

MiniPoly: Automatic Extraction of Efficient JavaScript Polyfill from Language Specification

Jungwoong Kim*, Youngmin Cho*, and Jihyeok Park

Department of Computer Science and Engineering, Korea University, South Korea



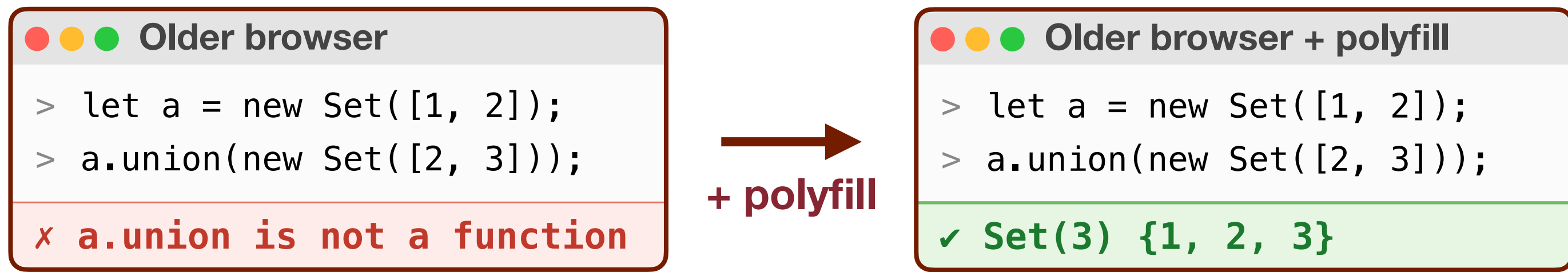
Artifact



Paper



1. Problem



A **polyfill** fills in missing built-in APIs, written in plain JavaScript

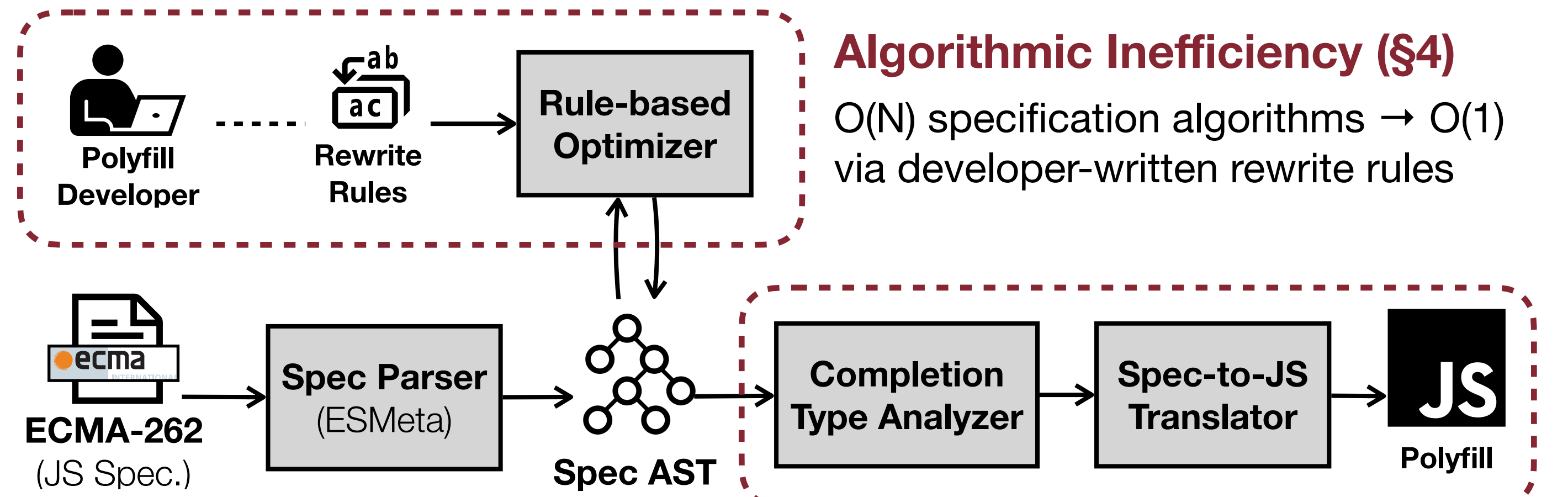


Manual translation **silently breaks the spec** Iterator never closed

```
- iteratorClose(iterator, 'normal', false); // core-js v3.47
+ iteratorClose(iterator.iterator, 'normal', false); // core-js v3.49
```

