XML Typelift

Michał J. Gajda Dmitry I. Krylov
https://www.migamake.com

2021-02-19

Problem

XML ecosystem: parsers

XML ecosystem: formats

Format market

Format market

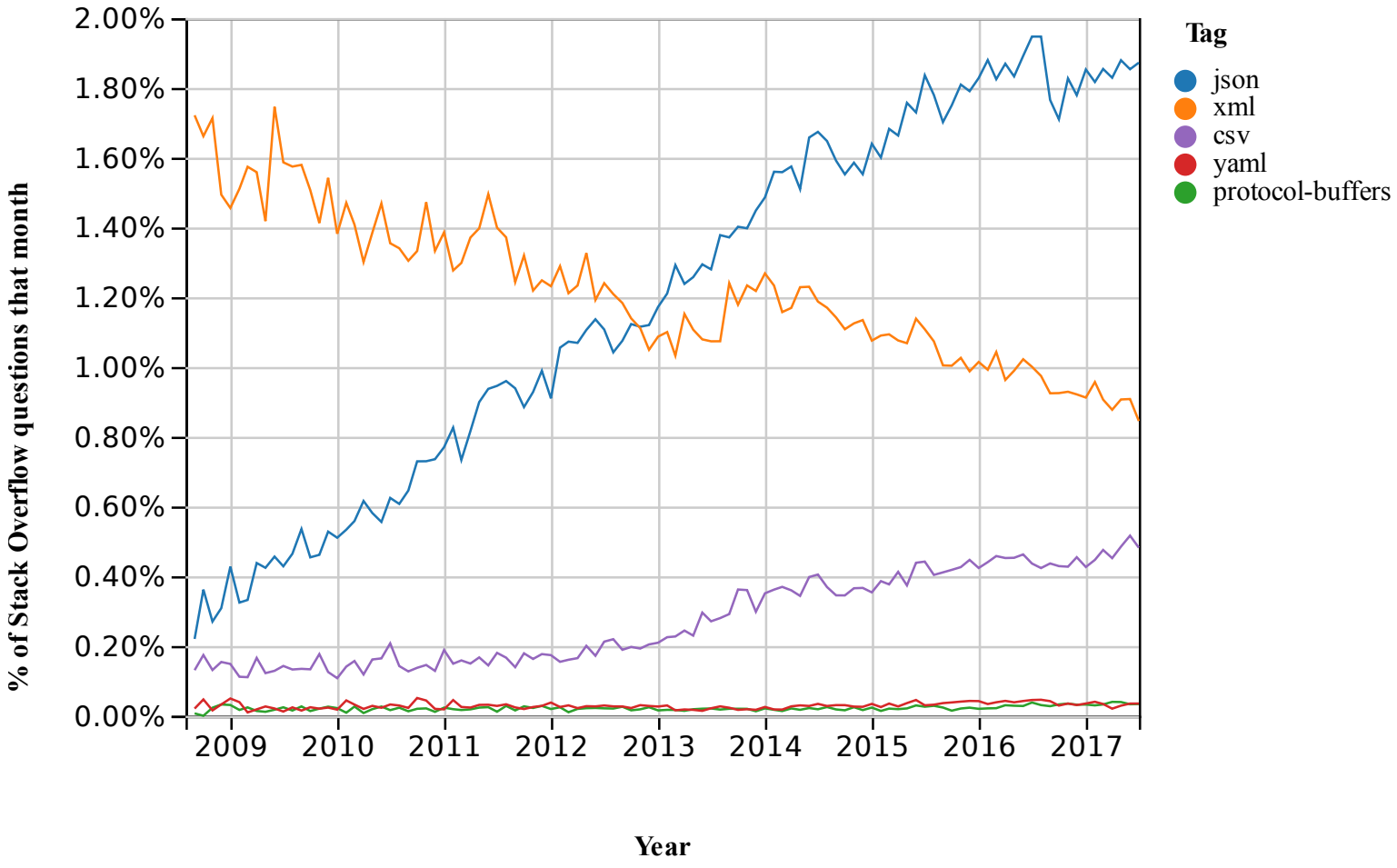


Figure 1: Data standards

XML example

XML example

```
<?xml version="1.0" encoding="utf-8"?>
<users> <user>
            <uid>123</uid>
            <name>John</name>
            <bday>1990-11-12</bday>
        </user>
</users>
```

XML Schema

XML Schema

```
<xs:schema
  xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name='users'>
    <xs:complexType>
      <xs:sequence>
        <xs:element name="user" type="UserType"
                                minOccurs="0"
                                maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:complexType name="UserType" mixed="false">
    <xs:sequence>
      <xs:element name="uid"  type="xs:int"/>
      <xs:element name="name" type="xs:string"/>
      <xs:element name="bday" type="xs:date"
                              minOccurs="0"/>
```

XML Schema 2

XML Schema 2

Tree of elements, attributes, and text nodes.

