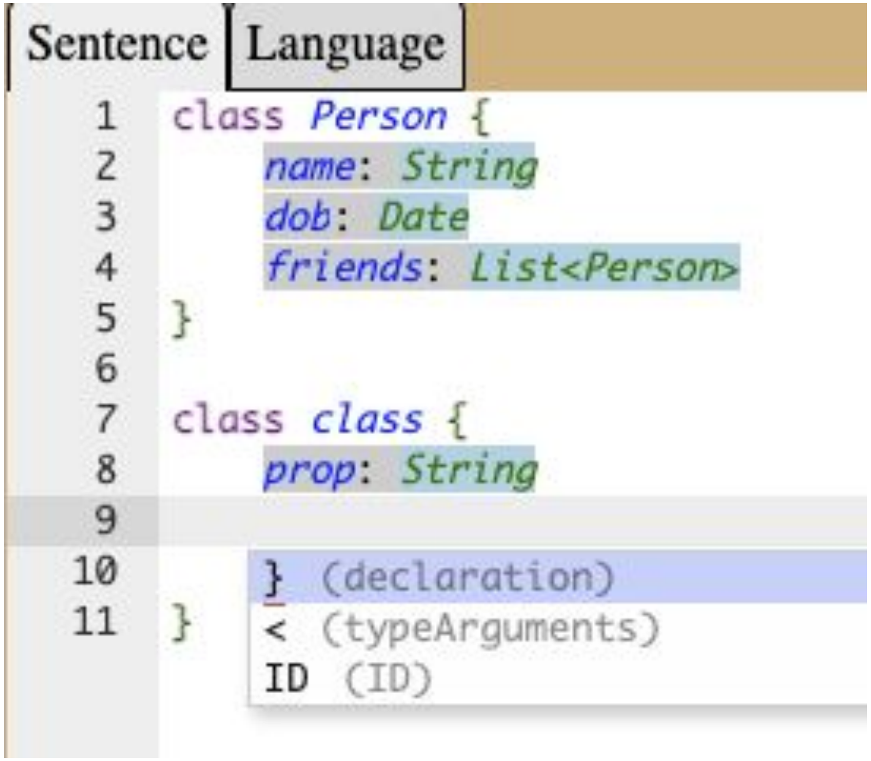
AGL: When a Regular Expression is not enough

Dr. David H. Akehurst



```
class Person {
    name: String
    dob: Date
    friends: List<Person>
}

class class {
    prop: String


}
```

AGL: When a Regular Expression is not enough
Overview

- Executive Summary
- Motivation
- What already exists?
- What is AGL?
- API
- Performance
- Problems
- Conclusion
- Demo
- Questions / Discussion

AGL: When a Regular Expression is not enough
Executive Summary

- Maturity
  - There are bugs, issues, and things-in-progress
  - I am currently the only user I know of
  - It is / has been used in commercial projects
- AGL is a runtime parser generator
  - Parser is generated at runtime no code generation
- Scan-on-demand - No need to worry about reserved words
  - There is no pre-parse scan step
  - Tokens are scanned for during parse time when they are needed
- GLR-based (with variations/extensions) - No rule restrictions
  - No need to worry about left-recursive or right-recursive rules or hidden-left recursion, etc
  - Ambiguity is permitted, but will slow down the parse.
- Support for grammar composition
  - via extension/inheritance
  - via embedding one grammar in another
- Implemented using kotlin multiplatform
  - Executes on (and usable with) JVM, JavaScript, (Web-assembly, and native code)
- Integration with Ace and Monaco Javascript Editors

AGL: When a Regular Expression is not enough
If you would rather read and play than listen

- Source Code
    - https://github.com/dhakehurst/net.akehurst.language
- Online Demo (older version)
    - https://info.itemis.com/demo/agl/editor
- Article
    - https://medium.com/@dr.david.h.akehurst/agl-your-dsl-in-the-web-c9f54595691b
- Documentation
    - https://medium.com/@dr.david.h.akehurst/a-kotlin-multi-platform-parser-usable-from-a-jvm-or-javascript-59e870832a79

