505 Overview

Welcome to UW CSE 505! This document provides a course overview answering "the five W's".

Who

Every course is a *collaboration* between staff and students to map out some intellectual territory.

Staff

The staff are like "trail guides" who have already spent some time exploring the ideas in the course and have mapped out a path to help us survey as much territory in as much detail as time allows.

This quarter we are very lucky to have two instructors and 3 teaching assistants:

- James Wilcox
- Zachary Tatlock
- Oliver Flatt
- Kevin Mu

Students

The students are adventurers who are eager to investigate ideas in the course. They have prepared extensively for the journey, often with decades of prior education and engineering practice, and are ready for a challenge.

This quarter we are especially lucky to have students from several groups across the Allen School:

- Professional Masters from the PMP Program
- PhD students from the Doctoral Program
- BSMS students from the Combined Bachelor/Masters Program
- BS students from the Undergraduate Program

We're super excited for this opportunity to share many diverse perspectives on the topics in 505 and build collaborations! The staff have been working hard to make 505 a course that has something for everyone 😃

What

505 is about the foundations of Programming Languages (PLs). Focusing on the foundations means that we will learn fundamental techniques that can be applied broadly in systems we already care about today, as well as many more systems that folks will care about in the future.

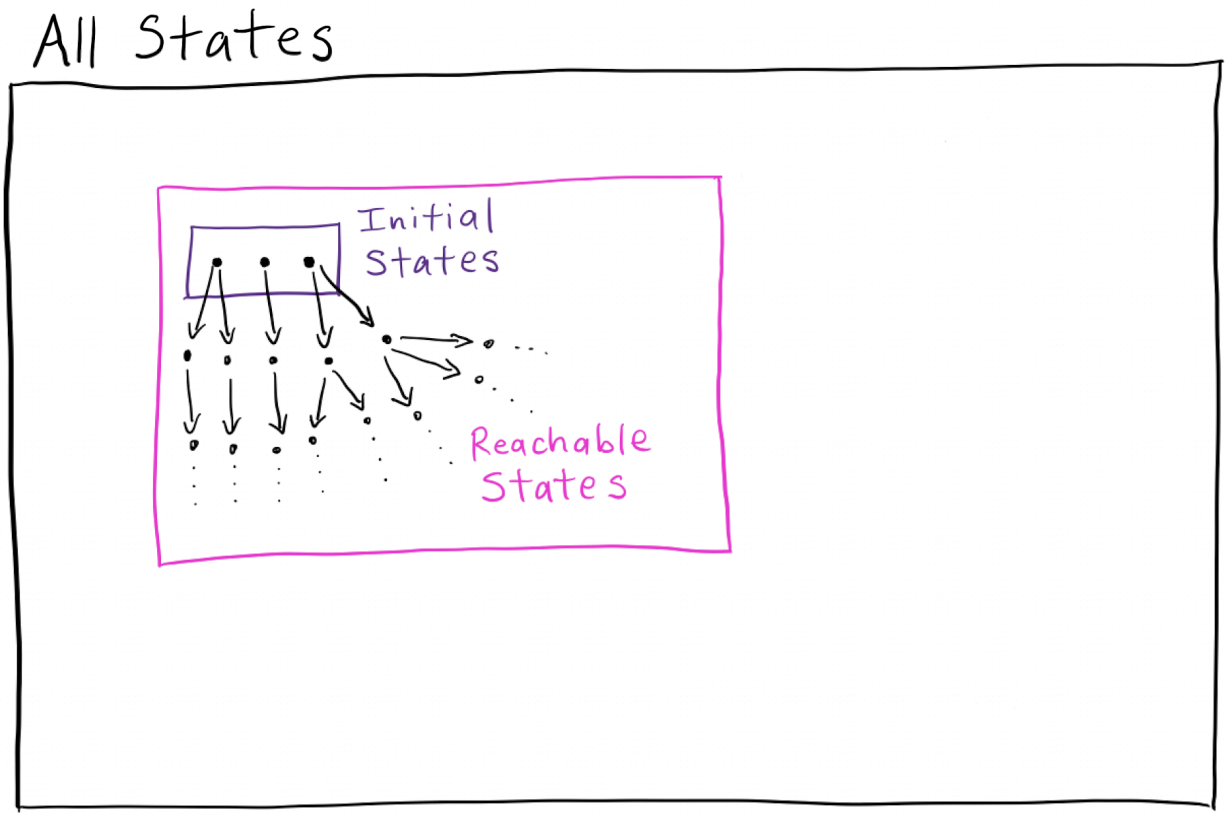
However, 10 weeks is a short time for exploring an entire field! To help make everything fit in the quarter, we will focus on models of PLs and programs. Because they are models, our example PLs and programs will often be simple "toys" designed to highlight key underlying principles. Below we sketch a handful of the most important principles we'll focus on in 505.

