TYPEWHICH Guide

Luna Phipps-Costin, Carolyn Jane Anderson, Michael Greenberg, and Arjun Guha

May 4, 2021

Contents

1 Introduction

TYPEWHICH is a type migration tool for the gradually-typed lambda calculus and the Grift programming language. Its distinguishing characteristics are the following:

1. TYPEWHICH formulates type migration as a MaxSMT problem.
2. TYPEWHICH always produces a migration, as long as the input program is well-scoped.
3. TYPEWHICH can optimize for different properties: it can produce the most informative types, or types that ensure compatibility with un-migrated code.

For more information on TYPEWHICH, see Phipps-Costin et al. (2021).

2 Building and Testing TYPEWHICH

2.1 Dependencies

To build TYPEWHICH from source, you will need:

1. The Rust language toolchain.
2. The Z3 build dependencies. On Ubuntu Linux, you can run the following command to get them:

```
sudo apt-get install libz3-dev
```

3. Python 3 and PyYAML to run the integration tests. These are installed by default on most platforms. If you can run the following command, then you already have them installed:

   ```
   python3 -c "import yaml"
   ```

2.2 Other Type Migration Tools

The TYPEWHICH benchmarking suite is setup to compare TYPEWHICH to several other type migration tools, some of these tools are in other repositories. You do not need these other tools to use TYPEWHICH, but you do need them to reproduce the evaluation from Phipps-Costin et al. (2021).

1. Rastogi et al. (2012): the TYPEWHICH code includes an implementation of this algorithm, and it has no external dependencies.
2. Migeed and Palsberg (2020) is implemented in Haskell. We have written a parser and printer for their tool that is compatible with TYPEWHICH. This modified implementation is available at the following URL:

   `https://github.com/arjunguha/migeed-palsberg-popl2020`

   Build the tool as described in the repository, and then copy (or symlink) the `MaxMigrate` program to `bin/MaxMigrate` in the TYPEWHICH directory.. On Linux, the executable is at:

   ```
   migeed-palsberg-popl2020/.stack-work/install/x86_64-linux-tinfo6/
   lts-13.25/8.6.5/bin/MaxMigrate
   ```

3. Siek and Vachharajani (2008) is implemented in OCaml 3.12 (which is quite old). The following repository has an implementation of the tool, with a modified parser and printer that is compatible with TYPEWHICH:

   `https://github.com/arjunguha/siek-vachharajani-dls2008`

   Build the tool as described in the repository, and then copy (or symlink) the `gtlc` program to `bin/gtubi` in the TYPEWHICH directory.

   Warning: It is quite hard to build OCaml 3.12 on a modern Linux system. The repository is configured to build a 32-bit Linux executable.

4. Campora et al. (2018) [FILL]

   `https://github.com/arjunguha/mgt`

2.3 Building and Testing

To build TYPEWHICH (and our implementation of Rastogi et al. (2012)), run the following command:

```
cargo build
```

Run the unit tests:

```
cargo test
```

Test TYPEWHICH using the Grift benchmarks:

```
./test-runner.sh grift grift
```

Finally, run the GTLC benchmarks without any third-party tools:

```
cargo run -- benchmark benchmarks.yaml \
  --ignore Gtubi MGT MaxMigrate > test.results.yaml
./bin/yamldiff test.expected.yaml test.results.yaml
```

3 Running TYPEWHICH

The TYPEWHICH executable is symlinked to `bin/TypeWhich`. TYPEWHICH expects its input program to be in a single file, and written in either Grift (extension `.grift`) or in a superset of the gradually typed lambda calculus (extension `.gtlc`), shown in Section 4.

Example Create a file called `input.gtlc` with the following contents:

```
(fun f. (fun y. f) (f 5)) (fun x. 10 + x)
```

This program omits all type annotations: TYPEWHICH assumes that omitted annotations are all **`any`**. We can migrate the the program using TYPEWHICH in two modes:

1. In *compatibility mode*, TYPEWHICH infers types but maintains compatibility with un-migrated code:

   ```
   $ ./bin/TypeWhich migrate input.gtlc
   (fun f:any -> int. (fun y:int. f) (f 5)) (fun x:any. 10 + x)
   ```

2. In *precise mode*, TYPEWHICH infers the most precise type that it can, though that may come at the expense of compatibility:

   ```
   $ ./bin/TypeWhich migrate --unsafe inpuy.gtlc
   (fun f:int -> int. (fun y:int. f) (f 5)) (fun x:int. 10 + x)
   ```

The TYPEWHICH executable supports several other sub-commands and flags. Run `./bin/TypeWhich -help` for more complete documentation.

