Read these slides

Especially if you're skipping lecture. Don't be uninformed or stressed for no reason.
Looking forward

Lectures: SMT applied to programs, a little bit about proof.

*Guest lectures coming soon!* Attend! See Piazza for schedule.
Looking forward

Lectures: SMT applied to programs, a little bit about proof.

**Guest lectures coming soon!** Attend! See Piazza for schedule.

Three assignments left:

(1) Basic intro to the tool we're seeing today. **No** challenge problem.
(2) A roughly 2-hour exercise on proof. **Only** a challenge problem.
(3) A roughly 2-hour exercise on LTL

Decided to simplify so that you don't get overloaded and can work on projects...
Looking forward

Lectures: SMT applied to programs, a little bit about proof.

**Guest lectures coming soon!** Attend! See Piazza for schedule.

Three assignments left:

(1) Basic intro to the tool we're seeing today. **No** challenge problem.
(2) A roughly 2-hour exercise on proof. **Only** a challenge problem.
(3) A roughly 2-hour exercise on LTL

Decided to simplify so that you don't get overloaded and can work on projects...

...but please continue to come to lectures; last few non-guest are hopefully interesting stuff, mix of applications and logic.
Challenge problems and expectations

Remember: if you checked off Electrum 1 challenge, you may skip any challenge.

Simplified + concrete remaining challenge expectations (same reasons):

(1) Queue Lock: reasonable transitions, see traces, **think** about overflow
(2) SMT: written portion, one constraint in stencil (see Piazza).
(3) Proofs: expect straightforward, exercise in natural deduction.

This isn't a trick. If you just want the grade, do enough to understand the context and idea. If you're curious, go further.
Post-class comment

Someone asked why I'm giving expectations re: Queue Lock, and even talking about it a bit, if it was technically due yesterday. A few reasons:

- There's an important educational point that not everyone will have realized from a partial completion of the project (related to abstraction choices!);
- There are still some people working on it (extensions, etc.); and
- Even if you've already handed it in, I wanted to reduce worry about checkoff requirements. I think it's reasonable that some of you have asked for more concrete criteria for sufficiency, so here some are.
Note on the Queue Lock

"One array element per process" means arr.length == #Processes; NOT "every process always has the same array element".

Think of taking a number at the deli counter. The paper numbers don't go on forever. They wrap! Same thing here, with unsigned integer overflow.
Note on the Queue Lock

"One array element per process" means arr.length == #Processes; NOT "every process always has the same array element".

Think of taking a number at the deli counter. The paper numbers don't go on forever. They wrap! Same thing here, with unsigned integer overflow.

Consider what overflow looks like in a simplified setting. What could happen if:

- 3 processes (and thus arr.length == 3)
- counter from 0 to 7 that overflows back to 0? (3 bit unsigned)
Note on the Queue Lock

"One array element per process" means arr.length == #Processes; NOT "every process always has the same array element".

Think of taking a number at the deli counter. The paper numbers don't go on forever. They wrap! Same thing here, with unsigned integer overflow.

Consider what overflow looks like in a simplified setting. What could happen if:

- 3 processes (and thus arr.length == 3)
- counter from 0 to 7 that overflows back to 0? (3 bit unsigned)
- A arrives, takes 0. B arrives, takes 1. C arrives, takes 2. ...
- A arrives, takes 3. B arrives, takes 4. C arrives, takes 5. ...
- A arrives, takes 6. B arrives, takes 7. C arrives, takes 0. ...
Note on the Queue Lock

"One array element per process" means arr.length == #Processes; NOT "every process always has the same array element".

Think of taking a number at the deli counter. The paper numbers don't go on forever. They wrap! Same thing here, with unsigned integer overflow.

Consider what overflow looks like in a simplified setting. What could happen if:

- 3 processes (and thus arr.length == 3)
- counter from 0 to 7 that overflows back to 0? (3 bit unsigned)
- A arrives, takes 0. B arrives, takes 1. C arrives, takes 2. ...
- A arrives, takes 3. B arrives, takes 4. C arrives, takes 5. ...
- A arrives, takes 6. B arrives, takes 7. C arrives, takes 0. ...

Array index examined: myNumber % 3.
Partnership/Group Dynamics Survey

Posted on Piazza Please submit this!

Helps me to spot bad dynamics or behavior before final project gets busy.

I do NOT want to be blind to issues.
Partnership/Group Dynamics Survey

Posted on Piazza Please submit this!

Helps me to spot bad dynamics or behavior before final project gets busy.

I do NOT want to be blind to issues.

- Minor disagreements: your problem
- Belittling, harassment, making you feel uncomfortable: my problem too.
- Partner (or you) maybe not OK? Reach out to me.
- No such thing as "not not-OK enough". If you're worried, come see me.
Loved them. Nobody's proposal was rejected.

Feedback pretty much of 2 kinds:

- Say more (e.g., richer reach goal or unclear acceptance criteria)
- WATCH OUT!
Proposals!

Loved them. Nobody's proposal was rejected.

Feedback pretty much of 2 kinds:

- Say more (e.g., richer reach goal or unclear acceptance criteria)
- WATCH OUT!

Glorious failure is acceptable in final projects.

(Running into problems you've been warned about is not "glorious").
Presentations

20 minutes each

80% celebratory, 20% evaluatory

If you build slides, don't build many (I want to see models, see instances, etc.)
Presentations

20 minutes each

80% celebratory, 20% evaluatory

If you build slides, don't build many (I want to see models, see instances, etc.)

I will absolutely interrupt with questions. Some examples:

- What abstraction choices did you make? Why?
- Any worries about what you've left out of the model?
- What would happen if you made <this change>?

Please make a custom visualization. (New guide coming out TODAY.)
In fact, about abstraction choices

They're really important, and easy to lose sight of in the churn

Document them. Think about them. Document your worries. Not about "bugs"!
In fact, about abstraction choices

They're really important, and easy to lose sight of in the churn

Document them. Think about them. Document your worries. Not about "bugs"!

Outside context problems are basically the worst. Can:

- Come from the tool (e.g., special flags needed to check liveness)
- Come from the domain (oops, "machine code" isn't bottom layer)
In fact, about abstraction choices

They're really important, and easy to lose sight of in the churn
Document them. Think about them. Document your worries. Not about "bugs"!

Outside context problems are basically the worst. Can:
- Come from the tool (e.g., special flags needed to check liveness)
- Come from the domain (oops, "machine code" isn't bottom layer)

Degrees of trouble:
- "Oh, but there could be a solar flare!"
- "Oh, but the network might drop a packet!"
- "Wait, do you not allow the adversary to learn from messages they see?"
Something new

You've seen SMT, and this week's assignment has you use it.

Challenge problem: prove an array invariant holds. (Unbounded length arrays!)

How might we build a whole programming language around these ideas?

How do (some) people talk about verification in context of programs?