Verification

Lecture 4

Bernd Finkbeiner
Peter Faymonville
Michael Gerke

UNIVERSITÄT
DES
SAARLANDES

REVIEW: Safety

- Safety properties ≈ "nothing bad should happen" [Lamport 1977]
- Typical safety property: mutual exclusion property
  - the bad thing (having > 1 process in the critical section) never occurs
- Another typical safety property is deadlock freedom

⇒ These properties are in fact invariants

- An invariant is an LT property
  - that is given by a condition Φ for the states
  - and requires that Φ holds for all reachable states
  - e.g., for mutex property $\Phi \equiv \neg crit_1 \lor \neg crit_2$

REVIEW: Invariants

- An LT property P_{inv} over AP is an invariant if there is a propositional logic formula Φ over AP such that:

$$P_{inv} = \left\{ A_0 A_1 A_2 \ldots \in \left(2^{AP}\right)^{\omega} \;\middle|\; \forall j \geq 0.\, A_j \models \Phi \right\}$$

  - Φ is called an invariant condition of P_{inv}
- Note that

  $TS \models P_{inv}$ iff $trace(\pi) \in P_{inv}$ for all paths π in TS
  iff $L(s) \models \Phi$ for all states s that belong to a path of TS
  iff $L(s) \models \Phi$ for all states $s \in Reach(TS)$
- Φ has to be fulfilled by all initial states and
  - satisfaction of Φ is invariant under all transitions in the reachable fragment of TS

Checking an invariant

- Checking an invariant for the propositional formula Φ
  - = check the validity of Φ in every reachable state
  - $\Rightarrow$ use a slight modification of standard graph traversal algorithms (DFS and BFS)
  - provided the given transition system *TS* is finite
- Perform a forward depth-first search
  - at least one state *s* is found with $s \not\models \Phi \Rightarrow$ the invariance of Φ is violated
- Alternative: backward search
  - starts with all states where Φ does not hold
  - calculates (by a DFS or BFS) the set $\bigcup_{s \in S, s \not\models \Phi} Pre^*(s)$

REVIEW: Time complexity

- Under the assumption that
  - $s' \in Post(s)$ can be encountered in time $\Theta(|Post(s)|)$

  $\Rightarrow$ this holds for a representation of $Post(s)$ by adjacency lists
- The time complexity for invariant checking is $\mathcal{O}(\,N * (1 + |\Phi|) + M\,)$
  - where N denotes the number of reachable states, and
  - $M = \sum_{s \in S} |Post(s)|$ the number of transitions in the reachable fragment of TS
- The adjacency lists are typically given implicitly
  - e.g., by a syntactic description of the concurrent processes as program graphs
  - $Post(s)$ is obtained by the rules for the transition relation

REVIEW: Safety properties

- LT property P_{safe} over AP is a safety property if
  - for all $\sigma \in \left(2^{AP}\right)^{\omega} \setminus P_{safe}$ there exists a finite prefix $\widehat{\sigma}$ of σ such that:

$$P_{safe} \cap \underbrace{\left\{\sigma' \in \left(2^{AP}\right)^{\omega} \mid \widehat{\sigma} \text{ is a prefix of } \sigma'\right\}}_{\text{all possible extensions of } \widehat{\sigma}} = \varnothing$$

  - any such finite word $\widehat{\sigma}$ is called a bad prefix for P_{safe}
- Minimal bad prefix for P_{safe}:
  - is a bad prefix $\widehat{\sigma}$ for P_{safe} for which no proper prefix of $\widehat{\sigma}$ is a bad prefix for P_{safe}

  $\Rightarrow$ minimal bad prefixes are bad prefixes of minimal length

REVIEW: Safety properties and finite traces

For transition system TS without terminal states

and safety property P_{safe}:

$$TS \models P_{safe} \text{ if and only if } Traces_{fin}(TS) \cap BadPref(P_{safe}) = \varnothing$$

where $BadPref(P_{safe})$ is the set of bad prefixes of P_{safe}

REVIEW: Closure

- For trace $\sigma \in \left(2^{AP}\right)^{\omega}$, let $pref(\sigma)$ be the set of finite prefixes of σ:

$$pref(\sigma) = \{ \widehat{\sigma} \in \left(2^{AP}\right)^{*} \mid \widehat{\sigma} \text{ is a finite prefix of } \sigma \}$$

  - if $\sigma = A_0 A_1 \ldots$ then $pref(\sigma) = \left\{\varepsilon, A_0, A_0A_1, A_0A_1A_2, \ldots\right\}$ is infinite
- For property P this is lifted as follows: $pref(P) = \bigcup_{\sigma \in P} pref(\sigma)$
- The closure of LT property P:

$$closure(P) = \left\{\sigma \in \left(2^{AP}\right)^{\omega} \mid pref(\sigma) \subseteq pref(P)\right\}$$

  - the set of infinite traces whose finite prefixes are also prefixes of P, or
  - infinite traces in the closure of P do not have a prefix that is not a prefix of P

Safety properties and closures

LT property P over AP is a safety property

if and only if $closure(P) = P$

Proof

closure$(P) = P \quad \Rightarrow \quad P$ is a safety property

We show that for all $\sigma \in (2^{AP})^\omega \setminus P$ there exists a finite prefix $\widehat{\sigma}$ of σ such that $P \cap \left\{\sigma' \in \left(2^{AP}\right)^\omega \mid \widehat{\sigma} \text{ is a prefix of } \sigma'\right\} = \varnothing$.

