Static analyzer for the Nix Expression Language

Статический анализатор для Nix Expression Language

Author: Kostyuchenko Ilya

Supervisor: Victor Kuliamin, Associate Professor, PhD

Source code: github.com/ilyakooo0/tix

> *Nix is a powerful package manager that makes package management reliable and reproducible.*
>
> – **nixos.org**

Nix

- Package manager
- Build system*
- Ad hoc development environments
- Easy (cross-community) dependency management
- Reproducible builds
- *And much more: NixOS, NixOps, Build Caching, Docker …*

Nix Expression Language

- Purely functional
- Lazy
- **Strict, but not static typing**

```
let pkgs = import ./nixpkgs {};
in derivation {
  name = "simple";
  builder = "${pkgs.bash}/bin/bash";
  args = [ ./simple_builder.sh ];
  gcc = pkgs.gcc;
  coreutils = pkgs.coreutils;
  src = ./simple.c;
  system = builtins.currentSystem;
}
```

Terms and definitions

Type system – a logical system comprising a set of rules that assigns a property called a type to the various constructs of a computer program

Type checking – the process of verifying and enforcing the constraints of types.

Hindley-Milner type system – a classical type system for the lambda calculus with parametric polymorphism.

Parametric polymorphism – a type of polymorphism where types are specified by abstract symbols that can represent any type.

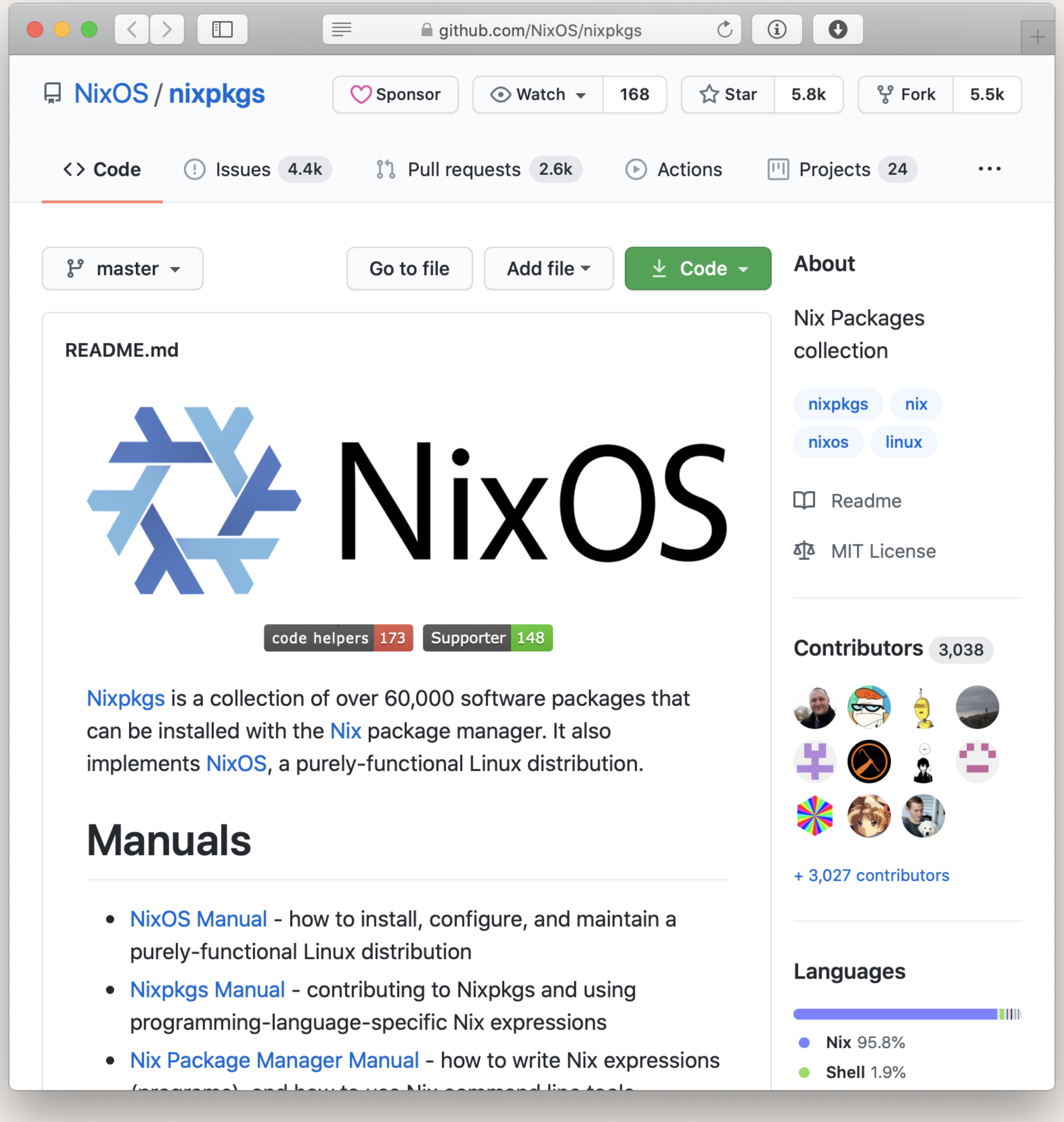
Row polymorphism – a kind of polymorphism that allows one to write programs that are polymorphic on record field types.

