

Lecture 15: Nov 6
Omega Languages and Fair Modules

- Reformulate model checking as well as refinement checking
 - Fairness in modules
 - Liveness in requirements
- First, let us study theory of ω -languages
- ω -word over A : infinite word over alphabet A
- ω -language over A : set of ω -words
- Questions:
 - What is the structure of ω -languages?
 - How do we define ω -languages from ordinary languages?

Response languages

- For a language L over A , $\text{recur}(L)$ is the ω -language

$$\{\underline{a} \mid \forall j \geq 0. \exists i \geq j. \bar{a}_{0..i} \in L\}$$

(infinitely many prefixes in L)

- The ω -language \mathcal{L} is a response language if there is a language L such that $\mathcal{L} = \text{recur}(L)$.
- Typical example: every request is followed by a response (starvation freedom)
- Infinitely many b symbols:

$$(a^*b)^\omega = \text{recur}((a^*b)^*)$$

- All safety, guarantee, and obligation languages are response languages

Closure properties of response

- Response languages are closed under union:

$$\text{recur}(L_1) \cup \text{recur}(L_2) = \text{recur}(L_1 \cup L_2)$$

- Closure under intersection not immediately obvious:

$$\text{recur}(L_1) \cap \text{recur}(L_2) \neq \text{recur}(L_1 \cap L_2)$$

- $\text{recur}(L_1) \cap \text{recur}(L_2)$ equals $\text{recur}(L_{12})$, where $\bar{a}_{0\dots m}$ is in L_{12} if

- it is in L_2 and

- it has a prefix $\bar{a}_{0\dots k}$ in L_1 with $\bar{a}_{0\dots j} \notin L_2$ for $k < j < m$.

- Not closed under complement: complement of $(a^*b)^\omega$ is not a response language
- Alternative definition of response languages: \mathcal{L} is response if it is intersection of countably many guarantee languages

Persistence languages

- For a language L over A , $\text{persist}(L)$ is the ω -language

$$\{\underline{a} \mid \exists j \geq 0. \forall i \geq j. \bar{a}_{0..i} \in L\}$$

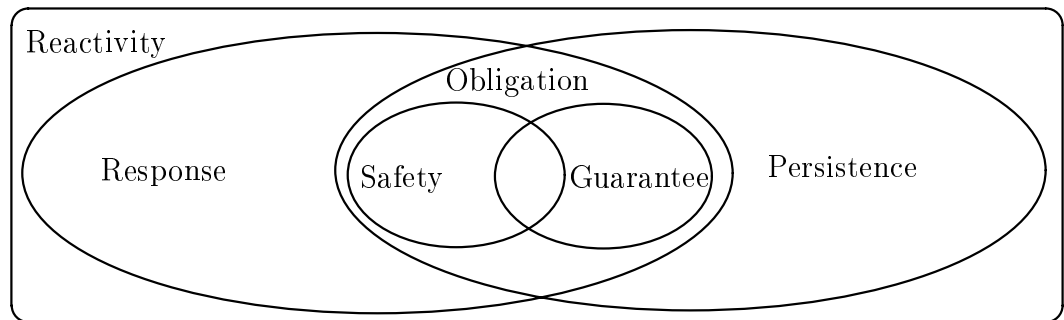
(eventually all prefixes in \mathcal{L})

- The ω -language \mathcal{L} is a persistence language if there is a language L such that $\mathcal{L} = \text{persist}(L)$.
- Typical persistence requirement: a protocol eventually stabilizes
- Only finitely many b symbols: $A^*a^\omega = \text{persist}(A^*a^*)$
- \mathcal{L} is a response language iff the complementary language $A^\omega \setminus \mathcal{L}$ is a persistence language.
- Persistence languages are closed under union, intersection, but not under complementation

Reactivity languages

- The ω -language \mathcal{L} is a 1-reactivity language if there exists a persistence language \mathcal{L}_1 and a response language \mathcal{L}_2 such that $\mathcal{L} = \mathcal{L}_1 \cup \mathcal{L}_2$.
- The ω -language \mathcal{L} is a k -reactivity language, for a natural number k , if there exist k 1-reactivity languages $\mathcal{L}_1, \dots, \mathcal{L}_k$ such that $\mathcal{L} = \mathcal{L}_1 \cap \dots \cap \mathcal{L}_k$
- Class of reactivity languages is closed under all boolean operations
- Example: For $A = \{a, b, c\}$,
 - the ω -language consisting of ω -words with infinitely many b symbols or only finitely many a symbols is a 1-reactivity language
 - ω -language consisting of ω -words with infinitely many b symbols and only finitely many a symbols is a 2-reactivity language

