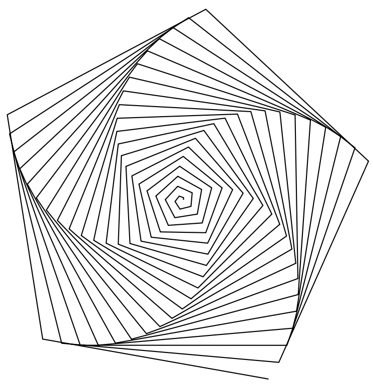
DeTEXtive: TEX Mode Analyzer

Jaeho Lee

December 21, 2022



Contents

1 Introduction

There has been—and is—a constant effort to "modernize" TeX and its extension LaTeX. Undisciplined users often fall into a rabbit hole of cryptic errors, due to the very fact that it is a turing-complete macro-expansion language that makes it super flexible and extensible. The following is a non-exhaustive list of some widely used static linters and/or analyzers around the language:

- TexLab: basic LSP features like `textDocument.definition`;
- ChkTeX: linter;
- TeXcount: count words and numbers.

However, these tools resort to ad hoc heuristics and pattern matching; we can do more:

1. mode detection,
2. typing, and
3. document analysis, e.g., pre-detecting overflows, sound prediction of page numbers.

This document describes DeTeXtive, a mode analyzer on (plain) TeX documents, based on abstract interpretation[1]. It collects modes in a label-wise manner so that a TeXnician can identify which part of his/her document is in which mode. It futher addresses possible mode errors in a faulty TeX document.

1.1 Modes in TeX

But wait, what is a *mode*? There are six different modes in TeX[2]:

1. vertical mode (the main vertical list of the pages),
2. internal vertical mode (a vertical list for a vbox),
3. horizontal mode (a horizontal list for a paragraph),
4. restricted horizontal mode (a horizontal list for an hbox),
5. math mode (in a horizontal list),
6. display math mode (interrupts the current paragraph).

TeX works in terms of boxes and glues to compose a document, and it does so by digesting tokens in a sophisticated manner. In the process, TeX is always in one of the aforementioned six modes.

It is always the vertical mode when TeX begins processing the tokens. Then it enters the horizontal mode to construct a paragraph, and exits back to the vertical mode when the paragraph ends. In a simple document without equations and nested structures, this is pretty much how TeX works—but other modes are visited when typesetting complex documents. One should consult the TeXbook[2] for the authoritative guide, especially the chapters 24, 25, and 26.

Unfortunately, the modes are implicit in a document, and a careless typesetter easily get overwhelmed by errors like

```
You can't use `macro parameter character #' in restricted horizontal mode.
```

This often leads to an unexpected result and might even prevent a document from being generated: `! Emergency stop.` To make matters worse, TEX tries to fix these errors, and online editors like Overleaf defaults to accepting these implicit fixes.

DeTEXtive aims to lift this burden from a typsetter by statically analyzing modes of each part of a document. It is defined over a small core of TEX called $\tau\epsilon\chi$, described below.

1.2 Our Target Language $\tau\epsilon\chi$

1.2.1 Some Restrictions

$\tau\epsilon\chi$ has three modes: horizontal, vertical, and math (unlike its brother TEX, which have six modes). Moreover, $\tau\epsilon\chi$ assumes its target TEX code have its control sequences[1] wrapped with `\expandafter`'s, e.g., `\expandafter\somecs\expandafter`. This restriction is to make control sequence expansions behave like a function call, which is necessary to write non-tail recursive calls that remember arguments in TEX.

We do not allow nested macro defintions as well, and all control sequences accept zero or one arguments.

To give a taste of what $\tau\epsilon\chi$ can express, some structures are listed in the following subsections.

1.2.2 Conditional

$\tau\epsilon\chi$ can express conditionals in TEX:

```
\def\condtest#1{%
  #1 is %
  \ifnum#1>0 positive%
  \else\ifnum#1<0 negative%
    \else zero%
    \fi
  \fi%
}
```

"`\condtest{42}, \condtest{-3}, and \condtest0.`" yields 42 is positive, -3 is negative, and 0 is zero.

More importantly, we have `\ifvmode`, `\ifhmode`, and `\ifmmode` to check a mode.

1.2.3 Loop

$\tau\epsilon\chi$ can also express loops in TEX:

```
\newcount\n
\def\johnny#1{%
  \n=0
  \loop\ifnum\n<#1
    \advance\n by1
    \noindent\number\n. \johnnytxt\endgraf%
  \repeat%
```

[1]We use the jargon *control sequence* and *macro* interchangeably.

```
}
\def\johnnytxt{\texttt{All work and no play makes Jack a dull boy}}
\johnny{9}
```

```
1. All work and no play makes Jack a dull boy
2. All work and no play makes Jack a dull boy
3. All work and no play makes Jack a dull boy
4. All work and no play makes Jack a dull boy
5. All work and no play makes Jack a dull boy
6. All work and no play makes Jack a dull boy
7. All work and no play makes Jack a dull boy
8. All work and no play makes Jack a dull boy
9. All work and no play makes Jack a dull boy
```

Note that `\loop` is defined tail-recursively as follows:

```
\def\loop#1\repeat{\def\body{#1}\iterate}
\def\iterate{\body\let\Next=\iterate\else\let\Next=\relax\fi\Next}
```

$\tau\epsilon\chi$ includes `loop` as a "primitive."

1.2.4 Control Sequences

TeX can behave as if it "remember"s macro arguments, so for the sake of our semantics.

```
\def\dec#1{%
  \ifnum#1=0
    .%
  \else
    #1%
    \expandafter\dec\expandafter{\number\numexpr#1-1\relax}%
    #1%
  \fi%
}%
\dec{5}
```

10987654321.12345678910

Moreover, control sequence names be stored:

```
\def\call#1{#1}
\expandafter\call\expandafter\dec{10}
```

10987654321.12345678910

The "environment" that only carries an (optional) argument, and all other variables are global.

```
\newcount\n \newcount\x
\def\sumup#1{%
  \x=#1
  \ifnum\x=0
    \n=0
  \else
```

```
    \advance\x by-1
    \expandafter\sumup\expandafter\x
    \advance\x by1
    \advance\n by\x
    \number\n\
  \fi%
}
\sumup{100}
```

1 3 6 10 15 21 28 36 45 55 66 78 91 105 120 136 153 171 190 210 231 253 276 300 325 351 378 406 435 465 496 528 561 595 630 666 703 741 780 820 861 903 946 990 1035 1081 1128 1176 1225 1275 1326 1378 1431 1485 1540 1596 1653 1711 1770 1830 1891 1953 2016 2080 2145 2211 2278 2346 2415 2485 2556 2628 2701 2775 2850 2926 3003 3081 3160 3240 3321 3403 3486 3570 3655 3741 3828 3916 4005 4095 4186 4278 4371 4465 4560 4656 4753 4851 4950 5050

2 Syntax of τϵχ

command	C	::=	l^+	*letter + α (11 (`[a-zA-Z]`), 12)*
		|	`^` `_`	*super/subscript (7 (`^`), 8 (`_`))*
		|	`num` E	*typeset number*
		|	x `+=` E	*numeric addition*
		|	x `=` E	*assignment*
		|	`hbox` C | `vbox` C	*box (must end with* `unbox`*)*
		|	`math` C	*math (must end with* `unbox`*)*
		|	`unbox`	*unnest a box*
		|	`hvswitch`	→↓ (`\par`, `\vskip`, `\hrule`, `\vfil`, *etc.*)
		|	`vhswitch`	↓→ (`\(no)indent`, `\vrule`, *etc.*)
		|	c E?	*control sequence application*
		|	`ret`$_c$	*return (tagged with a corresponding c)*
		|	`if` P C (`else` C)?	`\ifhmode`, `\ifvmode`, `\ifmmode`, `\ifnum`
		|	`loop` P C	*loop*
		|	C C	*sequence*
expression	E	::=	x	*variable*
		|	n	*integer*
		|	$E \oplus E$	*arithmetic* ($\oplus \in \{+, \times\}$)
		|	c	*control sequence name*
control seq def	F	::=	`def` c W? C	`\def\c`
predicate	P	::=	$\mathfrak{h}$ | $\mathfrak{v}$ | $\mathfrak{m}$	*mode check*
		|	$E \bowtie E$	*comparision* ($\bowtie \in \{=, <, >\}$)
document	D	::=	F^* C	*a list of defs followed by a command*