Relevance

The Nix ecosystem is slowly becoming popular.

The central repository contains more than 60000 packages.

There is a need to make writing Nix expressions less error-prone.



Nix Packages collection

Nixpkgs is a collection of over 60,000 software packages that can be installed with the Nix package manager. It also implements NixOS, a purely-functional Linux distribution.

Manuals

- NixOS Manual - how to install, configure, and maintain a purely-functional Linux distribution
- Nixpkgs Manual - contributing to Nixpkgs and using programming-language-specific Nix expressions
- Nix Package Manager Manual - how to write Nix expressions

Relevance

Nix Expression Language: **Strict, but not static typing**.

Adding static typing would:

The typechecker should help find issues in existing code

Relevance

Nix Expression Language: **Strict, but not static typing**.

Adding static typing would:

- Increase reliability of code

The typechecker should help find issues in existing code

Relevance

Nix Expression Language: **Strict, but not static typing**.

Adding static typing would:

- Increase reliability of code
- Lower costs of development (resolving issue earlier is cheaper)

The typechecker should help find issues in existing code

The goal

The goal

1. Develop a type static system for the existing Nix expression language

The goal

1. Develop a type static system for the existing Nix expression language
2. Develop a type checker for this type system

Existing typecheckers

Existing typecheckers

- github.com/regnat/ptyx - Abandoned and incomplete.

Existing typecheckers

- github.com/regnat/ptyx – Abandoned and incomplete.
- github.com/regnat/tix – An older version of ptyx. Abandoned, terribly incomplete.

Existing typecheckers

- github.com/regnat/ptyx – Abandoned and incomplete.
- github.com/regnat/tix – An older version of ptyx. Abandoned, terribly incomplete.
- github.com/haskell-nix/hnix – has a module with a type checker. Is very experimental and is not used.

Existing approaches

Type systems for functional languages is an actively researched field.

The Hindley–Milner type system*[1]* is a simple, but well-studied type system for a very simple language. It is fully decidable, does not require annotations. It is the basis for many real-world type systems.

Functional requirements

Functional requirements

- The system should perform type checking on expression.

Functional requirements

- The system should perform type checking on expression.
- The system should analyze the types and presence of attributes in attribute sets.

Functional requirements

- The system should perform type checking on expression.
- The system should analyze the types and presence of attributes in attribute sets.
- The system should report encountered typechecking.

Functional requirements

- The system should perform type checking on expression.
- The system should analyze the types and presence of attributes in attribute sets.
- The system should report encountered typechecking.
- The system should report the use of undefined terms.