Classes of ω -languages



Safety-Liveness Classification

- Safe languages: limit-closed, defined by finite prefixes
- Live languages: finite prefixes give no information
- The ω -language \mathcal{L} is live if $\text{pref}(\mathcal{L}) = A^*$ (every finite word can be extended to \mathcal{L}).
- The ω -language A^ω is the only ω -language over the alphabet A that is both safe and live
- Machine-closure: Given a safe ω -language \mathcal{L}_S and a live ω -language \mathcal{L}_L , the pair $(\mathcal{L}_S, \mathcal{L}_L)$ is machine-closed if $\text{pref}(\mathcal{L}_S \cap \mathcal{L}_L) = \text{pref}(\mathcal{L}_S)$.
- Intuition: after a finite prefix, if safety is not violated then there is a possibility to produce an extension in $\mathcal{L}_S \cap \mathcal{L}_L$

- Examples:
 - $\mathcal{L}_1 = (b^*a)^\omega$ is live
 - $\mathcal{L}_2 = ((a + b)a)^\omega$ is safe
 - $(\mathcal{L}_2, \mathcal{L}_1)$ is machine-closed
 - $\mathcal{L}_4 = a^\omega + b^\omega$ is safe
 - $(\mathcal{L}_4, \mathcal{L}_1)$ is not machine-closed

- Set of infinite executions will be specified by a machine-closed pair
 - Safety component specified by transition relation
 - Liveness component specified by fairness constraints

- Machine closure implies that fairness can be ignored while verifying safety properties

- Theorem: Let $(\mathcal{L}_S, \mathcal{L}_L)$ be a machine-closed specification of the ω -language \mathcal{L} , and let \mathcal{L}' be a safe language. Then, $\mathcal{L} \subseteq \mathcal{L}'$ iff $\mathcal{L}_S \subseteq \mathcal{L}'$

Topological Characterization

- All the classes can be defined using classical topological concepts
- Consider metric d over the set of all ω -words
 - $d(\underline{a}, \underline{a})$ equals 0
 - $d(\underline{a}, \underline{b})$ equals $1/2^i$ if i is the maximum integer j such that $\bar{a}_{0\dots j} = \bar{b}_{0\dots j}$
- Safe languages: closed sets
- Live languages: dense sets
- Guarantee languages: open sets
- Response languages: countable intersections of open sets G_δ
- Persistence languages: countable unions of closed sets F_σ

ω -Trajectories

- ω -trajectory of a transition graph G is an infinite path over states
- The ω -language \mathcal{L}_G of the transition graph G is the set of initialized ω -trajectories of G
- The language \mathcal{L}_G is limit-closed (safe)
- We want to add fairness assumptions so that some limits are disallowed

Towards Fairness

```
module AsyncCount is  
  interface Count:  $\mathbb{N}$   
  atom controls count  
  init  
     $\parallel$  true  $\rightarrow$  count' := 0  
  update  
     $\parallel$  true  $\xrightarrow{\alpha}$  count' := count + 1  
     $\parallel$  true  $\xrightarrow{\beta}$  count' := count
```

- Update Choice of an atom: a subset of the update command
- Sample update choices: α, β
- In each round, an update choice is either available or not
- In each round, an available update choice is either executed or not

Weak fairness

```
module FairCount is  
  interface Count:  $\mathbb{N}$   
  atom controls count  
  init  
     $\parallel$  true  $\rightarrow$  count' := 0  
  update weaklyfair  $\alpha$   
     $\parallel$  true  $\xrightarrow{\alpha}$  count' := count + 1  
     $\parallel$  true  $\xrightarrow{\beta}$  count' := count
```

- An infinite trajectory is weakly fair to α if there is no i such that
 - α is available in every round after i
 - α is never executed after round i
- Alternatively: either infinitely often disabled or infinitely often executed
- The ω -language of FairCount contains only those ω -trajectories that are weak-fair to α

Need for Strong Fairness

- Weak fairness requires that a choice that is continuously available is eventually executed.
- What if a choice is available repeatedly, but not continuously, but never executed?
- Example: a buffer that may lose messages

module *LossyBuffer* **is**

interface $y : \mathbb{E}$

external $x : \mathbb{E}$

passive atom controls y **reads** x, y **awaits** x

update

$\parallel x? \xrightarrow{\alpha} y?$

$\parallel x? \xrightarrow{\beta}$

- Following is not weak-fair to α

$$[(y = 0, x = 0)(y = 0, x = 1)]^\omega;$$

- Following is weak-fair to α (even though y is never issued)

$$[(y = 0, x = 0)(y = 0, x = 0)(y = 0, x = 1)(y = 0, x = 1)]^\omega$$

Strong Fairness

- How do we model the assumption that a repeatedly sent message is eventually delivered?

