SAT-Based Model Checking Without Unrolling

Author: Aaron R. Bradley
Reporter: Xie Li

April 12, 2022

Compute Largest Inductive Subclause

$$\psi \wedge c \wedge T \rightarrow c'$$

IC3: The Main Function

The main function:

bool prove():
if sat($I \wedge \neg P \vee I \wedge T \wedge \neg P'$) **then**
    return false
end if
$F_0 := I$, clauses$(F_0) := \emptyset$
$F_i := P$, clauses$(F_i) := \emptyset$ for all $i > 0$.
for $k := 1$ to $\cdots$ **do**
    if not strengthen(k) **then**
        return false
    end if
    propagateClauses(k)
    if clauses(F_i) = clauses(F_{i+1}) for some $1 \leq i \leq k$ **then**
        return true
    end if
end for

IC3: Strengthen Function

bool strengthen(k : level):
try:
while sat$(F_k \wedge T \wedge \neg P')$ **do**
    $s :=$ the predecessor extracted from witness
    $n :=$ `inductiveGeneralize`$(s, k-2, k)$ // why $k-2$?
    `pushGeneralization`$(\{(n+1, s)\}, k)$
end while
return true
except Counterexample: **return false**

IC3: Inductive Generalization Function

level `inductiveGeneralize`(s : state, min: level, k : level):
if $min < 0$ **and** sat$(F_0 \wedge T \wedge \neg s \wedge s')$ **then**
    raise Counterexample
end if
for $i := \max(1, min + 1)$ to k **do**
    if sat$(F_i \wedge T \wedge \neg s \wedge s')$ **then**
        `generateClause`$(s, i-1, k)$
        return $i - 1$
    end if
end for
`generateClause`$(s, i-1, k)$
return k

void `generateClause`(s, i, k): find inductive subclause of $\neg s$ for $F_0, F_1, \ldots, F_{i+1}$ and conjoin it to them respectively.

IC3: pushGeneralization Function

void `pushGeneralization`$(states$: (level ,state) set, k : level):
while true do
    $(n, s) :=$ choose from $states$, minimizing n.
    if $n > k$ **then**
        return
    end if
    if sat$(F_n \wedge T \wedge s')$ **then**
        $p :=$ the predecessor extracted from the witness
        $m :=$`inductivelyGeneralize`$(p, n-2, k)$// same reason for $n-2$
        $states := states \cup \{(m+1, p)\}$
    else
        $m :=$`inductivelyGeneralize`(s, n, k)
        $states := states \backslash \{n, s\} \cup \{(m+1, s)\}$
    end if
end while

A Demo Example

A Demo Example

Total Correctness

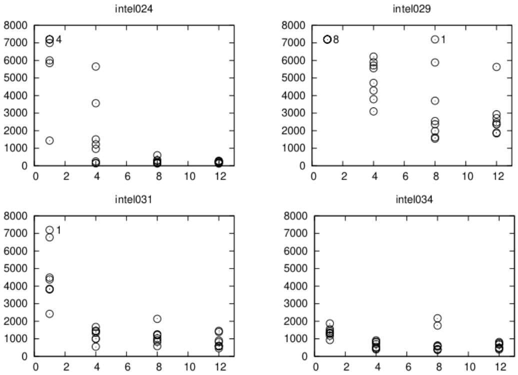
Theorem

For finite transition system S and safety property P the algorithm terminates and it returns true if and only if P is S-invariant.

Parallel Implementation

Finding inductive subclauses can be done in parallel and communicate through central server.

Experimental result:



Further Questions

- What about infinite transition system?
- What about more general transition system?