A full implementation of ixml

Giovanni Luigi Amico, Huy Nguyen, David Davitadze, Alex de Beurs and Vasco Roldão

Who are we?

iXML

Our Implementation

https://codeberg.org/06b/casper

Why a new implementation?

- Experimental features
  - Not operator
  - Except operator
  - Modularization
- Performance
- User experience

About Rust

- Statically typed
- Compiled
- Low level
- Memory Management
  - Ownership rules
  - checked at compile time
  - Borrow checker



Ownership

```
String a = “hello”;

var b = a;

System.out.println(a);
```

```
let a = String::from(“hello”);

let b = a;

println!(“{a}”);

// compile error //
```

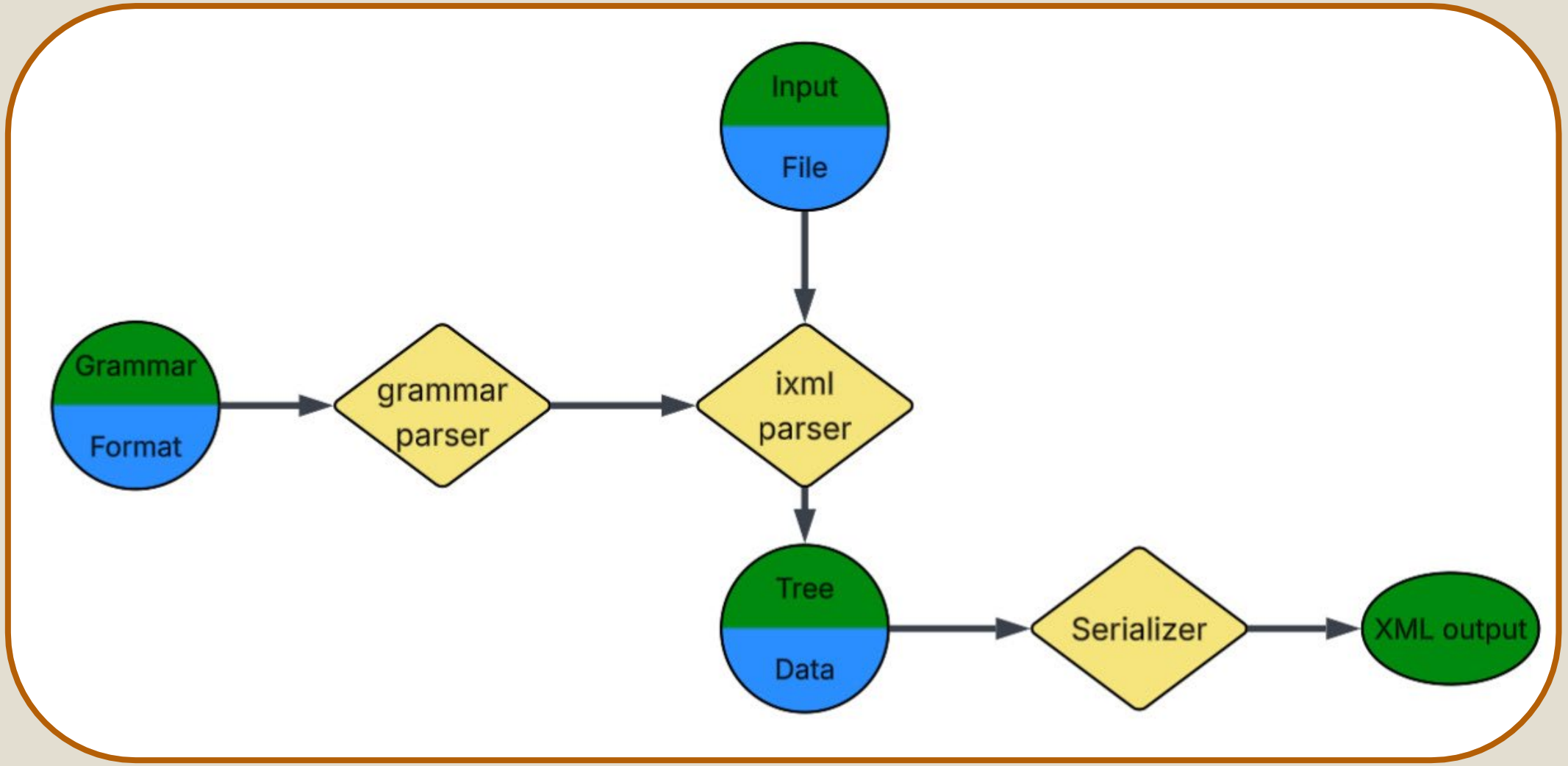


Why Rust?