AGL: When a Regular Expression is not enough
Motivation: History

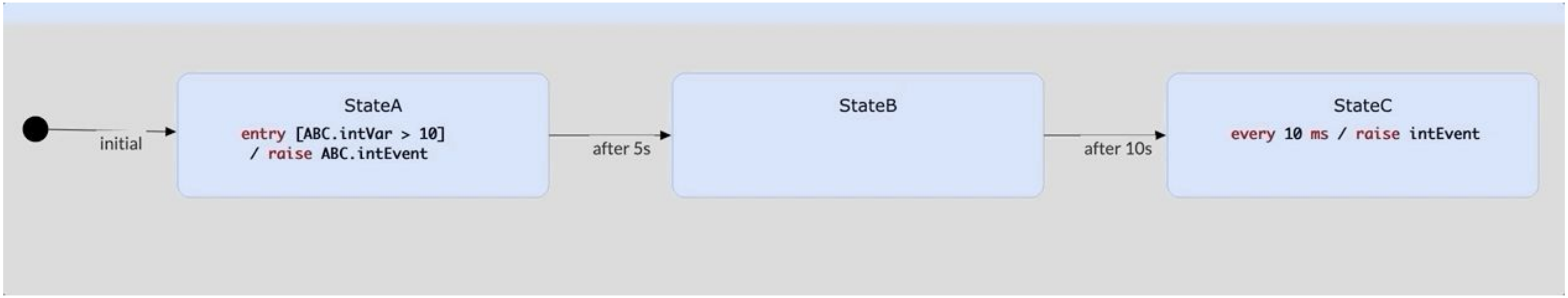
- Started pre **2007**(ish) consequence of modularising OCL (published 2008)
  - Implemented something, tried to publish, rejected because basically I knew nothing about parser algorithms.
- ANTLR v4 came out soon after, I thought that would be the solution as it implemented similar ideas (grammar inheritance).
  - Unfortunately not. (No hidden left recursion, grammar inheritance is insufficient, up-front code-generation step)
- Christmas **2014** I was bored, tried again.
  - Learnt lots more about parser **theory**.
  - Simple self motivation to solve/complete something I once started
- Research project at **itemis** required use of web-based DSL.
  - Switched to Kotlin.
- Many holidays, weekends, evenings later…….**AGL**

AGL: When a Regular Expression is not enough
Motivation: Use cases for AGL

- Text language embedded in a Graphical language
  - I need more than a Regular Expression
  - But I don’t want to “generate” a parser up front

