Ansible Is Turing Complete

Opdebeeck, Ruben; De Roover, Coen

Publication date:
2023

Document Version:
Accepted author manuscript

Ansible Is Turing Complete

Presentation Abstract

Ruben Opdebeeck
Ruben.Denzel.Opdebeeck@vub.be
Vrije Universiteit Brussel
Brussels, Belgium

Coen De Roover
Coen.De.Roover@vub.be
Vrije Universiteit Brussel
Brussels, Belgium

Unsurprisingly, many systems and languages are Turing complete. An interesting subset is that of systems which are Turing complete by accident [1]. A well-known example is that of C++ templates, which led to the practice of template metaprogramming. A more creative showcase of accidental Turing completeness can be found in *Cities: Skylines* [2], involving the flow of simulated city sewage to construct a 4-bit adder. More recently, it has been shown that Helm charts, which abstract over Kubernetes deployments, are Turing complete due to the inclusion of the go-template language [3]. This led some to suggest that every simple language eventually ends up Turing complete [3].

Although difficult to argue that it is a "simple" language [4], [5], Ansible is a popular configuration management language and widely used in practice [6]. At first glance, Ansible does not appear to offer language features that make it truly Turing complete. While it supports conditional and looping execution, such loops can only be executed over a finite sequence of elements. Nonetheless, one can envision various paths to Turing completeness in Ansible. A trivial path would be that of plugins, which allow developers to extend Ansible functionality through Python code. Thus, one could "cheat" by leveraging the Turing completeness of Python. Alternatively, Jinja2, embedded into Ansible as a templating language, could well be Turing complete, similar to the aforementioned embedding of go-template in Helm charts.

In this presentation, rather than relying on the Turing completeness of other languages, we instead show that we can achieve Turing completeness relying solely on a tiny subset of the core Ansible language. The presentation is structured into three parts, which are briefly described in the following sections.

I. Vital Language Constructs

The first part of the presentation will focus on the language constructs that, when combined, achieve general-purpose computations in Ansible. They are briefly summarised below.

- The `include_tasks` action enables us to implement control-flow jumps. Moreover, its ability to recursively include the file in which it is defined itself, enables unbounded recursion.
- The `set_fact` action enables defining and mutating data.
- The `when` task property enables conditional execution. Combined with the actions above, this enables conditional control-flow jumps, conditional recursion, and conditional data manipulations.
- A subset of the Jinja2 templating language enables us to read data, perform simple arithmetic, and list manipulation.

II. Recursive Algorithms

The second part of the presentation focuses on combining the above-mentioned language constructs to implement general recursive algorithms, such as the factorial, Fibonacci, and higher-order map functions. These implementations will illustrate the core insights necessary to achieve general-purpose computations, *e.g.*, implementing recursion in Ansible and management of memory and call stacks. As an example, Listing 1 depicts a tail-recursive implementation of the factorial function that exemplifies the usage of each of the above-mentioned constructs. The `set_fact` task (lines 1–4) sets the `result` and `n` variables. The `include_tasks` (lines 6–7) task recursively includes the same file. Both tasks are made conditional with the `when` property (lines 4 and 7), leading to both tasks being skipped when `n` is 0. Afterwards, the result of the "call" will be available in the variable `result`.

```
1 - set_fact:
2     result: '{{ (result | default(1)) * n }}'
3     n: '{{ n - 1 }}'
4   when: n != 0
5
6 # Recursive "call"
7 - include_tasks: fac.yaml
8   when: n != 0
```

Listing 1. `fac.yaml`, an implementation of the factorial function in Ansible.

III. A Scheme Interpreter

In the final part of our presentation, we will show that Ansible is Turing complete by implementing an "emulator" for a Turing complete language. To this end, we implement an interpreter for Scheme, a procedural, general-purpose programming language. Since implementing a Scheme interpreter is no easy feat, especially not in a language which is not meant to do general-purpose computing, we instead focus

on a subset of its most important semantics. To this end, we present an interpreter with support for Scheme's integers, mutable cons-cells, higher-order functions and calls, simple control structures, and various sorts of let-bindings. The implementation of this interpreter is openly available at https://github.com/ROpdebee/ansible-scheme.

This shows that Ansible is effectively (yet not efficiently!) a Turing complete language. During this part, we will discuss the challenges and implications that these findings bring for practitioners and researchers alike. Most importantly, Ansible being Turing complete poses a challenge for static analysis researchers, since all implications of the halting problem and Rice's theorem therefore apply.

References

[1] M. Rickard, "Accidentally Turing complete," https://matt-rickard.com/accidentally-turing-complete, 2022, accessed: 2023-06-16.

[2] D. Bali, "Cities: Skylines is Turing complete," https://medium.com/@balidani/cities-skylines-is-turing-complete-e5ccf75d1c3a, 2019, accessed: 2023-06-16.

[3] D. Mescheder, "Every simple language will eventually end up Turing complete," https://solutionspace.blog/2021/12/04/every-simple-language-will-eventually-end-up-turing-complete/, 2021, accessed: 2023-06-16.

[4] R. Opdebeeck and C. De Roover, "The pitfalls of Ansible's variable and template expression semantics," Presentation Abstract at the 1st Workshop on Configuration Languages (CONFLANG), 2021.

[5] R. Opdebeeck, A. Zerouali, and C. De Roover, "Smelly variables in ansible infrastructure code: Detection, prevalence, and lifetime," in *Proceedings of the 19th International Conference on Mining Software Repositories*, ser. MSR '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 61–72. [Online]. Available: https://doi.org/10.1145/3524842.3527964

[6] StackExchange, Inc. (2022) 2022 annual stackoverflow developer survey. [Online]. Available: https://survey.stackoverflow.co/2022/#section-most-popular-technologies-other-tools