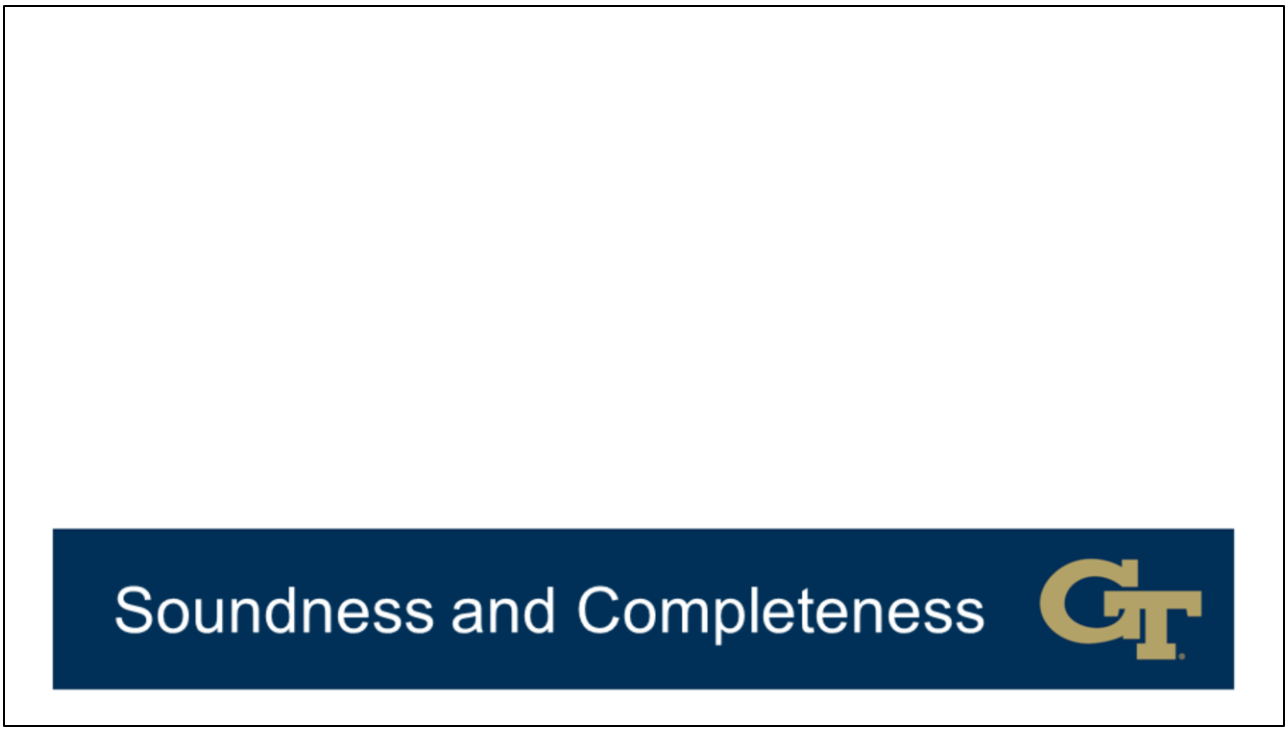
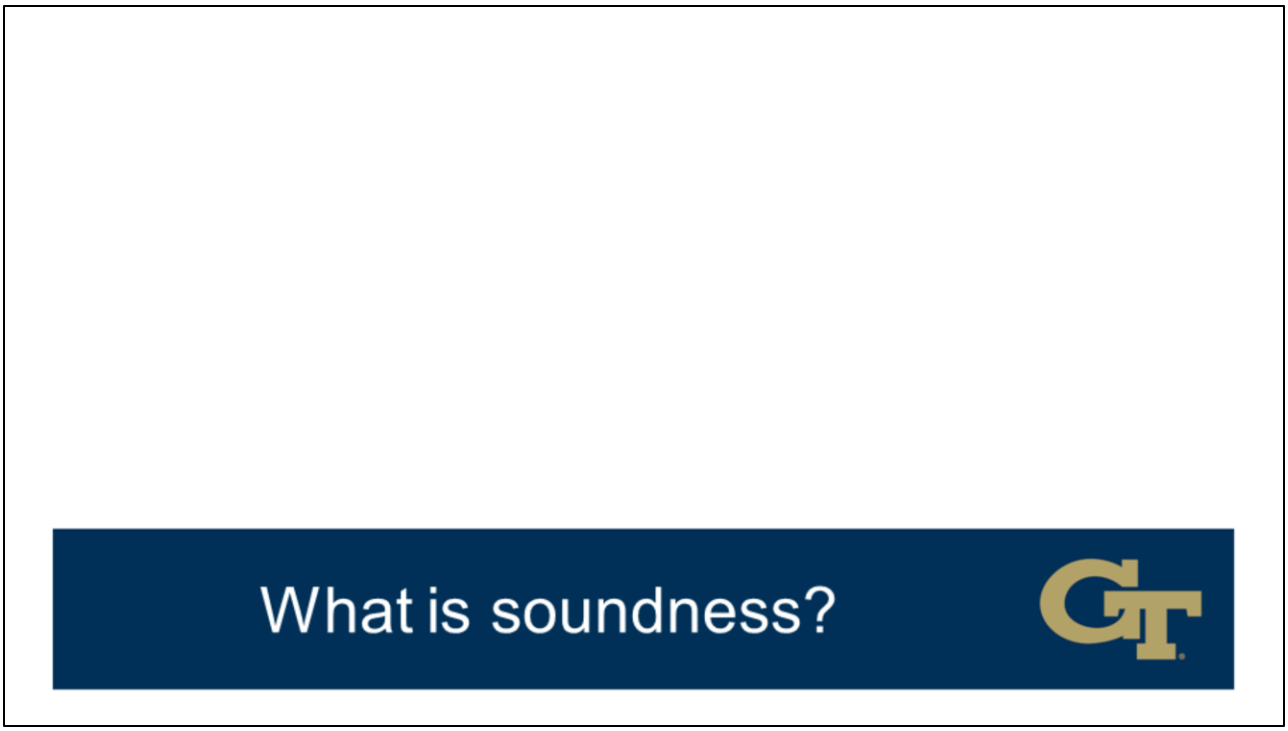
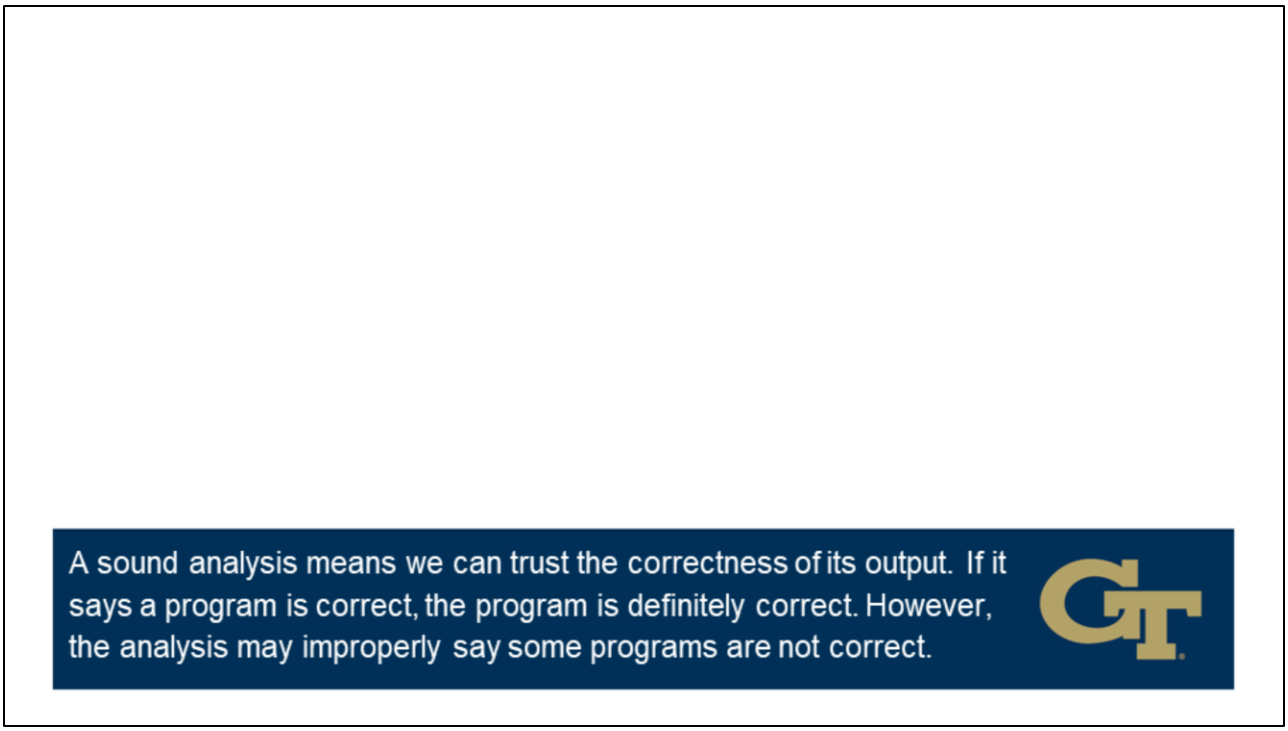


I'm Chris Poch, one of the Software Analysis instructors.