- `<xs:element minOccurs=9 maxOccurs=5>`
- `<xs:attribute>`
- `<xs:any>`

XML: composition

XML: composition

Composition of elements:

- `<xs:sequence>`
- `<xs:choice>`
- `<xs:sequence>`
- `<xs:all>`
- `mixed="true"`

XML: flat vs tree fragment types

XML: flat vs tree fragment types

- `<simpleContent>` – flat types
- `<complexContent>` – tree fragment types

Multiple character encodings

Multiple character encodings

- ASCII
- ASCII Latin-2
- UTF-8
- UTF-16
- UTF-32 # Solution

Vectorized string class

Vectorized string class

```
class VectorizedString where
  s index :: str -> Int -> Char
  -- | Find the first occurence
  elemIndexFrom :: Char -> str -> Int -> Maybe Int
  -- | Extract substring between offsets
  substring :: str -> Int -> Int -> str
  -- | Move cursor forward
  drop :: Int -> str -> str
  -- | Is the output exhausted
  null :: str -> Bool
```

Parsing DOM with `xeno`

Parsing DOM with xeno

```haskell
import Xeno.DOM
import qualified Data.ByteString as BS

processFile filename = do
    Right result <- BS.readFile filename
                    >>= Xeno.parse.inp
    print $ filter isName $ allNodes result
  where
    isName node =
      BS.toLower (Xeno.name n) == "name"
```

XML Typelift as type provider

XML Typelift as type provider

```
data Users = Users { users: [User] }

data User = User { uid :: Int
                 , name :: String
                 , bday :: Maybe Date }

parse :: ByteString -> Either String Users
```

XML Typelift usage

XML Typelift usage

```
import UsersSchema(parse, Users(..))

processFile filename = do
  Right users <- BS.readFile filename >>= parse
  print $ map name users
```

