Topics in program analysis

Arnab

June 19, 2024

Chapter 1

Properties and problems in program analysis

1.1 Analysis of general programs

1.1.1 Preliminaries

Deterministic Boolean program : A deterministic Boolean program is a sequential set of operations acting on Boolean-valued program variables V. A program state $\sigma \in \{0,1\}^{|V|}$ is an assignment of Boolean values to program variables V. The program is basically a mapping $\psi : \{0,1\}^{|V|} \to \{0,1\}^{|V|}$, i.e., it takes an input program state $\sigma_i \in \{0,1\}^{|V|}$ and outputs another program state $\sigma_o = \psi(\sigma_i)$. The projection $\psi_b : \{0,1\}^{|V|} \to \{0,1\}$ of ψ on any particular program variable $b \in V$ is a Boolean function.

A loop in a program consists of a Boolean predicate known as the **loop guard** which controls the execution of a sequential set of statements called the **loop body**. The loop body is executed iteratively until the loop guard is falsified. Given below is an example of a deterministic Boolean program P with a loop, defined over the program variables $V = \{n, flip\}$. In a general program, it is possible to have loops that do not terminate. As a result we additionally specify the maximum number of loop iterations we allow while considering the mapping ψ_P. Let us denote the mapping by ψ_P^k if we are only considering k loop iterations. For example, in the program given below, $\psi_P^1(n, \text{flip}) = (\neg n, 1)$ if flip $= 0$ and $\psi_P(n, \text{flip}) = (n, \text{flip})$ if flip $= 1$.

```
bool n,flip;
while (flip==0) {
    flip := 1 ;
    n:= !n;
    }
```

Without loss of generality, henceforth we refer to deterministic Boolean programs as deterministic programs.

In general, loops in programs pose a challenge as they are harder to analyse than other operations. One key tool for analysing a loop is the loop invariant. The loop invariant can be described as a property which if satisfied at the beginning of a loop iteration, also holds after the execution of the loop body. However, computing the loop invariant is challenging as it involves a fixed point computation. We define these notions more formally as follows:

Property : A property is a set of program states i.e. $T \subseteq \{0,1\}^{|V|}$. Alternatively, it can also be viewed as a Boolean valued function defined over $\{0,1\}^{|V|}$.

It is desirable for the property to have a "nice" structure for easier analysis. We describe two such structural features a property T should ideally have as follows:

- **Membership check for the property T should be doable efficiently**. This implies that, given a program state $e \in \{0,1\}^{|V|}$, testing whether e satisfies T, i.e., whether $e \in T$ should be achievable in **polynomial time**. In other words, membership in T should be efficiently decidable or the Boolean function representing T should be efficiently computable.
- **Succinct description of the property T**. This implies that the property $T \subseteq \{0,1\}^{|V|}$ can be expressed in some **compact form**, for instance, it should be expressible as a Boolean CNF formula or a low-degree polynomial over V or as a d-dimensional polytope where $d \leq |V|$.

One important thing to note is that a property can be defined irrespective of the characteristic of the program, that is, whether it is deterministic or probabilistic in nature.

In this context, we'll mostly deal with a particular property of a given program, known as the *invariant property*.

Definition 1.1.1. Exact invariant for a deterministic program : Given a deterministic program P and a set S of initial states, an exact invariant property is a set T of states such that starting from some state $x \in T$, if the program P is executed, it terminates in an output state y such that $y \in T$.

Problem 1.1.1. Given a deterministic program P, a set of initial states S and a formula T' for a candidate, verify whether T' is an exact invariant of the program P for the given set S.

Verifying whether a candidate T' is actually an exact invariant of the program P for a given set S of initial states, we need to check the following two conditions :

- $S \subseteq T'$.
- T' is closed with respect to the program P, i.e., starting from some state $x \in T'$, if the program P is executed, it terminates in an output state y such that $y \in T'$.

Algorithm 1 Invariant verifier for deterministic program

Require: Input (maybe with loops) deterministic program P, set S of initial states, candidate invariant T' expressed in CNF.

1: Obtain the formula $Y = F(X)$ for the input-output mapping for one execution/iteration of the program P, where X is the initial state and Y is the final/output state.
2: Call a SAT solver to check the satisfiability of the formula $S \wedge \neg T'$. If it's unsatisfiable, proceed to the next step. Else, REJECT.
3: Call a SAT solver to check the satisfiability of the formula $T'(X) \wedge F \wedge \neg T'(Y)$. If it's unsatisfiable, ACCEPT. Else, REJECT.

Correctness analysis of Algorithm 1

For a given deterministic program P, a set S of initial states and a candidate invariant T' (both expressible as CNF formulas), the task to verify whether T' is an exact invariant for the set S comprise checking the following two conditions :

1. S is contained in the set T', i.e., any satisfying assignment of the CNF formula S also satisfies T'.

   Lemma 1. The formula $S \wedge \neg T'$ is unsatisfiable if and only if $S \subseteq T'$.