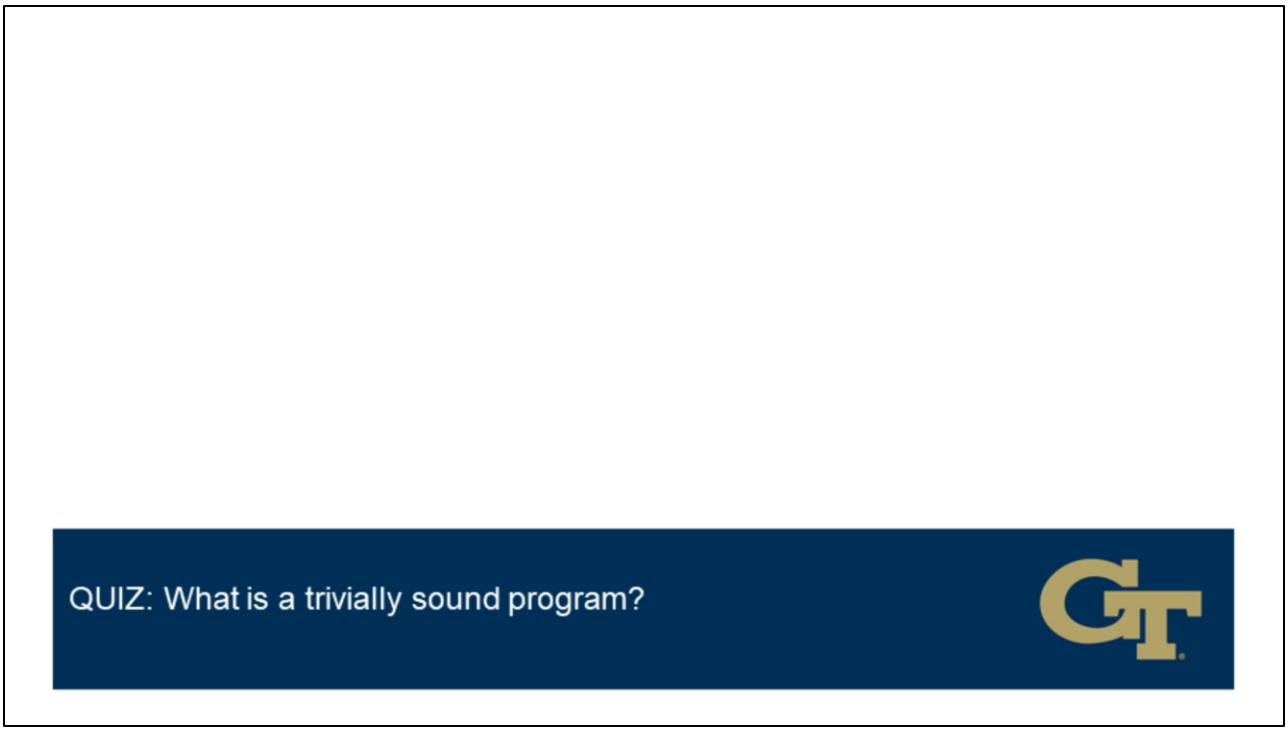
In the Introduction to Software Analysis lesson, we introduced the concept of Soundness and Completeness. These are important concepts we will return to in other lessons. I want to spend a few minutes to reinforce what we've already introduced and to provide a foundation for what's to come.



What is soundness? Let’s start with a definition.



A sound analysis means we can trust the correctness of its output. If it says a program is correct, the program is definitely correct. However, the analysis may improperly say some programs are not correct.



One useful way to think about soundness is to think about what a trivially sound analysis does. In other words, what is the simplest code possible that could be considered a sound analysis. This code is not useful; it always returns the same value for every program being analyzed.

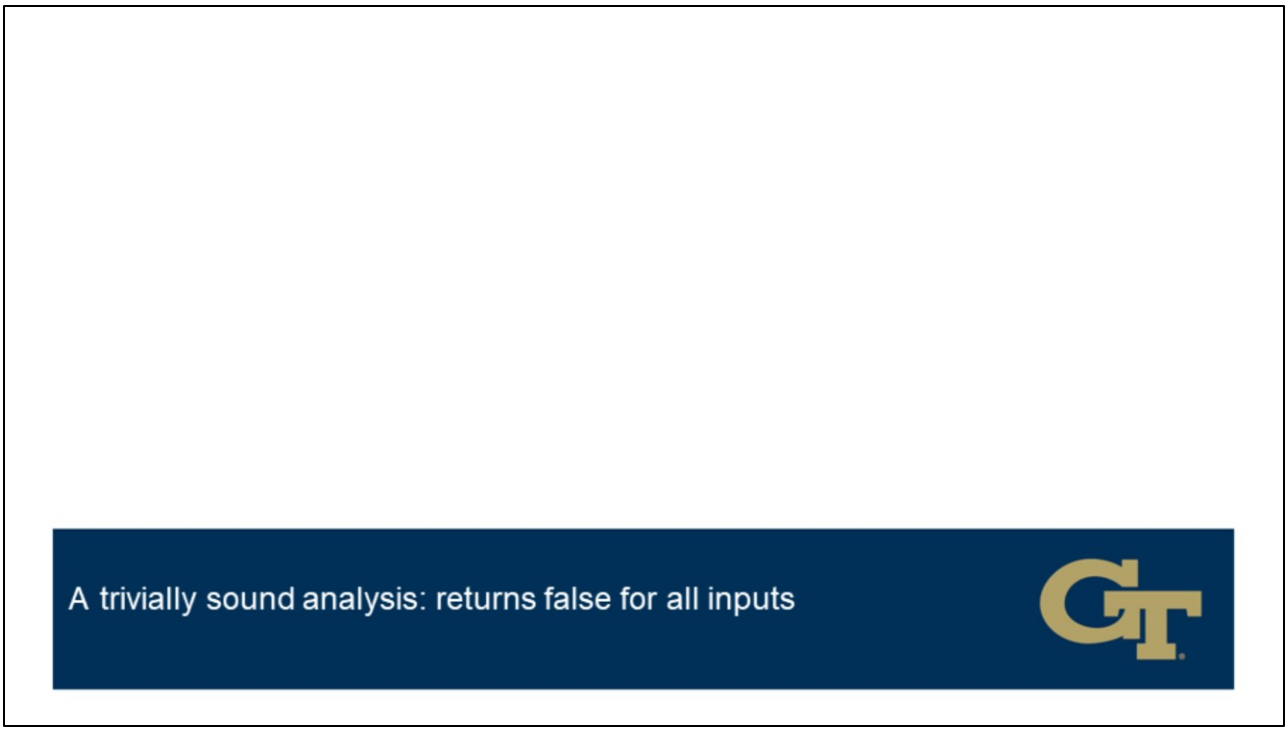
For our quiz, please complete the trivial analysis with the value that makes the analysis sound.

```
bool trivialSoundAnalysis() {
    return _______;
}
```

QUIZ: Trivially Sound Analysis

```
bool trivialSoundAnalysis() {
    return false;
}
```

A trivially sound analysis rejects all inputs because if we can’t be absolutely certain that an input we’re analyzing is correct, we should reject it. So the trivial case is rejecting all programs.



Stated more formally, a trivially sound analysis returns false for all inputs.

