. connective $\circ$, its dual $\bullet$ is if $\boxed{\neg(\varphi \circ \psi) \equiv (\neg\varphi \bullet \neg\psi)}$
- unary: $\boxed{\neg\circ\varphi \equiv \bullet\neg\varphi}$
- $\vee, \wedge$ are duals, $\neg$ is its own dual

- set of connectives is complete if every other conn. can be defined in terms of them
e.g. $\{\wedge, \neg\}$ is complete

- temporal dualities: $\boxed{\begin{array}{ll}\neg\Box\varphi \sim \Diamond\neg\varphi & \neg(\varphi U \psi) \sim (\neg\varphi)\, R\, (\neg\psi) \quad \neg\bigcirc\varphi \sim \bigcirc\neg\varphi \\ \neg\Diamond\varphi \sim \Box\neg\varphi & \neg(\varphi R \psi) \sim (\neg\varphi)\, U\, (\neg\psi)\end{array}}$

→ $\{U, \bigcirc\}$ is complete:

$$\boxed{\begin{array}{ll}\Diamond\varphi \sim \top U \varphi & \varphi R \psi \sim \neg(\neg\varphi\, U\, \neg\psi) \\ \Box\varphi \sim \bot R \varphi & \varphi W \psi \sim \Box\varphi \vee \varphi U \psi\end{array}}$$

→ properties:
- safety: $\Box\varphi$, conditional safety: $\varphi \supset \Box\psi$ (φ, ψ: FO) + invariance of property (e.g. mutual excl.: $\Box\neg(C_1 \wedge C_2)$ – not both P_1 and P_2 in the crit. sect.)
- liveness: $\Diamond\varphi$, conditional liveness: $\varphi \supset \Diamond\psi$ – guarantee that some event happens
- partial correctness: $\Box(\text{terminates}(P) \supset \psi)$
- full p.c. $\{\varphi\}P\{\psi\}$: $\varphi \supset \Box(\text{terminates}(P) \supset \psi)$
- total correctness: $\Diamond(\text{terminates}(P) \wedge \psi)$
- full t.corr. : $\varphi \supset \Diamond(\text{terminates}(P) \wedge \psi)$

(φ is the precondition, ψ the postcondition of the program P)

- partial and t.c. are duals: let
$PC(\psi) = \Box(\text{terminates} \supset \psi)$, $TC(\psi) = \Diamond(\text{terminates} \wedge \psi)$
$\neg PC(\psi) \sim TC(\neg\psi)$, $\neg TC(\psi) \sim PC(\neg\psi)$
- obligation: $\Box\varphi \vee \Diamond\psi$, or, equivalently, $\boxed{\Diamond\chi \supset \Diamond\psi}$
- every safety and liveness f. is also obligation f.
$\Box\varphi \sim \Box\varphi \vee \Diamond\bot$ and $\not\models \Box\top$
$\Diamond\varphi \sim \Box\bot \vee \Diamond\varphi$ $\not\models \Box\bot$
- recurrence: $\boxed{\Box\Diamond\varphi}$: infinitely many positions satisfy φ
- weak f. can be specified as r.: $\Box\Diamond(E(\Sigma) \supset T(\Sigma))$ or $\boxed{\Box(\Box E(\Sigma) \supset \Diamond T(\Sigma))}$
- persistence: $\Diamond\Box\varphi$: all but finitely many positions satisfy φ
- dual with recurrence: $\neg(\Box\Diamond\varphi) \sim (\Diamond\Box\neg\varphi)$, $\neg(\Diamond\Box\psi) \sim (\Box\Diamond\neg\psi)$
- reactivity: $\Box\Diamond\varphi \vee \Diamond\Box\psi$, or, $\boxed{\Box\Diamond\chi \supset \Box\Diamond\psi}$ (if there are inf. many χ, there must be inf. many ψ)
- Strong fairness is a form of reactivity: $\boxed{\Box\Diamond E(\Sigma) \supset \Box\Diamond T(\Sigma)}$

⇒ PROOF METHODS

- state s is P-accessible if it appears in some P-computation σ
- FO f. is P-state valid $P \models \varphi$ iff $s \models \varphi$ for all P-acc. states s
- LTL f. is P-valid $P \models \varphi$ iff $\sigma \models \varphi$ for all P-computations σ
- verific. cond of φ, ψ relative to trans. τ is given by $(\beta_\tau \wedge \varphi) \supset \psi'$
  - abbreviate to $\{\varphi\}\tau\{\psi\}$: τ leads from φ to ψ if it holds
- invariance rule: $\dfrac{\{\varphi\}\mathcal{T}\{\varphi\}}{\varphi \supset \Box\varphi}$ ($\mathcal{T}$ all trans. of process) — if every transition preserves φ, then φ must be invariant for the computation

⇒ initialized INV: $(1)\ \Theta \supset \varphi,\ (2)\ \{\varphi\}\mathcal{T}\{\varphi\} \Rightarrow \Box\varphi$ (unconditional inv. of φ)

⇒ accessibility rule: $\dfrac{(1)\ \Box\varphi \quad (2)\ \{\varphi \wedge \chi\}\tau\{\varphi \supset \psi\}}{\{\chi\}\tau\{\psi\}}$

- s: accessible χ-state; it is also $(\varphi \wedge \chi)$ state by (1)
- any τ-succ. is by (2) a $(\varphi \supset \psi)$ state, and by (1) it also must be a ψ state

⇒ $\vdash$ LIV is sound: if premises are true, so is the conclusion
- proof by induction (1) $P \models \Theta \supset \varphi$ (2) $P \models (\beta_\tau \wedge \varphi) \supset \psi'\ \forall \tau \Rightarrow P \models \Box\varphi$