Proof. $S \wedge \neg T'$ **is unsatisfiable implies** $S \subseteq T'$: Suppose $S \nsubseteq T'$. Thus, there exists a satisfying assignment σ of S which doesn't satisfy T', i.e., $\sigma \models (S \wedge \neg T')$, a contradiction.
$S \subseteq T'$ **implies** $S \wedge \neg T'$ **is unsatisfiable** : Suppose $S \wedge \neg T'$ is satisfiable and that $\sigma \models (S \wedge \neg T')$. This implies that $\sigma \models S$ and $\sigma \not\models T'$, i.e., $S \nsubseteq T'$, a contradiction. □

2. T' is closed with respect to the program P, i.e., starting from some state $x \in T'$, if the program P is executed, it terminates in an output state y such that $y \in T'$.

   Lemma 2. The formula $T'(X) \wedge F \wedge \neg T'(Y)$ is unsatisfiable if and only if T' is closed with respect to the program P.

   Proof. $T'(X) \wedge F \wedge \neg T'(Y)$ **is unsatisfiable implies** T' **is closed with respect to the program** P**.** : Suppose T' is not closed with respect to P, that is, there exists a valid transition (x, y) of the program (where $x, y \in \{0,1\}^{|V|}$) such that $x \models T'(X)$ and $y \not\models T'(Y)$. Thus, the formula $T'(X) \wedge F \wedge \neg T'(Y)$ has a satisfying assignment (x, y), a contradiction.
   T' **is closed with respect to the program** P **implies** $T'(X) \wedge F \wedge \neg T'(Y)$ **is unsatisfiable.** : Suppose there exists a satisfying assignment (x, y) for the formula $T'(X) \wedge F \wedge \neg T'(Y)$. □

Definition 1.1.2. Distance between sets of states for deterministic program: We can define two types of distances for quantifying the notion of "closeness/farness" of an invariant set T from a given candidate set T' in the following ways:

1. Given a candidate invariant T' and an exact invariant T, the *multiplicative distance* ***metric*** between T and T' can be defined as $d_m(T, T') = \frac{|T \triangle T'|}{|T \vee T'|}$.

2. Given a candidate invariant T' and an exact invariant T, the ***asymmetric*** *multiplicative distance* between T and T' can be defined as $d_m(T, T') = \frac{|T \triangle T'|}{|T'|}$.

3. Given a candidate invariant T' and an exact invariant T, the *symmetric difference* between T and T' can be defined as $d_m(T, T') = |T \triangle T'|$.

4. Given a candidate invariant T' and an exact invariant T for a deterministic program defined over variables V, the (additive) distance between T and T' can be defined as $d_a(T, T') = \Pr_{s \sim_U \{0,1\}^{|V|}}[T(s) \neq T'(s)] = \frac{|T \triangle T'|}{|\Omega|}$, where Ω is the universal set of program states.

Definition 1.1.3. k**-iterative closure of a set** : Given any set T' of program states for a deterministic program P, the k-iterative closure $\hat{T}'_k$ of T' is defined as follows :

$$\hat{T}'_k = \{\sigma_o \in \Omega \mid \sigma_o \text{ is reachable from } \sigma_i \text{ in at most } k \text{ iterations of } P, \text{ where } \sigma_i \in T'\}$$

Problem 1.1.2. Given a deterministic program P, a set S of initial states, a candidate T', an exact invariant T and parameters $\epsilon_1, \epsilon_2 \in [0, 1], k \in \mathbb{N}$, decide whether :

- k-iterative closure $\hat{T}'$ of T' is ϵ_1-close to T in the multiplicative sense OR
- k-iterative closure $\hat{T}'$ of T' is ϵ_2-far from T in the multiplicative sense.

Algorithm 2 Testing closeness/farness of the closure of a candidate set from a given invariant for deterministic program

Require: Input (maybe with loops) deterministic program P, set S of initial states, candidate invariant T' expressed in CNF, exact invariant T expressed in CNF, parameters $\epsilon_1 < \epsilon_2 \in (0,1), \delta \in (0,1)$ and $k \in \mathbb{N}$.

1: Set another parameter $\eta \in (0, ((\frac{\epsilon_2}{\epsilon_1})^{\frac{1}{4}} - 1))$.

2: Obtain the formula $Y = F(X)$ for the input-output mapping for one execution/iteration of the program P, where X is the initial state and Y is the final/output state.

3: Compute the k-iterative closure $\hat{T}'_k$ of the set T' using the k-fold input-output mapping $Y = F^k(X)$ for the program P. (The set $\hat{T}'_k$ should correspond to the set of witnesses of the formula $(T'(X) \bigvee_{i=1}^{k} (T'(X) \wedge F^i(X)))$.

4: Construct the two formulas $F'_1 = T \oplus \hat{T}'_k$ and $F'_2 = T \vee \hat{T}'_k$.