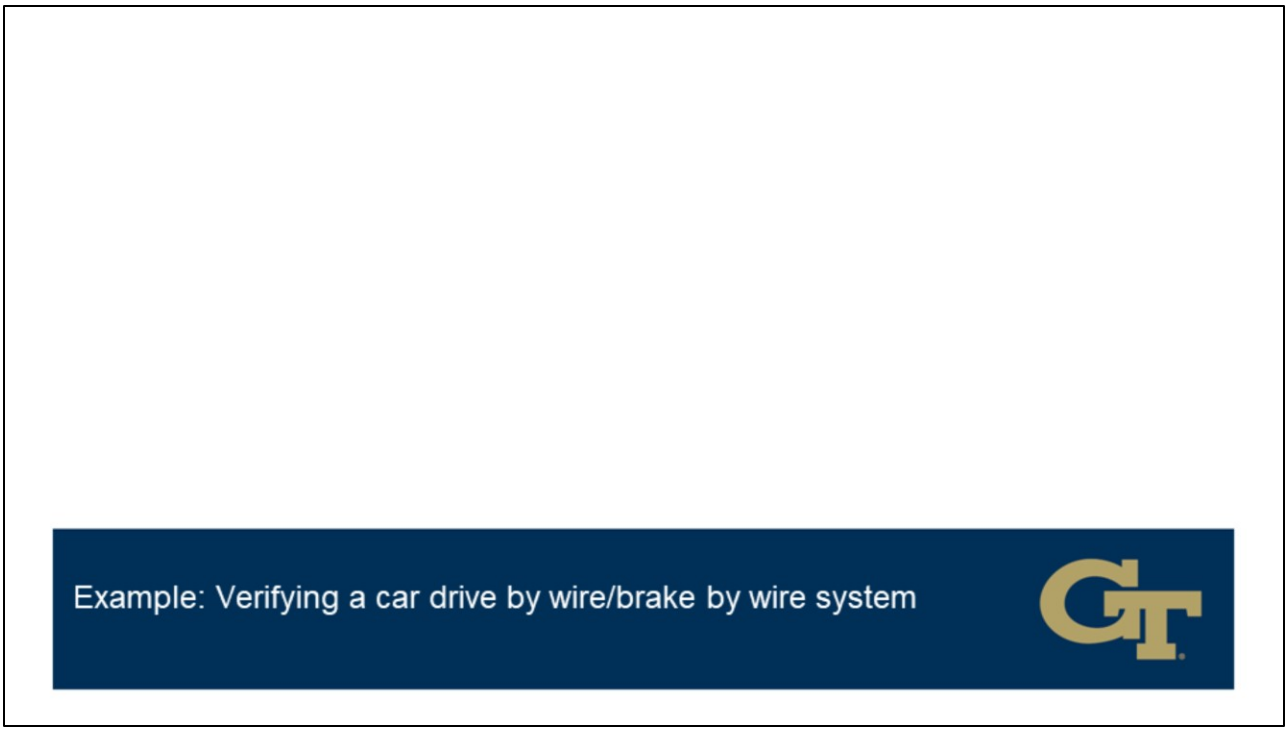
Let’s look at an example of when you would want a sound analysis.

Example: Verifying a car drive by wire/brake by wire system

Many cars these days are drive-by-wire and an increasing number are also brake-by-wire. What that means is that when you press one of the pedals, the accelerator or the brake pedal, instead of having a mechanical connection to the engine or the brakes, you're actually making an electrical connection to the car's computer, which then adjusts various actuators and solenoids that controls the brakes and the air intake on the engine. So most cars today are drive-by-wire and an increasing number are brake-by-wire, such as…

… the Toyota Prius Hybrid since its very first generation actually so over 20 years now …

…and also the Corvette C8 if the Prius isn’t really your speed.

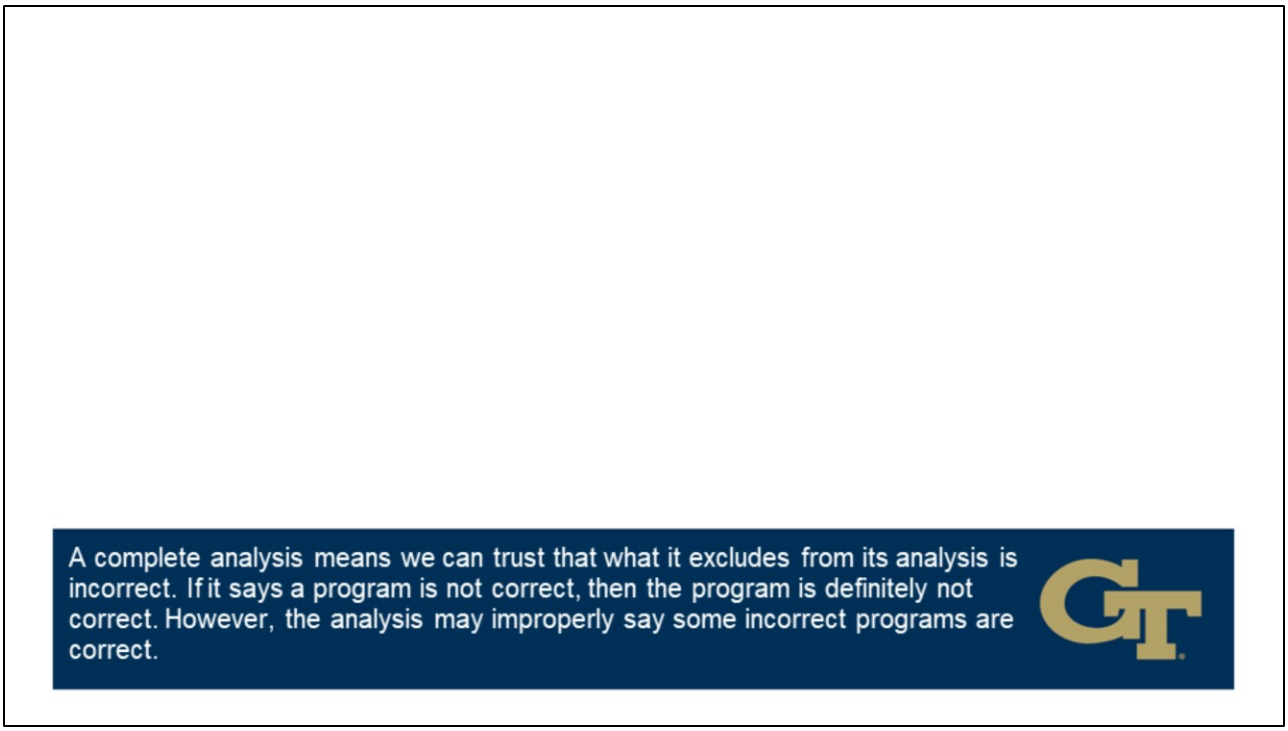


If those two cars can both be brake-by-wire and not have all kinds of problems - the Prius has been brake-by-wire for more than 20 years - if they can do that and not have massive issues with the braking system, we can clearly make a braking system that is completely electrical, software-controlled, and reliable.

It would be catastrophic if your car's controls didn't do what you told them to do, especially in an emergency situation. Can you imagine if you needed to stop quickly and you pushed the brake and nothing happened? Your car said: "Nope. Try again." Or worse, speed up. Because of these kinds of issues, we have to have absolute confidence that the program controlling the acceleration and braking in your car is correct. Therefore, we want a sound analysis.

What is completeness?

The other property we want to discuss today is completeness. What is completeness? Let's start with a definition.



A complete analysis means we can trust that what it excludes from its analysis is incorrect. If it says a program is not correct, then the program is definitely not correct. However, the analysis may improperly say some incorrect programs are correct.

QUIZ: What is a trivially complete program?

