Temporal Logic

Equivalence vs. Model Checking

- Up until now, we have described correctness as a form of equivalence between two systems
  - An implementation is correct if it's equivalent (i.e., strongly or branching bisimilar) to a specification
- An alternative approach: model checking
  - describe correctness as a set of logical formulas
  - Then, check that the implementation satisfies those formulas
- Benefits:
  - The properties are language-independent
  - Modularity (easy to add/change/remove properties)

Modal logics

- Reason about the sequencing and branching of transitions in an LTS
- Basic modal operators:
  - Possibility: from a state, there exists (at least) an outgoing transition labeled by a given action and leading to a state with a given property
  - Necessity: from a state, all the outgoing transitions labeled by a given action lead to states with a given property
- Hennessy-Milner Logic (HML): express properties of an LTS

HML syntax

φ ::=		true	constant "true"
	|	false	constant "false"
	|	$\neg\varphi$	negation
	|	$\varphi_1 \wedge \varphi_2$	conjunction
	|	$\varphi_1 \vee \varphi_2$	disjunction
	|	$\langle\, \alpha \,\rangle\, \varphi$	possibility ("diamond")
	|	$[\, \alpha \,]\, \varphi$	necessity ("box")

HML semantics

- When does a state s satisfy a formula φ? ($s \vDash \varphi$)
  - true: satisfied by all states; false: never satisfied
  - s satisfies $\neg\varphi$ iff. it does not satisfy φ
  - s satisfies $\varphi_1 \wedge \varphi_2$ iff. it satisfies both φ_1 and φ_2
  - s satisfies $\varphi_1 \vee \varphi_2$ iff. it satisfies either φ_1 or φ_2 (or both)
  - s satisfies $\langle\alpha\rangle\varphi$ iff. there is at least one state s' such that $s \xrightarrow{\alpha} s'$ and $s' \vDash \varphi$
  - s satisfies $[\alpha]\varphi$ iff. for every state s', If $s \xrightarrow{\alpha} s'$ then $s' \vDash \varphi$
  - Notice that $[\alpha]\varphi = \neg(\langle\alpha\rangle\neg\varphi)$
- An LTS satisfies φ if its initial state satisfies it

HML semantics (2/2)

- Given an LTS with states S and transition relation T, for every HML formula φ we define $[[\varphi]]$ as the set of states in S that satisfy φ
  - $[[\text{true}]] = S$, $[[\text{false}]] = \emptyset$
  - $[[\neg\varphi]] = S \setminus [[\varphi]]$
  - $[[\varphi_1 \wedge \varphi_2]] = [[\varphi_1]] \cap [[\varphi 2]]$
  - $[[\varphi_1 \vee \varphi_2]] = [[\varphi_1]] \cup [[\varphi 2]]$
  - $[[\langle \alpha \rangle \varphi]] = \{ s \in S \mid \exists s'.\ s \xrightarrow{\alpha} s' \wedge s' \in [[\varphi]] \}$
  - $[[[\alpha] \varphi]] = \{ s \in S \mid \forall s'.\ s \xrightarrow{\alpha} s' \Rightarrow s' \in [[\varphi]] \}$
- An LTS satisfies φ if its initial state is in $[[\varphi]]$

Action formulas

- $\langle\alpha\rangle\varphi$ = "After action α, possibly φ"
- $[\alpha]\varphi$ = "After action α, necessarily φ"

Sometimes it's useful to have shorter notations:

- $\langle\alpha_1 \vee \alpha_2\rangle\ \varphi = \langle\alpha_1\rangle\varphi \vee \langle\alpha_2\rangle\varphi$
  - "After α_1 or α_1, possibly φ"
- $[\alpha_1 \vee \alpha_2]\ \varphi = [\alpha_1]\varphi \wedge [\alpha_2]\varphi$
- $[\neg\alpha_2]\varphi$ = "After anything except α_2, necessarily φ"
- $\langle\mathrm{true}\rangle\varphi$ = "After anything, possibly φ"

HML Patterns

Property	*Is satisfied by s iff.*
• $\langle\alpha\rangle$true	*s* can perform α
• $[\alpha]$false	*s* cannot perform α

- $\langle$true$\rangle$ true — *s* is not deadlocked
  - Same as $\langle\alpha_1\rangle$true $\vee$ $\langle\alpha_2\rangle$true $\vee$ … (for all actions α_i)
- [true] false — *s* is deadlocked
  - Same as $[\alpha_1]$false $\wedge$ $[\alpha_2]$false $\wedge$ … (for all actions α_i)

Exercises

Write the following properties in HML:

1. From this state, only action *a* should be performed
2. From this state we can perform the sequence "*a*, *b*"
3. After *a*, action *b* is forbidden

Solutions

Write the following properties in HML:

1. From this state, only action *a* should be performed
2. From this state we can perform the sequence "*a*, *b*"
3. After performing *a*, action *b* is forbidden

1. $\langle a \rangle$ true $\wedge$ $[\neg\alpha]$ false
2. $\langle a \rangle \langle b \rangle$ true
3. $[a][b]$ false

