Model Checking

Gianpiero Cabodi

Outline

- Propositional Logic
- First-Order-Logic
- Temporal Logic (LTL)
- Temporal Logic (BTTL-CTL)
- Model Checking

Propositional Logic (calculus)

Syntax

P, Q, R,... — propositional symbols (atomic propositions)

t: true; f: false — constants

$\neg$P: not P   P $\wedge$ Q: P and Q   P $\vee$ Q: P or Q;

P $\rightarrow$ Q: if P then Q (proposition equivalent to $\neg$P$\vee$Q)

P $\leftrightarrow$ Q: P if and only if Q, i.e., P equivalent to Q
(proposition equivalent to (P$\wedge$Q)$\vee$($\neg$P$\wedge\neg$Q))

Propositional Logic (calculus)

Semantics

Given through the Truth Table:

P	Q	$\neg$P	P$\wedge$Q	P$\vee$Q	P$\rightarrow$Q	P$\leftrightarrow$Q
t	t	f	t	t	t	t
t	f	f	f	t	f	f
f	t	t	f	t	t	f
f	f	t	f	f	t	t

An **interpretation** is a function from the propositional symbols to {t, f}

Propositional Logic

- Formula F is **satisfiable** (consistent) iff it is **true** under **at least one** interpretation
- Formula F is **unsatisfiable** (inconsistent) iff it is **false** under **all** interpretations
- Formula F is **valid** iff it is **true** (consistent) under **all** interpretations
- *Interpretation I satisfies a formula F* (*I is a **model** of F*) iff F is true under I.
  Notation: I $\models$ F
- **Theorem:** A formula F is valid (a ***tautology***) iff $\neg$F is unsatisfiable. Notation: $\models$ F

Propositional Logic

- The relationship between F to $\neg$F can be visualized by "mirror principle":

All formulas in propositional logic

Valid formulas	Satisfiable, but non-valid formulas	Unsatisfiable formulas
G ◄- - - -	- - - F ◄- - - -► $\neg$F - - -	- - - -► $\neg$G

- To determine if F is satisfiable or valid, test finite number (2^n) of interpretations of the n atomic propositions occurring in F
  ... but it is an exponential method... **satisfiability is an NP-complete problem**

First-Order-Logic

- *Propositional logic*: reasoning about complete sentences.
- *First-order logic*: also reasoning about *individual objects* and *relationships between them*.

Syntax

- **Objects** (in FOL) are denoted by expressions called ***terms***:
  Constants a, b, c,... ; *Variables* u, v, w,... ;
  $f(t_1, t_2,..., t_n)$ where $t_1, t_2,..., t_n$ are terms and f a *function symbol* of n arguments
- **Predicates**:
  true (T) and *false* (F)
  $p(t_1, t_2,..., t_n)$ where $t_1, t_2,..., t_n$ are *terms* and p a *predicate symbol* of n arguments
- **Formulas**:
  Predicates
  P and Q formulas, then $\neg P$, $P \wedge Q$, $P \vee Q$, $P \rightarrow Q$, $P \leftrightarrow Q$ are formulas
  x a variable, P a formula, then $\forall x.P$, $\exists x.Q$ are formulas (x is not free in P, Q)

First-Order-Logic

Semantics of a first-order logic formulae G: interpretation for function, constant and predicate symbols in G and assigning values to free variables

First-Order Interpretations (Structures) M: M = (D, I)

- D is a non-empty domain of the structure
- I is an interpretation function, assigns function, constant and predicate symbols:
  (1) For every function symbol f of rank n>0, $I(f): D^n \rightarrow D$ is an n-ary function.
  (2) For every constant c, I(c) is an element of D.
  (3) For every predicate symbol P of rank n≥0, $I(P): D^n \rightarrow \{F, T\}$ is an n-ary predicate.

Evaluation

- For every M, a formula can be evaluated to T or F according to the following rules:
  (1) Evaluate truth values of formulas P and Q, and then the truth values of
  $\neg P$, $P \wedge Q$, $P \vee Q$, $P \rightarrow Q$, $P \leftrightarrow Q$ using propositional logic
  (2) $\forall x$. P evaluates to T if truth value of G is T for every $d \in D$; otherwise, it is F
  (3) $\exists x$.P evaluates to T if truth value of G is T for at least one $d \in D$; otherwise, it is F

First-Order Logic (Predicate Calculus)

- First-Order Logic speaks about objects, which are the domain of discourse or the universe.
- First-Order Logic is also concerned about Properties of these objects (called Predicates), and the Names of these objects.
- Also we have Functions of objects and Relations over objects. (For example, Socrates' father is a function of Socrates, while Socrates' son(s) is a relation about the object Socrates). (Properties would be then mapped to relations on objects).

