

---

# Appendix for ABSINT-AI: Agentic Heap Abstractions for Abstract Interpretation

---

Anonymous Author(s)

Affiliation

Address

email

## 1 A Abstract Interpretation Details

### 2 A.1 Analysis details

3 **Functions** In Javascript, functions are stored as objects on the heap. We include a `__code__` property  
4 storing the function body to be executed. At the beginning of the analysis, ABSINT-AI scans the  
5 entire program, and generates a *schema* for each function. The schema for each function contains  
6 which variables are local to the function and which variables are accessed by other functions. We  
7 refer to variables that are local as *private*, and variables that are accessed by other functions as *shared*.  
8 Each time a function is executed, an environment is initialized according to the schema for that  
9 function. When a function is defined, is initialized with a `__hf__` field set to the current heap frame.  
10 The `__hf__` field is used to model scopes and closures. When the function returns, the stack frame  $\sigma$   
11 is popped from the stack, and the stack pointer is decremented.

12 **Scopes and Closures** Whenever a function is called, a new stack frame  $\sigma$  is pushed, along with a  
13 corresponding heap frame. The stack pointer for the current stack frame is updated to point to  $\sigma$ .  
14 The private variables for that function are stored in the stack frame  $\sigma$ , and any shared variables are  
15 stored in the heap frame. The heap frame is initialized with a parent field `__parent__` which is used  
16 to model the scope chain. The `__parent__` field points to the `__hf__` field for the function being  
17 initialized.

18 To lookup a variable name in the environment, ABSINT-AI first checks the current stack frame. If it  
19 finds a value for the variable, it returns the value. If it doesn't, it checks the corresponding heap frame  
20 for the stack frame, and then follows the chain of `__parent__` pointers until it finds the variable.

21 **Recursion** ABSINT-AI keeps track of all functions that have been called but have not finished  
22 executing yet. Whenever it encounters a recursive call, ABSINT-AI sets the return value to a recursive  
23 placeholder and stores a hash of the function that is called. When the function returns, ABSINT-AI  
24 checks the return values and any allocated heap objects for recursive placeholders for the function  
25 and fills them in with the return values.

### 26 A.2 Environment

27 In this section we describe how ABSINT-AI represents the abstract state. We define concrete and  
28 abstract values.  $H_L$  refers to the concrete heap,  $H_G$  refers to the global heap, and  $\sigma$  refers to the  
29 stack.  $\tau$  is an abstract type,  $C$  refers to constants,  $obj$  and  $\widetilde{obj}$  refer to concrete and abstract objects.

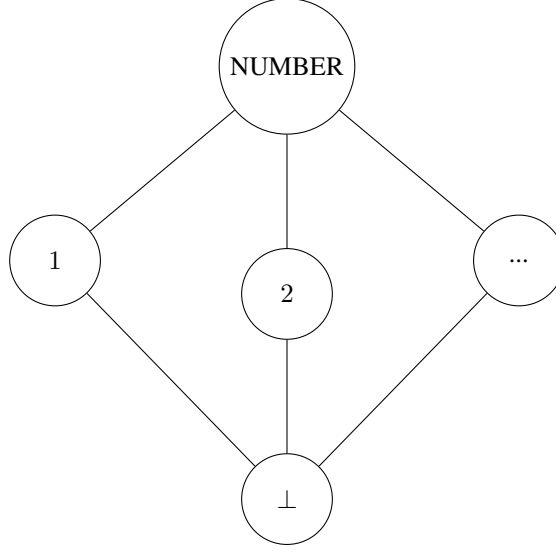


Figure 1: Number Lattice.

