I'm going to Hurl

Nicole Tietz-Sokolskaya*

Abstract

Modern programming languages have begun to use exceptions but have yet to explore the full utility of this feature. This work introduces Hurl, an exceptional programming language, which leverages the full power of exceptions to toss out any other control flow constructs. We show that this language produces nothing of value.

1 Introduction

Exception handling traditionally has been used to handle errors and unusual events. Using exception handling for control flow has been explored in some mainstream languages, such as Python, where exceptions are used to end generators[4]. No languages known to the author have committed to fully relying on exception handling as the primary control flow primitive. There are other languages that strive for minimalism through the instructions for a Turing machine[1].

This work introduces the Hurl programming language[6]. This language uses two forms of exception handling, along with anonymous functions, to provide a complete programming language suitable for no sort of practical work. This language provides the user with a novel approach to writing the most basic programs, which some may take as an enjoyable challenge[1]. Through the use of Hurl, the programmer also enjoys increased job security from its resistance to understanding by other humans and by LLMs, providing a strong motivation for job preservation.

*My employer had nothing to do with this

[1]This is hypothetical, as the author has not found any willing users. She will not discuss the unwilling users.

2 The Hurl Language

The core of hurl is exception handling. In this section, we introduce the syntax for the core features, give examples from the standard library, explain the available tooling, and review the license choice.

2.1 Syntax

The core syntax of hurl allows you to bind dynamically-typed variables, declare anonymous functions, and use exceptions. We chose not to innovate in much of the core syntax, to remain familiar while deviating in the novel application of exceptions as primary control flow.

Here is an example of the syntax to declare variables, add comments, and perform arithmetic.

```
let language = "hurl";
let year = 2023;
year = year + 1; # happy new year!
let authors = ["Nicole", "just me"];
```

Variables are dynamically typed and must always be initialized. Every statement ends with a semicolon. If you assign to a variable, it finds the variable in the closest surrounding scope and assigns to it, but panics if there is none found.

We can also create anonymous functions.

```
let add = func(a, b) {
    hurl x + y;
};
```

There are a few notable things here: each statement ends in a semi-colon; we use the `hurl` keyword for the first time, since we do not have return values[2]; and since it is dynamically typed, you can pass in strings which will then be concatenated.

[2]The `return` keyword exists and does not do what you expect, unless your brain is as deviant as mine.

Functions can also be recursive. They are not bound to a name until after the function is declared, but since names are resolved at runtime we get to just use the outer name as it will resolve then. Here's an example of an infinite loop using this trick:

```
let overflow = func(depth) {
    println("depth: ", depth);
    overflow(depth + 1);
};

overflow(1);
```

If we run this, we get a stack overflow once we hit the end. This happens because we have no tail-call optimization. Tail-call optimization was considered but was left out because ~~it was too hard to add~~ it is a fun challenge for the programmer to work around.

And now we reach Hurl's defining feature, exception handling. This is the primary control flow mechanism, as you cannot do any useful work in Hurl without it: even heating an office via CPU busy loops cannot happen without exception handling, because otherwise you will stack overflow before you generate sufficient heat.

There are two main ways to throw exceptions in Hurl:

- `hurl` an exception and it will unwind the stack until it finds a handler which catches it. Execution then continues from that handler.
- `toss` an exception and it will walk the stack until it finds a handler to catch it, then it will walk back to where it was tossed from and resume execution from there. Resuming execution requires the handler using the 'return' keyword.

`hurl` is what we normally see as exceptions in other languages. `toss` is a gimmick that strongly resembles continuations[2] by suspending execution to run a handler then resuming. We believe that only `hurl` is necessary, but `toss` is super cute and has to be included for the language to get widespread adoption.

There are three ways to catch an exception:

- `catch (<value>)` will match the literal value provided; any other value falls through to following catch statements
- `catch as <ident>` will catch any value and provide it in the catch body as a bound variable, only in scope for the catch body
- `catch into <ident>` takes *no* body and catches any value, assigning it into the provided already-defined identifier in some scope, then resumes execution after the handlers.

