

SAT Encoding of Fault-span Generation for Automated Synthesis problem



JINGSHU CHEN
CSE 814
INSTRUCTOR: DR. DILLON

Background



- Asserting correctness in a program is one of the most important aspect and application of formal methods
- Two approaches:
 - correct-by-verification, and
 - correct-by-construction.
 - e.g. automated verification vs. Automated synthesis

Background



- **Automated verification**
 - after-the-fact task,
 - **iterative** procedure
 - A cycle of design, verification and subsequently manual revision, if the verification step does not succeed.
- **Requirements of a program evolve during the program life cycle => **automated synthesis****
 - Incomplete specification
 - Change of environment
 - Impossible to anticipate all possible physical events at design time

Background



- **Automated synthesis**
 - Study the problem of revising an existing fault-intolerant program by solely adding fault-tolerance to the program
 - Preserves existing properties
 - Satisfies a set of new properties
 - Fault-span generation is one of the key problems
- **Fault-span**
 - a subset of state space
 - Key operation: reachability analysis/image computation

SAT-related



- Tremendous progress in SAT solver technology over the past decade
- SAT solver has been successfully used in model checking
- SAT solver:
 - Boolean formula in CNF representation
 - CNF: $(A \vee B) \wedge (\neg B \vee C \vee \neg D) \wedge (D \vee \neg E)$
 - Keys: non-chronological backtracking, efficient conflict driven learning of conflict clauses, and improved decision heuristics.
 - Main solvers: CHAFF, GRASP, MINISAT...

Roadmap



- **A plain SAT-encoding method**
- **Optimizations**
- **Conclusion and future work**

A plain SAT-encoding method(1)



- A program P is a tuple like this form: $P = \langle \psi_P, I_P \rangle$
 - \langle transitions, states set \rangle : a state is a combination of variables assignment
 - Transition: (guard)state \rightarrow (target)state
 - Fault is represented as transition
 - Fault-span generation:
 - ✦ Compute the reachable states, given a set of current states and a transition set

```
//recomputing the fault-span
REPEAT
  S2:= S1;
  S1:= FWReachStates(I1,  $\psi_1 \vee f$ );
  S1 := S1-fte;
  mt := mt  $\wedge$  S1;
   $\psi_1$  :=  $\psi_1$ - group( $\psi_1 \wedge mt$ );
UNTIL(S1=S2)
```

A plain SAT-encoding method(2)



- Note: it is customary to convert any DNF to CNF form by introducing intermediate variables.
- Encode the set of states as a set of DNF
 - $(a_1, b_2, c_0), (a_1, b_2, c_1)$ $(a_1 \wedge b_2 \wedge c_0) \vee (a_1 \wedge b_2 \wedge c_1)$
- Encode transitions as CNF
 - $T(s, i, s') : s \rightarrow s'$ (guard state \rightarrow target state) $S \wedge S'$
- Fault-span generation
 - Key step: $\bar{S}_{i-1}(x) \wedge T(x, i, x') \wedge \neg S_{reach}(x')$

Optimization- efficient state set representation



- **States may use the common variable**
 - initial state: (a_0, d_1, e_0)
 - Transition: $(a_0, d_1, e_0) \rightarrow (a_1, d_1, e_0)$
 $(a_1, d_1, e_0) \rightarrow (a_0, b_0, d_0)$
 - Reachable states:
 $(a_0, d_1, e_0), (a_1, d_1, e_0), (a_0, b_0, d_0)$
- **An efficient representation:**
 - hash table
 - Trie
 - Node: assume that the variables in the cubes are ordered

Optimization- efficient state set representation



- **Hash table**

- each entry of table stores a set of cubes that made up of the same variables.
- each entry is stored as a trie form.
- The hash value for a cube is a bit string.
 - ✦ Contains 1 for each variable present
 - ✦ Contains 0 for each variable not present in the cube
 - ✦ The trailing 0s are removed to get a short bit string
 - ✦ e.g. variables: a,b,c,d,e,f,... a cube $d (a \wedge \neg b \wedge \neg f)$ is 101001.

Optimization- efficient state set representation



- **Hash table**

- The cubes in the same table entry have the same length of bitstring
- Encode bit corresponding to a literal is 1 if it is positive, and 0 otherwise.
 - ✦ E.g. the cube $d(a \wedge \neg b \wedge \neg f)$ is stored as 100 .

- **Note: the property of Boolean function can be utilized to produce more compact representation of state.**

- E.g. 1 is an exponentially compact representation than DNF

$$(a \wedge b) \vee (a \wedge \neg b) \vee (\neg a \wedge b) \vee (\neg a \wedge \neg b)$$

Optimization- group transition



- Symmetric property of distributed processes
- A simple example:
 - $V_p = \{a, b\}$ and $R_p = \{a\}$: p is not allowed to read b .
 - As far as p is concerned, $\neg a \wedge \neg b \wedge a' \wedge \neg b'$ $\neg a \wedge b \wedge a' \wedge b'$
- each transition (s_0, s_0') in T_p is associated with

$$\begin{aligned} Group_p(s_0, s_0') &= \bigvee_{(s_1, s_1') \models T_p} \\ & \left(\bigwedge_{v \notin R_p} (v(s_0) = v(s_0') \wedge v(s_1) = v(s_1')) \right) \wedge \\ & \left(\bigwedge_{v \in R_p} (v(s_0) = v(s_1) \wedge v(s_0') = v(s_1')) \right) \wedge \end{aligned}$$

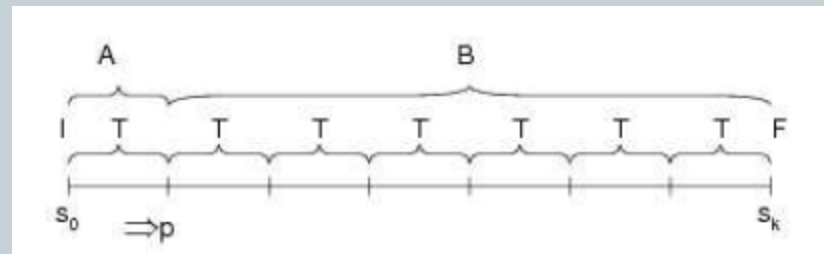
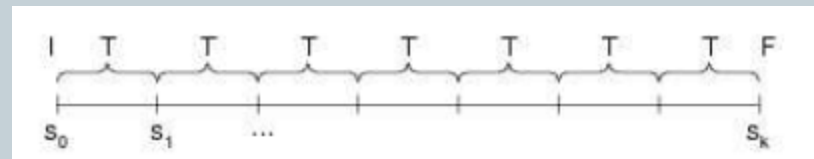
Optimization-Interpolant-based method



An interpolant [9] for a pair of clause sets (A, B) is a formula P with the following properties:

- A implies P ,
- $P \wedge B$ is unsatisfiable, and
- P refers only to the common variables of A and B .

Optimization-Interpolant-based method



Optimization-Interpolant-based method



- **Open questions:**
 - $A = ?$
 - $B = ?$
 - $P = \text{Fault-span}$

Conclusion & Future work



- How to effectively utilize features of automated synthesis algorithm to encode the problem in SAT-accepted format is very important.
- How to utilize SAT solver is very important and needs further exploration. (e.g. interpolant-based method)
- An interesting work direction is to utilize checking satisfiability for QBF to do fault-span generation



- QA