Verification = Logic + Algorithmics*

Moshe Y. Vardi

Rice University

*See `http://www.cs.rice.edu/~vardi` for related papers.

Traditional Verification

Deductive Approach:

- Express the correctness of the protocol as a formula in some logic.
- Prove validity of the formula.

Disadvantage: *difficult*!

- Only a small numbers of teams have been able to apply this approach to industrial-scale protocols.

Modern Verification

Model Checking:

- Represent a *finite-state* protocol as a *finite* structure.
- Check that the structure is a model of the specification.

Advantage: feasible

- [Apt + Kozen, 1985]: "one of the most exciting developments in the theory of program correctness"
- Impressive record of applicability over the last decade

Linear Temporal Logic

Temporal logic: logic of temporal sequences

Main feature: time is implicit

- next φ: φ holds in the next state.
- eventually φ: φ holds eventually
- always φ: φ holds from now on
- φ until ψ: φ holds until ψ holds.

Examples

- always $\neg(CS_1 \wedge CS_2)$: mutual exclusion (safety)
- always (**Request** $\rightarrow$ eventually **Grant**): liveness
- always (**Request** $\rightarrow$ **Request** until **Grant**): liveness
- always eventually **Request** $\rightarrow$ eventually **Grant**: liveness
- always $(CS_1 \rightarrow (CS_1$ until $\neg CS_1 \wedge (\neg CS_1$ until $CS_2)))$: precedence

LTL over Finite Computations

- $\sigma, i \models \text{next } \varphi$ if $i < |\sigma|$ and $\sigma, i+1 \models \varphi$

Compilation Theorem I: Given an LTL formula φ, we can construct a finite automaton A_φ such that a computation σ satisfies φ iff A_φ accepts σ. Also, $|A_\varphi| = 4^{|\varphi|}$.

Applications:

- **Satisfiability**: φ is satisfiable iff $L(A_\varphi) \neq \emptyset$
- **Validity**: φ is valid iff $L(A_{\neg\varphi}) = \emptyset$

Compilation

Closure: $cl(\varphi)$: subformulas of φ and their negations

- $cl($always eventually $p)$ = $\{$always eventually p, $\neg$always eventually p, eventually p, $\neg$eventually p, $p, \neg p\}$

$A_\varphi = (\Sigma, S, S_0, \rho, F)$

- *Alphabet*: $\Sigma = 2^{Prop}$
- *States*: $S \subseteq 2^{cl(\varphi)}$
  - $\psi \in t$ iff $\neg\psi \notin t$
  - $\psi_1 \wedge \psi_2 \in t$ iff $\psi_1 \in t$ and $\psi_2 \in t$
- *Initial states*: $S_0 = \{s \in S : \varphi \in s\}$
- *Accepting states*: $F = \{\emptyset\}$

Transition Relation

Case I: $t' \neq \emptyset$

$(t, a, t') \in \rho$:

- $a \subseteq t$
- next $\psi \in t$ iff $\psi \in t'$
- ψ_1 until $\psi_2 \in t$ iff $\psi_2 \in t$ or $(\psi_1 \in t$ and ψ_1 until $\psi_2 \in t')$

Case II: $t' = \emptyset$

$(t, a, t') \in \rho$:

- $a \subseteq t$
- next $\psi \notin t$
- ψ_1 until $\psi_2 \in t$ iff $\psi_2 \in t$

Algorithmics

Proposition: $L(A) \neq \emptyset$ iff there is a path in A from an initial state to an accepting state

Complexity:

- *Linear time*: breadth-first search
- *Nondeterministic logarithmic space*: nondeterministic walk

Algorithmics: graph reachability

Complexity of LTL satisfiability and validity:

- Exponential time
- Polynomial space

Model Checking

The following are equivalent:

- P satisfies φ
- $L(P) \subseteq L(A_\varphi)$
- $L(P) \cap \overline{L(A_\varphi)} = \emptyset$
- $L(P) \cap L(A_{\neg\varphi}) = \emptyset$
- $L(P \times A_{\neg\varphi}) = \emptyset$

Complexity:

- *Time*:
  - linear in $|P|$
  - exponential in $|\varphi|$
- *Space*:
  - polylogarithmic in $|P|$
  - polynomial in $|\varphi|$

SPIN: time and space optimizations

Infinite Computations

The problem: fulfillment of eventualities

always eventually P:

- *Finite computations*: P holds in last state
- *Infinite computations*: P holds i.o.

always eventually P and *always eventually* Q:

- *Finite computations*: $P \wedge Q$ holds in last state
- *Infinite computations*: P holds i.o. and Q holds i.o.