- Performance
  - Memory
  - Speed
- Memory safety
- Rich standard library
- Tooling
- Error handling



Program Structure



Grammar parser

- Options
  - `Bootstrap`
  - `Handwritten`

- Reasons
  - Error messages
  - Warnings
  - Performance

Generated Parser

- Earley algorithm
- SPPF representation
- BSR parse forest representation

Earley algorithm

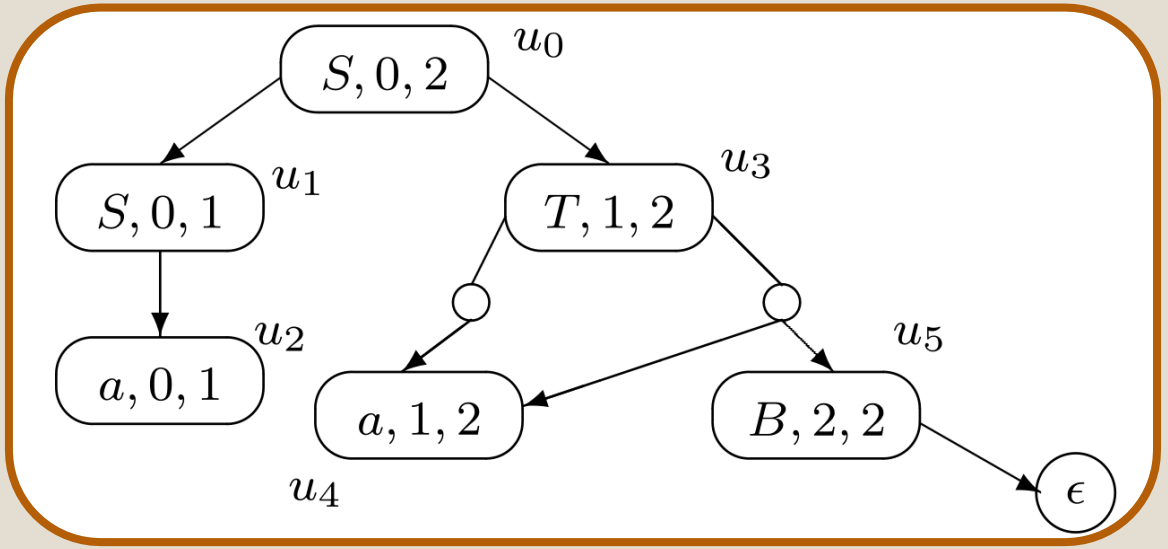
J. Aycock and R. N. Horspool, "Practical Earley Parsing," in The Computer Journal, vol. 45, no. 6, pp. 620-630, Jan. 2002

- Why?
  - Easy implementation
  - Ambiguous grammars
- Consequences
  - Nullable rules
  - SPPF generation

SPPF Generation

Elizabeth Scott,SPPF-Style Parsing From Earley Recognisers,Electronic Notes in Theoretical Computer Science,Volume 203, Issue 2,2008,Pages 53-67,ISSN 1571-0661

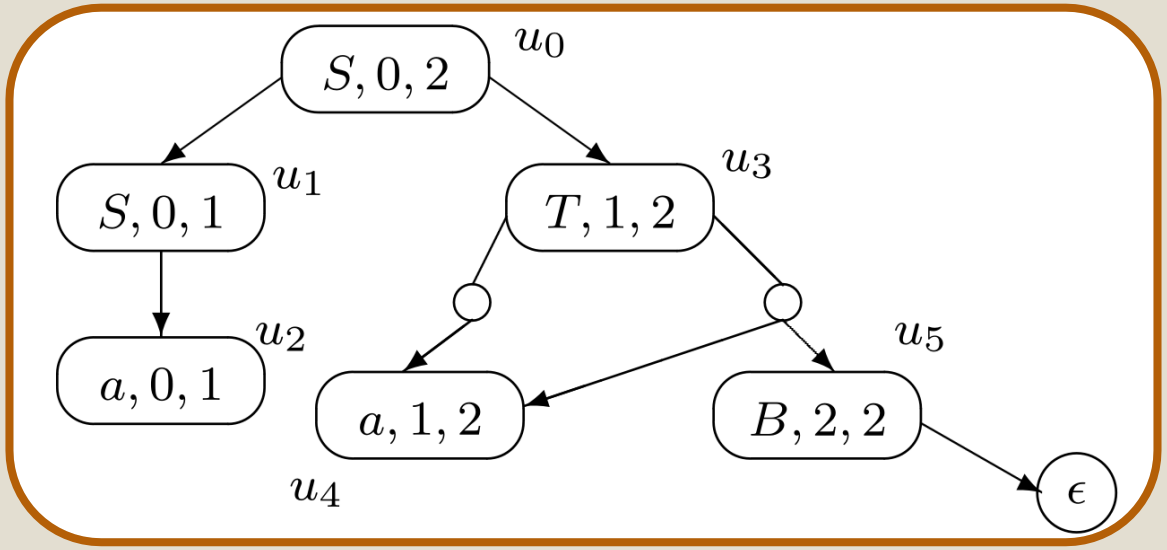
- SPPF Representation
  - Compact representation
  - Complicated Tree Extraction
- Runtime
  - Worst case $O(n^3)$



