

Welterweight Go: Boxing, Structural Subtyping and Generics (Artifact)

Following the Documentation recommendations, this README has the following sections:

- [1. Purpose and claims](#)
 - [2. Getting started](#): Download, installation, and sanity-testing instructions
 - [3. Evaluation instructions](#)
 - [4. Additional artifact description](#)
-

1. Purpose and claims

This artifact demonstrates a minimal prototype implementation of Welterweight Go (WG) as a command-line interpreter tool `wgi`.

WG captures a functional core subset of Go centering on interfaces, structs and methods with Go's structural typing, along with a range of features that exercise Go's type system and runtime mechanisms for type coercions, including anonymous types, underlying types, (dynamic) type assertions, (static) type conversions, generics, type sets, and generic type constraints. WG further extends Go with new features, including generic methods, method set intersections for constraints, and recursively bounded generics.

The `wgi` interpreter can be used to perform a few functions based on the formal model presented in the paper:

- Type check and run WG programs according to the formal reduction of WG terms.
- "Compile" (translate) a WG program to LWG, a low-level representation that corresponds to the Go runtime.
- Type check and run LWG programs according to the low-level type system and reduction of LWG terms.

The main (and only) statement in the submitted paper regarding our implementation and artifact is the below. While the paper does not discuss this implementation (it focusses on presenting our formal developments), we submit this artifact as an accompaniment.

I.84 *We have developed a prototype based on our presented compilation approach. We plan to submit it for artifact evaluation.*

This artifact supports the above statement by providing:

- The source code of our **implementation** as a Go project: see [2.4](#).
 - The `Dockerfile` includes the build commands; `wgi` is pre-built inside the image but the build commands can be rerun.

This artifact further provides:

- The source code for **examples** from our paper and the literature: see [3.1](#).
 - The pre-compiled `wgi` binary and a `Makefile` for **running** the examples: see [2.5](#) and [3.1](#).
-
-

2. Getting started: Download, installation, and sanity-testing instructions

- [2.1. Archive contents](#)
 - [2.2. Prerequisites](#)
 - [2.3. Initial setup and sanity-test](#)
 - [2.4. Main directories](#) inside the artifact container
 - [2.5. Main commands](#) for compiling/running examples
 - [2.6. Usage notes](#)
-

2.1. Archive contents

The artifact archive `paper779.zip` contains:

- The **README** for the artifact (i.e., this document) in three formats:
 - `README-paper779-artifact.html` with `pandoc.css` for formatting – has clickable links;
 - `README-paper779-artifact.md` – links clickable depending on your markdown viewer;
 - `README-paper779-artifact.pdf`.
 - The **main artifact** as a Docker image: `pop126-paper779-docker.tar.gz`
 - A copy of the `Dockerfile` used to build the image. It can be used to rebuild the image (for your own platform) if you wish.
 - For reference, it includes the command-line instructions used to compile the software projects.
 - The submission version of our **paper**: `pop126-paper779.pdf`
-

2.2. Prerequisites

Prerequisites:

- Docker is installed and running – e.g., see this [tutorial](#).

Notes on the Docker image:

- The supplied image has been built on `linux/amd64` and has been successfully tested on `linux/arm64` (Mac M4).
 - You may get a `WARNING: The requested image's platform (linux/amd64) does not match...` but in our testing the image still works by default (via emulation).
 - An explicit emulation command (cf. step 3 in [2.3](#)) is:

```
docker run --rm -it --platform linux/amd64 paper779-artifact
```

- Alternatively, rebuild the image yourself (for your platform); takes just a few minutes. On the command-line go to the directory with the `Dockerfile` and run:

```
docker build -t paper779-artifact .
```

You can then (e.g.) follow step 3 onwards in [2.3](#).

- You can install additional software within the container as required, e.g.,

2.3. Initial setup and sanity-test

Steps from scratch:

1. Put the Docker image `pop126-paper779-docker.tar.gz` in some local directory; say, `$MY_DIR`.

2. **Load** the image. In `$MY_DIR`, do:

```
docker load -i pop126-paper779-docker.tar.gz
```

This only needs to be done once for all future sessions.

3. **Run** a container with an interactive session. In `$MY_DIR`, do:

```
docker run --rm -it paper779-artifact
```

Do this to start each session.

