Process Composition with Typed Unix Pipes

Michael Sippel

2023/09/29

Motivation

Source: Film "The UNIX System: Making Computers More Productive", 1982, Bell Labs

Motivation

Example

Show three least recently accessed directories in PATH

```
echo -n $PATH
| xargs -d: stat -c %X,%n
| sort -n
| head -3
| cut -d, -f2
```

Motivation

Example

Show three least recently accessed directories in PATH

```
echo -n $PATH
| xargs -d: stat -c %X,%n
| sort -n
| head -3
| cut -d, -f2
```

/home/micha/bin:/usr/local/sbin:/usr/local/bin:/usr/bin:/bin

Motivation

Example

Show three least recently accessed directories in PATH

```
echo -n $PATH
| xargs -d: stat -c %X,%
| sort -n
| head -3
| cut -d, -f2
```

```
/home/micha/bin:/usr/local/sbin:/usr/local/
```

```
1695801132,/home/micha/bin
1695803447,/usr/local/sbin
1695814536,/usr/local/bin
1695802139,/usr/bin
1695802144,/bin
```

Motivation

Example

Show three least recently accessed directories in PATH

```
echo -n $PATH
| xargs -d: stat -c %X,%
| sort -n
| head -3
| cut -d, -f2
```

```
/home/micha/bin:/usr/local/sbin:/usr/local/
```

```
1695801132,/home/micha/bin
```

```
1695801132,/home/micha/bin
1695802139,/usr/bin
1695802144,/bin
1695803447,/usr/local/sbin
1695814536,/usr/local/bin
```

Motivation

Example

Show three least recently accessed directories in PATH

```
echo -n $PATH
| xargs -d: stat -c %X,%
| sort -n
| head -3
| cut -d, -f2
```

```
/home/micha/bin:/usr/local/sbin:/usr/local/
```

```
1695801132,/home/micha/bin
```

```
1695801132,/home/micha/bin
```

```
1695801132,/home/micha/bin
1695802139,/usr/bin
1695802144,/bin
```

Motivation

Example

Show three least recently accessed directories in PATH

```
echo -n $PATH
| xargs -d: stat -c %X,%
| sort -n
| head -3
| cut -d, -f2
```

```
/home/micha/bin:/usr/local/sbin:/usr/local/
```

```
1695801132,/home/micha/bin
```

```
1695801132,/home/micha/bin
```

```
1695801132,/home/micha/bin
```

```
/home/micha/bin
/usr/bin
/bin
```

Consequences of invalid compositions

Problem Analysis

Example

Show three least recently accessed directories in PATH

```
echo -n $PATH
| xargs -d: stat -c %X,%n
| sort -n
| head -3
| cut -d, -f2
```

Consequences of invalid compositions

Problem Analysis

Example

Show three least recently accessed directories in PATH

```
echo -n $PATH
| xargs ~~-d:~~ stat -c %X,%n
| sort -n
| head -3
| cut -d, -f2
```

Consequences of invalid compositions

Problem Analysis

Example

Show three least recently accessed directories in PATH

```
echo -n $PATH
| xargs ~~-d:~~ stat -c %X,%n
| sort -n
| head -3
| cut -d, -f2
```

```
stat: cannot statx '/home/micha/bin:/usr/local/sbin:
/usr/local/bin:/usr/bin:/bin': No such file or directory
```

Consequences of invalid compositions

Problem Analysis

- runtime error (parser error / invalid input value)
  - maybe helpful, maybe not
  - mistake might propagate before causing error
- mistake unnoticed → invalid output data / unwanted operation

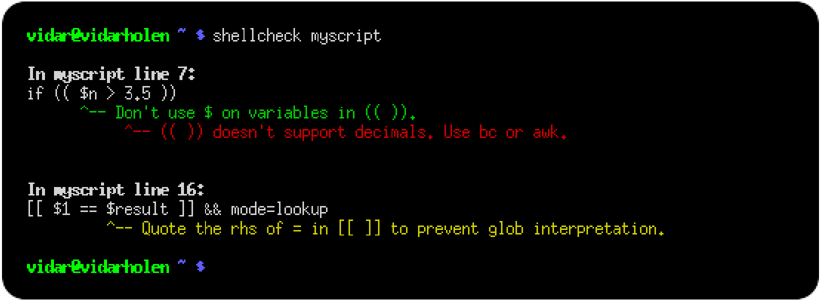
Related Work

On one hand POSIX and some strict implementations e.g. dash. alternative shell implementations...

