Compilers Wrapup

Martin Kellogg

Course Announcements

- Exam on Thursday, May 15 at 2:30pm
  - **in-person** in CULM Lect 1

Course Announcements

- Exam on Thursday, May 15 at 2:30pm
  - **in-person** in CULM Lect 1
- I will hold a review session early next week
  - keep an eye on Discord for a poll about when

Course Announcements

- Exam on Thursday, May 15 at 2:30pm
  - **in-person** in CULM Lect 1
- I will hold a review session early next week
  - keep an eye on Discord for a poll about when
- PA4 is due at **the same time as the exam**
  - no extensions on this one

Course Announcements

- Exam on Thursday, May 15 at 2:30pm
  - **in-person** in CULM Lect 1
- I will hold a review session early next week
  - keep an eye on Discord for a poll about when
- PA4 is due at **the same time as the exam**
  - no extensions on this one
- I was in Ottawa last week for ICSE
  - unfortunately, I got sick while I was there and didn't have time to properly prep this lecture

Course Announcements

- Exam on Thursday, May 15 at 2:30pm
  - **in-person** in CULM Lect 1
- I will hold a review session early next week
  - keep an eye on Discord for a poll about when
- PA4 is due at **the same time as the exam**
  - no extensions on this one
- I was in Ottawa last week for ICSE
  - unfortunately, I got sick while I was there and didn't have time to properly prep this lecture
  - so, you're going to be treated to nonsensical rambling instead
    - importantly: it **won't be on the exam**

Agenda

- Review of course themes
- Other stuff you can do with compilers-like techniques
  - + CS 735 Sp26 preview
- Thoughts about the future
- Time for you to do course evaluations and/or Q&A

Course Themes

- Relationship between programming and math
- Importance of getting details right
- Automated reasoning via inference rules
- Transformation pipelines

Course Theme: Programming and Math

Course Theme: Programming and Math

- Lambda calculus vs Turing machine basis for CS

Course Theme: Programming and Math

- Lambda calculus vs Turing machine basis for CS
- Functional programming
  - Code is more like math functions
  - Referential transparency makes reasoning easier
  - Transformations rather than destructive updates

Course Theme: Programming and Math

- Lambda calculus vs Turing machine basis for CS
- Functional programming
  - Code is more like math functions
  - Referential transparency makes reasoning easier
  - Transformations rather than destructive updates
- Curry-Howard Correspondence
  - proofs = programs
  - theorem we're trying to prove = type of the program
  - proof of a theorem exists = instance of that type exists
  - much of PL theory is built on top of this idea
    - see 735 preview later

Course Theme: Detail-orientation

Course Theme: Detail-orientation

- Vibes-based programming is ok when the spec is fuzzy
  - e.g., “show a human a picture of a cat”

Course Theme: Detail-orientation

- Vibes-based programming is ok when the spec is fuzzy
  - e.g., “show a human a picture of a cat”
- When the spec is tight, however, you have to get every detail right in order for anything to work
  - this is the natural state of computers
    - the only reason you might believe otherwise is that many others have worked hard to make that so

Course Theme: Detail-orientation

- Vibes-based programming is ok when the spec is fuzzy
  - e.g., “show a human a picture of a cat”
- When the spec is tight, however, you have to get every detail right in order for anything to work
  - this is the natural state of computers
    - the only reason you might believe otherwise is that many others have worked hard to make that so
- Lower abstraction level = must get more details right
  - to understand bugs in these systems, must understand both theory and practical implementation
  - job security: how good are LLMs at getting every detail right?

Course Theme: Detail-orientation

- Vibes-based programming is ok when the spec is fuzzy
  - e.g., “show a human a picture of a cat”
- When the spec is tight, however, you have to get every detail right in order for anything to work
  - this is the natural state of computers
    - the only reason you might believe otherwise is that many others have worked hard to make that so
- Lower abstraction level = must ge
  - to understand bugs in these s
    theory and practical impleme
  - job security: how good are LLMs at getting every detail right?

Is a compiler an easy or hard software engineering problem?

