Some Invariants to Consider

Mark S. Miller, Agoric

tc39 July 2020

Taxonomy of invariant-likes

Genuine “must” invariants

Questionable invariants

Almost “legacy carve-out” invariants

Almost “should” invariants

Almost “to repair” invariants

Genuine “must” Invariants

Memory safety — unforgeable refs
  - encapsulation: #name, proxy, weakmap

Realm isolation
  - disjoint subgraphs (but for creation)
  - no communication channel (registered symbols)

“The” Object invariants (Proxy enforcement)

Membrane graph impenetrability

Questionable Invariants

`Object.prototype.toString.call(obj)`
was an unforgeable brand check

`typeof a === typeof b` implies
`a == b` iff `a === b`

`typeof (a - b) === “number”`

Object allocation always succeeds

Almost “legacy carve-out” Invariants

	Strict	**Sloppy**

Mandatory	Purely Statically Scoped
	caller insensitivity
	closure encapsulation
		`.caller`
		`.arguments`
		nested function decl
		`with`

Normative Optional	`document.all` “Incumbent” host hooks	

Almost “should” Invariants

No hidden primordial state or IO, except

- `Date.now`, `Math.random`
- `Temporal` compromise

Exotic slots accessed almost only on `this`

- Membrane **practical** transparency
- `Promise.resolve(promiseForProxy)`

Interleaving points marked by `yield` or `await`

- repel Zalgo

Almost “to repair” Invariants

Transparent shimmability

  vs `Function.prototype.toString`

Deterministic parsing

  vs html comments

Non-standard properties deletable

Need test262 integration tests

Alex Vincent submitted
membrane integration tests

Questions?