SAFE 2.0 User Manual

Jihyeok Park, Yeonhee Ryou, and Sukyoung Ryu

Chapter 1

Foreword

1.1 Audience

This document is for users of SAFE (Scalable Analysis Framework for ECMAScript) 2.0, a scalable and pluggable analysis framework for JavaScript web applications. General information on the SAFE project is available at an invited talk at ICFP 2016 [26]:

`https://www.youtube.com/watch?v=gEU9utf0sxE`

and the source code and publications are available at:

`https://github.com/sukyoung/safe`

For more information, please contact the main developers of SAFE at `safe [at] plrg.kaist.ac.kr`.

SAFE has been used by:

- JSAI [12] @ UCSB
- ROSAEC [1] @ Seoul National University
- K framework [21] @ UIUC
- Ken Cheung [5] @ HKUST
- Web-based vulnerability detection [14] @ Oracle
- Tizen [9] @ Linux Foundation

1.2 Contributors

The main developers of SAFE 2.0 are as follows:

- Jihyeok Park
- Yeonhee Ryou
- Sukyoung Ryu

and the following have contributed to the source code:

- Minsoo Kim (Built-in function modeling)
- PLRG @ KAIST and our colleagues in S-Core and Samsung Electronics (SAFE 1.0)

1.3 License

The SAFE source code is released under the BSD license:

`github.com/sukyoung/safe/blob/master/LICENSE`

1.4 Installation

We assume you are using an operating system with a Unix-style shell (for example, Mac OS X, Linux, or Cygwin on Windows). Assuming `SAFE_HOME` points to the SAFE directory, you will need to have access to the following:

- J2SDK 1.8. See `http://java.sun.com/javase/downloads/index.jsp`
- Scala 2.12. See `http://scala-lang.org/download`
- sbt version 0.13. See `http://www.scala-sbt.org`
- Bash version 2.5, installed at `/bin/bash`. See `http://www.gnu.org/software/bash/`

In your shell startup script, add `$SAFE_HOME/bin` to your path. The shell scripts in this directory are Bash scripts. To run them, you must have Bash accessible in `/bin/bash`.

Type `sbt compile` and then `sbt test` to make sure that your installation successfully finishes the tests. Two regression test suites are provided with SAFE and can be analyzed automatically:

```
$ sbt test
$ sbt test262Test
```

In addition to the SAFE-specific test suite, SAFE 2.0 has been tested using Test262, the official ECMAScript (ECMA-262) conformance suite:

`https://github.com/tc39/test262`

Not a single test should end in a failure.

Once you have built the framework, you can call it from any directory, on any JavaScript file, simply by typing one of available commands at a command line as explained in Chapter 3.

1.4.1 IntelliJ configuration

IntelliJ users can use IntelliJ 2016.2.4 with the latest Scala plugin as follows:

1. Create a new project from existing sources (aka. `Import project`).
2. Choose `build.sbt` in the SAFE 2.0 root to import.
3. Choose JDK 1.8 as the project JDK.
4. Manually download `xtc.jar` in to `lib/`
5. Goto `Project Settings` → `Modules` → `root (module)` → `Dependencies`
6. Open `SBT:unmanaged-jars` dependencies.
7. Remove broken entries for `spray-json` and `xtc`.
8. Add `(+) .jars` for the two libraries above.
9. Run the `buildParsers` task in SBT.

Chapter 2

SAFE

2.1 Introduction to SAFE 1.0

Analyzing real-world JavaScript web applications is a challenging task. On top of understanding the semantics of JavaScript [2], it requires modeling of web documents [27], platform objects [9], and interactions between them. Not only JavaScript itself but also its usage patterns are extremely dynamic [25, 24]. Most of web applications load JavaScript code dynamically, which makes pure static analysis approaches inapplicable.

To analyze JavaScript web applications in the wild mostly statically, we have developed SAFE and extended it with various approaches. We first described quirky language features and semantics of JavaScript that make static analysis difficult and designed SAFE to analyze pure JavaScript benchmarks [15]. It provides a default static analyzer based on the abstract interpretation framework [7], and it supports flow-sensitive and context-sensitive analyses of stand-alone JavaScript programs. It performs several preprocessing steps on JavaScript code to address some quirky semantics of JavaScript such as the `with` statement [18]. The pluggable and scalable design of the framework allowed experiments with JavaScript variants like adding a module system [11, 6] and detecting code clones [5].

We then extended SAFE to model web application execution environments of various browsers [20] and platform-specific library functions [4, 22]. To provide a faithful (partial) model of browsers, we support the configurability of HTML/DOM tree abstraction levels so that users can adjust a trade-off between analysis performance and precision depending on their applications. To analyze interactions between applications and platform-specific libraries specified in Web APIs written in Web IDLs, we developed automatic modeling of library functions from Web APIs and detect possible misuses of Web APIs by web applications. The same technique can support analysis of libraries specified in TypeScript [17]. Analyzing real-world web applications requires more scalable analysis than analyzing stand-alone JavaScript programs [13, 19].

The baseline analysis is designed to be sound, which means that the properties it computes should over-approximate the concrete behaviors of the analyzed program. However, SAFE may contain implementation bugs leading to unsound analysis results. Moreover, some components of SAFE may be intentionally unsound, or soundy [16]. To lessen the burden of analyzing the entire concrete behaviors of programs, we may use approximate call graphs [8] from WALA [10] to analyze a fraction of them, or utilize dynamic information statically [23] to prune relatively unrelated code.

2.2 Introduction to SAFE 2.0

Based on our experiments and experiences with SAFE 1.0, we now release SAFE 2.0, which is aimed to be a playground for advanced research in JavaScript web applications. Thus, we intentionally designed it to be lightweight, highly parametric, and modular.

The important changes from SAFE 1.0 include the following:

- SAFE 2.0 has been tested using Test262, the official ECMAScript (ECMA-262) conformance suite.
- SAFE 2.0 now uses sbt instead of ant to build the framework.
- SAFE 2.0 provides a library of abstract domains that supports parameterization and high-level specification of abstract semantics on them.
- Most Java source files are replaced by Scala code and the only Java source code remained is the generated parser code.
- Several components from SAFE 1.0 may not be integrated into SAFE 2.0. Such components include interpreter, concolic testing, clone detector, clone refactoring, TypeScript support, Web API misuse detector, and several abstract domains like the string automata domain.

We have the following roadmap for SAFE 2.0:

- SAFE 2.0 will make monthly updates.
- The next update will include a SAFE document, browser benchmarks, and more Test262 tests.
- We plan to support some missing features from SAFE 1.0 incrementally such as a bug detector, DOM modeling, and jQuery analysis.
- Future versions of SAFE 2.0 will address various analysis techniques, dynamic features of web applications, event handling, modeling framework, compositional analysis, and selective sensitivity among others.

2.3 A sample use of SAFE

Let us consider a very simple JavaScript program stored in a file name "sample.js" located in the current directory:

```
with({a: 1}) {a = 2;}
```

Then, one can see how SAFE desugars the `with` statement by the command below:

```
safe astRewrite sample.js
```

which shows an output like the following:

```
The command 'astRewrite' took 178 ms.
  {
    <>alpha<>1 = <>Global<>toObject({
      a : 1
    });
    ("a" in <>alpha<>1 ? <>alpha<>1.a = 2 : a = 2);
  }
```

where the names prefixed by <> are generated by SAFE. SAFE translates the rewritten JavaScript source code to its intermediate representation format, and one can see the result by the command below:

```
safe compile sample.js
```

which shows an output like the following:

```
The command 'compile' took 382 ms.
{
  {
    <>new1<>1 = {
      a : 1
    }
    <>Global<>ignore1 = <>Global<>toObject(<>new1<>1)
    <>alpha<>2 = <>Global<>ignore1
  }
  if("a" in <>alpha<>2)
  {
    <>obj<>3 = <>Global<>toObject(<>alpha<>2)
    <>obj<>3["a"] = 2
    <>Global<>ignore2 = <>obj<>3["a"]
  }
  else
  {
    a = 2
    <>Global<>ignore2 = 2
  }
}
```

The SAFE analysis is performed on control flow graphs of programs, which can be built by the command below:

```
safe cfgBuild sample.js
```

resulting an output as follows:

```
The command 'cfgBuild' took 492 ms.
function[0] top-level {
  Entry[-1] -> [0]

  Block[0] -> [2], [1], ExitExc
    [0] noop(StartOfFile)
    [1] <>new1<>1 := alloc() @ #1
    [2] <>new1<>1["a"] := 1
    [3] <>Global<>ignore1 :=
          <>Global<>toObject(<>new1<>1) @ #2
    [4] <>alpha<>2 := <>Global<>ignore1

  Block[1] -> [3], ExitExc
    [0] assert("a" in <>alpha<>2)
    [1] <>obj<>3 := <>Global<>toObject(<>alpha<>2) @ #3
    [2] <>obj<>3["a"] := 2
    [3] <>Global<>ignore2 := <>obj<>3["a"]

  Block[2] -> [3], ExitExc
    [0] assert(! "a" in <>alpha<>2)
    [1] a := 2
    [2] <>Global<>ignore2 := 2

  Block[3] -> Exit
    [0] noop(EndOfFile)

  Exit[-2]

  ExitExc[-3]
}
```

Finally, the following command:

```
safe analyze sample.js
```

analyzes the JavaScript program in the file and shows the analysis results:

```
The command 'analyze' took 1002 ms.
** heap **
#Global -> [[Class]] : "Object"
...
#1 -> [[Class]] : "Object"
      [[Extensible]] : true
      [[Prototype]] : #Object.prototype
      "a" -> [ttt] 2
      Set(a)

** context **
##Collapsed -> [[Default]] @-> ⊥(value)
               * Outer: null
#GlobalEnv -> Top(global environment record)
              * Outer: null
              ...
this: #Global

** old address set **
mayOld: (1)
mustOld: (1)

- # of iteration: 6
- # of user functions: 1
- # of touched blocks: 6
    user blocks: 6
    modeling blocks: 0
- # of instructions: 13
```

Chapter 3

Reference manual

We describe SAFE commands and their basic usage.

3.1 SAFE commands

One can run a SAFE command as follows:

```
safe {command} [-{option}]*
     [-{phase}:{option}[={input}]]* {filename}+
```

For example, the following command analyzes JavaScript code stored in a file name "sample.js" located in the current directory without showing detailed information from the `astRewriter` phase but printing the result of the `cfgBuilder` phase into a file name "out":

```
safe analyze -silent
             -cfgBuilder:out=out sample.js
```

Each command has its own available options. The most common options are as follows:

- `-{phase}:silent`

  SAFE does not show messages during the phase.

- `-{phase}:out={out}`

  SAFE writes the result of the phase to a file `out`.

and the global options are `-silent` and `-testMode`.

The currently supported commands and their options are as follows:

- `parse -parser:out={out}`

  parses the JavaScript code in a given file.

- ```
  astRewrite -astRewriter:silent
             -astRewriter:out={out}
  ```

  generates a simplified Abstract Syntax Tree (AST) of the JavaScript code in a given file.

- ```
  compile -compiler:silent
          -compiler:out={out}
  ```

  generates an Intermediate Representation (IR) of the JavaScript code in a given file.

- ```
  cfgBuild -cfgBuilder:silent
           -cfgBuilder:out={out}
           -cfgBuilder:dot={name}
  ```

  generates a Control Flow Graph (CFG) of the JavaScript code in a given file.

  If `-cfgBuilder:dot=name` is given, SAFE writes the resulting CFG in a graph visualization format to file names `name.gv` and `name.pdf`.

- ```
  analyze -analyzer:silent
          -analyzer:out={out}
          -analyzer:console
          -analyzer:html={name}
  ```

  analyzes the JavaScript code in a given file.

  If `-analyzer:console` is given, SAFE enables a user to debug analysis results by investigating the intermediate status of the analysis.

  If `-analyzer:html=name` is given, SAFE writes the resulting CFG with states that can be investigated to file name `name.html`.

  We describe these facilities in the next section.

- `help` shows the usage of SAFE commands to the standard output.

The `parse` command parses the JavaScript code in a given file and rewrites obvious dynamic code generation into other statements without using dynamic code generation but with the same semantics. For example, the following JavaScript code

```
function f() { return 3; }
eval("f()")
```

is rewritten as follows:

```
function f() { return 3; }
f();
```

The `astRewrite` command parses the JavaScript code in a given file and rewrites its AST representation into a simpler AST. The `astRewriter` phase performs three kinds of AST transformations:

- `Hoister` lifts the declarations of functions and variables inside programs and functions up to the beginning of them.

- `Disambiguator` checks some static restrictions and renames identifiers to unique names.

- `WithRewriter` rewrites the `with` statements that do not include any dynamic code generation such as `eval` into other statements without using the `with` statement but with the same semantics.

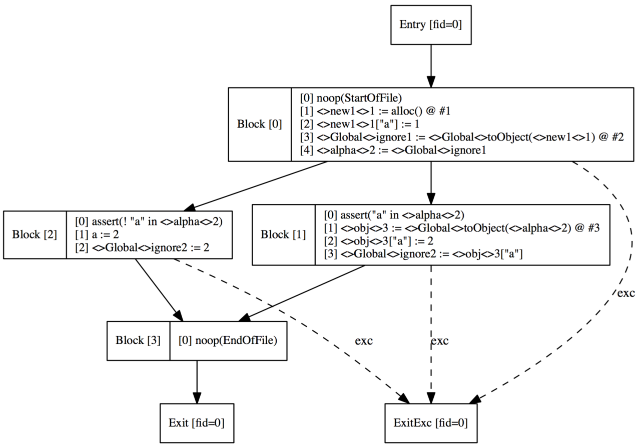
Note that building a graph visualization format of CFGs requires the `dot` program from Graphviz [3] be in your path. For example, the following command:

```
safe cfgBuild -cfgBuilder:dot=dot sample.js
```

runs the following command:

```
dot -Tpdf dot.gv -o dot.pdf
```

to produce something like the following:



3.2 SAFE analyzer debugging

When the `-analyzer:console` option is given to the `analyze` command, SAFE provides a REPL-style console debugger. For example, the following command:

```
safe analyze -analyzer:console test.js
```

shows the list of available commands for debugging and the starting point of the analysis:

```
Command list:
- help
- next       jump to the next iteration. (same as "")
- jump       Continue to analyze until the given
             iteration.
- print      Print out various information.
- result     Print out various information.
- run_insts  Run instruction by instruction.
- move       Change a current position.
- home       Reset the current position.
- run        Run until meet some break point.
- break      Add a break point.
- break-list Show the list of break points.
- break-rm   Remove a break point.
For more information, see 'help <command>'.

<function[0] top-level: Entry[-1], ()> @test.js:1:1
Iter[0] >
```

The current status is denoted as follows:

```
<function [{fid}] {fun-name}: {block-kind}[{bid}],
 {call-context}> @{filename}:{span}
Iter[{#iteration}] >
```

where `fid` and `fun-name` are the id and the name of the current function, respectively, `block-kind` and `bid` are the kind and the id of the current block, respectively, `call-context` is the call context of the current analysis, `filename` is the name of the file being analyzed, `span` is the location of the current analysis, and `#iteration` is the iteration number of the current analysis.

A block is one of the following kinds:

- Entry: the entry block of a function
- Block: a normal block with instructions
- Exit: the exit block of a function
- ExitExc: a block denoting uncaught exceptions in a function
- Call: a block denoting a function call
- AfterCall: a block receiving a return value of a function call
- AfterCatch: a block receiving uncaught exceptions after a function call
- ModelBlock: a block denoting a modeled function

The `help` command displays a list of available commands and the `help <command>` command displays the usage of the `<command>`. For example:

```
<function[0] top-level: Entry[-1], ()> @test.js:1:1
Iter[0] > help print
usage: print state(-all) ({keyword})
       print block
       print loc {LocName} ({keyword})
       print func {functionID}
       print worklist
       print ipsucc
       print trace
       print cfg
```

shows the usage of the `print` command.

The `next` command proceeds the analysis of the current block, which is the default command. For example:

```
<function[0] top-level: Entry[-1], ()> @test.js:1:1
Iter[0] >

<function[0] top-level: Block[0], ()> @test.js:1:1-7:18
Iter[1] >
```

The `jump {#iteration}` command proceeds the analysis until the given number of iterations. For example:

```
<function[0] top-level: Entry[-1], ()> @test.js:1:1
Iter[0] > jump 10

<function[0] top-level: Block[4], ()> @test.js:7:5-21:1
Iter[10] >
```

The `print` command displays the status just before analyzing the current block. We describe it in Section 3.2.1.

The `result` command displays the status after analyzing the current block:

- `result (exc-)state(-all) ({keyword})`

  It displays the state in the same way as the `print` command does, and it can additionally show the exception state generated after the analysis.

- `result (exc-)loc {LocName}`

  It finds and displays the location in the same way as the `print` command does, and it can additionally find and display the location from the exception state generated after the analysis.

The `run_insts` command shows the list of instructions in the current block, and it enables to analyze each instruction. It opens a sub-console, which provides 3 kinds of commands:

- `s` shows the state
- `q` quits the analysis
- `n` analyzes the next instruction; the default command

For example:

```
<function[0] top-level: Block[4], ()> @test.js:8:5-26:1
Iter[10] > run_insts
Block[4] -> Exit, ExitExc
  [0] shift := <>Global<>ignore6
  [1] __result1 := shift !== "x"
  [2] __expect1 := false
  [3] <>obj<>10 := <>Global<>toObject(obj) @ #13
  [4] __result2 := <>obj<>10["length"] !== 1
  [5] __expect2 := false
  [6] <>obj<>11 := <>Global<>toObject(obj) @ #14
  [7] __result3 := <>obj<>11[0] !== "y"
  [8] __expect3 := false
  [9] <>obj<>12 := <>Global<>toObject(obj) @ #15
  [10] __result4 := <>obj<>12[1] !== undefined
  [11] __expect4 := false
  [12] noop(EndOfFile)

inst: [0] shift := <>Global<>ignore6
('s': state / 'q': stop / 'n','': next)
>

inst: [1] __result1 := shift !== "x"
('s': state / 'q': stop / 'n','': next)
>
```

The `move {fid}:{{bid}|entry|exit|exitExc}` command moves the current block to the given block denoted by the id of a function, the id of a block, and the kind of the block. For example:

```
<function[0] top-level: ExitExc[-3], ()> @test.js:26:1
Iter[12] > move 0:exit
* current control point changed.

<function[0] top-level: Exit[-2], ()> @test.js:26:1
Iter[12] >
```

The `home` command moves the current block back to the original block to be analyzed. For example:

```
<function[0] top-level: Exit[-2], ()> @test.js:26:1
Iter[12] > home
* reset the current control point.

<function[0] top-level: ExitExc[-3], ()> @test.js:26:1
Iter[12] >
```

The `run` command proceeds the analysis until encountering a break point. A short-key for this command is Ctrl-d. For example:

```
<function[0] top-level: Entry[-1], ()> @test.js:1:1
Iter[0] > break 0:exit

<function[0] top-level: Entry[-1], ()> @test.js:1:1
Iter[0] > run

<function[0] top-level: Exit[-2], ()> @test.js:26:1
Iter[11] >
```

The `break` command sets up a break point at the given block. For example:

```
<function[0] top-level: Entry[-1], ()> @test.js:1:1
Iter[0] > break 0:exit

<function[0] top-level: Entry[-1], ()> @test.js:1:1
Iter[0] > run

<function[0] top-level: Exit[-2], ()> @test.js:26:1
Iter[11] >
```

The `break-list` command shows a list of blocks with break points. For example:

```
<function[0] top-level: Exit[-2], ()> @test.js:26:1
Iter[11] > break-list
* 2 break point(s).
  [0] function[0] Exit[-2]
  [1] function[0] Entry[-1]
```

The `break-rm {break-order}` command removes the break point of a given block denoted by the order in the result of `break-list`. For example:

```
<function[0] top-level: Exit[-2], ()> @test.js:26:1
Iter[11] > break-list
* 2 break point(s).
  [0] function[0] Exit[-2]
  [1] function[0] Entry[-1]
```

```
<function[0] top-level: Exit[-2], ()> @test.js:26:1
Iter[11] > break-rm 0
* break-point[0] removed.
[0] function[0] Exit[-2]

<function[0] top-level: Exit[-2], ()> @test.js:26:1
Iter[11] > break-list
* 1 break point(s).
  [0] function[0] Entry[-1]
```

3.2.1 Analyzer debugging with printing

The `print` command displays the status just before analyzing the current block.