4. Run a pre-written **example**. Inside the container, do:

```
cd $WG_HOME
wg -v -eval=10 examples/wg/hello/hello.wg
```

Expected output. The program should terminate successfully and you should see:

```
[parse] Parsing AST:
package main;
type World[] struct {};
func (x0 World[]) hello[]() World[]@World[] { return x0.hello[]() };
func main() { _ = World[]{}.hello[]() }
```

...snip...

```
9: [R-Call] World[] (World[] (World[] (World[] (World[] (World[] (World[]
(World[] (World[] (World[] {}.hello[ ]))))))))
[Eval] Checking OK: World[]
10: [R-Call] World[] (World[] (World[] (World[] (World[] (World[] (World[]
(World[] (World[] (World[] (World[] {}.hello[ ]))))))))
[Eval] Checking OK: World[]
World[] (World[] (World[] (World[] (World[] (World[] (World[] (World[] (World[] (World[] {}.hello[
 ]))))))))
```

5. **Kick-the-tires**: run all tests in one go. Inside the container, do:

```
cd $WG_HOME
make test
```

Expected output. A mix of examples and unit tests will be run. It may take around 1 or 2 minutes in total. You will see the file name of each example/test possibly followed by some output depending on the program.

Overall you should see:

```
cd test && go test
-- ../examples/wg/oops1a20/fig4/functions.wg
main.FF[ ] {}
Bool[]@Bool[ ] (FF[ ], map[Cond_D:FF.Equal_D:FF.Equal_D Not_D:FF.Not_D], FF[ ] {})
```

```
-- ../examples/wg/oops1a20/fig6/lists.wg
```

...snip...

```
-- ../examples/wg/pop126/ex5_1/process3.wg
```

```
-- ../examples/wg/pop126/ex5_1/process4.wg
```

PASS

```
ok      github.com/rhu1/wgg/test      86.930s
```

2.4. Main directories inside the artifact container

Directory	Path inside container	Env var	Notes
Go path	/home/paper779/artifact/go	GO_PATH	Workspace dir of Go installation
WG home	\$GO_PATH/src/github.com/rhu1/wg/	WG_HOME	WG project base dir
Tests	\$WG_HOME/test		WG project test dir
Examples	\$WG_HOME/examples		WG examples
Go bin	\$GOPATH/bin/		Installation dir of <code>wgi</code> binary

The Go bin dir is included in `$PATH`.

2.5. Commands for compiling/running individual examples

Building and installing `wgi` inside the artifact.

- The `wgi` binary has been pre-compiled and installed for convenience. See the `Dockerfile` for the command-line instructions (just a few lines of code); they can be repeated inside the container if desired.
- See `$WG_HOME/README.md` for more notes on building and installing `wgi`.

Compiling/running individual WG examples.

The following commands should be run from `$WG_HOME`.

- Type check and run a WG program:

```
wgi -v -eval=10 examples/wg/hello/hello.wg
```

- `10` is the number of evaluation steps (≥ 0) to attempt. An error is printed if the program terminates before reaching that number.
- Specify `-1` to attempt evaluation until termination. Non-terminating programs will run until interrupted by the user (e.g., press `ctrl-c`).

- Compile to LWG, type check and print:

```
wgi -v -compile examples/wg/hello/hello.wg
```

- Compile to LWG, type check and run:

```
wgi -v -eval-lwg=-1 examples/wg/pop126/fig1/join0p.wg
```

- Again, `-1` attempts evaluation until termination.
 - Specify `$N 0$` to attempt `N` steps.
- Compile a terminating WG program; then run the WG and LWG programs and check their final result values correspond (cf. the “lollipop” relation, Fig. 21 in Appendix C.3):

```
wgi -v -lolli=-1 examples/wg/pop126/fig1/join0p.wg
```

- Non-terminating programs will run until interrupted by the user (e.g., press `ctrl-c`); the value correspondence check will not be performed.

Some combinations are possible; e.g.

```
wgi -v -eval=10 -compile -eval-lwg=10 examples/wg/hello/hello.wg
```

See [3. Evaluation instructions](#) for more information on the examples.

2.6. Usage notes

Please note the following when using `wgi`.

- This is a minimal and simplistic implementation. While functional, some parts are still being developed and improved. Some aspects of the coding, such as naming conventions, correspond to the formal definitions in the paper but are otherwise not optimal software engineering wise.
 - Similarly, the implementation does not attempt to optimise the formal definitions from the paper in any way. E.g., redundant (but harmless) type coercions and runtime mechanisms required only for metatheoretical purposes are produced in full.
- Error messages and general usability are to be improved.
 - Most errors are reported in “debugging mode” by printing a full stack trace with links to the relevant lines in the source code.
- There is no support for any form of syntactic sugar.
 - *All* sets of empty brackets, such as `[]` for an empty list of generic type parameters, must be written out in full, even if they may be omitted in actual Go.
 - *All* separators (e.g., semicolons after various declarations) must be written out in full, even if they may be omitted in actual Go. The basic rule is that WG must be written similarly to writing an actual Go program all on a single line.
 - See the grammar definitions in the paper and the examples under `WG_HOME/examples`.
- The only base type supported at present is `int`, and the only operation is `+`.