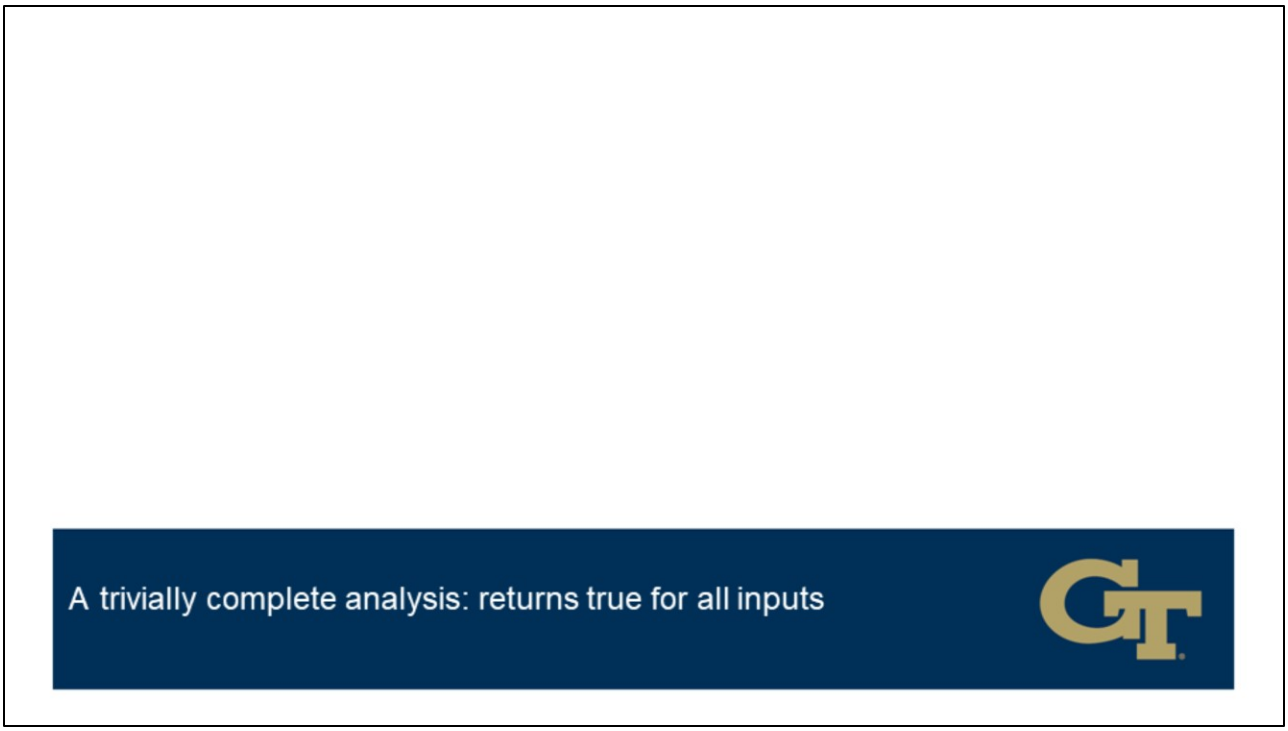
Let’s take another quiz. Just like we did for soundness, let’s think about what a trivially complete analysis does.

```
bool trivialCompleteAnalysis() {
    return _______;
}
```

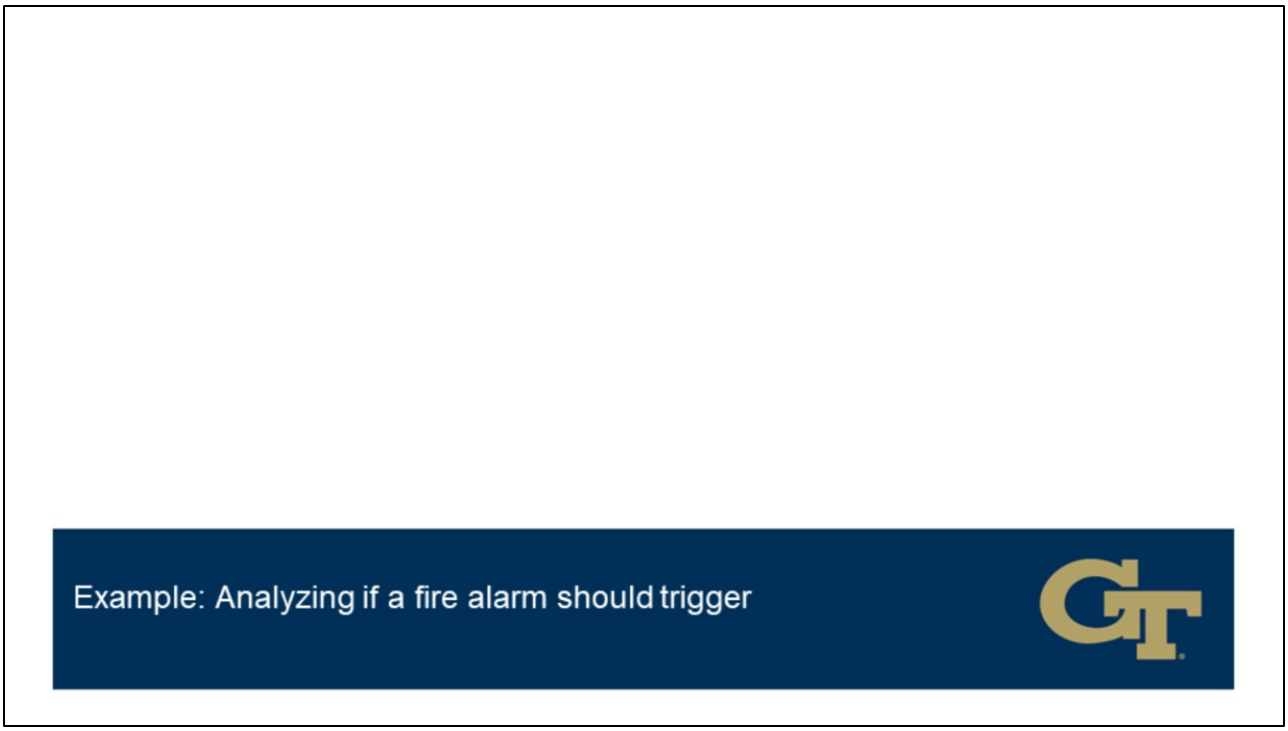
QUIZ: Trivially Complete Analysis

```
bool trivialCompleteAnalysis() {
    return true;
}
```

A trivially complete analysis accepts all inputs because if we can't be absolutely certain that an input we're analyzing has an error, we should accept it. So the trivial case is accepting all programs.

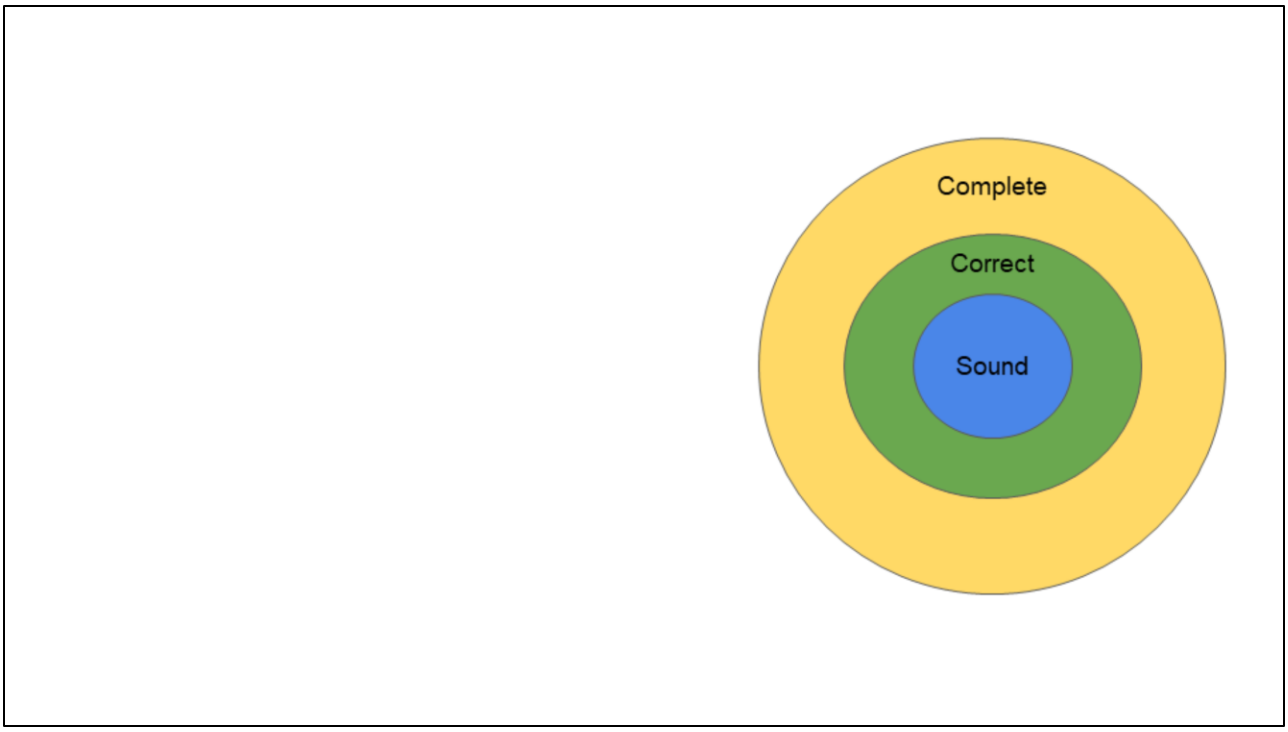


Stated more formally, a trivially complete analysis is never going to incorrectly say that an analysis is incorrect so it's just going to say every analysis could potentially be correct. Now a trivially complete analysis, just like our trivially sound analysis, is not going to be useful in practice but it's a useful mind game of how these analyses work. So when would you want a complete analysis?



