Analysis of Boolean Programs

Patrice Godefroid

Microsoft Research

Mihalis Yannakakis

Columbia University

What is a Boolean Program?

- All variables have Boolean type, recursive procedures
- Ex:

```
bool[8] x; // 8-bit global variable
procedure foo()
{
  print(``a'');
  if (x>0)
  {
    x = x-1;
    foo();
  }
  print(``b'');
  return;
}
```

Both programs print a^n b^n

```
procedure bar(bool[8] y)
{
  print(``a'');
  if (y>0)
  {
    bar(y-1);
  }
  print(``b'');
  return;
}
```

local var (saved on stack)

- Exponentially more succinct than pushdown automata
- Popular abstract domain for static SW model checking
  - Ex of tools: SLAM, BLAST, YASM, TERMINATOR, YOGI,…
  - Precise control-flow representation
  - Data part represented by Bool. predicates (predicate abstraction)
  - Many interesting properties still decidable

Analysis of Boolean Programs

- Prior work: several algorithms
  - For reachability, LTL model checking, …
  - Run in time exponential in the program size, or worse
  - Often no detailed complexity analysis
  - No lower bounds (can one do better?)

- This work: study of the worst-case complexity of
  - Reachability, cycle detection, LTL, CTL, CTL* model checking
  - For boolean programs and particular sub-classes
    - Deterministic, hierarchical (no recursion), acyclic, I/O bounded,…
  - We present upper bounds and matching lower bounds in all cases
    - All our algorithms are optimal in complexity-theoretic sense
  - Note: different from prior results on pushdown automata

Boolean Programs = ERSMs

- ERSMs = Extended Recursive State Machines
- ERSM generalizes:
  - RSM: Recursive State Machine (with no Boolean variables)
  - EHSM: Extended Hierarchical State Machine (no recursion)
  - HSM: Hierarchical State Machine (EHSM with no variables)
  - EFSM: Extended Finite State Machine (one procedure only)
  - FSM: Finite State Machine (one procedure and no variables)
- Other particular cases considered:
  - I/O bounded: number of I/O (local and global) vars $< c \,.\, \log|A|$
    where c is some fixed constant and $|A|$ is the size of the program
  - Deterministic programs, acyclic programs,…

Reachability Analysis: Results

Theorem: Reachability for ERSMs is EXPTIME-complete

- even for deterministic, acyclic ERSMs

Proof:

- Upper bound: from prior work
- Lower bound: with a

Boolean program simulating an alternating PSPACE machine

```
procedure Top()
{
  if Acc(q0, 0, Initial Tape)
     then print(‘‘M accepts’’);
}

bool Acc(state q, head location h, Tape T)
{
  if (q in Q_T) then return true;
  if (q in Q_F) then return false;

  bool res;
  if (q in Q_∃) then res = false;
  else res = true; // case (q in Q_∀)

  for each (q',s,D) in δ_M(q,T[h])
  {
    compute new tape location h' and tape T';
    if (q in Q_∃) then res=res∨Acc(q',h',T');
    else res=res∧Acc(q',h',T');
  }
  return res;
}
```

Reachability: Particular Cases

Other results:

Class of Program	Restriction	General Case	I/O Bounded
ERSM		EXPTIME	PSPACE
EHSM		PSPACE	PSPACE
EHSM	nondeterministic acyclic	PSPACE	NP
EHSM	deterministic acyclic	PSPACE	P

For acyclic EHSMs of bounded depth: NP-complete

LTL Model Checking

Theorem: The program complexity of LTL model checking is the same as for reachability analysis, for ERSMs and all the previous sub-classes considered

Proofs:

- Automata-theoretic approach (standard)
  - Negation of LTL formula -> Buchi automaton
  - Product construction
  - Detect a cycle or infinite stack that is accepting
- Lower bounds: derived from reachability results
- Upper bounds:
  - Easy cases by reduction to non-extended cases (RSMs etc.)
    Ex: ERSM case is EXPTIME-complete
  - Harder cases: new algorithms (with automata-theoretic approach)
    Ex: I/O Bounded ERSM is PSPACE-complete

Branching-Time Properties

Theorem: The program complexity of CTL model checking for ERSMs is 2EXPTIME-complete

Proof:

- Upper bound: easy (reduction to RSM CTL model checking)
- Lower bound:
  - using a nondeterministic Boolean program simulating an alternating EXPSPACE machine (see next slide)
  - and the CTL formula E(C -> EX(CheckMode $\wedge$ AF(OK)) U Success)

