CpSc 538G: Course Overview

Mark Greenstreet

September 7, 2016

Outline:

- What is verification?
- Course mechanics
  - You will survive.
  - You'll even have fun.

What is verification?

- We use mathematical methods, mostly mathematical logic, to show that a hardware, software, or cyber-physical design has some desired properties.
  - Often, this is seen as “finding bugs.”
  - Bug finding can be very important from a safety and/or cost point of view.
  - Formal methods also allow us to build more highly optimized designs:
- The kinds of techniques that we use:
  - Boolean satisfiability (SAT)
  - SAT augmented with decision procedures for other domains (SMT)
  - Reachability computation: model checking
  - Symbolic execution, interpolation, abstraction, approximation.

A bit more about SMT

- An SMT solver takes a formula written with:
  - Boolean variables and operations.
  - Integer variables, arithmetic, and comparisons.
  - Real-number variables, arithmetic, and comparisons.
  - Plus some other types such as enumerations, lists, bit-vectors.
- The SMT solver determines if the formula is satisfiable
  - Can it find values for the variables that satisfy the formula?
  - If "yes", the SMT solver provides an example.
  - If "no", some solvers can provide refutation proofs.
  - It may also say "unknown" – that means the heuristics failed – perhaps due to resource bounds.

Z3 booleans

```
1>> [a, b] = Bools(['a', 'b'])
2>> x = And(a, b)
3>> y = Not(Or(Not(a), Not(b)))
4>> prove(x == y)  # De Morgan's Law
proved
5>> solve(x)
[b = True, a = True]
```

- `[a, b] = Bools(['a', 'b'])` creates two Z3, boolean variables named `'a'` and 'b' and assigns them to the python variables `a` and `b`.
- `x = And(a, b)` creates a symbolic expression for the conjunction of `a` and `b` and assigns that symbolic expression to the python variable called `x`.

Z3 booleans

```
6>> [a, b] = Bools(['a', 'b'])
7>> x = And(a, b)
8>> y = Not(Or(Not(a), Not(b)))
9>> prove(x == y)  # De Morgan's Law
proved
10>> solve(x)
[b = True, a = True]
```

- `prove(x == y)` checks that the claim holds for all values of the symbolic variables (i.e. `'a'` and `'b'`). It does this by showing that `Not(x == y)` is unsatisfiable.
- `solve(x)` finds a solution to the formula `x`. In other words, it finds values of the symbolic variables `'a'` and `'b'` such that `And(a, b)` is true.

Z3 integers

```
11>> [i, j, k] = Ints(['i', 'j', 'k'])
12>> solve(And(0 < i, 0 < j, 0 < k, \
                i*i + j*j == k*k))
[k = 15, j = 9, i = 12]
13>> solve(And(0 < i, i < 100, \
                0 < j, j < 100, \
                0 < k, k < 200  \
                i*i + j*j == k*k))
no solution
14>> solve(And(0 < i, i < 100, \
                0 < j, j < 100, \
                0 < k,          \
                i*i + j*j == k*k))
no solution # but it took about 1.5 minutes
```

Uninterpreted Functions

```
15>> f = Function("f", IntSort(), IntSort(), IntSort())
16>> solve(f(i,j) == i+j))
[i = -38, j = 38, f = [(-38, 38) -> 0, else -> 0]]
```

- `f = Function("f", IntSort(), IntSort(), IntSort())` says, "I'm thinking of a function. Call it `f`. `f` has two parameters that are both integers, and `f` returns a value that is an integer."
- `f(i,j) == i+j)`
  Says, the function that I'm thinking of has the property that for the specific values tht you choose for `i` and `j`, it must be the case that `f(i,j) == i+j`.
- `solve(f(i,j) == i+j))` asks Z3 to find at example of a function that satisfies these constraints.
  - In this case, it found a solution with `i = -38`, `j == 38`, and `f(*,*) -> 0`. I.e., `f` produces `0` no matter what its arguments are!

Uninterpreted Functions

```
17>> f = Function("f", IntSort(), IntSort(), IntSort())
18>> solve(f(i,j) == i+j))
[i = -38, j = 38, f = [(-38, 38) -> 0, else -> 0]]
```

- `f = Function("f", IntSort(), IntSort(), IntSort())` says, "I'm thinking of a function. Call it `f`. `f` has two parameters that are both integers, and `f` returns a value that is an integer."
- We'll see that such functions can be useful, but they're not the functions that you are used to.

Just for fun

I found this puzzle at http://www.logic-puzzles.org:

- Witching Hour starts sometime before the program on channel 7.
- Nate Nichol's program airs on channel 4.
- Ken Kirby's program doesn't begin at 11:45 pm.
- Jack Jensen's show starts 30 minutes before Penny Pugh's show.
- The show on channel 4 starts sometime after the show on channel 13.
- Jack Jensen's program is Tonight Again.
- ...

See http://www.cs.ubc.ca/~mrg/cs538g/2016-1/notes/09-07/tv.py for the full description.