30  $val$  and  $\widetilde{val}$  refer to the values that a variable can take.

$$\begin{aligned}
 val &::= a \mid obj \mid \widetilde{val} \\
 \widetilde{val} &::= C \mid \widetilde{a} \mid \tau \mid \widetilde{obj} \\
 \tau &::= Bool \mid Null \mid Num \mid String \\
 obj &::= \tau \rightarrow val \mid C \rightarrow val \\
 \widetilde{obj} &::= \tau \rightarrow \widetilde{val} \mid C \rightarrow \widetilde{val} \\
 H_L &::= a \rightarrow val \\
 H_G &::= \widetilde{a} \rightarrow \widetilde{val} \\
 \sigma &::= C \rightarrow val
 \end{aligned}$$

### 31 A.3 Syntax

$$\begin{aligned}
 op &::= + \mid - \mid \div \mid \cdot \mid \dots \\
 E &::= id \mid E.field \mid E[E] \mid foo(E) \mid E_1[E_2](E_3, E_4, \dots) \mid \text{function}(x_0, x_1, \dots)\{S\} \\
 &\quad \mid \text{new } foo(E_1, E_2, \dots) \mid C \mid \{f : E\} \\
 varDef &::= \text{var } id = E \mid \text{let } id = E \mid \text{const } id = E \\
 Stmt &::= varDef \mid id = E \mid \\
 &\quad E.f = E \mid E[E] = E \mid \text{def } foo(x_1, x_2, \dots, x_n)\{Stmt\} \mid \\
 &\quad \text{if } (E)\{Stmt\} \text{ else } \{Stmt\} \mid \text{class } foo\{Stmt\} \mid \\
 &\quad \text{return } E \mid \text{for } (varDef; E; Stmt)\{Stmt\} \\
 &\quad \text{for } (varDef \text{ in } E)\{Stmt\} \mid \text{while } (E)\{Stmt\} \mid Stmt; Stmt
 \end{aligned}$$

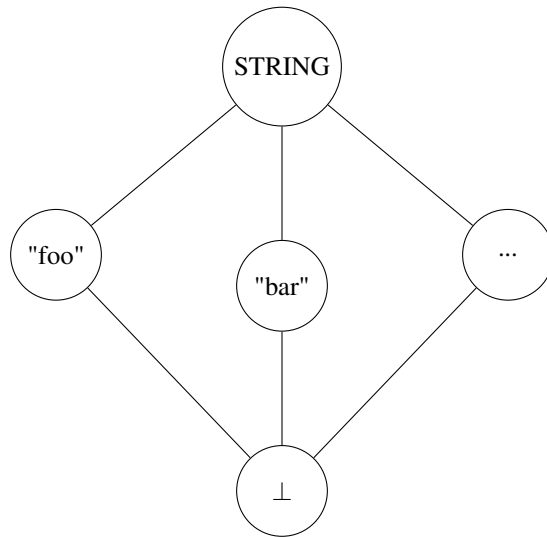


Figure 2: String Lattice.

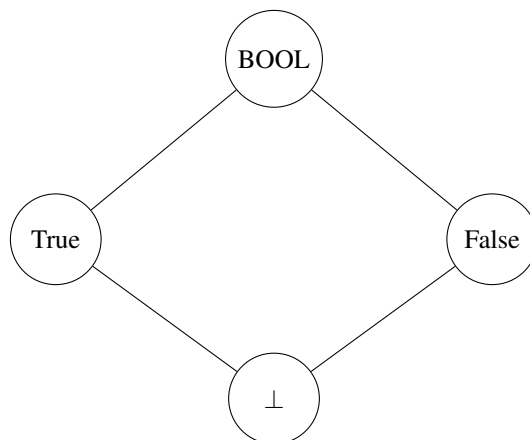


Figure 3: Boolean Lattice.

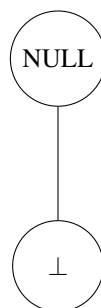


Figure 4: Null Singleton.

