String Analysis for Javascript programs using TAJS

Dr. L. Lu
Koby Picker
Christian Maldonado

Week 3:
Widening Operators, and TAJS

Fixed Point Analysis

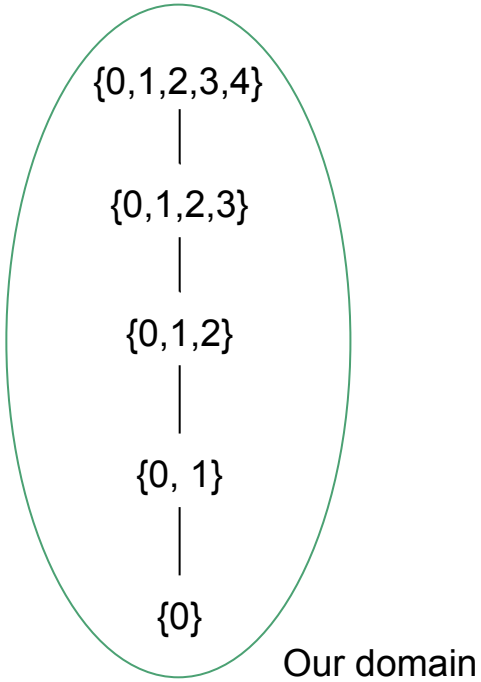
$i_0 = \{0\}$

$i_1 = \{0, 1\}$

$i_2 = \{0, 1, 2\}$

...

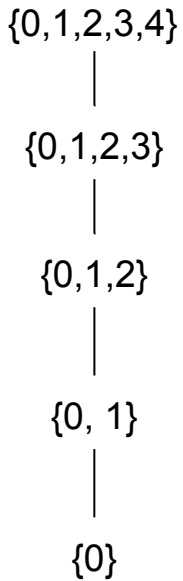
```
var i = 0;
while (true) {
    i++;
}
```