Syntax of First-Order Logic

- Using functions and relations, and using the notation x_i to denote a variable (of type Object) to name individuals (so that we have a set of Vars = $\{x_1, x_2, \ldots, x_n\}$) we could define the syntax of formulas in First-Order Logic.

Terms & Formulas

- Terms of First-Order Logic formulas are defined recursively as follows:
  - Vars $\subseteq$ Terms
  - If $t_1, t_2, \ldots, t_k \in$ Terms and f is a k-ary function name, then $f(t_1, t_2, \ldots, t_k) \in$ Terms
- Formulas of First-Order Logic could be defined as:
  - If $t_1, t_2, \ldots, t_k \in$ Terms and P is a k-ary relation name
  - then $P(t_1, t_2, \ldots, t_k)$ is an atomic formula
  - If θ,ψ are formulas then $(\neg\theta)$, $(\theta\wedge\psi)$ are formulas.
  - If θ is a formula and $x \in$Vars then $(\exists x)\ \theta$ and $(\forall x)\ \theta$ are formulas.

An Example

$((\forall x(H(x)\rightarrow M(x))\wedge(\exists x)(G(x)\wedge H(x)))$
$\rightarrow (\exists\ x)(G(x) \wedge M(x))$

If all humans are mortal and some Greeks are human then some Greeks are mortal.

Hierarchy of Logic

- First-Order logic is concerned about objects
  - logic quantifiers (∀,∃) quantify over elements (objects).
- Second-order logic: elementary elements are functions and relations (i.e., sets of objects)
- Third-order logic: main objects are sets of sets of objects.
  - logic quantifiers (∀,∃) quantify over relations and functions.

Higher Order Logic

- **Example 1:** (mathematical induction)
  - $\forall P.\ [P(0) \wedge (\forall n.\ P(n) \rightarrow P(n+1))] \rightarrow \forall n.P(n)$ (Impossible to express it in FOL)
- **Example 2:** Function Rise defined as
  - Rise (c, t) = $\neg c(t) \wedge c(t+1)$

  Rise expresses the notion that a signal *c* rises at time *t*.

  Signal is modeled by a function c: $N \rightarrow \{F,T\}$, passed as argument to Rise.

  Result of applying Rise to c is a function: $N \rightarrow \{F,T\}$.

Higher Order Logic

- **Advantage:** high expressive power!
- **Disadvantages:**
- Incompleteness of a sound proof system for most higher-order logics
  - **Theorem** (Gödel, 1931) *There is no complete deduction system for the second-order logic.*
- Reasoning more difficult than in FOL, need ingenious inference rules and heuristics.
- Inconsistencies can arise in higher-order systems if semantics not carefully defined
  - "*Russell Paradox*": Let P be defined by $P(Q) = \neg Q(Q)$. By substituting P for Q, leads to $P(P) = \neg P(P)$,

    (P: bool $\rightarrow$ bool, Q: bool $\rightarrow$ bool) contradiction!

Temporal Logics

- Temporal logic is a type of modal logic that was originally developed by philosophers to study different *modes* of "truth"
- Temporal logic provides a formal system for qualitatively describing and reasoning about how the truth values of assertions change *over time*
- It is appropriate for describing the time-varying behavior of systems (or programs)

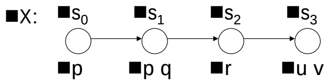
Classification of Temporal Logics

- The underlying nature of time:
  - ***Linear***: at any time there is only one possible future moment, linear behavioral trace
  - ***Branching***: at any time, there are different possible futures, tree-like trace structure
- Other considerations:
  - Propositional vs. first-order
  - Point vs. intervals
  - Discrete time vs. continuous time
  - Past vs. future

Linear Temporal Logic

- Time lines
  - Underlying structure of time is a totally ordered set (S,<), isomorphic to (N,<): Discrete, an initial moment without predecessors, infinite into the future.
- Let AP be set of atomic propositions, a *linear time structure* M=(S, x, L)
  - S: a set of states
  - x: $N \rightarrow S$ an infinite sequence of states, $(x=s_0,s_1,\ldots)$
  - L: $S \rightarrow 2^{AP}$ labeling each state with the set of atomic propositions in AP true at the state.