An example is if you're analyzing whether a fire alarm should trigger. When you might have a fire, you want the fire alarm to go off even if the conditions are such that you cannot guarantee something is in flames. It's much better to sound the alarm and be wrong than to fail to sound the alarm and have a problem and maybe have people injured or worse.

Let me show you a diagram of how soundness and completeness fit together.



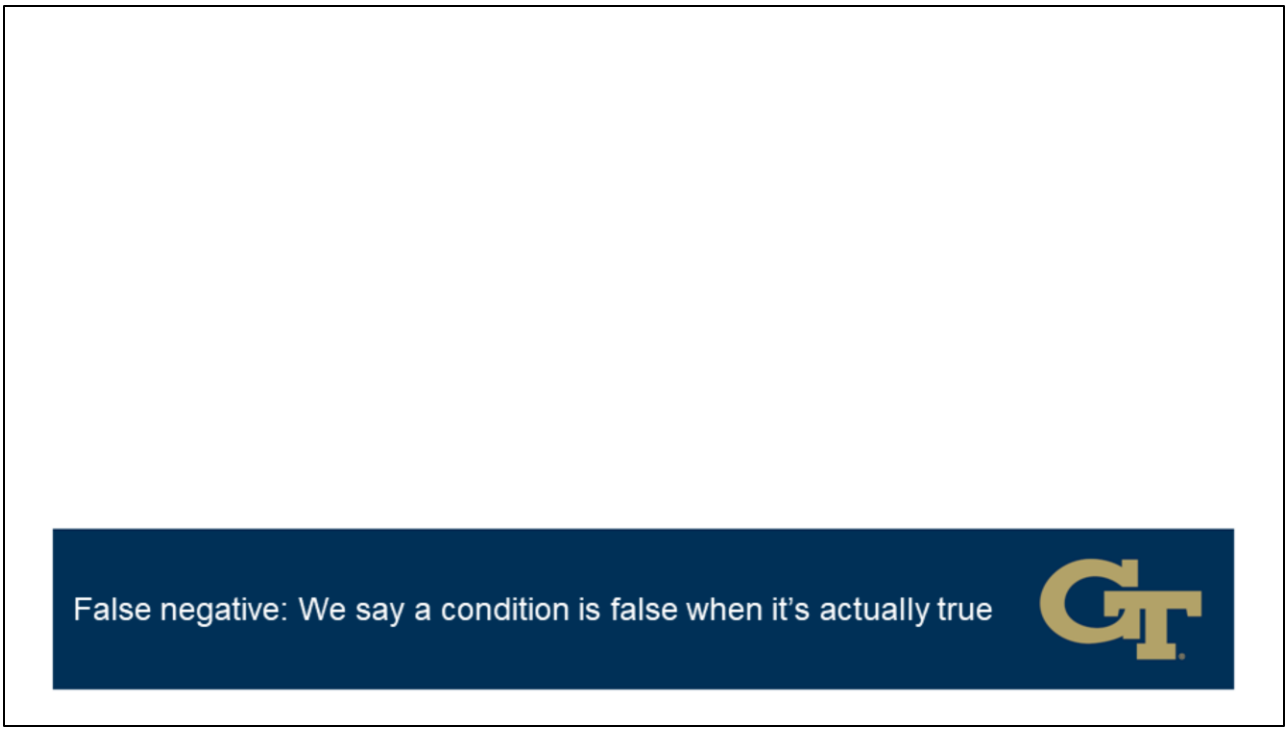
If the space of all programs is the entire screen and we're analyzing for correct programs, we can say that there are more complete analyses, than correct programs, than sound analyses.

All sound analyses are correct, and all correct programs are complete, but we can't flip it around and make the reverse of that true.

There's something that we discussed in the previous lesson that you'll notice I haven't mentioned yet: false positives and false negatives. Why haven't I discussed that? Well first, let me define our terms.

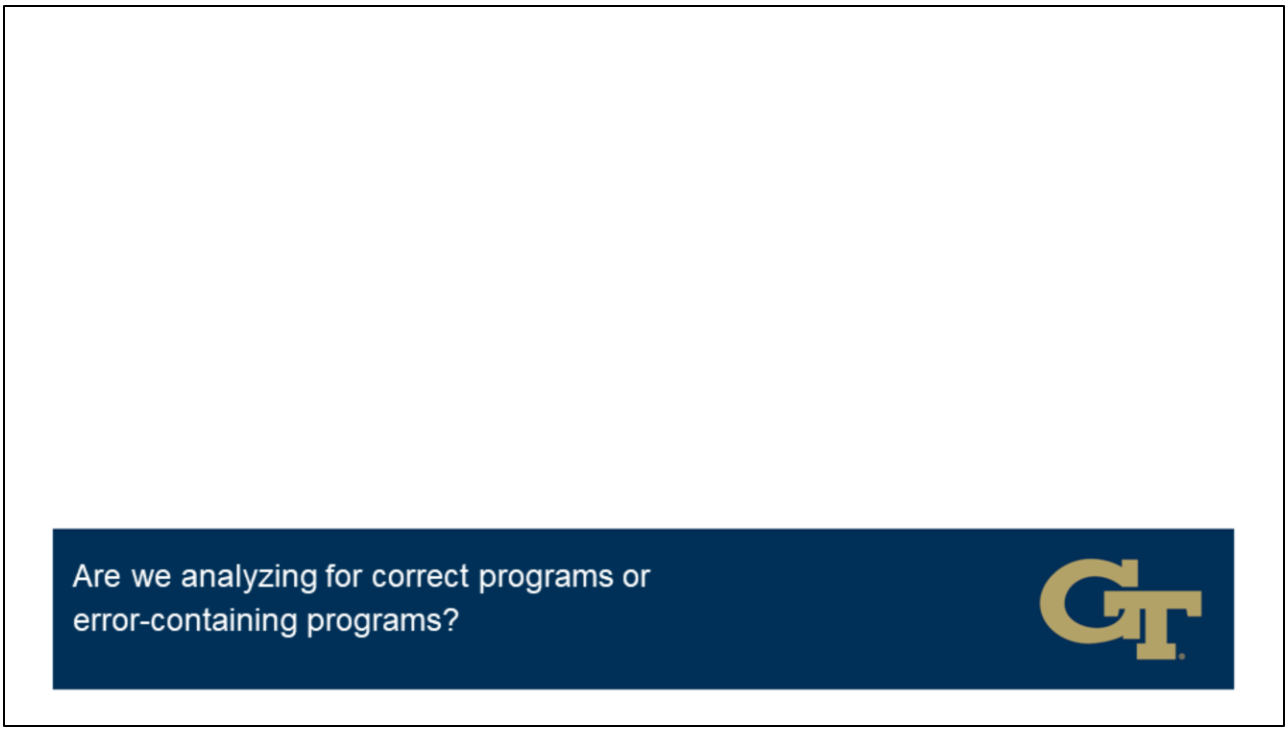
False positive: We say a condition is true when it's actually false

A false positive is when we say a condition is true when it's actually false. So an example is when I sound the fire alarm and nothing is wrong.



On the other hand, a false negative is when we say a condition is false when it's actually true. So an example of this is if I say the plane can't take off because we're missing passengers and it turns out the ticket agent miscounted and everyone is already on board.

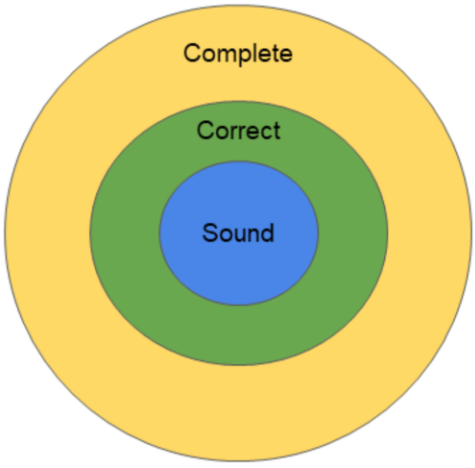
So what's the confusion here? All of it seems pretty straightforward so far. Well what's confusing is perspective. Ok, you say "how is that confusing?" Well soundness and completeness are relative terms; they depend on what we're looking for.