⇒ J-LIVE: just liveness rule:

$$\frac{(1)\ \{\varphi\}\mathcal{T}\{\varphi \vee \psi\} \quad (2)\ \{\varphi\}\mathcal{T}_k\{\psi\} \quad (3)\ \varphi \supset (\psi \vee E(\mathcal{T}_k))}{\varphi \supset (\varphi \mathcal{U} \psi)}$$

- only way out of φ is ψ-state
- if not $\varphi \supset \psi$, a "helpful" proc. P_k is continually enabled
- by justice P_k must be taken at some point (thus leading to ψ state). conclusion implies $\varphi \supset \Diamond\psi$

- is sound wrt just transitions: proof by contrad. (assume that φ never happens)

- J-chain: generalization of J-live: if given a sequence of assertions $\varphi_0 .. \varphi_r$ s.t. the init. state satisfies φ_r and goal is φ_0, repeatedly apply J-live to show $\varphi_i \supset \Diamond\varphi_{i-1}$ which leads to $\varphi_r \supset \Diamond\varphi_0$

- well-founded strict partial order $(A, \prec)$ if it does not contain infinite desc. chain $a_0 \succ a_1 \succ \ldots$ ($(\mathbb{N}, <)$ is well-founded, $(\mathbb{Z}, <)$ is not)

- J-WELL: want to prove $P \models \Diamond\psi$. given w-f set $(A, \prec)$ we must find a parameterized assertion $\varphi(n)$ relating state s to a parameter $n \in A$
  - if $s \models \varphi_i$ then either $s' \models \psi$ or $s' \models \varphi_j$ with $j < i$. any comput. which fails to reach ψ generates an infinite desc. chain of A-elements + impossible
  - with multiple processes: at least one proc. needs to decrease n at each step
  ⇒ helpfulness function identifies the "helpful" process (also in J-chain)

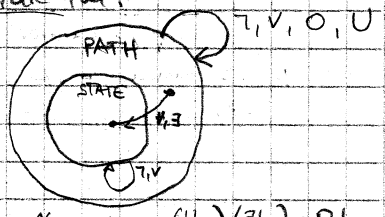
⇒ LIVE: takes into account compassion (w/ semaphore shuffs.)

$$\frac{(1)\ \{\varphi\}\mathcal{T}\{\varphi \vee \psi\} \quad (2)\ \{\varphi\}\mathcal{T}_k\{\psi\} \quad (3)\ \varphi \supset \Diamond(\psi \vee E(\mathcal{T}_k))}{\varphi \supset \varphi \mathcal{U} \psi}$$

- sound w.r.t. compassionate trans.
- proof similar to J-live, ignoring process k (when P_k is enabled we are already at goal state satisfying $\psi \vee E(\mathcal{T}_k)$)

⇒ CTL

⇒ CTL* (Φ: set of prop vars (countable) $p_1, p_2, \dots$)
- state fm: every p_i is a state fm; $\neg\varphi$, $\varphi \vee \psi$ where φ, ψ state fm.
  $\forall\varphi$, $\exists\varphi$ where φ is a path fm.
- path fm: state fm is also a path fm
  $\neg\varphi$, $\varphi \vee \psi$, $O\varphi$, $\varphi U \psi$ where φ, ψ - path fm.



- duality of $\forall, \exists$: $\forall\varphi \sim \neg E \neg\varphi$; $\exists\varphi \sim \neg A \neg\varphi$
- relation $R \subseteq W \times W$ is total if every elt. in W is R-related to some elt in W: $(\forall a)(\exists b) aRb$
- Kripke frame: (W, R)
  - W nonempty set of worlds, R total accessibility relation btw worlds
- Kripke model: $(W, R, V) = M$ where (W, R) is K. frame and:
  - $V: W \to \mathcal{P}(\Phi)$ labels each world with a set of prop vars.
  - path in M: infinite seq $\sigma = w_0 w_1 w_2 \dots$ s.t. $\forall i$ $w_i R w_{i+1}$ ("infinite branch in a tree corresp to K. frame)

⇒ let $\varphi, \varphi_1, \varphi_2$ be state fm., ψ, ψ_1, ψ_2 path fm.

- true in a world w:

$M, w \models p_i$	iff $p_i \in V(w)$
$M, w \models \neg\varphi$	$M, w \not\models \varphi$
$M, w \models \varphi_1 \vee \varphi_2$	$M, w \models \varphi_1$ or $M, w \models \varphi_2$
$M, w \models \forall\psi$	$M, \sigma \models \psi$ for every path σ
$M, w \models \exists\psi$	$M, \sigma \models \psi$, starting in w (some path)

- true on a path σ:

$M, \sigma \models \varphi$	iff $M, \sigma_0 \models \varphi$
$M, \sigma \models \neg\psi$	$M, \sigma_0 \not\models \psi$
$M, \sigma \models \psi_1 \vee \psi_2$	$M, \sigma \models \psi_1$ or $M, \sigma \models \psi_2$
$M, \sigma \models O\psi$	$M, \sigma^1 \models \psi$
$M, \sigma \models \psi_1 U \psi_2$	$M, \sigma^k \models \psi_2$ for some $k \geq 0$, $M, \sigma^i \models \psi_1$ $\forall\, 0 \leq i < k$

⇒ CTL: sublogic of CTL* - can't express e.g. fairness
- state fm: every p_i, $\neg\varphi$, $\varphi \vee \psi$ (φ, ψ state fm) $\forall\varphi$, $\exists\psi$ (φ, ψ path fm)
- path fm: $O\varphi$, $\varphi U \psi$ with φ, ψ state fm

⇒ LTL: sublogic of CTL* - can't express e.g. "reset" properties
- if φ is a path fm, $\forall\varphi$ is LTL fm.; $\neg\varphi$, $\varphi \vee \psi$, $O\varphi$, $\varphi U \psi$ with φ, ψ path fm.
- every p_i is a state and path fm.

