CS 6110 Software Correctness, Spring 2022
Lec7

Ganesh Gopalakrishnan
School of Computing
University of Utah
Salt Lake City, UT 84112

URL: bit.ly/cs6110s22

SCHOOL OF COMPUTING
THE UNIVERSITY OF UTAH

Slides for Lec7 : Agenda

- Review of material thus far
- (If time permits): Intro to Murphi

Summary of basic topics in Logic studied

- Review of Logic
  - Why review logic?
    - Logic is the "calculus of computer science"
  - Why focus on propositional logic (Boolean Logic)?
    - That is how most problems are modeled
  - Why SAT?
    - This is one of the central problems in CS
    - This is also one of the central success stories in verification
- Tseitin transformation, equisat, equivalence
  - Help review notions coming in later chapters
  - Helped us reinforce our readings of Bradley/Manna
  - Tseitin transformation is central to SMT – a core verification success story
- CNF, DNF
    - CNF preferred over DNF because of
      - Size-explosion possible if constraints arising in modeling problems turned into DNF
      - Naturalness of constraints modeled (thus, a majority of SAT-solvers are CNF-solvers)
- BDDs and minimal DFA
  - BDDs are canonical representations of Boolean functions
  - We could visualize whether a formula is valid or satisfiable
- Conversion of DNF to CNF via BDDs
  - This was not Tseitin transformation … but involved tracing paths to the "0" node

Review question

Q1: When a DNF formula

- a.b.c.d + a.b.c.!d + a.b.!c.d + a.b.!c.!d + …. + !a.!b.!c.d + !a.!b.!c.!d
- Is converted to CNF by distributing + over . , and vice-versa,

(a) The resulting CNF has

1. The same size
2. much higher size (typically exponential wrt the input formula size)

(b) The resulting CNF is

1. Equisat
2. Equivalent (pick the stronger of the two claims)

Select answers for Q1(a) [1 or 2] and Q1(b) [1 or 2]

Review question

Q2: When a BDD for a function has paths leading to ‘0’ labeled by

- a.b.c ,
- a.c.!d , and
- !c.d

- Then, the equivalent CNF for that BDD is
  - …fill your answer …

Review question

Q3: The size of a BDD depends on these; also this could be true

1. The order of the variables chosen
2. The nature of the Boolean function
3. Often we can improve the initial order obtain through an algorithm called sifting that locally rearranges variables to shrink the BDD
4. If the problem of determining the optimal variable for a BDD is solved in P-time, then we would have shown P=NP. (And if I don't know what NP-completeness means, I promise I will make it a point to have a discussion about it on Piazza!)

Review question

Q4: These are true facts about BDDs and DFA

1. BDDs are minimal DFA in disguise. (And if I don't understand DFA minimization, I'll make it a point to ask on Piazza.)
2. While BDDs tend to resemble decision trees, there is one crucial difference: the order of variables going from root to leaf could be different for different paths in a decision tree. (There are also other differences.)
3. For some functions such as addition, BDDs tend to be always compact if one chooses the right var order
4. For some functions such as multiplication, BDDs are guaranteed exp (think of the Boolean function for the middle-answer-bit of the product of two numbers)

Review question

Q5: BDDs are

1. Just a curiosity
2. Were predominantly used for formal verification till about year 2000 when Boolean SAT tools started taking over
3. BDDs tend to saturate at 1000 variables (depending on the problem)
4. Boolean SAT can handle millions of variables, although there are cases where SAT can become expensive at lower number of variables. Formal verification has largely become a success story thanks to advances in SAT (SAT-solvers have become four orders of magnitude faster in the last two decades.)

Review question

Q6: Two formulae f1 and f2 are equivalent if

1. f1 -> f2
2. f2 XOR f1 is false

Q7: Two formulae f1 and f2 are equisat if

1. For all assignments sigma : sigma |= f1 if-and-only-if sigma |= f2
2. f1 is satisfiable IF-AND-ONLY-IF f2 is satisfiable
   - Which boils down to this
     - exists sigma1 : sigma1 |= f1 ⇔ exists sigma2 : sigma2 |= f2
   - Remember that in Equisat, f1 and f2 need not have the same set of variables

Review question

Q8: Consider f1 = a.!a

and f2 = (!a+a+p)(!p+a)(!p+!a)

Show that f1 and f2 are not equisat

Review question

Q9: Consider f1 = a.!a

and f2 = (!a+a+p)(!p+a)(!p+!a).(p)

Now show that f1 and f2 are equisat

Observation

Q8,Q9 gist: the main practical goal of equisat transformations is to preserve unsat in the transformation chain ☺

Basic topics in LTL and model-checking

- When we want to verify systems, we often create models
  - They are like airplane scale models flown in wind tunnels
- Experience shows that when you create a finite-state model of the “real system” and exhaustively examine the finite-state model, you often find corner cases that are missed when testing the real system
  - i.e. shrink the real system
    - By preserving all corner-cases
      - This is an art-form
      - This is error-prone → no guarantees
    - BUT
      - When you do it well, and you find a bug, then you have found a bug in the original
- The idea of model-checking is to find bugs
  - NOT to prove a system correct
    - That is just too ambitious … can be done,but is a taller order!
      - We will write inductive-assertions proofs of loop programs later