Note that ?, $^+$, and * are taken from regular expression denotations, i.e., E? and (`else` …)? is optional, l^+ is one or more letters, and F^* is zero or more F's. Numbers like 11, 12, 7, 8 refer to the category codes of tokens in TeX and are written for reference. It is an error to use letters with category codes 7 and 8 in a non-math environment.

l^+ implicitly changes a mode to horizontal when in a vertical mode, and keeps the mode in a math mode. `num` E is used to "typeset" an expression E to a document; In the mode's perspective, it implicitly makes a transition to a horizontal mode when in a vertical mode, just like l^+.

Box commands like `hbox`, `vbox`, and `math` nests "modes" and must end with `unbox`, analogous to a control sequence definition ending with a `ret`. Explicitly issuing `hvswitch` or `vhswitch` should change a mode without nesting.

A variable can store a number or a control sequence name—in TEX's world, the first is a counter and the latter is a simple macro assignment via `\def\newx{\oldx}`.

A document consists of a list of control sequence defintions followed by a command.

Finally, we assume that target programs to be "type"-correct, i.e., they do not contain commands like `x += csname` yet may contain mode-erratic commands like `hbox ^`.

3 Semantics of τϵχ

The concrete and abstract semantics are defined à la section 8.2 For a Language with Functions and Recursive Calls of [1].

3.1 Concrete Transitional Semantics

3.1.1 Concrete Semantic Domains

$\langle \ell, d, t, m, \sigma, \kappa, \phi, f \rangle$	$\in$	$\mathbb{S}$	$=$	$\mathbb{L} \times \mathbb{D} \times \mathbb{T} \times \mathbb{M} \times \mathbb{E} \times \mathbb{K} \times \mathbb{I} \times \mathbb{F}$	
d	$\in$	$\mathbb{D}$	$=$	$\{\mathfrak{h}, \mathfrak{v}, \mathfrak{m}\}$	modes
t	$\in$	$\mathbb{T}$	$=$	$\mathbb{D}^*$	nested modes
v	$\in$	$\mathbb{V}$	$=$	$\mathbb{Z} \cup \mathbb{C}$	values
m	$\in$	$\mathbb{M}$	$=$	$\mathbb{X} \times \mathbb{I} \to \mathbb{V}$	memories
σ	$\in$	$\mathbb{E}$	$=$	$\mathbb{X} \to \mathbb{I}$	environments
κ	$\in$	$\mathbb{K}$	$=$	$(\mathbb{L} \times \mathbb{E})^*$	continuations
ϕ	$\in$	$\mathbb{I}$			instances
f	$\in$	$\mathbb{F}$	$=$	$\{\checkmark\} \cup \mathbb{L} \times \mathbb{D} \times \mathbb{D}$	implicit fix
x	$\in$	$\mathbb{X}$	$=$	$\mathbb{A} \cup \mathbb{G}$	variables
ℓ	$\in$	$\mathbb{L}$			labels
c	$\in$	$\mathbb{C}$			control sequence names
a_c	$\in$	$\mathbb{A}$	$=$	$\{\bullet\} \cup \mathbb{C}$	control sequence arguments
		$\mathbb{G}$			global variables

In the above, we used a notation $S^* = \bigcup_{i \geq 0} S^i$ to represent stacks.

There are three elements in modes $\mathbb{D} = \{\mathfrak{h}, \mathfrak{v}, \mathfrak{m}\}$. TEX makes implicit fixes, and we capture it in τϵχ via $f = (\ell, d, d') \in \mathbb{F}$, which means "at l, it is expected to be d' but was d;" $f = \checkmark$ means there are no errors (yet).

Environments are from variables to values, but only global variables and function arguments are present. When ϕ_I is the initial instance, $\sigma(x) = \phi_I$ if $x \in \mathbb{G}$. Note that $\mathbb{A} = \{\bullet\} \cup \mathbb{C}$—this means that there is only one argument for a control sequence ($\mathbb{C}$) or none ($\bullet$).

The initial state should be $I = \{\langle \ell_0, \mathfrak{v}, \varnothing_{\mathbb{T}}, \varnothing_{\mathbb{M}}, \varnothing_{\mathbb{E}}, \varnothing_{\mathbb{K}}, \phi_0, \checkmark \rangle\}$

3.1.2 Concrete Transition

To syntactically analyze the deterministic portion of the control flow, we define the ℓ-labeled version of the syntax:

$$\begin{array}{lcl}
\ell C & ::= & \ell : l^+ \mid \ell : \hat{\_} \\
 & \mid & \ell : \texttt{num}\ E \mid \ell : x\ \texttt{+=}\ E \mid \ell : x\ \texttt{=}\ E \\
 & \mid & \ell : \texttt{hbox}\ \ell C \mid \ell : \ \texttt{vbox}\ \ell C \mid \ell : \texttt{math}\ \ell C \mid \ell : \texttt{unbox} \\
 & \mid & \ell : \texttt{hvswitch} \mid \ell : \texttt{vhswitch} \\
 & \mid & \ell : c\ E? \mid \ell : \texttt{ret}_c \\
 & \mid & \ell : \texttt{if}\ P\ \ell C\ (\texttt{else}\ \ell C)? \mid \ell : \texttt{loop}\ P\ \ell C \mid \ell : \ell C\ \ell C \\
\ell F & ::= & \texttt{def}\ c\ E?\ \ell C \\
\ell D & ::= & \ell F^*\ \ell C\ \ell_{\text{end}}
\end{array}$$

We now collect the function graphs of next, nextTrue, and nextFalse by evaluating $\langle\!\langle \ell D, \ell_{\text{end}} \rangle\!\rangle$ via the following $\langle\!\langle \ell C, \ell' \rangle\!\rangle$ where $\ell = \mathsf{label}(\ell C)$:
$\langle\!\langle \ell C, \ell' \rangle\!\rangle$ = case ℓC of ℓ :