- Fair buffer:

module *FairBuffer* **is**

interface $y : \mathbb{E}$

external $x : \mathbb{E}$

passive atom controls y **reads** x, y **awaits** x

update stronglyfair α

$\parallel x? \xrightarrow{\alpha} y?$

$\parallel x? \xrightarrow{\beta}$

- An infinite trajectory is strongly fair to α if either α is executed infinitely often or α is available only finitely often
- The ω -language of FairBuffer contains only those ω -trajectories that are strong-fair to α

Fair Module

- Reactive module $P +$
- Weak fairness: a finite set $WeakF_P$ of update choices of $P+$
- Strong fairness: a finite set \mathbf{P} of update choices of P .

Mutual Exclusion

- Goal: to verify liveness requirements such as
 - deadlock freedom: if some process requests an entry then some process is eventually granted critical section
 - starvation freedom: if a process requests an entry then is eventually granted critical section
- These do not hold on all ω -trajectories
- Solution: add fairness to rule out trajectories where available choices are ignored forever
- Example: We do not know how long a process stays inside its CS, but we want it to eventually exit it

Synchronous Mutual Exclusion

module Q_1 **is**

atom controls pc_1 **reads** pc_1, pc_2

update weaklyfair α_1

$\parallel pc_1 = out$	\rightarrow	
$\parallel pc_1 = out$	\rightarrow	$pc'_1 := req$
$\parallel pc_1 = req \wedge pc_2 \neq in$	\rightarrow	$pc'_1 := in$
$\parallel pc_1 = in$	\rightarrow	
$\parallel pc_1 = in$	$\xrightarrow{\alpha_1}$	$pc'_1 := out$

module Q_2 **IS**

atom controls pc_2 **reads** pc_1, pc_2

update weaklyfair α_2

$\parallel pc_2 = out$	\rightarrow	
$\parallel pc_2 = out$	\rightarrow	$pc'_2 := req$
$\parallel pc_2 = req \wedge pc_1 = out$	\rightarrow	$pc'_2 := in$
$\parallel pc_2 = in$	\rightarrow	
$\parallel pc_2 = in$	$\xrightarrow{\alpha_2}$	$pc'_2 := out$

$FairSyncMutex = Q_1 \parallel Q_2$

Asynchronous Mutual Exclusion

module \mathcal{P}_1 **is**

lazy atom controls pc_1, x_1 **reads** pc_1, pc_2, x_1, x_2

update weaklyfair α_1, β_1

	$pc_1 = out$	\rightarrow	$pc'_1 := req; x'_1 := x_2$
	$pc_1 = req \wedge pc_2 = out$	$\xrightarrow{\beta_1}$	$pc'_1 := in$
	$pc_1 = req \wedge x_1 \neq x_2$	$\xrightarrow{\beta_1}$	$pc'_1 := in$
	$pc_1 = in$	$\xrightarrow{\alpha_1}$	$pc'_1 := out$

module \mathcal{P}_2 **is**

lazy atom controls pc_2, x_2 **reads** pc_1, pc_2, x_1, x_2

update weaklyfair α_2, β_2

	$pc_2 = out$	\rightarrow	$pc'_2 := req; x'_2 := \neg x_1$
	$pc_2 = req \wedge pc_1 = out$	$\xrightarrow{\beta_2}$	$pc'_2 := in$
	$pc_2 = req \wedge x_1 = x_2$	$\xrightarrow{\beta_2}$	$pc'_2 := in$
	$pc_2 = in$	$\xrightarrow{\alpha_2}$	$pc'_2 := out$

$FairPete = \mathbf{hide} \ x_1, x_2 \ \mathbf{in} \ \mathcal{P}_1 \parallel \mathcal{P}_2$

Using Fairness

- Use judiciously, only when needed (analysis is more difficult)
- In deterministic modules (eg. synchronous circuits) no fairness needed
- Typical conditions:
 - Asynchronous modules: to rule out stuttering forever
 - Abnormal conditions (eg. message loss): fairness needed to design solutions
 - To model random phenomena

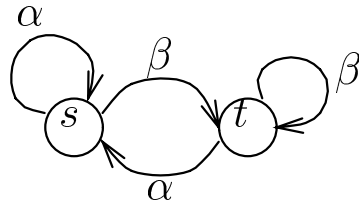
Operations on Fair Modules

- How do we define complex modules from simpler ones?
- Parallel Composition $\mathcal{P} = (P, WeakF_P, StrongF_P)$ and $\mathcal{Q} = (Q, WeakF_Q, StrongF_Q)$
 - P and Q must be compatible
 - result is $(P||Q, WeakF_{P \cup Q}, StrongF_{P \cup Q})$.
- Variable renaming: $\mathcal{P}[\rho]$
- Variable Hiding: **hide** x **in** \mathcal{P}