2.1.1 Examples

Using what we've defined previously, we can compute a factorial. This example leverages all three types of `catch` handlers (though only uses `hurl`, not `toss`).

```
let factorial = func(n) {
    try {
        hurl n == 0;
    } catch (true) {
        hurl 1;
    } catch (false) {
        let next = 1;
        try {
            factorial(n - 1);
        } catch into next;
        hurl n * next;
    };
};

try {
    factorial(10);
} catch as x {
    println("factorial(10) = ", x);
};
```

And the result:

```
> hurl run examples/factorial.hurl
factorial(10) = 3628800
```

Closures may also be created. These are omitted due to space constraints.

2.2 Standard Library

Hurl has a good-enough set of built-ins and standard library for getting toy programming problems done.

The full list of built-in functions is available at the Hurl homepage (hurl.wtf). It includes print functions, number conversions, floor/ceiling, and other functionality that the author could have done in Hurl code but ~~was too lazy~~ wanted to optimize for real-world usage and efficiency.

The standard library is entirely written in Hurl and provides abstractions over top of Hurl's syntax to get a more familiar programming feel. To use this functionality, you place it in a file somewhere on your disk[3], then include it with `include <filename-expr>;` where the expression is anything that evaluates to a filename, and it resolves relative to the first interpreted file.

[3] A package manager is in the nice-to-have future projects list.

The full standard library cannot be included here, but we will illustrate how this useful functionality is defined and utilized through illustrating loops.

We can define a loop simply as follows. It takes in two arguments: a function which is the body of the loop, and a set of local variables to invoke the body of the function with. It requires that the body `toss` the values for the next iteration to be bound, and then `hurl` an exception, either true to continue iterating or false to stop iterating.

```
let loop = func(body, locals) {
    let new_locals = locals;
    try {
        body(locals);
    } catch (true) {
        loop(body, new_locals);
    } catch (false) {
        hurl new_locals;
    } catch as update {
        new_locals = update;
        return;
    };
};
```

Here is how you use `loop` to compute the Fibonacci sequence.

```
include "../lib/loop.hurl";

# fibonacci in hurl
let fib = func(locals) {
    let a = locals.1;
    let b = locals.2;
    let iters = locals.3;
    let max_iter = locals.4;

    toss [b, a + b, iters + 1, max_iter];
    hurl iters < max_iter;
};

try {
    loop(test, [0, 10]);
} catch as retval {
    println("computed: ", retval);
};
```

With these custom functions, we're able to build out fairly familiar, but still abnormal, control flow. The uncanny valley ended up being crucial to our experimental results.

The remainder of the standard library defines various conditional, looping, and mathematical constructs. The most notable piece is in the definition of `foreach`, we have to chunk the iterated list and recursively call `foreach` as we approach the limits of the operating system stack. This limitation is due to not having any tail-call optimization and could be improved in future interpreters.

2.3 Tooling and Documentation

Hurl has robust documentation built out and available on the Hurl homepage, at hurl.wtf. It covers all the essentials for a respectable language: installation, usage, syntax, patterns, standard library, debugging, and examples!

One core piece of Hurl's ecosystem is good tooling. Like all good languages, to get adoption, we are building out the tooling users expect to have. The first piece is a code formatter. This exists, and successfully formats Hurl code that you run it on. Future tooling could include an LSP server so that editors can provide useful completions and a package manager.

2.4 License

Hurl is licensed under three licenses:

- GNU Affero General Public License (AGPL-3.0)
- Gay Agenda License (GAL-1.0)
- Commercial licenses

These licenses were chosen carefully to further the goals of the project: advancing the rights and wealth of queer people (especially the wealth of the author). This goal is achieved by using a strong copyleft license, the AGPL-3.0, which allows widespread adoption of the tool outside of work and then, when people bring its value into their jobs, means that their companies will need to purchase a commercial license.

The alternative open-source license, GAL-1.0[5], was written for this project by a wholly unqualified software engineer. She took the MIT license and added requirements, such as supporting LGBTQ rights and saying "be gay, do crime" at least once during use of the software.