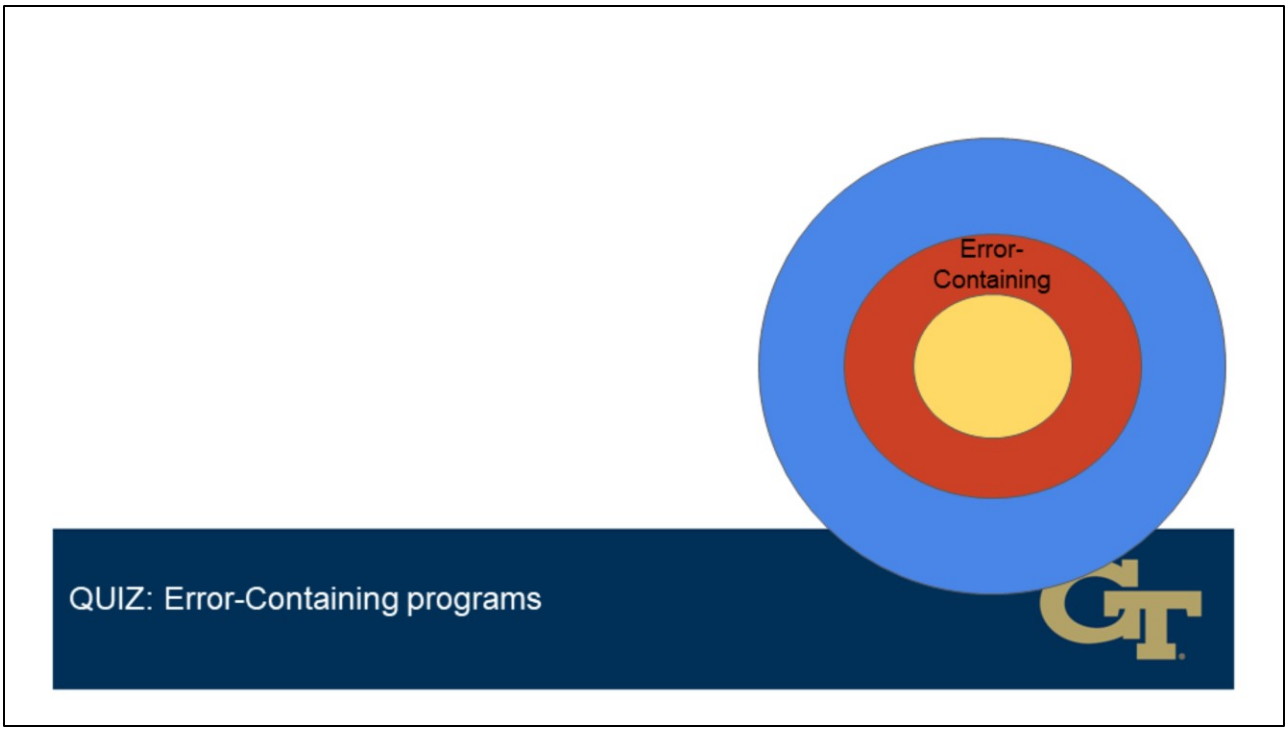
Are we analyzing for correct programs or are we looking for error-containing programs? So to explain this a little better, let me give an example using another relative concept: Left and right.

I'm holding up a Georgia Tech flag here. It's to my right side, but that's only because you're looking at my front side. If you're standing behind me, it's not on my right side, actually it's to my left side. Now nothing about me or the flag changed. All that changed here is your point of reference.

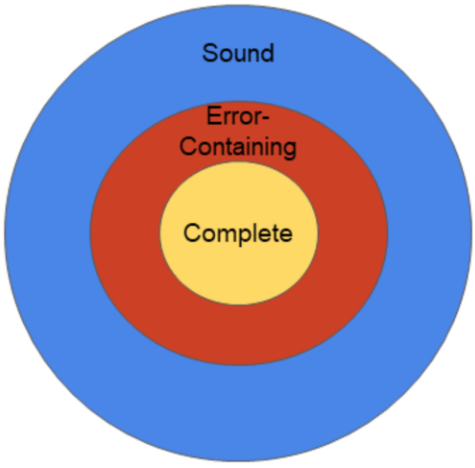
So what does that mean for soundness and completeness? If we're using relative terms like false positive or false negative, we need to be certain that we know what we're looking for. Typically, you're going to be looking for either error-containing programs or correct programs.



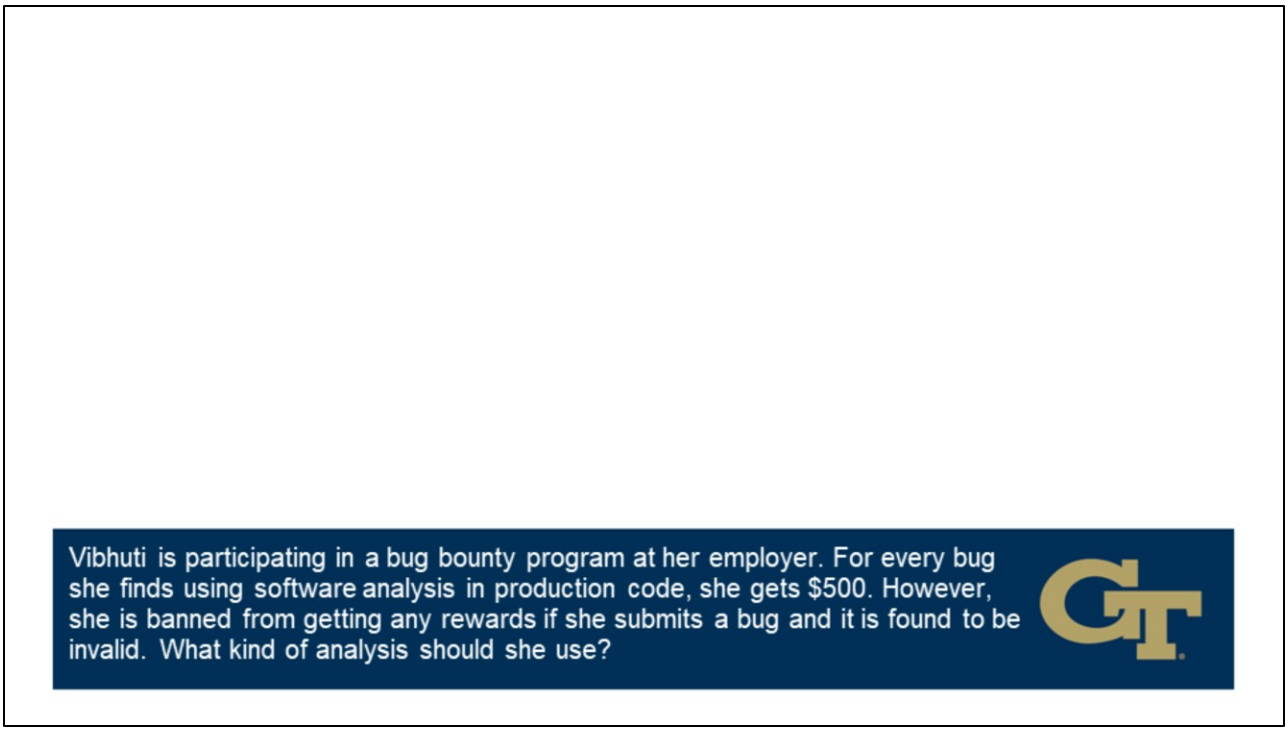
If we go back to the diagram from earlier, you'll notice what's in the middle ring: correct programs. Ok, but what if we instead are looking at error-containing programs?



Let’s make it a quiz. Where do Soundness and Completeness fall on our diagram when we switch out Correct Programs for Error-Containing programs?