- take an element $\sigma \in (2^{AP})^\omega \setminus P$
- since $\sigma \notin P = \mathit{closure}(P)$,
  there exists a finite prefix $\widehat{\sigma}$ of σ with $\widehat{\sigma} \notin \mathit{pref}(P)$
- by the definition of $\mathit{pref}(P)$,
  there is no $\sigma' \in P$ such that $\widehat{\sigma} \in \mathit{pref}(\sigma')$.
- hence, $\widehat{\sigma}$ is a bad prefix for P.

Proof (cont'd)

P is a safety property $\Rightarrow$ $closure(P) = P$

It suffices to show that $closure(P) \subseteq P$, because $P \subseteq closure(P)$ holds for all properties.
Proof by contradiction.

- assume there is some $\sigma \in closure(P) \setminus P$.
- since P is a safety property and $\sigma \notin P$, σ has a finite prefix $\widehat{\sigma} \in BadPref(\sigma)$.
- As $\sigma \in closure(P)$, we have $\widehat{\sigma} \in pref(\sigma) \subseteq pref(P)$.
- Hence, there exists a word $\sigma' \in P$ such that $\widehat{\sigma}$ is a prefix of σ'.
- This contradicts that P is a safety property.

Finite trace equivalence and safety properties

For TS and TS' be transition systems (over AP) without terminal states:

$$Traces_{fin}(TS) \subseteq Traces_{fin}(TS')$$

if and only if

for any safety property P_{safe} : $TS' \models P_{safe} \;\Rightarrow\; TS \models P_{safe}$

$$Traces_{fin}(TS) = Traces_{fin}(TS')$$

if and only if

TS and TS' satisfy the same safety properties

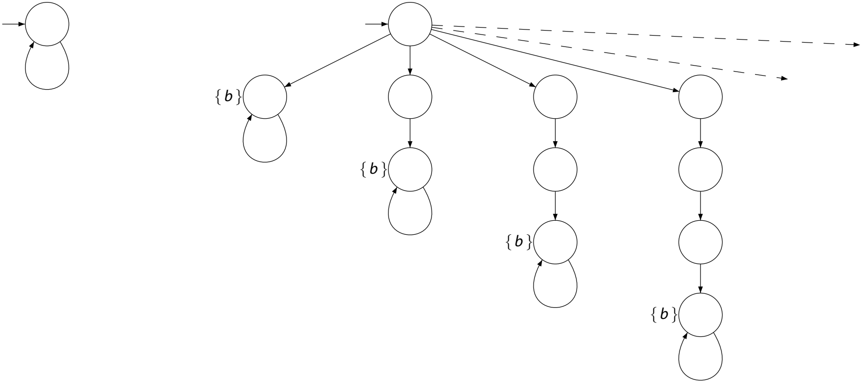
REVIEW: Finite vs. infinite traces

For *TS* without terminal states and finite TS'

trace inclusion and finite-trace inclusion coincide

this does not hold for infinite TS' (cf. next slide)
but also holds for image-finite TS'

REVIEW: Trace inclusion ≠ finite trace inclusion



$$Traces(TS) \nsubseteq Traces(TS') \quad \text{and} \quad Traces_{fin}(TS) \subseteq Traces_{fin}(TS')$$

Proof

$Traces(TS) \subseteq Traces(TS') \quad \Rightarrow \quad Traces_{fin}(TS) \subseteq Traces_{fin}(TS')$ holds because $Traces_{fin}(TS) = pref(Traces(TS))$.

For image-finite TS:
$Traces_{fin}(TS) \subseteq Traces_{fin}(TS') \quad \Rightarrow \quad Traces(TS) \subseteq Traces(TS')$

- Let $A_0A_1 \ldots \in Traces(TS)$. We show that there exists a path $s_0s_1 \ldots \in Paths(TS')$ with $trace(s_0s_1 \ldots) = A_0A_1 \ldots$
- Since $Traces_{fin}(TS) \subseteq Traces_{fin}(TS')$ we know that, for every $m \in \mathbb{N}$, there exists a finite path $\pi^m = s_0^m s_1^m \ldots s_m^m \in Paths_{fin}(TS')$ such that $trace(\pi^m) = A_0A_1 \ldots A_m$.
- Careful: There is no guarantee that π^m is a prefix of π^{m+1}!