## 32 A.4 Semantics

### 33 A.4.1 Functions

34 This section is several functions we use, such looking up a variable name and initializing a new  
 35 schema for a function.

$$\begin{array}{l}
 \text{lookup(id)} \quad \frac{s \equiv \emptyset \quad \theta = \emptyset}{\langle \text{lookup}(H_L, H_G, s, id) \rightarrow \theta \rangle} \\
 \\
 \frac{s \in H_L \quad id \in H_L(s) \quad \theta = s}{\langle \text{lookup}(H_L, H_G, s, id) \rightarrow \theta \rangle} \\
 \\
 \frac{s \in H_G \quad id \in H_G(s) \quad \theta = s}{\langle \text{lookup}(H_L, H_G, s, id) \rightarrow \theta \rangle} \\
 \\
 \frac{s \in H_L \quad id \notin H_L(s) \quad \theta = \text{lookup}(H_L, H_G, H_L(s).par, id)}{\langle \text{lookup}(H_L, H_G, s, id) \rightarrow \theta \rangle} \\
 \\
 \frac{s \in H_G \quad id \notin H_G(s) \quad \theta = \text{lookup}(H_L, H_G, H_G(s).par, id)}{\langle \text{lookup}(H_L, H_G, s, id) \rightarrow \theta \rangle} \\
 \\
 \text{initialize(schema)} \quad \frac{H_L[a \mapsto \{schema.public, par \mapsto \sigma.hf\}] \quad \sigma'._secret \mapsto \{schema.secret\} \quad \sigma'.hf \mapsto a}{\text{initialize(schema)} \rightarrow H_L, H_G, \sigma :: \sigma'} \\
 \\
 \text{return\_from\_schema} \quad \frac{\sigma \equiv \sigma' :: v}{\text{return\_from\_schema} \rightarrow H_L, H_G, \sigma'}
 \end{array}$$

### 36 A.4.2 Small-step semantics

$$\langle H_L, H_G, \sigma, S \rangle \rightarrow \langle H'_L, H'_G, \sigma', S' \rangle$$