Binary Subtree Sets

Scott, Elizabeth, Adrian Johnstone, and L. Thomas Van Binsbergen. "Derivation representation using binary subtree sets." Science of Computer Programming 175 (2019): 63-84.

- "SPPF without edges"
- Simpler construction
- Tree extraction



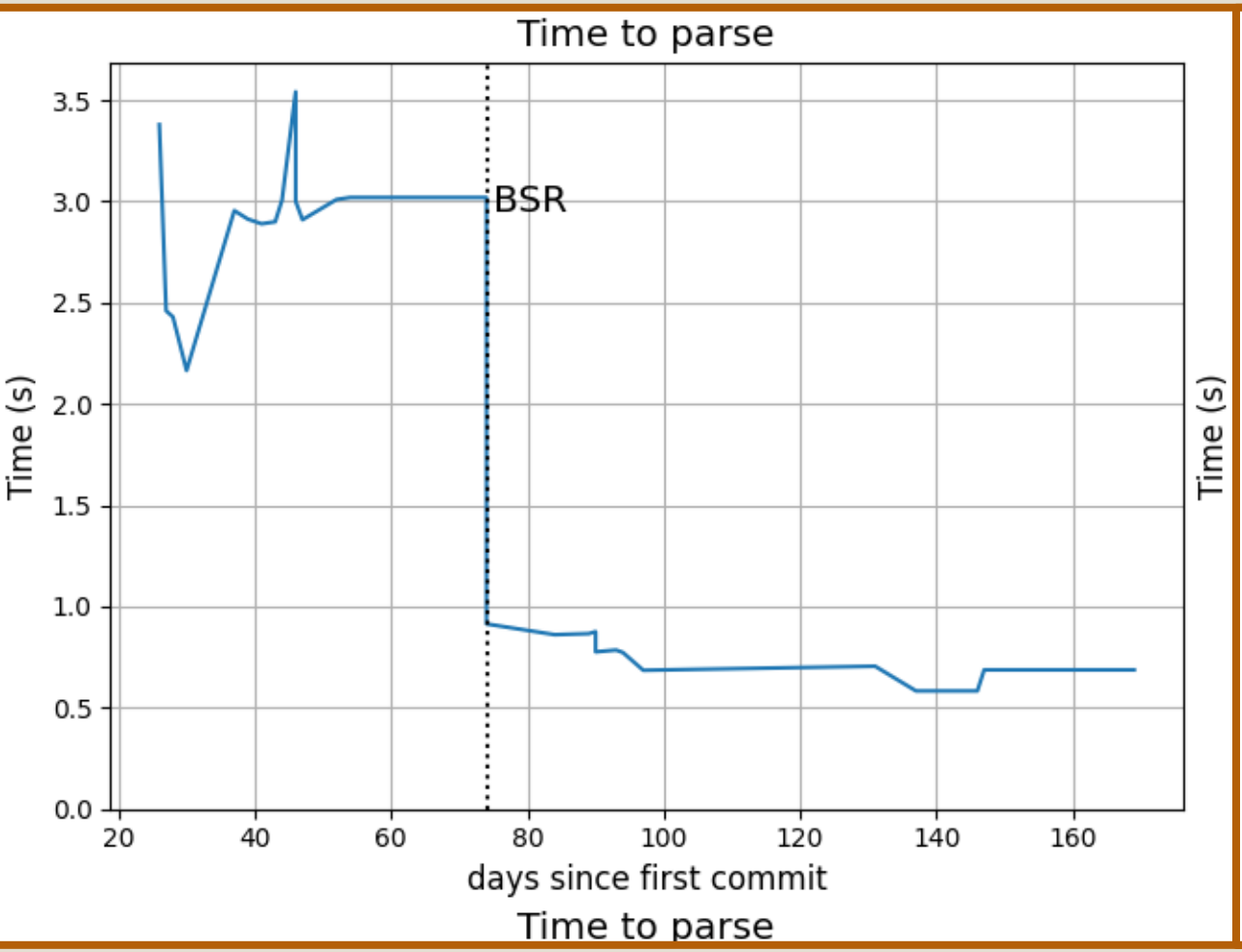
$(S ::= ST\cdot, 0, 1, 2)$