3. Evaluation instructions

- [3.1. Running and checking the tests and examples](#) from the paper and literature.

3.1. Running and checking the tests and examples from the paper and literature

For convenience, the below runs all the examples and tests in one go.

```
cd $WG_HOME
make test
```

We now explain how the examples and tests are organised.

The `make test` command simply enters the `$WG_HOME/test` directory and runs the `go test` command, which performs all of the tests coded inside these two files:

- `wg_test.go` – Tests the following:
 - `$WG_HOME/examples/wg/hello` – Hello world examples.
 - `$WG_HOME/examples/wg/pop126` – All of the examples from the submitted paper.
 - `$WG_HOME/test/wg` – Various positive (`ok`) and negative (`ko`) unit tests.
- `fgg_test.go` – Tests additional examples from the [artifact](#) of the work on [Featherweight Go](#) (FG).
 - All other subdirs of `$WG_HOME examples/wg` outside those already mentioned.

Regarding the source material of the examples:

- The subdirs of `$WG_HOME/examples/wg/pop126` correspond to the figures, sections and labelled examples of the submitted paper where the examples are drawn from. Some of these subdirs also contain variations of the given example. See those parts of the paper for explanations of the examples.
 - The uses of anonymous `func` function types and functions in Figs. 1 and 2 in the paper are encoded in WG using interfaces, structs and methods (cf. L92 in the paper).
 - Some uses of base types such as `float64` and `string` have been replaced by `int` as that is the only base type supported by the implementation at present.
- The subdirs of `$WG_HOME/examples/wg/oops1a20` are ported from the mentioned FG artifact and correspond to the figures in the [FG paper](#).
- All other subdirs are ported from the mentioned FG artifact.
 - Notably, `$WG_HOME/monom/mono-ko` contains examples that are not monomorphisable (e.g., they feature polymorphic recursion), and hence cannot be compiled in FG or actual Go but are fully supported in WG.

Regarding the expected output for the unit tests and examples when running `make test`:

- Negative tests: WG programs that do not type check.
 - All and only the unit tests under `$WG_HOME/test/wg/ko`.
 - (These are tested using `-eval=100` though the `eval` arg itself is moot.)
 - Expected output: the first line of the error message.
- Positive, non-terminating examples.
 - All under `examples/wg/hello`.
 - All under `examples/wg/monom/mono-ko/` except for `incompleteness-subtyping.wg`.
 - These are tested using `-eval=100` and `-eval-lwg=100`.
 - Expected output: no visible output (i.e. no error messages).
- Positive, terminating tests and examples.
 - All unit tests under `$WG_HOME/test/wg/ok`.
 - All examples under `examples/wg/` outside those already mentioned above.
 - These are tested using `-loli=-1`.
 - Expected output: the final result values of the WG program and the compiled LWG program on two separate lines. No error messages.

See [2.5](#) for explanation of the mentioned testing flags.

An example of output from `make test` for the latter case:

```
-- ../examples/wg/oops1a20/fig4/functions.wg
main.FF[ ]{}
Bool[ ]@Bool[ ](FF[ ],map[Cond_D:FF.Cond_D Equal_D:FF.Equal_D Not_D:FF.Not_D],FF[ ]{})
```

- First line is the WG source file.

- Second line is the final result value of the (well-typed) WG program (an empty `FF` struct literal).
- Third line is the final result value of the LWG program (an interface value with: metatheoretical interface type `Bool1@Bool`, concrete RTTI `FF`, method table with methods `Cond_D`, `Equal_D` and `Not_D`, and encapsulated empty struct value `{}` with metatheoretical struct type `FF`).

See the paper for the full details of WG and LWG.

See Appendix [A.1](#) for the output from a run of `make test` in full.