Proof (cont'd)

We construct $s_0 s_1 \ldots$ inductively as follows, maintaining the following invariant: for every $m \in \mathbb{N}$, there are infinitely many $m' > m$ such that $\pi^{m'} = s_0^{m'} \ldots s_{m'}^{m'}$ is an initial finite path fragment in TS', $trace(\pi^{m'}) = A_0 \ldots A_{m'}$, and $s_0 \ldots s_m = s_0^{m'} \ldots s_m^{m'}$.

- base case ($m = 0$): For each m' there is an initial path fragment $s_0^{m'} \ldots s_{m'}^{m'}$ with $trace(\pi^{m'}) = A_0 \ldots A_{m'}$. Since there are only finitely many initial states, there must exist some initial state s_0 such that there are infinitely many $m' > 0$ with an initial path fragment $\pi^{m'} = s_0^{m'} \ldots s_{m'}^{m'}$ such that $s_0^{m'} = s_0$ and $trace(\pi^{m'}) = A_0 \ldots A_{m'}$.
- induction step ($m \to m + 1$): by induction hypothesis, there exist infinitely many $m' > m$ such that $\pi^{m'} = s_0^{m'} \ldots s_{m'}^{m'}$ is an initial finite path fragment in TS', $trace(\pi^{m'}) = A_0 \ldots A_{m'}$, and $s_0 \ldots s_m = s_0^{m'} \ldots s_m^{m'}$. Since s_m has only finitely many successors, there must exist some successor s_{m+1} such that there are infinitely many $m' > m + 1$ such that $\pi^{m'} = s_0^{m'} \ldots s_{m'}^{m'}$ is an initial finite path fragment in TS', $trace(\pi^{m'}) = A_0 \ldots A_{m'}$, and $s_0 \ldots s_{m+1} = s_0^{m'} \ldots s_{m+1}^{m'}$.

REVIEW: Liveness properties

LT property P_{live} over AP is a liveness property whenever

$$pref(P_{live}) = \left(2^{AP}\right)^*$$

- A liveness property is an LT property
  - that does not rule out any prefix
- Liveness properties are violated in "infinite time"
  - whereas safety properties are violated in finite time
  - finite traces are of no use to decide whether P holds or not
  - any finite prefix can be extended such that the resulting infinite trace satisfies P

Example liveness properties

- "If the tank is empty, the outlet valve will eventually be closed"
- "If the outlet valve is open and the request signal disappears, the outlet valve will eventually be closed"
- "If the tank is full and a request is present, the outlet valve will eventually be opened"
- "The program terminates within 31 computational steps"
  - $\Rightarrow$ a finite trace may violate this; this is a safety property!
- "The program eventually terminates"

Liveness properties for mutual exclusion

- Eventually:
  - each process will eventually enter its critical section
- Repeated eventually:
  - each process will enter its critical section infinitely often
- Starvation freedom:
  - each waiting process will eventually enter its critical section

how to formalize these properties?

Liveness properties for mutual exclusion

$P = \{ A_0 A_1 A_2 \ldots \mid A_j \subseteq AP \;\; \& \;\; \ldots \}$ and $AP = \{wait_1, crit_1, wait_2, crit_2\}$

- Eventually:

$$(\exists j \geq 0.\, crit_1 \in A_j) \;\wedge\; (\exists j \geq 0.\, crit_2 \in A_j)$$

- Repeated eventually:

$$\left(\overset{\infty}{\exists}\, j \geq 0.\, crit_1 \in A_j\right) \;\wedge\; \left(\overset{\infty}{\exists}\, j \geq 0.\, crit_2 \in A_j\right)$$

- Starvation freedom:

$$\forall j \geq 0.\; (wait_1 \in A_j \;\Rightarrow\; (\exists k > j.\, crit_1 \in A_k)) \;\wedge$$
$$\forall j \geq 0.\; (wait_2 \in A_j \;\Rightarrow\; (\exists k > j.\, crit_2 \in A_k))$$

Safety vs. liveness

- Are safety and liveness properties disjoint? Almost.
- Is every linear-time property a safety or liveness property? No.
- But:

for any LT property P an equivalent LT property P' exists

which is a conjunction of a safety and a liveness property

$\Rightarrow$ safety and liveness provide an essential characterization of LT properties

Basic properties

If P (over AP) is both a safety and a liveness property then:

$$P = \left(2^{AP}\right)^{\omega}$$

For any LT properties P and P':

$$closure(P \cup P') = closure(P) \cup closure(P')$$

Proof

$closure(P) \cup closure(P') \subseteq closure(P \cup P')$

- $P \subseteq P \cup P'$ implies that $closure(P) \subseteq closure(P \cup P')$
- analogously, $P' \subseteq P \cup P'$, hence $closure(P') \subseteq closure(P \cup P')$.