Step 1: find the sets

Hosts: { Jack Jensen, Ken Kirby, Lina Lopez, Maddy Meyer, Nate Nichol, Oda Osborn, Penny Pugh }

Titles: { Green Tonight, Late Night, Red Eye Party, Tonight Again, Up To Two, Variety X, Witching Hour }

Channels: { 2, 4, 5, 7, 9, 11, 13 }

Times: { 10:15pm, 10:30pm, 10:45pm, 11:00pm, 11:15pm, 11:30pm, 11:45pm }

In Z3, we use `EnumSort`

```
Host, (Jack, Ken, Lina, Maddy, Nate, Oda, Penny) = \
  EnumSort("Host", \
           ("Jack Jensen", "Ken Kirby", "Lina Lopez", \
            "Maddy Meyer", "Nate Nichol", "Oda Osborn", \
            "Penny Pugh"))
```

- I'm using the Python bindings for the Z3 API.
- `EnumSort(`*SortName*`, (`*ElementNames*`, ...))` creates a new enumeration sort (a "sort" is the Z3 representation for a "type" with the Elements given. It returns a tuple consisting of the new sort, and a list of the elements for that sort.
- I created a sort for the titles in the same way.

Sets of integers

- Z3 support integers.
- I represent sets as a tuple listing the elements.
- I also give such sets names – these will come in handy in a moment.
- In particular

```
Channel = ("Channel", (2, 4, 5, 7, 9, 11, 13))
# I'll define time as minutes after 10pm
Time = ("Time", [x*15 for x in range(1,8)])
```

Mappings

- I want to say that there is a one-to-one mapping between any pair of the four sets defined above.
- It's sufficient to require that there by one-to-one mappings between the set `Time` and each of the three other sets.
  - I chose `Time` as my "canonical" representation because it will be handy to perform arithmetic on `Time` values.
- Uninterpreted functions let us say that a mapping is one-to-one.

One-To-One Mappings

To define a one-to-one mapping between sorts `P` and codeQ:

- Declare two, uninterpreted functions:

```
p_to_q = Function("P_to_Q", P, Q)
q_to_p = Function("Q_to_P", Q, P)
pq = True;
for each value v of P:
    pq = And(pq, q_to_p(p_to_q(v)) == v)
```

- Note: `P` and `Q` must have the same number of elements.
- Why does this ensure that `p_to_q` and `q_to_p` are one-to-one?
- This is tedious to type; so I wrote:

```
one_to_one_fun(sort1, sort2)
```

  in `puzzle_utils.py` that returns the two functions and their constraints.

Puzzle Mapping

```
t_host,    host,    f1 = one_to_one_fun(Host, Time)
t_channel, channel, f2 = one_to_one_fun(Channel, Time)
t_title,   title,   f3 = one_to_one_fun(Title, Time)
clues = [f1, f2, f3]
```

The Clues

```
#  1. Witching Hour starts sometime before the program on channel 7.
clues.append(t_title(wh) < t_channel(7))
#  2. Nate Nichol's program airs on channel 4.
clues.append(t_host(Nate) == t_channel(4))
#  3. Ken Kirby's program doesn't begin at 11:45 pm.
clues.append(t_host(Ken) != t11_45)
#  4. Jack Jensen's show starts 30 minutes before Penny Pugh's show.
clues.append(t_host(Jack) == t_host(Penny) - 30)
...
```

Solving the Puzzle

```
s = Solver()
s.add(And(*clues))
m = s.model()
for tt in Time[1]:
    ti = str(m.eval(title(tt)))
    ho = " is hosted by " + str(m.eval(host(tt)))
    ch = " on channel " + str(m.eval(channel(tt)))
    tm = str("10 " + (tt/60)) + ":" + \
    ("%(minutes)02d" % { "minutes" :  (tt % 60)})
    print ti + ho + ch + tm
```

- I also added code to make sure the solution is unique.

For more fun

- http://www.cs.ubc.ca/~mrg/cs538g/2016-1/notes/09-07/tv.py
  This example.
- http://www.cs.ubc.ca/~mrg/cs538g/2016-1/notes/09-07/puzzle_utils.py
  The utilities for solving puzzles. Really, it's just the `one_to_one_fun` function described on slide 12.
- http://www.cs.ubc.ca/~mrg/cs538g/2016-1/resources/guide-examples.htm
  An introduction to Z3 by Leo Moura.
- https://docs.python.org/2/tutorial/index.html
  A Python tutorial. Note that the Python Z3 bindings require Python 2.7. They won't work with Python 3.

Course mechanics

- We'll cover roughly one paper per lecture.
  - The reading list is at
    http://www.cs.ubc.ca/~mrg/cs538g/2016-1/ReadingList.html
  - Starting Sept. 14, you will be expected to write a short summaries for the papers. See slide 19.
- Each person will present one paper: see slide 21.
- There will be 4 ± 1 homework assignments: see slide 23.
- There will be a project: see slide 24.
  - Should take $\sim 40 - -60$ hours to your time.
  - The final output is a M.Sc. thesis proposal (or equivalent).

Grading

- 10% paper summaries.
- 10% class presentation.
- 40% homework.
- 40% course project.