Programs are Transition Systems

Conceptually, most programs are just transition systems. A transition system consists of a set of states **`S`** and transition relation, often called "→", on states from **`S`**. For states **`s1`** and **`s2`**, we say "**`s1`** can step to **`s2`**" if **`s1`** → **`s2`** holds. This encodes the idea that if our program is in state **`s1`**, then in one step it can reach ("transition to") state **`s2`**. Modeling programs as transition systems provides a generic approach for reasoning about their behavior and enables us to develop reusable proof techniques.

Obviously, engineers rarely specify their programs directly in terms of transition systems. Instead, they use the syntax of their programming language. But the syntax of a language only tells us which strings are "grammatically correct" programs; syntax doesn't tell us what programs **mean**. For that we have semantics. In particular we will focus on operational semantics which map a program's syntax into a transition system!

Once we have modeled a program as a transition system, we can describe its behavior in terms of the set of states it can reach after starting from some initial state.



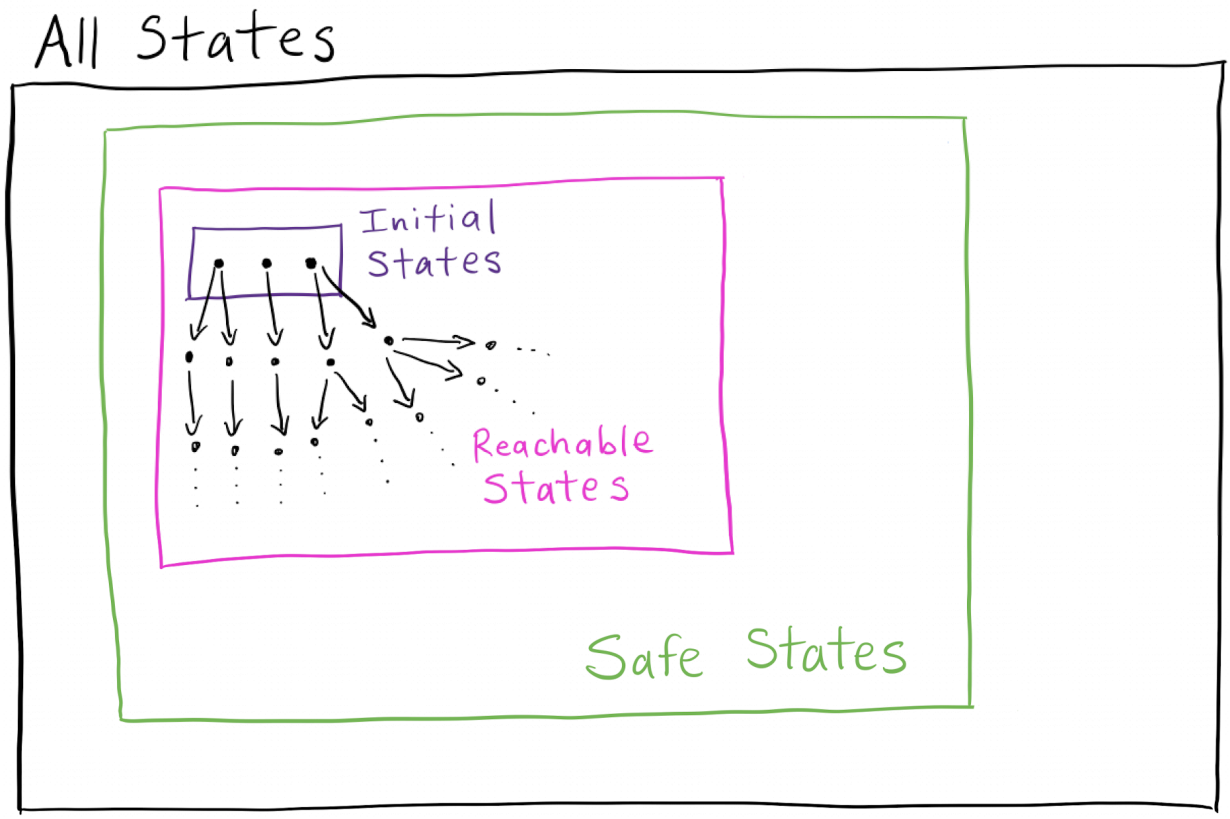
Given a notion of initial states and a step relation, we can define a program's reachable states inductively:

State s is Reachable if either:

- InitialState(s), read "s is an initial state" or
- There exists another state r such that both:
  - Reachable(r), read "r is reachable" and
  - $r \to s$, read "r can step to s"

Invariants and Induction

We will specify the correctness of programs in terms of the states that are reachable in their corresponding transition systems. Typically, we'll specify what it means for a state to be "safe" and then try to show that all of a program's reachable states are safe states:



In other words, we want to prove that “if a state is reachable, then it is safe”:

$$\forall\ s, \text{Reachable}(s) \Rightarrow \text{Safe}(s)$$

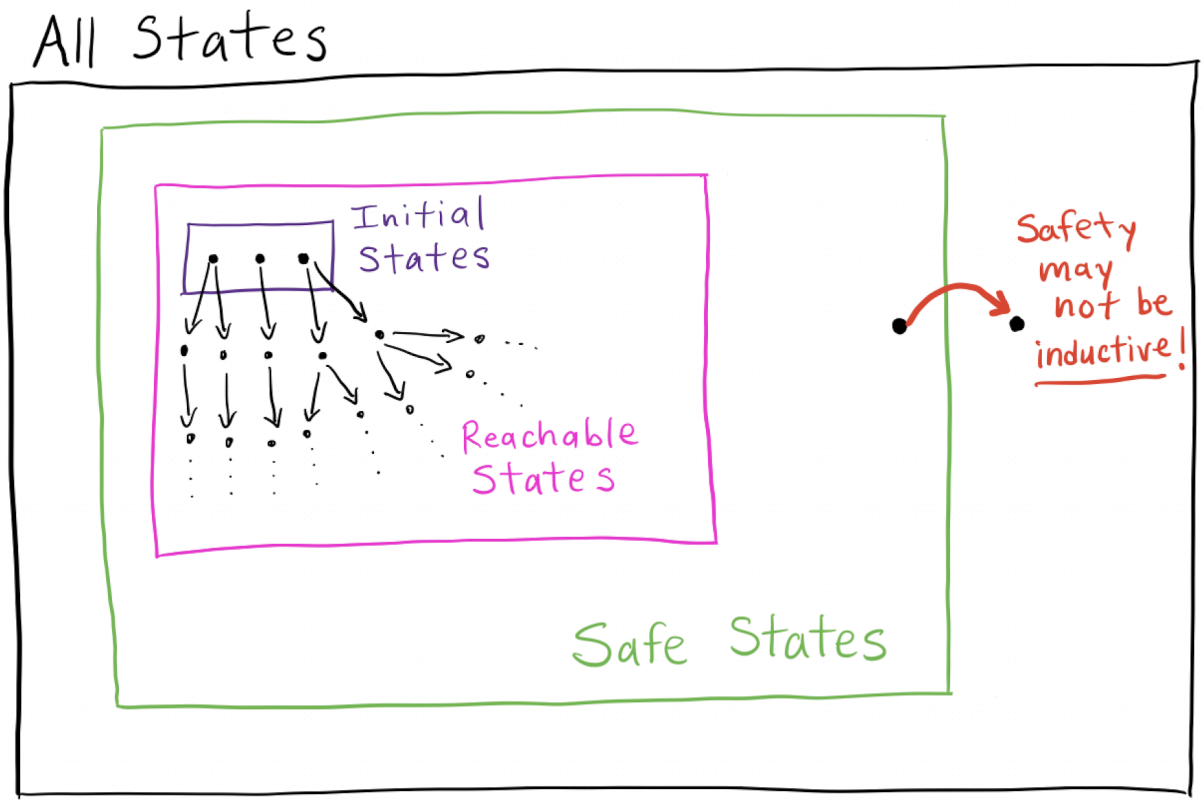
Because many interesting programs have infinitely many states, we’ll need to use induction. This breaks our proofs down into two smaller parts:

Base Case: $\forall\ s, \text{InitialState}(s) \Rightarrow \text{Safe}(s)$

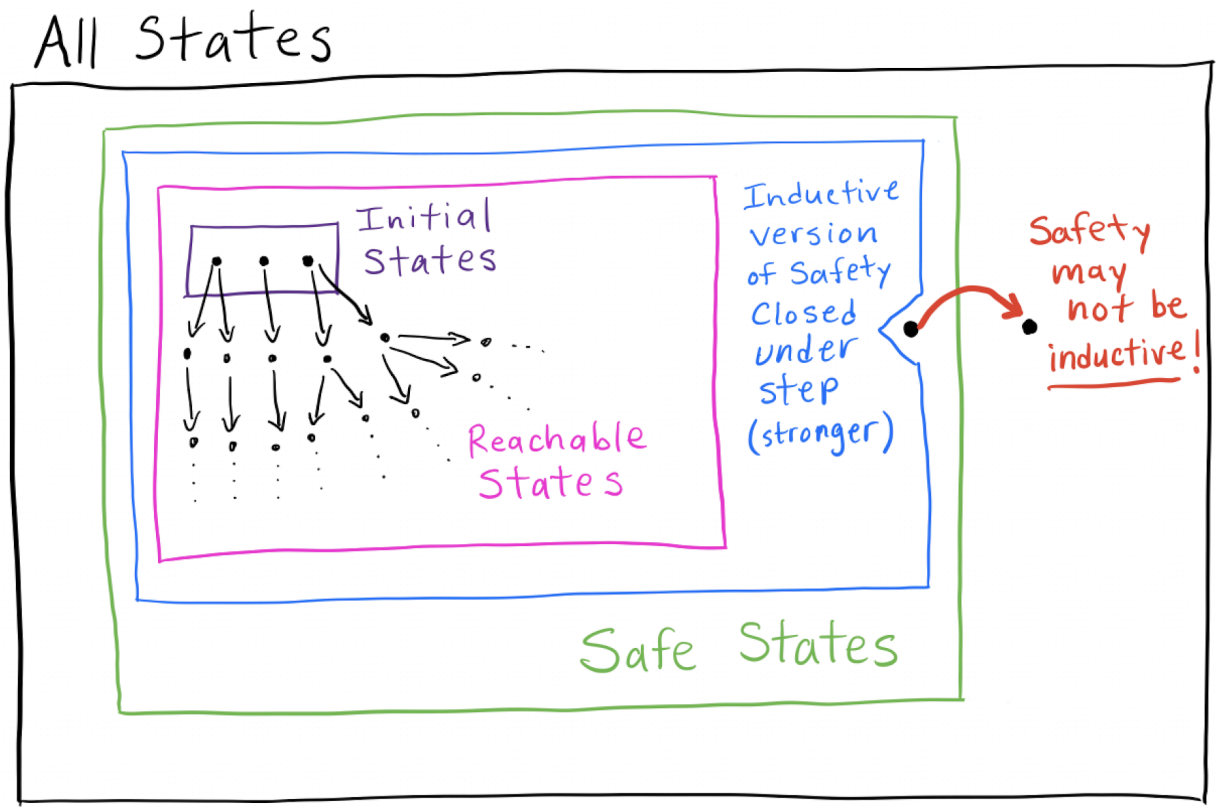
Inductive Case: $\forall\ s1\ s2, \text{Safe}(s1) \wedge (s1 \rightarrow s2) \Rightarrow \text{Safe}(s2)$