Boolean prgm simulating an alt EXPSPACE machine

```
Global variables:
g_s, g_s', s_new: previous/next/temporary symbol in
$\Sigma$ ($log(|\Sigma|)$ bits)
g_q': current state ($log(|Q|)$ bits)
g_h, g_h': previous/next location for the tape head (n bits)
g_d: depth (is either 0, 1, 2)
j: cell location (n bits) or UNDEF
T[j],T'[j]: symbol in $\Sigma$ ($log(|\Sigma|)$ bits) or UNDEF
// 2 symbols, not arrays
OK=false, CheckMode=false, Success=false: boolean
variables (false by default)

Top()
{
 j=UNDEF; T[j]=UNDEF; T'[j]=UNDEF;
 if Next($q_0$,0,$x_0$,0) then Success=true
 STOP;
}

bool GuessNextTapeCell(tapeLocation i, symbol s)
{
 boolean ret;
 if (g_h'==i) then g_s' = s; // record in g_s' the next symbol
read from the next location h'
 if (i<(2^n -1))
 {
   if (g_h==i+1) then s_new = g_s; // new symbol just written
at the previous location h
   else s_new = nondeterministically pick a symbol in
$\Sigma$; // $\exists$-nondeterminism
   ret=GuessNextTapeCell(i+1,s_new); // put s_new on the
stack of the ERSM
 }
 else
   ret=Next(g_q',g_h',g_s',g_d');
 if (CheckMode $\wedge$ i==j) then T[j]=s;
 return ret;
}
```

```
bool Next(state q, headLocation h, symbol s, depth d)
{
C:if (nondeterminism) then CheckMode=true; // in C:, $EX (CheckMode \wedge AF(OK))$ must
hold

 if (CheckMode)  // start CheckMode -- this is executed at most once!
 { // we check that the last 2 tape contents T and T' (last) are $\delta_M$-compatible
   if (d==0) then { OK=true; STOP; } // nothing to check
   j= nondeterministically pick a cell location // $0\leq j<2^n$ -- $\forall$-nondeterminism due to
$AF(OK)$
   return false; // dummy return value in this mode; start popping to get T'[j] and T[j]
 }
 if (q in $Q_T$) then return true;
 if (q in $Q_F$) then return false;
 boolean result;
 if (q in $Q_\exists$) then result=false;
 else result=true; // case where q in $Q_\forall$
 boolean ret;
 for each (q',s',D) in $\delta_M$(q,s) // with s=T[h]
 {
   if (D==L) then h '= h-1 else h' = h+1;  // set h' = new head location
   if (d<2) then g_d'=d+1;  // note: d is either 0, 1 or 2
   else g_d'=d;
   g_q' = q'; g_h' = h';     // global variables for next call of Next()
   g_s = s'; g_h = h;        // global variables for this call of Next()

   if (g_h==0) s_new = g_s;
   else s_new = nondeterministically pick a symbol in $\Sigma$; // $\exists$-nondeterminism

   ret=GuessNextTapeCell(0,s_new);

   if (CheckMode)
   {
    if (T[j]!=UNDEF $\wedge$ T'[j]==UNDEF) then // we got T'[j]
    {
      T'[j]=T[j];
      if (d>0) then return false; // continue popping to get T[j]
      else { T[j]=$x_j$; h'=h; }
    }
    // we are ready to check $\delta_M$-compatibility at position j
    if ((j!=h') $\wedge$ T'[j]==T[j]) then OK=true;  // the tape cell content must be unchanged
    if (j==h') then OK=true;    // nothing to check -- case enforced by construction
    STOP;
   }
   if (q in $Q_\exists$) then result = result $\vee$ ret;
   else result = result $\wedge$ ret;
 }
 return result;
}
```

Particular Cases, Other Results

- For deterministic ERSMs, the program complexity of CTL model checking is “only” EXPTIME-complete
- For EHSMs (deterministic or not): PSPACE-complete
  - Same as for HSMs !
- The program complexity of CTL* model checking is
  - 2EXPTIME-complete for ERSMs
  - EXPTIME-complete for deterministic ERSMs
  - PSPACE-complete for EHSMs

  (same program complexity as for CTL in all 3 cases)

Summary of Results

- Complexity bounds in the size of the program:

Class of Program	Restriction	LTL	CTL
FSM		Linear	Linear
EFSM		PSPACE	**PSPACE**
HSM		Linear	PSPACE
HSM	deterministic	Linear	**Linear**
EHSM		**PSPACE**	**PSPACE**
EHSM	deterministic	**PSPACE**	**PSPACE**
RSM		Cubic	EXPTIME
RSM	deterministic	**Linear**	**Linear**
ERSM		**EXPTIME**	**2-EXPTIME**
ERSM	deterministic	**EXPTIME**	**EXPTIME**