⇒ Modal logic: originally. $\Box$ = necessarily; $\Diamond$ = possibly
- φ is true in world w in model M as: (standard definitions for Bool oper.)
  $M, w \models \Box\varphi$ iff $M, w' \models \varphi$, $\forall w'$ s.t wRw'
  $M, w \models \Diamond\varphi$ iff $M, w' \models \varphi$, for some w' s.t wRw'
  - no restrictions on R (not even totality)
- $R \subseteq W \times W$ is Euclidean if $\forall abc\ (aRb \wedge aRc \supset bRc)$
- every normal modal logic has inf. rules:

$$\frac{\varphi \text{ is tautology}}{\varphi} \qquad \frac{\varphi, \varphi \supset \psi}{\psi} \qquad \frac{\varphi}{\Box\varphi}$$

- every normal logic has the "K axiom": $\Box(\varphi \supset \psi) \supset (\Box\varphi \supset \Box\psi)$
- LTL holds on the model $(\mathbb{N}, \leq)$

→ Processes and synchr.

- fine-grained atomicity – implemented directly by HW where proc. is executing
- critical reference: ref. to a var. changed by another process, assignment $x = e$ is at-most-once if either (1) e contains at most 1 crit. ref. and x is not referenced by another proc., or (2) e contains no crit. refs → then x may be read by other proc.
- if assignment satisfies at-most-once prop., it appears to be atomic
- paral. exec. rule: $\frac{\{P_i\}\ S_i\ \{Q_i\}}{\{P_1 \wedge \dots \wedge P_n\}\ co\ S_1; \dots; S_n;\ oc\ \{Q_1 \wedge \dots \wedge Q_n\}}$ are interference free

- a proc interferes with another if executes an assignment that invalidates an assertion in another process

⇒ critical assertion: pre- or post-cond not within await stmt

⇒ noninterference: let a be assign. in one process and $pre(a)$ its precond, C a crit. assertion in another proc. a does not interfere with C if $\{C \wedge pre(a)\}\ a\ \{C\}$

⇒ avoiding interference: 1) disjoint variables (make set of one proc. disjoint from ref. set of another, vice-versa – they can't interfere), 2) weakened assertions (e.g. used for scheduling problems where "best" is time-dependent property)
3) global invariants (if assertion I references global vars. put all crit ass. in the form $I \wedge L \Rightarrow$ ref either local vars of a process or a global that only P_i assigns to)
4) synchronization (e.g. mutual exclusion to "hide" critical ass.)

→ application of model checking: 1) choose properties critical to the design, 2) build verification model (as small as possible, but captures relevant properties), 3) select verif. method, 4) refine model and correctness requirements

→ causes of comb. complexity in practice: # and size of buff. chans, # of async processes

→ system: $\mathcal{L}(S)$, property: $\mathcal{L}(P)$; prove that $\mathcal{L}(S) \subseteq \mathcal{L}(P)$ (everything possible is valid)
→ if $\mathcal{L}(S) \cap \mathcal{L}(P) \neq \emptyset$: no accepted word in S disallowed by P

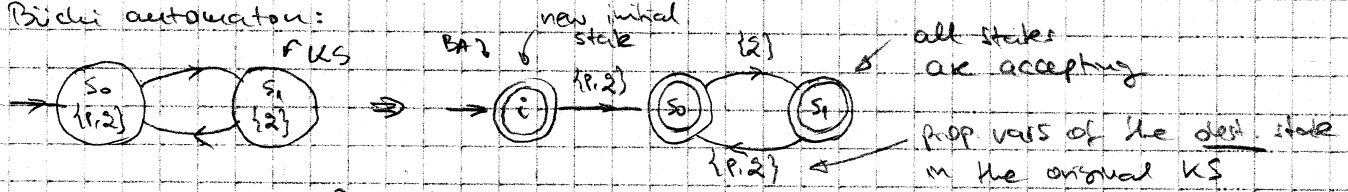
- types of errors: deadlocks, livelocks, starvation, race cond. ...
- correctness w.r.t. specification; abstraction; time and prob. - not considered in LTC
- dealing with infinite executions

→ FSM is a tuple (S, s_0, L, F, T) → $T \subseteq S \times L \times S$ transition relation
↳ finite set of labels (symbols)

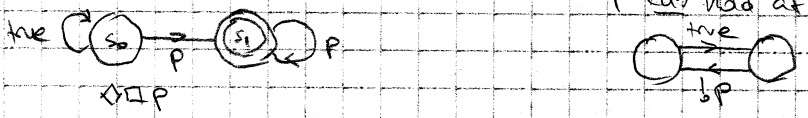
- deterministic or nondeterministic (det. dst state uniquely determined by src state and label)
- run of automaton: $\sigma = \{(s_0, l_0, s_1), (s_1, l_1, s_2), \dots\}$ s.t. $\forall i \geq 0$ $(s_i, l_i, s_{i+1}) \in T$
  - corresp. to a state sequence and a word in L
- accepting run: ~~in final transition~~; the final state is $\in F$
- language of aut: set of ~~words~~ words in L that corr. to accepting runs
- infinite run: or ω-run; Büchi acceptance: at least one state in F is visited inf. oft.
- generalized B. aut: $\mathcal{F} \subseteq \mathcal{P}(S)$: multiple accepting sets; run is accepting if for each $f_i \in \mathcal{F}$ we have $\inf(\sigma) \cap f_i \neq \emptyset$ - not more expressive than B. aut
- acceptance for finite runs: stutter extension: $\sigma(s_n, \varepsilon, s_n)^\omega$ (σ: finite)
  - ε: nil symbol; finite run is extended into inf. run by stuttering the final state on ε