Automata on Infinite Words

$A = (\Sigma, S, S_0, \rho, F)$

- *Alphabet*: Σ
- *States*: S
- *Initial states*: $S_0 \subseteq S$
- *Transition relation*: $\rho \subseteq S \times \Sigma \times S$
- *Acceptance condition*: F

Input word: $a_0, a_1, \ldots$

Run: $s_0, s_1, \ldots$

- $s_0 \in S_0$
- $(s_i, a_i, s_{i+1}) \in \rho$ for $i \geq 0$

Limit: $inf(r)$ – states visited i.o.

Goal: Compilation theorem for LTL on infinite computations

Streett Automata

Streett acceptance condition: $(L_1, U_1), \ldots, (L_k, U_k)$

- $L_i \subseteq S$, $U_i \subseteq S$
- *acceptance*: for each I, infinitely many visits in L_i $\Rightarrow$ infinitely many visits in U_i

Compilation Theorem II: [Grumberg + Long, 1991] For LTL use Streett acceptance condition:

- $L_i = \{t : \psi_1 \text{ until } \psi_2 \in t\}$
- $U_i = \{t : \psi_2 \in t\}$

Algorithmics: *Streett reachability* – is there an infinite path, from an initial state, that satisfies Streett condition

Cost: emptiness of Streett automata

- no linear-time algorithm known
- no space-efficient algorithm possible

Generalized Büchi Automata

Generalized Büchi acceptance condition: $F_1, \ldots, F_k$

- $F_i \subseteq S$
- *acceptance*: for each i, infinitely many visits in F_i

Compilation Theorem II: [Gerth, Peled, Vardi + Wolper, 1995] For LTL use generalized Büchi acceptance condition:

- $F_i = \{t : \psi_1$ until $\psi_2 \notin t$ or $\psi_2 \in t\}$

Algorithmics: *generalized Büchi reachability*

Advantage: easy emptiness test

- linear-time algorithm
- space-efficient algorithm

Disadvantage: tailored to "vanilla" LTL, fails for extensions (ETL, μTL)

- ψ_1 until ψ_2 is fulfilled immediately or persists

Büchi Automata

Büchi acceptance condition: F

- $F \subseteq S$
- *acceptance*: infinitely many visits in F

Compilation Theorem III: [Vardi + Wolper, 1994]
$A_\varphi =$ local automaton $\times$ *eventuality automaton*

- *local automaton*: checks local conditions
- *eventuality automaton*: ensures fulfullment of eventualities
- $F = S \times \{\emptyset\}$

Advantages:

- easy emptiness test
- handles extensions of LTL (i.e., ETL, μTL)

Disadvantage: complicated proof

Büchi Reachability

Proposition: $L(A) \neq \emptyset$ iff there is a path in A from an initial state s_0 to an accepting state t and from t to itself.

Algorithmics: Büchi reachability

- *Linear-time algorithm*: depth-first search
- *Nondeterministic logspace algorithm*: nondeterministic walk
- *Space-efficient algorithm*: hash functions

Logic vs. Algorithmics

It is in the eye of the beholder:

- *So far*: "algorithmics" means graph reachability
- *Proposal*: declare propositional logic as "algorithmics"

Alternating Reachability: reachability in and/or graphs

- $\vee$-node is reachable if *some* successor is reachable
- $\wedge$-node is reachable if *all* successors are reachable

Complexity:

- PTIME-complete
- linear-time algorithm
- essentially: HORNSAT

Alternating Automata

Nondeterminism$=\exists$ choice

- $\exists$ choice: *some* run accepts
- $\forall$ choice: *all* runs accept

Alternation: $\exists + \forall$ choice

2 Formalisms:

- $\exists$-states and $\forall$-states [Chandra et al.]
- Boolean transitions [Brzozowski + Leiss]

Nondeterminism vs. Alternation

Nondeterminism:

- $(s, a) \mapsto \{s_1, s_2, s_3\}$
- $\rho(s, a) = s_1 \vee s_2 \vee s_3$

Alternation: $\rho(s, a) = (s_1 \wedge s_2) \vee (s_1 \wedge s_3)$

Positive Boolean Formulas:
$B^+(X)$ – positive Boolean formulas over X

- *Example*: $x_1 \vee (x_2 \wedge x_3)$

Alternating Büchi Automata

- $A = (\Sigma, S, S_0, \rho, F)$
- $\rho : S \times \Sigma \rightarrow B^+(S)$

Example: $\rho(s, a) = (s_1 \wedge s_2) \vee (s_1 \wedge s_3)$

Run: An S-labeled tree that satisfies the transition functions

Acceptance: F is visited i.o. along *every* branch of the run

LTL vs. Alternating Automata

$\varphi \mapsto A_\varphi = (\Sigma, S, S_0, \rho, F)$

- $\Sigma = 2^{Prop}$
- $S = cl(\varphi)$
- $S_0 = \{\varphi\}$
- $F = \{\neg(\psi_1 \text{ until } \psi_2) \in cl(\varphi)\}$

