Temporal Verification of Reactive Systems

Temporal Verification of Reactive Systems

Z. Manna and A. Pnuelli

Springer-Verlag, 1995

Content

- Preliminary Concepts
- Invariance
  - Proof Methods
  - Applications
- Precedence
- General Safety

Preliminary Concepts

Fair Transition System

- Variables describe states and properties of program and they're typed.
- Variables are from universal set - vocabulary V
  for each $x \in V$ its primed $x' \in V$
- Expressions: funcs applied to vars and consts
- Atomic Formulas: propositions (boolean vars) and formulas (expressions with e.g. > predicate)
- Assertions (State Formulas): applied boolean connectives and quantification to atomic formulas; assertion is first order formula

Preliminary Concepts

Fair Transition System

- FTS, <V, θ, T, J, C>, represents reactive program.
- V - finite set of system vars - data & control ones
- θ - The initial condition. All states programs can start with. State is initial if it satisfies θ.
- T - Finite set of transitions; t $\subseteq$T - 1st order formula
- J $\subseteq$T - Just transitions; t that's continually enabled, but not taken beyond a certain point is disabled.
- C $\subseteq$T - Compassionate transitions; t is disabled if infinitely enabled but taken finitely many times only.

Preliminary Concepts

Fair Transition System

- t is enabled on the state s if t (s) = $\varnothing$
- s′ is a t-successor of state s if formula $\rho_t(V, V')$ evaluates to true
- One of T must be idling transition (V=V′).
- Interpretation of V-V is not restricted.
- Let $U \subseteq V$ be a set of variables and s^ to be U-variant of state s if s^ & s agree on the interpretation of V - U.

Preliminary Concepts

Temporal Logic

- Language for specifying properties of reactive systems
- Extended ordinary predicate logic with set of special temporal operators
- Underlying assertion language over symbols
- Formula in assertion language is state formula - assertion
- Temporal formula is constructed out of state formulas, boolean connectives ($\vee$, $\neg$), quantification ($\forall$, $\exists$) and temporal operators

Preliminary Concepts

Temporal Logic - Future Temporal Operators

- $\Box p$ Henceforth p
- $\Diamond p$ Eventually p
- $p \, U \, q$ p Until q
- $p \, W \, q$ p Waiting-for (Unless) q
- $\bigcirc p$ Next p

Preliminary Concepts

Temporal Logic - Past Temporal Operators

- $\boxminus p$ — So-far p
- $\diamondminus p$ — Once p
- $p \, S \, q$ — p Since q
- $p \, B \, q$ — p Back-to q
- $\ominus p$ — Previously p
- $\sim p$ — Before p

Preliminary Concepts

Temporal Logic

- Temporal formulas are interpreted over model, which is an infinite sequence of states σ
- Variables of V are partitioned into:
  - flexible - may assume different values in different states
  - rigid - must assume the same value in all states
- Previous value (x^-) $x = x^- + 1$
- Next value (x^+) $x^+ = x + 1$
- Extending language with location of control within program means specification of properties

Invariance
Proof Methods

- In *predictive* sequence of states each s_j interprets primed x' of sys var $x \in V$ in the same way s_{j+1} interprets x
- Verification condition - special formula expressing effect of a single transition t connecting two consecutive states
- Main role of verification condition is to establish that every t-successor of φ state in computation of program P is a ψ state.
  $\{\varphi\}\ t\ \{\psi\}$

Invariance

Proof Methods - Invariance Rule

- The Basic Rule

  For an assertion φ and premises B_1, B_2:

  $$\frac{B_1 \quad \theta \rightarrow \varphi \qquad B_2 \ \{\varphi\}\ T\ \{\varphi\}}{\Box\varphi}$$

  (basic invariance)

- Monotonocity and Conjunctiveness

  For assertions p and q:

  $$\frac{\Box p,\ p \rightarrow q}{\Box q}$$

  (monotonicity of invariances)