`print state(-all) ({keyword})`

The `print state` command displays the current state, and the `print state-all` command displays the current state including all system addresses. When a keyword is given, it displays only the parts that include the keyword. For example:

```
<function[0] top-level: Exit[-2], ()> @test.js:21:1
Iter[12] > print state result
           "__result1" -> [ttf] false
           "__result2" -> [ttf] false
           "__result3" -> [ttf] false
           Set(NaN, __result3, Function,
__EvalErrLoc, URIError, pop, JSON, Error, Number,
decodeURIComponent, __SyntaxErrProtoLoc, RangeError,
__RangeErrLoc, __ArrayConstLoc, __EvalErrProtoLoc,
Boolean, ReferenceError, __RefErrLoc, obj, __BOT,
encodeURIComponent, __TypeErrProtoLoc, Array,
EvalError, __expect1, encodeURI, eval, __expect3,
isFinite, __ErrProtoLoc, Object, __TOP, Math,
__TypeErrLoc, __URIErrProtoLoc, __result1,
parseFloat, __RangeErrProtoLoc, TypeError,
<>Global<>global, __ObjConstLoc, isNaN, __URIErrLoc,
Date, __NumTop, __expect2, decodeURI, RegExp,
__BoolTop, __UInt, parseInt, __result2, __StrTop,
Infinity, SyntaxError, __RefErrProtoLoc, __Global,
<>Global<>true, __SyntaxErrLoc, undefined, String)
```

`print block`

The `print block` command displays the information of a given block. For example:

```
<function[0] top-level: Block[0], ()> @test.js:1:1-7:18
Iter[1] > print block
Block[0] -> [1], ExitExc
  [0] noop(StartOfFile)
  [1] <>Global<>ignore1 := alloc() @ #1
  [2] obj := <>Global<>ignore1
  [3] <>obj<>1 := <>Global<>toObject(obj) @ #2
  [4] <>obj<>2 := <>Global<>toObject(Array) @ #3
  [5] <>obj<>3 :=
      <>Global<>toObject(<>obj<>2["prototype"]) @ #4
  [6] <>obj<>1["pop"] := <>obj<>3["pop"]
  [7] <>obj<>4 := <>Global<>toObject(obj) @ #5
  [8] <>obj<>4[4294967294] := "x"
  [9] <>obj<>5 := <>Global<>toObject(obj) @ #6
  [10] <>obj<>5["length"] := - 1
  [11] <>obj<>6 := <>Global<>toObject(obj) @ #7
  [12] <>arguments<>7 := allocArg(0) @ #8
  [13] <>fun<>8 :=
       <>Global<>toObject(<>obj<>6["pop"]) @ #9
```

`print loc {LocName} ({keyword})`

The `print loc {LocName}` command shows the object bound at a given location in the current state. When a keyword is given, it displays only the parts that include the keyword in the object. For example:

```
<function[0] top-level: ExitExc[-3], ()> @test.js:21:1
Iter[12] > print loc #1
#1 -> [[Class]] : "Object"
      [[Extensible]] : true
      [[Prototype]] : #Object.prototype
      "4294967294" -> [ttt] "x"
      "length" -> [ttt] -1
      "pop" -> [ttt] #Array.prototype.pop
      Set(pop, length, 4294967294)
```

`print func ({functionID})`

It displays the list of functions. If a function id is given, it displays the name and the span of it. For example:

```
<function[0] top-level: ExitExc[-3], ()> @test.js:21:1
Iter[12] > print func 0
* function name: top-level
* span info.   : test.js:1:1-21:1
```

`print worklist`

It shows the work in the current worklist. For example:

```
<function[42] []Array.prototype.pop: ExitExc[-3],
 (10)> @[]Array.prototype.pop:0:0
Iter[6] > print worklist
* Worklist set
(42:ExitExc[-3], (10)), (42:Exit[-2], (10)),
(0:AfterCall[2], ()), (0:ExitExc[-3], ())
```

`print ipsucc`

It displays the information of the current inter-procedural successor. For example:

```
<function[42] []Array.prototype.pop: ExitExc[-3],
 (10)> @[]Array.prototype.pop:0:0
Iter[6] > print ipsucc
* successor map
- src: FlowSensitiveCP(ExitExc[-3],(10))
- dst: FlowSensitiveCP(AfterCatch[3],()),
 mayOld: (10, 1, 8)
mustOld: (10, 1, 8)
```

`print trace`

It shows the current call trace. For example:

```
<function[42] []Array.prototype.pop: Entry[-1],
 (10)> @[]Array.prototype.pop:0:0
Iter[3] > print trace
* Call-Context Trace
Entry[-1] of function[42] []Array.prototype.pop with (10)
 1>  [0] call(<>fun<>8, <>obj<>6, <>arguments<>7) @ #10
test.js:7:11-20 @Call[1] of function[0] top-level with ()
```

`print cfg`

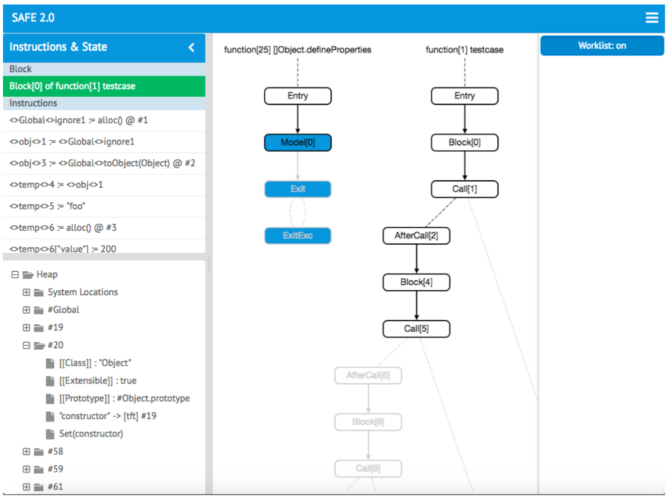
It prints the current CFG to files `cfg.gv` and `cfg.pdf`.

`print html`

It prints the current CFG and its state to the `cfg.html` file. We describe this facility in the next section.

3.2.2 Analyzer debugging with browsing

The `print html` command writes the current status into an HTML file so that a user can investigate the analysis status from a browser. Consider the following snapshot:



which shows the current CFG in the middle. Nodes in black lines denote the blocks that are analyzed, those in gray lines denote the blocks not yet being analyzed, and colored nodes denote the blocks that are currently in the worklist of the analyzer. One can toggle whether to show the nodes in the worklist by the menu button on the top right. When a user selects a block from the CFG, the list of the instructions in the block and the state just before analyzing the block are displayed on the left.

Appendix A

Appendix

This appendix provides the definitions of AST, IR, and translation rules from AST to IR in SAFE.

A.1 AST

This section describes each construct of the JavaScript language in both the BNF notation and its corresponding implementation. The implementation of AST nodes is available at:

`$SAFE_HOME/src/main/scala/kr/ac/kaist/safe/nodes/ast/`

p ::= $fd^* \; vd^* \; s^*$

```
Program(body: TopLevel)
TopLevel(fds: List[FunDecl], vds: List[VarDecl],
         stmts: List[SourceElements])
```

fd ::= $\texttt{function}\ f\texttt{((}x\texttt{,)}^*\texttt{)}\ \texttt{\{}fd^*\ vd^*\ s^*\texttt{\}}$

```
FunDecl(ftn: Functional, strict: Boolean)
Functional(fds: List[FunDecl],
           vds: List[VarDecl],
           stmts: SourceElements, name: Id,
           params: List[Id], body: String)
```

vd ::= $x\ (= e)^?$

```
VarDecl(name: Id, expr: Option[Expr],
        strict: Boolean)
```

s ::= $\texttt{\{}s^*\texttt{\}}$

```
ABlock(stmts: List[Stmt], internal: Boolean)
```

| $\texttt{var}\ vd(\texttt{,}vd)^*\texttt{;}$

```
VarStmt(vds: List[VarDecl])
```

| $\texttt{;}$

```
EmptyStmt()
```

| $e\texttt{;}$

```
ExprStmt(expr: Expr, internal: Boolean)
```

| $\texttt{if}\ (e)\ s\ (\texttt{else}\ s)^?$

```
If(cond: Expr, trueBranch: Stmt,
   falseBranch: Option[Stmt])
```

| $\texttt{switch}\ (e)\ \texttt{\{}cc^*\ (\texttt{default:}s^*)^?\ cc^*\texttt{\}}$

```
Switch(cond: Expr, frontCases: List[Case],
       defopt: Option[List[Stmt]],
       backCases: List[Case])
```

| $\texttt{do}\ s\ \texttt{while}\ (e)\texttt{;}$

```
DoWhile(body: Stmt, cond: Expr)
```

| $\texttt{while}\ (e)\ s$

```
While(cond: Expr, body: Stmt)
```

| $\texttt{for}\ (e^?\texttt{;}e^?\texttt{;}e^?)\ s$

```
For(init: Option[Expr], cond: Option[Expr],
    action: Option[Expr], body: Stmt)
```

| $\texttt{for}\ (lhs\ \texttt{in}\ e)\ s$

```
ForIn(lhs: LHS, expr: Expr, body: Stmt)
```

| $\texttt{for (var}\ vd(\texttt{,}vd)^*\texttt{;}e^?\texttt{;}e^?)\ s$

```
ForVar(vars: List[VarDecl], cond: Option[Expr],
       action: Option[Expr], body: Stmt)
```

| $\texttt{for (var}\ vd\ \texttt{in}\ e)\ s$

```
ForVarIn(vd: VarDecl, expr: Expr, body: Stmt)
```

| $\texttt{continue}\ x^?\texttt{;}$

```
Continue(target: Option[Label])
```

| $\texttt{break}\ x^?\texttt{;}$

```
Break(target: Option[Label])
```

| $\texttt{return}\ e^?\texttt{;}$

```
Return(expr: Option[Expr])
```

| $\texttt{with}\ (e)\ s$

```
With(expr: Expr, stmt: Stmt)
```

| $l\ \texttt{:}\ s$

```
LabelStmt(label: Label, stmt: Stmt)
```

| $\texttt{throw}\ e\texttt{;}$

```
Throw(expr: Expr)
```

| $\texttt{try\{}s^*\texttt{\}}(\texttt{catch(}x\texttt{)\{}s^*\texttt{\}})^?(\texttt{finally\{}s^*\texttt{\}})^?$

```
Try(body: List[Stmt], catchBlock: Option[Catch],
    fin: Option[List[Stmt]])
Catch(id: Id, body: List[Stmt])
```

| $\texttt{debugger;}$

```
Debugger()
```

cc ::= $\texttt{case}\ e\ \texttt{:}\ s^*$

```
Case(cond: Expr, body: List[Stmt])
```

e ::= $e\texttt{,}\ e$

```
ExprList(exprs: List[Expr])
```

| $e\ \texttt{?}\ e\ \texttt{:}\ e$

```
Cond(cond: Expr, trueBranch: Expr,
     falseBranch: Expr)
```

| $e\ \otimes\ e$

```
InfixOpApp(left: Expr, op: Op, right: Expr)
```

| $\ominus\ e$

```
PrefixOpApp(op: Op, right: Expr)
```

| $lhs\ \oslash$

```
UnaryAssignOpApp(lhs: LHS, op: Op)
```

| $lhs\ \odot\ e$

```
AssignOpApp(lhs: LHS, op: Op, right: Expr)
```

| lhs

```
LHS()
```

lhs ::= *lit*
    `Literal()`
  | x
    `VarRef(id: Id)`
  | `[`$(e^?$`,`$)^*$ `]`
    `ArrayExpr(elements: List[Option[Expr]])`
  | `{`$(m$`,`$)^*$`}`
    `ObjectExpr(members: List[Member]`
  | `(`e`)`
    `Parenthesized(expr: Expr)`
  | `function` $x^?$`((`x`,`$)^*$`) {`fd^* vd^* s^*`}`
    `FunExpr(ftn: Functional)`
  | *lhs*`[`e`]`
    `Bracket(obj: LHS, index: Expr)`
  | *lhs*`.`x
    `Dot(obj: LHS, member: Id)`
  | `new` *lhs*
    `New(lhs: LHS)`
  | *lhs*`((`e`,`$)^*$`)`
    `FunApp(fun: LHS, args: List[Expr])`

lit ::= `this`
    `This()`
  | `null`
    `Null()`
  | `true`
    `Bool(bool: Boolean)`
  | `false`
    `Bool(bool: Boolean)`
  | *num*
    `DoubleLiteral(text: String, num: Double)`
    `IntLiteral(intVal: BigInteger, radix: Integer)`
  | *str*
    `StringLiteral(quote: String, escaped: String)`
  | *reg*
    `RegularExpression(body: String, flag: String)`

m ::= *pr* `:` e
    `Field(prop: Property, expr: Expr)`
  | `get` *pr*`()` `{`fd^* vd^* s^*`}`
    `GetProp(prop: Property, ftn: Functional)`
  | `set` *pr*`(`x`)` `{`fd^* vd^* s^*`}`
    `SetProp(prop: Property, ftn: Functional)`

pr ::= x
    `PropId(id: Id)`
  | *str*
    `PropStr(str: String)`
  | *num*
    `PropNum(num: NumberLiteral)`

$\odot$ ::= `=` | `*=` | `/=` | `%=` | `+=` | `-=` | `[[=` | `>>=` | `>>>=` | `&=` | `^=` | `|=`
$\otimes$ ::= `&&` | `||` | `|` | `&` | `^` | `[[` | `>>` | `>>>` | `+` | `-` | `*` | `/`
  | `%` | `==` | `!=` | `===` | `!==` | `[` | `>` | `[=` | `>=`
  | `instanceof` | `in`
$\ominus$ ::= `++` | `--` | `~` | `!` | `+` | `-` | `delete` | `void` | `typeof`
$\oslash$ ::= `++` | `--`

Note that after the `Hoister` transformation of the `astRewriter` phase, ASTs should have the following invariants:

- The `expr` field of `VarDecl` should be `None`.
- `VarStmt`, `ForVar`, and `ForVarIn` should be removed.
- `StmtUnit` is an internally generated statement unit.

A.2 IR

This section describes each construct of the SAFE IR in both the BNF notation and its corresponding implementation. The implementation of IR nodes is available at:

`$SAFE_HOME/src/main/scala/kr/ac/kaist/safe/nodes/ir/`

$\underline{p}$ ::= $\underline{s}^*$
```
IRRoot(val fds: List[IRFunDecl],
       val vds: List[IRVarStmt],
       val irs: List[IRStmt])
```

$\underline{s}$ ::= $\underline{x}$ = $\underline{e}$
```
IRExprStmt(val lhs: IRId, val right: IRExpr,
           val ref: Boolean)
```
  | $\underline{x}$ = delete $\underline{x}$
```
IRDelete(val lhs: IRId, val id: IRId)
```
  | $\underline{x}$ = delete $\underline{x}$`[`$\underline{x}$`]`
```
IRDeleteProp(val lhs: IRId, val obj: IRId,
             val index: IRExpr)
```
  | $\underline{x}$`[`$\underline{x}$`]` = $\underline{e}$
```
IRStore(val obj: IRId, val index: IRExpr,
        val rhs: IRExpr)
```
  | $\underline{x}$ = `{`$(\underline{m}$`,`$)^*$`}`
```
IRObject(val lhs: IRId, val members: List[IRMember],
         val proto: Option[IRId])
```
  | $\underline{x}$ = `[`$(\underline{e}$`,`$)^*$`]`
```
IRArray(val lhs: IRId,
        val elements: List[Option[IRExpr]])
IRArgs(val lhs: IRId,
        val elements: List[Option[IRExpr]])
```
  | $\underline{x}$ = $\underline{x}$`(`$\underline{x}$`,`$\underline{x}$`)`
```
IRCall(val lhs: IRId, val fun: IRId,
       val thisB: IRId, val args: IRId)
```
  | $\underline{x}$ = $\underline{x}$`(`$\underline{x}$`(`,$\underline{x}$`)`$^?$`)`
```
IRInternalCall(val lhs: IRId, val fun: IRId,
               val first: IRExpr,
               val second: Option[IRId])
```
`toObject`, `toNumber`, `isObject`, `getBase`, `iteratorInit`, `iteratorHasNext`, `iteratorKey`
  | $\underline{x}$ = new $\underline{x}$`((`$\underline{x}$`,`$)^*$`)`
```
IRNew(val lhs: IRId, val fun: IRId,
      val args: List[IRId])
```
  | $\underline{x}$ = function $\underline{f}$`(`$\underline{x}$`,`$\underline{x}$`)` `{`$\underline{s}^*$`}`
```
IRFunExpr(val lhs: IRId, val fun: IRFunctional)
IRFunctional(val fromSource: Boolean,
             val name: IRId,
             val params: List[IRId],
             val args: List[IRStmt],
             val fds: List[IRFunDecl],
             val vds: List[IRVarStmt],
             val body: List[IRStmt])
```
  | function $\underline{f}$`(`$\underline{x}$`,`$\underline{x}$`)` `{`$\underline{s}^*$`}`
```
IRFunDecl(val ftn: IRFunctional)
```

| x = eval(e)
`IREval(val lhs: IRId, val arg: IRExpr)`
| break x
`IRBreak(val label: IRId)`
| return $e^?$
`IRReturn(val expr: Option[IRExpr])`
| with (x) s
`IRWith(val id: IRId, val stmt: IRStmt)`
| l : { s }
`IRLabelStmt(val label: IRId, val stmt: IRStmt)`
| var x
`IRVarStmt(val lhs: IRId, val fromParam: Boolean)`
| throw e
`IRThrow(val expr: IRExpr)`
| s^*
`IRSeq(val stmts: List[IRStmt])`
| if (e) then s (else s)$^?$
```
IRIf(val expr: IRExpr, val trueB: IRStmt,
    val falseb: Option[IRStmt])
```
| while (e) s
`IRWhile(val cond: IRExpr, val body: IRStmt)`
| try {s} (catch (x){s})$^?$(finally {s})$^?$
```
IRTry(val body: IRStmt, val name: Option[IRId],
     val catchB: Option[IRStmt],
     finallyB: Option[IRStmt])
```
| $\langle s^* \rangle$
`IRStmtUnit(List[IRStmt] stmts)`

e ::= $e \otimes e$
```
IRBin(val first: IRExpr, val op: IROp,
     val second: IRExpr)
```
| $\ominus e$
`IRUn(val op: IROp, val expr: IRExpr)`
| x[e]
`IRLoad(val obj: IRId, val index: IRExpr)`
| x
`IRUserId(val global: Boolean, val isWith: Boolean)`
| $\diamond$x
`IRTmpId(val global: Boolean)`
| *num*
`IRNumber(val text: String, val num: Double)`
| *str*
`IRString(val str: String)`
| true
`IRBool(val bool: Boolean)`
| false
`IRBool(val bool: Boolean)`
| undefined
`IRUndef()`
| null
`IRNull()`
| this
`IRThis()`

m ::= x : e
`IRField(val prop: IRId, val expr: IRExpr)`
| get f(x,x) {s^*}
`IRGetProp(val ftn: IRFunctional)`
| set f(x,x) {s^*}
`IRSetProp(val ftn: IRFunctional)`

The SAFE IR has the following assumptions and notations:

- We denote a list as a possibly empty, semicolon-separated sequence, enclosed by $\langle$ and $\rangle$.
- We denote a series of list appends as superscripted $*$ such as s^*.
- We abuse our notations by interchanging semicolon-separated sequences and lists.
- To denote an AST-level statement granularity in the translated IR statements, we use `IRStmtUnit` which is represented as green angle brackets $\langle\ \rangle$ in this document.
- Declarations of functions and variables are hoisted to their closest enclosing functions or the top level via the `Hoister` transformation of the `astRewriter` phase.
- Identifiers and labels that exist in the source program, except when they appear at top level or within the **`with`** statement, are already disambiguated via the `Disambiguator` transformation of the `astRewriter` phase so that they have unique names.