$closure(P \cup P') \subseteq closure(P) \cup closure(P')$

- Suppose $\sigma \in closure(P \cup P') \setminus (closure(P) \cup closure(P'))$.
- every finite prefix of σ is in $pref(P)$ or $pref(P')$ or both.
- case 1: there are infinitely many prefixes of σ in $pref(P)$. Then all finite prefixes of P are in $pref(P)$, hence $\sigma \in closure(P)$.
- case 2: there are infinitely many prefixes of σ in $pref(P')$. Then all finite prefixes of P are in $pref(P')$, hence $\sigma \in closure(P')$.
- case 3: there are only finitely many prefixes of σ in $pref(P)$ and only finitely many prefixes of σ in $pref(P')$. Then there are only finitely many prefixes of σ in $pref(P \cup P')$. Contradiction.

A non-safety and non-liveness property

"the machine provides infinitely often beer
after initially providing sprite three times in a row"

- This property consists of two parts:
  - it requires beer to be provided infinitely often
  - ⇒ as any finite trace fulfills this, it is a liveness property
  - the first three drinks it provides should all be sprite
  - ⇒ bad prefix = one of first three drinks is beer; this is a safety property
- Property is thus a conjunction of a safety and a liveness property

does this apply to all such properties?

Decomposition theorem

For any LT property P over AP there exists
a safety property P_{safe} and a liveness property P_{live}
(both over AP) such that:

$$P = P_{safe} \cap P_{live}$$

Proposal: $P = \underbrace{closure(P)}_{=P_{safe}} \cap \underbrace{\left(P \cup \left(\left(2^{AP}\right)^{\omega} \setminus closure(P)\right)\right)}_{=P_{live}}$

Proof

- $P_{safe} = closure(P)$ is a safety property: $closure(closure(P)) = closure(P)$.
- To show that P_{live} is a liveness property, we prove that $closure(P_{live}) \supseteq (2^{AP})^\omega$, which is equivalent to $pref(P_{live}) = (2^{AP})^*$.

$$
\begin{aligned}
closure(P_{live}) &= closure(P \cup ((2^{AP})^\omega \setminus closure(P))) \\
&= closure(P) \cup closure((2^{AP})^\omega \setminus closure(P)) \\
&\supseteq closure(P) \cup ((2^{AP})^\omega \setminus closure(P)) \\
&= (2^{AP})^\omega
\end{aligned}
$$

"Sharpest" decomposition theorem

Let P be an LT property and $P = P_{safe} \cap P_{live}$

where P_{safe} is a safety property and P_{live} a liveness property.

Then:

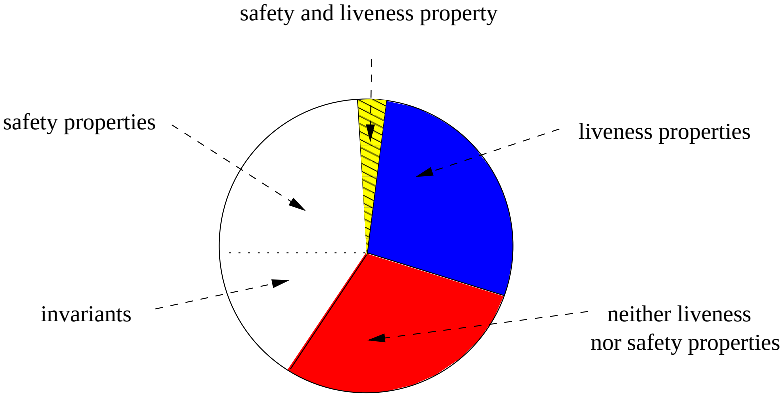
1. $closure(P) \subseteq P_{safe}$

2. $P_{live} \subseteq P \cup \left(\left(2^{AP}\right)^{\omega} \setminus closure(P) \right)$

$closure(P)$ is the strongest safety property and

$\left(\left(2^{AP}\right)^{\omega} \setminus closure(P) \right)$ the weakest liveness property

Classification of LT properties



Does this program terminate?

$$\text{Inc} \;|||\; \text{Reset}$$

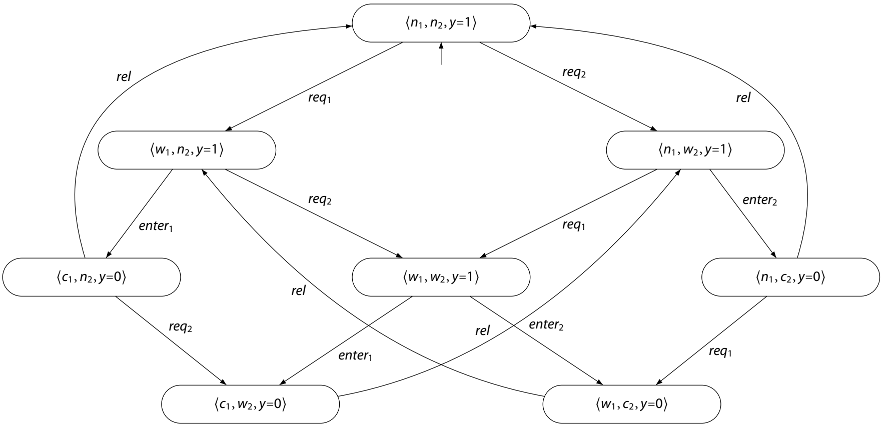
where

$$\textbf{proc}\ \text{Inc} \;=\; \textbf{while}\ \langle\, x \geq 0\ \textbf{do}\ x := x + 1 \,\rangle\ \textbf{od}$$