4. Additional artifact description

- [4.1. WG project structure](#)
 - [4.2. Writing your own examples](#)
-

4.1. WG project structure

The WG code project under `$WG_HOME` has a typical Go structure:

- `Makefile` – See the `Dockerfile` for the build/installation commands used.
- `cmd/wgi` – Contains the `main` function for the `wgi` executable.
- `examples` – See [3.1](#).
- `parser`
 - `parser/wg` – Parser sources generated by ANTLR (cf. `make generate-parser`).
 - `parser/pregen` – pre-generated parser sources (cf. `make install-pregen-parser`).
- `internal`
 - `internal/wg` – Main WG implementation.
 - `internal/lwg` – Main LWG implementation.
 - `internal/parser` – Converts ANTLR-generated CST to WG AST.
- `base` and `frontend` – Additional source directories.
- `test` – see [3.1](#).

See `$WG_HOME/README.md` for more notes on building and installing `wgi`.

4.2. Writing your own examples

Inside the container:

```
apt-get install nano vim -y          // Just examples; pick your favourite editor
cd $WG_HOME
vim tmp/scratch.wg                  // Or other editor
```

...edit and save the WG file...

```
wgi -v -eval=-1 tmp/scratch.wg
```

See [2.6](#) for notes (warnings) on writing WG code. No syntactic sugar is supported, so empty brackets (e.g., empty generic parameter lists) and declaration separators (e.g., semicolons) all need to be written out in full.

