SMACC: PARSER GENERATION AND MORE

Jason Lecerf | 09/13/2018

SMACC

The Smalltalk Compiler-Compiler

- Parser generator
- AST classes generator
- Rewriting engine
- More?

PARSER GENERATION...

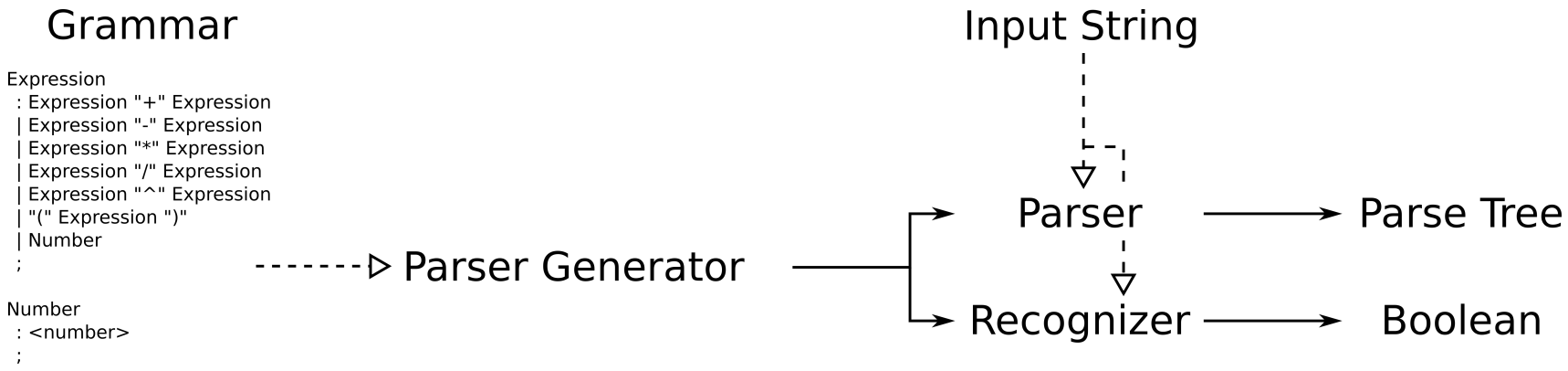


Figure: Basic workflow of parser generation

...WITH AST!

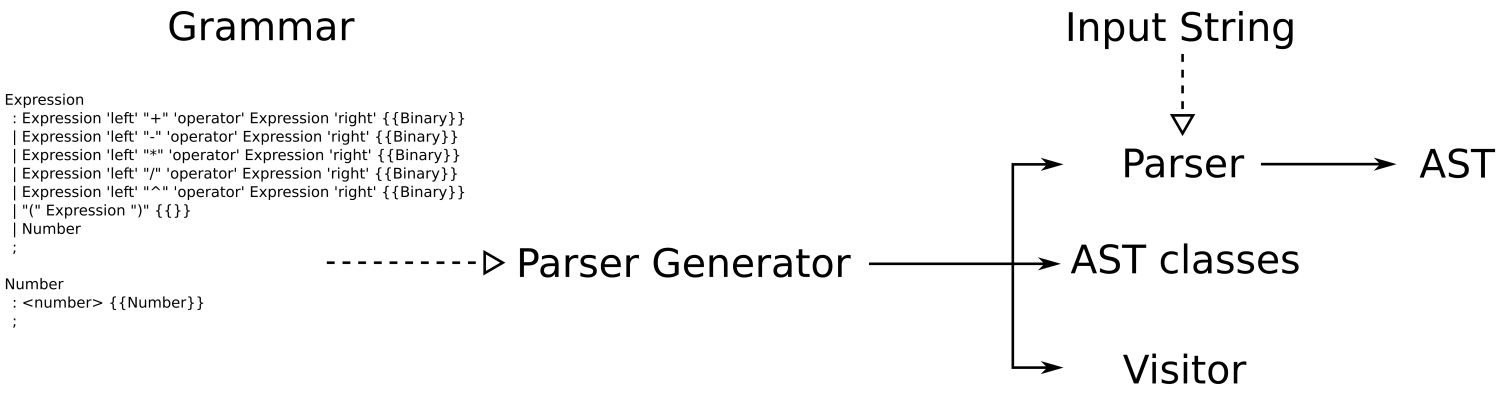


Figure: Now with AST and visitor generation

PARSING TECHNIQUES

LR

- Deterministic
- Practical parse time: $O(n)$, n is the size of the input
- Does not handle ambiguities in languages