demo for web prototype of

YAKINDU STATECHART TOOLS



```
entry [ABC.intVar > 10]
/ raise ABC.intEvent
```

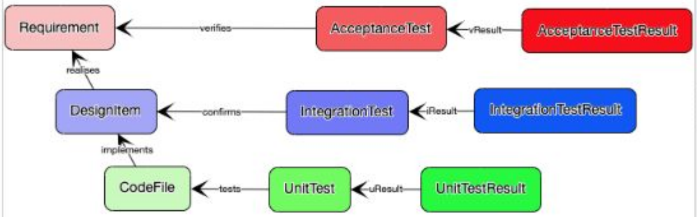
```
every 10 ms / raise intEvent
```

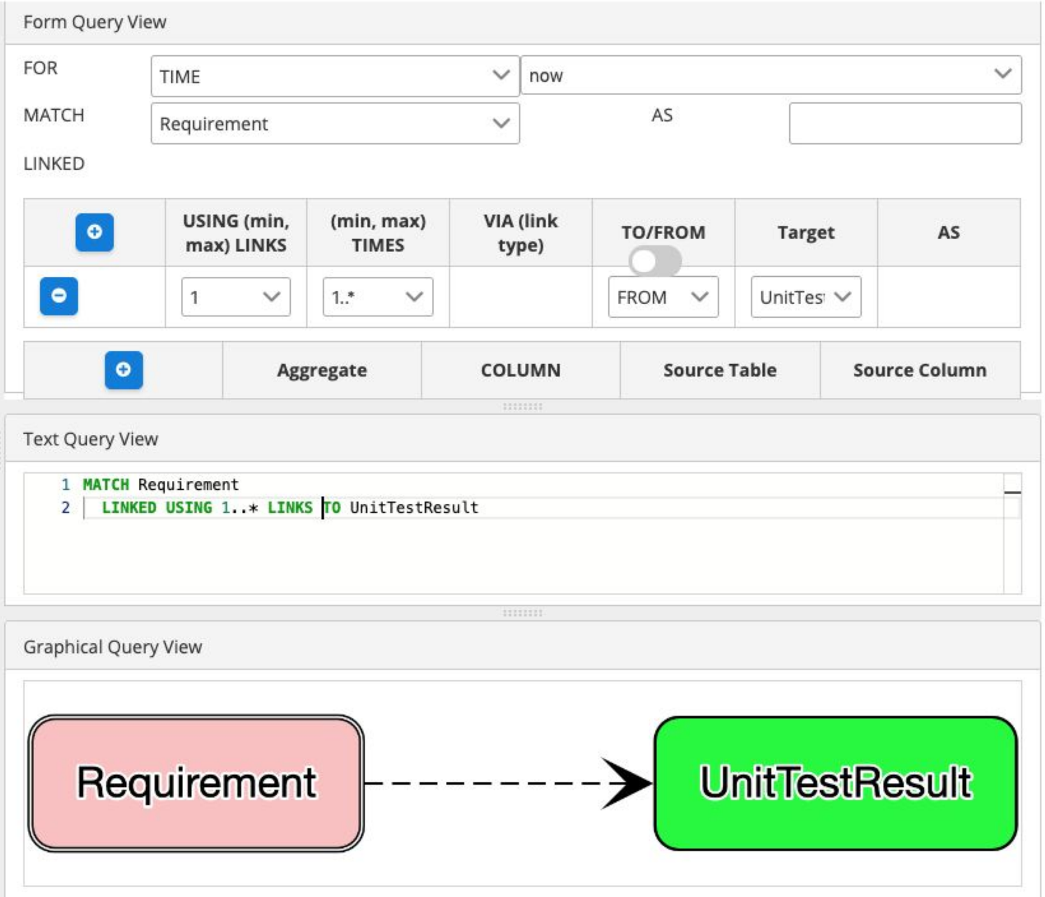
AGL: When a Regular Expression is not enough
Motivation: Use cases for AGL

- I have short sentences (not 1000s of lines and multiple files)
    - maybe queries in a web application

research into query languages related to:

YAKINDU TRACEABILITY





Form Query View

FOR	TIME	now	
MATCH	Requirement	AS	
LINKED			

⊕	USING (min, max) LINKS	(min, max) TIMES	VIA (link type)	TO/FROM	Target	AS
⊖	1	1..*		FROM	UnitTes	

⊕	Aggregate	COLUMN	Source Table	Source Column

Text Query View

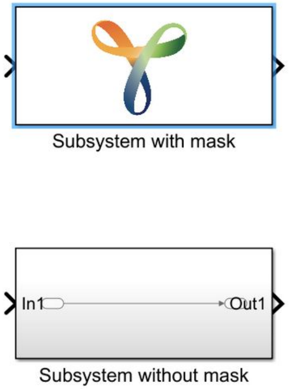
```
MATCH Requirement
  LINKED USING 1..* LINKS TO UnitTestResult
```