Invariance

Proof Methods - Invariance Rule

- Problem: p is an invariant of program P (i.e. $\Box p$ is P-valid), but the basic rule is not directly applicable
- Strategy #1: Stronger Assertion
  - φ is inductive if B_1 and B_2 are state valid (i.e. P-valid)
  - 1+3+5+...+(2k+1) is the square of an integer
    => $1+3+5...+(2k+1)=(k+1)^2$
- Strategy #2: Incremental Proofs
  - In some cases prove simpler invariant with basic rule and use it to establish a more complicated invariant
  - Derive P-state validity $p \rightarrow q$ from weaker $X \wedge p \rightarrow q$ plus P-invariant X

Invariance

Proof Methods - Finding Inductive Assertions

- The Bottom-Up Approach
  - algorithmic
  - solely based on the program's text (all what's given)
  - guaranteed to produce inductive assertions
- The Top-Down Approach
  - heuristic
  - based on the goal assertion (assume it to be given too), whose invariance is wished to be established,
  - based on the program's text as well
  - not guaranteed to produce an inductive assertion

Invariance

Proof Methods - Refining Invariants

- Coarse granularity => less complexity of verification
- Finer granularity => better efficiency of program
- Compromise:
  - first perform preliminary analysis of coarser version of program and verify its correctness
  - then derive finer version of program by refining large grouped statements into smaller statements
  - verify derived finer version

Invariance
Applications

- Many distributed applications have as their body identical processes
- Example: 3 processes access shared resource => How to prove that mutual exclusion is maintained?
- In some cases it is possible to treat family of programs P^1, P^2, P^3, … in uniform way, providing single proof of correctness of P^n for $n \geq 1$.
- Parameterization of 3 areas is the key:
  - programs
  - programs specifications
  - corresponding verification process

Invariance

Applications - Single-Resource Allocation

- Semaphores
  - Invariant means that at all times there can be at most one process in critical section.
- Synchronous Message Passing
  - M of C(ustomer) processes and arbiter process A
  - Communication channels for each C[i] to A; messages can be sent only with cooperation of allocator A
  - Resource is allocated via sending 0 and 1 from C[i]
  - Resource can be granted twice in succession to same C[i], but for long run another C[i] can not be ignored forever - invariant

Invariance

Applications - Single-Resource Allocation

- Bounded Overtaking
  - C[i]s and A as in Synch. Msg. Passing
  - Explicit round-robin order for scanning for requests
  - φ: $\neg$(at_c[k] $\wedge$ at_c[n]) for every k, n $\in$ [1..M] where k$\neq$n.
- Shared Variables
  - M customers(C[i] and arbiter (A), requests and grants r, g:array [1..M] of boolean
  - r[i]:=true // request by i
  - if g[i] = true // granted to i
  - φ: $\neg$(at_c[k] $\wedge$ at_c[n]) for every k, n $\in$ [1..M] where k$\neq$n.

Invariance

Applications - Multiple-Resource Allocation

- m resources shared among n processes
- Order of requesting resources - deadlock avoidance
- Symmetric solution uses identical components
- Dining philosophers
  - M philosophers and M chopsticks
  - request left chopstick and then right one - symmetric
  - φ: $\neg(at\_c(j] \wedge at\_c[j \oplus_M 1])$ for every $j \in [1..M]$
  - left chopstick first is symmetric, but no order of requesting
    => possible deadlock
    => break the symmetry

Invariance

Applications - Multiple-Resource Allocation

- Dining Philosophers - A Contrary Philosopher
  - Let M^{th} philosopher be the contrary one (right chopstick first)
  - no deadlock, but no symmetry too => 2 components
- Dining Philosophers - Exclusion
  - Let the entrance to dining room be guarded by semaphore initialized to value from <1, M-1>
  - Symmetry is broken dynamically => only one component

Invariance

Applications - Constructing Linear Invariants

- A restricted set of invariants can be constructed algorithmically using bottom-up approach since they're independent of any goal assertions.
- Variable y is called linear if effect of any transition can be expressed as y'=y+C, where C is some constant; C may vary from transition to transition.
- For example semaphore is linear variable.
- $\sum_i a_i \cdot y_i + \sum_l b_l \cdot at\_l = K$

Invariance

Applications - Completeness

- For every assertion p such as $\Box p$ is P-valid, there exists an assertion φ such that the premises of general rule of invariance are provable from state validities.