→ Kripke structure: $M = (W, R, W_0, V)$ (W_0: ↳ set of starting states) $V: W \to \mathcal{P}(AP)$: labeling state w/ set of prop. vars. $R \subseteq W \times W$: total accessibility relation

- path: inf. sequence $\sigma = w_0 w_1 \dots$ of worlds s.t. $\forall i \geq 0$ $w_i R w_{i+1}$ (infinite branch in a tree corresp. to the unwind of K. struct.)
- to Büchi automaton:



→ for any LTL fm. exists a B. aut:



"p can hold after even # of steps, but never after odd"
$\exists t.\ t \wedge \Box(t \to \bigcirc\neg t) \wedge \Box(\neg t \to \bigcirc t)$
$\Box(\neg t \to \neg p)$

- not pure LTL (because of ∃) ⇒ ω-regular prop.

→ system: Büchi aut. A: async prod. of automata A_i corresp to processes $A = \prod A_i$
→ property φ: LTL fm. translated into B. aut $\mathcal{B}$, and check $\mathcal{L}(A) \cap \overline{\mathcal{L}(\mathcal{B})} = \emptyset$
- complementation is difficult ⇒ translate negation of prop. into $\overline{\mathcal{B}}$ since $\overline{\mathcal{L}(\mathcal{B})} = \mathcal{L}(\overline{\mathcal{B}})$
- make sync. product $\overline{\mathcal{B}} \otimes A$ and check that $\mathcal{L}(A) \cap \mathcal{L}(\overline{\mathcal{B}}) = \emptyset$ - by looking for accept. cycles

---

→ LTL into aut: rewriting identities:
to rewrite: $\Diamond\psi \equiv \top \mathcal{U} \psi$
$\Box\psi \equiv \bot \mathcal{R} \psi$

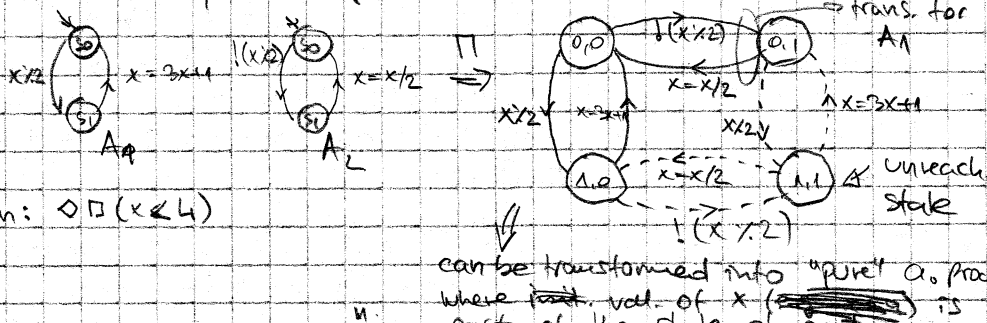
2. negat. normal form:
$\neg\neg\psi \equiv \psi$
$\neg(\varphi \mathcal{U} \psi) \equiv (\neg\varphi) \mathcal{R} (\neg\psi)$
$\neg(\varphi \mathcal{R} \psi) \equiv (\neg\varphi) \mathcal{U} (\neg\psi)$

3. recurrences:
$\varphi \mathcal{U} \psi \equiv \psi \vee (\varphi \wedge \bigcirc(\varphi \mathcal{U} \psi))$
$\varphi \mathcal{R} \psi \equiv \psi \wedge (\varphi \vee \bigcirc(\varphi \mathcal{R} \psi))$

- must be in negational normal form: ¬ only before prop. letters

→ async. prod. of $A_1 .. A_n$ = $A = (S, s_0, L, T, F)$ with $A = \Pi A_i$

- $A.S = \times A_i.S$ (cart. prod.), $A.s_0 = (A_1.s_0, .., A_n.s_0)$, $A.L = \cup A_i.L$
- $(A_1.s, .., A_n.s) \in A.F$ if some $A_i.s \in A_i.F$
- $A.T$: $((x_1, .., x_n), l, (y_1, .., y_n)) \in T$ if $\exists i$ s.t. $(x_i, l, y_i) \in A_i.T$ and $x_j = y_j$ for $i \neq j$
  - transitions of component aut. s.t. only one component can execute at a time
  - interleaving semantics

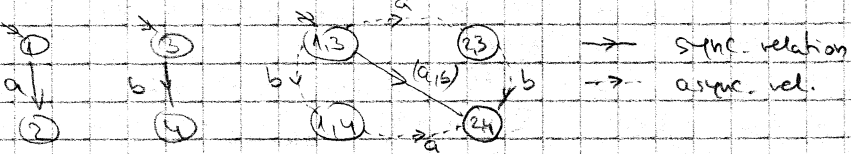


- property: $\Box\Diamond(x \geq 4)$; negation: $\Diamond\Box(x < 4)$

⇓ can be transformed into "pure" a. prod. where ~~not~~ val. of x (~~redacted~~) is part of the state of aut.