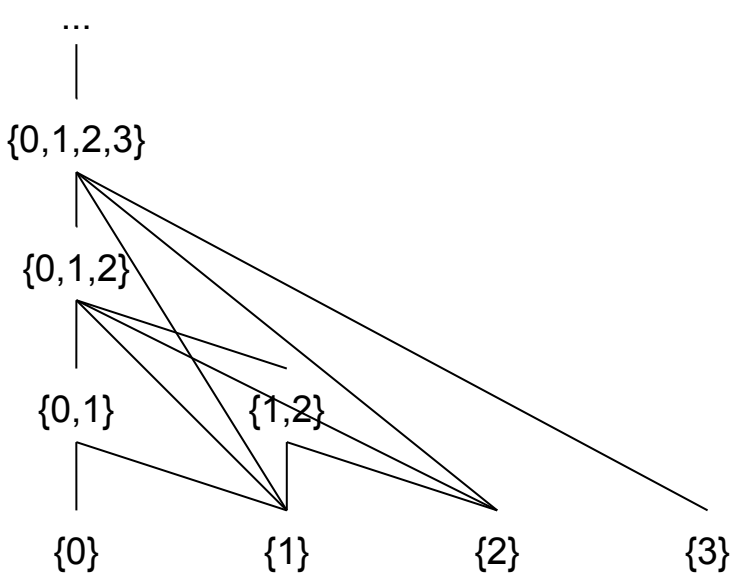


Our lattice

Adding *Partial Order* - $\preccurlyeq$

$A \preccurlyeq B \Leftrightarrow A \subseteq B$





Actually our lattice

The least upper bound, and the problem of infinity

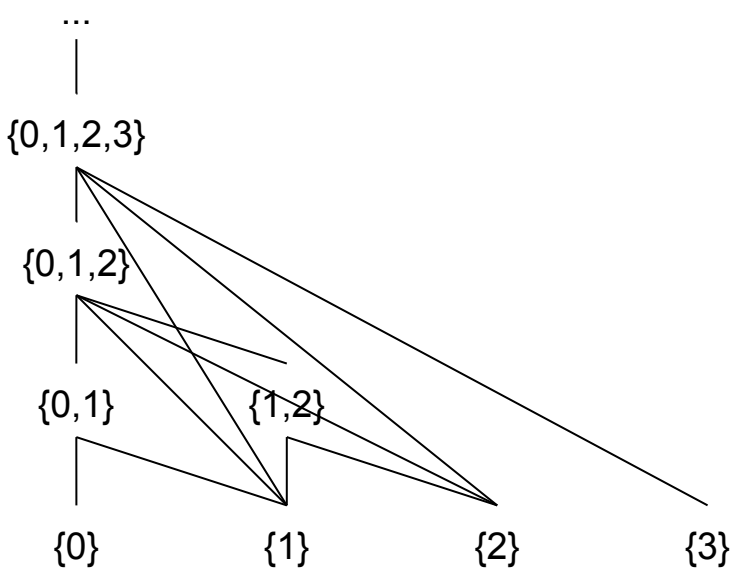
$i_0 = \{0\}$

$i_1 = \{0, 1\}$

$i_2 = \{0, 1, 2\}$

...

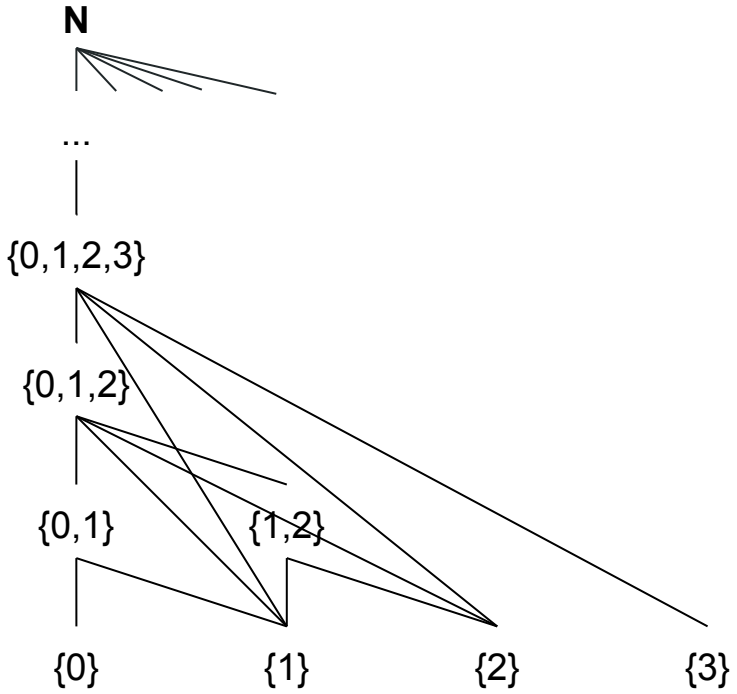
```
var i = 0;
while (true) {
    i++;
}
```



The Widening Operator

Define $\nabla : \mathbb{P}(\mathbf{N}) \times \mathbb{P}(\mathbf{N}) \rightarrow \mathbb{P}(\mathbf{N})$

$$A \nabla B = \begin{cases} \mathbf{N} \text{ if } A \subset B \text{ or } B \subset A \\ A \text{ otherwise} \end{cases}$$



Actually our abstract lattice

Safety vs. Precision

N is not very precise (well, not on the domain of **N**, anyway),
but it is safe and correct.