$$\textbf{proc}\ \text{Reset} \;=\; x := -1$$

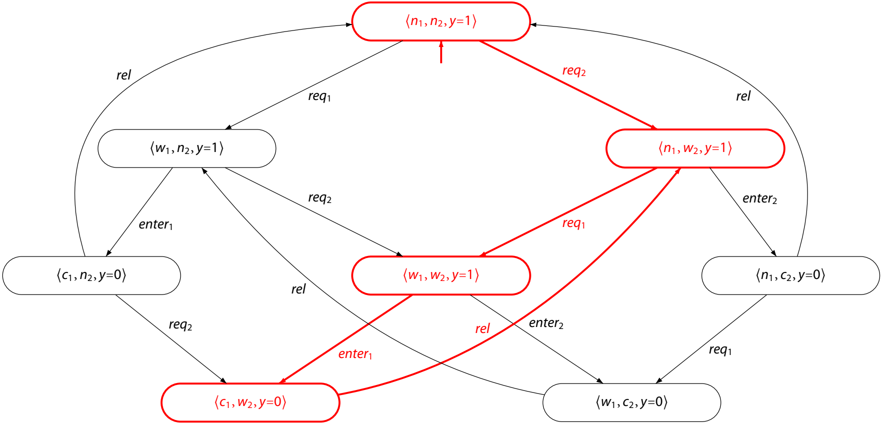
x is a shared integer variable that initially has value 0

Do we starve?



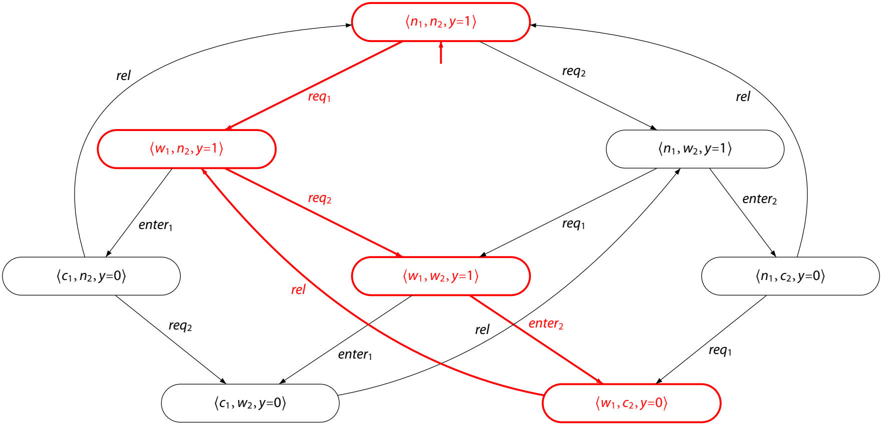
⟨n1, n2, y=1⟩
⟨w1, n2, y=1⟩
⟨n1, w2, y=1⟩
⟨c1, n2, y=0⟩
⟨w1, w2, y=1⟩
⟨n1, c2, y=0⟩
⟨c1, w2, y=0⟩
⟨w1, c2, y=0⟩
req1
req2
enter1
enter2
rel

Process two starves



process two finitely many times in critical section remains unfair

Process one starves



Fairness

- Starvation freedom is often considered under process fairness
  - ⇒ there is a fair scheduling of the execution of processes
- Fairness is typically needed to prove liveness
  - not for safety properties!
  - to prove some form of progress, progress needs to be possible
- Fairness is concerned with a fair resolution of nondeterminism
  - such that it is not biased to consistently ignore a possible option
- Problem: liveness properties constrain infinite behaviours
  - but some traces---that are unfair---refute the liveness property

Fairness constraints

- What is wrong with our examples? Nothing!
  - interleaving: not realistic as in reality no processor is infinitely faster than another
  - semaphore-based mutual exclusion: level of abstraction
- Rule out “unrealistic” runs by imposing fairness constraints
  - what to rule out? $\Rightarrow$ different kinds of fairness constraints
- “A process gets its turn infinitely often”
  - always — unconditional fairness
  - if it is enabled infinitely often — strong fairness
  - if it is continuously enabled from some point on — weak fairness