→ sync. prod. $P \otimes B = A = (S, s_0, L, T, F)$

- $A.S = P'.S \times B.S$; P' is the stutter closure of P: self loop labeled ε attached to every state in P w/o outgoing trans. in $P.T$
- $A.s_0 = (P.s_0, B.s_0)$, $A.L = \{(l_1, l_2) : l_1 \in P'.L, l_2 \in B.L\}$, $A.T = \{(t_1, t_2) : t_1 \in P'.T, t_2 \in B.T\}$
  $A.F = \{(s_1, s_2) : s_1 \in P.F, s_2 \in B.F\}$?
- trans correspond to joint trans. of component aut.
- in general $P \otimes B \neq B \otimes P$, but in spin this holds



→ PROMELA = process meta-language

- only finite-state models, boundedness guarantees decidability (inf. executions still allowed)
- processes can synchr: 1) global vars, 2) message passing ; max 255 procs
- parameter passing by value (no pointers, floating-point, ...); only global and proc-local scope
- basic data types: bit, bool, byte, chan, mtype, pid, short, int, unsigned
- record structures: typedef Field { short f = 3; byte g } ; 1-dim arrays
- message chans: asynch., synch (rendez-vous)

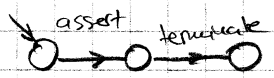
  [chan qname = [8] of {short, chan, record}]

  ch name may be local or global, but chan itself is always global (and can be sent through other chans)
  - sending: qname ! exp1, exp2, exp3 - executable only if chan not full
  - retrieving: qname ? var1, var2, var3 - — " — not empty
  - can use constants instead of vars to match patterns (executable only if match)
  - qname ? eval(var1), var2 - exec only if msg matches the curr. val. of var1
  - send/rcv - not expressions! - I/O stmts.
    (a>b && qname ? msg0) → not valid
    (a>b && qname ? [msg0]) → valid - true if executable, but actual rcv not exec'd
  - sorted send: q!!n,m,p - adds in numerical order
  - random rcv: q??n,m,p - any message (need not be the 1st)
  - brackets: q?[n,m,p] - side-effect free Bool. expr.
  - poll: q?<n,m,p> - receive w/o changing contents of q

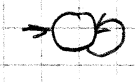
- channel capacity 0: rendezvous

CLAIMS

- behavior spec: what is possible (no probabilities) ↳ w/o assump. about relative speed of proc., exec. time of instr., prob. of events
- logical correctness props: what is valid
- ⇒ safety: something bad never happens
- ⇒ liveness: something good eventually happens
- ⇒ state prop: system invar., local process assertions, end-state labels
  ↳ every reach. state of syst. ↳ specific states ↳ proper termination of proc.
- ⇒ path prop: accept-states labels, progress-state lab., never claims, trace assert.
  ↳ prop. of msg. chan
- ⇒ assertions: assert(expr)
- ⇒ invariant: active proctype invariant() { assert.. }



↳ state spc. multiply by 3

{do: assert() od} →  → more trans., but not more states

- ⇒ end states: to recognize reachable deadlocks
  (vs. waiting loops: labels starting with "end"
  ↳ diff: only valid end state — end of code

- ⇒ progress state label: starting with "progress": every (not-infinite) execution cycle permitted by the model passes through at least one progr. label
  (otherwise there exist non-pr. cycles) [w/o pr.lab every cycle is non-pr.]

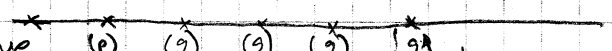
- ⇒ accept state label: starting with "accept": ✓ all cycles that do pass thr. at least one label. There should not be an execution that can pass through acc label infinitely often. (used with never claims)

- ⇒ fairness, finite progr. assumption: if a process can execute a stmt, it will eventually proceed with that execution
  - weak f.: executable infinitely long ⇒ eventually executed → O(n) penalty (in # of
  - strong f.: ~~executable~~ often ⇒ " → O(n²) penalty / act. proc.
  - can also be applied to nondet. selection a) within, b) between processes
  - Spin: only process sched. (but +LTL - any kind of f.)

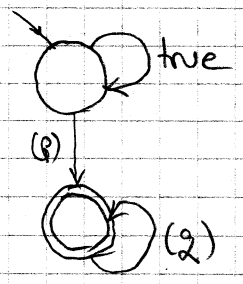
- ⇒ never claim: observer process executes synch. with the system
  - e.g.: truth of p is always followed by the truth of !q within a finite # of steps

p̄q̄ p q̄ p q̄ p q̄ p q̄ p q̄ p q̄ p q̄

negation: truth of p ~~never~~ followed within finite # of steps by truth of !q




```
never {
    do ::true
       ::(p) -> break
    od;
accept: do ::(q)
        od
}
```

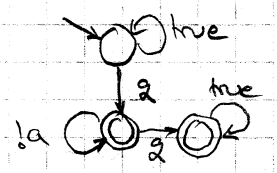


→ formalizing execution:
- seq. of 1) states, 2) events (state trans.), 3) propositions on states (state properties)
  ↳ SPIN

- ex: p ⇒ !q – monitor process:
```
active proctype invariant() {
  do ::assert(!p || !q)
  od }
```
- ex: every state where p holds is followed by a state where !q holds
```
act proct inv() {
  (p) →
  accept: do :: (q) od }
```
does not work!

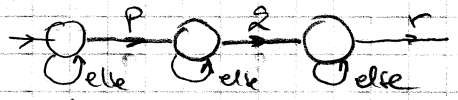
– invariants not possible to define properties about every single execution step
↳ never claim: checks system prop. just before and after each stmt, no matter in which process – checker executes synchronously w/ system
↳ should not contribute to behaviour (
- no side-effects, cannot block (blocking: pattern can't be matched; gives up on current exec. seq. and tries another)
- pausing: explicit self-loop on true
- complete match: reached end or closed accept cycle
- defined globally; can't refer to events, only to props. of states
- send/recv events: only trace assertions
- more expressive than LTL (equiv. to ω-word and Büchi aut.)