Generalized-LR (GLR)

- Non-deterministic
- Exponential with the number of ambiguities
- Optimal for nearly-deterministic grammars

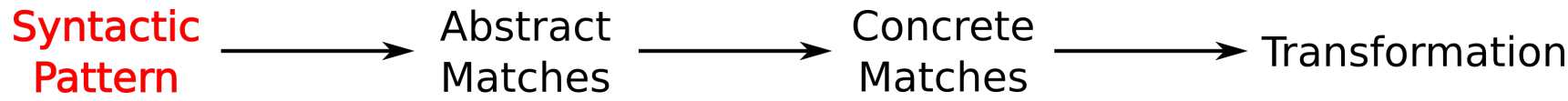
THE REWRITE ENGINE

Code transformation engine

- Pattern matching + transformation
- Works with syntactic patterns
- 2-step matching technique
- Based on parsing as intersection
- Automatic source transformation

```
1 ‘a‘ + ‘b‘ -> ‘a‘ ‘b‘ +
```

SYNTACTIC PATTERN AND METAVARIABLES



Syntactic

- written in the syntax of the host language
- will be parsed by the host language parser

```
1 ‘a‘ + ‘b‘
```

Metavariables

- Variables in the logic sense
- Bound after matching to a subtree
- Are a token of the host language

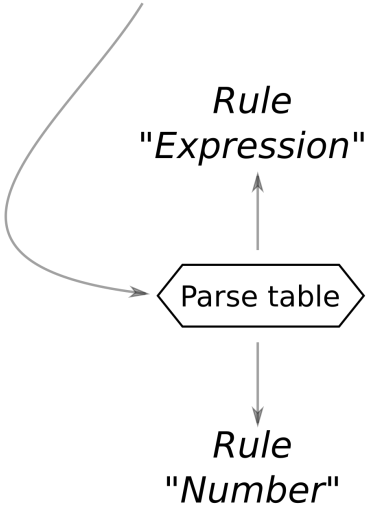
```
1 ‘esug‘ ^ 2
```

SYNTACTIC PATTERN MATCHING (1/3)

Syntactic Pattern → **Abstract Matches** → Concrete Matches → Transformation

Pattern `esug` ^ 2

Parsing... . `esug` ^ 2 `esug` . ^ 2 `esug` ^ . 2 `esug` ^ 2 .




```
Expression
 : Expression 'left' "+" 'operator' Expression 'right' {{Binary}}
 | Expression 'left' "-" 'operator' Expression 'right' {{Binary}}
 | Expression 'left' "*" 'operator' Expression 'right' {{Binary}}
 | Expression 'left' "/" 'operator' Expression 'right' {{Binary}}
 | Expression 'left' "^" 'operator' Expression 'right' {{Binary}}
 | "(" Expression ")" {{}}
 | Number
 ;

Number
 : <number> {{Number}}
 ;
```

NodeType — *typed esug metavariable*

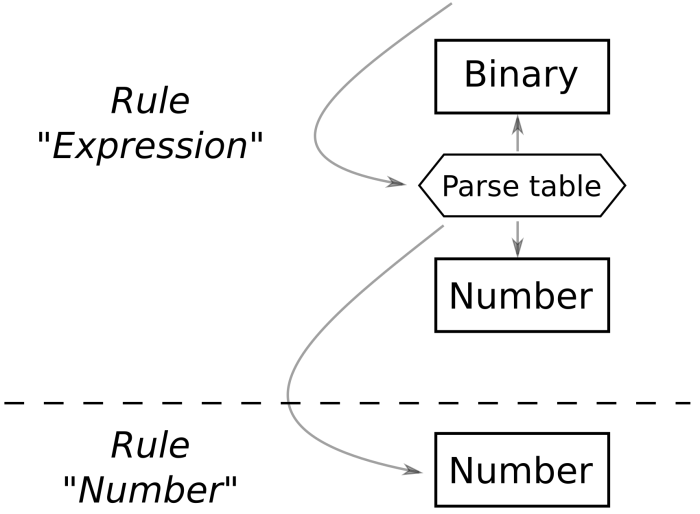
NodeType — *standard ast node*

SYNTACTIC PATTERN MATCHING (2/3)

Syntactic Pattern → **Abstract Matches** → Concrete Matches → Transformation

Pattern `esug` ^ 2

Parsing... . `esug` ^ 2 `esug` . ^ 2 `esug` ^ . 2 `esug` ^ 2 .