Using the inductive definition of Reachable, we can show these two parts are sufficient to prove that all states our program may end up visiting are in fact Safe.

The key challenge in verification is that our notion of “safe states” may not be inductive! That is, there may be safe states which can step to unsafe states. That doesn’t necessarily mean our program is incorrect: if safe states which step to unsafe states are not reachable, then our program may still be correct, but our notion of “safe” may not be provable by induction:



To handle this, we will use the same design pattern over and over: making a property inductive so that it “implies itself” in the inductive case. We need to define an alternate, “inductive version of safety” that both (1) implies our target notion of safety and (2) is closed under the step relation:



In practice, finding such an "inductive version of safety" can be extremely difficult. This is because our property must "imply itself" in the inductive case: just given the fact that it holds on a state **s** must be enough to imply that it holds on all states **s** can step to. This leads to a subtle tradeoff:

- If we make our property "stronger" (true for fewer states, i.e., a smaller region in the Venn diagram), then we "**know more**" from our induction hypothesis, but also have to "**show more**" to establish our goal.
    - If we make our property "way too strong" (too small), it may no longer even be true for all reachable states!
- Dually, if we make our property "weaker" (true for more states, i.e., a bigger region in the Venn diagram), then we "**know less**" from our induction hypothesis, but also have to "**show less**" to establish our goal.
    - If we make our property "way too weak" (too big), it may no longer imply our originally specified notion of safety!

Rather than requiring novel insight for every program we want to verify, we'll explore general patterns that provide "once and for all" recipes to prove that systems for proving properties of programs are correct. This approach of "proving things about programs that prove things" is known as metatheory and is one of the most fascinating parts of 505.

Mechanized Formal Verification and "Pen and Paper" Proofs

As we study all these topics in 505, we'll be using a special kind of laboratory called a proof assistant, in particular the highly influential Rocq proof assistant. In parallel, we will also often write more traditional "pen and paper" proofs using English and standard mathematical notation. Being able to work in either mode will help provide a deeper understanding of both.

Rocq allows us to construct *machine-checkable proofs*. This helps mitigate a thorny bootstrapping problem: if we think programs are so complex and subtle that we need to prove them correct formally, then why should we believe that our proofs about them are any more correct? (*Quis custodiet ipsos custodes?*)

To address this issue, Rocq strives to satisfy the de Bruijn Criterion: Rocq includes a small, heavily tested proof checker and so, as long as the proof checker is correct, we can be confident that any proofs we develop in Rocq will also be correct. Such leverage is kind of magical: by getting just one small piece of code right (the proof checker), we can confidently build an unbounded amount of verified infrastructure (our systems and their accompanying proofs)!

More importantly for 505, using a proof assistant like Rocq helps us state theorems very precisely and think very carefully about why they are true. At first, it can feel unnatural to approach proofs so mechanically, but the intuition we gain by working through each step in detail will help us tackle progressively larger and more complex problems.

Rocq also demonstrates a beautiful idea known as the Curry-Howard Correspondence: *types are theorems and programs are proofs*! We'll see this perspective in action throughout the course, but it primarily helps us understand how to formalize and reason about software systems in terms of inductive types and functions.

Finally, a solid chunk of modern programming languages research is carried out in proof assistants or uses terminology and notation that we can carefully study using proof assistants. Thus, using Rocq helps with another goal of 505: making it easier to continue studying programming languages after the course by demystifying jargon and dense notation.

Despite all these benefits, Rocq is primarily a tool for attaining extremely high confidence in proofs based on formal models of systems. In practice, we need to be able to roughly model systems that haven't been completely formalized and to communicate the high-level intuition behind proofs to our colleagues and customers. Thus, we will also practice writing proofs in a more traditional style and translating between mechanized and traditional proofs.