5: Call an approximate SAT counter twice and obtain the following values :

- $\alpha = \text{ApproxMC}(F'_1, \eta, \delta)$.
- $\beta = \text{ApproxMC}(F'_2, \eta, \delta)$.

6: If $\frac{\alpha}{\beta} < \epsilon_1(1+\eta)^2$, return ACCEPT.

7: If $\frac{\alpha}{\beta} > \frac{\epsilon_2}{(1+\eta)^2}$, return REJECT.

Correctness analysis of Algorithm 2

Lemma 3. Given the candidate invariant T' and the single iteration input-output mapping $Y = F(X)$ for the deterministic program P, both expressed as CNF, the witnesses of the formula $(T'(X) \bigvee_{i=1}^{k} (T'(X) \wedge F^i(X))$ correspond to the k-iterative closure of T'.

The witnesses of the formula $F'_1 = T \oplus \hat{T}'_k$ correspond to those states σ such that either - 1) $\sigma \in T$ and $\sigma \notin \hat{T}'_k$ or 2) $\sigma \notin T$ and $\sigma \in \hat{T}'_k$. Thus, the exact count of the number of witnesses of F'_1 (given by $|F'_1|$) would give us the quantity $|T \triangle \hat{T}'_k|$. Similarly, the exact count of the number of witnesses of F'_2 (given by $|F'_2|$) would give us the quantity $|T \vee \hat{T}'_k|$. The multiplicative distance $d_m(T, \hat{T}'_k) = \frac{|T \triangle \hat{T}'_k|}{|T \vee \hat{T}'_k|}$ between the given exact invariant T and the k-iterative closure $\hat{T}'_k$ of the candidate T' is given by the quantity $\frac{|F'_1|}{|F'_2|}$.

Lemma 4. Given parameters $\epsilon_1 < \epsilon_2 \in [0,1]$ and $\eta \in (0, ((\frac{\epsilon_2}{\epsilon_1})^{\frac{1}{4}} - 1))$, if $d_m(T, \hat{T}'_k) \leq \epsilon_1$, the algorithm ACCEPTS with probability atleast $1 - \delta$. Similarly, if $d_m(T, \hat{T}'_k) \geq \epsilon_2$, the algorithm REJECTS with probability $1 - \delta$.

Proof. In Line 5, the algorithm calls the approximate SAT counter ApproxMC twice, in order to obtain approximate values of $|F'_1|$ and $|F'_2|$ respectively. The two calls $\text{ApproxMC}(F'_1, \eta, \delta)$ and $\text{ApproxMC}(F'_2, \eta, \delta)$ returns two values α and β such that the following holds with probability atleast $1 - \delta$:

$$\alpha \in [\tfrac{|F'_1|}{1+\eta}, |F'_1|(1+\eta)]$$
$$\beta \in [\tfrac{|F'_2|}{1+\eta}, |F'_2|(1+\eta)]$$

Thus, the statistic $\frac{\alpha}{\beta} \in [\frac{|F'_1|/|F'_2|}{(1+\eta)^2}, (|F'_1|/|F'_2|)(1+\eta)^2]$ with probability atleast $1 - \delta$.

Case 1. $d_m(T, \hat{T}'_k) \leq \epsilon_1$: This implies that $|F'_1|/|F'_2| \leq \epsilon_1$. Thus, with probability atleast $1 - \delta$, the statistic $\frac{\alpha}{\beta} \leq (|F'_1|/|F'_2|)(1+\eta)^2 < \epsilon_1(1+\eta)^2$ and the algorithm ACCEPTS.

Case 2. $d_m(T, \hat{T}'_k) \geq \epsilon_2$: This implies that $|F'_1|/|F'_2| \geq \epsilon_2$. Thus, with probability atleast $1-\delta$, the statistic $\frac{\alpha}{\beta} \geq \frac{(|F'_1|/|F'_2|)}{(1+\eta)^2} > \frac{\epsilon_2}{(1+\eta)^2}$ and the algorithm REJECTS.

We would ideally like the two intervals $[\frac{\epsilon_1}{(1+\eta)^2}, \epsilon_1(1+\eta)^2]$ and $[\frac{\epsilon_2}{(1+\eta)^2}, \epsilon_2(1+\eta)^2]$ to be non-overlapping. In order to ensure this, we basically need the following inequality to be satisfied : $\epsilon_1(1+\eta)^2 < \frac{\epsilon_2}{(1+\eta)^2}$, from which we get the choice of η as $(0, ((\frac{\epsilon_2}{\epsilon_1})^{\frac{1}{4}} - 1))$. □

Problem 1.1.3. Given a deterministic program P, a set S of initial states, a candidate T', an exact invariant T and parameters $\epsilon_1, \epsilon_2 \in [0,1], k \in \mathbb{N}$, decide whether :

- T' is ϵ_1-close to T in the multiplicative sense OR
- T' is ϵ_2-far from T in the multiplicative sense.

REMARK : This problem can be approached by first sampling states from $T' \vee T$ to obtain a set S. Then, sample from $T' \triangle T$ conditioned on S.