Dualizing Transitions:

- $\overline{\psi} = \neg\psi$
- $\overline{true} = false$
- $\overline{false} = true$
- $\overline{\alpha \wedge \beta} = \overline{\alpha} \vee \overline{\beta}$
- $\overline{\alpha \vee \beta} = \overline{\alpha} \wedge \overline{\beta}$

Example: $\overline{p \vee (\neg q \wedge \text{ next } p \text{ until } q)}$ is
$\neg p \wedge (q \vee \neg \text{ next } p \text{ until } q)$

Transition Function

- $\rho(p, a) = true$ if $p \in a$
- $\rho(p, a) = false$ if $p \notin a$
- $\rho(\psi_1 \wedge \psi_2) = \rho(\psi_1, a) \wedge \rho(\psi_2, a)$
- $\rho(\neg\psi, a) = \overline{\rho(\psi, a)}$
- $\rho($ next $\psi, a) = \psi$
- $\rho(\psi_1$ until $\psi_2, a) = \rho(\psi_2, a) \vee (\rho(\psi_1), a) \wedge (\psi_1$ until $\psi_2))$

Advantages:

- easy translation
- syntax directed
- extensible

Alternating Automata vs. Games

$A = (\Sigma, S, S_0, \rho, F)$, $w = a_0, a_1, \ldots$

- *Players*: automaton vs. spoiler
- *Configuration*: (s, i), where $s \in S$ and $i \geq 0$
- *Initial configuration*: $(s_0, 0)$, where $s_0 \in S_0$
- *Round*: from $\rho(s, a_i)$
  - automaton chooses disjunctions
  - spoiler chooses conjunctions

  until a state $s' \in S$ is reached; then $(s', i+1)$ is next configuration
- *Win*: Automaton wins by reaching $true$ or visiting F i.o.

Claim: $w \in L(A)$ iff automaton has a winning strategy

Alternation vs. Nondeterminism

Theorem: [Miyano + Hayashi, 1984] Automaton has a winning strategy iff automaton has a memoryless winning strategy

Corollary: Let A be an alternating Büchi automaton. Then A is equivalent to a nondeterministoc Büchi automaton A' such that $|A'| = 3^{|A|}$.

Compilation:

- LTL $\mapsto$ alternating automata
- alternating automata $\mapsto$ nondeterministic automata

Algorithmics: Nontrivial, but only once.

Computation Tree Logic

CTL: logic of computation trees

Main feature: quantification over computations

- A next φ: φ holds in all successor states
- A eventually φ: φ holds eventually inevitably
- E always φ: φ holds from now on some computation
- E φ until ψ: φ holds until ψ holds on some computation

Example: A always (**Request** $\rightarrow$ A eventually **Grant**)

Complexity

- **Satisfiability**: EXPTIME-complete [Emerson + Halpern]
- **Model Checking**: PTIME-complete [Clarke et al.]

Automata-theoretic approach to satisfiability:

- **Compilation** [VW]: Given an CTL formula φ, we can construct a Büchi tree automaton A_φ such that a tree τ satisfies φ iff A_φ accepts τ. Also, $|A_\varphi| = 2^{O(|\varphi|)}$.
- **Emptiness Test for Büchi Tree Automata** [VW]: quadratic time

Difficulty: automata too large to be used for model checking

Alternation and CTL

Compilation: **Compilation** [Müller + Schupp]: Given an CTL formula φ, we can construct a alternating tree automaton A_φ such that a tree τ satisfies φ iff A_φ accepts τ. Also, $|A_\varphi| = O(|\varphi|)$.

Emptiness Test:

- *2-letter alphabet*: EXPTIME-complete [MS]
- *1-letter alphabet*: PTIME-complete [BVW]

Alternation and Model Checking

- $\varphi \mapsto A_\varphi$ (linear)
- $P \times A_\varphi \mapsto A_{P,\varphi}$ (linear)
- $L(A_{P,\varphi}) \neq \emptyset$ (linear)

Crux: $A_{P,\varphi}$ is a 1-letter alphabet

In Summary

- Logic is *high-level*, automata are low-level
- Specify with *logic*, but verify with automata
- *Alternation* bridges the gap
  - Unifying framework
  - Uniform algorithms

Caveat: further optimization required!