We strongly suspect license violations have occurred. The licenses were chosen to compel companies to purchase a commercial license from us for a small eight-figure price, commensurate with the value received.

So far, none have reached out, indicating that there has almost certainly been theft. It's also unlikely that all the users are choosing the Gay Agenda License,

because the assault on LGBTQ rights rages on; with such an earth-shattering project, if it were licensed under GAL, this assault would stop. We've delayed adding telemetry to Hurl not to protect our users, but to protect our delusion that widespread usage has occurred.

3 Experimental Results

We conducted real-world experiments with Hurl to validate some of its properties with real human subjects. We wanted to understand what the experience was like to program using only exceptions for control flow.

3.1 Methodology

We gave each human subject the task of completing Advent of Code[7] using Hurl to validate the standard library and evaluate human efficiency in the Hurl programming language. We evaluated machine subjects by giving them simple problems, like FizzBuzz[3]. We did not feel the need to go any further.

We did not reach statistical significance, because the results were such that it was unethical to continue with further trials.

3.2 Human Trials

Only one human was harmed in these trials. This is a significantly lower count of harmed humans than any other mainstream programming language. These trials were not approved by any internal or external review board.

For our sole human subject, we will use the pseudonym "Nicole" to preserve her privacy[4]. "Nicole" was given the task of doing Advent of Code in Hurl, with a minimum of reaching completing three days of it. She completed three days, choosing to do days 1, 2, and 4, and then she withdrew from further participation in the study. During these three days, she said "I'm going to Hurl" and "Oh god why did someone do this to me." Her attempts to find other people to participate were unsuccessful.

During the human trial, we found that a human can write reasonable code in Hurl, and some of the solutions verge on elegant if you're that kind of person. Additionally, since the interpreter is an absolute amateur hackjob, it lacks any faculties for debugging or performance analysis. This is key for job security, as it means that expertise in Hurl will be hard-won and replacing a Hurl expert will be impossible. How to get Hurl into production without losing your job is left as an exercise for the reader.

3.3 Machine Trials

We tested multiple LLMs[5] on their ability to write Hurl code. They consistently failed to write it in a correct way.

The LLMs were provided with the formal grammar for Hurl, as well as a small sample program. Then they were asked to write an additional small program. The failure modes can primarily be characterized as expecting Hurl to be a reasonably normal language. They would use the `return` statement to return values from a function, rather than as intended. They also tended to expect conditionals, like `if else`.

We expect further improvements with GPT-4 and other newer LLMs, but suspect that the deviation from programming norms provides a barrier for easy LLM production of Hurl code.

4 Conclusion

Hurl presents a novel approach to programming by using only exceptions. It provides the beginnings of robust tooling, and has the possibility of further extension. So far, trials look promising at delivering no real value. The most promising direction for providing value is through job security: if you're able to get it into production, no one else will touch it. It's unclear how to achieve this while remaining employed, which will be the subject of further research.

[4] Her real name is Nicole, and she is the author of this paper, but sshhhh I didn't tell you that.

[5] We think they were GPT-3.5 and Claude, but honestly we forget and lost the chat transcripts. Whoops!

References

[1] Brainfuck (programming language). https://en.wikipedia.org/wiki/Brainfuck. Accessed: 2024-03-04.

[2] Continuations. https://en.wikipedia.org/wiki/Continuation. Accessed: 2024-03-06.

[3] Fizz buzz. https://en.wikipedia.org/wiki/Fizz$_b uzz. Accessed: 2024 - 03 - 13$.

[4] Python standard library exceptions list. https://docs.python.org/3/library/exceptions.html. Accessed: 2024-03-04.

[5] Nicole Tietz-Sokolskaya. Gay agenda license text. https://git.sr.ht/ ntietz/hurl-lang/tree/main/item/LICENSE/GAL-1.0. Be gay, do crime.

[6] Nicole Tietz-Sokolskaya. Hurl, the exceptional language. https://hurl.wtf/. I'm sorry.

[7] Eric Wastl. Advent of code. https://adventofcode.com/.