Appendices

A.1. Expected output of `make test` in full

Below is the full output from a run of `make test`.

```
cd test && go test

-- ../examples/wg/oops1a20/fig4/functions.wg
main.FF[]{}
Bool[]@Bool[] (FF[],map[Cond_D:FF.Cond_D Equal_D:FF.Equal_D Not_D:FF.Not_D],FF[]{})

-- ../examples/wg/oops1a20/fig6/lists.wg
main.Cons[main.Bool[]]{head:main.FF[]{}, tail:main.Cons[main.Bool[]]{head:main.TT[]{}},
tail:main.Nil[main.Bool[]]{}
  List[Bool[]]@List[Bool[]](Cons[Bool[]],map[Map_D:Cons.Map_D],Cons[Bool[]]{Bool[]@Any[]
(FF[],map[],FF[]{}), List[Bool[]]@List[Bool[]](Cons[Bool[]],map[Map_D:Cons.Map_D],Cons[Bool[]]
{Bool[]@Any[] (TT[],map[],TT[]{}), List[Bool[]]@List[Bool[]]
(Nil[Bool[]],map[Map_D:Nil.Map_D],Nil[Bool[]]{}))})

-- ../examples/wg/oops1a20/fig7/graph.wg
main.A[]{}
main.A[]{}

-- ../examples/wg/oops1a20/fig10/nomono.wg
main.Box[main.Box[main.Box[main.Box[main.D[]]]]]{value:main.Box[main.Box[main.Box[main.D[]]]]
{value:main.Box[main.Box[main.D[]]]{value:main.Box[main.D[]]{}}}}
  Any[]@Any[] (Box[Box[Box[Box[D[]]]],map[],Box[Box[Box[Box[D[]]]]]{Box[Box[Box[D[]]]@Any[]
(Box[Box[Box[D[]]]],map[],Box[Box[Box[D[]]]]{Box[Box[D[]]]@Any[] (Box[Box[D[]]]],map[],Box[Box[D[]]]
{Box[D[]]]@Any[] (Box[D[]],map[],Box[D[]]{D[]@Any[] (D[],map[],D[]{}))}}))})

-- ../examples/wg/oops1a20/fig19/dispatcher.wg
main.Int[]{}
main.Int[]{}

-- ../examples/wg/misc/irregular/irregular.wg
main.Node[main.Nat[]]{label:main.Succ[]{pred:main.Succ[]{pred:main.Zero[]{}},
children:main.Node[main.Pair[main.Nat[]]{label:main.Pair[main.Nat[]]{fst:main.Succ[]
{pred:main.Succ[]{pred:main.Succ[]{pred:main.Zero[]{}}, snd:main.Succ[]{pred:main.Succ[]
{pred:main.Succ[]{pred:main.Succ[]{pred:main.Zero[]{}}}}},
children:main.Node[main.Pair[main.Pair[main.Nat[]]]{label:main.Pair[main.Pair[main.Nat[]]]
{fst:main.Pair[main.Nat[]]{fst:main.Succ[]{pred:main.Succ[]{pred:main.Succ[]{pred:main.Succ[]
{pred:main.Succ[]{pred:main.Zero[]{}}}}}, snd:main.Succ[]{pred:main.Succ[]{pred:main.Succ[]
{pred:main.Succ[]{pred:main.Succ[]{pred:main.Zero[]{}}}}}}},
snd:main.Pair[main.Nat[]]{fst:main.Succ[]{pred:main.Succ[]{pred:main.Succ[]{pred:main.Succ[]
{pred:main.Succ[]{pred:main.Succ[]{pred:main.Succ[]{pred:main.Zero[]{}}}}}, snd:main.Succ[]
{pred:main.Succ[]{pred:main.Succ[]{pred:main.Succ[]{pred:main.Succ[]{pred:main.Succ[]
{pred:main.Succ[]{pred:main.Succ[]{pred:main.Zero[]{}}}}}}}}}}},
children:main.Leaf[main.Pair[main.Pair[main.Pair[main.Nat[]]]]}]}
```



```

    Balanced[Nat[]]@Balanced[Nat[]](Node[Nat[]],map[BalancedMap_D:Node.BalancedMap_D],Node[Nat[]]
{Nat[]@Any[](Succ[],map[],Succ[]{Nat[]@Nat[](Succ[],map[Add_D:Succ.Add_D],Succ[]{Nat[]@Nat[]
(Zero[],map[Add_D:Zero.Add_D],Zero[]{}))}), Balanced[Pair[Nat[]]]@Balanced[Pair[Nat[]]]
(Node[Pair[Nat[]]],map[BalancedMap_D:Node.BalancedMap_D],Node[Pair[Nat[]]]{Pair[Nat[]]@Any[]
(Pair[Nat[]],map[],Pair[Nat[]]{Nat[]@Any[](Succ[],map[],Succ[]{Nat[]@Nat[]
(Succ[],map[Add_D:Succ.Add_D],Succ[]{Nat[]@Nat[](Succ[],map[Add_D:Succ.Add_D],Succ[]{Nat[]@Nat[]
(Zero[],map[Add_D:Zero.Add_D],Zero[]{}))}), Nat[]@Any[](Succ[],map[],Succ[]{Nat[]@Nat[]
(Succ[],map[Add_D:Succ.Add_D],Succ[]{Nat[]@Nat[](Succ[],map[Add_D:Succ.Add_D],Succ[]{Nat[]@Nat[]
(Zero[],map[Add_D:Zero.Add_D],Zero[]{}))}), Succ[],map[Add_D:Succ.Add_D],Succ[]{Nat[]@Nat[]
(Succ[],map[Add_D:Succ.Add_D],Succ[]{Nat[]@Nat[](Zero[],map[Add_D:Zero.Add_D],Zero[]{}))})}),
Balanced[Pair[Pair[Nat[]]]]@Balanced[Pair[Pair[Nat[]]]]
(Node[Pair[Pair[Nat[]]]],map[BalancedMap_D:Node.BalancedMap_D],Node[Pair[Pair[Nat[]]]]
{Pair[Pair[Nat[]]]@Any[](Pair[Pair[Nat[]]],map[],Pair[Pair[Nat[]]]{Pair[Nat[]]@Any[]
(Pair[Nat[]],map[],Pair[Nat[]]{Nat[]@Any[](Succ[],map[],Succ[]{Nat[]@Nat[]
(Succ[],map[Add_D:Succ.Add_D],Succ[]{Nat[]@Nat[](Succ[],map[Add_D:Succ.Add_D],Succ[]{Nat[]@Nat[]
(Zero[],map[Add_D:Zero.Add_D],Zero[]{}))}), Succ[],map[Add_D:Succ.Add_D],Succ[]{Nat[]@Nat[]
(Zero[],map[Add_D:Zero.Add_D],Zero[]{}))}), Nat[]@Any[](Succ[],map[],Succ[]{Nat[]@Nat[]
(Succ[],map[Add_D:Succ.Add_D],Succ[]{Nat[]@Nat[](Succ[],map[Add_D:Succ.Add_D],Succ[]{Nat[]@Nat[]
(Succ[],map[Add_D:Succ.Add_D],Succ[]{Nat[]@Nat[](Succ[],map[Add_D:Succ.Add_D],Succ[]{Nat[]@Nat[]
(Succ[],map[Add_D:Succ.Add_D],Succ[]{Nat[]@Nat[](Zero[],map[Add_D:Zero.Add_D],Zero[]
{}))})})})}), Pair[Nat[]]@Any[](Pair[Nat[]],map[],Pair[Nat[]]{Nat[]@Any[](Succ[],map[],Succ[]
{Nat[]@Nat[](Succ[],map[Add_D:Succ.Add_D],Succ[]{Nat[]@Nat[](Succ[],map[Add_D:Succ.Add_D],Succ[]
{Nat[]@Nat[](Succ[],map[Add_D:Succ.Add_D],Succ[]{Nat[]@Nat[](Succ[],map[Add_D:Succ.Add_D],Succ[]
{Nat[]@Nat[](Succ[],map[Add_D:Succ.Add_D],Succ[]{Nat[]@Nat[](Succ[],map[Add_D:Succ.Add_D],Succ[]
{Nat[]@Nat[](Succ[],map[Add_D:Succ.Add_D],Succ[]{Nat[]@Nat[](Succ[],map[Add_D:Succ.Add_D],Succ[]
{Nat[]@Nat[](Succ[],map[Add_D:Succ.Add_D],Succ[]{Nat[]@Nat[](Zero[],map[Add_D:Zero.Add_D],Zero[]
{}))})})})}), Succ[],map[Add_D:Succ.Add_D],Succ[]{Nat[]@Nat[](Zero[],map[Add_D:Zero.Add_D],Zero[]
{}))})}), Balanced[Pair[Pair[Pair[Nat[]]]]@Balanced[Pair[Pair[Pair[Nat[]]]]
(Leaf[Pair[Pair[Pair[Nat[]]]],map[BalancedMap_D:Leaf.BalancedMap_D],Leaf[Pair[Pair[Pair[Nat[]]]]
{}))})})})