Example

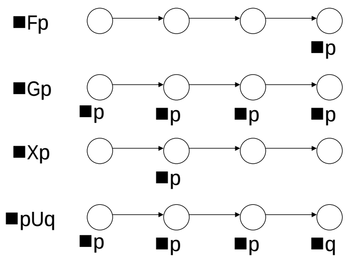


- AP = {p, q, r, u, v}
- $L(s_0)$ = {p}, $L(s_1)$ = {p, q}, $L(s_2)$ = {r}, $L(s_3)$ = {u, v},.......

Propositional Linear Temporal Logic (PLTL)

- Classical propositional logic + temporal operators
- **Basic temporal operators**
  - F*p* ("eventually *p*", "sometime *p*")
  - G*p* ("always *p*", "henceforth *p*")
  - X*p* ("next time *p*")
  - *p*U*q* ("*p* until *q*")
- Other common notation:
  - G = $\Box$
  - F = $\Diamond$
  - X = O

Examples



PLTL Syntax

- The set of formulas of PLTL is the least set of formulas generated by the following rules:
  - Atomic propositions are formulas,
  - *p* and *q* formulas: $p \wedge q$, $\neg p$, *p*U*q*, and X*p* are formulas.
- The other formulas can be introduced as abbreviations:
- $p \vee q$ abbreviates $\neg(\neg p \wedge \neg q)$
- $p \Rightarrow q$ abbreviates $\neg p \vee q$
- $p \equiv q$ abbreviates $(p \Rightarrow q) \wedge (q \Rightarrow p)$,
- *true* abbreviates $p \vee \neg p$,
- *false* abbreviates ¬*true*,
- F*p* abbreviates (*true* U *p*),
- G*p* abbreviates ¬F¬*p*.

Examples

- *$p \Rightarrow$ Fq: "if p is true now then at some future moment q will be true."*
- *G($p \Rightarrow$ Fq): "whenever p is true, q will be true at some subsequent moment."*

PLTL semantics

Semantics of a formula *p* of PLTL with respect to a linear-time structure M=(S, x, L)

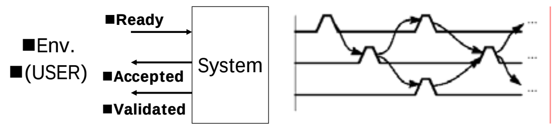
- (M, x) $\models$ *p* means that "in structure M, formula *p* is true of timeline x."
- x^i: suffix of x starting at s_i, $x^i = s_i, s_{i+1}, \ldots$
- Semantics
  - (M, x) $\models$ p iff p $\in L(s_0)$, for atomic proposition p
  - (M, x) $\models p \wedge q$ iff (M, x) $\models p$ and (M, x) $\models q$
  - (M, x) $\models \neg p$ iff it is not the case that (M, x) $\models p$
  - (M, x) $\models$ X*p* iff $x^1 \models p$
  - (M, x) $\models$ F*p* iff $\exists j.(x^j \models p)$
  - (M, x) $\models$ G*p* iff $\forall j.(x^j \models p)$
  - (M, x) $\models p$ U q iff $\exists j.(x^j \models q$ and $\forall$ k, $0 \le k < j$ $(x^k \models p))$

PLTL semantics

- Duality between linear temporal operators
  - $\models G\neg p \equiv \neg Fp$
  - $\models F\neg p \equiv \neg Gp$
  - $\models X\neg p \equiv \neg Xp$
- PLTL formula p is *satisfiable* iff there exists M=(S, x, L) such that $(M, x) \models p$ (any such structure defines a ***model*** of p).

Example

A simple interface protocol, pulses one clock period wide



Example

- Safety property — nothing bad will ever happen:
- **$\forall$t. (Validated (t) $\Rightarrow$ $\neg$Validated (t + 1)**)
- **$\Box$ (Validated $\Rightarrow$ O $\neg$Validated**)
- **G (Validated $\Rightarrow$ X $\neg$Validated**)
- Liveness property — something good will eventually happen:
- **$\forall$t. Ready (t) $\Rightarrow$ $\exists$t' $\geq$ t.Accepted (t')**
- **$\Box$ (Ready $\Rightarrow$ $\Diamond$Accepted)**
- **G (Ready $\Rightarrow$ F Accepted)**