XML Typelift benchmarks: speed

XML Typelift benchmarks: speed

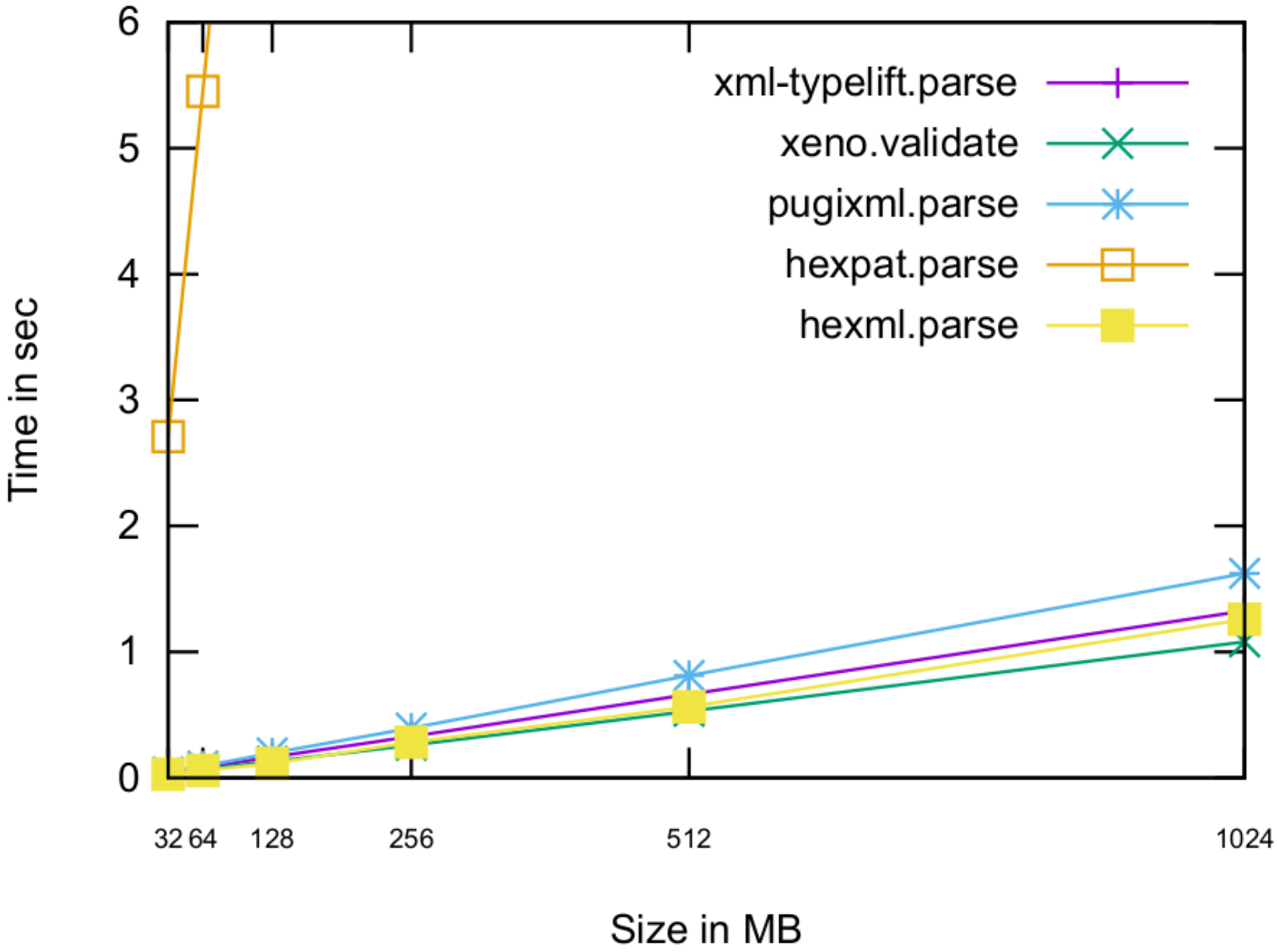


Figure 2: Benchmark against other tools

XML Typelift benchmarks: memory

XML Typelift benchmarks: memory

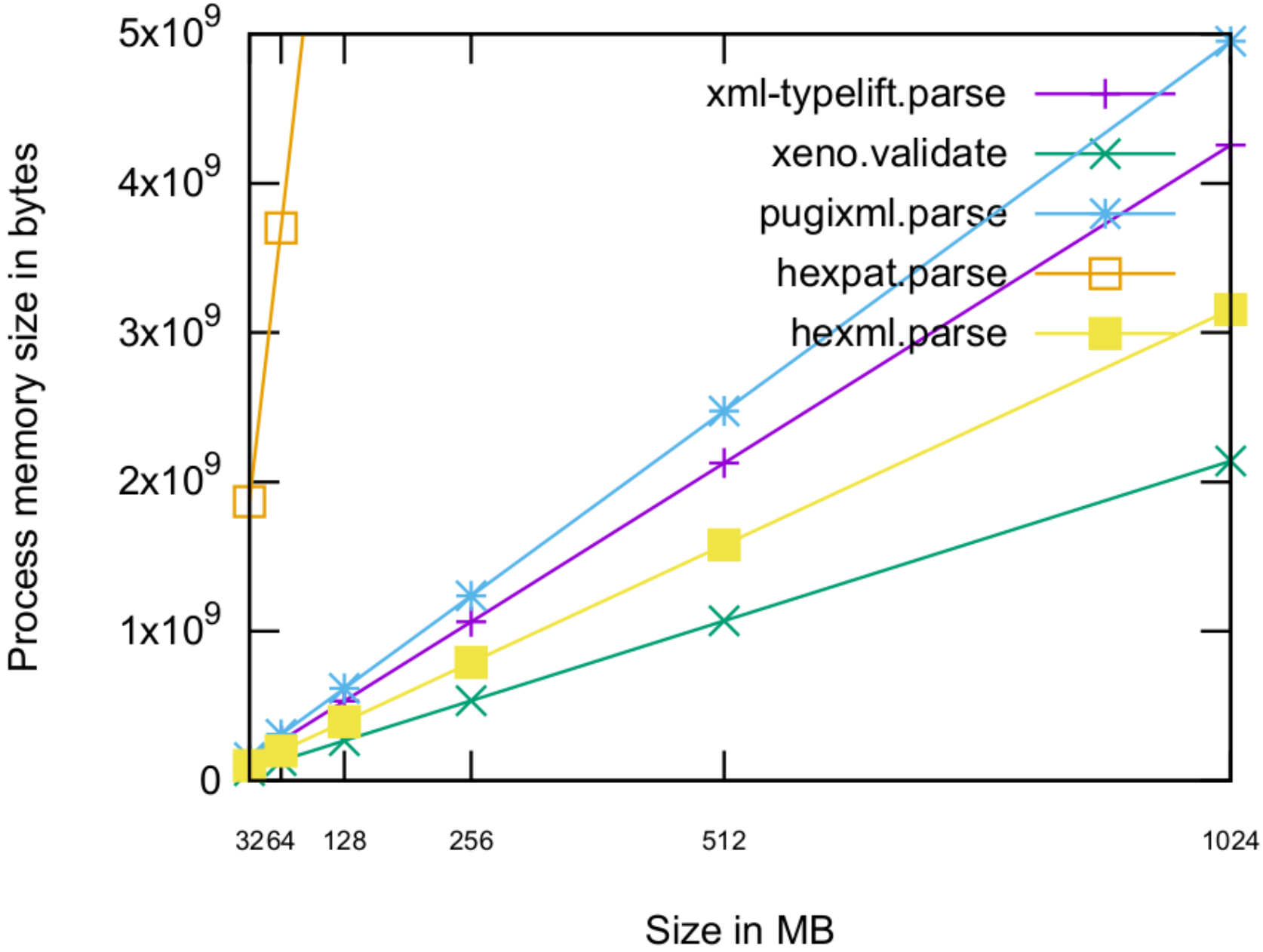


Figure 3: Benchmark against other tools

Parser 1

Parser 1

Parse to DOM with `xeno`, then make a data structure.

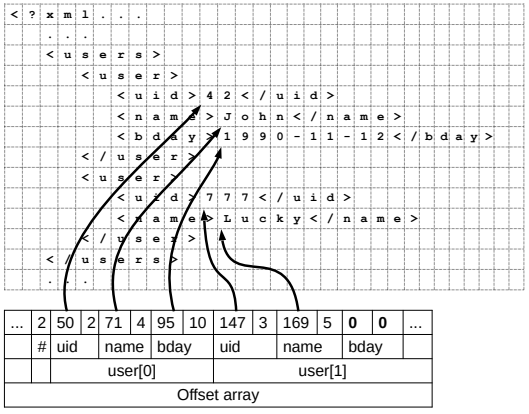
Parser 2: will SAX help?

Parser 2: will SAX help?

```
data SAXEvent = OpenElt  XMLString
              | CloseElt XMLString
              | Attr     XMLString XMLString
              | ...
newtype SAXStream = SAXStream [SAXEvent]
```

Parser 3: offset array

Parser 3: offset array



Parser 4: clever string comparison

Parser 4: clever string comparison

```
parseUser userStart =
  if    s_index bs  userStart       == '<'
     && s_index bs (userStart + 1) == 'u'
     && s_index bs (userStart + 2) == 's'
     && s_index bs (userStart + 3) == 'e'
     && s_index bs (userStart + 4) == 'r'
     && s_index bs (userStart + 5) == '>'
```

Parser 6

Parser 6

Uses SAX and then:

```
parseUser :: SaxParser User
parseUser = withTag "user"
          $ \_ -> User <$> tagValue "uid"