- $l^+ : \{\mathsf{next}(\ell) = \ell'\}$
- $\hat{\_} : \{\mathsf{next}(\ell) = \ell'\}$
- $\texttt{num}\ E : \{\mathsf{next}(\ell) = \ell'\}$
- $x\ \texttt{+=}\ E : \{\mathsf{next}(\ell) = \ell'\}$
- $x\ \texttt{=}\ E : \{\mathsf{next}(\ell) = \ell'\}$
- $\texttt{hbox}\ \ell C' : \{\mathsf{next}(\ell) = \mathsf{label}(\ell C')\} \cup \langle\!\langle \ell C', \ell' \rangle\!\rangle$
- $\texttt{vbox}\ \ell C' : \{\mathsf{next}(\ell) = \mathsf{label}(\ell C')\} \cup \langle\!\langle \ell C', \ell' \rangle\!\rangle$
- $\texttt{math}\ \ell C' : \{\mathsf{next}(\ell) = \mathsf{label}(\ell C')\} \cup \langle\!\langle \ell C', \ell' \rangle\!\rangle$
- $\texttt{unbox} : \{\mathsf{next}(\ell) = \ell'\}$
- $\texttt{hvswitch} : \{\mathsf{next}(\ell) = \ell'\}$
- $\texttt{vhswitch} : \{\mathsf{next}(\ell) = \ell'\}$
- $c\ E? : \{\mathsf{next}(\ell) = \ell'\}$
- $\texttt{ret}_c : \{\}$ (to be determined at run-time)
- $\texttt{if}\ P\ \ell C_1\ (\texttt{else}\ \ell C_2)? : \{\mathsf{nextTrue}(\ell) = \mathsf{label}(\ell C_1), (\mathsf{nextFalse}(\ell) = \mathsf{label}(\ell C_2))?\} \cup \langle\!\langle \ell C_1, \ell' \rangle\!\rangle \cup (\langle\!\langle \ell C_2, \ell' \rangle\!\rangle)?$ $((\ldots)?$ parts are omitted if the `else` clause is not present)
- $\texttt{loop}\ P\ \ell C' : \{\mathsf{nextTrue}(\ell) = \mathsf{label}(\ell C'), \mathsf{nextFalse}(\ell) = \ell'\} \cup \langle\!\langle lC', \ell \rangle\!\rangle$
- $\ell C_1\ \ell C_2 : \{\mathsf{next}(\ell) = \mathsf{label}(\ell C_1)\} \cup \langle\!\langle \ell C_1, \mathsf{label}(\ell C_2) \rangle\!\rangle \cup \langle\!\langle \ell C_2, \ell' \rangle\!\rangle$

Note that ℓF itself is not ℓ-labeled, thus $\langle\!\langle \ell D, \ell_{\text{end}} \rangle\!\rangle = \langle\!\langle \ell C, \ell_{\text{end}}$ where $\ell D = \ell F^*\ \ell C\ \ell_{\text{end}}$.

Now the state transition relation $\langle \ell, d, t, m, \sigma, \kappa, \phi, f \rangle \hookrightarrow \langle \ell', d', t', m', \sigma', \kappa', \phi', f' \rangle$ can be defined as follows:
If $f = \checkmark$, collect $\hookrightarrow$ for each case ℓC of ℓ :

- l^+ : $\langle \ell, d, t, m, \sigma, \kappa, \phi, f \rangle \hookrightarrow \langle \mathsf{next}(\ell), \mathsf{weakHSwitch}(d), t, m, \sigma, \kappa, \phi, f \rangle$
- $\hat{\_}$: $\langle \ell, d, t, m, \sigma, \kappa, \phi, f \rangle \hookrightarrow \langle \mathsf{next}(\ell), \mathfrak{m}, t, m, \sigma, \kappa, \phi, \mathsf{fix}(\ell, d, \mathfrak{m}) \rangle$
- `num` E : $\langle \ell, d, t, m, \sigma, \kappa, \phi, f \rangle \hookrightarrow \langle \mathsf{next}(\ell), \mathsf{weakHSwitch}(d), t, m, \sigma, \kappa, \phi, f \rangle$
- x `+=` E : $\langle \ell, d, t, m, \sigma, \kappa, \phi, f \rangle \hookrightarrow \langle \mathsf{next}(\ell), d, t, \mathsf{advance}_x(m, \mathsf{eval}_E(m, \sigma), \sigma), \sigma, \kappa, \phi, f \rangle$
- x `=` E : $\langle \ell, d, t, m, \sigma, \kappa, \phi, f \rangle \hookrightarrow \langle \mathsf{next}(\ell), d, t, \mathsf{update}_x(m, \mathsf{eval}_E(m, \sigma), \sigma), \sigma, \kappa, \phi, f \rangle$
- `hbox` $\ell C'$: $\langle \ell, d, t, m, \sigma, \kappa, \phi, f \rangle \hookrightarrow \langle \mathsf{next}(\ell), \mathfrak{h}, \mathsf{pushMode}(t, \mathfrak{h}), m, \sigma, \kappa, \phi, f \rangle$
- `vbox` $\ell C'$: $\langle \ell, d, t, m, \sigma, \kappa, \phi, f \rangle \hookrightarrow \langle \mathsf{next}(\ell), \mathfrak{v}, \mathsf{pushMode}(t, \mathfrak{v}), m, \sigma, \kappa, \phi, f \rangle$
- `math` $\ell C'$: $\langle \ell, d, t, m, \sigma, \kappa, \phi, f \rangle \hookrightarrow \langle \mathsf{next}(\ell), \mathfrak{m}, \mathsf{pushMode}(t, \mathfrak{m}), m, \sigma, \kappa, \phi, f \rangle$
- `unbox` : $\langle \ell, d, t, m, \sigma, \kappa, \phi, f \rangle \hookrightarrow \langle \mathsf{next}(\ell), d', t', m, \sigma, \kappa, \phi, f \rangle$ where $\langle d', t' \rangle = \mathsf{popMode}(t)$
- `hvswitch` : $\langle \ell, d, t, m, \sigma, \kappa, \phi, f \rangle \hookrightarrow \langle \mathsf{next}(\ell), \mathfrak{v}, t, m, \sigma, \kappa, \phi, \mathsf{fix}(\ell, d, \mathfrak{h}) \rangle$
- `vhswitch` : $\langle \ell, d, t, m, \sigma, \kappa, \phi, f \rangle \hookrightarrow \langle \mathsf{next}(\ell), \mathfrak{h}, t, m, \sigma, \kappa, \phi, \mathsf{fix}(\ell, d, \mathfrak{v}) \rangle$
- $c\ E?$: $\langle \ell, d, t, m, \sigma, \kappa, \phi, f \rangle \hookrightarrow \langle \mathsf{body}(c), d, t, \mathsf{bind}_{a?}(m, \phi', v), \mathsf{newEnv}_{a?}(\sigma, \phi'), \mathsf{pushCtx}(\kappa, \mathsf{next}(\ell), \sigma), \phi', f' \rangle$ where $a? = \mathsf{arg}(c)$, $v = \mathsf{eval}_{E?}(m, \sigma)$, and $\phi' = \mathsf{tick}(\phi)$.
- $\mathtt{ret}_c$: $\langle \ell, d, t, m, \sigma, \kappa, \phi, f \rangle \hookrightarrow \langle \ell', d, t, m, \sigma', \kappa', \phi', f \rangle$ where $\langle \ell', \sigma', \kappa' \rangle = \mathsf{popCtx}(\kappa)$
- `if` P ℓC_1 (`else` ℓC_2)?, when P is a numeric comparison $E_1 \bowtie E_2$,
  - $\langle \ell, d, t, m, \sigma, \kappa, \phi, f \rangle \hookrightarrow \langle \mathsf{nextTrue}(\ell), d, t, \mathsf{filter}_{E_1 \bowtie E_2}(m, \sigma), \sigma, \kappa, \phi, f \rangle$
  - $\langle \ell, d, t, m, \sigma, \kappa, \phi, f \rangle \hookrightarrow \langle \mathsf{nextFalse}(\ell), d, t, \mathsf{filter}_{\neg(E_1 \bowtie E_2)}(m, \sigma), \sigma, \kappa, \phi, f \rangle$

  otherwise if P is a mode check d',
  - $\langle \ell, d, t, m, \sigma, \kappa, \phi, f \rangle \hookrightarrow \langle \mathsf{nextTrue}(\ell), \mathsf{filter}_{d'}(d), t, m, \sigma, \kappa, \phi, f \rangle$
  - $\langle \ell, d, t, m, \sigma, \kappa, \phi, f \rangle \hookrightarrow \langle \mathsf{nextFalse}(\ell), \mathsf{filter}_{\neg d'}(d), t, m, \sigma, \kappa, \phi, f \rangle$
- `loop` P $\ell C'$, when P is a numeric comparison $E_1 \bowtie E_2$,
  - $\langle \ell, d, t, m, \sigma, \kappa, \phi, f \rangle \hookrightarrow \langle \mathsf{nextTrue}(\ell), d, t, \mathsf{filter}_{E_1 \bowtie E_2}(m, \sigma), \sigma, \kappa, \phi, f \rangle$
  - $\langle \ell, d, t, m, \sigma, \kappa, \phi, f \rangle \hookrightarrow \langle \mathsf{nextFalse}(\ell), d, t, \mathsf{filter}_{\neg(E_1 \bowtie E_2)}(m, \sigma), \sigma, \kappa, \phi, f \rangle$