The Big Picture

Like all academic and engineering disciplines, the modern study of programming languages and verification does not take place in a vacuum. There is a rich history of ideas with both skeptics

and proponents of formal techniques arguing vigorously. There are also many practical concerns and non-technical stakeholders that we should keep in mind while exploring the potential applications of rigorous verification.

To help develop such a "big picture" view of programming languages and verification, we'll read, reflect on, and discuss a handful of papers together throughout the course. These discussions will explore the potential needs for and/or implications of rigorous PL and formal verification techniques.

Why

What's the point of taking a graduate-level course on programming languages? While it is unlikely that many course participants will use Rocq in their day-to-day work in the near future, we think there are a couple of key benefits from taking 505:

- We gain access to new perspectives that can help us design better systems.
    - Thinking in terms of transition systems, invariants, and induction makes it easier to design, document, and test any applications we may work on in the future.
    - In fact, just the exercise of formally specifying the correctness of a system in detail often provides many of the benefits of full formal verification! However, we need to understand verification in order to write such precise specifications.
- We acquire a deeper understanding of ideas that were already familiar.
    - As the saying goes, "travel to understand where you're from".
    - By seeing other, very different, ways of building systems and reasoning about their behavior, we can better appreciate the tradeoffs in tools that we've been using long after they've become second nature.
- We get the opportunity to learn and study some beautiful ideas.
    - Life is short! Taking time to appreciate deep insights and great works from the past is a worthwhile goal in its own right.
    - We will also learn how to use rigorous tools and explore questions like "Where does correctness come from?" and "What are the limitations of correctness?"
        - A fascinating parallel comes from the study of high-precision machining with similar basic questions like "What is flatness?" and "How do we establish flatness?"
- Get ready for the future?
    - Who knows, maybe one day software companies will be liable for their products and companies will have to honor warranties in order to be competitive.
    - In such a world, the economics of formal verification may become much more compelling, and students from 505 should have a head start as we'll already

have practice formalizing and making rigorous guarantees about software systems!

How

We can only really learn new material by experimenting with it and getting feedback on our efforts. In 505, participants will get practice with ideas from the course through homework assignments and reading reflections. 505 staff will grade homeworks and reflections to provide feedback and assign “points” for each submitted deliverable.

Homework Assignments

We are planning on 6 homework assignments this quarter. Each homework will involve formalizing programs and proving properties about their behavior, both in Rocq and doing some traditional “pen and paper” proofs (i.e., written in English + mathematical notation). We also expect several homework assignments to involve writing some code in the languages we are formalizing.

The first homework will be short, mainly to ensure that everyone has their software environment set up for 505 and is familiar with the basics of writing proofs in Rocq and in traditional “pen and paper” style. This first homework will be worth **50** points and is the only homework that course participants must complete individually.

Subsequent homeworks will be worth **150** points and *can be completed with a partner*. Out of the 150 total points, approximately 30 points come from “challenge problems” which may require substantially more work. We expect completing the normal problems to take roughly 6 hours of work over a two-week period, though how long each homework takes and how long each group takes can vary considerably. Please see the grading policy below for more detail on how points add up to determine an overall course grade.

Homework assignments will be posted on the course website and should be submitted by uploading your code and proofs to the 505 Gradescope page. The first homework should be submitted individually. For subsequent homeworks, if you are working with a partner, **please submit just one copy** on Gradescope, and **add each group member’s name** using the Gradescope interface.

Reading Reflections and Discussions

We will read and discuss 5 papers this quarter. Several of the readings are relatively non-technical; they're intended to foster discussion and get us thinking about the "big picture" for software reliability and verification.

Each reading reflection will have three short components:

1. **Summary and Reconstruction**
    - A concise summary of the paper in your own words.
    - Focus on the parts of the paper that relate to your personal answers that follow, with attention to the supporting evidence the paper gives.
2. **Something that resonated with you**
    - Discuss an aspect of the paper that resonates with your personal experience. Have you seen examples of the paper's ideas in your work? Did the paper unlock something for you that will cause you to approach future problems differently? This part will often focus on the paper's reusable, conceptual insights, but that's just a trend, not a requirement.
3. **Something you felt was missing**
    - Discuss an aspect of the paper that seems to contradict your personal experience, or something from your personal experience that doesn't seem to fit into the paper's theory.
    - Alternatively (or additionally), a succinct exploration of what future research or industrial applications this paper might inspire

Each component is worth 15 points and should be roughly 5 sentences, though you are certainly welcome to write more, especially in the second and third parts. Each part is graded on a ternary scale: 5 points for being a nonempty string, 10 points for not sounding like an LLM, and 15 points for connecting to your personal experience. There are 45 available points (15 points x 3 parts).

There are 5 additional points available, if you would like to invest additional time, thought, and effort in your reflection. In that case, please indicate using the checkbox on the Gradescope submission that you are attempting to get the additional 5 points. We will take that into consideration while grading your reflection.