Algorithm 3 Testing closeness/farness of a candidate set from a given invariant for deterministic program

---

Require: Input (maybe with loops) deterministic program P, set S of initial states, candidate invariant T' expressed in CNF, exact invariant T expressed in CNF, parameters $\epsilon_1 < \epsilon_2 \in (0,1), \delta \in (0,1)$.

1: Set another parameter $\eta \in (0, ((\frac{\epsilon_2}{\epsilon_1})^{\frac{1}{4}} - 1))$.
2: Construct the two formulas $F'_1 = T \oplus T'$ and $F'_2 = T \vee T'$.
3: Call an approximate SAT counter twice and obtain the following values :
   - $\alpha = \text{ApproxMC}(F'_1, \eta, \delta)$.
   - $\beta = \text{ApproxMC}(F'_2, \eta, \delta)$.
4: If $\frac{\alpha}{\beta} < \epsilon_1(1+\eta)^2$, return ACCEPT.
5: If $\frac{\alpha}{\beta} > \frac{\epsilon_2}{(1+\eta)^2}$, return REJECT.

Problem 1.1.4. Given a deterministic program P, a set S of initial states, a candidate T', an exact invariant T and parameters $\epsilon_1, \epsilon_2 \in [0,1], k \in \mathbb{N}$, decide whether :

- T' is ϵ_1-close to T in the additive distance OR
- T' is ϵ_2-far from T in the additive distance.

REMARK : Sampling states can be the way to go forward. For a state s sampled uniformly at random from Ω, $\Pr_{s \sim_U \Omega}[T'(s) \neq T(s)] = \frac{|T \triangle T'|}{|\Omega|}$.

Let $X = \sum_{s=1}^{m} I_s$, where I_s is the indicator RV such that $I_s = 1$ if $s \models T \oplus T'$. $\Pr[|X - m\frac{|T \triangle T'|}{|\Omega|}| \geq t] \leq \exp(-2t^2/m)$.

For approximation with probability atleast $1-\delta$ to within t-additive error, we need $m \geq \frac{2+t}{t^2}\log(\frac{2}{\delta})$ samples.

1.2 Analysis of probabilistic programs

Definition 1.2.1. Boolean probabilistic program : A Boolean probabilistic program is a Boolean program (possibly with loops) over a set of program variables V, wherein the sequential operations are performed based on some independent and fair internal coin flips.

In addition to the set of program variables V and a Boolean workspace, the probabilistic program now has access to random bits, represented by R. A biased coin toss with probability p can be simulated using $O(\log(1/p))$ fair coins (with the associated probability being $1/2$). Using q fair random coins, we can simulate any event with probability $p = \frac{x}{2^q}$, $x \in [2^q]$. If the Boolean program uses m bits for the workspace and q random bits each sampled with probability $1/2$, we can view the Boolean probabilistic program as mapping $\psi : \{0,1\}^{|V|} \times \{0,1\}^m \times \{0,1\}^q \to \{0,1\}^{|V|} \times \{0,1\}^m$.

In other words, a Boolean probabilistic program is a randomized procedure which maps the input states $\sigma \in \{0,1\}^{|V|}$ to a distribution over $\{0,1\}^{|V|+|R|}$ where the internal coin flips are represented by a collection of random bits $R = (r_1, r_2, ..., r_q) \in \{0,1\}^q$ for q coin flips.

```
bool n,
flip;
while (flip==0) {
{
(flip := 1 [p] n:= !n)}
}
(n)
```

Definition 1.2.2. Exact invariant for a probabilistic program (*probability* 1-*invariant*) : Given a probabilistic program P (equipped with internal random variables R) and a set S of initial states, an exact invariant is a set T of states such that starting from some state $x \in T$, if the program P is executed, it terminates in an output state $y(x, R)$ such that $y \in T$ with probability 1, i.e., for all choices of the internal random bits, the output state satisfies the invariant T.

$$\Pr_R[y(x, R) \in T \mid x \in T] = 1$$

Problem 1.2.1. Given a probabilistic program P, a set of initial states S and a formula T' for a candidate, verify whether T' is an exact invariant of the program P for the given set S.

Verifying whether a candidate T' is actually an exact invariant of the program P (equipped with internal random variables R) for a given set S of initial states, we need to check the following two conditions :

- $S \subseteq T'$.
- T' is closed with respect to the program P, i.e., starting from some state $x \in T'$, if the program P is executed, it terminates in some output state $y(x, R)$ such that $y \in T'$ with probability 1.

Algorithm 4 Exact invariant verifier for probabilistic program

Require: Input (maybe with loops) probabilistic program P, set S of initial states, candidate invariant T' expressed in CNF.

