Notion of Fairness

*Fairness:* a path $p$ is fair w.r.t. property $\psi$ if $\psi$ is true on $p$ infinitely often.

We may want to evaluate $A$ and $E$ constraints only over those paths.

Example: each process will run infinitely often: a process can stay in a critical section arbitrarily long, as long as it eventually leaves.

Two types of fairness: simple
  *Property $\phi$ is true infinitely often.*
and compound
  *If $\phi$ is true infinitely often, then $\psi$ is also true infinitely often.*

SMV can deal only with simple fairness.

---

**Formal Definition of Fairness**

Let $C = \{\psi_1, \psi_2, \ldots, \psi_n\}$ be a set of $n$ fairness constraints. A computation path $s_0, s_1, \ldots$ is fair w.r.t. $C$ if for each $i$ there are infinitely many $j$ s.t. $s_j \models \psi_i$, that is, each $\psi_i$ is true infinitely often along the path.

We use $A_C$ and $E_C$ for the operators $A$ and $E$ restricted to fair paths.

$E_C U$, $E_C G$ and $E_C X$ form an adequate set.

Also, a path is fair iff any suffix of it is fair. Finally,

$$E_C[\phi U \psi] = E[\phi U(\psi \land E_C G \phi)]$$

$$E_C X \phi = EX(\phi \land E_C G \phi)$$

We only need an algorithm for $E_C G \phi$. 
Algorithm for $E_C G \phi$

- Restrict the graph to states satisfying $\psi_i$ of the resulting graph, we want to know from which states there is a fair path.
- Find the maximal strongly connected components (SCCs) of the restricted graph;
- Remove an SCC if, for some $\psi_i$, it does not contain a state satisfying $\psi_i$. The resulting SCCs are the fair SCCs. Any state of the restricted graph that can reach one has a fair path from it.
- Use backwards breadth-first searching to find the states on the restricted graph that can reach a fair SCC.

Complexity: $O(n \times |f| \times (S + |R|))$ (still linear in the size of the model and formula).

Guidelines for Modeling with SMV

- Identify inputs from the environment.
- Make sure that the environment is non-deterministic. All constraints on the environment should be carefully justified.
- Determine the states of the system and its outputs. Model them in terms of the environmental inputs.
- Specify fairness criteria, if any. Justify each criterion. Remember that you can over-specify the system. Fairness may not be implementable, and in fact may result in no behaviors.
- Specify correctness in CTL. Comment each CTL property in English.
- Ensure that CTL properties are not satisfied vacuously. That is, each universally-quantified property should be paired up with an existentially-specified property. Also check that LHSs of implications are not always false.

Examples:

AG (a) —
AG (a → b) —
**Symbolic model checking**

Why?
Saves is from constructing a model's state space. Effective "cure" for state space explosion problem.

How?
Sets of states and transition relations are represented by formulas, and set operations are defined in terms of formula manipulations.

Data structures
BDDs - allow for efficient storage and manipulation of logic formulas.

**Digression: Fixpoint Computation**

Definition: Let $S$ be a set of states and $F : \mathcal{P}(S) \rightarrow \mathcal{P}(S)$ a function on the power set of $S$.
1. We say that $F$ is monotone if $X \subseteq Y$ implies $F(X) \subseteq F(Y)$ for all subsets $X$ and $Y$ of $S$.
2. A subset $X$ of $S$ is called a fixed point (fixpoint) of $F$ if $F(X) = X$.

Example: Let $S = \{s_0, s_1\}$ and $F(Y) = Y \cup \{s_0\}$ for all subsets $Y$ of $S$. $F$ is monotone, because $Y \subseteq Y'$ implies $Y \cup \{s_0\} \subseteq Y' \cup \{s_0\}$. $F$ has two fixpoints: $\{s_0\}$ (least) and $\{s_0, s_1\}$ (greatest).