- ex 1: "!a always followed by a q before next a"




```
never {
  do :: true
     :: a -> break
  od
accept: do :: !a
        :: q -> break
  } od
```

→ ex 2: no exec. where first p, then q, then r
never {p; q; r} → wrong – only first 3 steps




```
never {
  do :: p -> break  do :: q -> break  do :: r -> break
     :: else         :: else          :: else
  od;               od;              od;
}
```

→ remote reference: proctype[pidnr]@label → true if currently at label
→ checking for termination: proctype runner() {... L: (false) }
- if (runner@L) → terminated (once L reached, never proceeds beyond)
- or check _nr_pr global (# of running processes)

→ trace assertion: trace { do :: p!a; q?b od; } – every send of msg a to chan p is followed by recv of msg b from q
- can use only send/rcv stmts
- if must include all possible sends for a named chan. (similarly, rcvs.)

→ notrace: particular sequence is impossible
```
notrace {
  if :: p!a; q?b
     :: q?b; p!a
  fi
}
```
fully matched on closing curly brace

→ assertion: impossible for exp. to be false when ass. reached
→ end-state l.: imp. for sys to term. w/o all act proc. terminated or stopped at end-state
→ progr. state l.: imp. to execute w/o passing it. often thr. at least 1 pState
→ accept state l.: imp to execute while passing —"— —"— acc state
→ never claim: imp. to exhibit behaviour that completely matches the claim
- trace: imp. to exhibit beh. that does not completely match pattern

→ search for counter-examples!

⇒ stmt: executable or blocked; expr. are exec if evaluate to true or nonzero value; expr. must be w/o sideeffects (except run)
- run operator returning PID or 0 (max 255 processes)
- always exec: x++, x--, x=x+1, x=run P(), print, assert (b=c++ not valid exp - side eff!)
- exec when true: (x), (1), run P(), skip, true, else, timeout
- send exec when chan non-full; recv exec when non-empty and cond met
- if: non-det selection: if :: x>=y → m=x fi; or: if :: n=0 :: n=1 fi ← pick a number
  (x = guard, m=x = action, n=0 = no guard (true))
  - need explicit else (w/o else would block until guard becomes true): if :: x<=y → x=y-x :: else fi;
  - else is a var. that becomes true when no other stmt. in the same process is executable

→ timeout: "system-level" else: true iff no other stmt in the system is exec.
⇒ do: if in a cycle; use goto or break to exit

→ atomic { guard ⇒ stmt... } - exec if guard is executable; stmts executed w/o interleaving with other processes. If a stmt blocks, atomicity is lost (but regained when atomic becomes exec again)
⇒ d_step { guard ⇒ stmt... } - more efficient than atomic; must be deterministic and may not block (used for model reduction together with atomic)
- goto in and out of d_step is forbidden! d_step { do... break od; } must insert skip
- goto and atomic: atomicity pres. only if goto jumps from one to another atomic and target is exec.
- d-step executed as single transition; infinite loops in atomic!! and d-step!! not detected

→ has priority over P

⇒ escape sequence: {P} unless {Q}: exec starts with P. before each stmt in P is exec'd, executability of 1st stmt in Q is checked - as soon as it is executable, control goes to Q
⇒ inline: textual subst. macro (inline swap(a,b) { tmp=a; a=b; b=tmp; })
⇒ proctype defines an automaton (S, s0, L, T, F) → accepting final states
  ↳ states correspond to control points; ↳ labels of transitions
  ↳ trans. rel: flow of control (labeled with basic stmts)
- 6 basic stmts: assignment, assert, print, send, receive, and expression stmts
  others specify ctl flow and cannot appear as labels on trans
  - every basic stmt. has a precondition (=when executable) and effects
- semantic engine operates a global reachability graph and deals only with local states and transitions (it doesn't know about e.g. if, goto).
  ⇒ global system state: variables, messages, channels, processes
  ⇒ state trans.: basic stmts labelling the trans; trans. selection and execution

⇒ variable: {name, scope, domain, inival, curval}
⇒ message: finite, ordered set of variables
⇒ channel: {ch-id, nslots, contents} - always global scope (although var may be local / global)
⇒ process: {pid, local_vars, local_states, inistate, curstate, transitions}
  local_vars: finite set; local_states: set of integers; transitions: subset of lstates

⇒ transition: {tr_id, src_state, dst_state, cond, effect, priority, rv}
  rv: predefined sys. vars used for unless and rendezvous

⇒ global state: {gvars, procs, chans, exclusive, handshake, timeout, else, stutter}
  gvars: set of global vars; procs: set of processes; chans: set of msg chans
  exclusive, handshake: used for atomic, d_step, rendez. (predefined sys vars - integers)
  timeout, else, stutter: predefined system boolean vars for stutter extension rule

→ global system state: state vector

TODO: semantics engine!

- E set of all executions, f correctness property
- model satisfies f if all executions do: $(E \vdash f) \Leftrightarrow (\forall \delta \in E \Rightarrow \delta \vdash f)$
  violates f if at least one violates: $\neg(E \vdash f) \Leftrightarrow (\exists \delta \in E \wedge \neg(\delta \vdash f))$

- showing $\neg(E \vdash f)$ is not the same as showing $(E \vdash \neg f)$

$\Updownarrow$ $\qquad$ $\Downarrow$

$\exists \delta \in E \wedge \neg(\delta \vdash f)$ $\qquad$ $\forall \delta \in E \Rightarrow (\delta \vdash \neg f) \stackrel{?}{\equiv} \neg(\delta \vdash f)$

$\boxed{(E \vdash \neg f) \Rightarrow \neg(E \Rightarrow f)}$ but not other way!