4 Input Language

`./doc/doc.pdf` has the same content as this file, but with slightly better formatting.

TYPEWHICH supports a superset of the GTLC, written in the following syntax:

[FILL] A few cases missing

b	:=	**true** | **false**	Boolean literal
n	:=	... | -1 | 0 | 1 | ...	Integer literals
s	:=	`"..."`	String literals
c	:=	b | n | s	Literals
T	:=	**any**	The unknown type
	|	**int**	Integer type
	|	**bool**	Boolean type
	|	T_1 **->** T_2	Function type
	|	**(** T **)**	
e	:=	x	Bound identifier
	|	c	Literal
	|	e **:** T	Type ascription
	|	**(** e **)**	Parenthesis
	|	**fun** x **.** e	Function
	|	e_1 e_2	Application
	|	e_1 **+** e_2	Addition
	|	e_1 ***** e_2	Multiplication
	|	e_1 **=** e_2	Integer equality
	|	e_1 **+?** e_2	Addition or string concatenation (overloaded)
	|	**(**e_1**,**e_2**)**	Pair
	|	**fix**f **.**e	Fixpoint
	|	**if** e_1 **then** e_2 **else** e_3	Conditional
	|	**let** x **=** e_1 **in** e_2	Let binding
	|	**let rec** x **=** e_1 **in** e_2	Recursive let binding

5 Experiments

To run the full suite of experiments, you will need to install the third-party type migration tools.

To run the experiments, use the following command:

```
./bin/TypeWhich benchmark benchmarks.yaml > RESULTS.yaml
```

It prints progress on standard error. The output is a YAML file of results.

5.1 Validation

1. The benchmarking script does a lot of validation itself.
2. In `RESULTS.yaml`, look for the string "Disaster". It should not appear!
3. In `RESULTS.yaml`, look for the string `manually_verify`. These are results from experiments where (1) we could not crash the migrated program, and (2) the migrated program has fewer 'any's than the original. So, the table of results counts this migration as one that is 100% compatible with untyped contexts. But, it requires a manual check.
4. Finally, you can compare `RESULTS.yaml` with a known good output from benchmarking:

   ```
   ./bin/yamldiff RESULTS.yaml expected.yaml
   ```

5.2 Results

To generate the summary table found in Phipps-Costin et al. (2021), use the following command:

```
./bin/TypeWhich latex-benchmark-summary RESULTS.yaml
```

To generate the appendix of results:

```
./bin/TypeWhich latex-benchmarks RESULTS.yaml
```

6 Benchmarks

The TYPEWHICH repository has several benchmarks:

1. The `migeed` directory contains the benchmarks from Migeed et al., written in the concrete syntax of TYPEWHICH.
2. The `adversarial` directory contains the "challenge set" from the TYPEWHICH paper.
3. The `grift-suite` directory contains tests from Grift. The `mu/` directory has been modified to use Dyn where it originally used recursive types.
4. The `grift-suite/benchmarks` contains benchmarks from `https://github.com/Gradual-Typing/benchmarks` with the following adjustments:
   (a) The getters and setters in n-body have been removed. They were neither used nor exported we opted to remove these functions from the benchmark. This is discussed in the paper.
   (b) We have changed where in the program some benchmarks print a terminating newline for consistency between the static and dynamic versions.
   (c) Benchmarks that rely on modules are removed

References

J. P. Campora, S. Chen, M. Erwig, and E. Walkingshaw. Migrating gradual types. *Proceedings of the ACM on Programming Languages (PACMPL)*, 2(POPL), Dec. 2018.

Z. Migeed and J. Palsberg. What is decidable about gradual types? *Proceedings of the ACM on Programming Languages (PACMPL)*, 4(POPL), Dec. 2020.

L. Phipps-Costin, C. J. Anderson, M. Greenberg, and A. Guha. Solver-based gradual type migration. `https://khoury.northeastern.edu/~arjunguha/main/papers/2021-typewhich.html`, 2021. In submission.

A. Rastogi, A. Chaudhuri, and B. Hosmer. The ins and outs of gradual type inference. In *ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*, 2012.

J. G. Siek and M. Vachharajani. Gradual typing with unification-based inference. In *Dynamic Languages Symposium (DLS)*, 2008.