- Zsh
- Fish
- PowerShell
- NuShell
- Elvish
- for 30years, **bash** remained dominant

Related Work

- ShellCheck



```
vidar@vidarholen ~ $ shellcheck myscript

In myscript line 7:
if (( $n > 3.5 ))
      ^-- Don't use $ on variables in (( )).
           ^-- (( )) doesn't support decimals. Use bc or awk.


In myscript line 16:
[[ $1 == $result ]] && mode=lookup
         ^-- Quote the rhs of = in [[ ]] to prevent glob interpretation.

vidar@vidarholen ~ $
```

Source: `https://github.com/koalaman/shellcheck`

Requirements

Concept

- static typechecking
- applicable to existing utility programs

Requirements

Concept

Example

Show three least recently accessed directories in PATH

```
echo -n $PATH

| xargs ~~-d:~~ stat -c %X,%n

| sort -n

| head -3

| cut -d, -f2
```

Requirements

Concept

Example

Show three least recently accessed directories in PATH

stdin-type: -

stdout-type: $\langle \mathit{Seq} \quad \mathit{Path} \rangle$

```
echo -n $PATH
| xargs ~~-d:~~ stat -c %X,%n
| sort -n
| head -3
| cut -d, -f2
```

Requirements

Concept

Example

Show three least recently accessed directories in PATH

```
echo -n $PATH
```

stdin-type: -

stdout-type: $\langle \mathit{Seq} \quad \mathit{Path} \rangle$

```
| xargs ~~-d:~~ stat -c %X,%n
```

stdin-type: $\langle \mathit{Seq} \quad \mathit{Path} \rangle$

stdout-type: $\langle \mathit{Seq} \quad \mathit{Date}, \mathit{Path} \rangle$

```
| sort -n

| head -3

| cut -d, -f2
```

Ladder Types

Concept

- **Intuition**: capture 'represented-as' relation of layered encodings.

Ladder Types

Concept

- **Intuition**: capture 'represented-as' relation of layered encodings.
- **Formally**: new type-constructor $T_1 \sim T_2$

Ladder Types

Concept

- **Intuition**: capture 'represented-as' relation of layered encodings.
- **Formally**: new type-constructor $T_1 \sim T_2$

Example

$$
\begin{array}{l}
\langle \text{Seq} \quad \text{Path} \rangle \\
\sim \langle \text{Seq} \quad \langle \text{Seq} \quad \text{Char} \rangle \rangle \\
\sim \langle \text{SepSeq} \quad \text{Char} \quad \text{':'} \rangle \\
\sim \langle \text{Seq} \quad \text{Char} \rangle
\end{array}
$$

Ladder Types

Concept

Example

```
echo -n $PATH
| xargs ~~-d:~~ stat -c %X,%n
| sort -n
| head -3
| cut -d, -f2
```

Ladder Types

Concept

Example

```
echo -n $PATH

| xargs ~~-d:~~ stat -c %X,%n

| sort -n

| head -3

| cut -d, -f2
```

stdin-type: -

stdout-type: $\langle \textbf{Seq} \quad \textbf{Path} \rangle$

$\sim \langle \textbf{Seq} \quad \langle \textbf{Seq} \quad \textbf{Char} \rangle \rangle$

$\sim \langle \textbf{SepSeq} \quad \textbf{Char} \quad \textbf{':'} \rangle$

$\sim \langle \textbf{Seq} \quad \textbf{Char} \rangle$

Ladder Types

Concept

Example

`echo -n $PATH`

`| xargs` ~~`-d:`~~ `stat -c %X,%n`

`| sort -n`

`| head -3`

stdin-type: -

stdout-type: $\langle \mathbf{Seq} \quad \mathbf{Path} \rangle$

$\sim \langle \mathbf{Seq} \quad \langle \mathbf{Seq} \quad \mathbf{Char} \rangle \rangle$

$\sim \langle \mathbf{SepSeq} \quad \mathbf{Char} \quad \mathbf{':'} \rangle$

$\sim \langle \mathbf{Seq} \quad \mathbf{Char} \rangle$

stdin-type: $\langle \mathbf{Seq} \quad \mathbf{Path} \rangle$

$\sim \langle \mathbf{Seq} \quad \langle \mathbf{Seq} \quad \mathbf{Char} \rangle \rangle$

$\sim \langle \mathbf{SepSeq} \quad \mathbf{Char} \quad \mathbf{'\backslash n'} \rangle$

$\sim \langle \mathbf{Seq} \quad \mathbf{Char} \rangle$

stdout-type: $\langle Seq \quad Date, Path \rangle$

Ladder Types

Concept

Example

```
echo -n $PATH

| xargs -d: stat -c %X,%n

| sort -n

| head -3
```

stdin-type: -

stdout-type: $\langle \textbf{Seq} \quad \textbf{Path} \rangle$

$\sim \langle \textbf{Seq} \quad \langle \textbf{Seq} \quad \textbf{Char} \rangle \rangle$