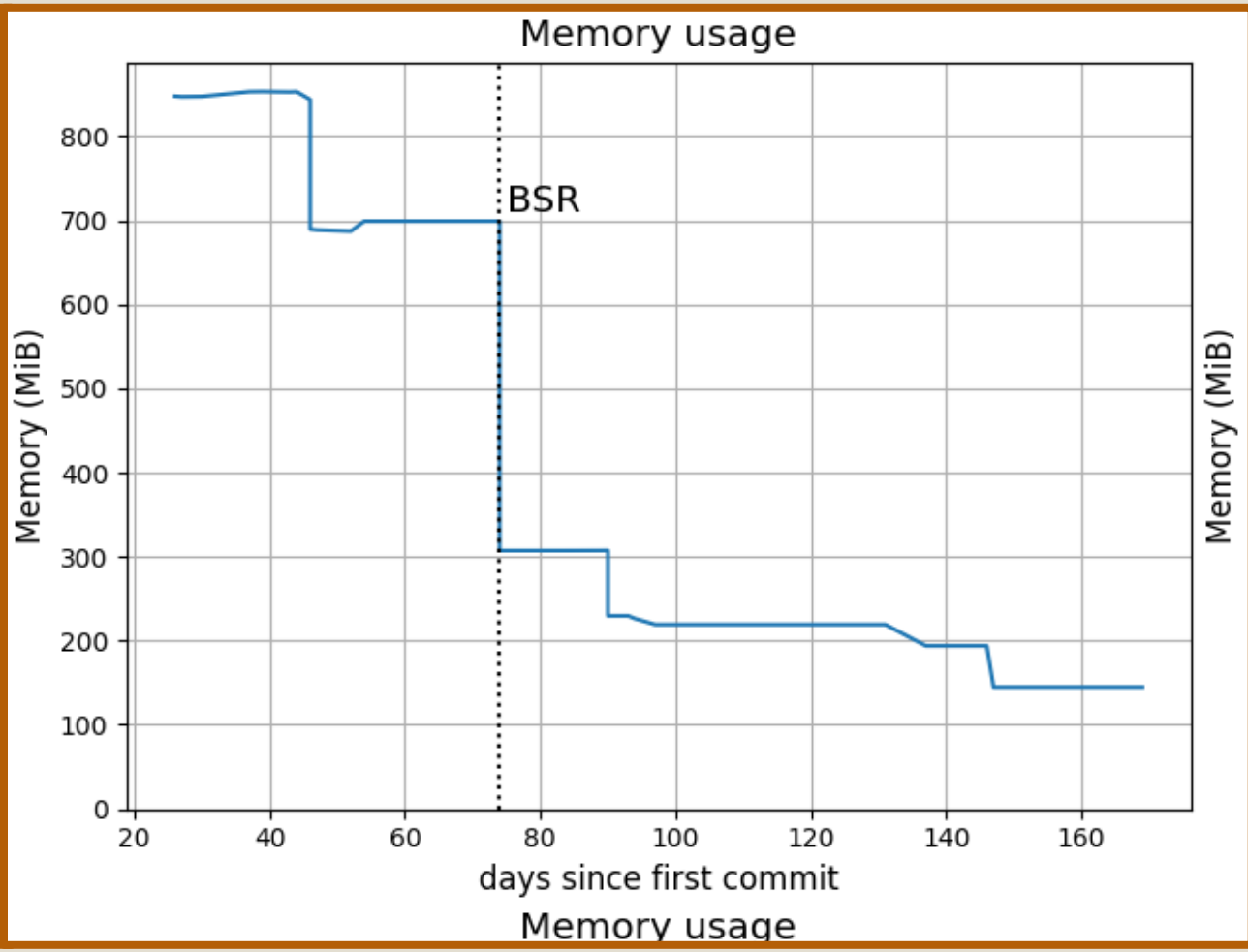
$(S ::= a\cdot, 0, 0, 1)$

$(T ::= aB\cdot, 1, 2, 2)$

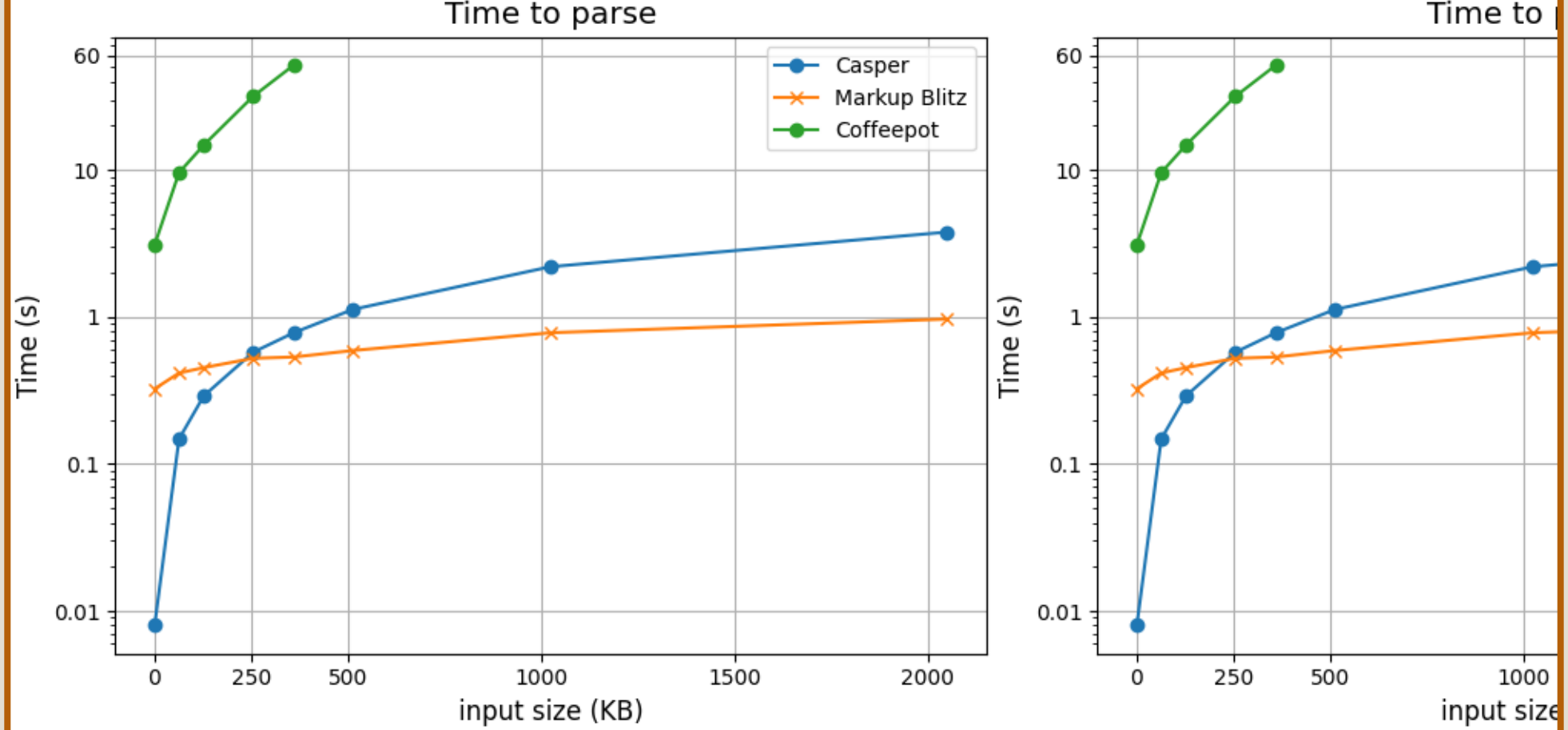