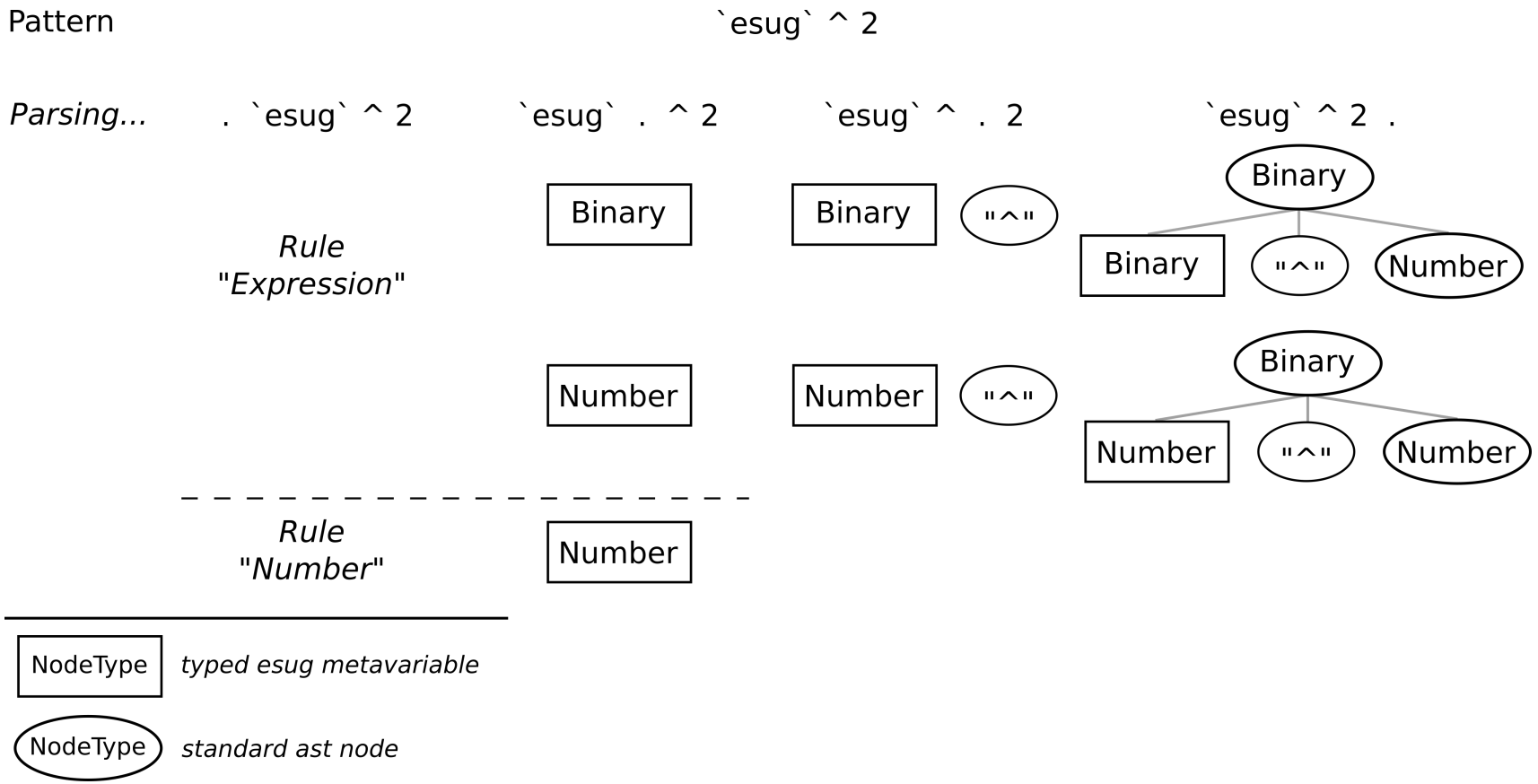

```
Expression
 : Expression 'left' "+" 'operator' Expression 'right' {{Binary}}
 | Expression 'left' "-" 'operator' Expression 'right' {{Binary}}
 | Expression 'left' "*" 'operator' Expression 'right' {{Binary}}
 | Expression 'left' "/" 'operator' Expression 'right' {{Binary}}
 | Expression 'left' "^" 'operator' Expression 'right' {{Binary}}
 | "(" Expression ")" {{}}
 | Number
 ;

Number
 : <number> {{Number}}
 ;
```