```

```
-- ../examples/wg/misc/irregular/irregular2.wg
```

```

Node[Nat[]]{Succ[]{Succ[]{Zero[]{}}, Leaf[Pair[Nat[]]]{}}
Balanced[Nat[]]@Balanced[Nat[]](Node[Nat[]],map[BalancedMap_D:Node.BalancedMap_D],Node[Nat[]]
{Nat[]@Any[](Succ[],map[],Succ[]{Nat[]@Nat[](Succ[],map[Add_D:Succ.Add_D],Succ[]{Nat[]@Nat[]
(Zero[],map[Add_D:Zero.Add_D],Zero[]{}))}), Balanced[Pair[Nat[]]]@Balanced[Pair[Nat[]]]
(Leaf[Pair[Nat[]]],map[BalancedMap_D:Leaf.BalancedMap_D],Leaf[Pair[Nat[]]]{}})

```

```
-- ../examples/wg/misc/monomorph/monomorph.wg
```

```

main.Pair[main.List[main.Bool[]], main.List[main.Nat[]]]{Fst:main.Cons[main.Bool[]]{head:main.FF[]
{}}, tail:main.Cons[main.Bool[]]{head:main.TT[]{}}, tail:main.Nil[main.Bool[]]{}}},
Snd:main.Cons[main.Nat[]]{head:main.Succ[]{pred:main.Succ[]{pred:main.Zero[]{}},
tail:main.Cons[main.Nat[]]{head:main.Succ[]{pred:main.Succ[]{pred:main.Succ[]{pred:main.Zero[]{}},
tail:main.Cons[main.Nat[]]{head:main.Succ[]{pred:main.Succ[]{pred:main.Succ[]{pred:main.Succ[]
{pred:main.Zero[]{}}, tail:main.Nil[main.Nat[]]{}}}}}},
main.Pair[main.List[main.Bool[]], main.List[main.Nat[]]]{Fst:List[Bool[]]@Any[]
(Cons[Bool[]],map[],Cons[Bool[]]{Bool[]@Any[]{FF[],map[],FF[]{}}, List[Bool[]]@List[Bool[]]
(Cons[Bool[]],map[Map_D:Cons.Map_D],Cons[Bool[]]{Bool[]@Any[]{TT[],map[],TT[]{}},
List[Bool[]]@List[Bool[]](Nil[Bool[]],map[Map_D:Nil.Map_D],Nil[Bool[]]{}})), Snd:List[Nat[]]@Any[]
(Cons[Nat[]],map[],Cons[Nat[]]{Nat[]@Any[]{Succ[],map[],Succ[]{Nat[]@Nat[]
(Succ[],map[Add_D:Succ.Add_D],Succ[]{Nat[]@Nat[](Zero[],map[Add_D:Zero.Add_D],Zero[]{}))}),
List[Nat[]]@List[Nat[]](Cons[Nat[]],map[Map_D:Cons.Map_D],Cons[Nat[]]{Nat[]@Any[]{Succ[],map[],Succ[]
{Nat[]@Nat[](Succ[],map[Add_D:Succ.Add_D],Succ[]{Nat[]@Nat[](Succ[],map[Add_D:Succ.Add_D],Succ[]
{Nat[]@Nat[](Zero[],map[Add_D:Zero.Add_D],Zero[]{}))}), List[Nat[]]@List[Nat[]]
(Cons[Nat[]],map[Map_D:Cons.Map_D],Cons[Nat[]]{Nat[]@Any[]{Succ[],map[],Succ[]{Nat[]@Nat[]
(Succ[],map[Add_D:Succ.Add_D],Succ[]{Nat[]@Nat[](Succ[],map[Add_D:Succ.Add_D],Succ[]{Nat[]@Nat[]
(Succ[],map[Add_D:Succ.Add_D],Succ[]{Nat[]@Nat[](Zero[],map[Add_D:Zero.Add_D],Zero[]{}))}),
List[Nat[]]@List[Nat[]](Nil[Nat[]],map[Map_D:Nil.Map_D],Nil[Nat[]]{}}))})})})