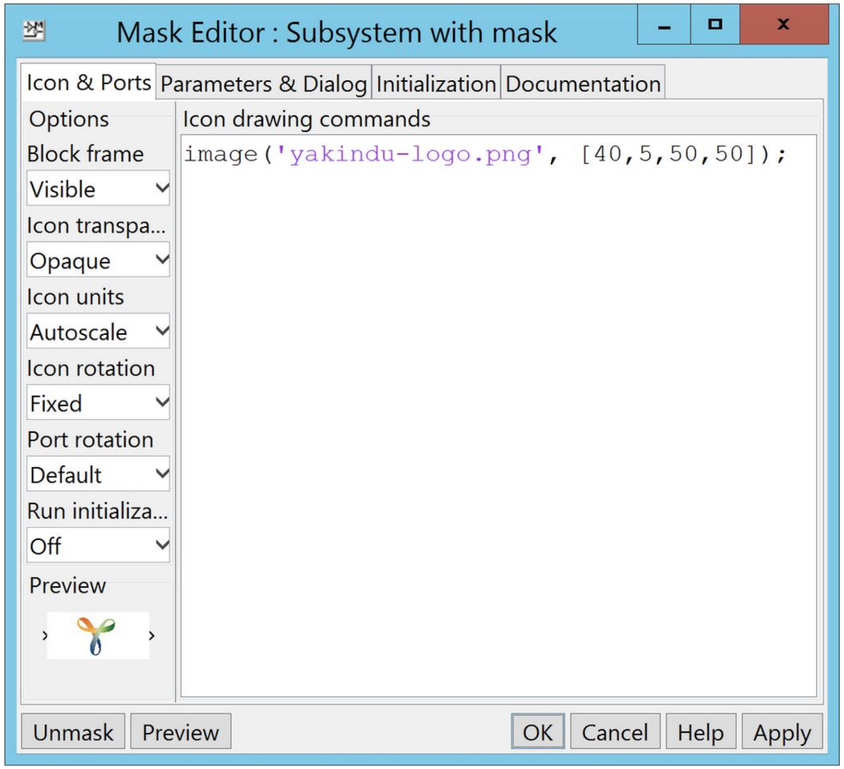
Graphical Query View

AGL: When a Regular Expression is not enough
Motivation: Use cases for AGL

- Parsing Matlab Script to create graphical icons

YAKINDU MODEL VIEWER





Options

Block frame: Visible

Icon transpa...: Opaque

Icon units: Autoscale

Icon rotation: Fixed

Port rotation: Default

Run initializa...: Off

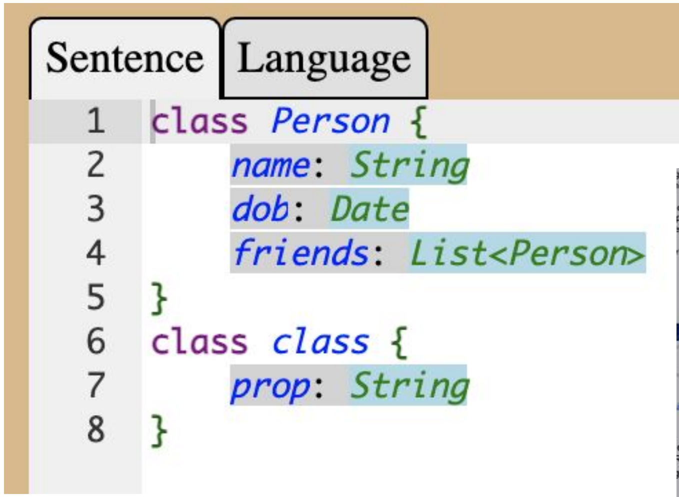
Preview

Icon drawing commands

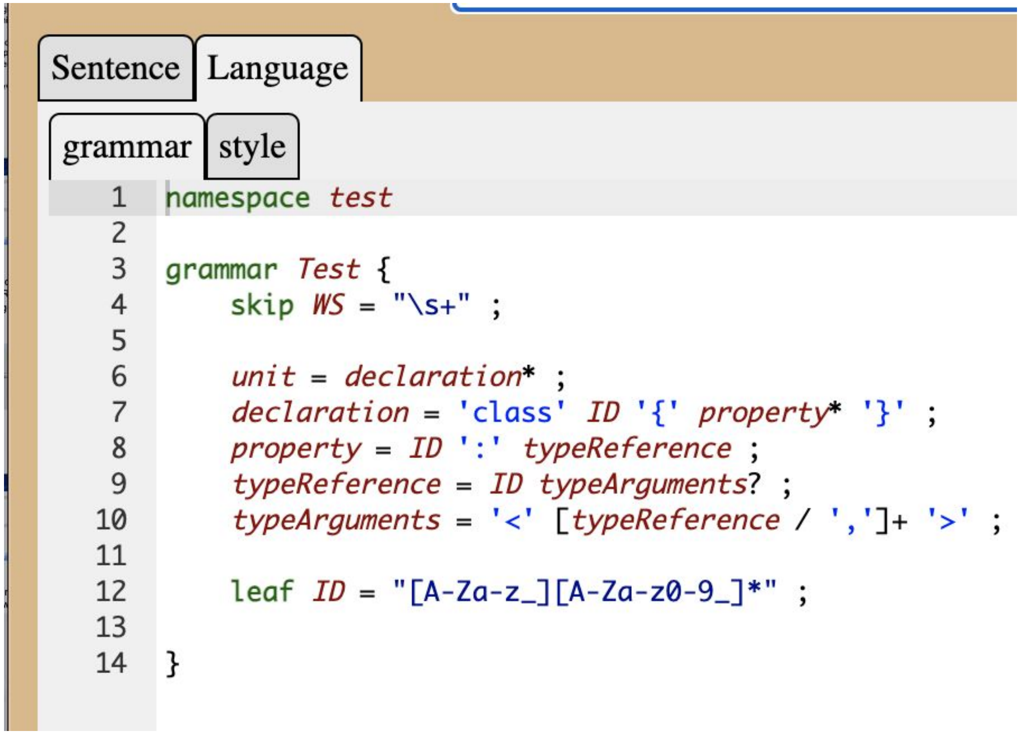
```
image('yakindu-logo.png', [40,5,50,50]);
```

AGL: When a Regular Expression is not enough