Example: Let $G(Y) =$ if $Y = \{s_0\}$ then $\{s_1\}$ else $\{s_0\}$
$G$ is not monotone (e.g., $\{s_0\} \subseteq \{s_0, s_1\}$, but $G(\{s_0\}) \not\subseteq G(\{s_0, s_1\})$). $G$ does not have any fixpoints!
Fixpoints and Model Checking

- Monotone functions always have a least and a greatest fixpoint.
- Meaning of EG, AF, EU can be expressed via greatest or least fixpoints of monotone functions on \( P(S) \).

Notation: \( F^i(X) \) means that \( F \) is applied \( i \) many times

\( \text{Ex: } F(Y) = Y \cup \{s_0\} \).
Then \( F^2(Y) = F(F(Y)) = ... \)

When will fixpoints be achieved? The Knaster-Tarski Theorem:
Let \( S \) be a set \( \{s_0, s_1, ..., s_n\} \) with \( n+1 \) elements. If \( F : \mathcal{P}(S) \rightarrow \mathcal{P}(S) \) is a monotone function, then \( F^{n+1}(\emptyset) \) is the least fixpoint of \( F \) and \( F^{n+1}(S) \) is the greatest fixpoint of \( F \).
For proof, see section 3.9 of the book.

More notation: \( \mu y.F(y) \) means the least fixpoint of \( F(y) \); \( \nu y.F(y) \) means the greatest fixpoint of \( F(y) \).

Example

Model checking \( s_0 \models EF(\neg a \land \neg b) \).

1. Model

2. \((\neg a \land \neg b) \lor EX(\text{false})\)

3. \((\neg a \land \neg b) \lor EX(\neg a \land \neg b)\)

4. \((\neg a \land \neg b) \lor EX((\neg a \land \neg b) \lor EX(\neg a \land \neg b))\)
Symbolic Model Checking

- A system state represents an interpretation (truth assignment) for a set of propositional variables \( V \).

- Formulas represent sets of states that satisfy it.
  \( a \) - set of states in which \( a \) is true - \( \{s_0, s_1\} \)
  \( b \) - set of states in which \( b \) is true - \( \{s_1, s_2\} \)
  \( a \lor b = \{s_0, s_1, s_2\} \)

- State transitions are described by relations over two sets of variables, \( V \) (source state) and \( V' \) (destination state).
  Transition from \( s_2 \) to \( s_3 \) is described by
  \( (\neg a \land b \land \neg a' \land \neg b') \).
  Transition from \( s_0 \) to \( s_1 \) and \( s_2 \), and from \( s_1 \) to \( s_2 \) and to itself is described by \( (a \land b') \).
  Relation \( R \) is described by
  \( (a \land b') \lor (\neg a \land b \land \neg a' \land \neg b') \)

Symbolic model checking
(Cont’d)

The meaning for CTL formulas can be redefined in terms of sets of states:

\[
\begin{align*}
s \models f & \iff s \in f \text{ where } f \in V \\
s \models \neg f & \iff s \in \neg f \\
s \models f \lor g & \iff s \in (f \lor g) \\
s \models \text{EX} f & \iff s \in (\exists V'(R \land f(V/V'))) \\
s \models \text{AX} f & \iff s \in \neg(\exists V'(R \land \neg f(V/V'))) \\
s \models \text{E}(fUg) & \iff s \in \mu y.(g \lor (f \land \text{EX} \neg y)) \\
s \models \text{A}(fUg) & \iff s \in \mu y.(g \lor (f \land \text{AX} y))
\end{align*}
\]
Symbolic model checking

- A CTL formula $f$ is evaluated for a model by deriving a propositional logic expression that describes the set of states satisfying the CTL formula for the model.
- The model checker verifies that the interpretation of the model’s initial state $s_0$ satisfies the expression.

Example - check $s_0 \models EX(\neg a \land \neg b)$, i.e., compute a formula representing the states that have successors where $(\neg a \land \neg b)$ is true:
- $R$ - the transition relation
- $f$ - the formula being checked
- $f(a, b/a', b')$ - substitution, leading to reasoning about the next state
- Replace formulas like $\exists v, (f)$ by $(f(v/true) \lor f(v/false))$.