$$\begin{array}{l}
\text{id} \quad \frac{}{\langle H_L, H_G, \sigma, \text{id} \rangle \rightarrow \langle H_L, H_G, \sigma, \text{lookup}(\text{id}) \rangle} \\
\text{E.field} \quad \frac{\langle H_L, H_G, \sigma, E \rangle \rightarrow \langle H'_L, H'_G, \sigma', V \rangle}{\langle H_L, H_G, \sigma, \text{E.field} \rangle \rightarrow \langle H'_L, H'_G, \sigma', \text{get}(V, \text{field}) \rangle} \\
E_1[E_2] \quad \frac{\langle H_L, H_G, \sigma, E_2 \rangle \rightarrow \langle H'_L, H'_G, \sigma', V_2 \rangle \quad \langle H'_L, H'_G, \sigma', E_1 \rangle \rightarrow \langle H''_L, H''_G, \sigma'', V_1 \rangle}{\langle H_L, H_G, \sigma, E_1[E_2] \rangle \rightarrow \langle H'_L, H'_G, \sigma', \text{get}(V_1, V_2) \rangle} \\
\text{foo}(E_0, E_1, \dots) \quad \frac{\langle \text{lookup}(\text{foo}) \rightarrow V, V\_type \equiv \text{Function} \rangle \quad \langle H_L, H_G, \sigma, E_0, E_1, \dots \rangle \rightarrow \langle H'_L, H'_G, \sigma', V_0, V_1, \dots \rangle}{\langle H_L, H_G, \sigma, \text{foo}(E_0, E_1, \dots) \rangle \rightarrow \langle H'_L[x_0 \mapsto V_0, x_1 \mapsto V_1, \dots], H'_G, \sigma', \text{initialize}(V\_code); V\_code \rangle} \\
E_1[E_2](E_3, E_4, \dots) \quad \frac{\langle H_L, H_G, \sigma, E_0, E_1, \dots \rangle \rightarrow \langle H'_L, H'_G, \sigma', V_0, V_1, V_2, \dots \rangle \quad \langle \text{get}(V_0, V_1) \rightarrow V, V\_type \equiv \text{Function} \rangle}{\langle H_L, H_G, \sigma, \text{foo}(E_0, E_1, \dots) \rangle \rightarrow \langle H'_L[x_0 \mapsto V_0, x_1 \mapsto V_1, \dots], H'_G, \sigma'[\text{this} \mapsto V_0], V\_code \rangle} \\
\text{function}(x_0, x_1, \dots)\{S\} \quad \frac{}{\langle H_L, H_G, \sigma, \text{function}(x_0, x_1, \dots) \rangle \rightarrow \langle H'_L[a' \mapsto \{\dots, \text{prototype} : a\}, a \mapsto \dots], H'_G, \sigma', a' \rangle} \\
\text{new foo}(E_0, E_1, \dots) \quad \frac{\langle \text{lookup}(\text{foo}) \rightarrow V \rangle \quad \langle V\_type \equiv \text{Class} \rangle \quad \langle E_0, E_1, \dots \rangle \rightarrow \langle V_0, V_1, \dots \rangle}{\langle H_L, H_G, \sigma, \text{new foo}(E_0, E_1, \dots) \rangle \rightarrow \langle H'_L, H'_G, \sigma'[\text{this} \mapsto V], \text{init}(); \text{get}(\text{prototype}(V), \text{constructor})(V_0, V_1, \dots) \rangle} \\
\frac{\langle \text{lookup}(\text{foo}) \rightarrow V \rangle \quad \langle V\_type \equiv \text{Function} \rangle \quad \langle E_0, E_1, \dots \rangle \rightarrow \langle V_0, V_1, \dots \rangle}{\langle H_L, H_G, \sigma, \text{new foo}(E_0, E_1, \dots) \rangle \rightarrow \langle H'_L, H'_G, \sigma', V\_code(V_0, V_1, \dots) \rangle} \\
\{f_1 : E_1, f_2 : E_2, \dots\} \quad \frac{\langle H_L, H_G, \sigma, E_1, E_2, \dots \rangle \rightarrow \langle H'_L, H'_G, \sigma', V_1, V_2, \dots \rangle}{\langle H_L, H_G, \sigma, \{f_1 : E_1, f_2 : E_2, \dots\} \rangle \rightarrow \langle H_L[a \mapsto \{f_1 : V_1, f_2 : V_2, \dots, \_type : \text{object}\}], H_G, \sigma, a \rangle} \\
(\text{var } x = E) \quad \frac{\langle H_L, H_G, \sigma, E \rangle \rightarrow \langle H'_L, H'_G, \sigma', V \rangle \quad \theta = \text{lookup}(x) \quad \theta \in H_L \quad \text{fr} = H_L[\theta] \quad \text{fr}' = \text{fr}[\text{id} \mapsto V]}{\langle H_L, H_G, \sigma, x = E \rangle \rightarrow \langle H'_L[\theta \mapsto \text{fr}'], H'_G, \sigma', \text{skip} \rangle} \\
\frac{\langle H_L, H_G, \sigma, E \rangle \rightarrow \langle H'_L, H'_G, \sigma', V \rangle \quad \theta = \text{lookup}(x) \quad \theta \in H_G \quad \text{fr} = H_G[\theta] \quad \text{fr}' = \text{fr}[\text{id} \mapsto V \cup \text{fr}[\text{id}]]}{\langle H_L, H_G, \sigma, x = E \rangle \rightarrow \langle H'_L, H'_G[\theta \mapsto \text{fr}'], \sigma', \text{skip} \rangle} \\
(x.f = E) \quad \frac{\text{lookup}(x) \equiv a \quad \theta = H_L(a) \quad \langle H_L, H_G, \sigma, E \rangle \rightarrow \langle H'_L, H'_G, \sigma', V \rangle}{\langle H_L, H_G, \sigma, x.f = E \rangle \rightarrow \langle H'_L[\theta[f \mapsto V]], H'_G, \sigma', \text{skip} \rangle} \\
\frac{\text{lookup}(x) \equiv \tilde{a} \quad \tilde{\theta} = H_G(\tilde{a}) \quad \langle H_L, H_G, \sigma, E \rangle \rightarrow \langle H'_L, H'_G, \sigma', V \rangle}{\langle H_L, H_G, \sigma, x = E \rangle \rightarrow \langle H'_L, H'_G[\tilde{\theta}[f \mapsto V], \sigma', \text{skip} \rangle}
\end{array}$$

