Property Specification Formalisms*

Moshe Y. Vardi

1 Finite state machines

- Turing Machines = universal model of computability
  $\neq$ universal model of computers
  Finite State Machines = universal model of computers

- A Finite State Machine (FSM) M is a tuple $\langle \Sigma, \Delta, S, s_0, f, g \rangle$, where
  - Σ is a set of inputs
  - Δ is a set of outputs
  - S is a set of states,
  - $s_0 \in S$ is the initial state,
  - $f : S \times \Sigma \rightarrow S$ is the transition function,
  - $g : S \rightarrow \Delta$ is the output function,

- Languages to describe FSMs:
  - Statecharts
  - Esterel
  - HDL

- Computation: $(d_0, s_0, c_0), (d_1, s_1, c_1), \ldots$ such that
  - $d_i = g(s_i)$
  - $s_{i+1} = f(s_i, c_i)$

  (Trace equations)

- Traces:
  - Full traces: elements of $(\Sigma \times \Delta \times S)^\omega$
    (White Box View)
  - Visible traces: elements of $(\Sigma \times \Delta)^\omega$
    (Black Box View / Behavioral view)

*Notes of Prof. Vardi's lecture held in Bertinoro on May 23, 2006 (by Stefano Tonetta)

- The language $L(M)$ of M is defined as the set of all visible traces of M.
- Note: we see traces as lines (linear view); the trace set can alternatively bee seen as a tree (branching view).
- Note that our model is deadlock-free (*receptiveness*); a state cannot refuse an input.
- Sequential circuits can be modeled with FSMs: $C = \langle I, O, R, s_0, f, g \rangle$, where I and O are signal set, and R is a register set; then
  - $\Sigma = 2^I$
  - $\Delta = 2^O$
  - $S = 2^R$
  - $s_0 \in 2^R$
  - $f : 2^I \times 2^R \to w^R$
  - $g : 2^R \to 2^O$

2 Specification

- behavioral specification $\sigma : (\Sigma \times \Delta)^\omega \to \{0, 1\}$
  i.e., $L(\sigma) \subseteq (\Sigma \times \Delta)^\omega$
- **Correctness:**
  The FSM M implements the specification σ iff $L(M) \subseteq L(\sigma)$
- **Property specification languages:** How do we specify traces?
- Example: lecture specification:
  1. only one person talks at a time – mutual exclusion (*safety*)
  2. requests should be granted (*liveness*)
- (2) is not well defined: there are three possible interpretations:
  a Every momentary request has to be granted
  b Every continuous request has to be granted
  c Every repeated request has to be granted

  $\Rightarrow$ Need of precise languages
- In the following let us consider $PROP$ as a set of propositions (for example, in the case of a circuit $PROP = I \cup O$), $\Sigma = 2^{PROP}$, and traces as elements of Σ^ω (i.e. $\tau \in (2^{PROP})^\omega$).
- Suppose $A = \langle N, +, *, <, exp \rangle$

– Example (Fermat's last theorem):
$\neg(\exists x)(\exists y)(\exists z)(\exists n)(exp(x,n) + exp(y,n) = exp(z,n) \wedge n > 2)$

- Let's consider a proposition as a function $p : N \rightarrow \{0,1\}$ or equivalently $p \subseteq N$.
- We can define specifications with the First-Order Logics (FOL) over traces $T = \langle N, +1, <, p_1, ..., p_n \rangle$, where each p_i is a unary relation.
- Example
  1. $(\forall x)(\neg(p(x) \wedge q(x)))$ – mutual exclusion
  2. a $(\forall x)(r(x) \rightarrow (\exists y)(y \geq x \wedge g(x)))$ – every response is eventually granted
     b $(\forall x)(r(x) \rightarrow (\exists y)(y \geq x \wedge g(x) \wedge (\forall z)((z < y \wedge z \geq x) \rightarrow r(z))))$ – every continuous response is eventually granted – difficult!

2.1 Linear Temporal Logics

- LTL - Pnueli (1977)
- **Formulas**:
  1. Every prop is a formula.
  2. Formulas are closed under Boolean connectives.
  3. Formulas are closed under temporal connectives.
- **Temporal connectives**:
  – Unary: next (X), always (G), eventually (F).
  – Binary: until (U), release (R).
- Definition of $\tau \models \phi$:
  1. $\tau, i \models p$ iff $\tau(i)(p) = 1$
  2. $\tau, i \models \neg\phi$ iff $\tau, i \not\models \phi$
     $\tau, i \models \phi \wedge \psi$ iff $\tau, i \models \phi$ and $\tau, i \models \psi$
  3. $\tau, i \models X\phi$ iff $\tau, i+1 \models \phi$
     $\tau, i \models F\phi$ iff for some $j \geq i\ : \tau, j \models \phi$
     $\tau, i \models G\phi$ iff for all $j \geq i\ : \tau, j \models \phi$
     $\tau, i \models \phi U \psi$ iff for some $j \geq i\ : \tau, j \models \psi$ *and* for all $i \leq k < j\ : \tau, k \models \phi$
     $\tau, i \models \phi R \psi$ iff for all $j \geq 0$, either $: \tau, j \models \psi$ *or* for some $i \leq k < j\ : \tau, k \models \phi$
- $\tau \models \phi$ if $\tau, 0 \models \phi$