A.3 AST to IR

This section describes the SAFE translation rules from AST to IR, whose implementation is available at:

`$SAFE_HOME/src/main/scala/kr/ac/kaist/safe/compiler/Translator.scala`

- We use Σ to disambiguate the generated labels and temporary variables in the AST to IR translation. For the presentation brevity, we simply add the newly generated names to Σ.
  - In the actual implementation, we need to create a unique id for each generated name and add the binding information from the general name to the unique id to Σ. For example, when we say "Σ; $\diamond$break", we actually create a unique id for $\diamond$break, say $\diamond$break$_{42}$, and add it to Σ as Σ; $\diamond$break $\mapsto$ $\diamond$break$_{42}$. When we look up the environment by Σ($\diamond$break), the unique $\diamond$break$_{42}$ is returned.
  - In the scope when the generated name is created, we don't add it to the environment but use the unique id instead of the general name. For example, when we say "$\diamond$eq = Σ($\diamond$val)===$\diamond$break;", we create a unique id for $\diamond$eq, say $\diamond$eq$_{910157}$, and it is acually "$\diamond$eq$_{910157}$ = Σ($\diamond$val)===$\diamond$break$_{42}$;".
  - To be clear, we use blue for the binding sites of such names and red for the use sites of such names.
- We denote a fresh variable name as $\diamond$ and its variants.
- We use the following:
  ===, $\diamond$toObject, $\diamond$toNumber, $\diamond$isObject, $\diamond$iteratorInit, $\diamond$iteratorHasNext, $\diamond$iteratorNext, $\diamond$global, $\diamond$getBase
- To reduce the number of temporary variables, we use global variables to denote constants such as 1 and true which is represented in green 1 and true in this document.
- We wrap a possibly identical assignment with a box so that the actual implementation, `Translator`, can eliminate identical assignments.

Here are the types of the environment used to disambiguate the generated labels and temporary variables and the translation functions for different language constructs:

Σ : `Env`
$ast2ir_p$:
```
Program -> IRRoot
```
$ast2ir_{fd}$:
```
FunDecl -> Env -> IRFunDecl
```
$ast2ir_{vd}$:
```
VarDecl -> Env -> IRVarStmt
```
$ast2ir_s$:
```
Stmt -> Env -> IRStmt
```
$ast2ir_{case}$:
```
List[Case] * Option[List[Stmt]] * List[Case]
     -> Env ->List[Option[Expr] * IRId] -> IRStmt
```
$ast2ir_{scond}$:
```
List[Option[Expr] * IRId] -> Env -> IRStmt
```
$ast2ir_{lval}$:
```
Expr -> Env -> List[IRStmt] -> IRExpr -> boolean
     -> List[IRStmt] * IRExpr
```
$ast2ir_e$:
```
Expr -> Env -> IRId -> List[IRStmt] * IRExpr
```
$ast2ir_{lhs}$:
```
LHS -> Env -> IRId -> List[IRStmt] * IRExpr
```
$ast2ir_{lit}$:
```
LIT -> Env -> IRId -> List[IRStmt] * IRExpr
```
$ast2ir_m$:
```
Member -> Env -> IRId -> List[IRStmt] * IRMember
```
$ast2ir_{pr}$:
```
Property -> IRId
```

```
⊙ ::= * | / | % | + | - | [[ | >> | >>> | & | ^ | |
⊖ ::= ~ | ! | + | - | delete | void | typeof
⊗ ::= | | & | ^ | [[ | >> | >>> | + | - | * | / | % | ==
    | != | === | !== | [ | > | [= | >= | instanceof | in
```

$ast2ir_p[\![fd^*\ vd^*\ s^*]\!]$ =
$\langle (ast2ir_{fd}[\![fd]\!](\langle\rangle))^*\ (ast2ir_{vd}[\![vd]\!](\langle\rangle))^*\ (ast2ir_s[\![s]\!](\langle\rangle))^* \rangle$

$ast2ir_{fd}[\![$`function` $f((x,)^*)$ `{`$fd^*\ vd^*\ s^*$`}`$]\!](\Sigma)$ =
function $\underline{f}$(◇this, ◇arguments){
$(ast2ir_{fd}[\![fd]\!](\Sigma))^*$
(var $\underline{x_i}$)*
$(ast2ir_{vd}[\![vd]\!](\Sigma))^*$
($\underline{x_i}$ = ◇arguments["i"])*
where $\underline{x_i}$ is not the name of any of fd
$(ast2ir_s[\![s]\!](\Sigma;$ ◇this; ◇arguments$))^*$}

A function always receives explicit "this" and "arguments" arguments so that the desugaring of `this` *and* `arguments` *is correct. Currently, "arguments" denotes copies of the arguments instead of their aliases.*

$ast2ir_{vd}[\![$`var` $x]\!](\Sigma)$ =
var $\underline{x}$

$ast2ir_s[\![$`{`s^*`}`$]\!](\Sigma)$ =
$\langle (ast2ir_s[\![s]\!](\Sigma))^* \rangle$

$ast2ir_s[\![$`;`$]\!](\Sigma)$ =
$\langle\rangle$

$ast2ir_s[\![e$`;`$]\!](\Sigma)$ =
LET $(\underline{s}^*, \underline{e}) = ast2ir_e[\![e]\!](\Sigma)(\underline{◇\_})$
IN $\langle \underline{s}^*;\ \boxed{\underline{◇\_} = \underline{e}} \rangle$

$ast2ir_s[\![$`if (`e_1`&&`$\cdots$`&&`e_n`)` s_1 (`else` $s_2)^?]\!](\Sigma)$ =
LET $(\underline{s}_i^*, \underline{e}_i) = ast2ir_e[\![e_i]\!](\Sigma)$(◇new$_i$) *where* $1 \le i \le n$
IN $\langle \underline{s}_1^*$;
◇label : {
if ($\underline{e}_1$)
then $\langle \underline{s}_2^*$; $\cdots$;
if ($\underline{e}_{n-1}$)
then $\langle \underline{s}_n^*$;
if ($\underline{e}_n$)
then {$ast2ir_s[\![s_1]\!](\Sigma)$; break ◇label}$\rangle \cdots \rangle$;
$(ast2ir_s[\![s_2]\!](\Sigma))^?$}$\rangle$

Candidate for optimization
$ast2ir_s[\![$`if (`e_1`||`e_2`)` s_1 (`else` $s_2)^?]\!](\Sigma)$ =
LET $(\underline{s}_1^*, \underline{e}_1) = ast2ir_e[\![e_1]\!](\Sigma)$(◇new$_1$)
$(\underline{s}_2^*, \underline{e}_2) = ast2ir_e[\![e_2]\!](\Sigma)$(◇new$_2$)
IN $\langle \underline{s}_1^*$;
◇label$_2$: {
◇label$_1$: {
if ($\underline{e}_1$)
then break ◇label$_1$; $\underline{s}_2^*$;
if ($\underline{e}_2$) then break ◇label$_1$;
$(ast2ir_s[\![s_2]\!](\Sigma)$; break$)^?$ ◇label$_2$
}; $ast2ir_s[\![s_1]\!](\Sigma)$}$\rangle$

$ast2ir_s[\![$`if (`e`)` s_1 (`else` $s_2)^?]\!](\Sigma)$ =
LET $(\underline{s}^*, \underline{e}) = ast2ir_e[\![e]\!](\Sigma)$(◇new)
IN $\langle \underline{s}^*$; if ($\underline{e}$) then $ast2ir_s[\![s_1]\!](\Sigma)$ (else $ast2ir_s[\![s_2]\!](\Sigma))^?\rangle$

$ast2ir_s[\![$`switch (`e`) {`cc_1^* (`default:`$s^*)^?$ cc_2^*`}`$]\!](\Sigma)$ =
LET $(\underline{s}^*, \underline{e}) = ast2ir_e[\![e]\!](\Sigma)$(◇val)
IN $\langle$◇break : {
$\underline{s}^*$; $\boxed{◇\text{val} = \underline{e};}$
$ast2ir_{case}[\![(\text{rev } cc_2^*)(s^*)^?(\text{rev } cc_1^*)]\!](\Sigma;$ ◇break; ◇val$)$}$\rangle$

$ast2ir_{case}[\![($`case` e : $s_1^*) :: cc_2^*\ (s_2^*)^?\ cc_1^*]\!](\Sigma)(c^*)$ =
$\langle$◇label : {$ast2ir_{case}[\![cc_2^*\ (s_2^*)^?\ cc_1^*]\!](\Sigma)((e,$ ◇label$) :: c^*)$};
$(ast2ir_s[\![s_1]\!](\Sigma))^*\rangle$

$ast2ir_{case}[\![()\ (s^*)^?\ cc_1^*]\!](\Sigma)(c^*)$ =
$\langle$◇label : {$ast2ir_{case}[\![()\ ()\ cc_1^*]\!](\Sigma)(c^*@[((),$ ◇label$)])$};
$((ast2ir_s[\![s]\!](\Sigma))^*)^?\rangle$

$ast2ir_{case}[\![()\ ()\ ($`case` e : $s^*) :: cc_1^*]\!](\Sigma)(c^*)$ =
$\langle$◇label : {$ast2ir_{case}[\![()\ ()\ cc_1^*]\!](\Sigma)((e,$ ◇label$) :: c^*)$};
$(ast2ir_s[\![s]\!](\Sigma))^*\rangle$

$ast2ir_{case}[\![()\ ()\ ()]\!](\Sigma)((e, \underline{l})^*)$ =
$\langle ast2ir_{scond}[\![(e, \underline{l})^*]\!](\Sigma)$;
break Σ(◇break)$\rangle$

$ast2ir_{scond}[\![(e, \underline{l}) :: (c^*)]\!](\Sigma)$ =
LET $(\underline{s}^*, \underline{e}) = ast2ir_e[\![e]\!](\Sigma)$(◇cond)
IN $\langle \underline{s}^*$;
if (Σ(◇val) === $\underline{e}$) then break $\underline{l}$ else $ast2ir_{scond}[\![c^*]\!](\Sigma)\rangle$

$ast2ir_{scond}[\![((),\underline{l})]\!](\Sigma) \ =$
$\langle \mathsf{break}\ \underline{l}\rangle$

$ast2ir_{scond}[\![()]\!](\Sigma) \ =$
$\langle\rangle$
Where c is either $(e,\underline{l})$ or $((),\underline{l})$.

$ast2ir_s[\![\texttt{do}\ s\ \texttt{while}\ \texttt{(}e\texttt{);}]\!](\Sigma) \ =$
LET $(\underline{s}^*, \underline{e}) = ast2ir_e[\![e]\!](\Sigma)(\underline{\diamond\mathsf{new}_1})$
IN $\langle\underline{\diamond\mathsf{break}}$: {
    $\underline{\diamond\mathsf{continue}}$: {$ast2ir_s[\![s]\!](\Sigma; \underline{\diamond\mathsf{break}}; \underline{\diamond\mathsf{continue}})$};
    $\underline{s}^*$;
    while ($\underline{e}$) {
        $\underline{\diamond\mathsf{continue}}$: {$ast2ir_s[\![s]\!](\Sigma; \underline{\diamond\mathsf{break}}; \underline{\diamond\mathsf{continue}})$};
        $\underline{s}^*$;
    }
}$\rangle$

$ast2ir_s[\![\texttt{while}\ \texttt{(}e\texttt{)}\ s]\!](\Sigma) \ =$
LET $(\underline{s}^*, \underline{e}) = ast2ir_e[\![e]\!](\Sigma)(\underline{\diamond\mathsf{new}_1})$
IN $\langle\underline{\diamond\mathsf{break}}$: {
    $\underline{s}^*$;
    while ($\underline{e}$) {
        $\underline{\diamond\mathsf{continue}}$: {$ast2ir_s[\![s]\!](\Sigma; \underline{\diamond\mathsf{break}}; \underline{\diamond\mathsf{continue}})$};
        $\underline{s}^*$;
    }
}$\rangle$

$ast2ir_s[\![\texttt{for}\ \texttt{(}e_1^?\texttt{;;}e_3^?\texttt{)}\ s]\!](\Sigma) \ =$
LET $((\underline{s}_1^*, \underline{e}_1) = ast2ir_e[\![e_1]\!](\Sigma)(\underline{\diamond\_}))^?$
    $((\underline{s}_3^*, \underline{e}_3) = ast2ir_e[\![e_3]\!](\Sigma)(\underline{\diamond\_}))^?$
IN $\langle\underline{\diamond\mathsf{break}}$: {
    $(\underline{s}_1^*; \boxed{\underline{\diamond\_} = \underline{e}_1})^?$
    while (true) {
        $\underline{\diamond\mathsf{continue}}$: {$ast2ir_s[\![s]\!](\Sigma; \underline{\diamond\mathsf{break}}; \underline{\diamond\mathsf{continue}})$};
        $(\underline{s}_3^*; \boxed{\underline{\diamond\_} = \underline{e}_3})^?$
    }
}$\rangle$

$ast2ir_s[\![\texttt{for}\ \texttt{(}e_1^?\texttt{;}e_2\texttt{;}e_3^?\texttt{)}\ s]\!](\Sigma) \ =$
LET $((\underline{s}_1^*, \underline{e}_1) = ast2ir_e[\![e_1]\!](\Sigma)(\underline{\diamond\_}))^?$
    $(\underline{s}_2^*, \underline{e}_2) = ast2ir_e[\![e_2]\!](\Sigma)(\underline{\diamond\mathsf{new}_2})$
    $((\underline{s}_3^*, \underline{e}_3) = ast2ir_e[\![e_3]\!](\Sigma)(\underline{\diamond\_}))^?$
IN $\langle\underline{\diamond\mathsf{break}}$: {
    $(\underline{s}_1^*; \boxed{\underline{\diamond\_} = \underline{e}_1})^?$
    $\underline{s}_2^*$;
    while ($\underline{e}_2$) {
        $\underline{\diamond\mathsf{continue}}$: {$ast2ir_s[\![s]\!](\Sigma; \underline{\diamond\mathsf{break}}; \underline{\diamond\mathsf{continue}})$};
        $(\underline{s}_3^*; \boxed{\underline{\diamond\_} = \underline{e}_3})^?$
        $\underline{s}_2^*$;
    }
}$\rangle$

$ast2ir_s[\![\texttt{for}\ \texttt{(}lhs\ \texttt{in}\ e\texttt{)}\ s]\!](\Sigma) \ =$
LET $(\underline{s}^*, \underline{e}) = ast2ir_e[\![e]\!](\Sigma)(\underline{\diamond\mathsf{new}_1})$
IN $\langle\underline{\diamond\mathsf{break}}$: {
    $\underline{s}^*$;
    $\underline{\diamond\mathsf{obj}}$ = $\diamond$toObject($\underline{e}$);
    $\underline{\diamond\mathsf{iterator}}$ = $\diamond$iteratorInit($\underline{\diamond\mathsf{obj}}$);
    $\underline{\diamond\mathsf{cond}_1}$ = $\diamond$iteratorHasNext($\underline{\diamond\mathsf{obj}}$, $\underline{\diamond\mathsf{iterator}}$);
    while ($\underline{\diamond\mathsf{cond}_1}$) {
        $\underline{\diamond\mathsf{key}}$ = $\diamond$iteratorNext($\underline{\diamond\mathsf{obj}}$, $\underline{\diamond\mathsf{iterator}}$);
        $ast2ir_{lval}[\![lhs]\!](\Sigma)(;\ \underline{\diamond\mathsf{key}})(\mathrm{false}).\_1$;
        $\underline{\diamond\mathsf{continue}}$:{$ast2ir_s[\![s]\!](\Sigma; \underline{\diamond\mathsf{break}}; \underline{\diamond\mathsf{continue}})$};
        $\underline{\diamond\mathsf{cond}_1}$ = $\diamond$iteratorHasNext($\underline{\diamond\mathsf{obj}}$, $\underline{\diamond\mathsf{iterator}}$);
    }
}$\rangle$

$ast2ir_s[\![\texttt{continue;}]\!](\Sigma) \ =$
$\langle\mathsf{break}\ \Sigma(\underline{\diamond\mathsf{continue}})\rangle$

$ast2ir_s[\![\texttt{continue}\ l\texttt{;}]\!](\Sigma) \ =$
$\langle\mathsf{break}\ \Sigma(\underline{l})\rangle$

$ast2ir_s[\![\texttt{break;}]\!](\Sigma) \ =$
$\langle\mathsf{break}\ \Sigma(\underline{\diamond\mathsf{break}})\rangle$

$ast2ir_s[\![\texttt{break}\ l\texttt{;}]\!](\Sigma) \ =$
$\langle\mathsf{break}\ \underline{l}\rangle$

$ast2ir_s[\![\texttt{return;}]\!](\Sigma) \ =$
$\langle\mathsf{return}\rangle$

$ast2ir_s[\![\texttt{return}\ e\texttt{;}]\!](\Sigma) \ =$
LET $(\underline{s}^*, \underline{e}) = ast2ir_e[\![e]\!](\Sigma)(\underline{\diamond\mathsf{new}_1})$
IN $\langle\underline{s}^*;\ \mathsf{return}\ \underline{e}\rangle$

$ast2ir_s[\![\texttt{with}\ \texttt{(}e\texttt{)}\ s]\!](\Sigma) \ =$
LET $(\underline{s}^*, \underline{e}) = ast2ir_e[\![e]\!](\Sigma)(\underline{\diamond\mathsf{new}_1})$
IN $\langle\underline{s}^*$;
    $\underline{\diamond\mathsf{new}_2}$ = $\diamond$toObject($\underline{e}$);
    with ($\underline{\diamond\mathsf{new}_2}$) $ast2ir_s[\![s]\!](\Sigma)\rangle$