Functional requirements

- The system should perform type checking on expression.
- The system should analyze the types and presence of attributes in attribute sets.
- The system should report encountered typechecking.
- The system should report the use of undefined terms.
- The system should report the resulting types of expressions to the user.

Methods and algorithms

The type system will be heavily influenced by the Hindley–Milner*[1]* type system.

The type system will implement parametric polymorphism to support code that does not require a specific type.

The type system will implement arbitrary-ranked polymorphism*[3]* with deep instantiation to disambiguate types in existing untyped code.

The type system will implement row polymorphism to work with attribute sets.

Used tools

1. Haskell
2. Visual Studio Code
3. Haskell Language Server
4. Haskell Stack

Implementation

The typechecker is implemented in a two-step architecture:

Implementation

The typechecker is implemented in a two-step architecture:

1. Inferring the types of terms and collecting constraints

Implementation

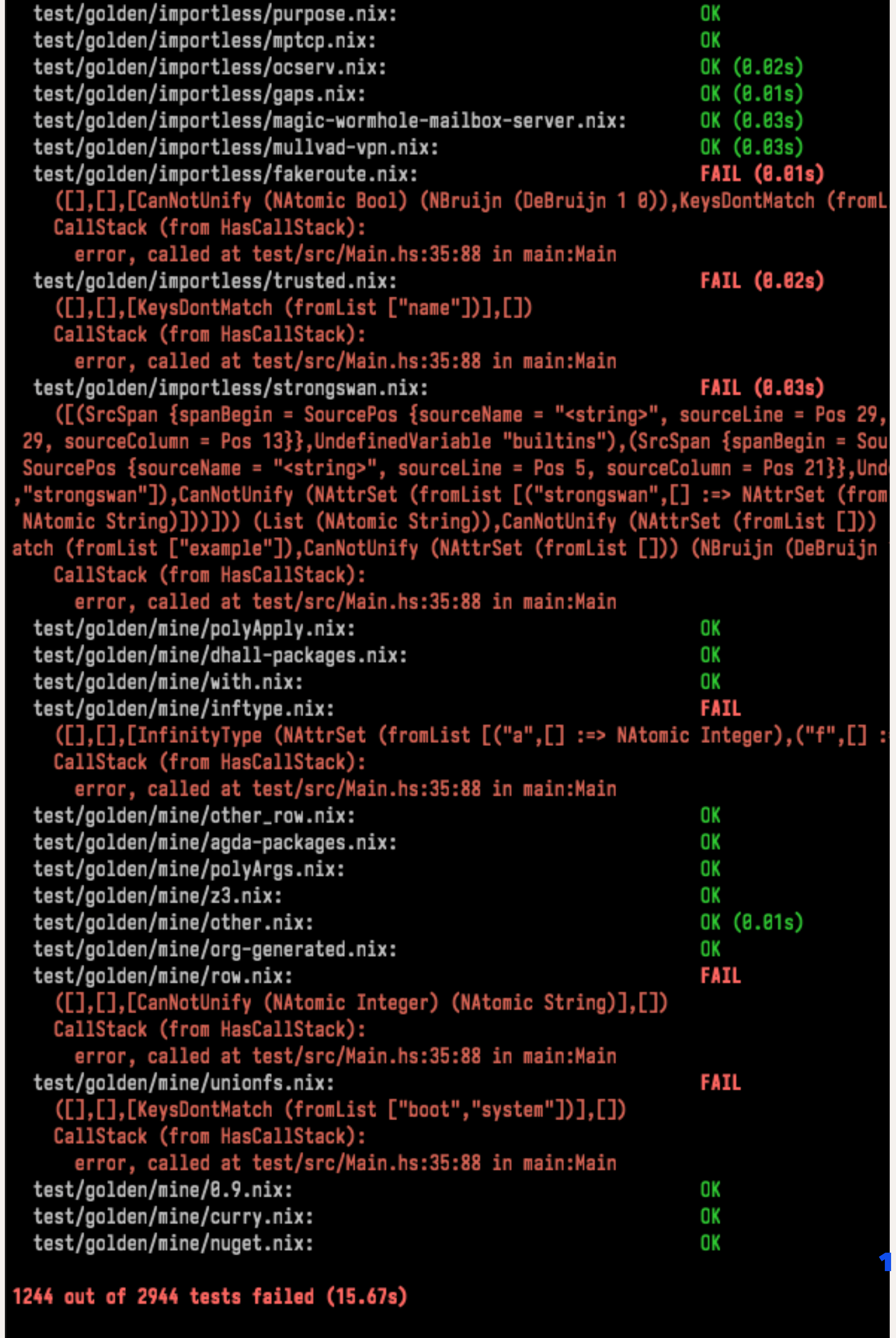
The typechecker is implemented in a two-step architecture:

1. Inferring the types of terms and collecting constraints
2. Solving the collected constraints

Current results

A typechecker executable that successfully typechecks *most* of the selected files from *nixpkgs*.

The typechecker currently has some support for all language constructs.



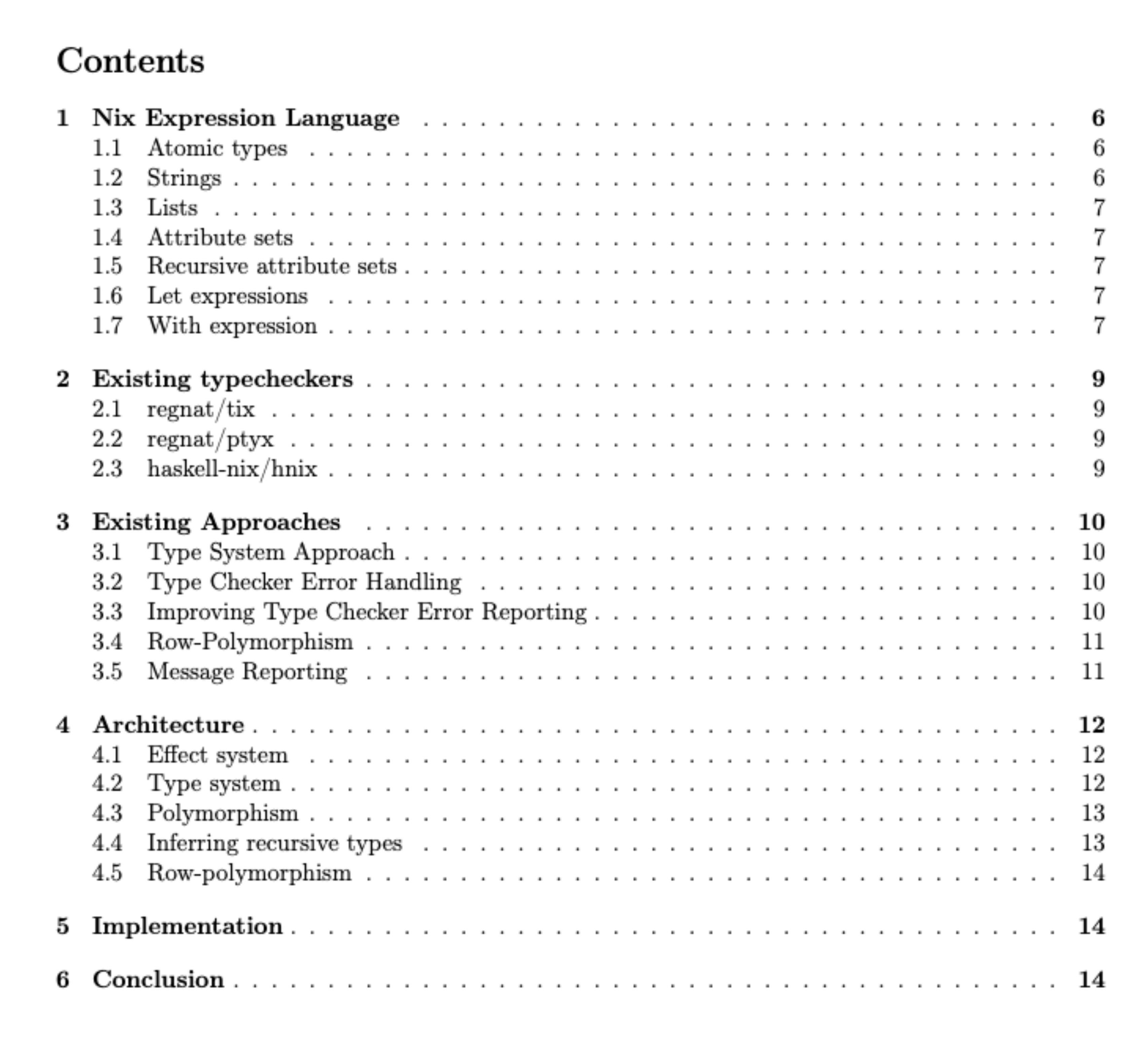
```
  test/golden/importless/purpose.nix:                             OK
  test/golden/importless/mptcp.nix:                               OK
  test/golden/importless/ocserv.nix:                              OK (0.02s)
  test/golden/importless/gaps.nix:                                OK (0.01s)
  test/golden/importless/magic-wormhole-mailbox-server.nix:       OK (0.03s)
  test/golden/importless/mullvad-vpn.nix:                         OK (0.03s)
  test/golden/importless/fakeroute.nix:                           FAIL (0.01s)
    ([],[],[CanNotUnify (NAtomic Bool) (NBruijn (DeBruijn 1 0)),KeysDontMatch (fromL
    CallStack (from HasCallStack):
      error, called at test/src/Main.hs:35:88 in main:Main
  test/golden/importless/trusted.nix:                             FAIL (0.02s)
    ([],[],[KeysDontMatch (fromList ["name"])],[])
    CallStack (from HasCallStack):
      error, called at test/src/Main.hs:35:88 in main:Main
  test/golden/importless/strongswan.nix:                          FAIL (0.03s)
    ([(SrcSpan {spanBegin = SourcePos {sourceName = "<string>", sourceLine = Pos 29,
 29, sourceColumn = Pos 13}},UndefinedVariable "builtins"),(SrcSpan {spanBegin = Sou
 SourcePos {sourceName = "<string>", sourceLine = Pos 5, sourceColumn = Pos 21}},Und
,"strongswan"]),CanNotUnify (NAttrSet (fromList [("strongswan",[] :=> NAttrSet (from
 NAtomic String)]))])) (List (NAtomic String)),CanNotUnify (NAttrSet (fromList []))