$\sim \langle \textbf{SepSeq} \quad \textbf{Char} \quad \text{':'} \rangle$

$\sim \langle \textbf{Seq} \quad \textbf{Char} \rangle$

stdin-type: $\langle \textbf{Seq} \quad \textbf{Path} \rangle$

$\sim \langle \textbf{Seq} \quad \langle \textbf{Seq} \quad \textbf{Char} \rangle \rangle$

$\sim \langle \textbf{SepSeq} \quad \textbf{Char} \quad \text{':'} \rangle$

$\sim \langle \textbf{Seq} \quad \textbf{Char} \rangle$

stdout-type: $\langle Seq \quad Date, Path \rangle$

Evaluation + Demo

1. cat foo | ~~xargs~~ cp bar
2. printf '%s: %s\n' foo ~~bar~~
3. echo $PATH | xargs ~~-d :~~ stat -c %x
4. find | xargs stat -c ~~%Y~~%y | sort -n | head -3
5. find | xargs stat -c %Y | sort ~~-n~~ | head -3
6. ls -l *.log | xargs rm
7. date ~~+%S~~+%s | xargs expr 2 +
8. find . -printf '%Tb\n' | sort ~~-M~~-m | uniq

TC	Runtime Error	Caught by Ladder Typing?	Caught by ShellCheck?
1	miss. operand	no	yes
2	–	no	yes
3	file not found	yes	no
4	–	yes	no
5	–	no	no
6	invalid option	yes	no
7	–	no	no
8	–	yes	no

Summary

Problem:

- combination of incompatible processes
- invalid data formats in pipelines
- multiple possible representations of same concept

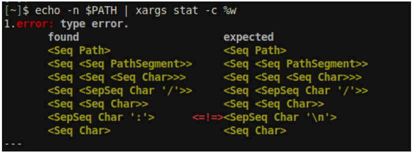
Goal:

- static typechecking
- preserve functionality of POSIX shell & utilities

Ladder-Types:

Date
$\sim \langle \text{TimeSince} \quad \text{UnixEpoch} \rangle$
$\sim \langle \text{Duration} \quad \text{Seconds} \rangle$
$\sim \mathbb{N}$
$\sim \langle \text{PosInt} \quad 10 \quad \text{BigEndian} \rangle$
$\sim \langle \text{Seq} \quad \langle \text{Digit} \quad 10 \rangle \rangle$
$\sim \langle \text{Seq} \quad \text{Ascii} \rangle$
$\sim \langle \text{Seq} \quad \text{Byte} \rangle$

Result:



```
[~]$ echo -n $PATH | xargs stat -c %w
1.error: type error.
       found                        expected
       <Seq Path>                   <Seq Path>
       <Seq <Seq PathSegment>>      <Seq <Seq PathSegment>>
       <Seq <Seq <Seq Char>>>       <Seq <Seq <Seq Char>>>
       <Seq <SepSeq Char '/'>>      <Seq <SepSeq Char '/'>>
       <Seq <Seq Char>>             <Seq <Seq Char>>
       <SepSeq Char ':'>       <=!=><SepSeq Char '\n'>
       <Seq Char>                   <Seq Char>
---
```

- improved robustness & debugability

Ladder Types

Concept

Definition (Ladder-Type)

Given a set of *base-types* B, the set of *ladder-types*[1] denoted $T(B)$ is inductively defined to contain terms of the following form:

- τ (Atomic Type) where $\tau \in B$
- $\langle \sigma \quad \tau \rangle$ (Type Application) with σ and τ types
- $\tau_1 \sim \tau_2$ (Ladder Type) with types τ_1 and τ_2
- $\tau_1 \rightarrow \tau_2$ (Function Type) where τ_1 and τ_2 are types

---

[1] restricted to *monotypes*, i.e. no type-variables for now

Ladder Types

Concept

Definition (Type Equivalence)

The relation $\equiv \subseteq T(B) \times T(B)$ is defined to be the reflexive, transitive and symmetric closure over the following equation which defines distributivity of $\sim$ over $\langle \dots \rangle$:

$$\langle \sigma \quad \tau \sim \tau' \rangle \equiv \langle \sigma \quad \tau \rangle \sim \langle \sigma \quad \tau' \rangle$$

Definition (Flatness)

A type term τ is *flat*, if none of its subterms is a ladder type.

Definition (Ladder Normal Form)

A type term τ is in *Ladder Normal Form (LNF)* if either τ is flat or τ is a ladder type $\tau = \tau_1 \sim \tau_2$ where τ_1 is flat and τ_2 is in LNF.