1: Obtain the formula $Y = F(X, R)$ for the input-output mapping for one execution/iteration of the program P with random variables R, where X is the initial state and Y is the final/output state.
2: Call a SAT solver to check the satisfiability of the formula $S \wedge \neg T'$. If it's unsatisfiable, proceed to the next step. Else, REJECT.
3: Call a SAT solver to check the satisfiability of the formula $T'(X) \wedge F \wedge \neg T'(Y)$. If it's unsatisfiable, ACCEPT. Else, REJECT.

In this context, for a probabilistic program P, we can define a few variants of *approximate* invariant for a given set S of initial states.

Definition 1.2.3. Approximate invariant for a probabilistic program : Given a probabilistic program P equipped with internal random variables R, a set S of initial states, parameters $\epsilon, \delta \in (0,1)$, an (ϵ, δ)-approximate invariant is a set T of states such that the following two-fold property holds for T -

- Let's define the concept of a *good state.* $x \in T$ is a good state if $\Pr_R[y(x,R) \in T \mid x \in T] \geq (1-\delta)$.
- T is an (ϵ, δ)-approximate invariant if $\Pr_{x \sim_U T}[x \text{ is a good state}] \geq (1-\epsilon)$.

In general, atleast $(1-\epsilon)$ fraction of the states in T should be "δ-close" from being closed with respect to the program P.

Definition 1.2.4. Distance between sets of states for probabilistic program: Given a probabilistic program P and for a given set S of initial states, we can define two types of distances for quantifying the notion of "closeness/farness" of an invariant set T from a given candidate set T' in the following ways :

1. Given a candidate invariant T' and an exact invariant T, the *multiplicative distance* ***metric*** between T and T' can be defined as $d_{m,mdm}(T,T') = \frac{|T \triangle T'|}{|T \vee T'|}$.
2. Given a candidate invariant T' and an exact invariant T, the ***asymmetric*** *multiplicative distance* between T and T' can be defined as $d_{m,amd}(T,T') = \frac{|T \triangle T'|}{|T'|}$.
3. Given a candidate invariant T' and an exact invariant T, the *symmetric difference* between T and T' can be defined as $d_{m,symd}(T,T') = |T \triangle T'|$.
4. We can also define another notion of distance between T and T' for a given probabilistic program P in terms of the total variation (TV) distance between the induced distributions $\mathcal{D}_x$ and $\mathcal{D}_y$ over all possible output states $\sigma_o \in \{0,1\}^{|V|}$ for any $x \in T, y \in T'$. Formally, we can define the distance as follows :

$$d_{TV}(T,T') = \min_{x \in T, y \in T'} d_{TV}(\mathcal{D}_x, \mathcal{D}_y)$$

5. Given a candidate invariant T', the probabilistic distance between T' and the class $\mathcal{I}$ of exact invariants can be given by $d_{m,prob}(T', \mathcal{I}) = \Pr_{x \sim_U T', R}[y(x,R) \notin T' \mid x \in T']$. This measure basically estimates the probability that a randomly selected transition from a state chosen uniformly at random from T' is a violating one.
6. Given a candidate invariant T' and an exact invariant T for a probabilistic program defined over variables V, the (additive) distance between T and T' can be defined as $d_a(T,T') = \Pr_{s \sim_U \{0,1\}^{|V|}}[T(s) \neq T'(s)] = \frac{|T \triangle T'|}{|\Omega|}$, where Ω is the universal set of program states.

Problem 1.2.2. Given a probabilistic program P, a set S of initial states, a candidate T' and a parameter $\epsilon \in [0,1]$, decide whether :

- T' is a *probability-1* invariant OR
- T' is ϵ-far from being a *probability-1* invariant in terms of the distance measure $d_{m,prob}(T', \mathcal{I}) = \Pr_{x \sim_U T', R}[y(x,R) \notin T' \mid x \in T']$, where $\mathcal{I}$ is the class of *probability-1* invariants.

Algorithm 5 One-sided tester of invariants for probabilistic program

Require: Input (maybe with loops) probabilistic program P, set S of initial states, candidate invariant T' expressed in CNF, parameter $\epsilon \in [0,1]$.

1: Set another parameter $\eta \in [0,1]$.

2: Obtain the formula $Y = F(X,R)$ for the input-output mapping for one execution/iteration of the program P with random variables R, where X is the initial state and Y is the final/output state.

3: Call a SAT solver to check the satisfiability of the formula $S \wedge \neg T'$. If it's unsatisfiable, proceed to the next step. Else, REJECT.

4: Call a SAT solver to check the satisfiability of the formula $F' = T'(X) \wedge F \wedge \neg T'(Y)$. If it's unsatisfiable, ACCEPT. Else, proceed.

5: Estimate the number of violating transitions and the size of the set corresponding to T' by calling an approximate SAT counter twice:

- $\alpha = \text{ApproxMC}(F', \eta, \delta)$
- $\beta = \text{ApproxMC}(T', \eta, \delta)$

6: If $\frac{\alpha}{\beta} > \frac{\epsilon 2^{|R|}}{(1+\eta)^2}$, REJECT.

