Everybody Clap Your Hands

The *Cha-Cha Slide* is Turing Complete

HARRISON GOLDSTEIN, University of Pennsylvania, Philadelphia, PA, USA

ABSTRACT

We describe a scheme for simulating a universal Turing machine using only line-dance instructions from the *Cha Cha Slide*. It would be rather annoying to use in practice (for everyone involved), but we hope someone will try it anyway.

1 INTRODUCTION

This time / We're gonna get funky / Funky
— *DJ Casper.*

In the year 2000, DJ Casper created the ultimate line-dance for the new millenium: the *Cha Cha Slide* [3]. The song reached number 36 on Romania's weekly charts between 2000 and 2004 [1] and quickly established itself as a go-to song for uncomfortable situations where no one actually wants to dance seriously. If you've been to a bar mitzvah, a sweet 16, or a lame wedding, you've probably encountered this phenomenon. The *Cha Cha Slide* is a dance that anyone can do—you just follow some instructions, get a bit funky, and try not to bump into anyone—but it is more than just a universal dance craze. The *Cha Cha Slide* is also a universal computer.

In this paper, we describe a reduction from a universal Turing machine [7] to dance instructions in the *Cha Cha Slide*. We show that, given enough dancers, floor space, and time, DJ Casper could have encoded an arbitrary computation in their funky song. While we do not recommend that anyone use their friends and family as a computer in this way, it is nice to know that such a scheme might work in a pinch.

We dispense with background and get right to our main contribution: §2 presents our reduction with all of the detail that one might need to thoroughly ruin a party. In §3 we describe a few extensions that may improve time and space effiency (as if that's actually something that someone might care about). In §4 we discuss potential issues with our approach. Finally, in §5 we discuss related work and in §6 we wrap up with some comments on Turing completeness as a metric for analyzing things other than Turing machines.

2 REDUCTION FROM (2, 18) TURING MACHINES

For our main result, we use the *Cha Cha Slide* to simulate a (2,18) Turing machine (that is, one with 2 internal states and 18 tape symbols). These machines are known to be universal [6].

Without loss of generality, assume our machine M is a 7-tuple defined as follows:

$\Gamma = \{0, \ldots, 18\}$	Tape Alphabet
0	Blank Symbol
$\Sigma = \Gamma \setminus \{0\}$	Input Alphabet
$Q = \{\mathsf{N}, \mathsf{F}\}$	States
N	Initial State
$F \subseteq Q$	Final States
δ	Transition Function

The machine is given an infinite tape containing symbols from Γ as input, including a finite portion containing characters from Σ. The δ function takes a state and a symbol (read at the current tape head) and produces a new state, a new symbol (to write at the tape head), and an instruction to move either left or right. Since Q and Γ are finite, we can express δ as a table like the one in Table 1. Our goal is to simulate M using the *Cha Cha Slide*.

Input	Output
(N, 5)	(F, 12, R)
(N, 0)	(N, 3, L)
(F, 18)	(F, 0, L)
⋮	⋮

Table 1. The transition function, δ, of M.

We use a line of dancers as an analog for the machine's tape. The position of the DJ marks machine head; we call this space the dance floor the *hot seat*. This setup is pictured in Figure 1. The tape symbols are tracked by the dancers: each dancer memorizes a symbol of the input tape and updates that symbol over the course of the dance. Note that this construction technically requires an infinite number of dancers, but approximating a true Turing machine with a finite one is common in practice [5].

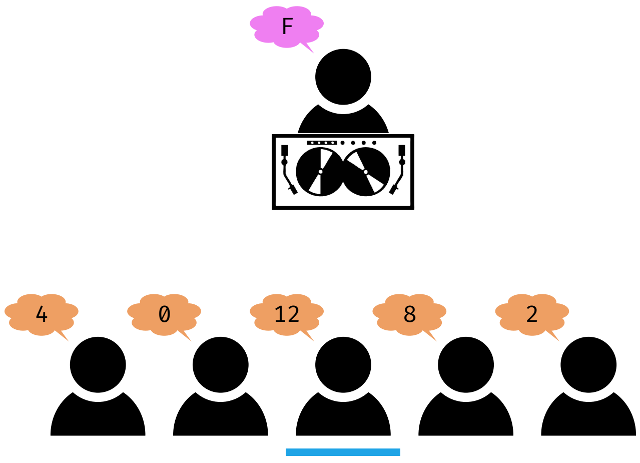

Fig. 1. Simulating a Turing machine with dancers. The DJ keeps track of the machine state, and each dancer keeps track of a single tape symbol. The dancer at the center of the floor is "in the hot seat."