Fairness

This program terminates under unconditional fairness:

$$\textbf{proc}\ \text{Inc} \quad = \quad \textbf{while}\ \langle\, x \geq 0\ \textbf{do}\ x := x + 1\,\rangle\ \textbf{od}$$

$$\textbf{proc}\ \text{Reset} \quad = \quad x := -1$$

x is a shared integer variable that initially has value 0

Fairness constraints

- Unconditional fairness

  an activity is executed infinitely often
- Strong fairness

  if an activity is infinitely often enabled (not necessarily always!)
  then it has to be executed infinitely often
- Weak fairness

  if an activity is continuously enabled (no temporary disabling!)
  then it has to be executed infinitely often

we will use actions to distinguish fair and unfair behaviours

Fairness definition

For $TS = (S, Act, \rightarrow, I, AP, L)$ without terminal states, $A \subseteq Act$, and infinite execution fragment $\rho = s_0 \xrightarrow{\alpha_0} s_1 \xrightarrow{\alpha_1} \ldots$ of TS:

1. ρ is unconditionally A-fair whenever:

$$\text{true} \implies \underbrace{\forall k \geq 0.\ \exists j \geq k.\ \alpha_j \in A}_{\text{infinitely often } A \text{ is taken}}$$

2. ρ is strongly A-fair whenever:

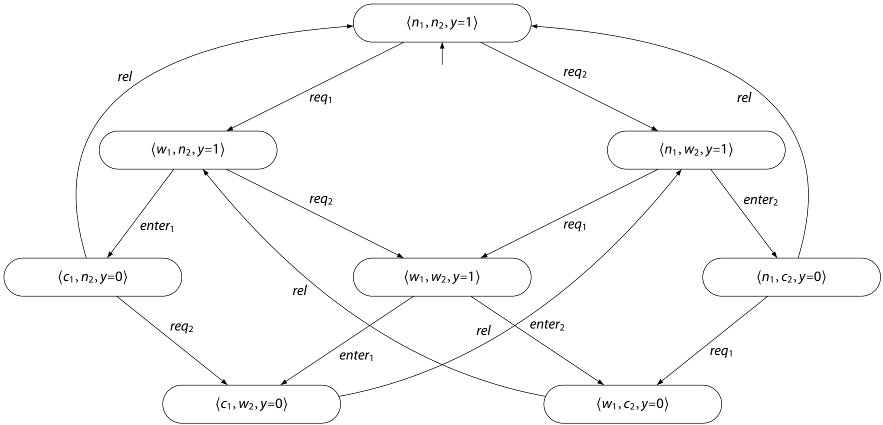
$$\underbrace{(\ \forall k \geq 0.\ \exists j \geq k.\ Act(s_j) \cap A \neq \varnothing\ )}_{\text{infinitely often } A \text{ is enabled}} \implies \underbrace{(\ \forall k \geq 0.\ \exists j \geq k.\ \alpha_j \in A\ )}_{\text{infinitely often } A \text{ is taken}}$$

3. ρ is weakly A-fair whenever:

$$\underbrace{(\ \exists k \geq 0.\ \forall j \geq k.\ Act(s_j) \cap A \neq \varnothing\ )}_{A \text{ is eventually always enabled}} \implies \underbrace{(\ \forall k \geq 0.\ \exists j \geq k.\ \alpha_j \in A\ )}_{\text{infinitely often } A \text{ is taken}}$$

where $Act(s) = \left\{ \alpha \in Act \mid \exists s' \in S.\ s \xrightarrow{\alpha} s' \right\}$

Example (un)fair executions



Which fairness notion to use?

- Fairness constraints aim to rule out "unreasonable" runs
- Too strong? $\Rightarrow$ relevant computations ruled out

  verification yields:
  - "false": error found
  - "true": don't know as some relevant execution may refute it
- Too weak? $\Rightarrow$ too many computations considered

  verification yields:
  - "true": property holds
  - "false": don't know, as refutation maybe due to some unreasonable run

Relation between fairness constraints

unconditional A-fairness $\Longrightarrow$ strong A-fairness $\Longrightarrow$ weak A-fairness

Fairness assumptions

- Fairness constraints impose a requirement on any $\alpha \in A$
- In practice: different constraints on different action sets needed
- This is realised by fairness assumptions

Fairness assumptions

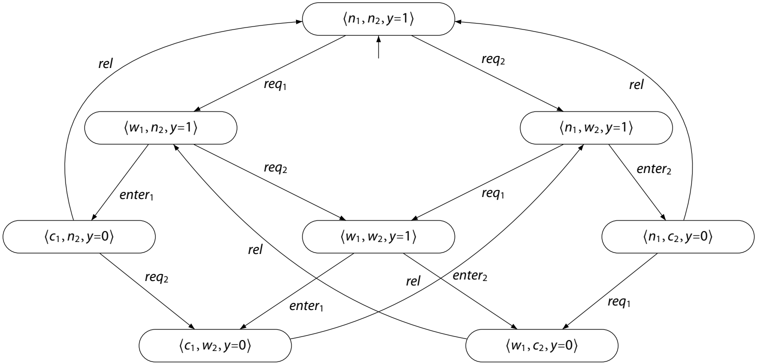
- A fairness assumption for *Act* is a triple

$$\mathcal{F} = (\mathcal{F}_{ucond}, \mathcal{F}_{strong}, \mathcal{F}_{weak})$$

  with $\mathcal{F}_{ucond}, \mathcal{F}_{strong}, \mathcal{F}_{weak} \subseteq 2^{Act}$.
- Execution ρ is $\mathcal{F}$-fair if:
  - it is unconditionally A-fair for all $A \in \mathcal{F}_{ucond}$, and
  - it is strongly A-fair for all $A \in \mathcal{F}_{strong}$, and
  - it is weakly A-fair for all $A \in \mathcal{F}_{weak}$