$$\begin{array}{c}
(\mathbf{x}[E] = E') \quad \frac{\text{lookup}(x) \equiv a \quad \theta = H_L(a) \quad \langle H_L, H_G, \sigma, E, E' \rangle \rightarrow \langle H'_L, H'_G, \sigma', V, V' \rangle}{\langle H_L, H_G, \sigma, x[f] = E \rangle \rightarrow \langle H'_L[\theta[V \mapsto V']], H'_G, \sigma', \text{skip} \rangle} \\
\\
\frac{\text{lookup}(x) \equiv \tilde{a} \quad \tilde{\theta} = H_G(\tilde{a}) \quad \langle H_L, H_G, \sigma, E, E' \rangle \rightarrow \langle H'_L, H'_G, \sigma', V, V' \rangle}{\langle H_L, H_G, \sigma, x = E \rangle \rightarrow \langle H'_L, H'_G[\tilde{\theta}[V \mapsto V']], \sigma', \text{skip} \rangle} \\
\\
(\text{def foo}(x_0, x_1, \dots, x_n) \{ \text{Stmt} \}) \quad \frac{\theta = \text{lookup}(\text{foo}) \quad \theta \in \sigma}{\langle H_L, H_G, \sigma, x[f] = E \rangle \rightarrow \langle H_L[a \mapsto \dots, \text{prototype} : a', a' \mapsto \{\}], H_G, \sigma[\theta \mapsto a], \text{skip} \rangle} \\
\\
\frac{\theta = \text{lookup}(\text{foo}) \quad \theta \in H_L}{\langle H_L, H_G, \sigma, x[f] = E \rangle \rightarrow \langle H_L[a \mapsto \dots, \text{prototype} : a', a' \mapsto \{\}], \theta \mapsto a, H_G, \sigma, \text{skip} \rangle} \\
\\
\frac{\theta = \text{lookup}(\text{foo}) \quad \theta \in H_G}{\langle H_L, H_G, \sigma, x[f] = E \rangle \rightarrow \langle H_L, H_G[a \mapsto \dots, \text{prototype} : a', a' \mapsto \{\}], \theta \mapsto \theta \cup a, \sigma, \text{skip} \rangle} \\
\\
(\mathbf{x}[E] = E') \quad \frac{\text{lookup}(x) \equiv a \quad \theta = H_L(a) \quad \langle H_L, H_G, \sigma, E, E' \rangle \rightarrow \langle H'_L, H'_G, \sigma', V, V' \rangle}{\langle H_L, H_G, \sigma, x[f] = E \rangle \rightarrow \langle H'_L[\theta[V \mapsto V']], H'_G, \sigma', \text{skip} \rangle} \\
\\
\frac{\text{lookup}(x) \equiv \tilde{a} \quad \tilde{\theta} = H_G(\tilde{a}) \quad \langle H_L, H_G, \sigma, E, E' \rangle \rightarrow \langle H'_L, H'_G, \sigma', V, V' \rangle}{\langle H_L, H_G, \sigma, x = E \rangle \rightarrow \langle H'_L, H'_G[\tilde{\theta}[V \mapsto V']], \sigma', \text{skip} \rangle} \\
\\
\text{if (E) \{ Stmt \} else \{ Stmt' \}} \quad \frac{\langle H_L, H_G, \sigma, E \rangle \rightarrow \langle H'_L, H'_G, \sigma', \text{False} \vee \emptyset \rangle}{\langle H_L, H_G, \sigma, \text{if (E) \{ Stmt \} else \{ Stmt' \}} \rangle \rightarrow \langle H'_L, H'_G, \sigma', \text{Stmt} \rangle} \\