  otherwise if P is a mode check d',
  - $\langle \ell, d, t, m, \sigma, \kappa, \phi, f \rangle \hookrightarrow \langle \mathsf{nextTrue}(\ell), \mathsf{filter}_{d'}(d), t, m, \sigma, \kappa, \phi, f \rangle$
  - $\langle \ell, d, t, m, \sigma, \kappa, \phi, f \rangle \hookrightarrow \langle \mathsf{nextFalse}(\ell), \mathsf{filter}_{\neg d'}(d), t, m, \sigma, \kappa, \phi, f \rangle$
- ℓC_1 ℓC_2 : $\langle \ell, d, t, m, \sigma, \kappa, \phi, f \rangle \hookrightarrow \langle \mathsf{next}(\ell), d, t, m, \sigma, \kappa, \phi, f \rangle$

Otherwise, if $f \neq \checkmark$, abort.

Semantic operators used above are defined as

- $\mathsf{weakHSwitch} : \mathbb{D} \to \mathbb{D}$,
- $\mathsf{fix} : \mathbb{L} \times \mathbb{D} \times \mathbb{D} \to \{\checkmark\} \cup \mathbb{L} \times \mathbb{D} \times \mathbb{D}$,
- $\mathsf{pushMode} : \mathbb{T} \times \mathbb{D} \to \mathbb{T}$,

- $\mathsf{popMode} : \ \mathbb{T} \to \mathbb{D} \times \mathbb{T}$,
- $\mathsf{advance}_x : \ \mathbb{M} \times \mathbb{Z} \times \mathbb{E} \to \mathbb{M}$,
- $\mathsf{update}_x : \ \mathbb{M} \times \mathbb{V} \times \mathbb{E} \to \mathbb{M}$,
- $\mathsf{eval}_{E?} : \ \mathbb{M} \times \mathbb{E} \to \mathbb{V}$,
- $\mathsf{filter}_{\neg ? E_1 \bowtie E_2} : \ \mathbb{M} \times \mathbb{E} \to \mathbb{M}$,
- $\mathsf{filter}_{\neg ? d'} : \ \mathbb{D} \to \mathbb{D}$,
- $\mathsf{fetch} : \ \mathbb{M} \times \mathbb{X} \times \mathbb{I} \to \mathbb{V}$,
- $\mathsf{body} : \ \mathbb{C} \to \mathbb{L}$,
- $\mathsf{arg} : \ \mathbb{C} \to \mathbb{A}$,
- $\mathsf{bind}_{a?} : \ \mathbb{M} \times \mathbb{I} \times \mathbb{V} \to \mathbb{M}$,
- $\mathsf{newEnv}_{a?} : \ \mathbb{E} \times \mathbb{I} \to \mathbb{E}$,
- $\mathsf{pushCtx} : \ \mathbb{K} \times \mathbb{L} \times \mathbb{E} \to \mathbb{K}$,
- $\mathsf{popCtx} : \ \mathbb{K} \to \mathbb{L} \times \mathbb{E} \times \mathbb{K}$,
- $\mathsf{tick} : \ \mathbb{I} \to \mathbb{I}$,

where

- $\mathsf{weakHSwitch}(d) = \begin{cases} \mathfrak{m} & \text{if } d = \mathfrak{m}, \\ \mathfrak{h} & \text{otherwise;} \end{cases}$
- $\mathsf{fix}(\ell, d, d') = \begin{cases} \checkmark & \text{if } d = d', \\ (\ell, d, d') & \text{otherwise;} \end{cases}$
- $\mathsf{pushMode}(t, d) = d.t$, i.e., stack d on t;
- $\mathsf{popMode}(d.t) = \langle d, t \rangle$;
- $\mathsf{advance}_x(m, n, \sigma) = \mathsf{update}_x(m, n + \mathsf{fetch}(m, x, \sigma), \sigma)$;
- $\mathsf{update}_x(m, v, \sigma) = \mathsf{update}(m, v, x, \sigma) = m[\langle x, \sigma(x) \rangle \mapsto v]$;
- eval is dispatched into two cases:
  - $\mathsf{eval}_{\bullet}(m, \sigma) = 0$ is the case when an empty expression is met, which is the case for a control sequence application that takes no arguments[2], otherwise,
  - $\mathsf{eval}_E(m, \sigma) =$ a trivial case-by-case evaluation;
- filter is (overloaded and) defined only in the following cases:
  - $\mathsf{filter}_{E_1 \bowtie E_2}(m, \sigma) = m \quad$ if $\mathsf{eval}_{E_1}(m, \sigma) \bowtie \mathsf{eval}_{E_2}(m, \sigma)$ is true,
  - $\mathsf{filter}_{\neg(E_1 \bowtie E_2)}(m, \sigma) = m \quad$ if $\mathsf{eval}_{E_1}(m, \sigma) \bowtie \mathsf{eval}_{E_2}(m, \sigma)$ is false,

[2]It is safe to set the value to be zero instead of using a special symbol, as it is impossible to refer to the argument of a control sequence that takes no arguments and bind will do nothing.

  - $\mathsf{filter}_{d'}(d) = d \quad \text{if } d = d'$, and
  - $\mathsf{filter}_{\neg d'}(d) = d \quad \text{if } d \neq d'$;

  otherwise, the transition relation does not exist;

- $\mathsf{fetch}(m, x, \sigma) = m(x, \sigma)$;
- $\mathsf{body}(c) =$ the label of the body of c;
- $\mathsf{arg}(c) = \begin{cases} a_c & \text{if } c \text{ takes an argument,} \\ \bullet & \text{if } c \text{ takes no arguments;} \end{cases}$
- $\mathsf{bind}_{a?}(m, \phi, v) = \begin{cases} m & \text{if } a? = \bullet, \\ m[\langle a, \phi\rangle \mapsto v] & \text{otherwise;} \end{cases}$
- $\mathsf{newEnv}_{a?}(\sigma, \phi) = \begin{cases} \sigma & \text{if } a? = \bullet, \\ \sigma[a \mapsto \phi] & \text{otherwise;} \end{cases}$
- $\mathsf{pushCtx}(\kappa, \ell, \sigma) = \langle \ell, \sigma\rangle.\kappa$;
- $\mathsf{popCtx}(\langle \ell, \sigma\rangle.\kappa) = \langle \ell, \sigma, \kappa\rangle$; and
- $\mathsf{tick}(\phi) = \phi'$ where ϕ' is always fresh.

Note that filter's are partial functions. Technically, popMode and popCtx are partial for empty stacks, but such operations do not happen on syntactically correct τεχ programs.

To clarify the notation •, it is used to represent a "none" value for optional variables like $a?$.