Fair Graphs

- Transition graphs + fairness constraints
- Operational semantics for fair modules
- Action of a graph $G = (\Sigma, \sigma^I, \rightarrow)$:
a subset of \rightarrow
- Fairness constraint: pair of actions (α, β)
- Fairness Assumption: set of fairness constraints
- A fair graph \mathcal{G} consists of a transition graph G
and a fairness assumption F for G

Examples



- α -fair: infinitely many visits to s
- β -fair: infinitely many visits to t
- $f_1 = (\rightarrow, \beta)$: f_1 -fair means infinitely many visits to t : $(s^*t)^\omega$
- $f_2 = (\alpha, \beta)$: if α -fair then β -fair
- Let $f_3 = (\rightarrow, \alpha)$, and $F = \{f_1, f_3\}$. Then, F -fair means infinitely many visits to both s and t : $(s^*t)^\omega \cap (t^*s)^\omega$
- $f_4 = (\alpha, \emptyset)$: f_4 -fair means only finitely visits to s : $(s + t)^*t^\omega$

Semantics of Fair Graphs

- For an action α , the ω -trajectory \underline{s} is α -fair if $s_i \xrightarrow{\alpha} s_{i+1}$ for infinitely many i
- For a fairness constraint $f = (\alpha, \beta)$, \underline{s} is f -fair if it is β -fair or not α -fair
 - Intuitively, if α repeats then so should β
- For a fairness assumption F , \underline{s} is F -fair if it is f -fair for all $f \in F$
- The fair language $\mathcal{L}_{\mathcal{G}}$ of a fair graph \mathcal{G} is the set of initialized F -fair trajectories
- Thm: The ω -language $\mathcal{L}_{\mathcal{G}}$ is a reactivity language

Restricted Types of Constraints

- Weak-fair constraint: (\rightarrow, α)
 - (\rightarrow, α) -fair trajectory means α -fair
 - Unconditional repetition
- Weak-fair graphs: all fairness constraints are weak
 - Easier to analyze
- Constraints specified by regions instead of actions
 - For a region σ , the ω -trajectory \underline{s} is σ -fair if $s_i \in \sigma$ for infinitely many i
 - A fair graph with fairness assumption of the form $\{(\sigma_1, \tau_1), \dots, (\sigma_k, \tau_k)\}$ is called a Streett automaton
 - Weak-fair graph with single constraint specified by a region is called Büchi automaton
 - Büchi Automata to be studied later as a requirements language

Machine closed graphs

- A fair graph \mathcal{G} is said to be machine-closed if every trajectory of \mathcal{G} can be extended to a fair trajectory.
- Intuition: fairness assumption constrains only what happens in the limit, and cannot be violated after finitely many steps
- Analog of seriality for transition graphs: there is always a way to continue
- The set of prefixes of fair trajectories equals the set of (finite) trajectories
- How does it help: to verify safety properties, we can ignore fairness constraints

Local fairness

- Syntactic way to ensure machine closure
- A fairness constraint (α, β) is local if for all $s \xrightarrow{\alpha} t$, there is a state $u \in \Sigma$ such that $s \xrightarrow{\beta} u$.
 - whenever α is available so is β
- A locally-fair graph is a fair graph (G, F) such that
 - G is serial
 - F contains only local fairness constraints
- Every locally fair graph is machine closed

Execution of locally fair graphs

Input: a locally fair graph \mathcal{G} and a state s ;

Output: a source- s fair trajectory \underline{s} of \mathcal{G} .

Queue: a queue of fairness constraints

Initialization

$s_0 := s$;

Queue equals F (in some order)

Update rounds.

for $i := 0$ to ∞ do

Let Queue be $f_1 f_2 \dots f_n$ with $f_k = (\alpha_k, \beta_k)$;

if $Available(\alpha_k, s_i)$ for some $1 \leq k \leq n$

then

$j := \min \{k \mid Available(\alpha_k, s_i)\}$;

$s_{i+1} := Execute(\beta_j, s_i)$;

Queue := $f_1 \dots f_{j-1} f_{j+1} \dots f_n f_j$

else $s_{i+1} := Execute(\rightarrow, s_i)$

From Modules to Graphs

- With every update choice α , associate two actions:
 - Availability action: $avail_\alpha$
 - Execution action: $exec_\alpha$

- For every weak-fairness choice a , add the constraint

$$(\rightarrow_P, exec_a \cup (\rightarrow_P \setminus avail_a))$$

- For every strong-fairness choice a , add the constraint

$$(avail_a, exec_a)$$

- Every fair module \mathcal{P} defines a fair graph $\mathcal{G}_\mathcal{P}$
- The resulting fair graph is local
- If no strong-fairness constraints, the resulting graph is weak-fair