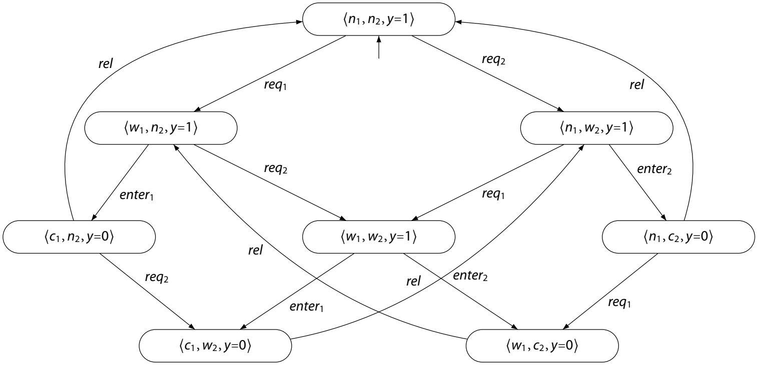
fairness assumption $(\varnothing, \mathcal{F}', \varnothing)$ denotes strong fairness; $(\varnothing, \varnothing, \mathcal{F}')$ weak, etc.

Fairness for mutual exclusion



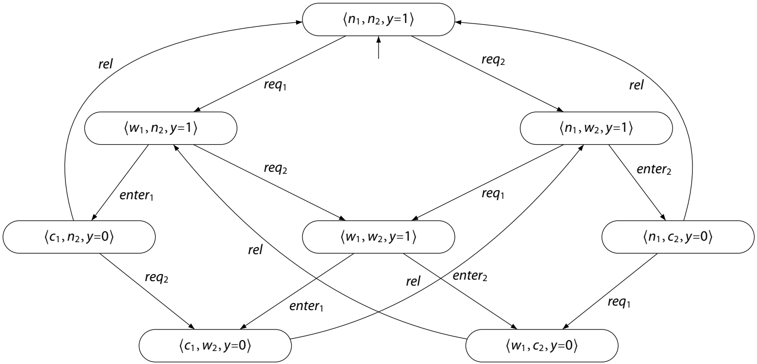
$$\mathcal{F} = \left(\varnothing, \underbrace{\big\{\{\, enter_1, enter_2 \,\}\big\}}_{\mathcal{F}_{strong}}, \varnothing\right)$$

Fairness for mutual exclusion



$$\mathcal{F} = (\varnothing, \underbrace{\{\{ enter_1 \}, \{ enter_2 \}\}}_{\mathcal{F}_{strong}}, \varnothing)$$

Fairness for mutual exclusion



$$\mathcal{F}' = \Big(\varnothing, \underbrace{\{\{\,enter_1\,\},\{\,enter_2\,\}\}}_{\mathcal{F}_{strong}}, \underbrace{\{\{\,req_1\,\},\{\,req_2\,\}\}}_{\mathcal{F}_{weak}}\Big)$$

in any $\mathcal{F}'$-fair execution each process infinitely often requests access

Fair paths and traces

- Path $s_0 \to s_1 \to s_2 \dots$ is $\mathcal{F}$-fair if
  - there exists an $\mathcal{F}$-fair execution $s_0 \xrightarrow{\alpha_1} s_1 \xrightarrow{\alpha_2} s_2 \dots$
  - $FairPaths_{\mathcal{F}}(s)$ denotes the set of $\mathcal{F}$-fair paths that start in s
  - $FairPaths_{\mathcal{F}}(TS) = \bigcup_{s \in I} FairPaths_{\mathcal{F}}(s)$
- Trace σ is $\mathcal{F}$-fair if there exists an $\mathcal{F}$-fair execution ρ with $trace(\rho) = \sigma$
  - $FairTraces_{\mathcal{F}}(s) = trace(FairPaths_{\mathcal{F}}(s))$
  - $FairTraces_{\mathcal{F}}(TS) = trace(FairPaths_{\mathcal{F}}(TS))$

these notions are only defined for infinite paths and traces; why?