```

```
-- ../examples/wg/monom/mono-ko/box.wg
```

```
-- ../examples/wg/monom/mono-ko/box2.wg
```

```
-- ../examples/wg/monom/mono-ko/incompleteness-subtyping.wg
```

```
S[]{}

```

```
S[]{}

```

```
-- ../examples/wg/monom/mono-ko/monom-imp.wg
```

```
-- ../examples/wg/monom/mono-ko/mutual-poly-rec.wg

-- ../examples/wg/monom/mono-ko/mutual-rec-iface.wg

-- ../examples/wg/monom/mono-ko/nested-fix.wg

-- ../examples/wg/monom/mono-ko/two-type-param.wg

-- ../examples/wg/hello/hello.wg

-- ../examples/wg/hello/fmtprintf/fmtprintf.wg


-- ./wg/ok/Test01.wg
A[]{}
A[]{}


-- ./wg/ok/Test02.wg
S[]{}
Any[]@Any[](S[],map[],S[]{}))


-- ./wg/ok/Test03.wg
A[]{}
A[]{}


-- ./wg/ok/Test04.wg
B[]{}
B[]{}


-- ./wg/ok/Test05.wg
MyInt[](43)
MyInt[](43)


-- ./wg/ok/Test06.wg
A[]{}
A[]{}


-- ./wg/ko/Test07.wg
TypeDecl not OK:


-- ./wg/ok/Test09.wg
A[]={int(42)}
A[]={int(42)}


-- ./wg/ok/Test09.wg
A[]={int(42)}
A[]={int(42)}


-- ./wg/ok/Test10.wg


-- ./wg/ok/Test11.wg
S[]{}
S[]{}

```

```

-- ./wg/ko/Test12.wg
Arg expr type must be assignable to param type: arg=A3[], param=I2[]@I[I2[]]

-- ./wg/ok/Test13.wg
Cell[int]{int(42)}
interface {}@interface {}{(Cell[int],map[],Cell[int]{int@interface {}{(int,map[],int(42))}})

-- ./wg/ko/Test14.wg
Type actual must implement type formal: actual=Cell[int], param=I[]

-- ./wg/ko/Test15.wg
Arg expr type must be assignable to param type: arg=Cell[int], param=struct { f int@int }@I[]

-- ./wg/ko/Test16.wg
Type actual must implement type formal: actual=struct { x int@int }, param=I[]

-- ./wg/ok/Test17.wg
int(3)
int(3)

-- ./wg/ok/Test18.wg
int(84)
int(84)

-- ./wg/ok/Test19.wg
MyInt[](84)
MyInt[](84)

-- ./wg/ok/Test20.wg
int(84)
int(84)

-- ./wg/ko/Test21.wg
TypeDecl not OK:

-- ./wg/ko/Test22.wg
TypeDecl not OK:

-- ./wg/ok/Test23.wg
struct { f int@int }{int(42)}
struct { f int@int }{int(42)}

-- ./wg/ok/Test24.wg
int(42)
int(42)

-- ./wg/ko/Test25.wg
Type I[] not struct: I[]{int(42)}:

-- ../examples/wg/pop126/fig1/join.wg
int(42)
int(42)

-- ../examples/wg/pop126/fig1/joinOp.wg