There are several ways to earn the additional 5 points in a reading reflection:

- Connect the paper back to what we are learning in class. How are the authors' concerns addressed or not addressed by the techniques we are studying?
- Connect the paper to broader historical and technical context in society. What are the far-reaching implications of the authors' claims? Will we care about these ideas next year? In 10 years? In 100 years?
- Propose a fairly detailed follow up investigation or further investment in the ideas from the paper. What would be a good research project taking the next steps to explore the authors' claims more deeply? Is there a startup company you could build around the ideas from the paper?
- Read additional related resources (many linked to from the Readings page and in the Community Notes) and incorporate discussions of those resources into your reflection.

There are probably many more ways to earn the extra 5 points in reflections too! The points above are just some suggestions to get you thinking. **Be creative!**

Links to PDFs for each paper and the reflection due dates will be posted on the course website. Reading reflections should be entered in the relevant assignment on the 505 Gradescope page. The Gradescope assignment textbox interface is admittedly a bit spartan, but using Gradescope provides the staff with a uniform interface for reviewing reading reflections. You might find it helpful to draft your responses in a separate text editor or word processor, and then copy-paste your answers into Gradescope.

After reading reflections are due, we may briefly discuss the paper together during lecture in smaller breakout groups and as a class all together.

Community Docs

A great way to deeply understand new concepts is by drafting, editing, and discussing notes with colleagues. To help practice these skills and further build a supportive community, this quarter we'll be collaboratively building the 505 Community Notes.

There is no prescribed approach to building the Community Notes; they should reflect your understanding of the PL topics we're learning and what was helpful to you along the way. The notes should also help serve as a guide for others who are reviewing lecture material or perhaps looking for a slightly different perspective on some topic.

While we're not dictating any particular required content in the Community Notes, starting from a blank page can be intimidating, so we may seed the doc with some examples to get things moving. We expect folks may want to add things like:

- Summaries (and corrections!) of lecture material.
- Examples, figures, and diagrams that highlight important PL concepts.
- Finer-grained timestamp breakdowns for the topics covered in lecture videos.
- Overviews as well as detailed examples of various Rocq tactics.
- High-level "pen and paper" proofs of key theorems we see in Rocq during lecture.

A real easy way to contribute to the docs is by porting over and cleaning up versions of the notes from the previous offering!

505 is also supported by several additional important documents you can contribute to:

- Software Setup
    - Instructions for installing Rocq, VS Code, and plugins on various platforms.

- Style Guide
    - Guidance and best practices for writing proofs, etc.
- Lecture Slides
    - Course participants can leave comments via the Google Slides interface.
- Lecture and Homework Code
    - You can fork and issue merge requests via the CSE Gitlab.
- Rocq Fundamentals
    - This 505 reference document shows more of what's going on "under the hood" at a lower level when we use Rocq.

To encourage contributions and recognize the hard work it takes to write high quality docs, each week you will be able to earn up to 10 extra points for contributing the 505 Community Notes or other course documents. **This does not necessarily require a significant amount of time!** We welcome contributions of any size, and you are encouraged to help out however and whenever you can.

To calibrate, here are some rough examples of potential contributions and approximately how many points they may be worth:

Pts	Example Community Docs Contribution
1 point	Fixed some typos and added some links or other citations.
5 points	Added a clear, actionable comment pointing out how something is confusing.
5 - 10 points	Added or enhanced a timestamp breakdown for a lecture video.
5 - 10 points	Incorporated conclusions from the discussion board back into the docs.
10 points	Added a figure or example to highlight some PL concept.
10 points	Expanded on a section to resolve a comment left on why it was confusing.
10 points	Drafted a new section of the doc on a lecture topic or proof technique.
10 points	Edited a section of the doc for clarity and consistency with other sections.
10 points	Made a short video demonstration of some proof technique or similar.

In terms of effort, we roughly expect:

- 1 point contributions to require a couple of minutes
- 5 point contributions to require around 15 minutes
- 10 points contributions to require 30 minutes

As with any deliverable, Community Docs contributions should be submitted in the 505 Gradescope (sign up using add code **X2G5Y5**). In these “docs assignments" you will be able to indicate how many points you feel your contribution should earn and briefly describe how you helped grow or improve the docs. We will look at your contribution and may assess that it is worth more or fewer points than you estimated, but we will always give thanks and feedback for every contribution.

Giving and Accepting Feedback

We often identify closely with our work, whether it is in code, written English, a visual diagram, or a video tutorial. It can be very difficult to accept even well-intentioned constructive feedback. Knowing this, many also hesitate to give such feedback. We want 505 to be a space where folks feel comfortable exploring, making mistakes, learning, and helping each other along the way.

Always be extra patient, kind, and respectful when you leave comments on or edit someone else’s work! When you receive feedback or see that someone has tried to improve your own work, thank them for their effort and take a few moments to understand their contributions and the motivation behind them. If you disagree with edits or suggested improvements, discuss the tradeoffs with your colleague calmly and respectfully. Remember that our docs are all versioned, so nothing should ever be lost. Please also review the 505 Ground Rules.

Feedback and Grades

The course staff will strive to provide prompt and helpful feedback for each deliverable. This feedback will also indicate how many points each participant earned.

OK, so what’s the deal with this whole “points” thing? It’s an **experiment** to help empower you to choose your target overall course grade and best manage your time.