Extending for Strings and Automata

i_0 = {“yes”}

i_1 = {“yes”, “yes yes”}

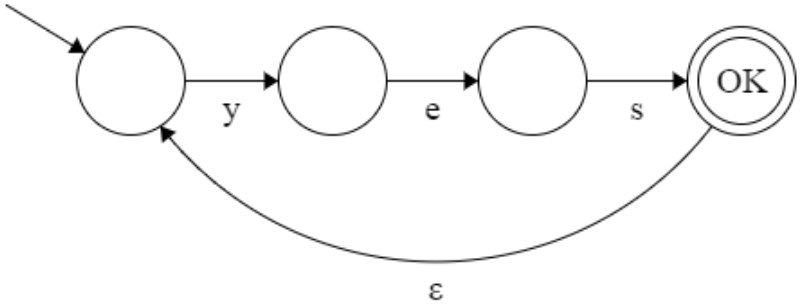
i_2 = {“yes”, “yes yes”, “yes yes yes”}

…

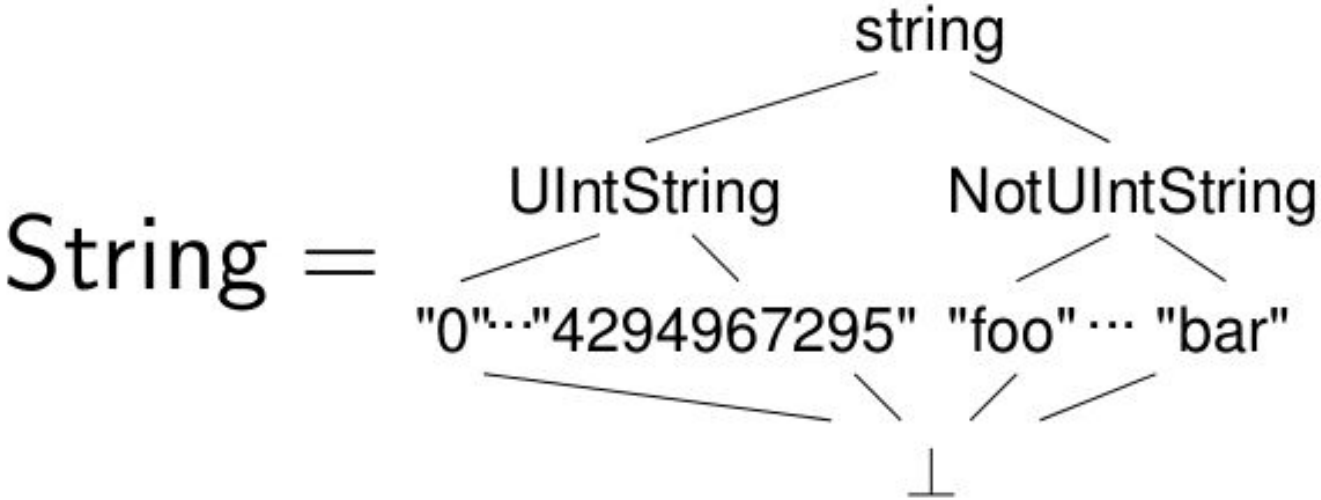
```
var i = "yes\n";
while (true) {
    i += "yes\n";
}
```

We could widen *i* to the set of all strings

Or we could widen it to this: `yes(yes)*`

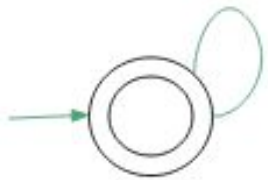


TAJS String Lattice



Strings using automaton in TAJS

- Over 19 functions that represent the possible string value categories.
- Example:
  - `isMaybeAnyStr()`



  - `isMaybeStrUInt()`

$$\text{Str} \cap \boxed{\text{UInt}} \neq \emptyset$$

Cont.

- Examples:
  - isStrIdentifier()

$$\{Str\} \subseteq \mathcal{L}(Identifiers)$$

  - joinAnyStrIdentifier()

$$Str \cup Identifiers$$

Objectives

- Continue studying TAJS source code to find places where modifications are required and start modifying.
- Study how widening operators would be implemented on the tool using FSA.

References

[1]Cousot, Patrick, and Radhia Cousot. "Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints." In Proceedings of the 4th ACM SIGACT-SIGPLAN symposium on Principles of programming languages, pp. 238-252. ACM, 1977.

[2]Jensen, Simon Holm, Anders Møller, and Peter Thiemann. "Type analysis for JavaScript." In Static Analysis, pp. 238-255. Springer Berlin Heidelberg, 2009.

[3] http://www.brics.dk/automaton/