2. Structure

MiniPoly replaces the **by hand** step by automatically extracting polyfills from the specification



Algorithmic Inefficiency (§4)

$O(N)$ specification algorithms $\rightarrow O(1)$ via developer-written rewrite rules

Completion Records (§3)

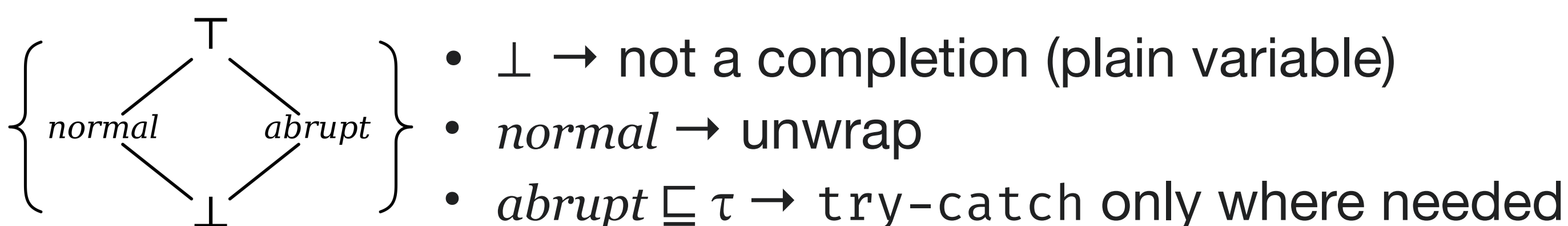
Translate spec-internal control flow into native JS reifying it with try-catch only where needed

3. Challenge I: Completion Records

- The spec reifies JS control flow as **Completion Records**.
- Naively translating with try-catch is unsound when: completion records are stored or passed as values.

```
7.4.11 IteratorClose ( iteratorRecord, completion )
3. Let innerResult be Completion(GetMethod(iterator, "return"))
4. If innerResult is a normal completion, then
   c. Set innerResult to Completion(Call(return, iterator)).
5. If completion is a throw completion, return ? completion
6. If innerResult is a throw completion, return ? innerResult
8. Return ? completion.
```

Flow-sensitive analysis infers each value's completion type



3. Let *innerResult* be Completion(GetMethod(*iterator*, "return"))

```
→ try { var innerResult = ... ; var innerResult_flag = "normal"; }
   catch (_innerResult_err) { ... ; var innerResult_flag = "abrupt"; }
```

8. Return ? *completion*.

```
→ if (completion_flag === "abrupt") throw completion;
   return completion; // analysis: completion = normal
```

4. Challenge II: Algorithmic Inefficiency

- The spec defines **behavior**, not an **optimized** algorithm - e.g. *Set.prototype.delete* scans $[[SetData]]$ in $O(N)$

```
Original Spec
1. For each element _e_ of _S_.[[SetData]], do
  1. If _e_ is not ~empty~ and SameValue(_e_, _value_) is *true*, then
    1. Replace the element of _S_.[[SetData]] whose value is _e_ ...
    1. Return *true*.
```

```
DSL
pattern: |
  1. For each element {{ elem : var }} of {{ base : ref }}, do
    1. If {{ elem }} is not ~empty~ and SameValue({{ elem }},
       {{ value : var }}) is *true*, {{ body : step }}
where:
  - isSetData(base)
  - { isSetData: "_ ([[SetData]] | .[[WeakSetData]]) .copy*" }
replace: 1. If SET_DATA_HAS({{ base }}, {{ value }}) is *true*, {{ body }}
subrules:
  - pattern: {{ elem }}
  - replace: {{ value }}
```

```
Rewritten Spec
1. If SET_DATA_HAS(_S_.[[SetData]], _value_) is *true*,
  1. Replace the element of _S_.[[SetData]] whose value is _value_ ...
  1. Return *true*.
```