In many courses, students struggle to anticipate how much effort must be invested on each assignment in order to achieve a desired overall grade. Making matters worse, instructors often grade on a curve, which pits students against one another in unhealthy competition. Students default to fighting for every single point rather than focusing on mastering new material.

But this is all nonsense. We are here to learn, not strive to outcompete our friends and colleagues!

The staff believe that, from the very beginning of the course, students should be able to decide what overall grade they would like to achieve in the course. This choice should be informed by **rough** estimates of how much effort earning that grade may entail. Everyone is busy; you are best positioned to decide how to most wisely invest your time.

So, we're testing a scheme to see how that works. Here's the **planned** breakdown of possible points that can be earned throughout the course (subject to minor tweaks as the course progresses):

Deliverable	**Points**		
	Core	Extra Points	Total
Homework 1	50	0	50
Homeworks 2 - 6	120 × 5 = 600	30 × 5 = 150	150 × 5 = 750
Readings 1 - 5	45 × 5 = 225	5 × 5 = 25	50 × 5 = 250
Community Docs	0	10 × 10 = 100	10 × 10 = 100
Overall	**875**	**275**	**1,150**

Overall course grades will reflect total points earned according to the formula:

$$\lfloor \text{Points} / 25 \rfloor / 10$$

Which leads to the following breakdown (maxed out at 4.0, see detailed version):

4.0 if points >= 1000	3.6 if points >= 900	3.2 if points >= 800
3.9 if points >= 975	3.5 if points >= 875	3.1 if points >= 775
3.8 if points >= 950	3.4 if points >= 850	3.0 if points >= 750
3.7 if points >= 925	3.3 if points >= 825	2.9 if points >= 725

There are roughly 110 points available per week throughout the course. Points get a bit harder to earn as the quarter progresses and the material becomes more complex.

Not everyone needs to get every point on every deliverable. Everyone should be able to get most of the points in a reasonable amount of time, but for students who want to dig more into the material, the extra points provide additional opportunities to practice and engage with the course more deeply.

If you decide to target an overall course grade above 3.5, then you (and, on homeworks, your partner) will need to earn points from challenge problems, extra 5 points from reading responses, or extra points from the community notes. Some challenge problems may be harder to earn than others, though we will always do our best to roughly estimate difficulty for each problem.

We emphasize again that **it's OK** to skip challenge problems if doing so is compatible with your target course grade or you are especially busy some weeks! It's fine to target and work toward whatever overall course grade you believe is the best fit for you given other demands on your time and energy.

Late Policy

Homework solutions, reading reflections, and community doc contributions will always be due by 5pm PT on Fridays. These deadlines are strict.

However, you may use up to two late days for each assignment. Submitting any time after the deadline uses at least one full late day. No credit will be granted after 8am on Monday following a Friday deadline.

Course staff may be slower at answering homework questions on the 505 message board or chat over the weekend. Coming to regular office hours is the surest way to get high quality assistance. The next best option is to ask questions early on the course discussion board. *You are strongly advised to submit deliverables early to avoid any unexpected issues.*

Academic Integrity

When you submit a homework solution or reading reflection, you are asserting that the work it contains is your own. Any attempt to misrepresent the work you did will be dealt with via the appropriate University mechanisms. Please carefully review the UW Academic Misconduct Process which provides the following policy:

> *Engineering is a profession demanding a high level of personal honesty, integrity and responsibility. Therefore, it is essential that engineering students, in fulfillment of their academic requirements and in preparation to enter the engineering profession, shall adhere to the University of Washington's Student Code of Conduct.*
>
> *Any student in this course suspected of academic misconduct (e.g., cheating, plagiarism, or falsification) will be reported to the College of Engineering Dean's Office and the University's Office of Community Standards and Student conduct. (See CoE website for more detailed explanation of the academic misconduct adjudication process). Any student found to have committed academic misconduct will receive a 0-grade on impacted academic work (e.g., assignments, project, or exams).*

If there is any chance you have violated this policy, you must clearly indicate in your submission what work was not entirely your own. If you do, the worst that will happen is you may lose some points on an assignment. This is much better than the alternative.

Where

Course material will be presented during lectures on Tuesdays on Thursdays each week, with each lecture being offered **twice:** 90-minute segments on Tuesdays and Thursdays at 10:00am in **CSE G04** and a combined 180 minute session Tuesday evenings at 6:30pm in **CSE2 G10**. You can attend whichever lecture is most convenient for you, regardless of whether you registered for “505” or “P505”.

Welcoming, Supportive Space

As trail guides and adventurers, we will be existing in a community together this quarter as we explore the foundations of programming languages. Every community develops shared values and norms that guide interactions among its members. **Our primary goal is to create a welcoming environment where everyone can feel comfortable trying new things and tackling hard problems.** No one can be expected to learn effectively if they are being harassed or bullied, nor if they are made to feel like they do not belong.

Ground Rules

To lay the groundwork for such an environment, we have set out the following ground rules.