atch (fromList ["example"]),CanNotUnify (NAttrSet (fromList [])) (NBruijn (DeBruijn
    CallStack (from HasCallStack):
      error, called at test/src/Main.hs:35:88 in main:Main
  test/golden/mine/polyApply.nix:                                 OK
  test/golden/mine/dhall-packages.nix:                            OK
  test/golden/mine/with.nix:                                      OK
  test/golden/mine/inftype.nix:                                   FAIL
    ([],[],[InfinityType (NAttrSet (fromList [("a",[] :=> NAtomic Integer),("f",[] :
    CallStack (from HasCallStack):
      error, called at test/src/Main.hs:35:88 in main:Main
  test/golden/mine/other_row.nix:                                 OK
  test/golden/mine/agda-packages.nix:                             OK
  test/golden/mine/polyArgs.nix:                                  OK
  test/golden/mine/z3.nix:                                        OK
  test/golden/mine/other.nix:                                     OK (0.01s)
  test/golden/mine/org-generated.nix:                             OK
  test/golden/mine/row.nix:                                       FAIL
    ([],[],[CanNotUnify (NAtomic Integer) (NAtomic String)],[])
    CallStack (from HasCallStack):
      error, called at test/src/Main.hs:35:88 in main:Main
  test/golden/mine/unionfs.nix:                                   FAIL
    ([],[],[KeysDontMatch (fromList ["boot","system"])],[])
    CallStack (from HasCallStack):
      error, called at test/src/Main.hs:35:88 in main:Main
  test/golden/mine/0.9.nix:                                       OK
  test/golden/mine/curry.nix:                                     OK
  test/golden/mine/nuget.nix:                                     OK

1244 out of 2944 tests failed (15.67s)
```