Computation

\[
EX(\neg a \land \neg b) = \\
\exists a', b'((a \land b') \lor (\neg a \land b \land a' \land \neg b')) \\
\land((\neg a \land \neg b)(a, b/a', b')) = \\
\exists a', b'(((a \land b') \lor (\neg a \land b \land a' \land \neg b')) \\
\land(\neg a' \land \neg b')) = \\
\exists a', b'((a \land b' \land \neg a' \land \neg b') \\
\lor(\neg a \land b \land \neg a' \land \neg b')) = \\
\exists a', b'(false) \lor (\neg a \land b \land \neg a' \land \neg b') = \\
\exists a', b'((\neg a \land b \land \neg a' \land \neg true) \lor \\
(\neg a \land b \land \neg a' \land \neg false)) = \\
\exists a'(\neg a \land b \land \neg a') = (\neg a \land b \land \neg true) \\
\lor(\neg a \land b \land false) = \neg a \land b
\]
Evaluation of example

The computed propositional logic formula represents the set of states whose interpretations satisfy $\text{EX}(\neg a \land \neg b)$, that is, $\{s_2\}$.

$\text{EX}(\neg a \land \neg b)$ is a theorem (i.e., $s_0 \models \text{EX}(\neg a \land \neg b)$) if the values of $a$ and $b$ in $s_0$ satisfy $\neg a \land \neg b$.

In $s_0$, $a \equiv \text{true}$, $b \equiv \text{false}$. So, $s_0 \not\models \neg a \land \neg b$. Thus, $s_0 \not\models \text{EX}(\neg a \land \neg b)$.

Symbolic model checking

Example: calculate $\text{EF}(\neg a \land \neg b)$ for our model:

1) $(\neg a \land \neg b) \lor \text{EX} \ false = (\neg a \land \neg b)$
2) $(\neg a \land \neg b) \lor \text{EX} (\neg a \land \neg b) = (\neg a \land \neg b) \lor (\neg a \land b) = \neg a$
3) $(\neg a \land \neg b) \lor \text{EX} (\neg a) =$
   $(\neg a \land \neg b) \lor \exists \ a', b'(((a \land b') \lor (\neg a \land b \land \neg a' \land \neg b'))$
   $\land (((a/a')))) =$
   $(\neg a \land \neg b) \lor \exists \ a', b'(((a \land b') \lor (\neg a \land b \land \neg a' \land \neg b')) \land \neg a' =$
   $(\neg a \land \neg b) \lor \exists \ a', b'(((a \land b' \land \neg a') \lor (\neg a \land b \land \neg a' \land \neg b')) =$
   $(\neg a \land \neg b) \lor \exists \ a'(((a \land \text{true} \land \neg a') \lor (\neg a \land b \land \neg a' \land \text{true})$
   $\lor (((a \land \text{false} \land \neg a') \lor (\neg a \land b \land \neg a' \land \text{false})))) =$
   $(\neg a \land \neg b) \lor \exists \ a'(((a \land \neg a') \lor (\neg a \land b \land \neg a' \land \neg true)) =$
   $(\neg a \land \neg b) \lor (a \land \text{false}) \lor ((a \land \neg false) \lor (\neg a \land b \land \neg false)) =$
   $(\neg a \land \neg b) \lor ((\text{false} \lor (false)) \lor (a) \lor (a \land b)) =$
   $(\neg a \land \neg b) \lor (\neg a \land b) \lor a = \text{true}$
4) $(\neg a \land \neg b) \lor \text{EX} (\text{true}) = \text{true}$
Symbolic model checking

Note that the formulas at the end of each step correspond to the set of states that is shaded after that step.

- The first formula, \( \neg a \land \neg b \), corresponds to \( \{ s_3 \} \)
- The second formula, \( \neg a \), corresponds to \( \{ s_2, s_3 \} \)
- The last formula, \( true \), corresponds to the set of all states.