\\
\frac{\langle H_L, H_G, \sigma, E \rangle \not\rightarrow \langle H'_L, H'_G, \sigma', \text{False} \vee \emptyset \rangle}{\langle H_L, H_G, \sigma, \text{if (E) \{ Stmt \} else \{ Stmt' \}} \rangle \rightarrow \langle H'_L, H'_G, \sigma', \text{Stmt'} \rangle} \\
\\
\text{class foo}[M_1, M_2, \dots, M_N] \quad \frac{\text{class\_obj} = \{ M_1, M_2, \dots, M_N \}}{\langle H_L, H_G, \sigma, \text{class foo}[M_1, M_2, \dots, M_N] \rangle \rightarrow \langle H_L[a \mapsto \text{class\_obj}], H_G, \sigma, \text{skip} \rangle} \\
\\
(\text{return } E) \quad \frac{\langle H_L, H_G, \sigma, E \rangle \rightarrow \langle H'_L, H'_G, \sigma', V \rangle}{\langle H_L, H_G, \sigma, \text{return } E \rangle \rightarrow \langle H'_L, H'_G, \sigma'[\text{returns} \mapsto \sigma'[\text{returns}] \cup V], \text{skip} \rangle} \\
\\
\text{for ((let | var] id in E) \{ Stmt \}) \quad \frac{\langle H_L, H_G, \sigma, E \rangle \rightarrow \langle H'_L, H'_G, \sigma', V \rangle \quad V.\_\text{proto\_} \equiv \emptyset \quad \text{isEmpty}(V) \equiv \text{True}}{\langle H_L, H_G, \sigma, \text{for ((let | var] id in E) \{ Stmt \}} \rangle \rightarrow \langle H'_L, H'_G, \sigma', \text{skip} \rangle} \\
\\
\frac{\langle H_L, H_G, \sigma, E \rangle \rightarrow \langle H'_L, H'_G, \sigma', V \rangle \quad V.\_\text{proto\_} \not\equiv \emptyset \quad \text{isEmpty}(V) \equiv \text{True}}{\langle H_L, H_G, \sigma, \text{for ((let | var] id in E) \{ Stmt \}} \rangle \rightarrow \langle H'_L, H'_G, \sigma', \text{for ((let | var] id in V.\_\text{proto\_}) \{ Stmt \}} \rangle} \\
\\
\frac{\langle H_L, H_G, \sigma, E \rangle \rightarrow \langle H'_L, H'_G, \sigma', V \rangle \quad V \equiv X :: V' \quad \text{varDef.type} \equiv \text{let}}{\langle H_L, H_G, \sigma, \text{for (let id in E) \{ Stmt \}} \rangle \rightarrow \langle H'_L, H'_G, \sigma'', \text{initialize(Stmt); let id=X; Stmt; for (let id in V') \{ Stmt \}} \rangle} \\
\\
\frac{\langle H_L, H_G, \sigma, E \rangle \rightarrow \langle H'_L, H'_G, \sigma', V \rangle \quad V \equiv X :: V' \quad \text{varDef.type} \equiv \text{var}}{\langle H_L, H_G, \sigma, \text{for (let id in E) \{ Stmt \}} \rangle \rightarrow \langle H'_L, H'_G, \sigma', \text{var id=X; Stmt; for (let id in V') \{ Stmt \}} \rangle}
\end{array}$$