Problem 1.2.3. Given a probabilistic program P, a set S of initial states, a candidate T', an exact invariant T and parameters $\epsilon_1, \epsilon_2 \in [0,1]$, decide whether :

- T' is ϵ_1-close to T in the multiplicative sense OR
- T' is ϵ_2-far from T in the multiplicative sense.

The distance notion to be used in the case of probabilistic programs needs to be decided very carefully.

Chapter 2

Implementation details

2.1 Problems on implementation

Problem 2.1.1. Given a deterministic program P, a set of initial states S and a formula T' for a candidate, verify whether T' is an exact invariant of the program P for the given set S.

2.1.1 Tools used in the problem

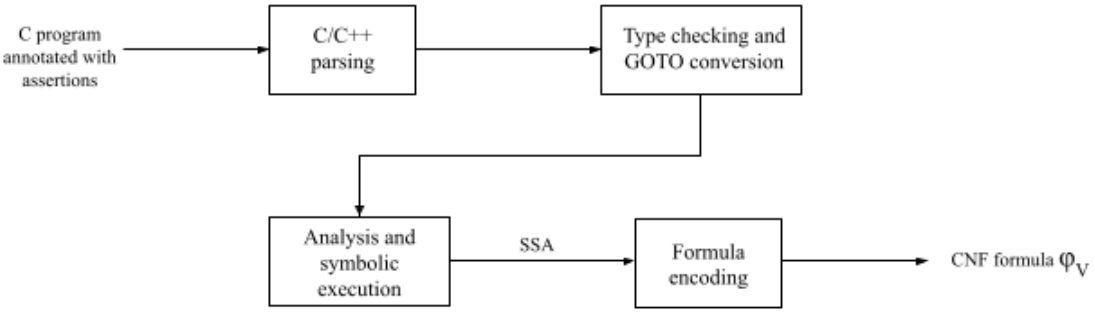
- The C Bounded Model Checker (CBMC) : This particular tool demonstrates the violation of assertions in C programs. CBMC translates an input C program (over program variables V) annotated with assertions and with loops unrolled upto a specified depth, in a *bit-precise* manner, into a CNF formula φ_V. CBMC generates the formula in such a way that the set of satisfying assignments $S = \{\sigma \mid \varphi_V(\sigma) = 1\}$ to φ_V correspond to the set of program paths leading to violation of atleast one non-trivial assertion. CBMC won't generate any formula if the program is either not annotated or all the assertions in the program are trivial, that is, true or false. A model of the formula φ_V thus corresponds to a counterexample. In order to do so, CBMC performs the following steps (which are relevant to our task) outlined in Figure 1 and described in Section 2.1.1.
- CryptoMiniSat SAT solver.

Workflow of C Bounded Model Checker (CBMC)

The sequence of steps followed by CBMC in order to convert an input Boolean C program annotated by assertions into an equivalent propositional formula have been detailed as follows :

- The parsing and type checking module parses the input C program and transforms it into a GOTO program via replacing the `break` and `continue` statements by equivalent `goto` statements and `for` and `do-while` loops by equivalent `while` loops. The loop constructs are unwound a given number of times as supplied by the user followed by an extra assertion, known as the **unwinding assertion** which ensures that the loop terminates after executing the required number of times. Recursive function calls and backward `goto` statements are similarly unwound upto a certain depth governed by an unwinding assertion.
- The resultant GOTO program from the preceding step only consists of (possibly nested) `if` instructions, assignments, assertions, labels and `goto` instructions with forward branching. This program is then modified, using a set A of auxiliary variables, into a static single assignment (SSA) form so that every new definition of a variable results in a new instance. The variable instances are numbered so that each use of a variable may be easily linked back to a single definition point. Here is a simple example of such a transformation :

Figure 2.1: CBMC overview



Input Boolean program P:

```
x = x || y;
y = x && y;
assert(!y);
```

SSA form of the program P:

```
x2 = x1 || y1;
y2 = x2 && y1;
assert(!y2);
```

- The annotated program P in SSA form is then converted into a CNF formula in a pretty much straightforward manner. For example, the CNF formula $F' = F \wedge \neg Z$ is generated for the above program. The respective formulas F and Z are given by

$$\begin{aligned} F &:= (x_2 = x_1 \vee y_1) \wedge \\ &\quad (y_2 = x_2 \wedge y_1) \\ Z &:= \neg y_2 \end{aligned}$$

where $x_2, y_2 \in A$ correspond to the intermediate/auxiliary variables in the SSA form. Since the main objective of CBMC is to output a trace of execution of the program where atleast one of the assertions fail, it simply includes the negated assertions in the formula, which in this case is $\neg Z := y_2$. An important aspect to note about CBMC is that it will only generate a CNF formula iff atleast one of the assertions is non-trivial.

Algorithm 6 Invariant verifier for deterministic program

Require: Input (maybe with loops) deterministic C program P, set of program variables V, candidate invariant T expressed in CNF, a dummy assertion Z expressed in CNF.