Constraints

- Fairness constraint:
  - **G**(*Accepted* $\Rightarrow$ **F** *Ready*)

  models a live environment for System
- Behavior of environment (constraint):
  - **G** (*Ready* $\Rightarrow$ **X**($\neg$*Ready* **U** *Accepted*))
- What about other properties of *Accepted* (initial state, periodic behavior), etc.?
  - Prove the system property under the assumption of valid environment constraints

Branching Time Temporal Logic (BTTL)

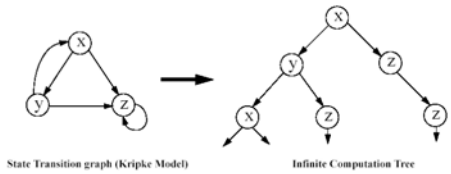
- **Structure** of time: **an infinite tree**, each instant may have many successor instants. Along each path in the tree, the corresponding timeline is isomorphic to **N**
- **State quantifiers**: Xp, Fp, Gp, pUq (like in linear temporal logic)
- **Path quantifiers:** *for All paths* (A) and *there Exists a path* (E) from a given state
- Other frequent notation:
  - G = $\Box$
  - F = $\Diamond$
  - X = O
  - A = $\forall$
  - E = $\exists$

BTTL and LTL

- In linear time logic, temporal operators describe events along a single future, however, when a linear formula is used for specification, there is usually an *implicit universal quantification* over all possible futures (linear traces)
- In contrast, in branching time logic the operators usually reflect the branching nature of time by allowing *explicit quantification* over possible futures in any state
  - One supporting argument for branching time logic is that it offers the ability to reason about *existential* properties in addition to *universal* properties
  - But, it requires some knowledge of internal state for branching, closer to implementation than LTL that describes properties of observable traces and has simpler fairness assumptions

CTL: a BTTL

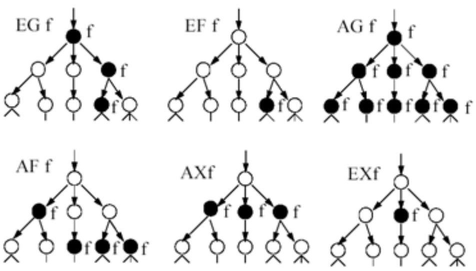
- CTL = Computation Tree Logic
- Example of Computation Tree:
  - Paths in the tree = possible computations or behaviors of the system



CTL syntax

- Every atomic proposition is a CTL formula
- If f and g are CTL formulas, then so are $\neg$f, f$\wedge$g, AXf, EXf, A(fUg), E(fUg)
- Other operators:
  - AFg = A(true U g)   EFg = E(true U g)
  - AGf = $\neg$E(true U $\neg$f)   EGf = $\neg$A(true U $\neg$f)
- EX, E(...U...), EG are sufficient to characterize the entire logic:
  - EFp = E(true U p)
  - AXp = $\neg$EX$\neg$p
  - AGp = $\neg$EF$\neg$p
  - A(qUp) = $\neg$(E(($\neg$p U$\neg$q)$\wedge$$\neg$p)$\vee$EG$\neg$p)

Intuitive Semantics of Temporal Operators

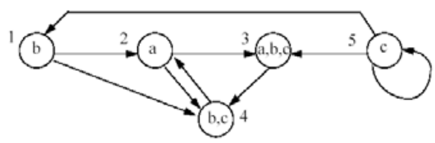


CTL semantics

- A *Kripke structure*: triple M = <S, R, L>
  - S: set of states   R $\subseteq$ S x S: transition relation
  - L: S $\rightarrow$ 2^{AP}: (Truth valuation) set of atomic propositions true in each state
- R is *total*:
  - $\forall$s $\in$S there exists a state s' $\in$ S such that (s, s') $\in$ R
- *Path* in M:
  - infinite sequence of states, x = s_0, s_1, ... , i $\geq$ 0, (s_i, s_{i+1}) $\in$ R
- x^i: suffix of x starting at s_i, x^i = s_i, s_{i+1}, ...
- xi denotes the suffix of x starting at si: xi = si, si+1, ...