Course Theme: Automated Reasoning

Course Theme: Automated Reasoning

- Interesting properties of programs are usually undecidable

Course Theme: Automated Reasoning

- Interesting properties of programs are usually undecidable
- Computers do deductive reasoning via inference rules
  - we use abstraction + approximation to ensure that any proof that finishes really is a proof

Course Theme: Automated Reasoning

- Interesting properties of programs are usually undecidable
- Computers do deductive reasoning via inference rules
  - we use abstraction + approximation to ensure that any proof that finishes really is a proof
- When reasoning about why a program or optimization is correct, you should try to formalize your thinking
  - if you can write down the reasons that the property of interest is true, an automated reasoning tool can prove it

Course Theme: Automated Reasoning

- Interesting properties of programs are usually undecidable
- Computers do deductive reasoning via inference rules
  - we use abstraction + approximation to ensure that any proof that finishes really is a proof
- When reasoning about why a program or optimization is correct, you should try to formalize your thinking
  - if you can write down the reasons that the property of interest is true, an automated reasoning tool can prove it
- Inference-rule-based systems are arbitrarily extensible
  - abstract interpretation, pluggable types are two ways
  - hard part: choosing the right set of rules + abstractions

Course Theme: Transformation Pipelines

Course Theme: Transformation Pipelines

- Compilers are structured as a pipeline of transformations
  - Lexer -> Parser -> Typechecker -> IRs -> Assembly

Course Theme: Transformation Pipelines

- Compilers are structured as a pipeline of transformations
  - Lexer -> Parser -> Typechecker -> IRs -> Assembly
- Each stage should preserve semantics
  - but what “preserve semantics” means can be in the eye of the beholder (e.g., undefined behavior in C)

Course Theme: Transformation Pipelines

- Compilers are structured as a pipeline of transformations
  - Lexer -> Parser -> Typechecker -> IRs -> Assembly
- Each stage should preserve semantics
  - but what “preserve semantics” means can be in the eye of the beholder (e.g., undefined behavior in C)
- The better we are at doing proofs, the more transformations we can safely do
  - performance and correctness come together if we get it right

Course Theme: Transformation Pipelines

- Compilers are structured as a pipeline of transformations
  - Lexer -> Parser -> Typechecker -> IRs -> Assembly
- Each stage should preserve semantics
  - but what “preserve semantics” means can be in the eye of the beholder (e.g., undefined behavior in C)
- The better we are at doing proofs, the more transformations we can safely do
  - performance and correctness come together if we get it right
- Many programs can be structured this way
  - even in domains where it may not be obvious

Other Stuff You Can Do With Compilers

- In fact, any system that can be structured as a transformation pipeline can be viewed as a compiler

Other Stuff You Can Do With Compilers

- In fact, any system that can be structured as a transformation pipeline can be viewed as a compiler
  - allows us to come up with analogies for:
    - type systems (what do we want to prove about this data?)
    - optimizations
    - etc

Other Stuff You Can Do With Compilers

- In fact, any system that can be structured as a transformation pipeline can be viewed as a compiler
  - allows us to come up with analogies for:
    - type systems (what do we want to prove about this data?)
    - optimizations
    - etc
- Many real-life systems have this property
  - any batch-processing system
  - many physical machines (“assembly line”)
  - ???

Example: Compilers for Knitting

Example: Compilers for Knitting

Algebraic Semantics for Machine Knitting

Nat Hurtig 31 March 2025

As programming languages researchers, we're entitled to a certain level of mathematical rigor behind the languages we write and analyze. Programming languages have *semantics*, which are definitions of what statements in the language mean. We

Example: Compilers for Knitting

As programming languages researchers, we're entitled to a certain level of mathematical rigor behind the languages we write and analyze. Programming languages have *semantics*, which are definitions of what statements in the language mean. We

Compilers for X

- Techniques invented for compilers are useful any time you are transforming structured data and you want to:

Compilers for X

- Techniques invented for compilers are useful any time you are transforming structured data and you want to:
  - reject malformed data (typecheck it!)

Compilers for X

- Techniques invented for compilers are useful any time you are transforming structured data and you want to:
  - reject malformed data (typecheck it!)
  - preserve some properties of the data (prove your transformations safe!)

Compilers for X

- Techniques invented for compilers are useful any time you are transforming structured data and you want to:
  - reject malformed data (typecheck it!)
  - preserve some properties of the data (prove your transformations safe!)
  - improve other properties (optimize it!)