Ladder Types

Concept

Example

Consider the following two *equivalent* types:

- $\langle \text{Seq} \quad \langle \text{Digit} \quad 10 \rangle \rangle \sim \langle \text{Seq} \quad \text{Char} \rangle$ is in LNF
- $\langle \text{Seq} \quad \langle \text{Digit} \quad 10 \rangle \sim \text{Char} \rangle$ is not, since there occurs a ladder-type constructor inside a parameter application. LNF can be reached by applying $\rightarrow_D$ once.

Ladder Types

Concept

Example

Consider the following two *equivalent* types:

- $\langle \text{Seq} \quad \langle \text{Digit} \quad 10 \rangle\rangle \sim \langle \text{Seq} \quad \text{Char}\rangle$ is in LNF
- $\langle \text{Seq} \quad \langle \text{Digit} \quad 10\rangle \sim \text{Char}\rangle$ is not, since there occurs a ladder-type constructor inside a parameter application. LNF can be reached by applying $\rightarrow_D$ once.

Corollary

It follows from lemma **??***, that exhaustive application of the rewrite rule $\rightarrow_D$ yields the unique ladder-normal-form of the input term in every case. Thus, without loss of generality we can assume that all types are in normal form.*

Ladder Types

Concept

$\sim$ is distributive over $\langle \ldots \rangle$

Example

Consider the following two *equivalent* types:

- $\langle \mathsf{Seq} \quad \langle \mathsf{Digit} \quad 10 \rangle \rangle \sim \langle \mathsf{Seq} \quad \mathsf{Char} \rangle$ is in LNF
- $\langle \mathsf{Seq} \quad \langle \mathsf{Digit} \quad 10 \rangle \sim \mathsf{Char} \rangle$ is not, since there occurs a ladder-type constructor inside a parameter application.

Typed Process Invocations

Concept

Example

`date +%s` has **stdout**-type

- $$\begin{aligned} & \quad \text{Date} \\ \sim & \ \langle \text{TimeSince} \quad \text{UnixEpoch} \rangle \\ \sim & \ \langle \text{Duration} \quad \text{Seconds} \rangle \\ \sim & \ \mathbb{N} \\ \sim & \ \langle \text{PosInt} \quad 10 \quad \text{BigEndian} \rangle \\ \sim & \ \langle \text{Seq} \quad \langle \text{Digit} \quad 10 \rangle \sim \text{Ascii} \sim \text{Byte} \rangle \end{aligned}$$

Typed Process Invocations

Concept

Example

`date +%s` has **stdout**-type

- $$\begin{aligned} & \text{Date} \\ \sim\ & \langle \text{TimeSince}\quad \text{UnixEpoch} \rangle \\ \sim\ & \langle \text{Duration}\quad \text{Seconds} \rangle \\ \sim\ & \mathbb{N} \\ \sim\ & \langle \text{PosInt}\quad 10\quad \text{BigEndian} \rangle \\ \sim\ & \langle \text{Seq}\quad \langle \text{Digit}\quad 10 \rangle \sim \text{Ascii} \sim \text{Byte} \rangle \end{aligned}$$

... in Ladder-Normal Form:

- $$\begin{aligned} & \text{Date} \\ \sim\ & \langle \text{TimeSince}\quad \text{UnixEpoch} \rangle \\ \sim\ & \langle \text{Duration}\quad \text{Seconds} \rangle \\ \sim\ & \mathbb{N} \\ \sim\ & \langle \text{PosInt}\quad 10\quad \text{BigEndian} \rangle \\ \sim\ & \langle \text{Seq}\quad \langle \text{Digit}\quad 10 \rangle \rangle \\ \sim\ & \langle \text{Seq}\quad \text{Ascii} \rangle \\ \sim\ & \langle \text{Seq}\quad \text{Byte} \rangle \end{aligned}$$

Typed Process Invocations

Concept

- assign ladder-type per filedescriptor
- (alternatively, fd may remain untyped)
- types may depend on process arguments & environment

Type Inference

Concept

Let `A`,`B` be process invocations...

- For a pipeline A | B to be valid, the stdout-type of A must be compatible with stdin-type of B,
  i.e. A's stdout-type must be a subtype of B's stdin-type,
- A | B inherits stdin-type from A
- A | B inherits stdout-type from B

Typecheck Algorithm

Concept

Example

$$P_1 \mid P_2 \mid P_3 \mid \ldots$$

- iterate over pipeline
- check subtyping-relation of stdout-type and stdin-type
- abort if stdout-type is no subtype of stdin-type

Typing Assertions

Implementation

- infinitely many process invocations
- group by regexp (not ideal, but easy implementation)
- for each regexp-command-pattern define type per filedescriptor

.zshrc

Implementation

```
preexec() {
  ~/syntaxAlchemist/target/release/shell \
      --check-expr="$1"
}
```