Motivation: Use cases for AGL

- I want to change my language definition at runtime



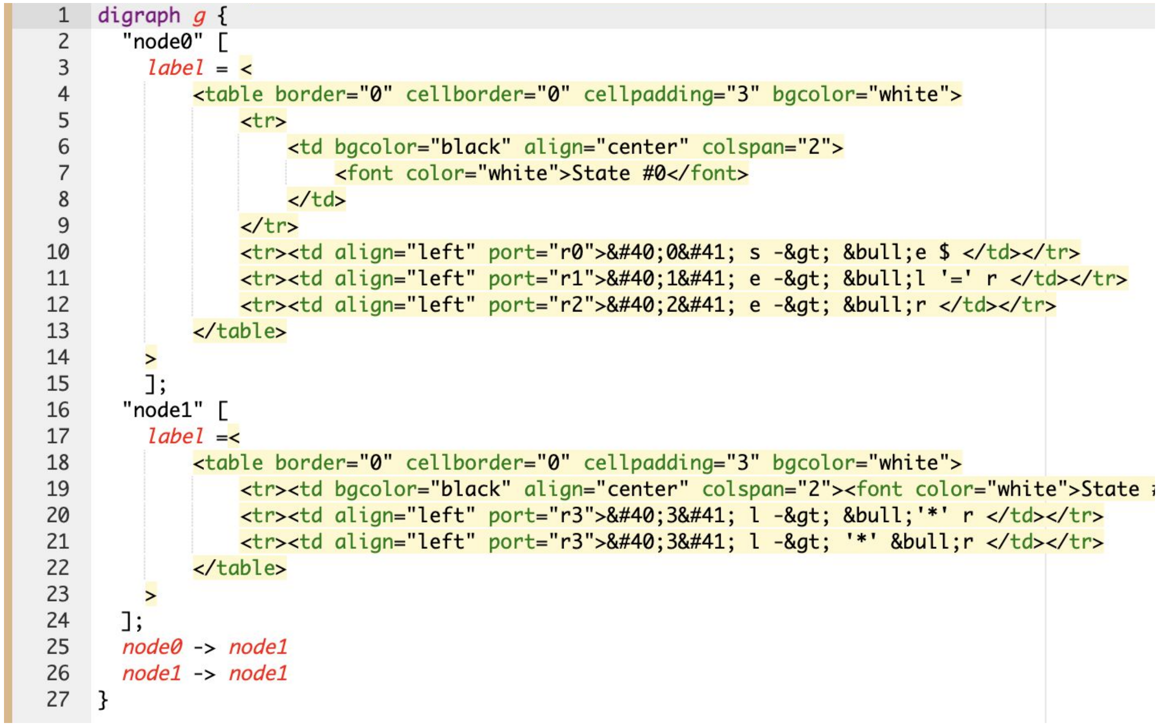
```
class Person {
      name: String
      dob: Date
      friends: List<Person>
}
class class {
      prop: String
}
```



```
namespace test

grammar Test {
    skip WS = "\s+" ;

    unit = declaration* ;
    declaration = 'class' ID '{' property* '}' ;
    property = ID ':' typeReference ;
    typeReference = ID typeArguments? ;
    typeArguments = '<' [typeReference / ',']+ '>' ;

    leaf ID = "[A-Za-z_][A-Za-z0-9_]*" ;

}
```