$(B ::= \cdot, 2, 2, 2)$

Performance

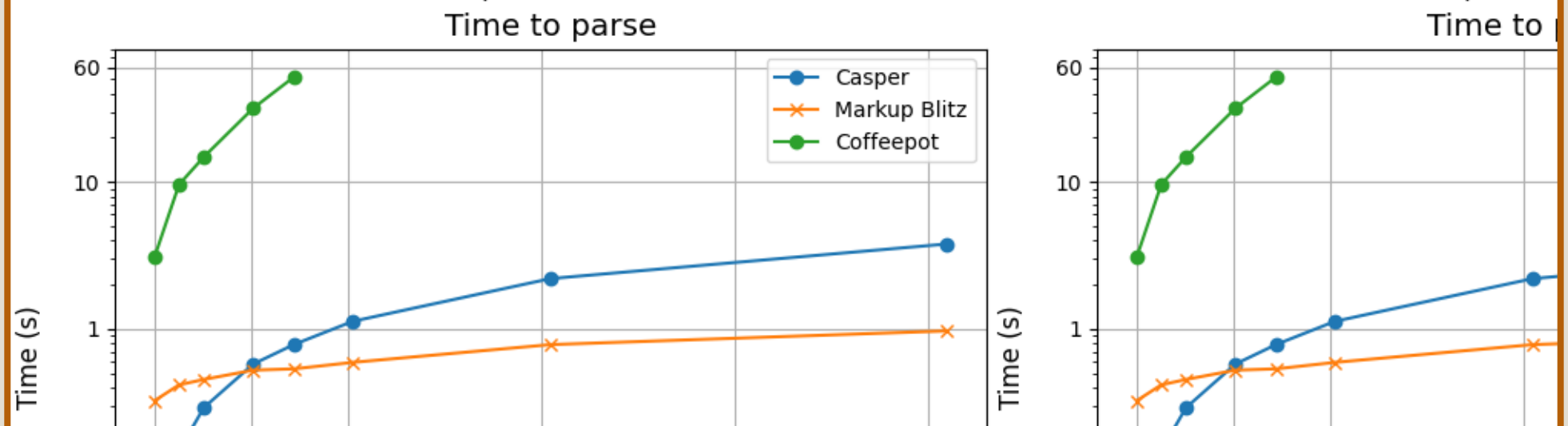




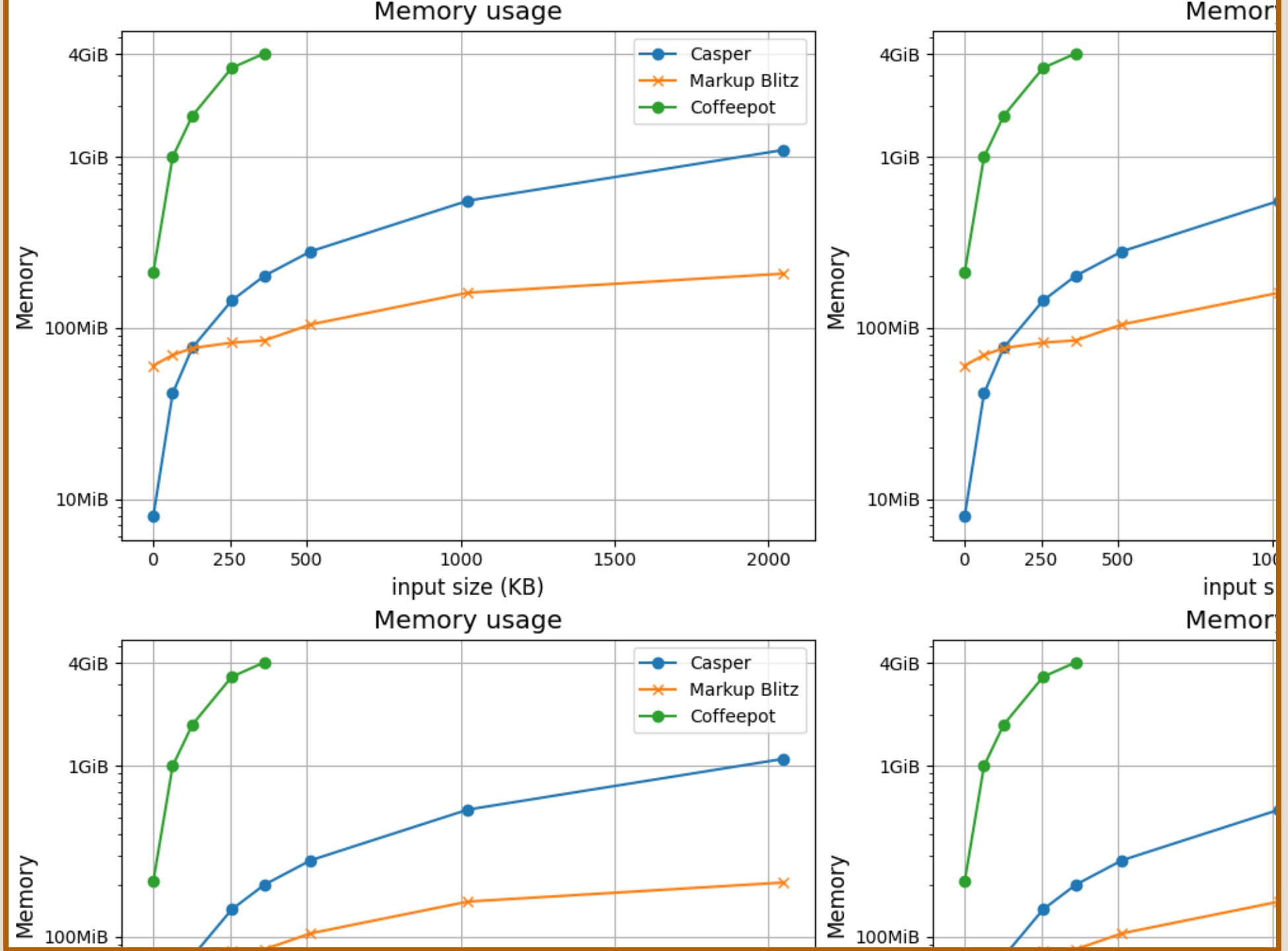
`examples/json.ixml` on 256KB JSON file.