Pros and Cons

- Often cannot express full requirements
  - instead, check several smaller properties
- Few real systems have sufficiently small state space to allow direct checking
  - must generally abstract them or "down-scale" them. Abstractions may enable checking systems with virtually unlimited number of states
- Largely automatic and fast
- Produces counterexamples
- Can handle systems with 100-200 state variables
- Generally used for debugging rather than assurance
- Usually, find more problems by exploring all the behaviors of a downscaled system than by testing only some of the behaviors of full system.
Model-checking Examples (Up to 1996)

IEEE Futurebus+ (1992, Clarke, CMU)
- Used SMV to verify the cache coherence protocol described in IEEE Futurebus+ Standard 896.1-1991
- Discovered a number of previously undetected errors in the design
- First attempt to find errors in an IEEE standard

IEEE SCI (1992, Dill, Stanford)
- Murphi verification system
- Verified cache coherence protocol of Scalable Coherent Interface, IEEE Standard 1596-1992
- Based their model directly on the C code given as definition of the standard
- Verified just small instances of the system
- Found several errors, ranging from omissions of variable initializations to subtle logical errors
- The protocol has already been simulated and implemented!!

Model-checking Examples (Cont’d)

Stereo Components (1994-96)
- 1994. Proved (manually) correctness of a control protocol used in Philips stereo components
- 1995. Verified an abstraction of protocol using symbolic model checker HyTech and inferred more efficient timing of the protocol
- 1996. Model checked the entire protocol

ISDN/ISUP (1989-92)
- NewCoRe Project - first full-scale application of formal verification methods in routine software design project within AT&T
- 5 people formalized 145 requirements in temporal logic and did proofs with a special-purpose model checker
- 7500 lines of Specification and Description Language (SDL) source code verified
- 112 errors were revealed in high-level designs
Model-checking Examples (Cont’d)

PowerScale (1995, Bull with Berimag Laboratory)
- Used LOTOS to describe processors, memory controller, and bus arbiter of the multiprocessor architecture PowerScale
- Architecture is based on IBM’s Power PC microprocessor
- Identified four correctness properties expressing essential requirements for proper functioning of the arbitration algorithm
- Established correctness automatically in a few minutes

Model-checking Examples (Cont’d)

Buildings (1995, North Caroline State)
- Used Concurrency Workbench to analyze timing properties of a distributed active structural control system
- System was designed to make buildings more resistant to earthquakes
- First coded in a timed version of the CCS. Resulting model contained over $2.12 \times 10^{19}$ states and was not analyzable
- Applied semantic minimization feature of Concurrency Workbench
- Analyzed properties and uncovered an error in a timer setting
- If undetected, this error would cause active structural control component to worsen, rather than dampen, the vibration experienced by buildings during earthquakes
State of the Art Model Checkers

Temporal logic model checkers

- EMC - Clarke and Emerson 81-86
- CESAR - Queille and Sifakis 1982
- SMV - McMillan 1993 - first to use BDDs
- Spin - Gerth et al. 1995 - uses partial order reduction to reduce the state explosion problem
- Murphi - Dill et al. 1992 is based on Unity programming language
- Concurrency Workbench - Cleaveland et al. 1993 - verifies CCS processes for properties expressed as mu-calculus formulas
- HyTech - Alur et al. 1996 - for hybrid systems
- Kronos - Daws and Yovine 1995 - for real-time systems
- FORMAT - Damm et al. 1996, SVE - Filkorn et al. 1994 and CV - Deharbe and Borriione 1995 - focus on hardware verification

State of the Art Model Checkers
(Cont’d)

Behavior conformance checkers

- Cospan/FormalCheck system - DePalma and Glaser 1996 - based on showing inclusion between omega automata
- FDR - Roscoe 1994 - checks refinement between CSP programs
- Concurrency Workbench checks a similar notion of refinement between CCS programs
- Concurrency Workbench and Auto - Roy and de Simone 1990 - can be used to minimize systems with respect to observational equivalence and to determine if two systems are observably correct
State of the Art Model Checkers
(Cont’d)

Combination checkers

- Berkeley’s HSIS - Hojati et al. 1993 - combines model checking with language inclusion
- Stanford’s STep - Björner et al. 1996 - with deductive methods
- VIS - Vayton et al. 1996 - with logic synthesis
- PVS Theorem prover - Owre, Rushby, Shankar 1992
- has a model checker for the model mu-calculus
- METAFram – Steff et al. 1996 - an environment that supports model checking in the entire software development process.