The dance floor is always in one of two states, Funky or not Not Funky, which correspond to the states F and N from the machine M. The DJ can memorize the current state if they wish, but if everyone is doing their job it should be obvious to all involved whether the room is Funky or not.

At each step of the dance, the dancer in the hot seat communicates their tape symbol to the DJ. They could yell the number out, but that might kill the vibe, so a better approach would be to have them hold up their tape number as counted on their fingers. Since few people have 18 fingers, the count can be shown in binary as illustrated in Figure 2. To avoid being too rude, the original Turing machine might do well to avoid symbol 4 when possible. In any case, the important thing is that the hot-seat dancer communicates their cell's content to the DJ.

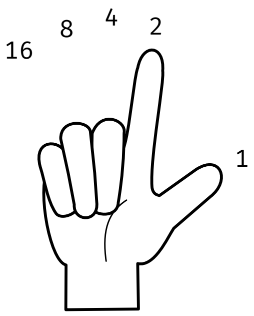

Fig. 2. A hand counting in binary. Each finger represents a power of 2. This hand is currently showing 3.

Knowing both the Funkyness of the room and the value of the tape at the hot seat, the DJ can execute a step of the δ function. They do this with a transition table like the one in Table 2. This table implements the same transition as the one in Table 1, and we explain each call in the following paragraphs.

Input	Output
(Not Funky, 5)	"*Now it's time to get funky / 12 hops this time / Slide to the right*"
(Not Funky, 0)	"*Freeze! / 3 hops this time / Slide to the left*"
(Funky, 18)	"*Now it's time to get funky / Cha-cha now y'all / Slide to the left*"
⋮	⋮

Table 2. An example of the transition function for the DJ. Implements the same transitions as Table 1.

State Transitions. The first part of the δ function changes the state between N and F or Not Funky and Funky. The calls should be self-explanatory: "*Now it's time to get funky*" makes it Funky, and "*Freeze!*" makes things decidedly less Funky.

Writing to the Tape. Writing the blank symbol, 0, is a special action, so we accomplish that with "*Cha-cha now y'all*". There are multiple reasonable schemes for writing a number to the tape. The most obvious is to use "hops": if the DJ calls "*n hops this time*"', the dancer in the hot seat must memorize the symbol n. This is convenient, because if a dancer is made to hop 18 times, they'll almost certainly remember it! Table 2 uses this scheme. Unfortunately, the actual *Cha Cha Slide* only ever calls between 1 and 6 hops, so 18 hops is technically not part of the song. We propose a few more options:

- **Additive Hops.** The DJ can simply call the appropriate number of hops back-to-back. For example, to encode the symbol 8, the DJ might call "*5 hops this time*" followed by "*3 hops this time*".
- **Hops and Stomps.** The song also includes calls "*Right foot let's stomp*", "*Right foot two stomps*", and the symmetrical calls for the left. We can use these along with hops to encode symbols via multiplication by assigning each of those four calls to one of the four primes 7, 11, 13, and 17. The call "*2 hops this time*" followed by "*Right foot let's stomp*" results in $2 \times 7 = 14$ written to the tape. This is amost certainly a bad idea.

Remember that though the dancer in the hot seat is the only one changing their symbol, everyone is encouraged to hop and stomp along.

Moving the Tape. The tape head is moved left and right using "*Slide to the left*" and "*Slide to the right*". Assuming the DJ is facing the dancers (and the dancers know their left from their right), these operations move the tape head to the left and the right as expected.

It should be clear that this scheme faithfully simulates M (again, modulo finiteness), and thus that the *Cha Cha Slide* can simulate an arbitrary (2,18) Turing machine, and *thus* that the *Cha Cha Slide* can simulate any computation. We have not formalized these results in LPC or Agda, as that would be a massive waste of time.

3 EXTENSIONS

The reduction above only uses a handful of the moves available in the *Cha Cha Slide*. Here are a few ideas for how a few more lines might be useful.

Multiple Tapes. Given a grid of dancers, rather than a single line, it would be possible to simulate a *multi-tape* Turing machine. The instruction "*Take it back now y'all*" shifts the current tape back, and shifts a new tape into the hot seat. Shifting forward is a bit trickier, but it can be done with "*Turn it out / Turn it out / Take it back now y'all / Turn it out / Turn it out*". This is a lot of effort, but it works: the first two turns leave every dancer facing the back of the room, then moving back moves the tapes forward, and then doing two more turns faces everyone forward again. (Some might argue that "*Reverse, reverse*" would be a better way to get the room to turn around. But we've heard arguments that "*Reverse, reverse*" means you should reverse twice, which, by idempotence, means that the instruction would be a no-op.)