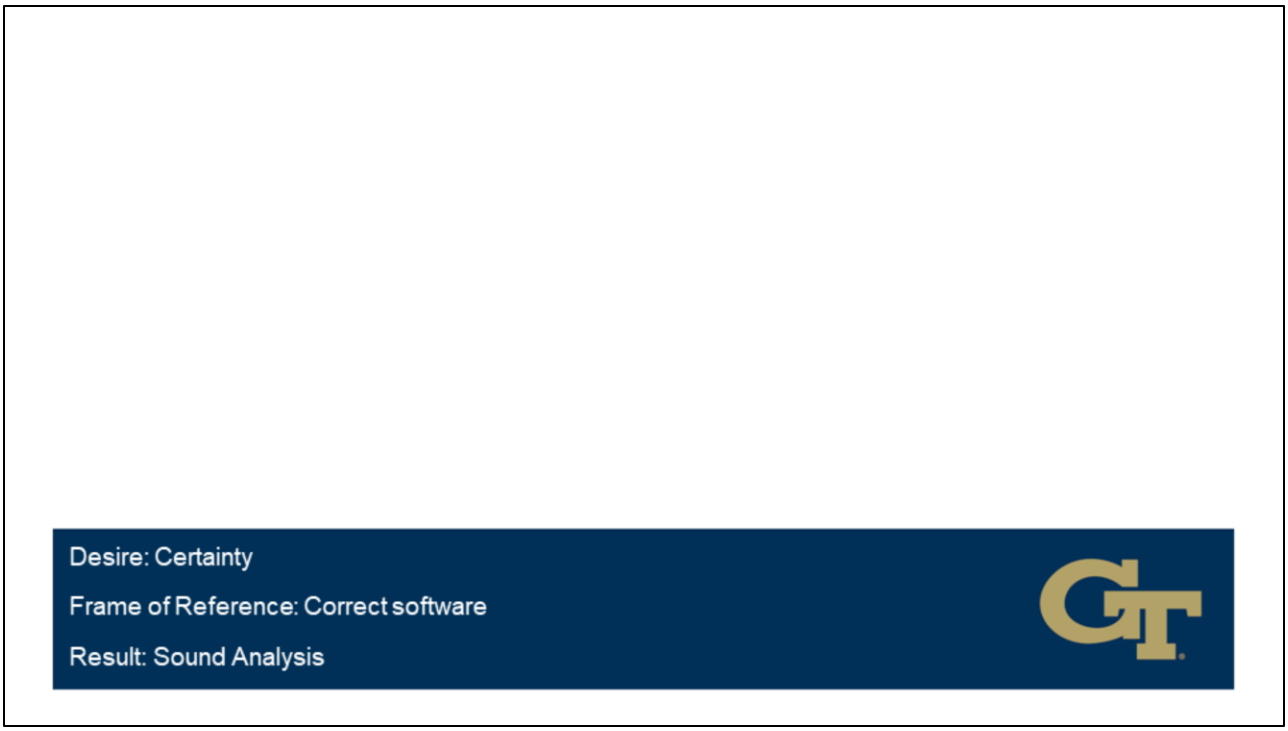


The correct answer is that soundness and completeness flip locations. If we're looking for error-containing programs, out of the set of all possible analyses, there are more sound analyses than there are error-containing programs than there are complete analyses- but the reverse is not true! So you see that when we flip our perspective, we flip where sound and complete are just by what we're looking for.

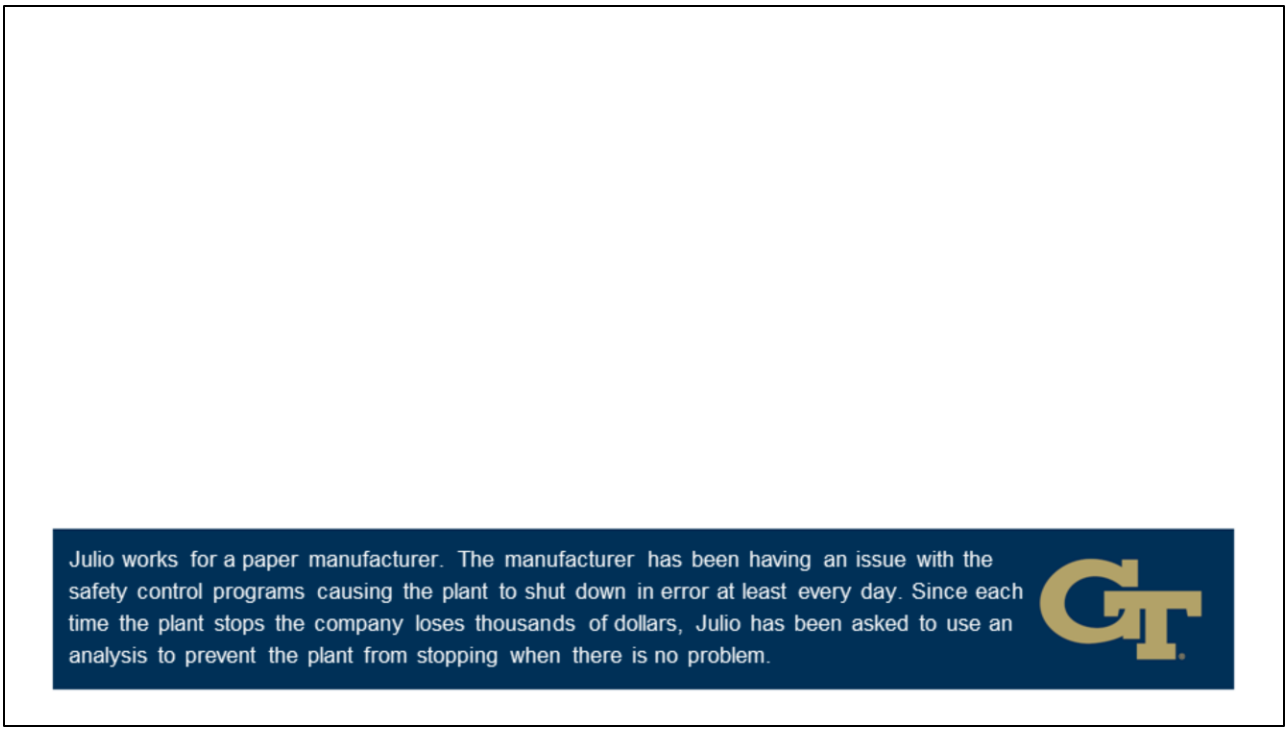


Let’s work through some examples.

Vibhuti is participating in a bug bounty program at her employer. For every bug she finds using software analysis in production code, she gets $500. However, she is banned from getting any rewards if she submits a bug and it is found to be invalid. What kind of analysis should she use?

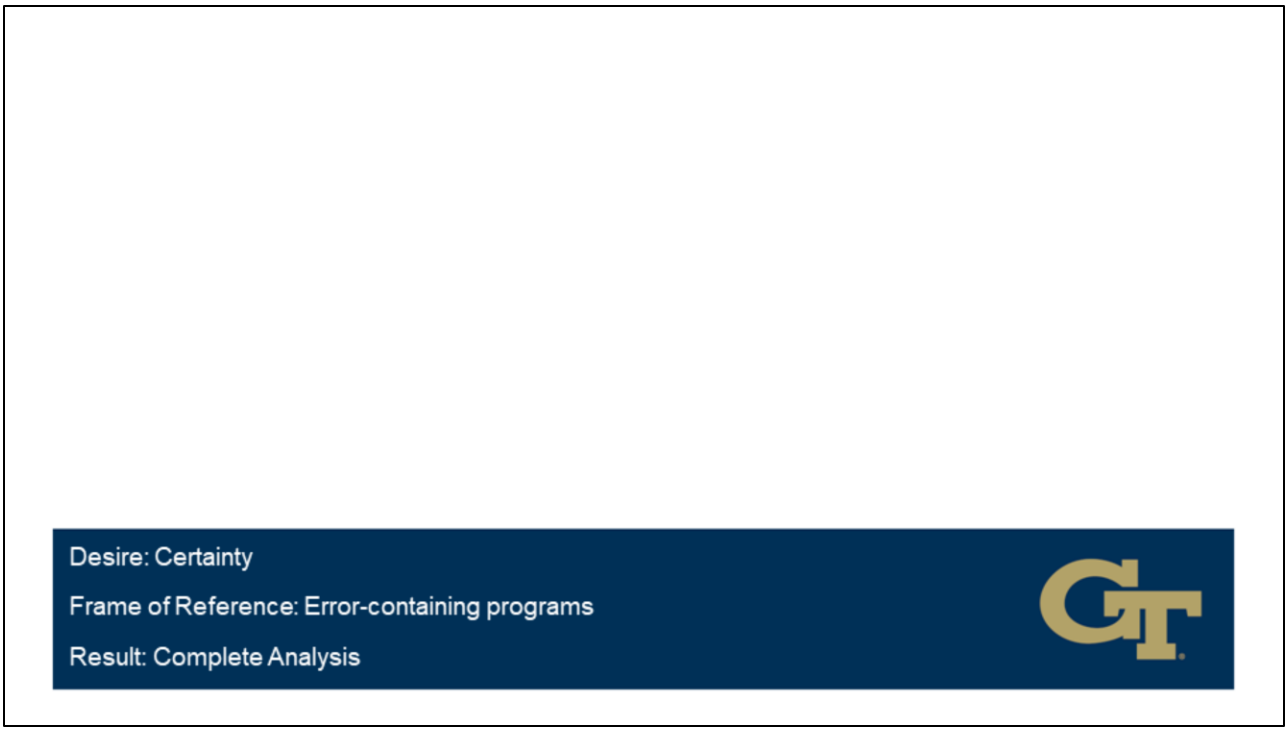


What is most important here? [click] Certainty. Vibhuti gets kicked out if she submits any bugs that turn out to be working correctly. Our point of reference is correct software. [click] When we're looking for correct programs and want absolute certainty about our results, we want a sound analysis [click]. Any analysis that is not sound risks submitting an invalid bug. It's better to miss out on some potential bug bounties than to be kicked out of the program!