1: Obtain the CNF formula $F' = F \wedge \neg Z$ and the auxiliary variable mapping $A = f(V)$ via the function call $CBMC(L, V, P)$ to C Bounded Model Checker.
2: Trim the formula $F' = F \wedge \neg Z$ to obtain the CNF formula F.
3: Obtain the CNF formulas $T(V)$ and $T(A)$ corresponding to the candidate invariant formulas corresponding to the initial and the final states respectively.
4: Pass the CNF formula $F'' = T(V) \wedge F \wedge \neg T(A)$ to the SAT solver CryptoMiniSat. If it outputs *unsatisfiable*, ACCEPT. Else, REJECT.

2.2 Our work

2.2.1 Verifying a candidate invariant for deterministic Boolean programs

Analysis of the algorithm

Analysing the formula corresponding to the valid runs of the program

One interesting thing to notice is that the CNF formula $F' = F \wedge \neg Z$ that is generated by CBMC for an input C program P (defined over program variables V) annotated by the dummy assertion Z, comprises two sub-formulas F and $\neg Z$. Let A be the set of auxiliary variables used to rename the program variables in the SSA form of P. The sub-formula $F(v, a) : \{0,1\}^{|V|+|A|} \rightarrow \{0,1\}$ corresponding to the SSA form of the C program evaluates to true iff $(v, a) \in (V \times A)$ is a valid single iteration trace of the program.

Thus, the formula $F'(v, a) = F(v, a) \wedge \neg Z(a)$ evaluates to true iff (v, a) is a valid trace of a single iteration of the program P, not satisfying the property Z. We need this dummy property Z since otherwise CBMC won't be generating a formula F at all for the valid runs of the program P.

For the example Boolean program P in SSA form given Section 2.1.1, $V = \{x_1, y_1\}$, $A = \{x_2, y_2\}$ and $Z = \neg y_2$. The formula $F(x_1, y_1, x_2, y_2) = (x_2 = x_1 \vee y_1) \wedge (y_2 = x_2 \wedge y_1)$ evaluates to 1 iff $(x_1, y_1, x_2, y_2) \in (V \times A)$ correspond to a valid trace of the program and to 0 otherwise. If the input program is deterministic, there exists exactly one valid run of the program. For instance $F(x_1 = 0, y_1 = 0, x_2 = 0, y_2 = 0) = 0$ since y_2 can never be set to 1 in the program if $x_2 = y_1 = 0$. Now, $F' = F \wedge \neg Z = F \wedge \neg y_2$. Consequently, the set $\sigma_{F'}$ of *satisfying assignments* to F' correspond to all the valid runs of the program which ends up with the variable y being true in the original program. Thus F' corresponds to the Boolean function $\psi_{P,y}$ for the variable y.

Which formula corresponds to the invariant property?

For an input (maybe with loops) Boolean C program P over the set V of program variables, generating the CNF formula F corresponding to the valid runs of the program is vital in order to reason about the behaviour of the program. Specifically, given an initial state $s_i \in \{0,1\}^{|V|}$ of P and a candidate invariant property in the form of a CNF formula I, the task of verifying whether T indeed satisfies the invariant condition for P boils down to checking whether the following formula F'' is unsatisfiable :

$$F'' = T(s_i) \wedge \neg I(\psi_P(s_i))$$
$$\text{or equivalently } F'' = T(s_i) \wedge F(s_i, A(s_i)) \wedge \neg T(F(s_i, A(s_i)))$$

where $F(s_i, A(s_i))$ is the formula for which, given an initial state $s_i \in \{0,1\}^{|V|}$, there is only one satisfying assignment $\sigma_F \in \{0,1\}^{|V|+|A|}$ of F. This is because P is a deterministic program and hence, for a given starting state, it admits only one valid run, which is essentially captured by σ_F. Therefore, the satisfying assignments of $F'' = T(V) \wedge F \wedge \neg T(A)$ would capture all those valid runs of the program P which would start from some state satisfying the property T end up in a state not satisfying T.

Thus, passing the formula F'' to the CryptoMiniSat SAT solver would output *unsatisfiable* iff from any initial state $\sigma_i \in (\{0,1\}^{|V|} \cap T)$,that is, from a state σ_i satisfying the property T, the valid run starting from σ_i ends up in a final state $\sigma_o \in \{0,1\}^{|V|+|A|}$, which satisfies the given candidate invariant property T, i.e.,

the formula T indeed corresponds to an invariant class of states under P. Otherwise, it outputs a satisfying assignment $(\sigma_i, \sigma_o) = \sigma_{cex} \in \{0,1\}^{|V|+|A|}$ for the formula F'', which in effect acts as a counterexample for the given candidate T.

2.2.2 Verifying a candidate invariant for a probabilistic Boolean program

Problem 2.2.1. Given a probabilistic program P, a set of initial states S and a formula T' for a candidate, verify whether T' is an exact invariant of the program P for the given set S.

