Questions?

Top
Turing-Completeness

• Lambda calculus is Turing-complete
• C and OCaml can express all the same computations
• And so can a lot of other things:

Magic: The Gathering is Turing Complete

Alex Churchill, Stella Biderman, Austin Herrick

Magic: The Gathering is a popular and famously complicated trading card game about magical combat. In this paper we show that optimal play in real-world Magic is at least as hard as the Halting Problem, solving a problem that has been open for a decade. To do this, we present a methodology for embedding an arbitrary Turing machine into a game of Magic such that the first player is guaranteed to win the game if and only if the Turing machine halts. Our result applies to how real Magic is played, can be achieved using standard-size tournament-legal decks, and does not rely on stochasticity or hidden information. Our result is also highly unusual in that all moves of both players are forced in the construction. This shows that even recognising who will win a game in which neither player has non-trivial decisions to make for the rest of the game is undecidable. We conclude with a discussion of the implications for a unified computational theory of games and remarks about the playability of such a board in a tournament setting.
Turing-Completeness: Representation

• Functional, imperative, etc. languages can all simulate each other

• We can define any language in OCaml (syntax, type system, semantics, interpreter)
  — Or in C, or in Java, or...
Choosing a Language

• We don’t have to choose a language based on what’s possible to compute

• Reasons for choosing a language:
  — Familiarity
  — Existing/legacy code
  — Support infrastructure (libraries, IDEs, ...)
  — Performance
  — Paradigm/ease of expressing certain kinds of ideas

• Exercise: Why might you write a new language instead of using an existing one?
Making a New Language

• Reasons for making a new language:
  — To change the syntax
  — To make a design decision in the other direction
  — To make a common pattern easier to write
  — To make (certain kinds of) programs run faster
  — To combine features that can’t easily be added on to an existing language
  — More reasons than I can think of!
Why are there so many languages?

• Genealogy: [https://erkin.party/blog/190208/spaghetti/](https://erkin.party/blog/190208/spaghetti/)

• Over 800 listed on [https://www.rosettacode.org/wiki/Category:Programming_Languages](https://www.rosettacode.org/wiki/Category:Programming_Languages)

• It’s not that hard to make a new programming language!
Why are there so many languages?

• Example: the ML family

• 1970s: Robin Milner’s group at U. Edinburgh develops Meta Language (ML) to implement a proof-checking program

• 1980s: the original ML team develops Standard ML (SML), while a team at INRIA builds their own version, Caml

• 1996: the Caml team rebuilds the compiler from scratch and adds object-oriented features, making OCaml
  — Meanwhile, SML adds its own set of extensions! vectors, etc.

• 2005: Don Syme at Microsoft Research implements a .NET version of (part of) OCaml, which becomes F#

• And it keeps going: Manticore for parallel programming, F* and Coq for theorem proving, …
Questions?

Top
Combining Language Features

• Functional programming features in imperative/OO languages:

```python
f = lambda x : x + 1  # evaluates to a closure

match event.get():
    case Click((x, y), button=Button.LEFT):
        handle_click_at(x, y)
    case Click():
        pass  # ignore other clicks
```
Combining Language Features

• Functional programming features in imperative/OO languages:
  
• F# combines OO and universal polymorphism
  
• There are now two “anything” types: Object and ‘a
  — Generic functions are defined with universal types
  — Other .NET methods (ToString, etc.) are defined on Object
  — Type inference may infer Object or ‘a as the type of an argument depending on how it’s used, unpredictably
  — Reflection makes this even worse: “generic” functions can case on the type of the input, turning polymorphism ad-hoc
Turing-Incomplete Languages

• We don’t have to choose language based on what’s possible to compute
• And our languages don’t have to be able to compute everything!
• There are some useful Turing-incomplete domain-specific languages (DSLs):
  — Regular expressions
  — HTML
  — Some versions of SQL
  — Interactive theorem provers
Questions?