Current results

Paper status

1. Nix Expression Language (80%)
2. Existing typecheckers (100%)
3. Existing approaches (100%)
4. Architecture (20%)
5. Implementation (0%)
6. Conclusion (60%)



Contents

What is left

What is left

- Better type rendering (predicates)

What is left

- Better type rendering (predicates)
- Subsumption

What is left

- Better type rendering (predicates)
- Subsumption
- A lot of minor big fixes in type inference

What is left

- Better type rendering (predicates)
- Subsumption
- A lot of minor big fixes in type inference
- Add type information about built-in functions

Potential future developments

Potential future developments

- Add support for user-supplied type annotations.

Potential future developments

- Add support for user-supplied type annotations.
  - Fork parser

Potential future developments

- Add support for user-supplied type annotations.
  - Fork parser
  - Bidirectional type inference*[4]*

Potential future developments

- Add support for user-supplied type annotations.
  - Fork parser
  - Bidirectional type inference*[4]*
- Type holes

Potential future developments

- Add support for user-supplied type annotations.
    - Fork parser
    - Bidirectional type inference*[4]*
- Type holes
- Language Server Integration

References

1. Damas L., Milner R. Principal type-schemes for functional programs //Proceedings of the 9th ACM SIGPLAN-SIGACT symposium on Principles of programming languages. – 1982. – C. 207-212.

2. Jones S. P. et al. Practical type inference for arbitrary-rank types //Journal of functional programming. – 2007. – T. 17. – №. 1. – C. 1-82. MLA

3. Stuckey P. J., Sulzmann M., Wazny J. Improved inference for checking type annotations //arXiv preprint cs/0507036. – 2005. MLA

Demonstration

Parametric polymorphism

```
let f = x: y: { a = x; b = y + 1;}; in f
∀ α. α -> Integer -> {a = α; b = Integer;}
```

Parametric polymorphism

```
let f = x: y: { a = x; b = y + 1;}; in f
∀ α. α -> Integer -> {a = α; b = Integer;}


let f = x: y: { a = x; b = y + 1;}; in f "hello" 42
{a = String; b = Integer;}
```

Row-polymorphism with *attribute sets*

```
let f = x: x.a; in f
```

```
∀ α β. (α.a = β) => α -> β
```

Row-polymorphism with *attribute sets*

```
let f = x: x.a; in f
```

```
∀ α β. (α.a = β) => α -> β
```

```
let f = x: x.a; in f {a = 1; b = "hello";}
```

```
Integer
```

Row-polymorphism with *attribute sets*

```
let f = x: x.a; in f

∀ α β. (α.a = β) => α -> β


let f = x: x.a; in f {a = "1"; b = "hello";}

String
```

Row-polymorphism with *attribute sets*

```
let f = x: x.a; in f
∀ α β. (α.a = β) => α -> β


let f = x: x.a; in f {b = "hello";}
Error: key "a" NotPresent in { b = String; }
```

Row-polymorphism with *attribute sets*

```
let f = x: x // {a = 69;}; in f
```

```
∀ α β. (α // {a = Integer;} ~ β) => α -> β
```

Row-polymorphism with *attribute sets*

```
let f = x: x // {a = 69;}; in f
∀ α β. (α // {a = Integer;} ~ β) => α -> β
```

```
let f = x: x // {a = 69;}; in f {a = "hello"; b = "world";}
{a = Integer; b = String;}
```

Row-polymorphism with *attribute sets*

```
let f = x: x // {a = 69;}; in f
∀ α β. (α // {a = Integer;} ~ β) => α -> β
```

```
let f = x: x // {a = 69;}; in f {}
{a = Integer;}
```