Fair satisfaction

- *TS* satisfies LT-property *P*:

$$TS \models P \quad \text{if and only if} \quad Traces(TS) \subseteq P$$

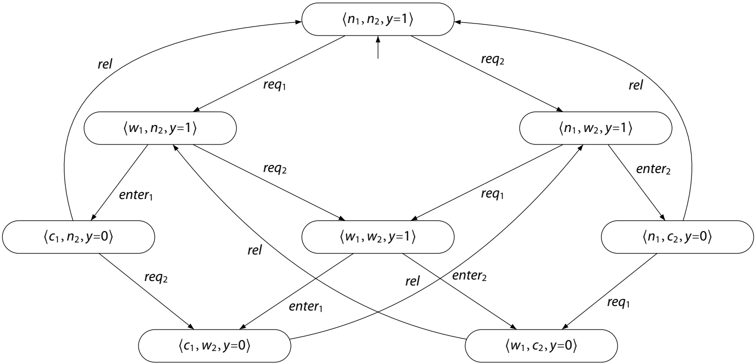
  - *TS* satisfies the LT property *P* if all its observable behaviors are admissible

- *TS* fairly satisfies LT-property *P* wrt. fairness assumption $\mathcal{F}$:

$$TS \models_{\mathcal{F}} P \quad \text{if and only if} \quad FairTraces_{\mathcal{F}}(TS) \subseteq P$$

  - if all paths in *TS* are $\mathcal{F}$-fair, then $TS \models_{\mathcal{F}} P$ if and only if $TS \models P$
  - if some path in *TS* is not $\mathcal{F}$-fair, then possibly $TS \models_{\mathcal{F}} P$ but $TS \not\models P$

Fairness for mutual exclusion



$TS \not\models$ "every process enters its critical section infinitely often"

and $TS \not\models_{\mathcal{F}}$ "every . . . often"

but $TS \models_{\mathcal{F}'}$ "every . . . often"

Fair concurrency with synchronization

$TS_i = (S_i, Act_i, \rightarrow_i, I_i, AP_i, L_i)$, for $1 \leq i \leq n$, has no terminal states

$$TS = TS_1 \parallel TS_2 \parallel \ldots \parallel TS_n$$

TS_i and TS_j ($i \neq j$) synchronize on their common actions:

$$Syn_{i,j} = Act_i \cap Act_j$$

$Syn_{i,j} \cap Act_k = \varnothing$ for any $k \neq i, j$
For simplicity, it is assumed that TS has no terminal states

how to establish a fair communication mechanism?

Asynchronous concurrent systems

concurrency = interleaving (i.e., nondeterminism) + fairness

Some fairness assumptions

- Strong fairness constraint: $\{Act_1, Act_2, \ldots, Act_n\}$
  - TS_i executes an action (not necessarily a sync!) infinitely often provided TS is infinitely often in a (global) state with a transition of TS_i enabled
- Strong fairness constraint: $\{\, \{ \alpha \} \mid \alpha \in Syn_{i,j}, 0 < i < j \leq n \,\}$
  - every individual synchronization is forced to happen infinitely often
- Strong fairness constraint: $\{\, Syn_{i,j} \mid 0 < i < j \leq n \,\}$
  - every pair of processes is forced to synchronize infinitely often
- Strong fairness constraint: $\{\, \bigcup_{0<i<j\leq n} Syn_{i,j} \,\}$
  - a synchronization (possibly the same) takes place infinitely often

Realizable fairness

For *TS* with set of actions *Act* and fairness assumption $\mathcal{F}$ for *Act*:
$\mathcal{F}$ is realizable for *TS* if for any $s \in Reach(TS)$: $FairPaths_{\mathcal{F}}(s) \neq \varnothing$

every initial finite execution fragment of *TS* can be completed to a fair execution

The suffix property

$$\underbrace{s'_0 \xrightarrow{\beta_1} s'_1 \xrightarrow{\beta_2} \ldots \xrightarrow{\beta_n} s'_n}_{\text{arbitrary starting fragment}} = \underbrace{s_0 \xrightarrow{\alpha_1} s_1 \xrightarrow{\alpha_2} s_2 \xrightarrow{\alpha_3} \ldots}_{\text{fair continuation}}$$

Realizable fairness and safety

For TS and safety property P_{safe} (both over AP)

and $\mathcal{F}$ a realizable fairness assumption for TS:

$$TS \models P_{safe} \quad \text{if and only if} \quad TS \models_{\mathcal{F}} P_{safe}$$

Summary LT properties

- LT properties are finite sets of infinite words over 2^{AP} (= traces)
- An invariant requires a condition Φ to hold in any reachable state
- Each trace refuting a safety property has a finite prefix causing this
  - invariants are safety properties with bad prefix $\Phi^*(\neg\Phi)$
  - a safety property is regular iff its set of bad prefixes is a regular language

  $\Rightarrow$ safety properties constrain finite behaviors
- A liveness property does not rule out finite behaviour

  $\Rightarrow$ liveness properties constrain infinite behaviors
- Any LT property is equivalent to a conjunction of a safety and a liveness property

Summary of fairness

- Fairness constraints rule out unrealistic traces
  - i.e., constraints on the actions that occur along infinite executions
  - important for the verification of liveness properties
- Unconditional, strong, and weak fairness constraints
  - unconditional $\Rightarrow$ strong fair $\Rightarrow$ weak fair
- Fairness assumptions allow distinct constraints on distinct action sets
- (Realizable) fairness assumptions are irrelevant for safety properties