Let's look at another example.

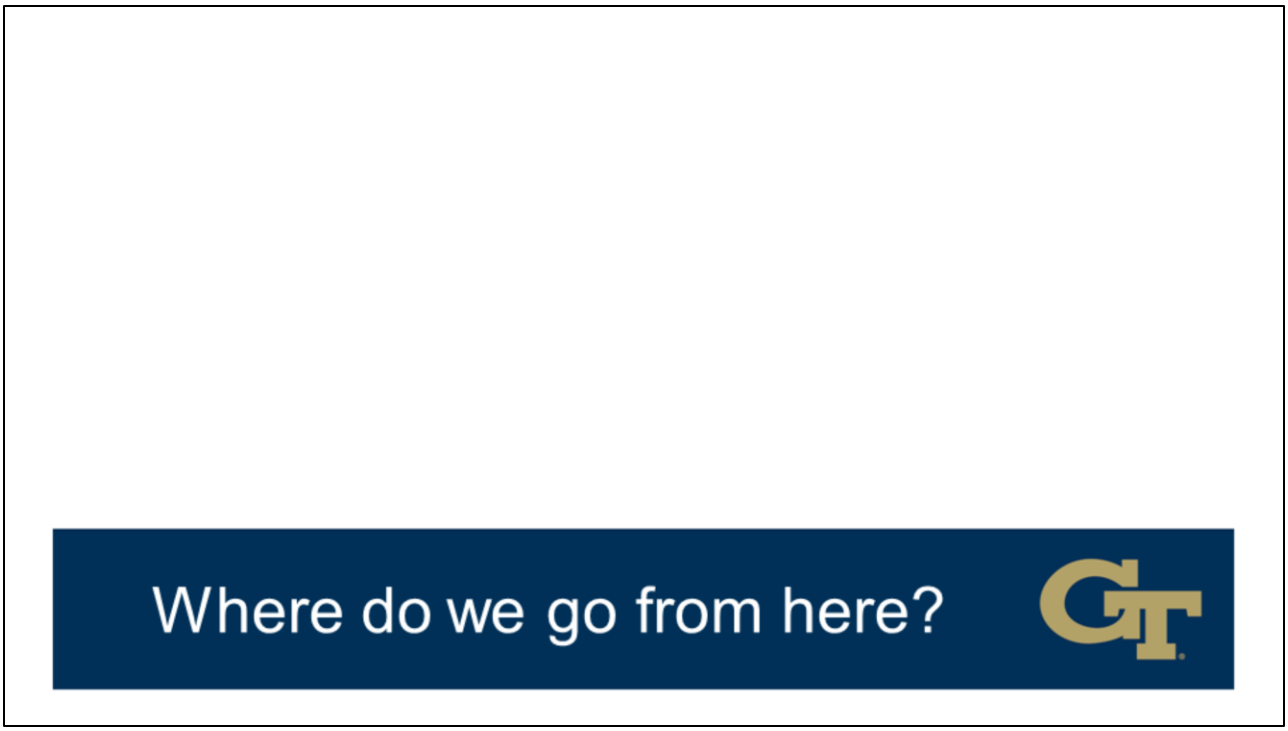
Julio works for a paper manufacturer. The manufacturer has been having an issue with the safety control programs causing the plant to shut down in error at least every day. Since each time the plant stops the company loses thousands of dollars, Julio has been asked to use an analysis to prevent the plant from stopping when there is no problem.



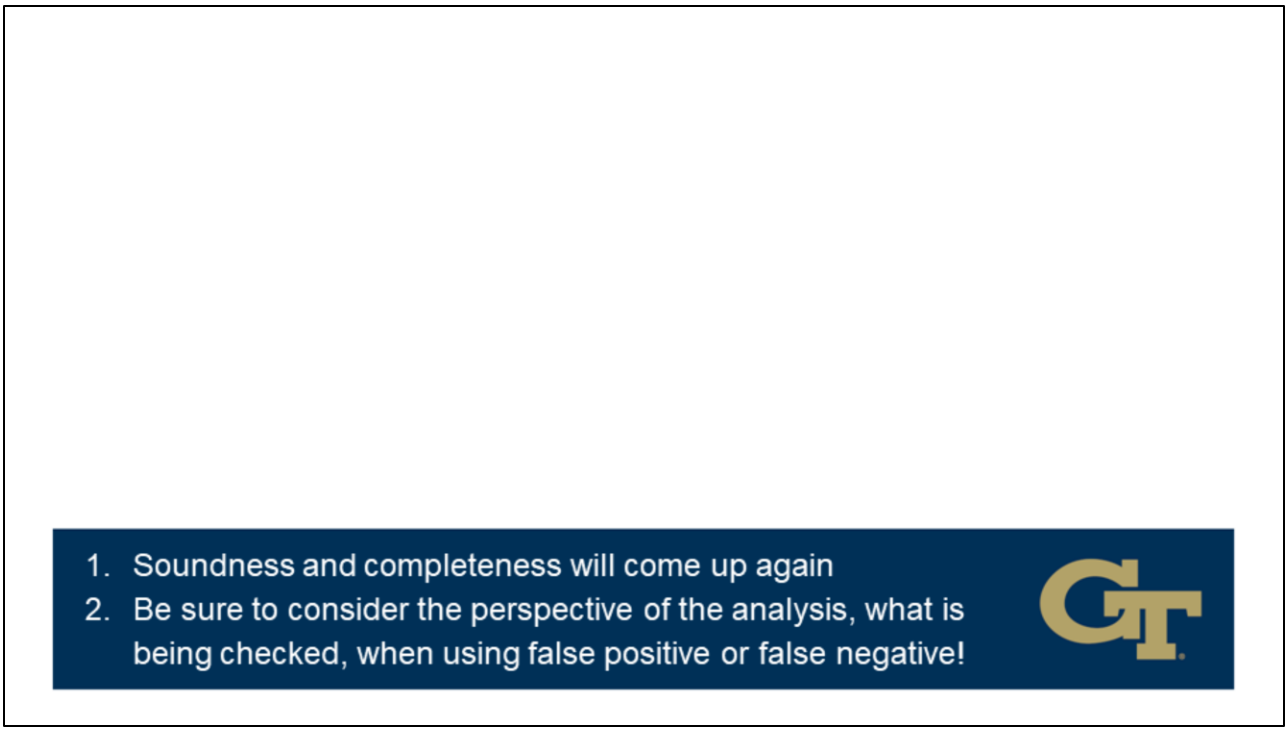
What is most important here? [click] Certainty, that is all errors are definitely errors. Our point of reference is error-containing programs. [click] When we're looking for error-containing programs and want absolute certainty about our results, we want a complete analysis [click]. Any analysis that is not complete risks shutting down the system when nothing is wrong. For a factory plagued with false alarms, it's better for some real problems to be potentially allowed than for the analysis to continue to shut down the machines when frequently nothing has been wrong.

Did you notice here that the set up for our examples were very similar? Both were looking for certainty, but since they had different frames of reference, the results were different: the first required a sound analysis and the second required a complete analysis.

One other thing you'll notice here is that when dealing with trade-offs, there is a certain level of opinion involved. What level of error is worth shutting down the factory over? The answer depends on your risk tolerance and the consequences of either allowing a problem to go unreported or rejecting something when nothing is wrong. That's why in these problems we have to indicate what is the most important objective so you can pick the correct analysis for that situation.



Where do we go from here? There are two takeaways we want you to have from this lesson.



First, soundness and completeness will come up again. These are critical concepts when we are talking about software analysis so if you are uncertain about these, you are going to want to do some more reading. We have a lot of good resources that we can point you to, both in the books we recommend and also a number of great websites and articles published on the web that discuss soundness and completeness in more depth. But you want to make sure you have this locked down because you're going to be confused if you don't have these concepts locked down.

You're also going to want to be sure you consider the perspective of the analysis, in other words, what is being checked, especially if you use terms false positive and false negative. I personally tend to avoid those two terms today, but many of the lesson videos include false positive and false negative terminology so you want to make sure that you understand the perspective because otherwise, you're going to think that the lesson is wrong and has things backwards just because you're perspective is different - just like the flag was on my right or my left depending on what side you were looking from.

So hopefully soundness and completeness make more sense to you in the context of our lessons now.