More States. While a 2 state machine is sufficient, adding more states might make the computation more efficient. Other ways to store state include:

- **Height.** Using the lines "*How low can you go?*" and "*Can you bring it to the top?*" the DJ can ask the room to stand taller or shorter. It may be possible to have arbitrary granularity here, but for simplicity we will say that this enables states High, Middle, and Low.
- **Charlie Brown.** By calling "*Charlie Brown*", the DJ can ask the room to dance like Charlie Brown. This suggests states Charlie Brown and Not Charlie Brown (the same call toggles the state either way).
- **Hands on Your Knees.** Unsurprisingly, "*Hands on your knees*" toggles between Hands on Your Knees and Not Hands on Your Knees.

We could potentially go even further, but since these different states can all be superimposed, they actually result in $2 \times 3 \times 2 \times 2 = 24$ different states! Not bad!

4 THREATS TO VALIDITY

One might be concerned, that, by the nature of Turing machines, the dance described in §2 might never end. We consider this to be a feature.

5 RELATED WORK

This paper is certainly not the first to find Turing completeness in an unexpected place. Both PowerPoint [8] and Magic the Gathering [2] have been shown to be Turing complete, and apparently the Java programming language has as well.

There are also existing connections between computation and dance. A blog post titled *The (Regular) Language of Dance* explores connections between finite automata and swing dance [4].

When searching the *Cha Cha Slide* on Google, you can click on a little button to make the page do the dance. This has nothing to do with Turing machines, it's just cute.

6 CONCLUSION

This paper is obviously just for fun, but our construction and ones like nicely highlight the shortcomings of Turing completeness as a metric. While it is incredibly useful to be able to rank forms of computation based on what they can *theoretically* compute, *practical* computation is also extremely important. In fact, it is often the case that a less powerful form of computation is more useful (e.g., regular expressions are far more useful for computing than the *Cha Cha Slide*).

These insights challenge two arguments that are common among Internet trolls:

- *"Why would you switch from language X to language Y, they're both Turing complete?"* A better-designed language might not be more powerful, but it can certainly be more useful. If we instantiate X to be *Cha Cha Slide* this argument all but evaporates.
- *"You're not a Real Programmer if you use X language—it's not even Turing complete!"* This is a toxic and exclusionary argument, and likely shouldn't even be engaged with as legitimate, but if one must argue then asking "Should I dance the *Cha Cha Slide* instead?" might shut the troll up.

Ultimately, we hope this paper serves as a funny reminder that theoretical properties are not particularly useful in a vacuum, and that theoretical computation alone does not get much done.

ACKNOWLEDGMENTS

This idea originally came up in a conversation with Jeff Tao, and I'll thank him whether or not he wants to be associated with this.

This work was done in my free time and does not represent the views of the University of Pennsylvania, my advisor, or any of my funders.

REFERENCES

[1] [n. d.]. Romanian Top 100. https://web.archive.org/web/20050221112859/http://www.rt100.ro/editie-top-100_x10120.html

[2] Alex Churchill, Stella Biderman, and Austin Herrick. 2019. Magic: The gathering is Turing complete. *arXiv preprint arXiv:1904.09828* (2019).

[3] DJ Casper. 2000. Cha Cha Slide.

[4] Harrison Goldstein. [n. d.]. The (Regular) Language of Dance. https://harrisongoldste.in/languages/2018/04/02/language-of-dance.html

[5] Herman H Goldstine and Adele Goldstine. 1946. The electronic numerical integrator and computer (eniac). *Math. Tables Aids Comput.* 2, 15 (1946), 97–110.

[6] Yurii Rogozhin. 1996. Small universal Turing machines. *Theoretical Computer Science* 168, 2 (1996), 215–240.

[7] A. M. Turing. 1950. Computing Machinery and Intelligence. *Mind* 59, 236 (1950), 433–460. http://www.jstor.org/stable/2251299

[8] Tom Wildenhain. 2017. On the Turing Completeness of MS PowerPoint. In *The Official Proceedings of the Eleventh Annual Intercalary Workshop about Symposium on Robot Dance Party in Celebration of Harry Q Bovik's*, Vol. 2. 102–106.