AGL: When a Regular Expression is not enough
Motivation: Use cases for AGL

- I want families of languages
  - Text in UML diagrams
  - Modularising OCL
  - Graphviz / DOT - XML embedded in graph description



```
digraph g {
  "node0" [
    label = <
        <table border="0" cellborder="0" cellpadding="3" bgcolor="white">
            <tr>
                <td bgcolor="black" align="center" colspan="2">
                    <font color="white">State #0</font>
                </td>
            </tr>
            <tr><td align="left" port="r0">&#40;0&#41; s -&gt; &bull;e $ </td></tr>
            <tr><td align="left" port="r1">&#40;1&#41; e -&gt; &bull;l '=' r </td></tr>
            <tr><td align="left" port="r2">&#40;2&#41; e -&gt; &bull;r </td></tr>
        </table>
    >
    ];
  "node1" [
    label =<
        <table border="0" cellborder="0" cellpadding="3" bgcolor="white">
            <tr><td bgcolor="black" align="center" colspan="2"><font color="white">State
            <tr><td align="left" port="r3">&#40;3&#41; l -&gt; &bull;'*' r </td></tr>
            <tr><td align="left" port="r3">&#40;3&#41; l -&gt; '*' &bull;r </td></tr>
        </table>
    >
  ];
  node0 -> node1
  node1 -> node1
}
```

AGL: When a Regular Expression is not enough

Motivation: The requirements I set myself

1. **Runtime build**: The parser should be built at runtime. I.e. no separate generate-code step or s generating the parser.
2. **No rule l** ng the grammar rules should b possible. i.e. it should b e any pattern of grammar rules to worry about limitations regarding le right, or hidden recursion. I.e. th arser should handle any valid EBNF-like grammar.
3. **No reserved words**: No limitation regarding reserved words. I.e. a grammar can be defined where key-words can be used as variable names.
4. **Lists of items**: The grammar language should have support for parsing lists of items that are represented as lists in the resulting parse tree.
5. **Grammar composition**: The parser should support families of languages. I.e. it should be possible to compose different grammars to form a new grammar (other than by copy and paste).
6. **Any goal rule**: The pars should support parsing a sente the given rules of the ny rule can be used as the 'goal' ru
7. **Multi-platf** should be executable on, as a mi irtual Machine (J Script platform. Ideally on other platforms also.
8. **Performant**: The parser must be performant enough to be usable. I.e. parsing a page of text with an unambiguous grammar should take under 1 second.

1-4: Ease of Use

5-6: Language Families

7: JS & JVM
Kotlin is Awesome

8: Low bar

AGL: When a Regular Expression is not enough
What already exists?: Existing Parsers

- Long list on Wikipedia - many with no available implementation

- Parser Combinators
  - Built at runtime
  - Typically LL and other restrictions on grammars

- ANTLR, Yacc/Lex, etc
  - All require pre-compile time code generation

- JSGLR, LaJa, and many others
  - All fall short, either require a scanner, rule restrictions, not JVM/JS compatible, etc

- Nothing implements grammar composition other than by extension/inheritance

AGL: When a Regular Expression is not enough
What already exists?: Algorithms

- LL
  - Rule restrictions
  - Even LL(*) cannot handle hidden left-recursion
  - GLL decreases restrictions
  - Papers on scannerless GLL
- Earley / Chart parsing
  - Not widely used
  - (Could do with further investigation)
- LR
  - LR(1) least restrictive (memory issues for implementation)
  - GLR decreases restrictions (performance traps)
  - Papers on scannerless GLR
  - RNGLR/BRNGLR, etc - no implementation found
- Others
  - Left-corner, head-corner, etc

- Most of the recent algorithm work does not seem to have left academia !