                       <*> tagValue "name"
                       <*> tagValue "bday"

-- Collection of useful combinators
withTag :: ByteString -> SaxParser a -> SaxParser a
withTag tagName sp =
  openTag tagName *> sp <* closeTag

tagValue :: (F.MonadFail                    m
            ,MonadParsec e [SAXEvent] m)
         => ByteString -> m ByteString
tagValue tagName = do
    void $ satisfy (==OpenElt tagName)
    -- ... auxiliary code skipped ...
```

Parser 7

Parser 7

Offset array **and** inlined string comparisons.

Experiment summary

Final code

Final code

```
parseUsersContent arrStart strStart = do
  (arrOfs1, strOfs1) <-
    inManyTags "user" arrStart strStart
               $ parseUserTypeContent
  return (arrOfs1, strOfs1)
parseUserTypeContent arrStart strStart = do
  (arrOfs1, strOfs1) <-
    inOneTag "uid"  strStart $ parseInt arrStart
  (arrOfs2, strOfs2) <-
    inOneTag "name" strOfs1 $ parseString arrOfs1
  (arrOfs3, strOfs3) <-
    inMaybeTag "bday" arrOfs2 strOfs2 parseDay
  return (arrOfs3, strOfs3)
```

Conclusion

Conclusion

- XML typelift is available
- Type providers can generate faster code
- No need to work extra on XML parsers
- Come and add features to support full XML!

References