$ast2ir_s[\![l\ \texttt{:}\ s]\!](\Sigma) \ =$
$\langle\underline{l}$: { $ast2ir_s[\![s]\!](\Sigma; \underline{l})$ }$\rangle$

$ast2ir_s[\![\texttt{throw}\ e\texttt{;}]\!](\Sigma) \ =$
LET $(\underline{s}^*, \underline{e}) = ast2ir_e[\![e]\!](\Sigma)(\underline{\diamond\mathsf{new}_1})$
IN $\langle\underline{s}^*;\ \mathsf{throw}\ \underline{e}\rangle$

$ast2ir_s[\![\texttt{try \{}s_1^*\texttt{\} (catch(}x\texttt{)\{}s_2^*\texttt{\})}^?\texttt{(finally \{}s_3^*\texttt{\})}^?]\!](\Sigma) \ =$
$\langle$try {$(ast2ir_s[\![s_1]\!](\Sigma))^*$}
 $(\texttt{catch(}\underline{x}\texttt{)\{}(ast2ir_s[\![s_2]\!](\Sigma))^*\texttt{\})}^?$
 $(\texttt{finally \{}(ast2ir_s[\![s_3]\!](\Sigma))^*\texttt{\})}^?\rangle$

$ast2ir_s[\![\texttt{debugger;}]\!](\Sigma) \ =$
$\langle\rangle$

$$\mathit{ast2ir}_{lval}[\![(e)]\!](\Sigma)(\underline{s}^*;\underline{e}')(\text{keepOld}) \;=\; \mathit{ast2ir}_{lval}[\![e]\!](\Sigma)(\underline{s}^*;\underline{e}')(\text{keepOld})$$

$$\mathit{ast2ir}_{lval}[\![x]\!](\Sigma)(\underline{s}^*;\underline{e})(\text{keepOld}) \;=$$
$$\text{IF keepOld THEN } (\langle \underline{\diamond\mathsf{old}}\ \texttt{=}\ \underline{x};\ \underline{s}^*;\ \underline{x}\ \texttt{=}\ \underline{e}\rangle, \underline{x})$$
$$\text{ELSE } \langle \underline{s}^*;\ \underline{x}\ \texttt{=}\ \underline{e}\rangle$$

$$\mathit{ast2ir}_{lval}[\![\mathit{lhs}\texttt{.}x]\!](\Sigma)(\underline{s}^*;\underline{e})(\text{keepOld}) \;=\; \mathit{ast2ir}_{lval}[\![\mathit{lhs}\texttt{["}x\texttt{"]}]\!](\Sigma)(\underline{s}^*;\underline{e})(\text{keepOld})$$

$$\mathit{ast2ir}_{lval}[\![\mathit{lhs}\texttt{[}e\texttt{]}]\!](\Sigma)(\underline{s}^*;\underline{e}')(\text{keepOld}) \;=$$
$$\text{LET } (\underline{s}_1^*, \underline{e}_1) = \mathit{ast2ir}_{lhs}[\![\mathit{lhs}]\!](\Sigma)(\underline{\diamond\mathsf{obj}_1})$$
$$(\underline{s}_2^*, \underline{e}_2) = \mathit{ast2ir}_e[\![e]\!](\Sigma)(\underline{\diamond\mathsf{field}_1})$$
$$\text{IN IF keepOld}$$
$$\text{THEN } ((\langle \underline{s}_1^*\texttt{;}\underline{\diamond\mathsf{obj}}\ \texttt{=}\ \diamond\mathsf{toObject}(\underline{e}_1)\texttt{;}\underline{s}_2^*\texttt{;}\ \underline{\diamond\mathsf{old}}\ \texttt{=}\ \underline{\diamond\mathsf{obj}}\texttt{[}\underline{e}_2\texttt{]}\texttt{;}\underline{s}^*\texttt{;}\underline{\diamond\mathsf{obj}}\texttt{[}\underline{e}_2\texttt{]}\ \texttt{=}\ \underline{e}'\rangle, \underline{\diamond\mathsf{obj}}\texttt{[}\underline{e}_2\texttt{]})$$
$$\text{ELSE } ((\langle \underline{s}_1^*\texttt{;}\underline{\diamond\mathsf{obj}}\ \texttt{=}\ \diamond\mathsf{toObject}(\underline{e}_1)\texttt{;}\underline{s}_2^*\texttt{;}\ \underline{s}^*\texttt{;}\underline{\diamond\mathsf{obj}}\texttt{[}\underline{e}_2\texttt{]}\ \texttt{=}\ \underline{e}'\rangle, \underline{\diamond\mathsf{obj}}\texttt{[}\underline{e}_2\texttt{]})$$

$$\mathit{ast2ir}_{lval}[\![e]\!](\Sigma)(\underline{s}^*;\underline{e})(\text{keepOld}) \;=$$
Warning: ReferenceError!

$$\mathit{ast2ir}_e[\![e_1\ \texttt{,}\ e_2]\!](\Sigma)(\underline{x}) \;=$$
$$\text{LET } (\underline{s}_1^*, \underline{e}_1) = \mathit{ast2ir}_e[\![e_1]\!](\Sigma)(\underline{\diamond\mathsf{y}})$$
$$(\underline{s}_2^*, \underline{e}_2) = \mathit{ast2ir}_e[\![e_2]\!](\Sigma)(\underline{x})$$
$$\text{IN } (\underline{s}_1^*\texttt{;}\underline{\diamond\mathsf{y}}\ \texttt{=}\ \underline{e}_1\texttt{;}\underline{s}_2^*, \underline{e}_2)$$

Candidate for optimization

$$\mathit{ast2ir}_e[\![e_a\texttt{\&\&}e_b\ \texttt{?}\ e_2\ \texttt{:}\ e_3]\!](\Sigma)(\underline{x}) \;=$$
$$\text{LET } (\underline{s}_a^*, \underline{e}_a) = \mathit{ast2ir}_e[\![e_a]\!](\Sigma)(\underline{\diamond\mathsf{new}_a})$$
$$(\underline{s}_b^*, \underline{e}_b) = \mathit{ast2ir}_e[\![e_b]\!](\Sigma)(\underline{\diamond\mathsf{new}_b})$$
$$(\underline{s}_2^*, \underline{e}_2) = \mathit{ast2ir}_e[\![e_2]\!](\Sigma)(\underline{x})$$
$$(\underline{s}_3^*, \underline{e}_3) = \mathit{ast2ir}_e[\![e_3]\!](\Sigma)(\underline{x})$$
$$\text{IN } (\underline{s}_a^*\texttt{;}\ \underline{\diamond\mathsf{label}}\ \texttt{: \{ if } (\underline{e}_a)\ \texttt{then}\ \langle\underline{s}_b^*\texttt{; if } (\underline{e}_b)\ \texttt{then \{}\underline{s}_2^*\texttt{; }\boxed{\underline{x}\ \texttt{=}\ \underline{e}_2}\texttt{; break }\underline{\diamond\mathsf{label}}\texttt{\}}\rangle\texttt{;}\ \underline{s}_3^*\texttt{; }\boxed{\underline{x}\ \texttt{=}\ \underline{e}_3}\texttt{\}}, \underline{x})$$

Candidate for optimization

$$\mathit{ast2ir}_e[\![e_a\texttt{||}e_b\ \texttt{?}\ e_2\ \texttt{:}\ e_3]\!](\Sigma)(\underline{x}) \;=$$
$$\text{LET } (\underline{s}_a^*, \underline{e}_a) = \mathit{ast2ir}_e[\![e_a]\!](\Sigma)(\underline{\diamond\mathsf{new}_a})$$
$$(\underline{s}_b^*, \underline{e}_b) = \mathit{ast2ir}_e[\![e_b]\!](\Sigma)(\underline{\diamond\mathsf{new}_b})$$
$$(\underline{s}_2^*, \underline{e}_2) = \mathit{ast2ir}_e[\![e_2]\!](\Sigma)(\underline{x})$$
$$(\underline{s}_3^*, \underline{e}_3) = \mathit{ast2ir}_e[\![e_3]\!](\Sigma)(\underline{x})$$
$$\text{IN } (\underline{s}_a^*\texttt{;}\ \underline{\diamond\mathsf{label}_2}\ \texttt{: \{}\ \underline{\diamond\mathsf{label}_1}\ \texttt{: \{ if } (\underline{e}_a)\ \texttt{then break }\underline{\diamond\mathsf{label}_1}\texttt{;}\underline{s}_b^*\texttt{; if } (\underline{e}_b)\ \texttt{then break }\underline{\diamond\mathsf{label}_1}\texttt{;}\ \underline{s}_3^*\texttt{; }\boxed{\underline{x}\ \texttt{=}\ \underline{e}_3}\texttt{; break }\underline{\diamond\mathsf{label}_2}\ \texttt{\};}\ \underline{s}_2^*\texttt{; }\boxed{\underline{x}\ \texttt{=}\ \underline{e}_2}\texttt{\}}, \underline{x})$$

$$\mathit{ast2ir}_e[\![e_1\ \texttt{?}\ e_2\ \texttt{:}\ e_3]\!](\Sigma)(\underline{x}) \;=$$
$$\text{LET } (\underline{s}_1^*, \underline{e}_1) = \mathit{ast2ir}_e[\![e_1]\!](\Sigma)(\underline{\diamond\mathsf{new}_1})$$
$$(\underline{s}_2^*, \underline{e}_2) = \mathit{ast2ir}_e[\![e_2]\!](\Sigma)(\underline{x})$$
$$(\underline{s}_3^*, \underline{e}_3) = \mathit{ast2ir}_e[\![e_3]\!](\Sigma)(\underline{x})$$
$$\text{IN } (\underline{s}_1^*\texttt{;if } (\underline{e}_1)\ \texttt{then \{}\underline{s}_2^*\texttt{; }\boxed{\underline{x}\ \texttt{=}\ \underline{e}_2}\texttt{\} else \{}\underline{s}_3^*\texttt{; }\boxed{\underline{x}\ \texttt{=}\ \underline{e}_3}\texttt{\}}, \underline{x})$$

$$\mathit{ast2ir}_e[\![\mathit{lhs}\ \texttt{=}\ e]\!](\Sigma)(\underline{x}) \;=$$
$$\text{LET } (\underline{s}^*, \underline{e}) = \mathit{ast2ir}_e[\![e]\!](\Sigma)(\underline{x})$$
$$\text{IN IF } \underline{e} \text{ contains } \mathit{lhs}$$
$$\text{THEN } \mathit{ast2ir}_{lval}[\![\mathit{lhs}]\!](\Sigma)(\ \underline{s}^*;\ \underline{e}\ )(\text{false})$$
$$\text{ELSE } (\mathit{ast2ir}_{lval}[\![\mathit{lhs}]\!](\Sigma)(\ \underline{s}^*;\ \underline{e}\ )(\text{false}).\_1, \underline{e})$$

$$\mathit{ast2ir}_e[\![\mathit{lhs}\odot\texttt{=}\ e]\!](\Sigma)(\underline{x}) \;=$$
$$\text{LET } (\underline{s}^*, \underline{e}) = \mathit{ast2ir}_e[\![e]\!](\Sigma)(\underline{\diamond\mathsf{y}})$$
$$\text{IN } (\mathit{ast2ir}_{lval}[\![\mathit{lhs}]\!](\Sigma)(\ \underline{s}^*;\ \underline{\diamond\mathsf{old}}\odot\underline{e})(\text{true}).\_1, \underline{\diamond\mathsf{old}}\odot\underline{e})$$

$$\mathit{ast2ir}_e[\![\texttt{++}e]\!](\Sigma)(\underline{x}) \;=$$
$$(\mathit{ast2ir}_{lval}[\![e]\!](\Sigma)(\underline{\diamond\mathsf{new}}\ \texttt{=}\ \diamond\mathsf{toNumber}(\underline{\diamond\mathsf{old}})\texttt{;}\ \underline{\diamond\mathsf{new}}\ \texttt{+1})(\text{true}).\_1, \underline{\diamond\mathsf{new}}\ \texttt{+1})$$

$$\mathit{ast2ir}_e[\![\texttt{--}e]\!](\Sigma)(\underline{x}) \;=$$
$$(\mathit{ast2ir}_{lval}[\![e]\!](\Sigma)(\underline{\diamond\mathsf{new}}\ \texttt{=}\ \diamond\mathsf{toNumber}(\underline{\diamond\mathsf{old}})\texttt{;}\ \underline{\diamond\mathsf{new}}\ \texttt{-1})(\text{true}).\_1, \underline{\diamond\mathsf{new}}\ \texttt{-1})$$

$$\mathit{ast2ir}_e[\![\texttt{delete}\ x]\!](\Sigma)(\underline{y}) \;=\; (\langle\underline{y}\ \texttt{= delete}\ x\rangle, \underline{y})$$

$$\mathit{ast2ir}_e[\![\texttt{delete}\ \texttt{(}x\texttt{)}]\!](\Sigma)(\underline{y}) \;=\; (\langle\underline{y}\ \texttt{= delete}\ x\rangle, \underline{y})$$

$$\mathit{ast2ir}_e[\![\texttt{delete}\ \mathit{lhs}\texttt{.}x]\!](\Sigma)(\underline{y}) \;=\; \mathit{ast2ir}_e[\![\texttt{delete}\ \mathit{lhs}\texttt{["}x\texttt{"]}]\!](\Sigma)(\underline{y})$$

$$\mathit{ast2ir}_e[\![\texttt{delete}\ \mathit{lhs}\texttt{[}e\texttt{]}]\!](\Sigma)(\underline{x}) \;=$$
$$\text{LET } (\underline{s}_1^*, \underline{e}_1) = \mathit{ast2ir}_{lhs}[\![\mathit{lhs}]\!](\Sigma)(\underline{\diamond\mathsf{obj}_1})$$
$$(\underline{s}_2^*, \underline{e}_2) = \mathit{ast2ir}_e[\![e]\!](\Sigma)(\underline{\diamond\mathsf{field}_1})$$
$$\text{IN } (\underline{s}_1^*\texttt{;}\underline{\diamond\mathsf{obj}}\ \texttt{=}\ \diamond\mathsf{toObject}(\underline{e}_1)\texttt{;}\underline{s}_2^*\texttt{;}\ \underline{x}\ \texttt{= delete}\ \underline{\diamond\mathsf{obj}}\texttt{[}\underline{e}_2\texttt{]}, \underline{x})$$

$$\mathit{ast2ir}_e[\![\texttt{delete}\ e]\!](\Sigma)(\underline{x}) \;=$$
$$\text{LET } (\underline{s}^*, \underline{e}) = \mathit{ast2ir}_e[\![e]\!](\Sigma)(\underline{\diamond\mathsf{y}})$$
$$\text{IN } (\underline{s}^*\texttt{;}\underline{\diamond\_}\ \texttt{=}\ \underline{e}, \texttt{true})$$

$$\mathit{ast2ir}_e[\![\ominus e]\!](\Sigma)(\underline{x}) \;=$$
$$\text{LET } (\underline{s}^*, \underline{e}) = \mathit{ast2ir}_e[\![e]\!](\Sigma)(\underline{\diamond\mathsf{y}})$$
$$\text{IN } (\underline{s}^*, \ominus\underline{e})$$

$$\mathit{ast2ir}_e[\![\mathit{lhs}\texttt{++}]\!](\Sigma)(\underline{x}) \;=$$
$$(\mathit{ast2ir}_{lval}[\![\mathit{lhs}]\!](\Sigma)(\underline{\diamond\mathsf{new}}\ \texttt{=}\ \diamond\mathsf{toNumber}(\underline{\diamond\mathsf{old}})\texttt{;}\ \underline{\diamond\mathsf{new}}\ \texttt{+1})(\text{true}).\_1, \underline{\diamond\mathsf{new}})$$

$$\mathit{ast2ir}_e[\![\mathit{lhs}\texttt{--}]\!](\Sigma)(\underline{x}) \;=$$
$$(\mathit{ast2ir}_{lval}[\![\mathit{lhs}]\!](\Sigma)(\underline{\diamond\mathsf{new}}\ \texttt{=}\ \diamond\mathsf{toNumber}(\underline{\diamond\mathsf{old}})\texttt{;}\ \underline{\diamond\mathsf{new}}\ \texttt{-1})(\text{true}).\_1, \underline{\diamond\mathsf{new}})$$

Candidate for optimization

$$\begin{array}{l} ast2ir_e[\![e_1\,\texttt{\&\&}\,e_2]\!](\Sigma)(\underline{x}) \ = \\ \text{LET } (\underline{s}_1^*, \underline{e}_1) = ast2ir_e[\![e_1]\!](\Sigma)(\underline{\diamond\mathsf{y}}) \\ \qquad (\underline{s}_2^*, \underline{e}_2) = ast2ir_e[\![e_2]\!](\Sigma)(\underline{\diamond\mathsf{z}}) \\ \text{IN } \quad (\underline{s}_1^*; \mathsf{if}\ (\underline{e}_1)\ \mathsf{then}\ \underline{s}_2^*;\ \underline{x} = \underline{e}_2\ \mathsf{else}\ \underline{x} = \underline{e}_1, \underline{x}) \end{array}$$

Candidate for optimization

$$\begin{array}{l} ast2ir_e[\![e_1\,\texttt{||}\,e_2]\!](\Sigma)(\underline{x}) \ = \\ \text{LET } (\underline{s}_1^*, \underline{e}_1) = ast2ir_e[\![e_1]\!](\Sigma)(\underline{\diamond\mathsf{y}}) \\ \qquad (\underline{s}_2^*, \underline{e}_2) = ast2ir_e[\![e_2]\!](\Sigma)(\underline{\diamond\mathsf{z}}) \\ \text{IN } \quad (\underline{s}_1^*; \mathsf{if}\ (\underline{e}_1)\ \mathsf{then}\ \underline{x} = \underline{e}_1 \mathsf{else}\ \underline{s}_2^*;\ \underline{x} = \underline{e}_2, \underline{x}) \end{array}$$

In order to preserve the semantics when the evaluation of $\underline{e}_1$ throws an exception, we force to evaluate $\underline{e}_1$ before evaluating $\underline{s}_2^*$ by introducing an assignment "$\underline{\diamond\mathsf{new}_1} = \underline{e}_1$" to avoid any side effects by $\underline{s}_2^*$. Note that we add the assignment only when $\underline{s}_2^*$ is not empty for a simple optimization.