AGL: When a Regular Expression is not enough
What is AGL?: GLR + modifications

- Move **lookahead** computation to **parse-time** (partially)
    - Rather than pre-computed in the automaton
    - Speeds up automaton generation
    - Slows down parse-time

- Compute automaton **states on-demand**
    - Reduces memory use to only what is required
    - Eliminates needs for time spent on up-front generation of the automaton

- Split reduce action into first and the rest
    - Similar to Left-corner parsing
    - Reduces stack length when parsing List rules (args = [expr / ‘,’]+)

- **Scan-on-demand**

- Enable **embedded grammars**
    - Possible because lookahead is partially computed at parse-time

AGL: When a Regular Expression is not enough
API - Kotlin

Define grammar using a String

```
val p = Agl.processor("""
    <grammar>
""")

val tree = p.parse("<sentence>")
```

Parse a sentence

AGL: When a Regular Expression is not enough
API - Java

```
String grammarStr = ...
LanguageProcessor p = Agl.INSTANCE.parse(grammarStr,null,null);

String sentence = ...
SharedPackedParseTree tree = p.parse(sentence);
```

AGL: When a Regular Expression is not enough
API - JavaScript

```
const grammarStr = ...
const proc = Agl.processorFromString(grammarStr);

const sentence = ...
const tree = proc.parse(sentence);
```

AGL: When a Regular Expression is not enough
Performance

- Its OK, its **usable** - see demo

- Nowhere near as fast as ANTLR V4
  - Only thing that I have been able to realistically compare with
  - Others either have no available implementation
  - Or no library of grammars

- Comparison and performance **improvements** are **in progress**

- Performance **impacted by** grammar **rules**
  - 5 different versions of Java 8 grammar
  - ANTLR execution of ANTLR-optimised by far the fastest
  - AGL-optimised fastest AGL execution
  - AGL executes ANTLR-std faster than ANTLR does

AGL: When a Regular Expression is not enough
Problems

- **Time** to work on it

- Finding other parser generators to compare with

- A performance bottleneck is, Ironically
  - Scanning, use of **Regular Expression engine** on JS
  - There is no ‘lookingAt’ function like there is in JVM
  - Workarounds are not ideal or slower
  - Writing own regex parser is slower - I tried!

- Reuse of automaton (parts) for different goal rules

- Interesting **side-effects** of scan-on-demand
  - Whitespace really is optional !
  - “classA” parses same as “class A”

AGL: When a Regular Expression is not enough
Conclusion

Met my initial requirements - Mostly

1. Runtime build: **Yes**
2. No rule limitations: **Yes** - GLR + my variations
3. No reserved words: **Yes** - Scan-on-demand
4. Lists of items: **Yes** - my List rules
5. Grammar composition: **Yes** - extension and embedding
6. Any goal rule: **Only in the API** - separate automaton for each
7. Multi-platform: **Yes** - thanks to Awesome Kotlin
8. Performant: Partial - **useable** but was hoping for better - may get there

AGL: When a Regular Expression is not enough
Demo description

- It is all executed in the browser

- There is NO server (other than to serve the .html, .css, .js files)

- The Ace and Monaco Integrations are separate libraries

- The demo shows:
  - Writing a sentence in a given language with
    - Syntax highlighting - based on scan initially, then a parse tree if available
    - Autocomplete
    - A parse tree displayed
    - A simple auto constructed ASM
  - Modify (at runtime) the grammar and the highlighting rules
  - Or just select a different grammar (a few built in examples)
  - Or write your own from scratch

AGL: When a Regular Expression is not enough
Questions

My questions

- Is it useful or a waste of my time to continue?

- Suggestions of similar/useful research I may have missed?
- Suggestions of similar implementations to compare with?

- Any grammars you would like me to test it with?

- Anyone got a use-case or application that would find this useful?
- Anyone got a commercial project that wants to use it?