Invariance

Applications - Finite-State Algorithmic Verification

- A program P is called *finite-state* program if each system variable $x \in V$ assumes only finitely many values in all computations of P.
- For the class of finite-state programs, the question of whether any given assertion p is invariant over P can be determined algorithmically.
- For finite-state program, the state transition graph G_P corresponding to P can be constructed.

Invariance

Applications - Finite-State Algorithmic Verification

- Constructing G_P
  - Initially, place as nodes all initial states - i.e. states satisfying θ.
  - Repeat following step, until no new nodes or edges can be added.

    For some $s \in G_P$, let s_1, .., s_k be the successor of s. Add all nodes among $\{s_1, .., s_k\}$ that are not already there and draw a directed edge connecting s to s_i for each i=1…k.
- Checking P-Invariances
  - Construct G_P and this way obtain all accessible states.
  - Check whether each accessible state satisfies p.

Precedence
Waiting-for Rule

- Basic Rule
  - $p \Rightarrow q \; W \; r$ //p, q, r are formulas
  - p-state is followed by q-interval that may be terminated only by an occurrence of r
  - q-interval may be empty, if the p-state also satisfies r
  - q-interval may also extend to infinity and no r-state is needed
  - For assertions φ, ψ

$$\frac{\{\varphi\} \; T \; \{\varphi \vee \psi\}}{\varphi \Rightarrow \varphi \; W \; \psi}$$

Precedence

Waiting-for Rule

- General Rule
  - Basic rule can only prove conclusions for which p = q
  - => strengthen q
  - For assertions p, q, r, φ and premises W1, W2, W3

    W1. $p \rightarrow \varphi \vee r$

    W2. $\varphi \rightarrow q$

    W3. $\{\varphi\}\ T\ \{\varphi \vee r\}$

    $p \Rightarrow q\ W\ r$

Precedence

Nested Waiting-for Rule

- $p \Rightarrow q_n \; W \; q_{n-1} \ldots q_1 \; W \; q_0$
- Basic Rule

$$\frac{\{\varphi_i\} \; T \; \{\vee_{j<i} \varphi_j\}}{(\vee_j \varphi_j) \Rightarrow \varphi_m \; W \; \varphi_{m-1} \; W \; \ldots \; \varphi_1 \; W \; \varphi_0}$$

- General Rule
  - Basic rule is not always applicable => strengthening
  - N1. $p \rightarrow \vee_j \varphi_j$
    N2. $\varphi_i \rightarrow q_i$
    N3. $\{\varphi_i\} \; T \; \{\vee_j \varphi_j\}$

$$\frac{}{p \Rightarrow q_m \; W \; q_{m-1} \; W \; \ldots \; q_1 \; W \; q_0}$$

Precedence
Verification Diagrams

- Used to visually summarize assertions under consideration and possible transitions among them
- Verification diagram is directed graph:
  - Assertions label nodes.
  - Edges represent transitions.
  - One of the nodes may be designated as a terminal (goal) node. There are no edges leading from such node.

Precedence

Completeness

- If the formula

  $$p \Rightarrow q_m \, W \, q_{m-1} \, W \ldots q_1 \, W \, q_0$$

  is P-valid, then there exist assertions $\varphi_0, \varphi_1 \ldots \varphi_m$ such that the premises of general waiting rule are provable from state validities.

General Safety

Invariance Rule for Past Formulas

- Safety property is any property that can be specified by a formula of the form $\Box p$ for some past formula p.
- Rules for proving invariance of past formulas are essentially the same rules as for state formulas
- Past general invariance
  - For past formulas φ, p

    P1. $\varphi \Rightarrow p$ $\qquad\qquad$ $//\Box(\varphi \rightarrow p)$

    P2. $\theta \rightarrow (\varphi)_0$

    P3. $\{\varphi\}\ T\ \{\varphi\}$

    $\Box p$

General Safety

Causality Rule

- Causality properties: properties expressible by formulas of the form $p \Rightarrow \Diamond\!\!\!\!-\; r$ - p-state must be preceded by an r-state
- p and r may be quire remote in time => Local case of causality: immediate predecessor of any p-state is an r-state =>
- Inverse verification condition
  $\{\varphi\}\ t^{-1}\ \{\psi\}$
  - If the next state achieved by performing t satisfies φ, then the current state must satisfy ψ