```

```
int(45)
int(45)

-- ../examples/wg/pop126/sec2_2/bar.wg
int(42)
int(42)

-- ../examples/wg/pop126/fig2/extjoinConsMyInt.wg
MyInt[](41)
MyInt[](41)

-- ../examples/wg/pop126/fig2/extjoinConsMyIntOp.wg
MyInt[](44)
MyInt[](44)

-- ../examples/wg/pop126/fig2/extjoinMyIntCons.wg
MyInt[](40)
MyInt[](40)

-- ../examples/wg/pop126/fig2/extjoinMyIntConsOp.wg
MyInt[](43)
MyInt[](43)

-- ../examples/wg/pop126/sec3/ordered.wg
int(1)
int(1)

-- ../examples/wg/pop126/sec3/ordered2.wg
MyInt[](3)
MyInt[](3)

-- ../examples/wg/pop126/sec3/ordered3.wg
struct {}{}
struct {}{}

-- ../examples/wg/pop126/sec3/ordered4.wg
Type actual must implement type formal: actual=A[], param=Ordered[]

-- ../examples/wg/pop126/sec3/ordered5.wg
struct {}{}
struct {}{}

-- ../examples/wg/pop126/ex3_1/assign.wg
Point[] {int(1), int(1)}
Point[] {int(1), int(1)}

-- ../examples/wg/pop126/ex3_1/assign2.wg
Point[] {int(1), int(1)}
Point[] {int(1), int(1)}

-- ../examples/wg/pop126/ex3_1/assign3.wg
Point[] {int(1), int(1)}
Point[] {int(1), int(1)}
```

```

-- ../examples/wg/pop126/ex3_1/assign4.wg
struct { x int@int; y int@int }{int(1), int(1)}
struct { x int@int; y int@int }{int(1), int(1)}

-- ../examples/wg/pop126/ex3_1/assign5.wg
struct { x int@int; y int@int }{int(1), int(1)}
struct { x int@int; y int@int }{int(1), int(1)}

-- ../examples/wg/pop126/ex3_2/constraint.wg
MyStruct[] {int(43)}
MyStruct[] {int(43)}

-- ../examples/wg/pop126/ex3_2/constraint2.wg
MyStruct[] {int(43)}
MyStruct[] {int(43)}

-- ../examples/wg/pop126/ex3_2/constraint3.wg
int(2)
int(2)

-- ../examples/wg/pop126/ex3_2/constraint4.wg
int(42)
int(42)

-- ../examples/wg/pop126/ex3_2/constraintOp.wg
int(85)
int(85)

-- ../examples/wg/pop126/sec4/shape.wg
Circle[] {Point[] {int(1), int(1)}, int(2)}
Shape[]@Shape[] (Circle[], map[draw_D:Circle.draw_D shape_D:Circle.shape_D], Circle[] {Point[] {int(1),
int(1)}, int(2)})

-- ../examples/wg/pop126/sec4/shape2.wg
Circle[] {Point[] {int(1), int(1)}, int(2)}
interface {}@interface {} (Circle[], map[], Circle[] {Point[] {int(1), int(1)}, int(2)})

-- ../examples/wg/pop126/sec4/shape3.wg
Circle[] {Point[] {int(1), int(1)}, int(2)}
interface {}@interface {} (Circle[], map[], Circle[] {Point[] {int(1), int(1)}, int(2)})

-- ../examples/wg/pop126/sec4/shape4.wg
Circle[] {Point[] {int(1), int(1)}, int(2)}
interface {}@interface {} (Circle[], map[], Circle[] {Point[] {int(1), int(1)}, int(2)})

-- ../examples/wg/pop126/sec4/shape5.wg
struct { p Point[]@Point[]; r int@int }{Point[] {int(1), int(1)}, int(2)}
interface {}@interface {} (struct { p Point[]@Point[]; r int@int }, map[], struct { p
Point[]@Point[]; r int@int }{Point[] {int(1), int(1)}, int(2)})

-- ../examples/wg/pop126/sec5/cell.wg
Cell[int] {int(42)}
Cell[int] {int@interface {} {int, map[], int(42)}}

-- ../examples/wg/pop126/sec5/cell2.wg

```

```
StringerCell[] {MyInt[] (42)}
StringerCell[] {MyStringer[] @MyStringer[] (MyInt[], map[String_D:MyInt.String_D], MyInt[] (42))}

-- ../examples/wg/pop126/ex5_1/process.wg
int(42)
int(42)

-- ../examples/wg/pop126/ex5_1/process2.wg
int(42)
int(42)

-- ../examples/wg/pop126/ex5_1/process3.wg
int(43)
int(43)

-- ../examples/wg/pop126/ex5_1/process4.wg
int(44)
int(44)
PASS
ok      github.com/rhu1/wg/test 105.873s
```