Time to parse
Casper
Markup Blitz
Coffeepot
Time (s)
60
10
1
0.1
0.01
0
250
500
1000
1500
2000
input size (KB)

Time to
Time (s)
60
10
1
0.1
0.01
0
250
500
1000
input size


Time to parse
Casper
Markup Blitz
Coffeepot
Time (s)
60
10
1

Time to
Time (s)
60
10
1


Memory usage
Casper
Markup Blitz
Coffeepot
4GiB
1GiB
100MiB
10MiB
Memory
0
250
500
1000
1500
2000
input size (KB)
Memory usage
Casper
Markup Blitz
Coffeepot
4GiB
1GiB
100MiB
Memory

Features

- Not operator
- Except operator
- Descriptive Feedback
- Modularization
- Web retrieval
- Roundtripping

Not Operator

- A! Succeeds if the following input does not match A.
- If it succeeds, we don’t consume anything.

Not Operator

Input document

```
```some code`
```
```
`more code`
```
```

The relationship is x^2.

Grammar

```
code: '````', body, '````'.

-body: ~[]*.
```

Not Operator

Input document

```
```some code`
```

The relationship is x^2.
```

Grammar

```
code: '````', body, '````'.

-body: ~['`'']*.
```

Not Operator

Input document

```
```
`some code`
```
```

The relationship is x^2.

Grammar

```
fence: -'````'.

-body: (fence!, ~[])*.
```

Except Operator

S: A / B “Everything that matches A but not B”

Difference with not?

```
S: ”username”!, [L]*
```

```
S: [L]* / “username”
```

“usernames”

Descriptive feedback

- Why?
  - Detecting the error
  - Grammar correctness
  - Grammar improvements
  - Testing
- How?
  - Error/warning location
  - Specified expected behavior
  - Different error/warning types

```
grammar.ixml:2:2: A: A* | 'a' | .
                     ^
grammar.ixml:2:2: warning: Rule 'A' has multiple alternatives
that are nullable, making it ambiguous on the empty string.
```

```
oops.ixml:2:5: S: @'a'.
               ^
oops.ixml:2:5: error: '@' is not a valid terminal mark; only '+'
or '-'
```

```
<stdin>:1:7: [2, 3 4]
                   ^
<stdin>:1:7: error: Expected one of ',', ']' or [#20;#a;#d;#9]
but got '4' instead
```

Modularization

The grammar can refer to rules from other grammars

- Why?
  - Allows for splitting complex grammars
  - Allows users to share portions of grammars
  - Allows modular web retrieval

Modularization / Web retrieval

Grammar

+uses math from math.ixml http://example.com/math.ixml

```
mathml: -'$', -math, -'$'.
```

Module

+shares math

```
math: ...
```

Roundtripping

https://homepages.cwi.nl/~steven/Talks/2024/06-07-prague/paper.html





Roundtripping

https://homepages.cwi.nl/~steven/Talks/2024/06-07-prague/paper.html

How?