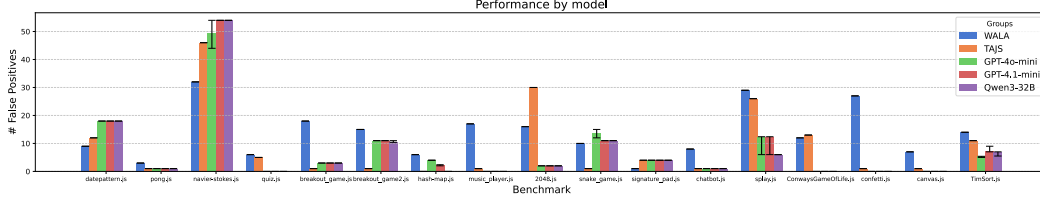


Figure 5: Performance per model on each benchmark program compared to WALA and TAJs.

38

$$\begin{aligned}
 \text{while (E) } \{ Stmt \} & \frac{\langle H_L, H_G, \sigma, E \rangle \rightarrow \langle H'_L, H'_G, \sigma', V \rangle \quad V \in \text{Falsey}}{\langle H_L, H_G, \sigma, \text{while (E) } \{ Stmt \} \rangle \rightarrow \langle H'_L, H'_G, \sigma', \text{skip} \rangle} \\
 & \frac{\langle H_L, H_G, \sigma, E \rangle \rightarrow \langle H'_L, H'_G, \sigma', V \rangle \quad V \notin \text{Falsey} \quad \langle H'_L, H'_G, \sigma', Stmt; \text{summarize}() \rangle \rightarrow \langle H'_L, H'_G, \sigma' \rangle}{\langle H_L, H_G, \sigma, \text{while (E) } \{ Stmt \} \rangle \rightarrow \langle H'_L, H'_G, \sigma', \text{skip} \rangle} \\
 & \frac{\langle H_L, H_G, \sigma, E \rangle \rightarrow \langle H'_L, H'_G, \sigma', V \rangle \quad V \notin \text{Falsey} \quad \langle H'_L, H'_G, \sigma', Stmt; \text{summarize}() \rangle \rightarrow \langle H''_L, H''_G, \sigma'' \rangle}{\langle H_L, H_G, \sigma, \text{while (E) } \{ Stmt \} \rangle \rightarrow \langle H''_L, H''_G, \sigma'', \text{while (E) } \{ Stmt \} \rangle}
 \end{aligned}$$

## 39 B Implementation and Dataset

40 **Implementation.** We implemented ABSINT-AI in 8049 lines of Python, and use Espree [1] to parse  
 41 the Javascript into an AST. We conducted the experiments on a Linux server with two AMD EPYC  
 42 7763 64-Core Processors, 128 cores, 1024GB RAM, and 4 NVIDIA RTX 6000 Ada Generation  
 43 GPUs.

### 44 B.1 Dataset

Table 1: Each program and a small description.

Program	#Lines	Description
CGOL.js	65	Conway’s Game of Life.
2048.js	234	The 2048 game implemented for the DOM.
breakout_game.js	158	An implementation of the Breakout arcade game for the DOM.
breakout_game2.js	91	A separate implementation of the Breakout arcade game for the DOM.
datepattern.js	91	Testing date string equality
hash-map.js	577	A JavaScript implementation of a HashMap.
confetti.js	400	Confetti animations in the DOM.
pong.js	243	Pong game in the DOM.
snake_game.js	102	Snake game in the DOM.
books.js	504	A library for storing books.
FlashSort.js	84	Flash Sort.
math_sprint.js	345	Math calculations in the DOM.
drawing-app.js	442	A drawing app in the DOM.
TimSort.js	113	Tim Sort.
navier-stokes.js	385	Fluid dynamics simulation using a simplified implementation of the Navier–Stokes equations.
music_player.js	196	Picking between songs to display in the DOM.
splay.js	406	An implementation of a Splay Tree in JavaScript.

## 45 References

46 [1] brettz9. Brettz9/espree: An esprima-compatible javascript parser. URL <https://github.com/brettz9/espree>.  
 47