Basic Model-Checking

- System model = async product of proctypes (in Promela)
- Property
  - Safety or Liveness
- Safety Property
  - Any property whose violation is a finite trace
- Liveness Property
  - Any property whose violation is an infinite trace
    - NOTE THAT infinite traces in a finite-state model are cycles
- Thus
  - Safety violation is presented as a finite trace
  - Liveness violation is presented as a bad cycle

What do bad cycles look like

- An example
  - Request ; ack ; Request ; ack ;
    …do this 4 times… ; [then] Request ; <a cycle of no acks>

What is fairness?

- Fairness properties are liveness properties
  - Thus an unfair execution is a bad cycle
- We looked at two “classical” notions of fairness
  - Just to know they exist … no deeper look now
  - Justice
  - Compassion
- We will revisit them soon
- But there are many other “fairness” properties that one can define (we will see more examples soon)

Linear-time Temporal Logic

- A logical system where one can produce truth-values based on infinite executions
  - i.e. based on executions that have cycles

- A ready way to produce a “big bag of cyclic executions” is thru a Kripke Structure

- Logic needs a STRUCTURE for interpretation
  - Hence we prefer to standardize executions into Kripke Structures

LTL from Rozier's paper

Definition 1. For every $p \in Prop$, p is a formula. If φ and ψ are formulas, then so are:

$$
\begin{array}{ccccc}
\neg\varphi & \varphi \wedge \psi & \varphi \rightarrow \psi & \varphi\ \mathcal{U}\ \psi & \Box\varphi \\
 & \varphi \vee \psi & \mathcal{X}\varphi & \varphi\ \mathcal{R}\ \psi & \Diamond\varphi
\end{array}
$$

LTL satisfaction from Rozier's paper

- $\pi, i \models p$ for $p \in Prop$ iff $p \in \pi(i)$.
- $\pi, i \models \neg\varphi$ iff $\pi, i \not\models \varphi$.
- $\pi, i \models \varphi \wedge \psi$ iff $\pi, i \models \varphi$ and $\pi, i \models \psi$.
- $\pi, i \models \varphi \vee \psi$ iff $\pi, i \models \varphi$ or $\pi, i \models \psi$.
- $\pi, i \models \mathcal{X}\varphi$ iff $\pi, i+1 \models \varphi$.
- $\pi, i \models \varphi \,\mathcal{U}\, \psi$ iff $\exists j \geq i$, such that $\pi, j \models \psi$ and $\forall k, i \leq k < j$, we have $\pi, k \models \varphi$.
- $\pi, i \models \varphi \,\mathcal{R}\, \psi$ iff $\forall j \geq i$, iff $\pi, j \not\models \psi$, then $\exists k, i \leq k < j$, such that $\pi, k \models \varphi$.
- $\pi, i \models \Box\varphi$ iff $\forall j \geq i$, $\pi, j \models \varphi$.
- $\pi, i \models \Diamond\varphi$ iff $\exists j \geq i$, such that $\pi, j \models \varphi$.

LTL satisfaction of a model (from Rozier)

We now restate the model-checking problem: program M satisfies ("models") formula φ iff every path π rooted at the initial state q of M satisfies φ, denoted $M, q \models \varphi$.

LTL examples (Rozier)

Examples of LTL properties:

- *Liveness: "Every request is followed by a grant"*
  $\Box(request \rightarrow \Diamond grant)$
- *Invariance: "At some point, p will hold forever"*
  $\Diamond\Box p$
- *"p oscillates every time step"*
  $\Box((p \wedge \mathcal{X}\neg p) \vee (\neg p \wedge \mathcal{X} p))$
- *Safety: "p never happens"*
  $\Box\neg p$
- *Fairness: "p happens infinitely often"*
  $(\Box\Diamond p) \rightarrow \varphi$
- *Mutual exclusion: "Two processes cannot enter their critical sections at the same time"*
  $\Box\neg(in\_CS_1 \wedge in\_CS_2)$
- *Partial correctness: "If p is true initially, then q will be true when the task is completed"*
  $p \rightarrow \Box(done \rightarrow q)$

Which are safety? Which are liveness?

We will
Go by
What the
Evidence of
Failure
Looks
Like
(is it a
Non-cyclic
Trace?
Is it a cycle?)

We can negate
EACH OF THESE!

And find a
Satisfying
Trace / cycle!