- ex:

```
byte x=0;
init {
  do :: x=0
     :: x=2
  od }
```

```
never {
  do :: assert(x==0) od
}
```

f violated by x=2

```
never {
  do :: assert(x==2) od
}
```

!f violated by x≠0

x is neither always ∅ nor is it always ≠∅

both $(E \vdash \neg f)$ and $\neg(E \vdash f)$

- debates

→ CTL vs LTL: CTL can express reset, LTL fairness ∅, CTL linear, LTL expt. M free of fm.
→ symbolic ver: BDDs – HW verif. (perf: variable ordering – NP complete); memory: # of nodes
→ explicit ver: PO reduction (optimal red: NP-compl.); memory: # of states
→ BFS adv.: shortest counterex. for safety; also more common in hybrid and real-time ver.
→ Tarjan vs. nested: Tarjan: all SCCs, easy expl. of strong fairness (nested doesn't find all (ultimate) acceptng runs)
→ event vs. state based: mutually interchangable, depending on props; combined appr. w/o penalty
→ RT vs timeless: SPIN: only rudimentary timeout
→ probability vs possibility
→ async vs. sync: sync: everything happens on clock ticks; async: beh. can be modelled in sync setting (distrib. syst ⇒ async); async checkers are more efficient but difficult to model sync beh. (SPIN: async model checker)
→ true vs. interleaving concurrency: true conc. requires more transitions; by representing shared data objs. at some granularity level it is possible to describe accurately possible beh. w/ interleaving conc.
→ open vs. closed: closed syst. must include all possible inputs and interactions w/ env.; open: env. beh. not known
→ forward reachability: verif. starts in initial state to compute set of successors; backward: from final state computes predecessors
→ compositional ver. method: minimizing state-space search; parallelizing w/o repeating comp. (many approaches non-comp.; underlying system not comp. by nature)
→ deductive vs. algorithmic – combine?

↓ any kind of system, requires much expertise

↘ restricted to finite-state systems

→ checking safety: basic DFS, stateless search, depth-limited, BFS
→ liveness: non-p cycles, acceptance cycles, nested DFS

→ search problem: given automaton $B \otimes \Pi A_i$
- find all runs that violate a safety property ("bad" reachable states), or liveness property (accepting ω-runs)
- want to avoid constructing the graph upfront

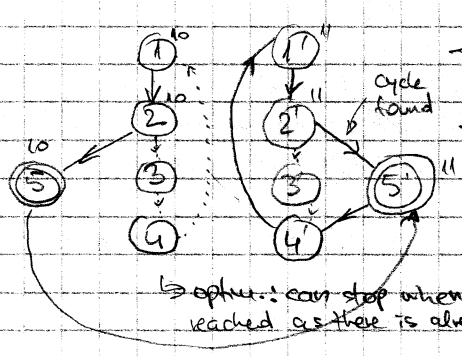
→ basic DFS: uses a global statespace set of visited states + stack
- check for safety each time a state is popped off the stack
- does not visit a state already in the global statespace

→ stateless DFS: instead of global statesp., examines just the current stack
- much redundant work

→ depth-bounded DFS: gives up when prespecified search depth exceeded
- stores min-depth with each node in the global statesp. (the min-depth is also updated if there exists a shorter path...)

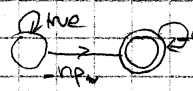
→ BFS: queue instead of stack — hard to report path leading to failure (no stack)
- pro: shortest path to safety violation, can't extend to detect cycles efficiently

→ omega-acceptance: existence of cyclic paths in product aut. that contain at least 1 accept. state
- DFS alone inadequate, Tarjan's SCC uses extra 2 ints per state
- nested DFS: there exists at least 1 accepting state reachable from init state and itself



↳ optim.: can stop when 1' is reached as there is already 1→5 path

- 2nd DFS uses 5 as the start node; combining two stack gives the complete trace for counter-example
- uses only two extra bits per state (10 = visited in 1st DFS, 01, 11)
- also used for LIP-cycle detection (LTL: $\Diamond\Box \neg np$)

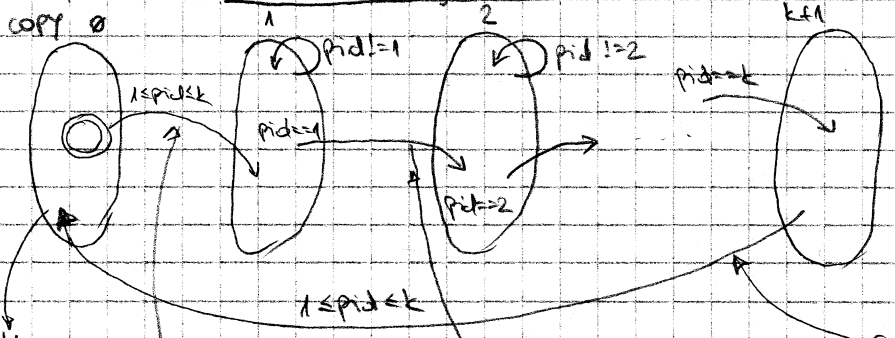



```
never {
do :: np -> break   accept:
   :: true          do :: np
od;                    od; }
```

2nd DFS sets to exec when the 1st starts to return from rec.

- 2nd search is started in postorder: `Search() { if () { Search() } ... if() { Search() } }`
- the algorithm is correct: it will detect at least one accept. cycle if such exists

→ weak fairness: Choueka's flag construction:



- k cache processes
- add nil trans from any state in copy i to corresp state in i+1 if i is blocked at that state
- adding k+2 bits to each state

only this copy preserves accept states

outg trans. of only accept. states point to corresp. state in copy 1

outs: all trans of proc. i point to corresp. state in copy i+1

all outg trans. point to corresp state in copy 0

LTL fairness: $\Box\Diamond p$

- all runs preserved, but unfolded. No accept cycles within copy 0
- all acc. cyc. must traverse all copies to return to copy 0 ⇒ necessarily weakly fair

(but this is essentially round-robin sched.)

