Process Composition with Typed Unix Pipes

Michael Sippel

2023/09/29
Motivation

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
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,%n
| sort -n
| head -3
| cut -d, -f2

/home/micha/bin:/usr/local/sbin:/usr/local/bin:
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,%n
| sort -n
| head -3
| cut -d, -f2
```

```
/home/micha/bin:/usr/local/sbin:/usr/local/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,%n
| sort -n
| head -3
| cut -d, -f2

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

```bash
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

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 $\rightarrow$ invalid output data / unwanted operation
On one hand POSIX and some strict implementations e.g. dash, alternative shell implementations...

- Zsh
- Fish
- PowerShell
- NuShell
- Elvish

- for 30 years, **bash** remained dominant
Related Work

- **ShellCheck**

```bash
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
Example

Show three least recently accessed directories in PATH

```bash
echo -n $PATH
```

```
stdin-type: -
```

```
stdout-type: ⟨Seq Path⟩
```

```bash
|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
```

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

```
| 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**

\[
\langle \text{Seq Path} \rangle \\
\sim \langle \text{Seq} \langle \text{Seq Char} \rangle \rangle \\
\sim \langle \text{SepSeq Char '}' \rangle \\
\sim \langle \text{Seq Char} \rangle
\]
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: ⟨Seq Path⟩
~ ⟨Seq ⟨Seq Char⟩⟩
~ ⟨SepSeq Char ':'⟩
~ ⟨Seq Char⟩
Ladder Types

Concept

Example

eecho -n $PATH

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

| sort -n

| head -3

stdin-type: -
stdout-type: ⟨Seq Path⟩
~ ⟨Seq ⟨Seq Char⟩⟩
~ ⟨SepSeq Char ‘:’⟩
~ ⟨Seq Char⟩

stdin-type: ⟨Seq Path⟩
~ ⟨Seq ⟨Seq Char⟩⟩
~ ⟨SepSeq Char ‘\n’⟩
~ ⟨Seq Char⟩

stdout-type: ⟨Seq Date, Path⟩
Ladder Types

Concept

Example

```
| `echo -n $PATH`

```

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

```

```
| `sort -n`

```

```
| `head -3`

```

stdin-type: -
stdout-type: ⟨Seq Path⟩
~ ⟨Seq ⟨Seq Char⟩⟩
~ ⟨SepSeq Char ’:’⟩
~ ⟨Seq Char⟩
stdin-type: ⟨Seq Path⟩
~ ⟨Seq ⟨Seq Char⟩⟩
~ ⟨SepSeq Char ’:’⟩
~ ⟨Seq Char⟩
stdout-type: ⟨Seq Date, Path⟩
Evaluation + Demo

1. `cat foo | xargs cp bar`
2. `printf '%s: %s
' foo bar`
3. `echo $PATH | xargs 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`

<table>
<thead>
<tr>
<th>TC</th>
<th>Runtime Error</th>
<th>Caught by Ladder Typing?</th>
<th>Caught by ShellCheck?</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>miss. operand</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>2</td>
<td>–</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>3</td>
<td>file not found</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>4</td>
<td>–</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>5</td>
<td>–</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>6</td>
<td>invalid option</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>7</td>
<td>–</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>8</td>
<td>–</td>
<td>yes</td>
<td>no</td>
</tr>
</tbody>
</table>
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
~ ⟨TimeSince UnixEpoch⟩
~ ⟨Duration Seconds⟩
~ ⟨N⟩
~ ⟨PosInt 10 BigEndian⟩
~ ⟨Seq ⟨Digit 10⟩⟩
~ ⟨Seq Ascii⟩
~ ⟨Seq Byte⟩

Result:

- 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:

- $\tau$ (Atomic Type) where $\tau \in B$
- $\langle \sigma \quad \tau \rangle$ (Type Application) with $\sigma$ and $\tau$ types
- $\tau_1 \sim \tau_2$ (Ladder Type) with types $\tau_1$ and $\tau_2$
- $\tau_1 \rightarrow \tau_2$ (Function Type) where $\tau_1$ and $\tau_2$ 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 $\langle \ldots \rangle$:
$\langle \sigma \; \tau \sim \tau' \rangle \equiv \langle \sigma \; \tau \rangle \sim \langle \sigma \; \tau' \rangle$

Definition (Flatness)
A type term $\tau$ is flat, if none of its subterms is a ladder type.

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

Consider the following two *equivalent* types:

- \( \langle \text{Seq} \; \langle \text{Digit} \; 10 \rangle \rangle \sim \langle \text{Seq} \; \text{Char} \rangle \) is in LNF
- \( \langle \text{Seq} \; \langle \text{Digit} \; 10 \rangle \; \sim \; \text{Char} \rangle \) 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:

- $\langle \text{Seq} \, \langle \text{Digit} \, 10 \rangle \rangle \sim \langle \text{Seq} \, \text{Char} \rangle$ is in LNF
- $\langle \text{Seq} \, \langle \text{Digit} \, 10 \rangle \sim \text{Char} \rangle$ 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:

- \( \langle \text{Seq} \ \langle \text{Digit} \ 10 \rangle \rangle \sim \langle \text{Seq} \ \text{Char} \rangle \) is in LNF
- \( \langle \text{Seq} \ \langle \text{Digit} \ 10 \rangle \sim \text{Char} \rangle \) is not, since there occurs a ladder-type constructor inside a parameter application.
Typed Process Invocations

Concept

Example

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

- Date
  - ∼ ⟨TimeSince UnixEpoch⟩
  - ∼ ⟨Duration Seconds⟩
  - ∼ N
  - ∼ ⟨PosInt 10 BigEndian⟩
  - ∼ ⟨Seq ⟨Digit 10⟩ ∼ Ascii ∼ Byte⟩
Typed Process Invocations

Concept

Example

date +%s has stdout-type

▶ Date
  ~ (TimeSince UnixEpoch)
  ~ (Duration Seconds)
  ~ N
  ~ (PosInt 10 BigEndian)
  ~ (Seq $\langle$Digit 10$\rangle$ Ascii Byte)

... in Ladder-Normal Form:

▶ Date
  ~ (TimeSince UnixEpoch)
  ~ (Duration Seconds)
  ~ N
  ~ (PosInt 10 BigEndian)
  ~ (Seq $\langle$Digit 10$\rangle$)
  ~ (Seq Ascii)
  ~ (Seq Byte)
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
preexec() {
    ~/syntaxAlchemist/target/release/shell \ 
    --check-expr="$1"
}