3.1.3 Concrete Semantics

Now that we have set up concrete semantic domains and a single-step transition relation $\hookrightarrow \in \mathbb{S} \to \wp(\mathbb{S})$, the concrete semantics for a set I of input states is the least fixpoint

$$\mathsf{lfp} F$$

of monotonic semantic function

$$\begin{aligned} F &: \wp(\mathbb{S}) \to \wp(\mathbb{S}) \\ F &= \lambda X.\, I \cup \mathsf{Step}(X) \end{aligned}$$

where

$$\begin{aligned} \mathsf{Step} &: \wp(\mathbb{S}) \to \wp(\mathbb{S}) \\ \mathsf{Step} &= \breve{\wp}(\hookrightarrow). \end{aligned}$$

3.2 Abstract Transitional Semantics

3.2.1 Abstract Semantic Domains

Let the abstraction of $\mathbb{S}$ be a CPO

$$\mathbb{S}^\# = \mathbb{L} \to \mathbb{D}^\# \times \mathbb{T}^\# \times \mathbb{M}^\# \times \mathbb{E}^\# \times \mathbb{K}^\# \times \mathbb{I}^\# \times \mathbb{F}^\#.$$

It should be Galois connected:

$$(\wp(\mathbb{S}), \subseteq) \underset{\alpha_{\mathbb{S}}}{\overset{\gamma_{\mathbb{S}}}{\leftrightarrows}} (\mathbb{S}^\#, \sqsubseteq_{\mathbb{S}})$$

where each abstract component domain is also a Galois connect CPO:

$$\begin{aligned}
(\wp(\mathbb{D}), \subseteq) &\underset{\alpha_{\mathbb{D}}}{\overset{\gamma_{\mathbb{D}}}{\leftrightarrows}} (\mathbb{D}^\#, \sqsubseteq_{\mathbb{D}}),\\
(\wp(\mathbb{T}), \subseteq) &\underset{\alpha_{\mathbb{T}}}{\overset{\gamma_{\mathbb{T}}}{\leftrightarrows}} (\mathbb{T}^\#, \sqsubseteq_{\mathbb{T}}),\\
(\wp(\mathbb{M}), \subseteq) &\underset{\alpha_{\mathbb{M}}}{\overset{\gamma_{\mathbb{M}}}{\leftrightarrows}} (\mathbb{M}^\#, \sqsubseteq_{\mathbb{M}}),\\
(\wp(\mathbb{E}), \subseteq) &\underset{\alpha_{\mathbb{E}}}{\overset{\gamma_{\mathbb{E}}}{\leftrightarrows}} (\mathbb{E}^\#, \sqsubseteq_{\mathbb{E}}),\\
(\wp(\mathbb{K}), \subseteq) &\underset{\alpha_{\mathbb{K}}}{\overset{\gamma_{\mathbb{K}}}{\leftrightarrows}} (\mathbb{K}^\#, \sqsubseteq_{\mathbb{K}}),\\
(\wp(\mathbb{I}), \subseteq) &\underset{\alpha_{\mathbb{I}}}{\overset{\gamma_{\mathbb{I}}}{\leftrightarrows}} (\mathbb{I}^\#, \sqsubseteq_{\mathbb{I}}),\\
(\wp(\mathbb{F}), \subseteq) &\underset{\alpha_{\mathbb{F}}}{\overset{\gamma_{\mathbb{F}}}{\leftrightarrows}} (\mathbb{F}^\#, \sqsubseteq_{\mathbb{F}}).
\end{aligned}$$

We now design the abstract component domains.

Modes: As there are only three elements in $\mathbb{D} = \{\mathfrak{h}, \mathfrak{v}, \mathfrak{m}\}$, we use a powerset domain

$$\mathbb{D}^\# = \wp(\mathbb{D}).$$

Nested modes: Due to recursive nesting of boxes, we keep only the top-most $k_{\mathbb{T}}$ modes as in k-CFA:

$$\mathbb{T}^\# = \bigcup_{0 \le i \le k_{\mathbb{T}}} \mathbb{D}^{\# i}.$$

Then abstraction and concretization functions can be defined as follows:

$$\begin{aligned}
\alpha_{\mathbb{T}}(T) &= \sqcup_{\mathbb{T}} \{T^\# \text{ is a prefix of } t \in T \mid |T^\#| \le k_{\mathbb{T}}\}\\
\gamma_{\mathbb{T}}(T^\#) &= \{t \in \mathbb{T} \mid T^\# \text{ is a prefix of } t \text{ up to } \sqcup_{\mathbb{D}}\}
\end{aligned}$$

where $\sqcup_{\mathbb{T}}$ is the longest common prefix up to $\sqcup_{\mathbb{D}}$. Note that $\sqcup_{\mathbb{D}} = \cup$ in the current choice of $\mathbb{D}^\#$.

This is a Galois connection. We first show $\alpha_{\mathbb{T}}(T) \sqsubseteq_{\mathbb{T}} T^\#$ implies $T \subseteq \gamma_{\mathbb{T}}(T^\#)$. Since

$$\begin{aligned}
\alpha_{\mathbb{T}}(T) \sqsubseteq_{\mathbb{T}} T^\# &\Leftrightarrow \sqcup_{\mathbb{T}} \{T'^\# \text{ is a prefix of } t \in T \mid |T'^\#| \le k_{\mathbb{T}}\}\\
&\Leftrightarrow \forall T'^\#.\, T'^\# \sqsubseteq_{\mathbb{T}} T^\#\\
&\Leftrightarrow \forall T'^\#.\, T^\# \text{ is a prefix of } T'^\#,
\end{aligned}$$

for any $t \in T$ we have $T^\#$ as a prefix of t. Thus $T \subseteq \gamma_{\mathbb{T}}(T^\#)$ by definition.

On the other hand, when for all $t \in T$, $T^\#$ is a prefix of t (up to $\sqcup_{\mathbb{D}}$), we can choose any prefix $T'^\#$ of t. Then $T^\#$ is still a prefix of $T'^\#$, including $T'^\#$ with a length less than or equal to $k_{\mathbb{T}}$. Therefore we have $\alpha_{\mathbb{T}}(T) \sqsubseteq_{\mathbb{T}} T^\#$ by definition.

Memories: Given that $\mathbb{V}^\#$ is a Galois connected CPO,

$$\mathbb{M}^\# = \mathbb{X} \times \mathbb{I}^\# \to \mathbb{V}^\#$$

is a Galois connected CPO.

Values: Thus we provide a Galois connected abstract domain for $\mathbb{V} = \mathbb{Z} \cup \mathbb{C}$ in a kindwise manner:

$$(\wp(\mathbb{Z} \cup \mathbb{C}), \subseteq) \underset{\alpha_{\mathbb{V}}}{\overset{\gamma_{\mathbb{V}}}{\leftrightarrows}} (\mathbb{Z}^\# \times \mathbb{C}^\#, \sqsubseteq_{\mathbb{V}}),$$

where we use $\mathbb{Z}^\#$ to be an interval abstraction and $\mathbb{C}^\# = \wp(\mathbb{C})$.

Variables: We use

$$\mathbb{X}^\# = \wp(\mathbb{X}).$$

Environments: We use

$$\mathbb{E}^\# = \mathbb{X} \to \mathbb{I}^\#.$$

Continuations: Unlike k-CFA style abstraction for nested modes, we resort to a context-insensitive analysis for control sequence calls.[3] That is,

$$\mathbb{K}^\# = \{\kappa^\#\}.$$

Instances: Similarly,

$$\mathbb{I}^\# = \{\phi^\#\}.$$

Implicit fixes: First divide $\mathbb{F} = \{\checkmark\} \cup \mathbb{L} \times \mathbb{D} \times \mathbb{D}$ into $\mathbb{F}' = \{\checkmark\}$ and $\mathbb{F}'' = \mathbb{L} \times \mathbb{D} \times \mathbb{D}$. If we have $\wp(\mathbb{F}') \underset{\alpha_{\mathbb{F}'}}{\overset{\gamma_{\mathbb{F}'}}{\leftrightarrows}} \mathbb{F}'^\#$ and $\wp(\mathbb{F}'') \underset{\alpha_{\mathbb{F}''}}{\overset{\gamma_{\mathbb{F}''}}{\leftrightarrows}} \mathbb{F}''^\#$, we have

$$\wp(\mathbb{F}' \cup \mathbb{F}'') \underset{\alpha_{\mathbb{F}}}{\overset{\gamma_{\mathbb{F}}}{\leftrightarrows}} \mathbb{F}'^\# \times \mathbb{F}''^\#$$

where $\alpha_{\mathbb{F}} = \lambda F.\langle \alpha_{\mathbb{F}'}(F \cap \mathbb{F}'^\#), \alpha_{\mathbb{F}''}(F \cap \mathbb{F}''^\#)\rangle$.