→ complexity:
- k processes (2..10), M reach. states (10^9..10^{11}), B states in property aut (1..4), S state size
- system size $P = M \cdot B \cdot S$

	memory nature	
safety	P	P
liveness	P	2P
live+fair	P	$2P(k+2)$

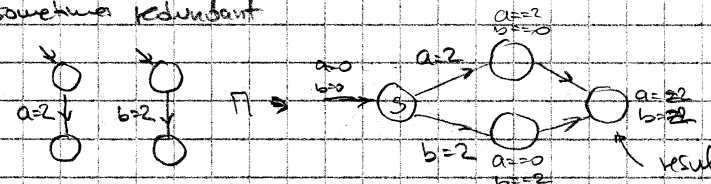
→ data objects: easily missed!
active [2] proctype bad() {
  int x; ...
}
- each int: 2^{32} states
- total 2^{64} states

- channels: 2 channels, s slots per chan, m dif. types of msgs;
total # of states: $R_Q = \left(\sum_{i=0}^{s} m^i\right)^2$ } ⇒ separate independent channels!

→ to reduce P = MBS: reducing any factor
- M can increase exp. with # of proc. and channels → usually not issue → risk of incompleteness
- B can increase exp. with length of LTL formula (lossless & lossy)
- algorithmic reduction: PO reduction (smaller M), compression, state recognizer instead of hash (reduce S)

→ PO reduction: async prod. is sometimes redundant

byte a,b;
active proctype A() {a=2;}
active proctype B() {b=2;}



- runs that differ only in the relat. order of independent operations are equivalent
- dependence: control, data, property (visibility) depends on the property
  ↳ accessing shared data
  ↳ execution order (e.g. sequencing within a process)

two transitions independent at state s:
- both are enabled & execution of neither can disable the other (control indep.)
- combined effect of both trans is independent of relative order of exec (no data / prop dep.)
- strong independence: if 2 trans are indep. at every state where both are enabled
- trans. is safe: strongly indep. from all other trans. in the system
- stmt. is conditionally safe: for condition C if it is safe in all states where c holds

⇒ reduction preserves all safety and liveness properties
- can result in exponential reduction in the size of reachable state space (M·B), no runtime overhead

→ statement merging: sequence of uncond. safe non-blocking transitions (e.g. x=1; x=y+z) predictably produces a non-interleaved run of states in the global graph

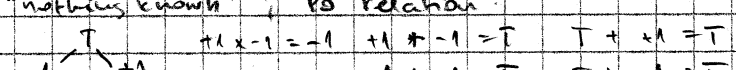
PRESENTATIONS

→ TIMED AUT:

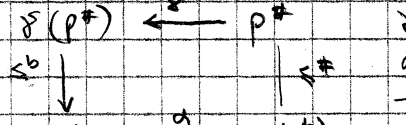
- real-time systems ; dense time domain
- ω-aut. augmented with finite set of clocks advancing synchr.
- each trans. can reset one or more clocks and/or have a condition
- clock constr. $\delta ::= x \leq n \mid n \leq x \mid \neg\delta \mid \delta_1 \wedge \delta_2$ time is increasing
- timed words: infinite seqs: $(a,0), (b,0.9), (a,1), (b,1.8), \ldots$
- Büchi cond, Müller cond ; determ. TA aut less expr. than DTBA (TTA, TBA equally exp.) nondet
- make a finite quotient out of infinite time space
- inclusion problem undecidable ($L(impl) \subseteq L(spec)$) ; emptiness problem decidable
- TTBA, TBA – not closed under complement
- DTMA is complementable ; DTBA can be complemented by some DTMA
- idea: transform TA to untimed FA w/o loss of inf. – make a time region aut.
- clocks compared only to nat. numbers ; clock region graph
- undecidable whether TA A accepts all timed words ⇒ inclusion undecidable

→ ABSTRACT INT:

- applicable to program analyzers, comparing formal semantics, designing proof methods...
- rules of sign calculus: concrete vals: integers, abstract vals: $+/-/0$ ($+1 \times +1 = +1$; $+1 + -1 = ?$)
- $\top$: "nothing known" ; Po relation:



- example: casting out nines: is multiplic. 217 × 38 correct?
- expr. syntax: $E \Rightarrow P = W$; $P \Rightarrow W \times N$; $N \Rightarrow D \mid WD$; $D = 0 \mid 1 \ldots 9$
- operational semantics: $[P = W]$ = true if $[P] = [W]$ ... $[WD] = 10 \times [W] + [D]$; $[0] = 0$ ...
- abstraction: $\alpha(x) = [x]_9$ if x is P, W, D mult. models
  $\alpha(P = W) =$ error if $[P]_9 \neq [W]_9$; unknown otherwise
- abstract compiler: transforms concrete expr. into abstr. expr. $C(E)$; e.g. $C(W_1 \times W_2) = C(W_1) \wedge C(W_2)$
- abst. interpreter: evaluates abstr. expr. e.g. $I(n_1 \times n_2) = (I(n_1) \times I(n_2)) \bmod 8$
- semantic analysis is correct since $\alpha(E) = I(C(E))$
- Galois connections: p^b concrete, $p^\#$ abstr. property



$\gamma : p^\# \Rightarrow p^b$ concretization f.
$\alpha : p^b \to p^\#$ abstraction f.
– α and γ determine each other
– composition of G.C. is a G.C.

→ RELY-GUARANTEE

- want to reason about the parts of the system in isolation
- need to know what interference might cause other subsys. – include in sys. spec.
- program represented as stack of commands (top equiv. to "locations" in SPL)
- ε: empty program, identity for $;_2$ and parallel composition
- program quadruple of predicates: (pre, rely, guar, post) $\in$
  - rely, guar should be transitive &.
  - when P is invoked, satisfies pre ; transitions caused by env. satisfy rely ;
  - any trans. caused by P satisfy guar ; in terminating state P satisfies post
- the proof system is sound and complete if the underlying data domain is