- $L(\phi) = \{\tau : \tau \models \phi\}$
- Example:
  1. $G\neg(p \wedge q)$ – mutual exclusion
  2. a. $G(r \rightarrow Fg)$ – every request is eventually granted

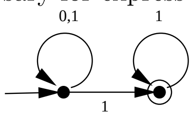
     b. $G(r \rightarrow rUg)$ – every continuous request is eventually granted

     c. $G((GFr) \rightarrow Fg)$ – every repeated requests is eventually granted (this is unintuitive, but think about the negation $F(GFr \wedge G\neg g)$)
- What is the relation between LTL and FOL?
- **Kamp's theorem:** FOL over traces and LTL have the same expressive power.
  - LTL→FOL: easy ($O(n)$) directly from LTL semantics
  - FOL→LTL: non-elementary ($O(2^{2^{2^{\cdots n}}})$)

2.2 Omega-regular languages

- FOL is not enough!
- We want to check if $M \models \phi$ (iff $L(M) \subseteq L(\phi)$).
- M is often a concurrent system ($M = M_1||M_2||...||M_{100}$).
- Instead of $M_1||M_2||...||M_{100}$, we focus on $M_{100} \models \phi$ and we abstract $M_1||M_2||...||M_{99}$.
- We represent $M_1||M_2||...||M_{99}$ with ψ and we check if $M_{100} \models \psi \rightarrow \phi$ (in *assume-guarantee* reasoning, ψ is called the *assumption*, while ϕ is called the *assertion*).
- We abstract the environment with an acceptor.
- A Büchi automaton (BA) is a tuple $A = \langle 2^{PROP}, Q, q_0, \delta, F\rangle$, where
  - Q is a set of states,
  - $q_0 \in Q$ is the initial state,
  - $\delta : Q \times 2^{PROP} \rightarrow 2^Q$ is the transition function,
  - $F \subseteq Q$ is the fairness condition.
- A run of a BA A over the infinite word $\tau = a_0, a_1, a_2, ... \in \Sigma^\omega$ is an infinite sequence $q_0, q_1, q_2, \ldots$ such that, for $i \geq 1$, $q_{i+1} \in \delta(q_i, a_i)$. A run is fair iff it visits F infinitely often. The word τ belongs to the language $L(A)$ of A iff there exists a fair run over τ.

- Note that non-determinism is necessary for abstraction; fairness is necessary for express many interesting properties:



Eventually 1 forever (or $FG1$).

- Büchi automata define the class of ω-*regular languages*.

- They are closed under Boolean operations.

- Fact: LTL$\equiv$FOL$\subset$ ω-regular languages.

  – Example:
  "p holds at every even moment" cannot be expressed in LTL.

- There are extensions of LTL that have ω-regular expressiveness:
  ETL $\equiv$ linear μ-calculus $\equiv$ QPTL $\equiv$ ω-regular languages. These languages are difficult for practitioners to use.

- **Regular expressions:**

  – ϵ is an RE denoting the empty word,

  – $a \in \Sigma$ is an RE denoting the unary a,

  – if e_1 and e_2 are RE, then e_1, e_2 is an RE
  ($L(e_1, e_2) = \{w_1, w_2 \mid w_1 \in L(e_1) \text{ and } w_2 \in L(e_2)\}$)

  – if e_1 and e_2 are RE, then $e_1|e_2$ is an RE
  ($L(e_1|e_2) = \{w \mid w \in L(e_1) \text{ or } w \in L(e_2)\}$)

  – if e is an RE, then e^* is an RE
  ($L(e^*) = \{w \mid |w| = 0$ or $w = w_1, w_2, ..., w_n$ with $w_i \in L(e)$ for $1 \leq i \leq n\}$

- RE are easy to understand and well known.

- $\tau, i, j \mathrel{|\!\!\equiv} e$ iff $\tau(i), ..., \tau(j) \in L(e)$.

- We obtain RELTL by adding two new operators to the syntax of LTL (we will use the syntax of ForSpec):

  – if r is a RE and ϕ is a formula, then r *trig* ϕ is an RELTL formula;

  – if r is a RE and ϕ is a formula, then r *seq* ϕ is an RELTL formula.

- Semantics:

  – $\tau, i \models r$ *trig* ϕ iff, for some $j \geq i$, $\tau, i, j \mathrel{|\!\!\equiv} r$ and $\tau, j \models \phi$;

  – $\tau, i \models r$ *seq* ϕ iff, for all $j \geq i$, if $\tau, i, j \mathrel{|\!\!\equiv} r$, then $\tau, j \models \phi$.

- Example:
  $G(((r \wedge \neg g), \neg g^*, (r \wedge \neg g), \neg g^*, (r \wedge \neg g))\ trig\ Xg)$.
- RELTL$\equiv$ ω-regular languages.

  $\Rightarrow$ we can translate an RELTL formula into a Büchi automaton and vice-versa (exponential)
- Different languages to describe RELTL properties:
  - ForSpec (Intel)
  - PSL (Accelera standard)
  - SVA (System Verilog Assertions)

3 Model Checking

- $L(M) \subseteq L(\psi)$

$\iff L(M) \cap L(\neg\psi) = \emptyset$

$\iff L(M) \cap L(A_{\neg\psi}) = \emptyset$

$\iff L(M \times A_{\neg\psi}) = \emptyset$ (can be solved using graph-search methods)