CS 476 – Programming Language Design

William Mansky

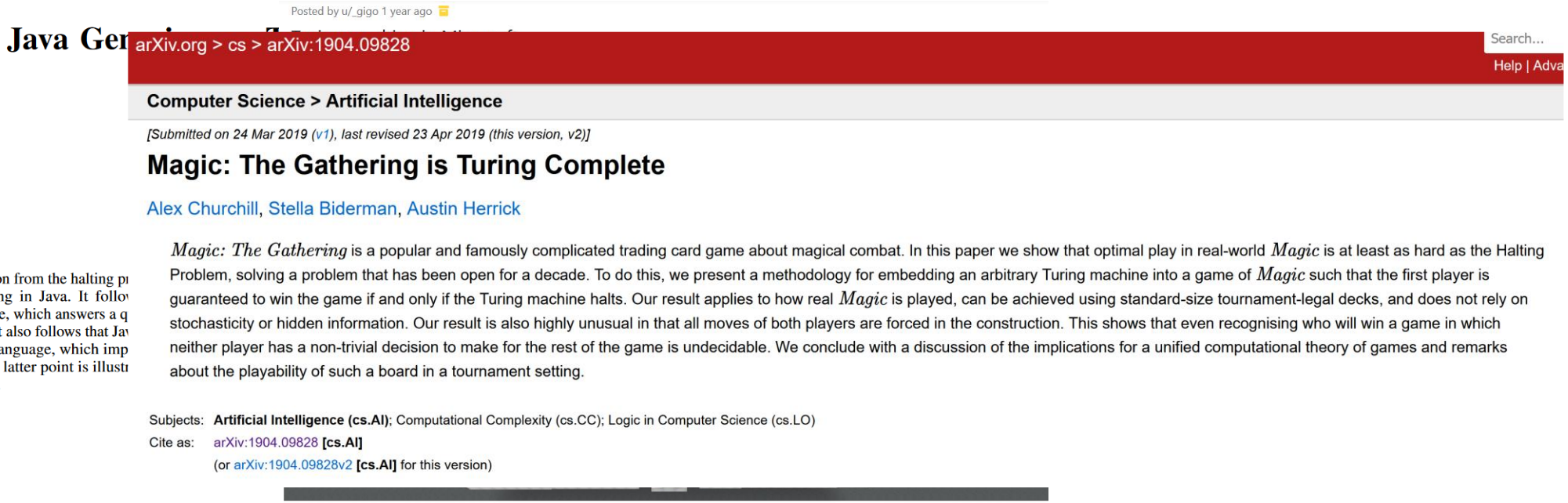
Questions

Nobody has responded yet.

Hang tight! Responses are coming in.

Turing-Completeness

- Almost every programming language is Turing-complete: it can theoretically express any computation
- And so are a lot of other things:



Posted by u/_gigo 1 year ago

Java Ge

arXiv.org > cs > arXiv:1904.09828

Search...

Help | Adva

Computer Science > Artificial Intelligence

[Submitted on 24 Mar 2019 (v1), last revised 23 Apr 2019 (this version, v2)]

Magic: The Gathering is Turing Complete

Alex Churchill, Stella Biderman, Austin Herrick

Abstract

This paper describes a reduction from the halting pr
machines to subtype checking in Java. It follo
checking in Java is undecidable, which answers a q
Kennedy and Pierce in 2007. It also follows that Jav
can recognize any recursive language, which imp
Gil and Levy from 2016. The latter point is illustr
generator for fluent interfaces.

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 a non-trivial decision 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.

Subjects: **Artificial Intelligence (cs.AI)**; Computational Complexity (cs.CC); Logic in Computer Science (cs.LO)

Cite as: arXiv:1904.09828 **[cs.AI]**

(or arXiv:1904.09828v2 **[cs.AI]** for this version)

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…
- We don't have to choose language based on what's *possible* to compute

Turing-Incomplete Languages

- We don’t have to choose language based on what’s *possible* to compute
- But also, 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
  - Configuration languages
  - Interactive theorem provers

Turing-Incomplete Languages

E ::= <#> | <ident>
| *E* + *E* | *E* – *E* | *E* * *E*
| <bool>
| *E* and *E* | *E* or *E*
| not *E*
| *E* = *E*

C ::= <ident> := *E*
| *C*; *C*
| skip
| if(*E*){ *C* } else { *C* }
| while(*E*) { *C* }

Turing-Incomplete Languages

E ::= <#> | <ident>
| *E* + *E* | *E* – *E* | *E* * *E*
| <bool>
| *E* and *E* | *E* or *E*
| not *E*
| *E* = *E*

C ::= <ident> := *E*
| *C*; *C*
| skip
| if(*E*){ *C* } else { *C* }
| while(*E*) max <#> { *C* }

```
while(x > 0) max 10 {
  y := y * x; x := x – 1; }
```

Turing-Incomplete Languages

E ::= <#> | <ident>
| *E* + *E* | *E* – *E* | *E* * *E*
| <bool>
| *E* and *E* | *E* or *E*
| not *E*
| *E* = *E*

C ::= <ident> := *E*
| *C*; *C*
| skip
| if(*E*){ *C* } else { *C* }
| while(*E*) max *E* { *C* }

```
while(x > 0) max x {
  y := y * x; x := x - 1; }
```

Turing-Incomplete Languages

$$\frac{(v > 0)}{(\texttt{while}(e)\ \texttt{max}\ v\ \{c\}, \rho) \rightarrow (\texttt{if}(e)\{\ c; \texttt{while}(e)\ \texttt{max}\ (v-1)\ \{c\}\ \}\ \texttt{else}\ \texttt{skip}, \rho)}$$

$$C ::= \ldots \mid \texttt{while}(E)\ \texttt{max}\ E\ \{\ C\ \}$$

```
while(x > 0) max x {
  y := y * x; x := x - 1; }
```

Turing-Incomplete Languages

- 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
- And we can make a Turing-incomplete general-purpose language just by bounding loops
  - But there will be some functions we can't compute, and we won't know which until we try!

Questions

Nobody has responded yet.

Hang tight! Responses are coming in.

Course Evaluations

- Exercise: Please take a few minutes to fill out the course evaluation, and then submit “finished” for Exercise 11/29.
- If you’d rather not and still want to get credit for the exercise, you can submit “finished” anyway.
- Thank you for your feedback!