Modify the grammar

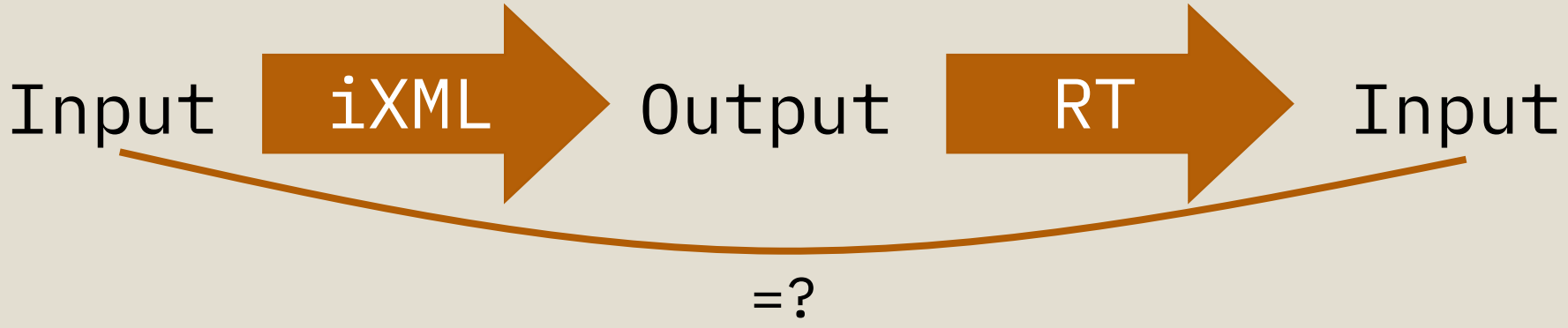
```
S: A.

A: -“b”
```

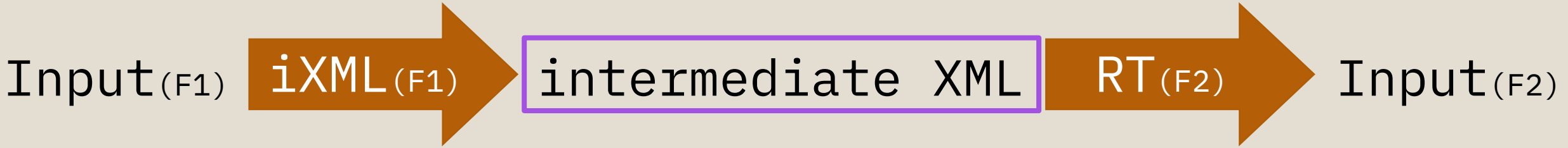
```
S: -“<S>”, A, -“</S>”

A: -“<A>”, +”b”, -“</A>”
```

Lies




```
S: -‘a’*.
```




```
FR: ‘Chat’                IT: ‘Gatto’
  | ‘Chien’.                | ‘Cane’.

<FR>Chat</FR>   !=   <IT>Gatto</IT>
```

```
root: -‘Chat’ +’Cat’     root: -‘Gatto’ +’Cat’
    | -‘Chien’ +’Dog’.         | -‘Cane’ +’Dog’.

                <root>Cat</root>
```

Testing

- Why?
  - Existing specification
- How?
  - Property-based tests
  - Conformance suite

Property-based testing

- Generated grammars
  - SPPF -> BSR parser
  - Roundtripping

Conformance suite

- Provided by the ixml working group
- Custom test runner

iXML conformance report

905 pass 7 fail 141 unsupported

```
schema 1 | tool conformance_report/run.py/0.2 | commit 340c10f34fc46914428dd9abdaa8bb6263f4242d | suite
ca938ecd62970366dc60845a8549cc2f63c18694 | dirty project=True suite=False | generated 2026-10-28T00:30:24+01:00 | platform Darwin
arm64 | elapsed 8477 ms
```

status: any | error kind: any | cause: any

catalog: any | case kind: any | expected: any

1053 visible

- ▶ accepted:expected=none:actual=none 6 cases fail
- ▶ xml_mismatch 1 cases fail
- ▶ unsupported:grammar_test_xml_output 86 cases unsupported
- ▶ unsupported:vxml_grammar 38 cases unsupported
- ▶ unsupported:unicode_tests 17 cases unsupported
- ▶ match 905 cases pass

▼ c-fc0e6d5d3d correct/test-catalog.xml::ixml tests/arith/arith test-case match

grammar

ixml/tests/correct/arith.ixml

```
expr: open, -arith, close.
open: "(".
close: ")".
arith: left, -op, right.
op: @plus.
plus: "+".
left: "a".
right:"b".
```

input

ixml/tests/correct/arith.inp

```
(a+b)
```

expected 1

xml ixml/tests/correct/arith.output.xml

```
<expr plus="+">
  <open>(</open>
  <left>a</left>
  <right>b</right>
  <close>)</close>
</expr>
```

actual stdout

returncode 0, 13 ms

```
<expr plus="+">
  <open>(</open>
  <left>a</left>
  <right>b</right>
  <close>)</close>
</expr>
```

repro

```
target/release/ixml-cli -g ixml/tests/correct/arith.ixml -i ixml/tests/correct/arith.inp
```

▼ c-fc0e6d5d3d correct/test-catalog.xml::ixml tests/arith/arith test-case match

grammar

ixml/tests/correct/arith.ixml

```
expr: open, -arith, close.
open: "(".
close: ")".
arith: left, -op, right.
op: @plus.
plus: "+".
left: "a".
right:"b".
```

input

ixml/tests/correct/arith.inp

```
(a+b)
```

expected 1

xml ixml/tests/correct/arith.output.xml

```
<expr plus="+">
  <open>(</open>
  <left>a</left>
  <right>b</right>
  <close>)</close>
</expr>
```

actual stdout

returncode 0, 13 ms

```
<expr plus="+">
  <open>(</open>
  <left>a</left>
  <right>b</right>
  <close>)</close>
</expr>
```

repro

```
target/release/ixml-cli -g ixml/tests/correct/arith.ixml -i ixml/tests/correct/arith.inp
```

Questions?

-giovaluigi1@gmail.com
-V.M.SantosRoldao@student.tudelft.nl
-A.Beurs@student.tudelft.nl