Examples of LTL properties:

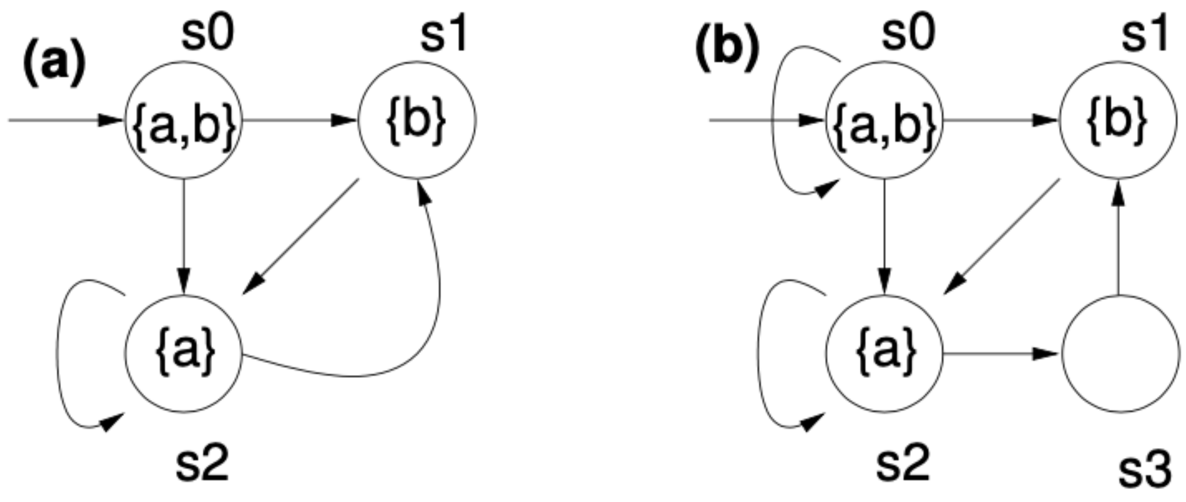
- *Liveness: "Every request is followed by a grant"*
  $\Box(request \rightarrow \Diamond grant)$
- *Invariance: "At some point, p will hold forever"*
  $\Diamond\Box p$
- *"p oscillates every time step"*
  $\Box((p \wedge \mathcal{X}\neg p) \vee (\neg p \wedge \mathcal{X} p))$
- *Safety: "p never happens"*
  $\Box\neg p$
- *Fairness: "p happens infinitely often"*
  $(\Box\Diamond p) \rightarrow \varphi$
- *Mutual exclusion: "Two processes cannot enter their critical sections at the same time"*
  $\Box\neg(in\_CS_1 \wedge in\_CS_2)$
- *Partial correctness: "If p is true initially, then q will be true when the task is completed"*
  $p \rightarrow \Box(done \rightarrow q)$

How did I “hack” distinguishing formulae?

- I hacked distinguishing formulae in CEATL – let’s see how

How did I "hack" distinguishing formulae?

- I hacked distinguishing formulae in CEATL – let's see how



Sb satisfies this and not Sa
I homed into the path that EXCLUDED
The ab loop

Then I summarized all the remaining
Paths!

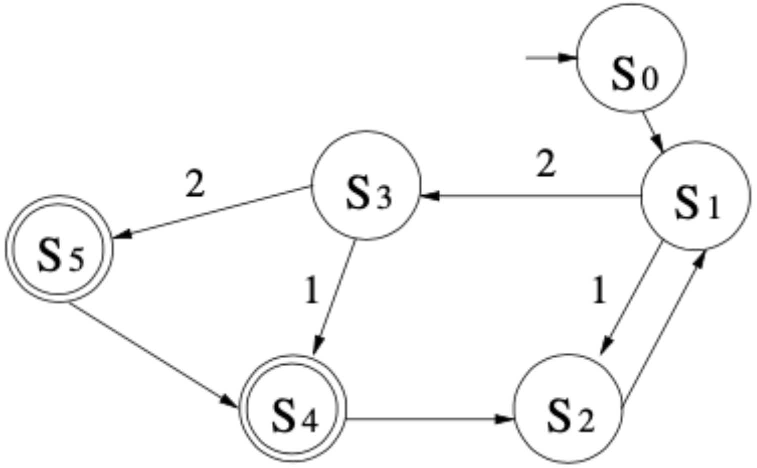
```
!(<>([](a && b))) -> ((<>([]a)) || (<> (!a && !b)))
```

How does bad cycle detection work in SPIN?

Hence,

$$L(S) \cap \overline{L(P)} \neq \emptyset.$$

When this condition is true, a bug has been found (*i.e.*, the property has been violated). This accepting run can be displayed as an error-trace or a MSC. Consequently, the debugging of concurrent systems can be reduced to emptiness checking of Büchi automata.



S_0	S_0	S_0	S_4	S_4
S_1	S_1	S_1	S_2	S_2
S_2	S_3	S_3	S_1	S_1
S_1	S_4	S_4	S_2	S_3
	S_2			S_4

Perform a Nested Depth-first Search

(brief walk-thru)

Now we looked at practical stuff

- Bubble-sorting
  - The 0/1 theorem for sorting networks
    - The analogy for programs is fairly OK
      - But can we build a bubbling argument with 0's and 1's?
        - I am going to try as follows

        - … explain…

- The Philosophers in Promela
- Ways to run SPIN

- Then the distributed termination
- What was the amazingly clever bug I put into Dist. Term?