General Safety

Causality Rule

- For past formulas p, r, φ

  C1. $p \Rightarrow \varphi \vee r$

  C2. $\theta \rightarrow \neg (\varphi)_0 \vee (r)_0$

  C3. $\{\varphi\}\ T^{-1}\ \{\varphi \vee r\}$

  $p \Rightarrow \diamondsuit\!\!\!-\ r$

- C1: any p-position must be r-position or satisfy φ
- C2: any position satisfying θ can not satisfy $(\varphi)_0$
- C3: immediate predecessor of φ must satisfy either φ or r

General Safety

Backward Analysis

- Useful for proving causality and safety properties
- Most striking example is proof by contradiction of simple invariances
- For example let ψ describe that more than 1process is in critical section, then we need to prove $\Box\neg\psi$
  - Let's have a chain of assertions $\psi = \psi_1, \psi_2, \ldots \psi_k$, where ψ_1 precedes ψ_2 and so on
  - Contradiction is obtained by reaching such ψ_k which can not be satisfied by any P-accessible state showing that ψ never arises and therefore $\neg\psi$ is an invariant of program

General Safety

Backward Analysis - Inference Rules

- Transitivity of causality

$$\frac{p \Rightarrow \ominus\, q \qquad q \Rightarrow \ominus\, r}{p \Rightarrow \ominus\, r}$$

- Case splitting for causality

$$\frac{p \Rightarrow \ominus\, q \qquad q \Rightarrow \ominus\, r}{p \vee q \Rightarrow \ominus\, r}$$

- Monotonocity of causality

$$\frac{p \Rightarrow q \qquad q \Rightarrow \ominus\, r \qquad r \Rightarrow \quad u}{p \Rightarrow \ominus\, u}$$

- p, q and r stand for past formulas

General Safety

Back-to Rule

- Extension of Wait-for formula to handle cases where p, q_0, …, q_m are past formulas
  $p \Rightarrow q_m \; B \; q_{m-1} \ldots q_1 \; B \; q_0$
- Basic symmetry
  $p \Rightarrow q_m \; W \; q_{m-1} \ldots q_1 \; W \; r \quad \sim$
  $(\neg q_1) \Rightarrow q_2 \; B \; q_3 \ldots q_m \; (\neg p) \; B \; r$
  - Every occurrence of $\neg q_1$ is preceded by a succession of q_2-interval, preceded by a q_3-interval and so on, until a $\neg p$ interval.
  - The sequence may be interrupted earlier or terminated by an occurrence of r.

General Safety

Back-to Formulas

- Back-to rule

  For past formulas p, q ,r φ

  B1. $p \Rightarrow \varphi \vee r$

  B2. $\varphi \Rightarrow q$

  B3. $\{\varphi \vee r\}\ T^{-1}\ \{\varphi\}$

  $p \Rightarrow q\ B\ r$

General Safety

Back-to Formulas

- Past Waiting-For

  For past formulas u, v, w, X

  W1. $u \Rightarrow X \vee w$

  W2. $X \Rightarrow v$

  W3. $\{X\}\ T\ \{X \vee w\}$

  $u \Rightarrow v\ W\ w$

General Safety

Nested Back-to Formulas

- For past formulas p, q_0, q_1, ...q_m and φ_0, φ_1, ..., φ_m

  N1. $p \Rightarrow \vee_j \varphi_j$

  N2. $\varphi_i \Rightarrow q_i$ for i=0, 1, ..., m

  N3. $\{\varphi_i\}\ T^{-1}\ \{\vee_i \varphi_j\}$ for i=0, 1, ..., m

  ---

  $p \Rightarrow q_m\ B\ q_{m-1} \ldots q_1\ B\ q_0$

General Safety
Completeness

- For every program P and past formula ψ such that $\Box\psi$ is P-valid, there exists a past formula φ such that the premises of past general invariance are provable from from state (first-order) validities.