Candidate for optimization

$$\begin{array}{l} ast2ir_e[\![e_1 \otimes e_2]\!](\Sigma)(\underline{x}) \ = \\ \text{LET } (\underline{s}_1^*, \underline{e}_1) = ast2ir_e[\![e_1]\!](\Sigma)(\underline{\diamond\mathsf{y}}) \\ \qquad (\underline{s}_2^*, \underline{e}_2) = ast2ir_e[\![e_2]\!](\Sigma)(\underline{\diamond\mathsf{z}}) \\ \text{IN } \quad \text{IF } \underline{s}_2^* \text{ is empty} \\ \qquad \text{THEN } (\underline{s}_1^*, \underline{e}_1 \otimes \underline{e}_2) \\ \qquad \text{ELSE } (\underline{s}_1^*;\ \underline{\diamond\mathsf{y}} = \underline{e}_1;\ \underline{s}_2^*, \underline{\diamond\mathsf{y}} \otimes \underline{e}_2) \end{array}$$

$$\begin{array}{l} ast2ir_e[\![lhs]\!](\Sigma)(\underline{x}) \ = \\ ast2ir_{lhs}[\![lhs]\!](\Sigma)(\underline{x}) \end{array}$$

$$\begin{array}{l} ast2ir_{lhs}[\![lit]\!](\Sigma)(\underline{x}) \ = \\ ast2ir_{lit}[\![lit]\!](\Sigma)(\underline{x}) \end{array}$$

$$\begin{array}{l} ast2ir_{lhs}[\![\texttt{arguments}]\!](\Sigma)(\underline{x}) \ = \\ (\langle\rangle, \Sigma(\diamond\mathsf{arguments})) \end{array}$$

$$\begin{array}{l} ast2ir_{lhs}[\![x]\!](\Sigma)(\underline{y}) \ = \\ (\langle\rangle, \underline{x}) \end{array}$$

Candidate for optimization

$$\begin{array}{l} ast2ir_{lhs}[\![\texttt{[}(e^?\texttt{,})^*\texttt{]}]\!](\Sigma)(\underline{x}) \ = \\ \text{LET } ((\underline{s}^*, \underline{e}) = ast2ir_e[\![e]\!](\Sigma)(\underline{\diamond\mathsf{elem}}))^* \\ \text{IN } \quad ((\underline{s}^*;\ \underline{\diamond\mathsf{elem}} = \underline{e})^*;\ \underline{x} = \texttt{[}(\underline{\diamond\mathsf{elem}}\texttt{,})^*\texttt{]}, \underline{x}) \end{array}$$

$$\begin{array}{l} ast2ir_{lhs}[\![\texttt{\{}(m\texttt{,})^*\texttt{\}}]\!](\Sigma)(\underline{x}) \ = \\ \text{LET } ((\underline{s}^*, \underline{mem}) = ast2ir_m[\![m]\!](\Sigma)(\underline{\diamond\mathsf{member}}))^* \\ \text{IN } \quad ((\underline{s}^*)^*;\ \underline{x} = \texttt{\{}(\underline{mem}\texttt{,})^*\texttt{\}}, \underline{x}) \end{array}$$

$$\begin{array}{l} ast2ir_{lhs}[\![\texttt{(}e\texttt{)}]\!](\Sigma)(\underline{x}) \ = \\ ast2ir_e[\![e]\!](\Sigma)(\underline{x}) \end{array}$$

$$\begin{array}{l} ast2ir_{lhs}[\![\texttt{function}\ f^?\texttt{(}(x\texttt{,})^*\texttt{)}\ \texttt{\{}fd^*\,vd^*\,s^*\texttt{\}}]\!](\Sigma)(\underline{y}) \ = \\ (\langle \underline{y} = \mathsf{function}\ \underline{f}^?\texttt{(}\underline{\diamond\mathsf{this}}\texttt{,}\ \underline{\diamond\mathsf{arguments}}\texttt{)\{} \\ \qquad (ast2ir_{fd}[\![fd]\!](\Sigma))^* \\ \qquad (\mathsf{var}\ \underline{x_i})^* \\ \qquad (ast2ir_{vd}[\![vd]\!](\Sigma))^* \\ \qquad (\underline{x_i} = \underline{\diamond\mathsf{arguments}}\texttt{["i"]})^* \\ \quad \textit{where } \underline{x_i} \textit{ is not the name of any of } fd \\ \qquad (ast2ir_s[\![s]\!](\Sigma;\ \underline{\diamond\mathsf{this}};\ \underline{\diamond\mathsf{arguments}}))^*\texttt{\}}\rangle, \underline{y}) \end{array}$$

$$\begin{array}{l} ast2ir_{lhs}[\![lhs\texttt{.}x]\!](\Sigma)(\underline{y}) \ = \\ ast2ir_{lhs}[\![lhs\texttt{["}x\texttt{"]}]\!](\Sigma)(\underline{y}) \end{array}$$

$$\begin{array}{l} ast2ir_{lhs}[\![lhs\texttt{["}x\texttt{"]}]\!](\Sigma)(\underline{y}) \ = \\ \text{LET } (\underline{s}_1^*, \underline{e}_1) = ast2ir_{lhs}[\![lhs]\!](\Sigma)(\underline{\diamond\mathsf{obj}_1}) \\ \text{IN } \quad (\underline{s}_1^*; \underline{\diamond\mathsf{obj}} = \diamond\mathsf{toObject}(\underline{e}_1), \underline{\diamond\mathsf{obj}}\texttt{["x"]}) \end{array}$$

$$\begin{array}{l} ast2ir_{lhs}[\![lhs\texttt{[}e\texttt{]}]\!](\Sigma)(\underline{x}) \ = \\ \text{LET } (\underline{s}_1^*, \underline{e}_1) = ast2ir_{lhs}[\![lhs]\!](\Sigma)(\underline{\diamond\mathsf{obj}_1}) \\ \qquad (\underline{s}_2^*, \underline{e}_2) = ast2ir_e[\![e]\!](\Sigma)(\underline{\diamond\mathsf{field}_1}) \\ \text{IN } \quad (\underline{s}_1^*; \underline{\diamond\mathsf{obj}} = \diamond\mathsf{toObject}(\underline{e}_1); \underline{s}_2^*, \underline{\diamond\mathsf{obj}}\texttt{[}\underline{e}_2\texttt{]}) \end{array}$$

Candidate for optimization

$$\begin{array}{l} ast2ir_{lhs}[\![\texttt{new}\ lhs\texttt{(}(e\texttt{,})^*\texttt{)}]\!](\Sigma)(\underline{x}) \ = \\ \text{LET } (\underline{s}_l^*, \underline{e}_l) = ast2ir_{lhs}[\![lhs]\!](\Sigma)(\underline{\diamond\mathsf{fun}_1}) \\ \qquad ((\underline{s}^*, \underline{e}) = ast2ir_e[\![e]\!](\Sigma)(\underline{\diamond\mathsf{y}}))^* \\ \text{IN } \quad (\underline{s}_l^*; \underline{\diamond\mathsf{fun}} = \diamond\mathsf{toObject}(\underline{e}_l);\ (\underline{s}^*;\ \underline{\diamond\mathsf{y}} = \underline{e})^*; \\ \qquad \underline{\diamond\mathsf{arguments}} = \texttt{[}(\underline{\diamond\mathsf{y}_i}\texttt{,})^*\texttt{]}; \\ \qquad \underline{\diamond\mathsf{proto}} = \underline{\diamond\mathsf{fun}}\texttt{["prototype"]}; \\ \qquad \underline{\diamond\mathsf{obj}} = \texttt{\{[[Prototype]] = }\underline{\diamond\mathsf{proto}}\texttt{\}}; \\ \qquad \underline{\diamond\mathsf{newObj}} = \mathsf{new}\ \underline{\diamond\mathsf{fun}}\texttt{(}\underline{\diamond\mathsf{obj}}\texttt{,}\ \underline{\diamond\mathsf{arguments}}\texttt{)}; \\ \qquad \underline{\diamond\mathsf{cond}} = \diamond\mathsf{isObject}(\underline{\diamond\mathsf{newObj}}); \\ \qquad \mathsf{if}(\underline{\diamond\mathsf{cond}})\ \mathsf{then}\ \underline{x} = \underline{\diamond\mathsf{newObj}}\ \ \mathsf{else}\ \underline{x} = \underline{\diamond\mathsf{obj}}, \underline{x}) \end{array}$$

$$\begin{array}{l} ast2ir_{lhs}[\![\texttt{new}\ lhs]\!](\Sigma)(\underline{x}) \ = \\ \text{LET } (\underline{s}^*, \underline{e}) = ast2ir_{lhs}[\![lhs]\!](\Sigma)(\underline{\diamond\mathsf{fun}_1}) \\ \text{IN } \quad (\underline{s}^*; \underline{\diamond\mathsf{fun}} = \diamond\mathsf{toObject}(\underline{e}); \\ \qquad \underline{\diamond\mathsf{arguments}} = \texttt{[]}; \\ \qquad \underline{\diamond\mathsf{proto}} = \underline{\diamond\mathsf{fun}}\texttt{["prototype"]}; \\ \qquad \underline{\diamond\mathsf{obj}} = \texttt{\{[[Prototype]] = }\underline{\diamond\mathsf{proto}}\texttt{\}}; \\ \qquad \underline{\diamond\mathsf{newObj}} = \mathsf{new}\ \underline{\diamond\mathsf{fun}}\texttt{(}\underline{\diamond\mathsf{obj}}\texttt{,}\ \underline{\diamond\mathsf{arguments}}\texttt{)}; \\ \qquad \underline{\diamond\mathsf{cond}} = \diamond\mathsf{isObject}(\underline{\diamond\mathsf{newObj}}); \\ \qquad \mathsf{if}(\underline{\diamond\mathsf{cond}})\ \mathsf{then}\ \underline{x} = \underline{\diamond\mathsf{newObj}}\ \ \mathsf{else}\ \underline{x} = \underline{\diamond\mathsf{obj}}, \underline{x}) \end{array}$$

$$\begin{array}{l} ast2ir_{lhs}[\![eval\texttt{(}e\texttt{)}]\!](\Sigma)(\underline{x}) \ = \\ \text{LET } (\underline{s}^*, \underline{e}) = ast2ir_e[\![e]\!](\Sigma)(\underline{\diamond\mathsf{new}_1}) \\ \text{IN } \quad (\underline{s}^*; \underline{x} = \mathsf{eval}(\underline{e}), \underline{x}) \end{array}$$

$$\begin{array}{l} ast2ir_{lhs}[\![\texttt{(}f\texttt{)}\texttt{(}(e\texttt{,})^*\texttt{)}]\!](\Sigma)(\underline{x}) \ = \\ ast2ir_{lhs}[\![f\texttt{(}(e\texttt{,})^*\texttt{)}]\!](\Sigma)(\underline{x}) \end{array}$$

Candidate for optimization

$$\begin{array}{l} ast2ir_{lhs}[\![f\texttt{(}(e\texttt{,})^*\texttt{)}]\!](\Sigma)(\underline{x}) \ = \\ \text{LET } ((\underline{s}^*, \underline{e}) = ast2ir_e[\![e]\!](\Sigma)(\underline{\diamond\mathsf{y}}))^* \\ \text{IN } \quad (\underline{\diamond\mathsf{obj}} = \diamond\mathsf{toObject}(\underline{f});\ (\underline{s}^*;\ \underline{\diamond\mathsf{y}} = \underline{e})^*; \\ \qquad \underline{\diamond\mathsf{arguments}} = \texttt{[}(\underline{\diamond\mathsf{y}_i}\texttt{,})^*\texttt{]}; \\ \qquad \underline{\diamond\mathsf{fun}} = \diamond\mathsf{getBase}(\underline{f}); \\ \qquad \underline{x} = \underline{\diamond\mathsf{obj}}(\underline{\diamond\mathsf{fun}}, \underline{\diamond\mathsf{arguments}}), \underline{x}) \end{array}$$

$$\begin{array}{l} ast2ir_{lhs}[\![\texttt{(}lhs\texttt{.}x\texttt{)}\texttt{(}(e\texttt{,})^*\texttt{)}]\!](\Sigma)(\underline{y}) \ = \\ ast2ir_{lhs}[\![lhs\texttt{["}x\texttt{"]}\texttt{(}(e\texttt{,})^*\texttt{)}]\!](\Sigma)(\underline{y}) \end{array}$$

$$\begin{array}{l} ast2ir_{lhs}[\![lhs\texttt{.}x\texttt{(}(e\texttt{,})^*\texttt{)}]\!](\Sigma)(\underline{y}) \ = \\ ast2ir_{lhs}[\![lhs\texttt{["}x\texttt{"]}\texttt{(}(e\texttt{,})^*\texttt{)}]\!](\Sigma)(\underline{y}) \end{array}$$

$$\begin{array}{l} ast2ir_{lhs}[\![\texttt{(}lhs\texttt{[}e'\texttt{])}\texttt{(}(e\texttt{,})^*\texttt{)}]\!](\Sigma)(\underline{x}) \ = \\ ast2ir_{lhs}[\![lhs\texttt{[}e'\texttt{]}\texttt{(}(e\texttt{,})^*\texttt{)}]\!](\Sigma)(\underline{x}) \end{array}$$

Candidate for optimization

$ast2ir_{lhs}[\![lhs[e'](\,(e,)^*)]\!](\Sigma)(\underline{x})\ =$
LET $(\underline{s}_l^*, \underline{e}_l) = ast2ir_{lhs}[\![lhs]\!](\Sigma)(\underline{\diamond\mathsf{obj}_1})$
$(\underline{s}'^*, \underline{e}') = ast2ir_e[\![e']\!](\Sigma)(\underline{\diamond\mathsf{field}_1})$
$((\underline{s}^*, \underline{e}) = ast2ir_e[\![e]\!](\Sigma)(\underline{\diamond\mathsf{y}}))^*$
IN $(\underline{s}_l^*;\underline{\diamond\mathsf{obj}} = \diamond\mathsf{toObject}(\underline{e}_l);\underline{s}'^*;$
$(\underline{s}^*;\ \underline{\diamond\mathsf{y}} = \underline{e})^*;$
$\underline{\diamond\mathsf{arguments}} = [(\underline{\diamond\mathsf{y}_i},)^*];$
$\underline{\diamond\mathsf{fun}} = \diamond\mathsf{toObject}(\underline{\diamond\mathsf{obj}}[\underline{e}']);$
$\underline{x} = \underline{\diamond\mathsf{fun}}(\underline{\diamond\mathsf{obj}},\ \underline{\diamond\mathsf{arguments}}), \underline{x})$

Candidate for optimization

$ast2ir_{lhs}[\![lhs((e,)^*)]\!](\Sigma)(\underline{x})\ =$
LET $(\underline{s}_l^*, \underline{e}_l) = ast2ir_{lhs}[\![lhs]\!](\Sigma)(\underline{\diamond\mathsf{obj}_1})$
$((\underline{s}^*, \underline{e}) = ast2ir_e[\![e]\!](\Sigma)(\underline{\diamond\mathsf{y}}))^*$
IN $(\underline{s}_l^*;\underline{\diamond\mathsf{obj}} = \diamond\mathsf{toObject}(\underline{e}_l);\ (\underline{s}^*;\ \underline{\diamond\mathsf{y}} = \underline{e})^*;$
$\underline{\diamond\mathsf{arguments}} = [(\underline{\diamond\mathsf{y}_i},)^*];$
$\underline{x} = \underline{\diamond\mathsf{obj}}(\diamond\mathsf{global},\ \underline{\diamond\mathsf{arguments}}), \underline{x})$

$ast2ir_{lit}[\![\mathtt{this}]\!](\Sigma)(\underline{x})\ =$
$(\langle\rangle, \Sigma(\underline{\diamond\mathsf{this}}))$

$ast2ir_{lit}[\![\mathtt{null}]\!](\Sigma)(\underline{x})\ =$
$(\langle\rangle, \mathsf{null})$

$ast2ir_{lit}[\![\mathtt{true}]\!](\Sigma)(\underline{x})\ =$
$(\langle\rangle, \mathsf{true})$

$ast2ir_{lit}[\![\mathtt{false}]\!](\Sigma)(\underline{x})\ =$
$(\langle\rangle, \mathsf{false})$

$ast2ir_{lit}[\![num]\!](\Sigma)(\underline{x})\ =$
$(\langle\rangle, \underline{num})$

$ast2ir_{lit}[\![str]\!](\Sigma)(\underline{x})\ =$
$(\langle\rangle, \underline{str})$

$ast2ir_{lit}[\![reg]\!](\Sigma)$
Regular expressions are desugared into construction of `RegExp` objects by `Disambiguator`

$ast2ir_m[\![pr\ :\ e]\!](\Sigma)(\underline{y})\ =$
LET $(\underline{s}^*, \underline{e}) = ast2ir_e[\![e]\!](\Sigma)(\underline{y})$
IN $(\underline{s}^*, ast2ir_{pr}[\![pr]\!]\ :\ \underline{e})$

$ast2ir_m[\![\mathsf{get}\ pr()\{fd^*\ vd^*\ s^*\}]\!](\Sigma)(\underline{x})\ =$
$(\langle\rangle, \mathsf{get}\ ast2ir_{pr}[\![pr]\!](\underline{\diamond\mathsf{this}},\ \underline{\diamond\mathsf{arguments}})\{$
$(ast2ir_{fd}[\![fd]\!](\Sigma))^*$
$(ast2ir_{vd}[\![vd]\!](\Sigma))^*$
$(ast2ir_s[\![s]\!](\Sigma;\underline{\diamond\mathsf{this}};\underline{\diamond\mathsf{arguments}}))^*\})$

$ast2ir_m[\![\mathsf{set}\ pr(x)\{fd^*\ vd^*\ s^*\}]\!](\Sigma)(\underline{y})\ =$
$(\langle\rangle, \mathsf{set}\ ast2ir_{pr}[\![pr]\!](\underline{\diamond\mathsf{this}},\ \underline{\diamond\mathsf{arguments}})\{$
$(ast2ir_{fd}[\![fd]\!](\Sigma))^*$
var $\underline{x}$
$(ast2ir_{vd}[\![vd]\!](\Sigma))^*$
$\underline{x} = \underline{\diamond\mathsf{arguments}}[\texttt{"0"}];$
where $\underline{x}$ is not the name of any of fd
$(ast2ir_s[\![s]\!](\Sigma;\underline{\diamond\mathsf{this}};\underline{\diamond\mathsf{arguments}}))^*\})$

A.4 CFG

This section describes each construct of the SAFE CFG in both the BNF notation and its corresponding implementation. The implementation of CFG nodes is available at:

`$SAFE_HOME/src/main/scala/kr/ac/kaist/safe/nodes/cfg/`

A.4.1 Syntax of CFG

$$
\begin{array}{rcl}
cfg \in \mathsf{CFG} & = & \wp(\mathsf{CFGFunction}) \\
f,\ \langle B, \hookrightarrow, C\rangle \in \mathsf{CFGFunction} & = & \\
 & & \wp(\mathsf{CFGBlock}) \times \wp(\mathsf{CFGEdge}) \times \wp(\mathsf{CFGCallTriple}) \\
b \in \mathsf{CFGBlock} & = & \mathsf{Entry} \mid \mathsf{Exit} \mid \mathsf{ExitExc} \mid \mathsf{Call} \mid \mathsf{AfterCall} \\
 & \mid & \mathsf{AfterCatch} \mid \mathsf{NormalBlock} \mid \mathsf{ModelBlock} \\
\mathsf{NormalBlock} & = & \mathsf{CFGInst}^+ \\
\mathsf{CFGEdge} & = & \mathsf{CFGBlock} \times \mathsf{CFGBlock} \times \mathsf{EdgeType} \\
\mathsf{EdgeType} & = & \mathtt{EdgeNormal} \mid \mathtt{EdgeExc} \\
C \in \mathsf{CFGCallTriple} & = & \mathsf{Call} \times \mathsf{AfterCall} \times \mathsf{AfterCatch} \\
i \in \mathsf{CFGInst} & = & \mathsf{CFGNormalInst} \mid \mathsf{CFGCallInst} \\
x \in \mathsf{CFGId} & = & \mathsf{String} \times \mathsf{VarKind} \\
s \in \mathsf{String} & & \\
\mathsf{VarKind} & ::= & \mathtt{GlobalVar} \mid \mathtt{PureLocalVar} \\
 & \mid & \mathtt{CapturedVar} \mid \mathtt{CapturedCatchVar} \\
a \in \mathsf{Address} & = & \mathsf{Integer} \\
v \in \mathsf{EJSVal} & = & \mathsf{Number} \mid \mathsf{String} \mid \mathsf{Boolean} \mid \mathsf{Null} \\
 & \mid & \mathsf{Undefined}
\end{array}
$$

$$
\begin{array}{lll}
\multicolumn{2}{l}{\mathsf{CFGNormalInst}} & \\
::= & x\ \mathtt{:=}\ \mathtt{alloc}(e^?)@a & \mathsf{CFGAlloc} \\
\mid & x\ \mathtt{:=}\ \mathtt{allocArray}(n)@a & \mathsf{CFGAllocArray} \\
\mid & x\ \mathtt{:=}\ \mathtt{allocArg}(n)@a & \mathsf{CFGAllocArg} \\
\mid & x\ \mathtt{:=}\ \mathtt{enterCode}(e) & \mathsf{CFGEnterCode} \\
\mid & x\ \mathtt{:=}\ e & \mathsf{CFGExprStmt} \\
\mid & x\ \mathtt{:=}\ \mathtt{delete}(e) & \mathsf{CFGDelete} \\
\mid & x\ \mathtt{:=}\ \mathtt{delete}(e, e) & \mathsf{CFGDeleteProp} \\
\mid & e\mathtt{[}e\mathtt{]}\ \mathtt{:=}\ e & \mathsf{CFGStore} \\
\mid & e\mathtt{[}s\mathtt{]}\ \mathtt{:=}\ e & \mathsf{CFGStoreStringIdx} \\
\mid & x\ \mathtt{:=}\ \mathtt{function}\ x^?(f)@a, a, a^? & \mathsf{CFGFunExpr} \\
\mid & \mathtt{assert}(e) & \mathsf{CFGAssert} \\
\mid & \mathtt{catch}(x) & \mathsf{CFGCatch} \\
\mid & \mathtt{return}(e^?) & \mathsf{CFGReturn} \\
\mid & \mathtt{throw}(e) & \mathsf{CFGThrow} \\
\mid & \mathtt{noop} & \mathsf{CFGNoOp} \\
\mid & x\ \mathtt{:=}\ x(e^*)@a^? & \mathsf{CFGInternalCall}
\end{array}
$$

$$
\begin{array}{lll}
\multicolumn{2}{l}{\mathsf{CFGCallInst}} & \\
::= & \mathtt{call}(e, e, e)@a, a & \mathsf{CFGCall} \\
\mid & \mathtt{construct}(e, e, e)@a, a & \mathsf{CFGConstruct}
\end{array}
$$

$$
\begin{array}{lll}
\multicolumn{2}{l}{e \in \mathsf{CFGExpr}} & \\
::= & x & \mathsf{CFGVarRef} \\
\mid & e \otimes e & \mathsf{CFGBin} \\
\mid & \ominus e & \mathsf{CFGUn} \\
\mid & e\mathtt{[}e\mathtt{]} & \mathsf{CFGLoad} \\
\mid & \mathtt{this} & \mathsf{CFGThis} \\
\mid & v & \mathsf{CFGVal}
\end{array}
$$

$$
\begin{array}{lll}
\ominus & ::= & \mathtt{void} \mid \mathtt{typeof} \mid \mathtt{+} \mid \mathtt{-} \mid \mathtt{\sim} \mid \mathtt{!} \\
\otimes & ::= & \mathtt{instanceof} \mid \mathtt{in} \mid \mathtt{|} \mid \mathtt{\&} \mid \mathtt{\hat{}} \mid \mathtt{\ll} \mid \mathtt{\gg} \mid \mathtt{\ggg} \mid \mathtt{+} \mid \mathtt{-} \mid \mathtt{*} \mid \mathtt{/} \mid \mathtt{\%} \\
 & \mid & \mathtt{==} \mid \mathtt{!=} \mid \mathtt{===!==} \mid \mathtt{<} \mid \mathtt{>} \mid \mathtt{<=} \mid \mathtt{>=}
\end{array}
$$

- Entry, Exit, and ExitExc blocks always uniquely exist for each CFGFunction, and an Entry block has no predecessor, Exit and ExitExc blocks have no successor.
  - $\forall\langle B, \_, \_\rangle \in \mathsf{CFGFunction}.$
    $(\nexists\ \mathtt{entry} \in \mathsf{Entry} \cap B),$
    $(\nexists\ \mathtt{exit} \in \mathsf{Exit} \cap B),$
    $(\nexists\ \mathtt{exitExc} \in \mathsf{ExitExc} \cap B)$
  - $\forall\langle B, \hookrightarrow, \_\rangle \in \mathsf{CFGFunction}.\ \nexists b \in B.$ s.t. $(b \hookrightarrow \mathtt{entry}) \lor (\mathtt{exit} \hookrightarrow b) \lor (\mathtt{exitExc} \hookrightarrow b)$ where $\mathtt{entry} \in \mathsf{Entry},\ \mathtt{exit} \in \mathsf{Exit},\ \mathtt{exitExc} \in \mathsf{ExitExc}$
- Each edge connects 2 blocks in the same CFGFunction.
  - $\forall\langle B, \hookrightarrow, \_\rangle \in \mathsf{CFGFunction}.\ \hookrightarrow \subseteq B \times B$
- Call, AfterCall, and AfterCatch blocks must be recorded in C without any edges between them, and the triple consisting of them is unique.
  - $\forall\langle B, \_, C\rangle \in \mathsf{CFGFunction}.$
    $\forall \mathtt{call} \in \mathsf{Call} \cap B.$
    $\forall \mathtt{acall} \in \mathsf{AfterCall} \cap B.$
    $\forall \mathtt{acatch} \in \mathsf{AfterCatch} \cap B.$
    $(\mathtt{call}, \_, \_),\ (\_, \mathtt{acall}, \_),\ (\_, \_, \mathtt{acatch}) \in C$
  - $\forall\langle \_, \_, C\rangle \in \mathsf{CFGFunction}.$
    $\forall(\mathtt{call}, \mathtt{acall}, \mathtt{acatch}), (\mathtt{call}', \mathtt{acall}', \mathtt{acatch}') \in C.$
    $\mathtt{call} = \mathtt{call}' \Leftrightarrow \mathtt{acall} = \mathtt{acall}'$
    $\Leftrightarrow \mathtt{acatch} = \mathtt{acatch}'$
- There is no instruction after `return` in a block. There is no instruction before `catch` in a block.
  - $\forall b \in \mathsf{NormalBlock}.$
    $(i_k \in b \land (i_k = \mathtt{return})) \to (\nexists i_{k'} \in b.\ k < k')$
  - $\forall b \in \mathsf{NormalBlock}.$
    $(i_k \in b \land (i_k = \mathtt{catch}) \to (\nexists i_{k'} \in b.\ k > k')$

A.4.2 CFG Data Types

$\mathtt{new}\ \mathsf{CFG}: \mathsf{CFGId}^* \to \mathsf{CFG}$

$$\mathsf{CFG}: \left\{\begin{array}{lll} \mathit{globalFunc} & : & \mathsf{CFGFunction} \\ \mathit{createFunction} & : & \mathsf{String} \times \mathsf{CFGId}^* \times \mathsf{CFGId}^* \\ & & \times\ \mathsf{String} \to \mathsf{CFGFunction} \\ \mathit{addEdge} & : & \wp(\mathsf{CFGBlock}) \times \wp(\mathsf{CFGBlock}) \\ & & \times\ \mathsf{EdgeType} \to \mathsf{Unit} \end{array}\right\}$$

$\mathtt{new}\ \mathsf{CFGFunction}: \mathsf{CFG} \times \mathsf{String} \times \mathsf{CFGId}^* \times \mathsf{CFGId}^* \times \mathsf{String} \to \mathsf{CFGFunction}$

$$\mathsf{CFGFunction}: \left\{\begin{array}{lll} \mathit{cfg} & : & \mathsf{CFG} \\ \mathit{argumentsName} & : & \mathsf{String} \\ \mathit{argVars} & : & \mathsf{CFGId}^* \\ \mathit{localVars} & : & \mathsf{CFGId}^* \\ \mathit{entry} & : & \mathsf{Entry} \\ \mathit{exit} & : & \mathsf{Exit} \\ \mathit{exitExc} & : & \mathsf{ExitExc} \\ \mathit{blocks} & : & \wp(\mathsf{CFGBlock}) \\ \mathit{captured} & : & \wp(\mathsf{CFGId}) \\ \mathit{createCall} & : & \mathsf{CFGCallInst} \times \mathsf{CFGId} \to \mathsf{Call} \\ \mathit{createBlock} & : & \mathsf{Unit} \to \mathsf{CFGBlock} \\ \mathit{createModelBlock} & : & \mathsf{Any} \to \mathsf{ModelBlock} \end{array}\right\}$$

$\mathtt{new}\ \mathsf{CFGBlock} : \mathsf{CFGFunction} \to \mathsf{CFGBlock}$

$$\mathsf{CFGBlock} : \left\{\begin{array}{lll} \mathit{func} & : & \mathsf{CFGFunction} \\ \mathit{succs} & : & \mathsf{EdgeType} \mapsto \wp(\mathsf{CFGBlock}) \\ \mathit{preds} & : & \mathsf{EdgeType} \mapsto \wp(\mathsf{CFGBlock}) \end{array}\right\}$$

$\mathtt{new}\ \mathsf{Call} : \mathsf{CFGFunction} \times \mathsf{CFGCallInst} \times \mathsf{CFGId} \to \mathsf{Call}$

$$\mathsf{Call} : \left\{\begin{array}{lll} \mathit{afterCall} & : & \mathsf{AfterCall} \\ \mathit{afterCatch} & : & \mathsf{AfterCatch} \\ \mathit{callInst} & : & \mathsf{CFGCallInst} \end{array}\right\}$$

$\mathtt{new}\ \mathsf{AfterCall} : \mathsf{CFGFunction} \times \mathsf{Call} \times \mathsf{CFGId} \to \mathsf{AfterCall}$

$$\mathsf{AfterCall} : \left\{\begin{array}{lll} \mathit{call} & : & \mathsf{Call} \\ \mathit{retVar} & : & \mathsf{CFGId} \end{array}\right\}$$

$\mathtt{new}\ \mathsf{AfterCatch} : \mathsf{CFGFunction} \times \mathsf{Call} \to \mathsf{AfterCatch}$

$$\mathsf{AfterCatch} : \left\{\mathit{call} : \mathsf{Call}\right\}$$

$\mathtt{new}\ \mathsf{NormalBlock}: \mathsf{CFGFunction} \to \mathsf{NormalBlock}$

$$\mathsf{NormalBlock}: \left\{\begin{array}{lll} \mathit{insts} & : & \mathsf{CFGInst}^* \\ \mathit{createInst} & : & \mathsf{CFGInst} \to \mathsf{Unit} \end{array}\right\}$$

$\mathtt{new}\ \mathsf{ModelBlock} : \mathsf{CFGFunction} \times \mathsf{Any} \to \mathsf{ModelBlock}$

$$\mathsf{ModelBlock} : \left\{\mathit{data} : \mathsf{Any}\right\}$$

$\mathtt{new}\ \mathsf{CFGId} : \mathsf{String} \times \mathsf{VarKind} \to \mathsf{CFGId}$

$$\mathsf{CFGId} : \left\{\begin{array}{lll} \mathit{name} & : & \mathsf{String} \\ \mathit{kind} & : & \mathsf{VarKind} \end{array}\right\}$$

A.4.3 Methods

Helper Functions

$\mathit{toSeq}(S)$: $\wp(\mathsf{Any}) \to \mathsf{Any}^*$
= convert a set of any elements into a sequence of them by an arbitrary order.

$\mathit{fold}(A)(b)(f)$: $\mathsf{Any}^* \times \mathsf{Any}' \times (\mathsf{Any} \times \mathsf{Any}' \to \mathsf{Any}') \to \mathsf{Any}'$
= if $(\mathit{length}(A) = 0)$ then b
else $\mathit{fold}(\mathit{tailOf}(A))(f(\mathit{headOf}(A), b))(f)$

$\mathit{iter}(A)(f)$: $\mathsf{Any}^* \times (\mathsf{Any} \to \mathsf{Unit}) \to \mathsf{Unit}$
= if $(\mathit{length}(A) = 0)$ then Unit
else $f(\mathit{headOf}(A))$
$\mathit{iter}(\mathit{tailOf}(A))(f)$

$\mathit{iter}(A)(\mathit{idx})(f)$: $\mathsf{Any}^* \times (\mathsf{Any} \to \mathsf{Unit}) \to \mathsf{Unit}$
= if $(\mathit{length}(A) = 0)$ then Unit
else $f(\mathit{headOf}(A), \mathit{idx})$
$\mathit{iter}(\mathit{tailOf}(A))(\mathit{idx} + 1)(f)$

$\mathit{getTail}(\mathit{cfg}, B, f)$: $\mathsf{CFG} \times \mathsf{CFGFunction} \times \mathsf{CFGBlock}^* \to \mathsf{CFGBlock}$
= if $(\mathit{length}(B) = 0)$ then
$f.\mathit{createBlock}$
else if $(\mathit{length}(B) = 1)$ then
$\mathit{headOf}(B)$
else $b \overset{let}{=} f.\mathit{createBlock}$
$\mathit{cfg}.\mathit{addEdge}(B, b)$
b

CFG Methods

$\texttt{new}\ \mathsf{CFG}(globalVars)$
$= \text{let } cfg \text{ be a new } \mathsf{CFG}$
$\quad cfg.globalFunc$
$\quad\quad \leftarrow \texttt{new}\ \mathsf{CFGFunction}(\text{``''}, \mathsf{Nil}, globalVars, \text{``top-level''})$
$\quad cfg$

$\mathsf{CFG}.createFunction(argsName, argVars, localVars, name)$
$= \texttt{new}\ \mathsf{CFGFunction}(\mathsf{this}, argsName, argVars, localVars, name)$

$\mathsf{CFG}.addEdge(B_1, B_2, ty = \texttt{EdgeNormal})$
$= iter(toSeq(B_1))(\lambda(b_1) \Rightarrow iter(toSeq(B_2))(\lambda(b_2) \Rightarrow$
$\quad\quad b_1.succs(ty) \leftarrow b_1.succs(ty) \cup \{b_2\}$
$\quad\quad b_2.preds(ty) \leftarrow b_2.preds(ty) \cup \{b_1\}$
$\quad))$

CFGFunction Methods

$\texttt{new}\ \mathsf{CFGFunction}(cfg, argsName, argVars, localVars, name)$
$= \text{let } func \text{ be a new } \mathsf{CFGFunction}$
$\quad entry \leftarrow \texttt{new}\ \mathsf{Entry}$
$\quad exitExc \leftarrow \texttt{new}\ \mathsf{ExitExc}$
$\quad exit \leftarrow \texttt{new}\ \mathsf{Exit}$
$\quad func.cfg \leftarrow cfg$
$\quad func.argumentsName \leftarrow argsName$
$\quad func.argVars \leftarrow argVars$
$\quad func.localVars \leftarrow localVars$
$\quad func.entry \leftarrow entry$
$\quad func.exit \leftarrow exit$
$\quad func.exitExc \leftarrow exitExc$
$\quad func.blocks \leftarrow \{entry, exit, exitExc\}$
$\quad func.captured \leftarrow \emptyset$
$\quad func$

$\mathsf{CFGFunction}.createCall(callInst, retVar)$
$= \texttt{new}\ \mathsf{Call}(\mathsf{this}, callInst, retVar)$

$\mathsf{CFGFunction}.createBlock$
$= \texttt{new}\ \mathsf{NormalBlock}(\mathsf{this})$

$\mathsf{CFGFunction}.createModelBlock(data)$
$= \texttt{new}\ \mathsf{ModelBlock}(\mathsf{this}, data)$

CFGBlock Methods

$\texttt{new}\ \mathsf{CFGBlock}(func)$
$= \text{let } b \text{ be a new } \mathsf{CFGBlock}$
$\quad b.func \leftarrow func$
$\quad b.succs \leftarrow \emptyset$
$\quad b.preds \leftarrow \emptyset$
$\quad b$

$\texttt{new}\ \mathsf{Call}(func, callInst, retVar)$
$= \text{let } call \text{ be a new } \mathsf{Call} \text{ initializing with } \texttt{new}\ \mathsf{CFGBlock}(func)$
$\quad call.afterCall \leftarrow \texttt{new}\ \mathsf{AfterCall}(call, retVar)$
$\quad call.afterCatch \leftarrow \texttt{new}\ \mathsf{AfterCatch}(call)$
$\quad call.callInst \leftarrow callInst$
$\quad call$

$\texttt{new}\ \mathsf{AfterCall}(func, call, retVar)$
$= \text{let } acall \text{ be a new } \mathsf{AfterCall} \text{ initializing with}$
$\quad\quad \texttt{new}\ \mathsf{CFGBlock}(func)$
$\quad acall.call \leftarrow call$
$\quad acall.retVar \leftarrow retVar$
$\quad acall$

$\texttt{new}\ \mathsf{AfterCatch}(func, call)$
$= \text{let } acatch \text{ be a new } \mathsf{AfterCatch} \text{ initializing with}$
$\quad\quad \texttt{new}\ \mathsf{CFGBlock}(func)$
$\quad acatch.call \leftarrow call$
$\quad acatch$

$\texttt{new}\ \mathsf{NormalBlock}(func)$
$= \text{let } b \text{ be a new } \mathsf{NormalBlock} \text{ initializing with}$
$\quad\quad \texttt{new}\ \mathsf{CFGBlock}(func)$
$\quad b.insts \leftarrow \mathsf{Nil}$
$\quad b$

$\mathsf{NormalBlock}.createInst(inst)$
$= \mathsf{this}.insts \leftarrow \mathsf{this}.insts :: inst$

$\texttt{new}\ \mathsf{ModelBlock}(func, data)$
$= \text{let } modelB \text{ be a new } \mathsf{ModelBlock} \text{ initializing with}$
$\quad\quad \texttt{new}\ \mathsf{CFGBlock}(func)$
$\quad modelB.data \leftarrow data$
$\quad modelB$

$\texttt{new}\ \mathsf{CFGId}(name, kind)$
$= \text{let } id \text{ be a new } \mathsf{CFGId}$
$\quad id.name \leftarrow name$
$\quad id.kind \leftarrow kind$
$\quad id$

A.5 IR to CFG

This section describes the SAFE translation rules from IR to CFG, whose implementation is available at:

`$SAFE_HOME/src/main/scala/kr/ac/kaist/safe/cfg_builder/CFGBuilder.scala`

LabelMap	$: \mathsf{JSLabel} \mapsto \wp(\mathsf{CFGBlock})$
JSLabel	$= \texttt{RetLabel} \mid \texttt{ThrowLabel} \mid \texttt{ThrowEndLabel} \mid \texttt{AfterCatchLabel} \mid \mathsf{UserLabel}$
UserLabel	$= \mathsf{String}$
$[\![-]\!]_{root}$	$: \mathsf{IRRoot} \to \mathsf{CFG}$
$[\![-]\!]_{fdvars}$	$: \mathsf{IRFunDecl}^* \to \mathsf{CFGId}^*$
$[\![-]\!]_{vds}$	$: \mathsf{IRVarStmt}^* \to \mathsf{CFGId}^*$
$[\![-]\!]_{args}$	$: \mathsf{IRStmt}^* \to \mathsf{CFGId}^*$
$[\![-]\!]_{functional}$	$: \mathsf{IRFunctional} \to \mathsf{CFGFunction}$
$[\![-]\!]_{fd}$	$: \mathsf{IRFunDecl} \times \mathsf{CFGFunction} \times \mathsf{NormalBlock} \to \mathsf{Unit}$
$[\![-]\!]_{fd^*}$	$: \mathsf{IRFunDecl}^* \times \mathsf{CFGFunction} \times \mathsf{NormalBlock} \to \mathsf{Unit}$
$[\![-]\!]_{stmt}$	$: \mathsf{IRStmt} \times \mathsf{CFGFunction} \times \mathsf{CFGBlock}^* \times \mathsf{LabelMap} \to \mathsf{CFGBlock}^* \times \mathsf{LabelMap}$
$[\![-]\!]_{stmt^*}$	$: \mathsf{IRStmt}^* \times \mathsf{CFGFunction} \times \mathsf{CFGBlock}^* \times \mathsf{LabelMap} \to \mathsf{CFGBlock}^* \times \mathsf{LabelMap}$
$[\![-]\!]_{mem}$	$: \mathsf{IRField} \times \mathsf{NormalBlock} \times \mathsf{IRId} \to \mathsf{Unit}$
$[\![-]\!]_{elem}$	$: \mathsf{IRExpr} \times \mathsf{NormalBlock} \times \mathsf{IRId} \times \mathsf{Integer} \to \mathsf{Unit}$
$[\![-]\!]_{expr}$	$: \mathsf{IRExpr} \to \mathsf{CFGExpr}$
$[\![-]\!]_{id}$	$: \mathsf{IRId} \to \mathsf{CFGId}$

We use the following global variables for describing the translation from IR to CFG:

- $*$ *catchVars* : $\wp(\mathsf{String})$ initializing with $\emptyset$
- $*$ *cfg* : CFG
- $*$ *cfgIdMap* : $\mathsf{String} \mapsto \mathsf{CFGId}$ initializing with [ ]
- $*$ *currentFunc* : $\mathsf{CFGFunction}$

To distinguish global variables, we use the prefix "(global)" and "$\leftarrow$" for assignments.

$$\begin{array}{l}
[\![\mathsf{IRRoot}(\mathit{fds}, \mathit{vds}, \mathit{stmts})]\!]_{root} \\
= \mathit{globalVars} \overset{let}{=} [\![\mathit{fds}]\!]_{fdvars} \ +\!+\ [\![\mathit{vds}]\!]_{vds} \\
\quad \text{(global)} :\ \mathit{cfg} \leftarrow \texttt{new}\ \mathsf{CFG}(\mathit{globalVars}) \\
\quad \mathit{globalFunc} \overset{let}{=} \mathit{cfg.globalFunc} \\
\quad \text{(global)} :\ \mathit{currentFunc} \leftarrow \mathit{globalFunc} \\
\quad \mathit{startBlock} \overset{let}{=} \mathit{globalFunc.breateBlock} \\
\quad \mathit{cfg.addEdge}(\mathit{globalFunc.entry}, \mathit{startBlock}) \\
\quad [\![\mathit{fds}]\!]_{fd^*}(\mathit{globalFunc}, \mathit{startBlock}) \\
\quad (B, L) \overset{let}{=} [\![\mathit{stmts}]\!]_{stmt^*}(\mathit{globalFunc}, \mathit{startBlock}, [\ ]) \\
\quad \mathit{cfg.addEdge}(B, \mathit{globalFunc.exit}) \\
\quad \mathit{cfg.addEdge}(L(\texttt{ThrowLabel}), \mathit{globalFunc.exitExc}, \texttt{EdgeExc}) \\
\quad \mathit{cfg.addEdge}(L(\texttt{ThrowEndLabel}), \mathit{globalFunc.exitExc}) \\
\quad \mathit{cfg.addEdge}(L(\texttt{AfterCatchLabel}), \mathit{globalFunc.exitExc}) \\
\quad \mathit{cfg}
\end{array}$$

$$\begin{array}{l}
[\![\mathit{fds}]\!]_{fdvars} \\
= \mathit{fold}(\mathit{fds})(\mathsf{Nil})(\lambda(\mathit{vars}, \begin{smallmatrix}\mathsf{function}\ f(_,_) \\ \{_,_,_,_\}\end{smallmatrix} \Rightarrow \mathit{vars} :: [\![f]\!]_{id})
\end{array}$$

$$\begin{array}{l}
[\![\mathit{vds}]\!]_{vds} \\
= \mathit{fold}(\mathit{vds})(\mathsf{Nil})(\lambda(\mathit{vars}, \mathsf{var}\ x) \Rightarrow \mathit{vars} :: [\![x]\!]_{id})
\end{array}$$

$$\begin{array}{l}
[\![\mathit{args}]\!]_{args} \\
= \mathit{fold}(\mathit{args})(\mathsf{Nil})(\lambda(\mathit{args}, x = \mathit{arguments}[i]) \Rightarrow \mathit{args} :: [\![x]\!]_{id})
\end{array}$$

$$\begin{array}{l}
[\![\begin{smallmatrix}\mathsf{IRFunctional}(_, name, params, \\ args, fds, vds, body)\end{smallmatrix}]\!]_{functional} \\
= \mathit{argVars} \overset{let}{=} [\![\mathit{args}]\!]_{args} \\
\quad \mathit{localVars} \overset{let}{=} [\![\mathit{fds}]\!]_{fdvars} +\!+ [\![\mathit{vds}]\!]_{vds} \\
\quad \mathit{argumentsName} \overset{let}{=} \mathit{headOf}(\mathit{params}).\mathit{name} \\
\quad \mathit{nameStr} \overset{let}{=} \mathit{name.name} \\
\quad \mathit{newFunc} \overset{let}{=} \mathit{cfg.createFunction}(\mathit{argumentsName}, \mathit{argVars}, \\
\quad \mathit{localVars}, \mathit{nameStr}) \\
\quad \mathit{oldFunc} \overset{let}{=} \mathit{currentFunc} \\
\quad \text{(global)} :\ \mathit{currentFunc} \leftarrow \mathit{newFunc} \\
\quad \mathit{startBlock} \overset{let}{=} \mathit{newFunc.createBlock} \\
\quad \mathit{cfg.addEdge}(\mathit{newFunc.entry}, \mathit{startBlock}) \\
\quad [\![\mathit{fds}]\!]_{fd^*}(\mathit{newFunc}, \mathit{startBlock}) \\
\quad (B, L) \overset{let}{=} [\![\mathit{body}]\!]_{stmt^*}(\mathit{newFunc}, \mathit{startBlock}, [\ ]) \\
\quad \mathit{cfg.addEdge}(B, \mathit{newFunc.exit}) \\
\quad \mathit{cfg.addEdge}(L(\texttt{RetLabel}), \mathit{newFunc.exit}) \\
\quad \mathit{cfg.addEdge}(L(\texttt{ThrowLabel}), \mathit{newFunc.exitExc}, \texttt{EdgeExc}) \\
\quad \mathit{cfg.addEdge}(L(\texttt{ThrowEndLabel}), \mathit{newFunc.exitExc}) \\
\quad \mathit{cfg.addEdge}(L(\texttt{AfterCatchLabel}), \mathit{newFunc.exitExc}) \\
\quad \text{(global)} :\ \mathit{currentFunc} \leftarrow \mathit{oldFunc} \\
\quad \mathit{newFunc}
\end{array}$$

$$\begin{array}{l}
[\![\mathsf{IRFunDecl}(\mathit{functional})]\!]_{fd}(\mathit{func}, \mathit{block}) \\
= \mathit{lhs} \overset{let}{=} [\![\mathit{functional.name}]\!]_{id} \\
\quad \mathit{newFunc} \overset{let}{=} [\![\mathit{functional}]\!]_{functional} \\
\quad (\mathit{addr1}, \mathit{addr2}) \overset{let}{=} (\texttt{new}\ \mathsf{Address},\ \texttt{new}\ \mathsf{Address}) \\
\quad \mathit{block.createInst}(\mathit{lhs} := \texttt{function}\ (\mathit{newFunc}\ @\ \mathit{addr1},\ \mathit{addr2}))
\end{array}$$

$$\begin{array}{l}
[\![\mathit{fds}]\!]_{fd^*}(\mathit{func}, \mathit{block}) \\
= \mathit{iter}(\mathit{fds})(\lambda(\mathit{fd}) \Rightarrow [\![fd]\!]_{fd}(\mathit{func}, \mathit{block}))
\end{array}$$

$$\begin{array}{l}
[\![\mathsf{IRStmtUnit}(\mathit{stmts})]\!]_{stmt}(f, B, L) \\
= [\![\mathit{stmts}]\!]_{stmt^*}(f, B, L)
\end{array}$$

$$\begin{array}{l}
[\![\mathsf{IRSeq}(\mathit{stmts})]\!]_{stmt}(f, B, L) \\
= [\![\mathit{stmts}]\!]_{stmt^*}(f, B, L)
\end{array}$$

$$\begin{array}{l}
[\![\mathsf{IRFunExpr}(\mathit{lhs}, \mathit{functional})]\!]_{stmt}(f, B, L) \\
= \mathit{newFunc} \overset{let}{=} [\![\mathit{functional}]\!]_{functional} \\
\quad (\mathit{addr1}, \mathit{addr2}, \mathit{addr3}) \overset{let}{=} (\texttt{new}\ \mathsf{Address},\ \texttt{new}\ \mathsf{Address}, \\
\qquad\qquad\qquad\qquad \texttt{new}\ \mathsf{Address}) \\
\quad \mathit{name} \overset{let}{=} [\![\mathit{functional.name}]\!]_{id} \\
\quad \mathit{tailBlock} \overset{let}{=} \mathit{getTail}(B, f) \\
\quad \mathit{if}\ \ (\mathit{name.kind} = \texttt{CapturedVar}) \\
\qquad \mathit{tailBlock.createInst}(\mathit{lhs} := \texttt{function}\ \mathit{name}\ (\\
\qquad\qquad\qquad \mathit{newFunc}\ @\ \mathit{addr1},\ \mathit{addr2}, \\
\qquad\qquad\qquad \mathit{addr3})) \\
\quad \mathit{else} \\
\qquad \mathit{tailBlock.createInst}(\mathit{lhs} := \texttt{function}\ (\mathit{newFunc}\ @\ \mathit{addr1}, \\
\qquad\qquad\qquad \mathit{addr2})) \\
\quad (\mathit{tailBlock}, L)
\end{array}$$

$$\begin{array}{l}
[\![x = \{\mathit{members}\}]\!]_{stmt}(f, B, L) \\
= \mathit{tailBlock} \overset{let}{=} \mathit{getTail}(B, f) \\
\quad \mathit{addr1} \overset{let}{=} \texttt{new}\ \mathsf{Address} \\
\quad \mathit{tailBlock.createInst}(x := \texttt{alloc}()\ @\ \mathit{addr1}) \\
\quad \mathit{iter}(\mathit{members})(\lambda(m) \Rightarrow [\![m]\!]_{mem}(\mathit{tailBlock}, x)) \\
\quad (\mathit{tailBlock}, L[\texttt{ThrowLabel} \mapsto \mathit{tailBlock} \cup L(\texttt{ThrowLabel})])
\end{array}$$

$$\begin{array}{l}
[\![x = \{\mathit{members}, \mathit{proto}\}]\!]_{stmt}(f, B, L) \\
= \mathit{tailBlock} \overset{let}{=} \mathit{getTail}(B, f) \\
\quad \mathit{protoId} \overset{let}{=} [\![\mathit{proto}]\!]_{id} \\
\quad \mathit{addr1} \overset{let}{=} \texttt{new}\ \mathsf{Address} \\
\quad \mathit{tailBlock.createInst}(x := \texttt{alloc}(\mathit{protoId})\ @\ \mathit{addr1}) \\
\quad \mathit{iter}(\mathit{members})(\lambda(m) \Rightarrow [\![m]\!]_{mem}(\mathit{tailBlock}, x)) \\
\quad (\mathit{tailBlock}, L[\texttt{ThrowLabel} \mapsto \mathit{tailBlock} \cup L(\texttt{ThrowLabel})])
\end{array}$$

$$\begin{array}{l}
[\![\mathsf{try}\{\mathit{body}\}\ \mathsf{catch}(x)\ \{\mathit{catIR}\}]\!]_{stmt}(f, B, L) \\
= \text{(global)} :\ \mathit{catchVars} \leftarrow \mathit{catchVars} \cup \{\mathit{x.name}\} \\
\quad \mathit{tryB} \overset{let}{=} f.\mathit{createBlock} \\
\quad \mathit{cfg.addEdge}(B, \mathit{tryB}) \\
\quad \mathit{catB} \overset{let}{=} f.\mathit{createBlock} \\
\quad \mathit{catB.createInst}(\texttt{catch}(transfunx_{id})) \\
\quad (\mathit{trybs}, \mathit{trylmap}) \overset{let}{=} [\![\mathit{body}]\!]_{stmt}(f, \mathit{tryB}, [\ ]) \\
\quad \mathit{cfg.addEdge}(\mathit{trylmap}(\texttt{ThrowLabel}), \mathit{catB}, \texttt{EdgeExc}) \\
\quad \mathit{cfg.addEdge}(\mathit{trylmap}(\texttt{ThrowEndLabel}), \mathit{catB}) \\
\quad \mathit{cfg.addEdge}(\mathit{trylmap}(\texttt{AfterCatchLabel}), \mathit{catB}) \\
\quad \mathit{tmplmap} \overset{let}{=} \mathit{trylmap} - \texttt{ThrowLabel} - \texttt{ThrowEndLabel} \\
\qquad\qquad - \texttt{AfterCatchLabel} \\
\quad (\mathit{catchbs}, \mathit{catchlamp}) \overset{let}{=} [\![\mathit{catIR}]\!]_{stmt}(f, \mathit{catB}, \mathit{tmplmap}) \\
\quad (\mathit{trybs}\ +\!+\ \mathit{catchbs}, L\ +\!+\ \mathit{catchlmap})
\end{array}$$

$$
\begin{array}{l}
[\![\textsf{try}\{\mathit{body}\}\ \textsf{finally}\ \{\mathit{finIR}\}]\!]_{stmt}(f, B, L) \\
= \mathit{tryB} \overset{let}{=} f.\mathit{createBlock} \\
\quad \mathit{cfg.addEdge}(B, \mathit{tryB}) \\
\quad \mathit{finB} \overset{let}{=} f.\mathit{createBlock} \\
\quad (\mathit{trybs}, \mathit{trylmap}) \overset{let}{=} [\![\mathit{body}]\!]_{stmt}(f, \mathit{tryB}, [\,]) \\
\quad (\mathit{finbs}, \mathit{finlamp}) \overset{let}{=} [\![\mathit{finIR}]\!]_{stmt}(f, \mathit{finB}, L) \\
\quad \mathit{cfg.addEdge}(\mathit{trybs}, \mathit{finB}) \\
\quad \mathit{reslmap} \overset{let}{=} \mathit{fold}(\mathit{trylmap} - \texttt{AfterCatchLabel}) \\
\qquad\qquad\qquad (\mathit{finlmap})(\lambda((l, B'), L') \Rightarrow \\
\qquad \mathit{dupB} \overset{let}{=} f.\mathit{createBlock} \\
\qquad (B'', L'') \overset{let}{=} [\![\mathit{finIR}]\!]_{stmt}(f, \mathit{dupB}, L') \\
\qquad \mathit{if}\ (l = \texttt{ThrowLabel}) \\
\qquad\quad \mathit{cfg.addEdge}(\mathit{trylmap}(\texttt{AfterCatchLabel}),\ \mathit{dupB}) \\
\qquad\quad \mathit{cfg.addEdge}(B', \mathit{dupB}, \texttt{EdgeExc}) \\
\qquad\quad L''[\texttt{ThrowEndLabel} \mapsto (L''(\texttt{ThrowEndLabel})\ ++\ B'')] \\
\qquad \mathit{else} \\
\qquad\quad \mathit{cfg.addEdge}(B', \mathit{dupB}, \texttt{EdgeExc}) \\
\qquad\quad L''[l \mapsto (L''(l)\ ++\ B'')] \\
\quad) \\
\quad (\mathit{finbs}, \mathit{reslmap})
\end{array}
$$

$$
\begin{array}{l}
[\![\begin{smallmatrix}\textsf{try}\{\mathit{body}\}\ \textsf{catch}(x)\ \{\mathit{catIR}\} \\ \textsf{finally}\ \{\mathit{finIR}\}\end{smallmatrix}]\!]_{stmt}(f, B, L) \\
= (\text{global}) :\ \mathit{catchVars} \leftarrow \mathit{catchVars} \cup \{x.\mathit{name}\} \\
\quad \mathit{tryB} \overset{let}{=} f.\mathit{createBlock} \\
\quad \mathit{cfg.addEdge}(B, \mathit{tryB}) \\
\quad \mathit{catB} \overset{let}{=} f.\mathit{createBlock} \\
\quad \mathit{catB.createInst}(\texttt{catch}(\mathit{transfunx}_{id})) \\
\quad \mathit{finB} \overset{let}{=} f.\mathit{createBlock} \\
\quad (\mathit{trybs}, \mathit{trylmap}) \overset{let}{=} [\![\mathit{body}]\!]_{stmt}(f, \mathit{tryB}, [\,]) \\
\quad \mathit{cfg.addEdge}(\mathit{trylmap}(\texttt{ThrowLabel}), \mathit{catB}, \texttt{EdgeExc}) \\
\quad \mathit{cfg.addEdge}(\mathit{trylmap}(\texttt{ThrowEndLabel}), \mathit{catB}) \\
\quad \mathit{cfg.addEdge}(\mathit{trylmap}(\texttt{AfterCatchLabel}), \mathit{catB}) \\
\quad (\mathit{catchbs}, \mathit{catchlamp}) \overset{let}{=} [\![\mathit{catIR}]\!]_{stmt}(f, \mathit{catB}, \mathit{tmplmap}) \\
\quad (\mathit{finbs}, \mathit{finlamp}) \overset{let}{=} [\![\mathit{finIR}]\!]_{stmt}(f, \mathit{finB}, L) \\
\quad \mathit{cfg.addEdge}(\mathit{trybs}\ ++\ \mathit{catchbs}, \mathit{finB}) \\
\quad \mathit{reslmap} \overset{let}{=} \mathit{fold}(\mathit{catchlmap} - \texttt{AfterCatchLabel}) \\
\qquad\qquad\qquad (\mathit{finlmap})(\lambda((l, B'), L') \Rightarrow \\
\qquad \mathit{dupB} \overset{let}{=} f.\mathit{createBlock} \\
\qquad (B'', L'') \overset{let}{=} [\![\mathit{finIR}]\!]_{stmt}(f, \mathit{dupB}, L') \\
\qquad \mathit{if}\ (l = \texttt{ThrowLabel}) \\
\qquad\quad \mathit{cfg.addEdge}(\mathit{catchlmap}(\texttt{AfterCatchLabel}), \mathit{dupB}) \\
\qquad\quad \mathit{cfg.addEdge}(B', \mathit{dupB}, \texttt{EdgeExc}) \\
\qquad\quad L''[\texttt{ThrowEndLabel} \mapsto (L''(\texttt{ThrowEndLabel})\ ++\ B'')] \\
\qquad \mathit{else} \\
\qquad\quad \mathit{cfg.addEdge}(B', \mathit{dupB}, \texttt{EdgeExc}) \\
\qquad\quad L''[l \mapsto (L''(l)\ ++\ B'')] \\
\quad) \\
\quad (\mathit{finbs}, \mathit{reslmap})
\end{array}
$$