Limitations of HML

- With HML (as shown so far), we can only describe a finite part of an LTS (up to a certain depth from the initial state)
- Some properties cannot be captured in this way
  - Safety: "Something bad never happens"
  - Must hold for every state of an LTS (= at all depths)
  - Liveness: "Eventually, something good happens"
  - Every path starting from s0 must reach a state where the "good" thing happens (depth may vary)

Safety and liveness: examples

- Typical example of safety: deadlock freedom
  - Safety: it never happens that the process is “stuck”
- Typical example of liveness: starvation freedom
  - For instance, in mutual exclusion
  - A process starves if it never enters the critical section
  - Liveness: eventually, a process enters the CS

Invariance

Stronger version of safety

"F invariantly holds" (where F is an HML formula)

Inv(F) =

	F	(initial state satisfies F)
$\wedge$	[true] F	(all its successors satisfy F)
$\wedge$	[true][true] F	(*their* successors satisfy F)
$\wedge$	[true][true][true] F…	

Possibility

Weaker version of liveness

"F possibly holds" (where F is an HML formula)

$$
\begin{array}{lll}
\mathrm{Pos}(F) = & & \\
 & F & \text{(F holds in the initial state)} \\
\vee & \langle \mathrm{true} \rangle F & \text{(or in some successor)} \\
\vee & \langle \mathrm{true} \rangle\langle \mathrm{true} \rangle F & \text{(or in some of } \textit{their} \text{ successors)} \\
\vee & \langle \mathrm{true} \rangle\langle \mathrm{true} \rangle\langle \mathrm{true} \rangle F \ldots &
\end{array}
$$

Recursive HML formulas

Idea: we can use recursion

$$\mathrm{Inv}(F) = F \wedge [\mathrm{true}]\, F \wedge [\mathrm{true}][\mathrm{true}]\, F \wedge \ldots$$
$$= F \wedge [\mathrm{true}]\, \mathrm{Inv}(F)$$

$$\mathrm{Pos}(F) = F \vee \langle \mathrm{true} \rangle\, F \vee \langle \mathrm{true} \rangle \langle \mathrm{true} \rangle\, F \vee \ldots$$
$$= F \vee \langle \mathrm{true} \rangle\, \mathrm{Pos}(F)$$

- How do we solve such formulas?
- What do they mean?

Fixed points (1/2)

- Let S the set of states of some LTS
- Let f a function from subsets to subsets of states
  - We can define a f_φ for every rec. HML formula φ
- A fixed point for f is a set $X \subseteq S$ such that $X = f(X)$
- If φ does not contain $\neg X$, f_φ admits a unique minimal fp.and unique maximal fp.
- We will specify which one we want:
  - $\mu X.\varphi(X)$ for the minimal (“mu”)
  - $\nu X.\varphi(X)$ for the maximal (“nu”)
- s satisfies such a formula if it belongs to the fp.

Fixed points (2/2)

- How do we choose between μ and ν? Informally:
  - μ describes finite execution trees (liveness)
  - ν describes infinite execution trees (safety)

- Examples: invariance and possibility
  - $\mathrm{Inv}(F) = \nu X \,.\, F \wedge [\mathrm{true}]\, X$
  - $\mathrm{Pos}(F) = \mu X \,.\, F \vee \langle \mathrm{true} \rangle\, X$

- HML with μ/ν is known as the modal μ-calculus

Exercises

- "The system can never perform an *error* action"

- "There is a livelock in this state"
  - Remember: Livelock = infinite sequence of τ actions
  - Hint: Don't think about Inv, Pos, etc.

Solution

- "The system can never perform an *error* action"
  - Inv([error]false)
  - $\nu X\ .\ [error]false \wedge [true]X$
- "There may be a livelock in this state"
  - Remember: Livelock = infinite sequence of τ actions
  - Hint: Don't think about Inv, Pos, etc.

  - This state can do a τ action…
  - … and go to a state that may have a livelock (recursive)
  - $\nu X\ .\ \langle\tau\rangle X$

Exercise

- Possibly, the system may have a livelock
  - Remember $\nu X . \langle\tau\rangle X$ = “This state may have a livelock”

Solution

- Possibly, the system may have a livelock
  - Remember $\nu X . \langle\tau\rangle X$ = “This state may have a livelock”

  - Pos(Livelock)
  - Pos($\nu X . \langle\tau\rangle X$)
  - $\mu Y . (\nu X . \langle\tau\rangle X) \vee \langle true\rangle Y$

Regular formulas

- Similar to regular expressions
- Allows to express sequences of actions
- We can use these formulas within [], $\langle\rangle$
  - This makes some properties more compact/readable

- "The system can never perform an *error* action"
  - [true* . *error*] false
  - true* = a sequence of any length (*) of any action (true)
  - . = concatenation

Tool support

- CAAL:
  - basic HML
  - maximal/minimal fixed points (`X max=`, `X min=`)
- CADP:
  - Basic HML + regular formulas
  - maximal/minimal fixed points (`nu X.`, `mu X.`)
  - Other operators, such as infinite looping
  - “This state may have a livelock” : `<i> @`