Paper summaries

- Each summary should address each of the questions below:
  1. What problem does the paper address?
  2. What is the key insight/idea in the paper's solution to this problem?
  3. What did the authors do to demonstrate their claims? (e.g. implement a tool, present a proof, run simulations, etc.)
  4. Is the support for the claims convincing? Why or why not?
  5. What are your questions or other comments about the paper?
- You can address each point with 1–3 sentences. These summaries should be short: at most one page.
- Paper summaries are due at 9am of the day that the paper is covered in class.
  - Submit your summaries by e-mail.
  - Plain text is preferred. PDF is acceptable.
- If you found a paper too hard to read:
  - Write a description of where you got stuck.
  - Write some questions that would help you understand the paper.

Paper summaries – Grading

- I'll grade each summary on a scale of 0-5 according to the five questions mentioned above.
  - If you think that one or more of the standard questions don't apply to a particular paper, just say why, and give a summary that makes sense for the paper.
  - Summaries are pretty much graded pass/fail – the important thing is that you read the paper and think about it enough to write down a short summary.
- The overall score for summaries is

$$\frac{\sum_{\text{summaries}} \text{score}(\text{summary})}{(\text{number of class sessions with at least one paper}) - 1}$$

- The -1 is because there is no summary required for September 14.
- Note: on days that there is a student presentation, you can submit up to two summaries.
  - This lets you make up a missing summary.
  - The max score for summaries is 100%.

Paper Presentations

- Each person will present one paper.
- There are many “For further reading” papers proposed on the reading list. They are fine choices for papers to present.
- Or you can present a paper related to your project.
- Or you can present another paper.
- Claim a paper by sending e-mail to me:
  - Tell me what paper and what date.
  - The *first* person to claim a paper gets it.
  - I’ll send you a confirmation and update the reading list.

Presentations Guidelines

- A presentation should take about 20 minutes and be like a conference talk.
- An effective conference talk is an **advertisement** for the paper:
  - State why the problem matters.
  - State the main contributions.
  - Sketch how the contributions are validated — focus on one or two *interesting* points about what they did.
  - Add your own comments, questions, and criticisms. Connect the paper with other papers that we've studied in class.
- You can prepare slides and/or use the whiteboard.
  - I *strongly* recommend giving a practice run of your talk to another student.

Homework

- There will be 4 ± 1 homework assignments.
- Homework problems include:
  - Trying something with real, formal verification tools.
  - Some “pencil-and-paper” problems to complement the programming.
- The first assignment should go out on Sept. 14.

Projects

The goal of the project is to produce a Master's thesis proposal including:

- A statement of what problem you are addressing.
- A review of relevant research (at least five papers)
- A simple experiment (e.g. write some code) to show that your idea will probably be feasible.
- A timeline for the thesis research and write-up.
- Identify the any resources you would need:
  - Special equipment.
  - Access to proprietary data.
  - Anything else
- Identify where the risks are in the research and how you plan to handle them.
  - If you knew the results ahead of time, it wouldn't be research.

Project ideas

- I'm very happy to see projects for any aspect of formal verification.
- I also like projects that connect formal verification with other areas.
- See

  http://www.cs.ubc.ca/~mrg/cs538g/2016-1/ProjectIdeas.html

  for some project ideas.
- I prefer individual projects, but I'll consider a proposal for a group project if:
  - A clear reason is given for why the project should be done as a group and not as separate projects.
  - Clear criteria are given for evaluating each member's contribution to the project.

Project Deadlines

- October 26: proposals due.
  - State the problem you plan to address.
  - State what approach you expect to use to address the problem.
  - List at least three papers that you plan to include in your literature survey.
- November 23: intermediate report due.
  - List the papers that you have read.
  - Describe what the progress you have made on evaluating the feasibility of your idea. For example, if you're writing a program, describe what the status of developing that code.
  - Describe any issues that have come up that could impact the project.
  - This report should be 1 or 2 pages long plus short summaries of the papers that you've read.
- December 16: Final project report due.

Project Grading

$<$ 80: You didn't do what you proposed, and you didn't try to revise the project goals if you encountered some unforeseen difficulty. Note that you can always check with me to revise the project proposal if you need to.

80 – 84: You did what you proposed, but you didn't demonstrate that you explored the topic in a way so that you learned something significant in the process.

85 – 89: You clearly learned something significant by doing the project. Make sure that your report clearly states what you learned by doing the project.

90 – 94: I learned something significant by reading your report.

95 – 100: This work is worthy of writing a paper that I expect to be a landmark in the field.

Will I survive this class?

Yes.

- Formal verification spans many areas of computer science and other fields including:
  - Mathematical logic, programming languages, digital logic design, computational complexity, computer architecture, temporal logic, robotics, differential equations, optimization.
- I expect that most of these will be new to most students in the class
  - Lectures will include many tutorials.
  - I will emphasize showing the connections between what you already know and other branches of computer science.
- The goals of the course are:
  - Give you an introduction to formal methods so you can do research in the area if you want to.
  - Give you a background so you can see how these methods are useful in other areas where you may end up doing your thesis research or working after you graduate.
  - Have fun exploring how formal methods solve challenging, real-world problems.