$$
\begin{array}{l}
[\![x = [\mathit{elems}]]\!]_{stmt}(f, B, L) \\
= \mathit{tailBlock} \overset{let}{=} \mathit{getTail}(B, f) \\
\quad \mathit{addr} \overset{let}{=} \texttt{new}\ \textsf{Address}\quad (\texttt{arguments}) \\
\quad \mathit{tailBlock.createInst}(x := \texttt{allocArg}(\mathit{length}(\mathit{elems}))\ @\ \mathit{addr}) \\
\quad \mathit{iter}(\mathit{elems})(0)(\lambda(e, k) \Rightarrow [\![e]\!]_{elem}(\mathit{tailBlock}, x, k)) \\
\quad (\mathit{tailBlock}, L[\texttt{ThrowLabel} \mapsto L(\texttt{ThrowLabel}) \cup \{\mathit{tailBlock}\})
\end{array}
$$

$$
\begin{array}{l}
[\![x = [\mathit{elems}]]\!]_{stmt}(f, B, L) \\
= \mathit{tailBlock} \overset{let}{=} \mathit{getTail}(B, f) \\
\quad \mathit{addr} \overset{let}{=} \texttt{new}\ \textsf{Address} \\
\quad \mathit{tailBlock.createInst}(x := \texttt{allocArray}(\mathit{length}(\mathit{elems}))\ @\ \mathit{addr}) \\
\quad \mathit{iter}(\mathit{elems})(0)(\lambda(e, k) \Rightarrow [\![e]\!]_{elem}(\mathit{tailBlock}, x, k)) \\
\quad (\mathit{tailBlock}, L[\texttt{ThrowLabel} \mapsto L(\texttt{ThrowLabel}) \cup \{\mathit{tailBlock}\})
\end{array}
$$

$$
\begin{array}{l}
[\![\textsf{break}\ \mathit{label}]\!]_{stmt}(\textsf{f}, \textsf{B}, \textsf{L}) \\
= \mathit{key} \overset{let}{=} \mathit{label.name} \\
\quad \mathit{bs} \overset{let}{=} L(\mathit{key}) \cup B \\
\quad (\emptyset, L[\mathit{key} \mapsto \mathit{bs}])
\end{array}
$$

$$
\begin{array}{l}
[\![x = \mathit{fun}(\mathit{thisB}, \mathit{args})]\!]_{stmt}(f, B, L) \\
= \mathit{tailBlock} \overset{let}{=} \mathit{getTail}(B, f) \\
\quad \mathit{thisId} \overset{let}{=} \texttt{new}\ \textsf{CFGId}(\texttt{<>this<>}, \texttt{PureLocalVar}) \\
\quad \mathit{tailBlock.createInst}(\mathit{thisId} := \texttt{enterCode}([\![\mathit{thisB}]\!]_{id})) \\
\quad (\mathit{addr1}, \mathit{addr2}) \overset{let}{=} (\texttt{new}\ \textsf{Address},\ \texttt{new}\ \textsf{Address}) \\
\quad \mathit{call} \overset{let}{=} \mathit{tailBlock.func.createCall}(\texttt{call}([\![\mathit{fun}]\!]_{id}, \mathit{thisId}, [\![\mathit{args}]\!]_{id}) \\
\qquad\quad @\ \mathit{addr1},\ \mathit{addr2}, [\![x]\!]_{id}) \\
\quad \mathit{cfg.addEdge}(\mathit{tailBlock}, \mathit{call}) \\
\quad (\mathit{call.afterCall}, L[\\
\qquad \texttt{ThrowLabel} \mapsto L(\texttt{ThrowLabel}) \cup \{\mathit{call}, \mathit{tailBlock}\}, \\
\qquad \texttt{AfterCatchLabel} \mapsto L(\texttt{AfterCatchLabel}) \\
\qquad\qquad\qquad\qquad \cup \{\mathit{call.afterCatch}\} \\
\quad])
\end{array}
$$

$$
\begin{array}{l}
[\![x = \mathit{cons}(\mathit{args})]\!]_{stmt}(f, B, L) \\
= \mathit{tailBlock} \overset{let}{=} \mathit{getTail}(B, f) \\
\quad (\mathit{addr1}, \mathit{addr2}) \overset{let}{=} (\texttt{new}\ \textsf{Address},\ \texttt{new}\ \textsf{Address})\quad (\mathit{irnew}) \\
\quad \mathit{call} \overset{let}{=} \mathit{tailBlock.func.createCall}(\\
\qquad \texttt{construct}([\![\mathit{cons}]\!]_{id}, [\![\mathit{args}(0)]\!]_{id}, [\![\mathit{args}(1)]\!]_{id}) \\
\qquad @\ \mathit{addr1}, \mathit{addr2}, [\![x]\!]_{id} \\
\quad) \\
\quad (\mathit{call.afterCall}, L[\\
\qquad \texttt{ThrowLabel} \mapsto L(\texttt{ThrowLabel}) \cup \{\mathit{call}, \mathit{tailBlock}\}, \\
\qquad \texttt{AfterCatchLabel} \mapsto L(\texttt{AfterCatchLabel}) \\
\qquad\qquad\qquad\qquad \cup \{\mathit{call.afterCatch}\} \\
\quad])
\end{array}
$$

$$
\begin{array}{l}
[\![x = \textsf{delete}\ y]\!]_{stmt}(f, B, L) \\
= \mathit{tailBlock} \overset{let}{=} \mathit{getTail}(B, f) \\
\quad \mathit{tailBlock.createInst}([\![x]\!]_{id} := \texttt{delete}([\![y]\!]_{id})) \\
\quad (\mathit{tailBlock}, L[\texttt{ThrowLabel} \mapsto L(\texttt{ThrowLabel}) \cup \{\mathit{tailBlock}\}])
\end{array}
$$

$$
\begin{array}{l}
[\![x = \textsf{delete}\ y[z]]\!]_{stmt}(f, B, L) \\
= \mathit{tailBlock} \overset{let}{=} \mathit{getTail}(B, f) \\
\quad \mathit{tailBlock.createInst}([\![x]\!]_{id} := \texttt{delete}([\![y]\!]_{id}, [\![z]\!]_{expr})) \\
\quad (\mathit{tailBlock}, L[\texttt{ThrowLabel} \mapsto L(\texttt{ThrowLabel}) \cup \{\mathit{tailBlock}\}])
\end{array}
$$

$$
\begin{array}{l}
[\![x = e]\!]_{stmt}(f, B, L) \\
= \mathit{tailBlock} \overset{let}{=} \mathit{getTail}(B, f) \\
\quad \mathit{tailBlock.createInst}([\![x]\!]_{id} := [\![e]\!]_{expr}) \\
\quad (\mathit{tailBlock}, L[\texttt{ThrowLabel} \mapsto L(\texttt{ThrowLabel}) \cup \{\mathit{tailBlock}\}])
\end{array}
$$

$$
\begin{aligned}
&[\![\mathsf{if}(cond)\ trueIR]\!]_{stmt}(f, B, L)\\
&= falseB \overset{let}{=} f.createBlock\\
&\quad cfg.addEdge(B, falseB)\\
&\quad cfgCond \overset{let}{=} [\![cond]\!]_{expr}\\
&\quad trueB.createInst(\mathtt{assert}(cfgCond))\\
&\quad falseB.createInst(\mathtt{assert}(\neg cfgCond))\\
&\quad (B', L') \overset{let}{=} [\![trueIR]\!]_{stmt}(f, trueB, L)\\
&\quad endB \overset{let}{=} f.createBlock\\
&\quad cfg.addEdge(\{falseB,\ B'\}, endBlock)\\
&\quad (endB, L'[\mathtt{ThrowLabel} \mapsto L'(\mathtt{ThrowLabel})\\
&\qquad\qquad\qquad \cup \{trueB,\ falseB\}])
\end{aligned}
$$

$$
\begin{aligned}
&[\![\mathsf{if}(cond)\ trueIR\ \mathsf{else}\ falseIR]\!]_{stmt}(f, B, L)\\
&= falseB \overset{let}{=} f.createBlock\\
&\quad cfg.addEdge(B, falseB)\\
&\quad cfgCond \overset{let}{=} [\![cond]\!]_{expr}\\
&\quad trueB.createInst(\mathtt{assert}(cfgCond))\\
&\quad falseB.createInst(\mathtt{assert}(\neg cfgCond))\\
&\quad (B', L') \overset{let}{=} [\![trueIR]\!]_{stmt}(f, trueB, L)\\
&\quad endB \overset{let}{=} f.createBlock\\
&\quad (B'', L'') \overset{let}{=} [\![falseIR]\!]_{stmt}(f, falseB, L')\\
&\quad cfg.addEdge(B'\ {+}{+}\ B'', endB)\\
&\quad (endB, L''[\mathtt{ThrowLabel} \mapsto L''(\mathtt{ThrowLabel})\\
&\qquad\qquad\qquad \cup \{trueB,\ falseB\}])
\end{aligned}
$$

$$
\begin{aligned}
&[\![l : \{s\}]\!]_{stmt}(f, B, L)\\
&= b \overset{let}{=} f.createBlock\\
&\quad (B', L') \overset{let}{=} [\![s]\!]_{stmt}(f, B, L)\\
&\quad label \overset{let}{=} l.name\\
&\quad cfg.addEdge(B', b)\\
&\quad cfg.addEdge(L'(label), b)\\
&\quad (b, L' - l)
\end{aligned}
$$

$$
\begin{aligned}
&[\![\mathsf{return}\ \underline{x}^?]\!]_{stmt}(f, B, L)\\
&= tailBlock \overset{let}{=} getTail(B, f)\\
&\quad tailBlock.createInst(\mathtt{return}())\\
&\quad (\emptyset, L[\mathtt{RetLabel} \mapsto L(\mathtt{RetLabel}) \cup \{tailBlock\}])
\end{aligned}
$$

$$
\begin{aligned}
&[\![\mathsf{return}\ x]\!]_{stmt}(f, B, L)\\
&= tailBlock \overset{let}{=} getTail(B, f)\\
&\quad tailBlock.createInst(\mathtt{return}([\![x]\!]_{id}))\\
&\quad (\emptyset, L[\mathtt{RetLabel} \mapsto L(\mathtt{RetLabel}) \cup \{tailBlock\}])
\end{aligned}
$$

$$
\begin{aligned}
&[\![o[p] = v]\!]_{stmt}(f, B, L)\\
&= tailBlock \overset{let}{=} getTail(B, f)\\
&\quad tailBlock.createInst([\![o]\!]_{expr}[[\![p]\!]_{expr}] := [\![v]\!]_{expr})\\
&\quad (tailBlock, L[\mathtt{ThrowLabel} \mapsto L(\mathtt{ThrowLabel}) \cup \{tailBlock\}])
\end{aligned}
$$

$$
\begin{aligned}
&[\![\mathsf{throw}\ e]\!]_{stmt}(f, B, L)\\
&= tailBlock \overset{let}{=} getTail(B, f)\\
&\quad tailBlock.createInst(\mathtt{throw}([\![e]\!]_{expr}))\\
&\quad (\emptyset, L[\mathtt{ThrowLabel} \mapsto L(\mathtt{ThrowLabel})] \cup \{tailBlock\})
\end{aligned}
$$

$$
\begin{aligned}
&[\![\mathsf{while}(e)\ s]\!]_{stmt}(f, B, L)\\
&\qquad = tailBlock \overset{let}{=} getTail(B, f)\\
&\qquad\quad head \overset{let}{=} f.createBlock\\
&\qquad\quad body \overset{let}{=} f.createBlock\\
&\qquad\quad out \overset{let}{=} f.createBlock\\
&\qquad\quad body.createInst(\mathtt{assert}(e))\\
&\qquad\quad out.createInst(\mathtt{assert}(\neg e))\\
&\qquad\quad cfg.addEdge(tailBlock, head)\\
&\qquad\quad cfg.addEdge(head, body)\\
&\qquad\quad cfg.addEdge(head, out)\\
&\qquad\quad (B', L') \overset{let}{=} [\![s]\!]_{stmt}(cfg, f, [body], L)\\
&\qquad\quad cfg.addEdge(B', head)\\
&\qquad\quad (out, L'[\mathtt{ThrowLabel} \mapsto L'(\mathtt{ThrowLabel})\\
&\qquad\qquad\qquad \cup \{body,\ out\}])
\end{aligned}
$$

$$
\begin{aligned}
&[\![stmts]\!]_{stmt^*}(f, B, L)\\
&\qquad = fold(stmts)((B, L))\\
&\qquad\qquad (\lambda((B', L'), stmt) \Rightarrow [\![stmt]\!]_{stmt}(f, B', L'))
\end{aligned}
$$

$$
\begin{aligned}
&[\![mem]\!]_{mem}(b, x)\\
&\qquad = lhs \overset{let}{=} [\![x]\!]_{id}\\
&\qquad\quad str \overset{let}{=} mem.prop.name\\
&\qquad\quad expr \overset{let}{=} [\![mem.expr]\!]_{emphexpr}\\
&\qquad\quad b.createInst(lhs[str] := expr)
\end{aligned}
$$

$$
\begin{aligned}
&[\![y]\!]_{elem}(b, x, k)\\
&\qquad = lhs \overset{let}{=} [\![x]\!]_{id}\\
&\qquad\quad expr \overset{let}{=} [\![x]\!]_{expr}\\
&\qquad\quad b.createInst(lhs[\text{“}k\text{”}] := expr)
\end{aligned}
$$

$$[\![x \otimes y]\!]_{expr} = [\![x]\!]_{expr} \otimes [\![y]\!]_{expr}$$

$$[\![\ominus x]\!]_{expr} = \ominus[\![x]\!]_{expr}$$

$$[\![o[e]]\!]_{expr} = [\![o]\!]_{expr}[[\![e]\!]_{expr}]$$

$$[\![\mathsf{this}]\!]_{expr} = \mathsf{this}$$

$$[\![v]\!]_{expr} = v$$

$[\![x]\!]_{id}$ = let *kind* be a VarKind based on the information from `CapturedVariableCollector`
`new` CFGId($x.name$, *kind*)

Bibliography

[1] Research on software analysis for error-free computing. `http://rosaec.snu.ac.kr`.

[2] ECMA-262: ECMAScript Language Specification, Edition 5.1. `http://www.ecma-international.org/ecma-262/5.1`, 2011.

[3] AT&T. Graphviz – graph visualization software. `http://www.graphviz.org`.

[4] SungGyeong Bae, Hyunghun Cho, Inho Lim, and Sukyoung Ryu. $\mathsf{SAFE}_{\mathsf{WAPI}}$: Web API misuse detector for web applications. In *FSE 2014*, pages 507–517.

[5] WaiTing Cheung, Sukyoung Ryu, and Sunghun Kim. Development nature matters: An empirical study of code clones in JavaScript applications. *Empirical Software Engineering*, 21(2):517–564, April 2016.

[6] Junhee Cho and Sukyoung Ryu. JavaScript module system: Exploring the design space. In *Modularity 2014*, pages 229–240.

[7] Patrick Cousot and Radhia Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *POPL 1977*, pages 238–252.

[8] Asger Feldthaus, Max Schäfer, Manu Sridharan, Julian Dolby, and Frank Tip. Efficient construction of approximate call graphs for javascript ide services. In *ICSE 2013*, pages 752–761.

[9] Linux Foundation. Tizen. `https://www.tizen.org`.

[10] IBM Research. T.J. Watson Libraries for Analysis (WALA). `http://wala.sf.net`, 2006.

[11] Seonghoon Kang and Sukyoung Ryu. Formal specification of a JavaScript module system. In *OOPSLA 2012*, pages 621–638.

[12] Vineeth Kashyap, Kyle Dewey, Ethan A. Kuefner, John Wagner, Kevin Gibbons, John Sarracino, Ben Wiedermann, and Ben Hardekopf. JSAI: A static analysis platform for JavaScript. In *FSE 2014*, pages 121–132.

[13] Yoonseok Ko, Hongki Lee, Julian Dolby, and Sukyoung Ryu. Practically tunable static analysis framework for large-scale JavaScript applications. In *ASE 2015*, pages 541–551.

[14] Oracle Labs. Web-based vulnerability detection. `https://labs.oracle.com`.

[15] Hongki Lee, Sooncheol Won, Joonho Jin, Junhee Cho, and Sukyoung Ryu. SAFE: Formal specification and implementation of a scalable analysis framework for ECMAScript. In *FOOL 2012*.

[16] Benjamin Livshits, Manu Sridharan, Yannis Smaragdakis, Ondřej Lhoták, J. Nelson Amaral, Bor-Yuh Evan Chang, Samuel Z. Guyer, Uday P. Khedker, Anders Møller, and Dimitrios Vardoulakis. In defense of soundiness: A manifesto. *Communication of ACM*, 58(2):44–46, 2015.

[17] Microsoft. Typescript. `http://www.typescriptlang.org`, 2012.

[18] Changhee Park, Hongki Lee, and Sukyoung Ryu. All about the `with` statement in JavaScript: Removing `with` statements in JavaScript applications. In *DLS 2013*, pages 73–84.

[19] Changhee Park and Sukyoung Ryu. Scalable and precise static analysis of JavaScript applications via loop-sensitivity. In *ECOOP 2015*, pages 735–756.

[20] Changhee Park, Sooncheol Won, Joonho Jin, and Sukyoung Ryu. Static analysis of JavaScript web applications in the wild via practical dom modeling. In *ASE 2015*, pages 552–562.

[21] Daejun Park, Andrei Ştefănescu, and Grigore Roşu. KJS: A complete formal semantics of JavaScript. In *PLDI 2015*, pages 428–438.

[22] Jihyeok Park. Javascript api misuse detection by using typescript. In *Modularity (SRC) 2014*, pages 11–12.

[23] Joonyoung Park, Inho Lim, and Sukyoung Ryu. Battles with false positives in static analysis of JavaScript web applications in the wild. In *ICSE 2016*, pages 61–70.

[24] Gregor Richards, Christian Hammer, Brian Burg, and Jan Vitek. The eval that men do: A large-scale study of the use of eval in JavaScript applications. In *ECOOP 2011*, pages 52–78.

[25] Gregor Richards, Sylvain Lebresne, Brian Burg, and Jan Vitek. An analysis of the dynamic behavior of javascript programs. In *PLDI 2010*, pages 1–12.

[26] Sukyoung Ryu. Journey to find bugs in JavaScript web applications in the wild. In *ICFP 2016*. ACM.

[27] W3C. Document Object Model Activity Statement. `http://www.w3.org/DOM/Activity`, 1998.