Compilers for X

- Techniques invented for compilers are useful any time you are transforming structured data and you want to:
  - reject malformed data (typecheck it!)
  - preserve some properties of the data (prove your transformations safe!)
  - improve other properties (optimize it!)
- If you can internalize this, you can be a very successful engineer

Compilers for X

- Techniques invented for compilers are useful any time you are transforming structured data and you want to:
  - reject malformed data (typecheck it!)
  - preserve some properties of the data (prove your transformations safe!)
  - improve other properties (optimize it!)
- If you can internalize this, you can be a very successful engineer
  - most SDEs in real life are not comfortable proving that each step of their system is correct

Compilers for X

- Techniques invented for compilers are useful any time you are transforming structured data and you want to:
  - reject malformed data (typecheck it!)
  - preserve some properties of the data (prove your transformations safe!)
  - improve other properties (optimize it!)
- If you can internalize this, you can be a very successful engineer
  - most SDEs in real life are not comfortable proving that each step of their system is correct
  - LLMs certainly cannot do this right now
    - ask me if Gemini can write a Cool compiler

CS 735 Preview

- On that note, I've mentioned a few times that I will teach a PhD-level PL class in Sp26
  - what will be different about this class?

CS 735 Preview

- On that note, I've mentioned a few times that I will teach a PhD-level PL class in Sp26
  - what will be different about this class?
- Much more of a focus on the duality of math and programming
  - unlike this class, I will expect "mathematical maturity"
    - i.e., you will do proofs

CS 735 Preview

- On that note, I've mentioned a few times that I will teach a PhD-level PL class in Sp26
  - what will be different about this class?
- Much more of a focus on the duality of math and programming
  - unlike this class, I will expect "mathematical maturity"
    - i.e., you will do proofs
- Wider survey of well-developed analysis techniques
  - abstract interpretation, type systems, Floyd-Hoare logic, typestate analysis, SMT solvers, model checking, etc.

CS 735 Preview

- On that note, I've mentioned a few times that I will teach a PhD-level PL class in Sp26
  - what will be different about this class?
- Much more of a focus on the duality of math and programming
  - unlike this class, I will expect "mathematical maturity"
    - i.e., you will do proofs
- Wider survey of well-developed analysis techniques
  - abstract interpretation, type systems, Floyd-Hoare logic, typestate analysis, SMT solvers, model checking, etc.
- Much less hands-on engineering, much more research
  - course project: build any new analysis tool

Thoughts About the Future

Thoughts About the Future

- Hopefully, this class has given you a solid foundation in thinking carefully about the programs that you write

Thoughts About the Future

- Hopefully, this class has given you a solid foundation in thinking carefully about the programs that you write
  - if so, I fear you will be a rare kind of programmer going forward
    - on the plus side, this is great for job security

Thoughts About the Future

- Hopefully, this class has given you a solid foundation in thinking carefully about the programs that you write
  - if so, I fear you will be a rare kind of programmer going forward
    - on the plus side, this is great for job security
- It looks like we're heading for a future where LLMs generate a lot of code for us

Thoughts About the Future

- Hopefully, this class has given you a solid foundation in thinking carefully about the programs that you write
  - if so, I fear you will be a rare kind of programmer going forward
    - on the plus side, this is great for job security
- It looks like we're heading for a future where LLMs generate a lot of code for us
  - this will make sound program analysis techniques even more important going forward
    - after all, how do you tell if the model's code is right?

That’s All, Folks

That’s All, Folks

- Please do fill out the course evaluation
  - I read them all carefully

That’s All, Folks

- Please do fill out the course evaluation
  - I read them all carefully
  - They’re needed for this class to be approved permanently

That’s All, Folks

- Please do fill out the course evaluation
  - I read them all carefully
  - They’re needed for this class to be approved permanently
  - I’m also happy to take more feedback via whatever channel
    - email is fine, but so is an angry letter taped to my office door

That’s All, Folks

- Please do fill out the course evaluation
  - I read them all carefully
  - They’re needed for this class to be approved permanently
  - I’m also happy to take more feedback via whatever channel
    - email is fine, but so is an angry letter taped to my office door
- Thanks for putting up with the rough edges of this course this semester

That’s All, Folks

- Please do fill out the course evaluation
  - I read them all carefully
  - They’re needed for this class to be approved permanently
  - I’m also happy to take more feedback via whatever channel
    - email is fine, but so is an angry letter taped to my office door
- Thanks for putting up with the rough edges of this course this semester
- It’s been fun!