Now it is straightforward: we choose $\mathbb{F}'^\# = \wp(\mathbb{F}')$ and $\mathbb{F}''^\# = \mathbb{L} \to \mathbb{D}^\# \times \mathbb{D}^\#$. Thus we have

$$\wp(\mathbb{F}) \underset{\alpha_{\mathbb{F}}}{\overset{\gamma_{\mathbb{F}}}{\leftrightarrows}} \{\emptyset, \{\checkmark\}\} \times (\mathbb{L} \to \mathbb{D}^\# \times \mathbb{D}^\#).$$

Note that every domain except $\mathbb{Z}^\#$ is of finite height, and we use a widening operator

$$[a_0, a_1] \triangledown_{\mathbb{Z}} [b_0, b_1] = [(a_0 \text{ if } a_0 \leq b_0, \text{ otherwise } -\infty), (a_1 \text{ if } a_1 \geq b_1, \text{ otherwise } +\infty)]$$

for it.

[3]Recursive calls in TeX is not seen often in day-to-day document typesetting, unlike deeply nested boxes.

3.2.2 Abstract Transition

Here is the definition of the abstract transition relation $\langle \ell, D^\#, T^\#, M^\#, \sigma^\#, \kappa^\#, \phi^\#, F^\# \rangle \hookrightarrow^\#$ $\langle \ell', D'^\#, T'^\#, M'^\#, \sigma'^\#, \kappa'^\#, \phi'^\#, F'^\# \rangle$:
If $F^\#.1 = \{\checkmark\}$ and $D^\# \neq \emptyset$, collect $\hookrightarrow^\#$ for each case ℓC of ℓ :

- $l^+ : \langle \ell, D^\#, T^\#, M^\#, \sigma^\#, \kappa^\#, \phi^\#, F^\# \rangle \hookrightarrow^\# \langle \mathsf{next}(\ell), \mathsf{weakHSwitch}^\#(D^\#), T^\#, M^\#, \sigma^\#, \kappa^\#, \phi^\#, F^\# \rangle$
- $\hat{\_} : \langle \ell, D^\#, T^\#, M^\#, \sigma^\#, \kappa^\#, \phi^\#, F^\# \rangle \hookrightarrow^\# \langle \mathsf{next}(\ell), \{\mathfrak{m}\}, T^\#, M^\#, \sigma^\#, \kappa^\#, \phi^\#, \mathsf{fix}^\#(\ell, D^\#, \{\mathfrak{m}\}) \rangle$
- `num` $E : \langle \ell, D^\#, T^\#, M^\#, \sigma^\#, \kappa^\#, \phi^\#, F^\# \rangle \hookrightarrow^\# \langle \mathsf{next}(\ell), \mathsf{weakHSwitch}^\#(D^\#), T^\#, M^\#, \sigma^\#, \kappa^\#, \phi^\#, F^\# \rangle$
- x `+=` $E : \langle \ell, D^\#, T^\#, M^\#, \sigma^\#, \kappa^\#, \phi^\#, F^\# \rangle \hookrightarrow^\# \langle \mathsf{next}(\ell), D^\#, T^\#, \mathsf{advance}^\#_x(M^\#, \mathsf{eval}^\#_E(M^\#, \sigma^\#), \sigma^\#), \sigma^\#, \kappa^\#, \phi^\#, F^\# \rangle$
- x `=` $E : \langle \ell, D^\#, T^\#, M^\#, \sigma^\#, \kappa^\#, \phi^\#, F^\# \rangle \hookrightarrow^\# \langle \mathsf{next}(\ell), D^\#, T^\#, \mathsf{update}^\#_x(M^\#, \mathsf{eval}^\#_E(M^\#, \sigma^\#), \sigma^\#), \sigma^\#, \kappa^\#, \phi^\#, F^\# \rangle$
- `hbox` $\ell C' : \langle \ell, D^\#, T^\#, M^\#, \sigma^\#, \kappa^\#, \phi^\#, F^\# \rangle \hookrightarrow^\# \langle \mathsf{next}(\ell), \{\mathfrak{h}\}, \mathsf{pushMode}^\#(T^\#, \{\mathfrak{h}\}), M^\#, \sigma^\#, \kappa^\#, \phi^\#, F^\# \rangle$
- `vbox` $\ell C' : \langle \ell, D^\#, T^\#, M^\#, \sigma^\#, \kappa^\#, \phi^\#, F^\# \rangle \hookrightarrow^\# \langle \mathsf{next}(\ell), \{\mathfrak{v}\}, \mathsf{pushMode}^\#(T^\#, \{\mathfrak{v}\}), M^\#, \sigma^\#, \kappa^\#, \phi^\#, F^\# \rangle$
- `math` $\ell C' : \langle \ell, D^\#, T^\#, M^\#, \sigma^\#, \kappa^\#, \phi^\#, F^\# \rangle \hookrightarrow^\# \langle \mathsf{next}(\ell), \{\mathfrak{m}\}, \mathsf{pushMode}^\#(T^\#, \{\mathfrak{m}\}), M^\#, \sigma^\#, \kappa^\#, \phi^\#, F^\# \rangle$
- `unbox` $: \langle \ell, D^\#, T^\#, M^\#, \sigma^\#, \kappa^\#, \phi^\#, F^\# \rangle \hookrightarrow^\# \langle \mathsf{next}(\ell), D'^\#, T'^\#, M^\#, \sigma^\#, \kappa^\#, \phi^\#, F^\# \rangle$ where $\langle D'^\#, T'^\# \rangle = \mathsf{popMode}^\#(T^\#)$
- `hvswitch` $: \langle \ell, D^\#, T^\#, M^\#, \sigma^\#, \kappa^\#, \phi^\#, F^\# \rangle \hookrightarrow^\# \langle \mathsf{next}(\ell), \{\mathfrak{v}\}, T^\#, M^\#, \sigma^\#, \kappa^\#, \phi^\#, \mathsf{fix}^\#(\ell, D^\#, \{\mathfrak{h}\}) \rangle$
- `vhswitch` $: \langle \ell, D^\#, T^\#, M^\#, \sigma^\#, \kappa^\#, \phi^\#, F^\# \rangle \hookrightarrow^\# \langle \mathsf{next}(\ell), \{\mathfrak{h}\}, T^\#, M^\#, \sigma^\#, \kappa^\#, \phi^\#, \mathsf{fix}^\#(\ell, D^\#, \{\mathfrak{v}\}) \rangle$
- $c\ \ E? : \langle \ell, D^\#, T^\#, M^\#, \sigma^\#, \kappa^\#, \phi^\#, F^\# \rangle \hookrightarrow^\#$
  $\langle \mathsf{body}(c), D^\#, T^\#, \mathsf{bind}^\#_{a?}(M^\#, \phi'^\#, V^\#), \mathsf{newEnv}^\#_{a?}(\sigma^\#, \phi'^\#), \mathsf{pushCtx}^\#(\kappa^\#, \mathsf{next}(\ell), \sigma^\#), \phi'^\#, F'^\# \rangle$
  where $a? = \mathsf{arg}(c)$, $V^\# = \mathsf{eval}^\#_{E?}(M^\#, \sigma^\#)$, and $\phi'^\# = \mathsf{tick}^\#(\phi^\#)$.
- $\texttt{ret}_c : \langle \ell, D^\#, T^\#, M^\#, \sigma^\#, \kappa^\#, \phi^\#, F^\# \rangle \hookrightarrow^\# \langle \ell', D^\#, T^\#, M^\#, \sigma'^\#, \kappa'^\#, \phi'^\#, F^\# \rangle$ where $\langle \ell'^\#, \sigma'^\#, \kappa'^\# \rangle =$ $\mathsf{popCtx}^\#_c(\kappa^\#)$ and $\ell' \in \ell'^\#$
- `if` $P\ \ \ell C_1$ `(else` ℓC_2)?, when P is a numeric comparison $E_1 \bowtie E_2$,
  - $\langle \ell, D^\#, T^\#, M^\#, \sigma^\#, \kappa^\#, \phi^\#, F^\# \rangle \hookrightarrow^\# \langle \mathsf{nextTrue}(\ell), D^\#, T^\#, \mathsf{filter}^\#_{E_1 \bowtie E_2}(M^\#, \sigma^\#), \sigma^\#, \kappa^\#, \phi^\#, F^\# \rangle$
  - $\langle \ell, D^\#, T^\#, M^\#, \sigma^\#, \kappa^\#, \phi^\#, F^\# \rangle \hookrightarrow^\# \langle \mathsf{nextFalse}(\ell), D^\#, T^\#, \mathsf{filter}^\#_{\neg(E_1 \bowtie E_2)}(M^\#, \sigma^\#), \sigma^\#, \kappa^\#, \phi^\#, F^\# \rangle$