- Truth of a CTL formula is defined inductively:
  - (M, s_0) $\models$ p iff p $\in$ L(s_0), for atomic proposition p
  - (M, s_0) $\models$ $\neg$p iff (M, s_0) $\not\models$ p
  - (M, s_0) $\models$ p $\wedge$ q iff (M, s_0) $\models$ p and (M, s_0) $\models$ q
  - (M, s_0) $\models$ AX*p* iff $\forall$ states t, (s_0,t) $\in$ R, (M,t) $\models$ *p*
  - (M, s_0) $\models$ EX*p* iff $\exists$ states t, (s_0,t) $\in$ R, (M,t) $\models$ *p*
  - (M, s0) $\models$ A (p U q) iff $\forall$ x = s_0, s_1, s_2, ..., $\exists$ j $\geq$ 0, (M, s_j) $\models$ q and $\forall$ k, 0$\leq$k<j, (M, s_k) $\models$ p
  - (M, s0) $\models$ E (p U q) iff $\exists$ x = s_0, s_1, s_2, ..., $\exists$ j $\geq$ 0, (M, s_j) $\models$ q and $\forall$ k, 0$\leq$k<j, (M, s_k) $\models$ p

Example Structure M <S,R,L>



- S = {1,2,3,4,5}, AP = {a,b,c},
- R = {(1,2), (2,3), (5,3), (5,5), (5,1), (2,4), (4,2), (1,4), (3,4)}
- L(1) = {b}, L(2) = {a}, L(3) = {a,b,c}, L(4) = {b,c}, L(5) = {c}

Example CTL formulas

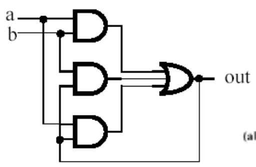
- EF(*started* $\wedge\neg$ *ready*): possible to get to a state where *started* holds but *ready* does not
- AG(*req* $\Rightarrow$ AF *ack*): if a *request* occurs, then there is eventually an *acknowledgment* (does not ensure that the number of *req* is the same as that of *ack* !)
- AG(AF *enabled*): *enabled* holds infinitely often on every computation path
- AG(EF *restart*): from any state it is possible to get to the *restart* state

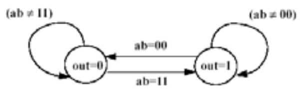
CTL*

- Computational tree logic CTL* combines branching-time and linear-time operators
- CTL* is sometimes referred to as **full branching-time logic**
- In CTL each linear-time operators G, F, X, and U must be immediately preceded by a path quantifier
- In CTL* a path quantifier can prefix an assertion composed of **arbitrary combinations of the usual linear-time operators (**F, G, X and U**)**
- Example: EFp is a basic modality of CTL; E(Fp $\wedge$ Fq) is a basic modality of CTL*

Example

Two input Muller C-element (assuming finite discrete delays)

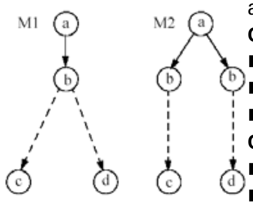




Example: **Specification in CTL**

- **Liveness**: *If inputs remain equal, then eventually the output will change to this value*.
- AG(A((a=0 $\wedge$ b=0) U (out=0 $\vee$ a=1 $\vee$ b=1)))
- AG(A((a=1 $\wedge$ b=1) U (out=1 $\vee$ a=0 $\vee$ b=0)))
- **Safety**: *If all inputs and the output have the same value then the output should not change until all inputs change their values.*
- AG((a=0 $\wedge$ b=0 $\wedge$ out=0) $\Rightarrow$ A(out=0 U (a=1 $\wedge$ b=1)))
- AG((a=1 $\wedge$ b=1 $\wedge$ out=1) $\Rightarrow$ A(out=1 U (a=0 $\wedge$ b=0)))
- What about the environment? It may have to be constrained to satisfy some **fairness**!

Linear vs. Branching Time TL



Trace set is the same in both M1 and M2: { ab... c, ab... d }

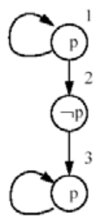
Characterization by LTL

- [a$\wedge$X(b$\wedge$Fc)]$\vee$[a$\wedge$X(b$\wedge$Fd)] =
- a$\wedge$X (b$\wedge$(F(c$\vee$d))) =
- a$\wedge$X (b$\wedge$(F(c)$\vee$F(d)))

Characterization by CTL

- M1 and M2: a$\wedge$AX(b$\wedge$(AF(c$\vee$d)))
- M2 only: a$\wedge$AX(b$\wedge$(AF(c)$\vee$AF(d)))