  - For CTL, deterministic Boolean programs are exponentially easier compared to nondeterministic ones, except for EHSMs
  - CTL harder than LTL for nondeterministic HSMs, RSMs, ERSMs

Visual Summary for Main Classes

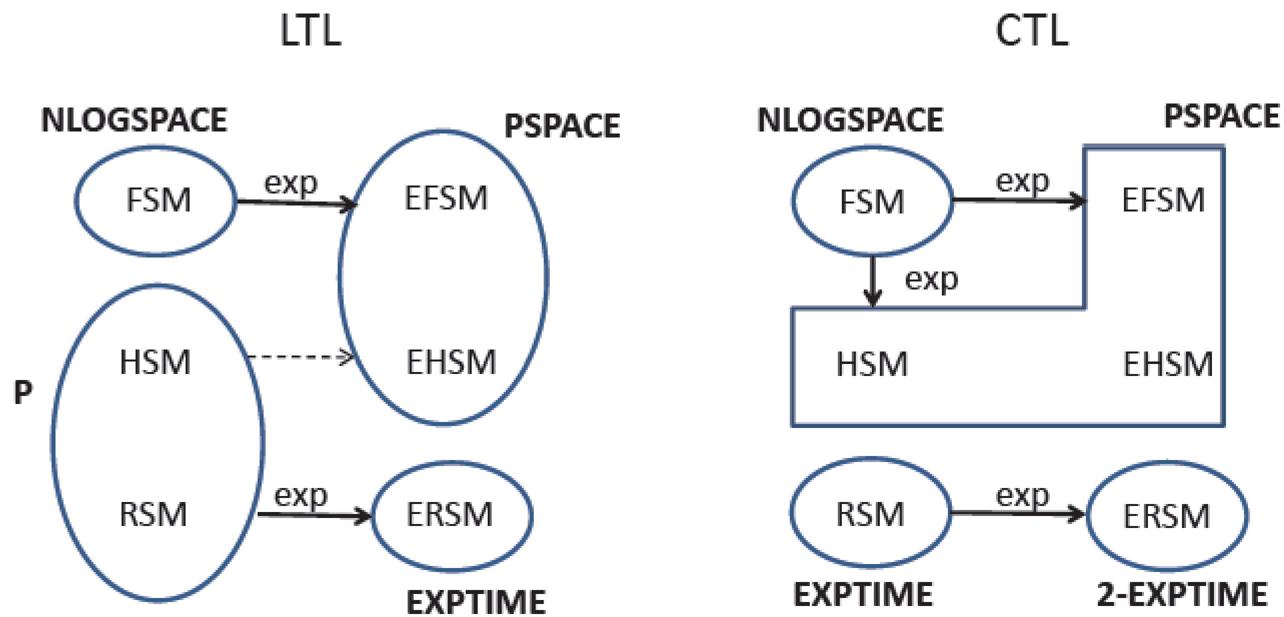

Fig. 4. Visual summary for the program complexity of LTL and CTL model checking.

- Adding Boolean variables ("E" extension):
  - exponentially more succinct
  - But not uniformly exponentially harder !
- See the cost of adding hierarchy, adding recursion

Impact on Logic Encodings

- These results shed new light on logic encodings for (classes of) Boolean programs
  - for VC-gen, SAT/SMT-based bounded model checking
- Example: reachability for EHSMs is PSPACE-complete
  - No precise polyn.-size encoding of EHSMs in propositional logic
  - But possible in QBF (can be reduced to QSAT)
- Example: reachability for acyclic EHSMs of bounded depth is NP-complete
  - Possible precise polynomial-size encoding in propositional logic (can be reduced to SAT)

Conclusion

- Boolean programs: natural program representation
  - Simple, elegant, concise, popular, useful
    - Used in static abstraction-based software model checking tools
  - Generalizes other representations (E/FSMs, HSMs, RSMs,…)
  - Interesting properties (this work!)
- This paper: 1^{st} comprehensive study of the worst-case complexity of basic analyses of Boolean programs
  - Reachability, cycle detection, LTL, CTL, CTL* model checking
  - Matching upper and lower bounds for all these problems
  - Sub-classes: explain what features contribute to complexity
    - Nondeterminism, cycles, variables, hierarchy, recursion, I/O-bound,…