  otherwise if P is a mode check d',
  - $\langle \ell, D^\#, T^\#, M^\#, \sigma^\#, \kappa^\#, \phi^\#, F^\# \rangle \hookrightarrow^\# \langle \mathsf{nextTrue}(\ell), \mathsf{filter}^\#_{d'}(D^\#), T^\#, M^\#, \sigma^\#, \kappa^\#, \phi^\#, F^\# \rangle$
  - $\langle \ell, D^\#, T^\#, M^\#, \sigma^\#, \kappa^\#, \phi^\#, F^\# \rangle \hookrightarrow^\# \langle \mathsf{nextFalse}(\ell), \mathsf{filter}^\#_{\neg d'}(D^\#), T^\#, M^\#, \sigma^\#, \kappa^\#, \phi^\#, F^\# \rangle$
- `loop` $P\ \ \ell C'$, when P is a numeric comparison $E_1 \bowtie E_2$,
  - $\langle \ell, D^\#, T^\#, M^\#, \sigma^\#, \kappa^\#, \phi^\#, F^\# \rangle \hookrightarrow^\# \langle \mathsf{nextTrue}(\ell), D^\#, T^\#, \mathsf{filter}^\#_{E_1 \bowtie E_2}(M^\#, \sigma^\#), \sigma^\#, \kappa^\#, \phi^\#, F^\# \rangle$
  - $\langle \ell, D^\#, T^\#, M^\#, \sigma^\#, \kappa^\#, \phi^\#, F^\# \rangle \hookrightarrow^\# \langle \mathsf{nextFalse}(\ell), D^\#, T^\#, \mathsf{filter}^\#_{\neg(E_1 \bowtie E_2)}(M^\#, \sigma^\#), \sigma^\#, \kappa^\#, \phi^\#, F^\# \rangle$

  otherwise if P is a mode check d',
  - $\langle \ell, D^\#, T^\#, M^\#, \sigma^\#, \kappa^\#, \phi^\#, F^\# \rangle \hookrightarrow^\# \langle \mathsf{nextTrue}(\ell), \mathsf{filter}^\#_{d'}(D^\#), T^\#, M^\#, \sigma^\#, \kappa^\#, \phi^\#, F^\# \rangle$
  - $\langle \ell, D^\#, T^\#, M^\#, \sigma^\#, \kappa^\#, \phi^\#, F^\# \rangle \hookrightarrow^\# \langle \mathsf{nextFalse}(\ell), \mathsf{filter}^\#_{\neg d'}(D^\#), T^\#, M^\#, \sigma^\#, \kappa^\#, \phi^\#, F^\# \rangle$

- $\ell C_1 \;\; \ell C_2 : \langle \ell, D^\#, T^\#, M^\#, \sigma^\#, \kappa^\#, \phi^\#, F^\# \rangle \hookrightarrow^\# \langle \mathsf{next}(\ell), D^\#, T^\#, M^\#, \sigma^\#, \kappa^\#, \phi^\#, F^\# \rangle$

Otherwise, if $F^\#.1 = \emptyset$ or $D^\# = \emptyset$, abort.

Abstract semantic operators used above are defined as

- $\mathsf{weakHSwitch}^\# : \mathbb{D}^\# \to \mathbb{D}^\#$,
- $\mathsf{fix}^\# : \mathbb{L} \times \mathbb{D}^\# \times \mathbb{D}^\# \to \mathbb{F}^\#$,
- $\mathsf{pushMode}^\# : \mathbb{T}^\# \times \mathbb{D}^\# \to \mathbb{T}^\#$,
- $\mathsf{popMode}^\# : \mathbb{T}^\# \to \mathbb{D}^\# \times \mathbb{T}^\#$,
- $\mathsf{advance}^\#_x : \mathbb{M}^\# \times \mathbb{Z}^\# \times \mathbb{E}^\# \to \mathbb{M}^\#$,
- $\mathsf{update}^\#_x : \mathbb{M}^\# \times \mathbb{V}^\# \times \mathbb{E}^\# \to \mathbb{M}^\#$,
- $\mathsf{eval}^\#_{E?} : \mathbb{M}^\# \times \mathbb{E}^\# \to \mathbb{V}^\#$,
- $\mathsf{filter}^\#_{\neg ?E_1 \bowtie E_2} : \mathbb{M}^\# \times \mathbb{E}^\# \to \mathbb{M}^\#$,
- $\mathsf{filter}^\#_{\neg ?d'} : \mathbb{D}^\# \to \mathbb{D}^\#$,
- $\mathsf{fetch}^\# : \mathbb{M}^\# \times \mathbb{X}^\# \times \mathbb{I}^\# \to \mathbb{V}^\#$,
- $\mathsf{bind}^\#_{a?} : \mathbb{M}^\# \times \mathbb{I}^\# \times \mathbb{V}^\# \to \mathbb{M}^\#$,
- $\mathsf{newEnv}^\#_{a?} : \mathbb{E}^\# \times \mathbb{I}^\# \to \mathbb{E}^\#$,
- $\mathsf{pushCtx}^\# : \mathbb{K}^\# \times \mathbb{L} \times \mathbb{E}^\# \to \mathbb{K}^\#$,
- $\mathsf{popCtx}^\#_c : \mathbb{K}^\# \to \mathbb{L}^\# \times \mathbb{E}^\# \times \mathbb{K}^\#$,
- $\mathsf{tick}^\# : \mathbb{I}^\# \to \mathbb{I}^\#$,

where

- $\mathsf{weakHSwitch}^\#(D^\#) = \{\mathsf{weakHSwitch}(d) \mid d \in D^\#\}$,
- $\mathsf{fix}^\#(\ell, D_1^\#, D_2^\#) = \langle H, \langle \ell, D_1'^\#, D_2'^\# \rangle \rangle$ where

$$H = \begin{cases} \{\checkmark\} & \text{if } D_1^\# \cap D_2^\# \neq \emptyset \\ \emptyset & \text{otherwise,} \end{cases}$$

  and

$$(D_1'^\#, D_2'^\#) = \begin{cases} (\emptyset, \emptyset) & \text{if } D_1^\# = \emptyset \text{ or } D_2^\# = \emptyset, \\ (\emptyset, \emptyset) & \text{if } |D_1^\#| = |D_2^\#| = 1 \text{ and } D_1^\# = D_2^\#, \\ (D_1^\#, D_2^\# - D_1^\#) & \text{if } |D_1^\#| = 1, \\ (D_1^\# - D_2^\#, D_2^\#) & \text{if } |D_2^\#| = 1, \\ (D_1^\#, D_2^\#) & \text{otherwise;} \end{cases}$$

- $\mathsf{pushMode}^\#(T^\#, D^\#) = \mathsf{trunc}_{k_\mathbb{T}}(D^\#.T^\#)$ where trunc_k truncates the stack to the topmost k elements if the length is greater than k.