Linear vs. Branching Time TL



- In LTL the property F(G p) holds ((on all paths) eventually always p), but
- In CTL this cannot be expressed: AF(AG p) does not hold, because of an infinite run on state 1. AG p holds on state 3 only, i.e., in state 1 the next state is either 1 or 2, the selfloop satisfies G p, but the transition to 2 (and then to 3) does not satisfy G p, hence AG p does not hold

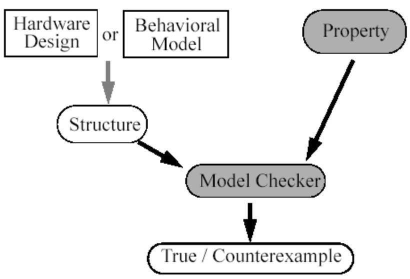
Linear vs. Branching Time TL

- LTL
  - easier inclusion of fairness constraints as preconditions in the same LTL language
  - AG EF p cannot be expressed
  - complexity of model checking: *exponential* in the length of the formula
- CTL
  - fairness properties GF p $\Rightarrow$ GF q not expressible
  - fairness constraints often specified using exception conditions H_i: computation paths along which states satisfy every $!(H_i)$ $(1 \leq i \leq n)$ infinitely often
  - complexity of model checking: *deterministic polynomial*

Model Checking Problem for Temporal Logic

- Given an FSM M (equivalent Kripke structure) and a temporal logic formula p, does M define a model of p?
  - Determine the truth of a formula with respect to a given (initial) state in M
  - Find all states s of M such that (M, s) p
- For any **propositional** temporal logic, the model checking problem is **decidable:** exhaustive search of all paths through the finite input structure

Structure of Model Checker



Structure of Model Checker

- Specification Language: CTL
- Model of Computation: Finite-state systems modeled by labeled state-transition graphs
  - (***Finite Kripke Structures***)
- If a state is designated as the *initial state*, the structure can be unfolded into an infinite tree with that state as the root: *Computation Tree*

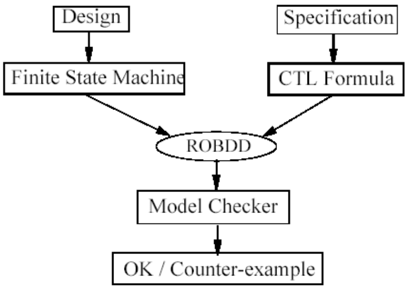
Model Checking Algorithms

- Original algorithm described in terms of *labeling* the CTL structure (Clark83)
  - Required explicit representation of the whole state space
- Better algorithm based on *fixed point* calculations
- Algorithm amenable to *symbolic* formulation
  - Symbolic evaluation allows implicit enumeration of states
  - Significant improvement in maximum size of systems that can be verified

Symbolic Model Checking

- Explicit State Representation. State Explosion Problem (about 10^8 states maximum)
- Breakthrough: Implicit State Representation using ROBDD (about 10^{20} states).
- Use Boolean characteristic functions represented by ROBDDs to encode sets of states and transition relations.

Symbolic Model Checking



Model Checking Tools

- **SMV (Symbolic Model Verifier)**
  - A tool for checking finite state systems against specifications in the temporal logic CTL.
  - Developed at Carnegie Mellon University by E. Clarke, K. McMillan et. al.
  - Supports a simple input language: SMV
  - For more information: http://www.cs.cmu.edu/~modelcheck/smv.html

Model Checking Tools

- **Cadence SMV**
- Updated version of SMV by K. McMillan at Berkeley Cadence Labs
- Input languages: extended SMV and synchronous Verilog
- Supports temporal logics CTL and LTL, finite automata, embedded assertions, and refinement specifications.
- Features compositional reasoning, link with a simple theorem prover, an easy-to-use graphical user interface and source level debugging capabilities
- For more information: http://www-cad.eecs.berkeley.edu/~kenmcmil/smv/

Model Checking Tools

- **NuSMV**
- Updated version of SMV by Cimatti and Roveri (IRST Trento)
- Input language: extended SMV
- Supports temporal logics CTL and LTL.

Model Checking Tools

- **VIS (Verification Interacting with Synthesis)**
  - A system for formal verification, synthesis, and simulation of finite state systems.
  - Developed jointly at the University of California at Berkeley and the University of Colorado at Boulder.
  - **F**eatures:
    - Fast simulation of logic circuits
    - Formal “implementation” verification (equivalence checking) of combinational and sequential circuits
    - Formal “design” verification using fair CTL model checking and language emptiness
  - • For more information: http://www-cad.eecs.Berkeley.edu/Respep/Research/vis/