Algorithm 7 Invariant verifier for probabilistic program

Require: Input (maybe with loops) probabilistic C program P, set of program variables V and internal random coins R, candidate invariant T expressed in CNF, a dummy assertion Z expressed in CNF.
Obtain the CNF formula $F' = F \wedge \neg Z$ and the auxiliary variable mapping $A = f(V, R)$ via the function call $CBMC(L, V, P)$ to C Bounded Model Checker.
2: Trim the formula $F' = F \wedge \neg Z$ to obtain the CNF formula F.
Obtain the CNF formulas $T(V)$ and $T(A)$ corresponding to the candidate invariant formulas corresponding to the initial and the final states respectively.
4: Pass the CNF formula $F'' = T(V) \wedge F \wedge \neg T(A)$ to the SAT solver CryptoMiniSat. If it outputs *unsatisfiable*, ACCEPT. Else, REJECT.

Analysis of the algorithm

Analysing the formula corresponding to the valid runs of the program

Just as in the case of deterministic programs, the CNF formula $F' = F \wedge \neg Z$ (where Z is the dummy assertion) is generated by CBMC for an input C program P (defined over program variables V and internal random variables R). The sub-formula $F(V, R, A)$ corresponds to the valid traces of the program P similar to the one for deterministic programs.The CBMC tool also outputs an auxiliary variable mapping $A = f(V, R)$ for the auxiliary variables.

One thing to observe carefully in this context is that any probabilistic program P with program variables V and internal random variables R can be viewed as an equivalent deterministic program P_{det} equipped with program variables $V' = V \cup R$.

Claim : Probabilistic program $P(V, R)$ and the deterministic program $P_{det}(V')$ are equivalent.

For a fixed $v \in V$, if we fix a $r \in R$ beforehand, P behaves exactly in a deterministic manner, i.e., an initial state (v, r) gets mapped exactly to one output state $a \in A$. Now, from the perspective of P, the random variables R are internal to the program and each $r \in R$ behave like fair coins, that is, they assume 0 or 1 with equal probability. Thus for a fixed initial state $\sigma_i \in \{0,1\}^{|V|}$, the output state σ_o will depend upon the seeding of the random variables R. Thus, for a fixed initial state $\sigma_i \in \{0,1\}^{|V|}$, there can be atmost $2^{|R|}$ choices for the output state σ_o depending on the valuations of the random variables R.

Here is a simple example of a probabilistic program P and its corresponding SSA form :
Input Boolean program P: SSA form of the program P:

```
if (r1 || r2)                if (r11 || r21)
{                            {
    a = a || b;                  a2 = a1 || b1;
    b = a && b;                  b2 = a2 && b1;
}                            }
else                         else
{                            {
    a = a && b;                  a3 = a1 && b1;
    b = a || b;                  b3 = a3 || b1;
}                            }
assert(!a || !b);            assert(!a4 || !b4);
```

In this example, the notations are given by :

Program variables $V = \{a, b\}$
Internal random variables $R = \{rp, rq\}$

For the SSA form of P, the corresponding variables are given by :

$V = \{a_1, b_1\}, R = \{r_{11}, r_{21}\}, A = \{a_2, b_2, a_3, b_3, a_4, b_4\}$. The formula $F' = F \wedge \neg Z$ is generated for the above program. The respective formulas F and P are given by

$$\begin{aligned} F &:= ((r_{11} \vee r_{21}) \wedge (a_2 = a_1 \vee b_1) \wedge (b_2 = a_2 \wedge b_1) \wedge (a_4 = a_2 \wedge b_4 = b_2)) \wedge \\ &\quad ((\neg r_{11} \wedge \neg r_{21}) \wedge (a_3 = a_1 \wedge b_1) \wedge (b_3 = a_3 \vee b_1) \wedge (a_4 = a_3 \wedge b_4 = b_3)) \\ Z &:= \neg(\neg a_4 \vee \neg b_4) \end{aligned}$$

Which formula serves the purpose for verifying an invariant property?

Given a probabilistic C program P (define over the program variables V with a set R of internal random variables) annotated with a dummy assertion Z as input, the CBMC tool outputs a formula $F'(V, R, A) = F(V, R, A) \wedge \neg Z(A)$, where A is the set of auxiliary variables used to rename the program variables in the SSA form of P. Similar to the case of deterministic program, the sub-formula $F(V, R, A)$ in effect corresponds to the function $F(v, a, r) : \{0, 1\}^{|V|+|R|+|A|} \rightarrow \{0, 1\}$ which evaluates to 1 for a given input $(v, r, a) \in (V \times R \times A)$ iff (v, r, a) is a valid single iteration trace of the program P, i.e., for a given initial state $x \in \{0, 1\}^{|V|}$ and a fixed setting $r \in \{0, 1\}^{|R|}$ of the random variables R, the program behaves in a deterministic manner and ends up in a final state $\sigma_o \subseteq (a \in \{0, 1\}^{|A|})$.