- $\mathsf{popMode}^\#(T^\#) = \begin{cases} \langle D^\#, T'^\# \rangle & \text{if } |T^\#| > 0, \\ \langle \{\mathfrak{h}, \mathfrak{v}, \mathfrak{m}\}, \top_{\mathbb{T}^\#} \rangle & \text{otherwise;} \end{cases}$
- $\mathsf{advance}^\#_x(M^\#, N^\#, \sigma^\#) = \mathsf{update}_x(M^\#, N^\# +^\# \mathsf{fetch}^\#(M^\#, \{x\}, \sigma^\#), \sigma^\#)$;
- $\mathsf{update}^\#_x(M^\#, V^\#, \sigma^\#, X^\#) = \begin{cases} M^\#[\langle x, \sigma^\#(x) \rangle \mapsto V^\#] & \text{when } X^\# = \{x\}, \\ \bigsqcup_{x \in X^\#} M^\#[\langle x, \sigma^\#(x) \rangle \mapsto M^\#(x, \sigma^\#(x)) \sqcup V^\#] & \text{otherwise;} \end{cases}$

  (Note that $\mathsf{update}^\#_x(M^\#, V^\#, \sigma^\#) = \mathsf{update}^\#(M^\#, V^\#, \sigma^\#, \{x\})$)
- $\mathsf{eval}^\#$ is dispatched into two cases:
  - $\mathsf{eval}^\#_\bullet(M^\#, \sigma^\#) = [0, 0]$ when an empty expression is met, otherwise,
  - $\mathsf{eval}^\#_E(M^\#, \sigma^\#) =$ a straightforward case-by-case abstract correspondent for $\mathsf{eval}_E(m, \sigma)$;
- $\mathsf{filter}^\#$ is (overloaded and) defined only in the following cases:
  - $\mathsf{filter}^\#_{E_1 \bowtie E_2}(M^\#, \sigma^\#) = \alpha_{\mathbb{M}}(\{m \in \gamma_{\mathbb{M}}(M^\#) \mid \exists \sigma \in \gamma_{\mathbb{E}}(\sigma^\#). \mathsf{eval}_{E_1}(m, \sigma) \bowtie \mathsf{eval}_{E_2}(m, \sigma) \text{ is true}\})$,
  - $\mathsf{filter}^\#_{\neg(E_1 \bowtie E_2)}(M^\#, \sigma^\#) = \alpha_{\mathbb{M}}(\{m \in \gamma_{\mathbb{M}}(M^\#) \mid \exists \sigma \in \gamma_{\mathbb{E}}(\sigma^\#). \mathsf{eval}_{E_1}(m, \sigma) \bowtie \mathsf{eval}_{E_2}(m, \sigma) \text{ is false}\})$,
  - $\mathsf{filter}^\#_{d'}(D^\#) = D^\# \cap \{d'\}$,
  - $\mathsf{filter}^\#_{\neg d'}(D^\#) = D^\# - \{d'\}$;

  otherwise, the transition relation does not exist;
- $\mathsf{fetch}^\#(M^\#, X^\#, \sigma^\#) = \bigsqcup_{x \in X^\#} M^\#(x, \sigma^\#(x))$;
- $\mathsf{bind}^\#_{a?}(M^\#, \phi^\#, V^\#) = \begin{cases} m & \text{if } a? = \bullet, \\ m[\langle a, \phi^\# \rangle \mapsto V^\#] & \text{otherwise;} \end{cases}$
- $\mathsf{newEnv}^\#_{a?}(\sigma^\#, \phi^\#) = \begin{cases} \sigma^\# & \text{if } a? = \bullet, \\ \sigma^\#[a \mapsto \phi^\#] & \text{otherwise;} \end{cases}$
- $\mathsf{pushCtx}^\#(\kappa^\#, \ell, \sigma^\#) = \kappa^\#$;
- $\mathsf{popCtx}^\#_c(\kappa^\#) = \langle \ell^\#, \sigma^\#, \kappa^\# \rangle$ where $\ell^\# = \{\ell \mid \langle \ell, \cdot, \cdot, \cdot, \cdot, \cdot, \cdot, \cdot \rangle \hookrightarrow^\# \langle \mathsf{body}(c), \cdot, \cdot, \cdot, \cdot, \cdot, \cdot, \cdot \rangle\}$ and $\sigma^\# = \lambda x. \phi^\#$;
- $\mathsf{tick}^\#(\phi^\#) = \phi^\#$.

Note again that $\mathbb{I}^\# = \{\phi^\#\}$ and $\mathbb{K}^\# = \{\kappa^\#\}$ are singleton sets.

3.2.3 Abstract Semantics

We define an abstract semantics as

$$\begin{aligned}
\mathbb{S}^\# &= \mathbb{L} \to \mathbb{D}^\# \times \mathbb{T}^\# \times \mathbb{M}^\# \times \mathbb{E}^\# \times \mathbb{K}^\# \times \mathbb{I}^\# \times \mathbb{F}^\# \\
F^\# &: \mathbb{S}^\# \to \mathbb{S}^\# \\
F^\#(X^\#) &= \alpha(I) \sqcup \mathsf{Step}^\#(X^\#) \\
\mathsf{Step}^\# &= \wp(\mathrm{id}, \sqcup_R) \circ \pi \circ \breve{\wp}(\hookrightarrow^\#) \\
\hookrightarrow^\# &\subseteq \mathbb{S}^\# \times \mathbb{S}^\# \\
\pi &: \wp(\mathbb{S}^\#) \to (\mathbb{L} \to \wp(\mathbb{D}^\# \times \mathbb{T}^\# \times \mathbb{M}^\# \times \mathbb{E}^\# \times \mathbb{K}^\# \times \mathbb{I}^\# \times \mathbb{F}^\#)) \\
\pi(X) &= \lambda \ell. \left\{ \langle D^\#, T^\#, M^\#, \sigma^\#, \kappa^\#, \phi^\#, F^\# \rangle \,\middle|\, \langle \ell, D^\#, T^\#, M^\#, \sigma^\#, \kappa^\#, \phi^\#, F^\# \rangle \in X \right\}
\end{aligned}$$

where $\sqcup_R$ is an upper bound operator of $\mathbb{D}^\sharp \times \mathbb{T}^\sharp \times \mathbb{M}^\sharp \times \mathbb{E}^\sharp \times \mathbb{K}^\sharp \times \mathbb{I}^\sharp \times \mathbb{F}^\sharp$.

4 Analysis

Using the abstract semantics presented above, we can consult the label-wise collected modes for a document inspection. In case the abstract interpreter meets a state $F^\sharp.1 = \emptyset$, it means that the document is surely in an error state, and a TEXnician should check $F^\sharp.2$ to see the possible first occurrences of implicit fixes. On the other hand, if $F^\sharp.2$ is empty, it means that the document is guaranteed to be in a clean state without any implicit fix.

References

[1] Xavier Rival and Kwangkeun Yi (2020) *Introduction to Static Analysis,* MIT Press.

[2] Donald E. Knuth (1986) *The TEXBook,* Addison-Wesley Professional.