- Syntactic matching alone is not enough: **Symbolic paths** decide which values to rewrite

<i>S</i> .[[SetData]]	<i>σ</i> .[[SetData]]	rewrite
resultSetData (copy of <i>S</i> .[[SetData]])	<i>σ</i> .[[SetData]].copy	rewrite
list (any other list)	<i>σ</i>	keep

5. Evaluation

- 89 built-ins** of ES2025 (Array, String, (Weak) Set, (Weak) Map, Promise) extracted and compared against **core-js** and **es-shims**

- RQ1: Correctness** (Test262^[1] pass rate)

	core-js	es-shims	MiniPoly
Array	98.08%	94.90%	98.93%
String	96.55%	94.40%	100.00%
Map	92.29%	91.70%	98.97%
Set	94.68%	79.50%	98.40%
Promise	91.30%	55.50%	99.40%
Total	95.70%	90.90%	99.80%

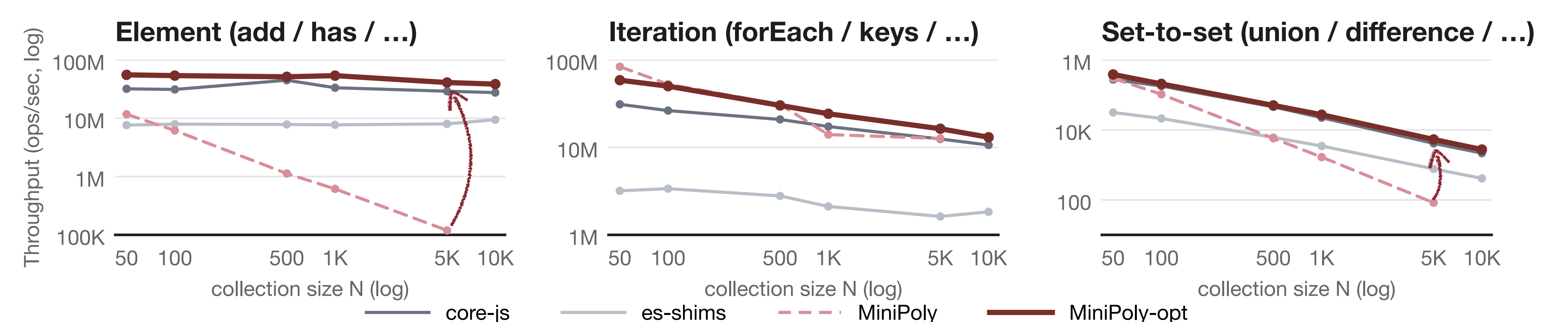
More correct: highest pass rate on every built-in

- RQ2: Automation** (manual effort as LoC)

es-shims	32,836 (25×)
core-js	14,312 (11×)
MiniPoly	1,295

Less effort: 11× / 25× less manually maintained code

- RQ3: Efficiency** (Set operations/sec, log-log)



As fast: 17 rewrite rules (195 LoC) turn $O(N)$ spec loops into $O(1)$ $\rightarrow$ On par with hand-optimized core-js

- RQ4: Forward Compatibility**

Proposal	Tests	core-js	MiniPoly
Map Upsert (ES2026)	65	61	65
WeakMap Upsert (ES2026)	77	58	77
Await Dictionary (Stage 2.7)	24	-	24

Future-ready: proposals supported without extra effort

- RQ5: Real-world Usability**

7 well-known libraries (lodash, express, axios, ws, ...)

10,772 / 10,795 tests passed

same as core-js (99.8%), no slowdown

23 misses: `_.isNative`, which no polyfill can pass

Drop-in ready: replaces built-ins without breaking libraries

[1] <https://github.com/tc39/test262>