- **Harassment, bullying, and threats have no place in 505.** This may sound obvious, but it is too important not to mention explicitly. There is a limitless number of ways to harass people, so we cannot list all of them, but they might include harassment based on gender, sexual orientation, disability, physical appearance, body size, skin color, race, or religion, as well as sexual images in public spaces, deliberate intimidation, stalking, following, harassing photography or recording, inappropriate physical contact, and unwelcome sexual or romantic attention. Don’t do any of these things!
- Be mindful of how you talk to and about other people in the class. Language shapes how we think, and this class is in large part about the importance of (programming) language. **Do not use language that creates the impression that someone does not belong in our community.**
    - **Blatant -isms**: saying things that are explicitly racist, sexist, homophobic, etc. is not allowed. For example, do not argue that some people are less intelligent because of their gender, race or religion.
    - **Subtle -isms** and small mistakes made in conversation are *not* a violation of these rules. However, repeating something after it has been pointed out to you that your language works against a welcoming environment, or antagonizing or arguing with someone who has pointed out your subtle -ism is considered unwelcoming behavior *and is not allowed*.
    - **Maliciousness**: deliberately attempting to make others feel bad, name-calling, singling out others for derision or exclusion is not allowed. For example, don’t tell someone they’re not a real programmer, or that they don’t belong in this class.

    - “**Jokes**” (or serious actions, for that matter) that point out ways in which someone might not seem to belong are not okay, either.
- **Do not use course communication channels for completely unrelated discussion.** It’s fine to have “partially off topic” conversations that might be interesting to the community, such as related topics, videos, or events that people might find interesting. But do not use even such miscellaneous channels to advertise things completely unrelated to the course. Do not direct message people or email them about topics unrelated to the course without their permission. For example, do not ask someone on a date.
- Another important aspect of our community ground rules is Academic Integrity. See that section for details.

How We Handle Ground Rules Violations

We want you to know that we have your back. We encourage you to get a staff member involved if you notice someone violating the Ground Rules and cannot or do not want to resolve it yourself. We recommend contacting staff via email for these issues.

We hope it doesn’t come to this, but at our discretion, we will escalate violations of these rules to the CSE advising staff, possibly leading to violators being permanently removed from the course. These rules are a rough sketch, and we can’t possibly write down all the ways someone might hurt the community, so we may escalate issues that are not explicitly written down here. On the other hand, when appropriate, we want to be forgiving, too: if it seems like you’ve made a good-natured mistake, we want to give you space to grow and learn.

Fostering a Welcoming Environment

It takes more to create a welcoming environment than just following the ground rules. Try to make an extra effort to be kind and empathetic in how you act! Here are a few suggestions on how to make our community even more welcoming:

- Assume that all your fellow students have what it takes to succeed in this course! No matter what people look like, everybody in 505 is here because they want to learn about the foundations of programming languages.
- Let people choose how much they want to share about themselves. It can be tempting to ask somebody about their background based on something about them you can see or hear. They might want to talk about themselves, but maybe they just want to learn about programming languages. Let them choose!
- Avoid “feigning surprise”. Don’t act surprised when someone says they don’t know something. This applies to both technical things ("What?! I can't believe you don't know what structural induction is!") and non-technical things ("You don't know who Benjamin Pierce is?!"). Feigning surprise has absolutely no social or educational benefit: When people feign surprise, it's usually to make them feel better about themselves and others

feel worse. And even when that's not the intention, it's almost always the effect. We want everyone to feel comfortable saying “I don’t know” and “I don’t understand”.

- Avoid “well-actually”. A well-actually happens when someone says something that's almost, but not entirely, correct, and you say, "well, actually…" and then give a very minor correction. This is especially annoying when the correction has no bearing on the actual conversation. This doesn't mean we aren’t interested in correctness or precision in this class—in fact, we are more interested in correctness and precision than pretty much anybody else! But almost every occurrence of a well-actually in our experience is about grandstanding and showing off, not truth-seeking.
- No subtle -isms. Subtle racism, sexism, homophobia, transphobia, and other kinds of bias are not welcoming. These are often small things that we all sometimes do by mistake. For example, saying "It's so easy my grandmother could do it" is a subtle -ism. Like the other bullet points in this list, it’s not a giant deal to mess up -- just apologize and move on.

Acknowledgments

We gratefully acknowledge !!con and the Recurse Center’s ideas on building community. Much of our ground rules are adapted from their codes of conduct and social rules documents.

Resources

- Course Website
    - Includes links to all the resources below plus:
    - Lecture times and corresponding Zoom link
    - Office hours and corresponding Zoom links
- Slides and Notes
    - Shared folder with non-code lecture material
- 505 Gitlab
    - Code from lecture and homework skeletons
- Discussion Board
    - The **official** channel for help, clarification, and announcements outside of lecture
- Gradescope
    - For submitting reading summaries and homework assignments
    - To sign up, use this add code: **X2G5Y5**

When

This spreadsheet has our rough plan for what we will cover and when deliverables are due this quarter, though remember:

"Plans are worthless, but planning is everything." [source]

Ten weeks is short, so we will try to balance between covering core concepts in depth and seeing as many useful techniques as possible!