NodeType *typed esug metavariable*

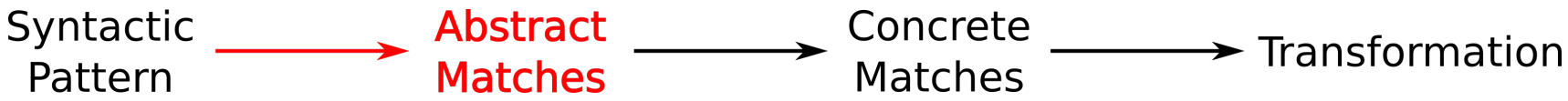
NodeType *standard ast node*

SYNTACTIC PATTERN MATCHING (3/3)

Syntactic Pattern → **Abstract Matches** → Concrete Matches → Transformation



PARSING AS INTERSECTION



Formal proof: "The intersection of a context-free language with a regular language is again a context-free language"

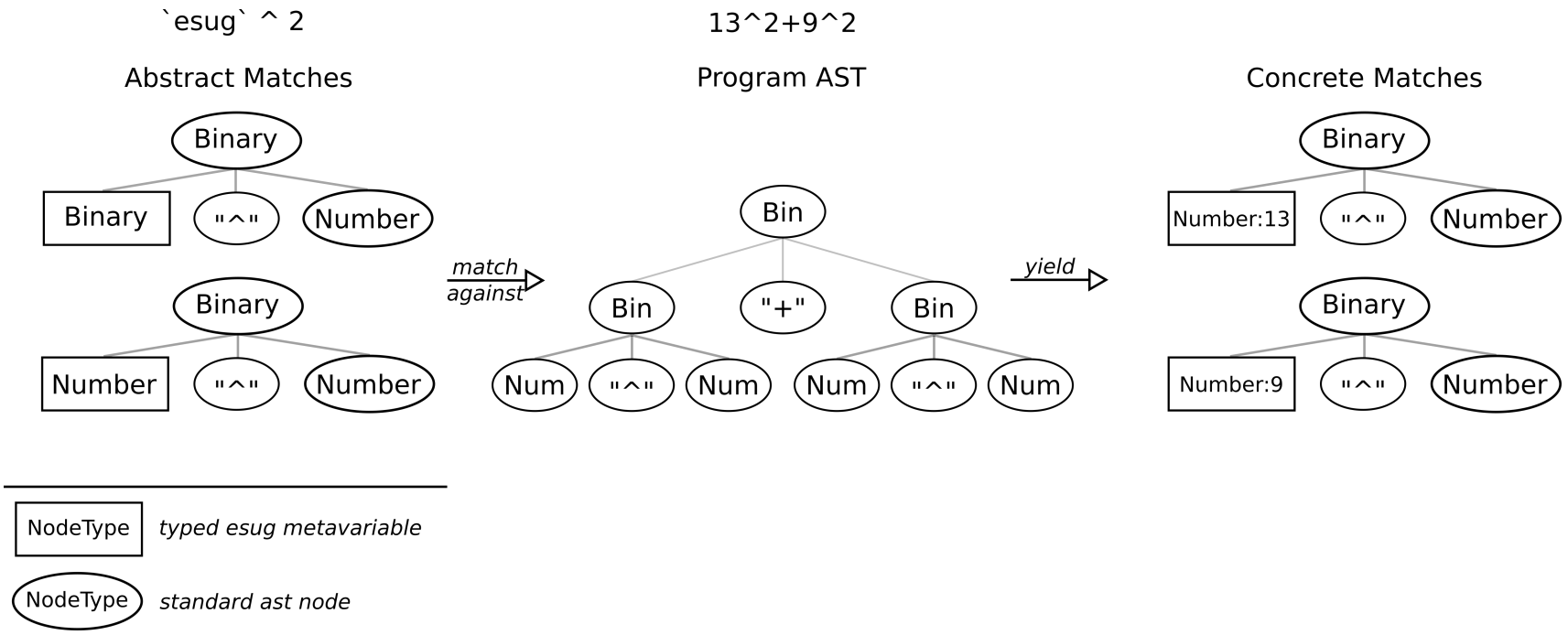
In our case:

- the first CF language is the host language
- the regular language is the pattern
- the second CF language is a grammar containing only the rules that can be reached from the pattern
  - this resulting grammar is equivalent to a forest of matches
  - by applying all of its rules

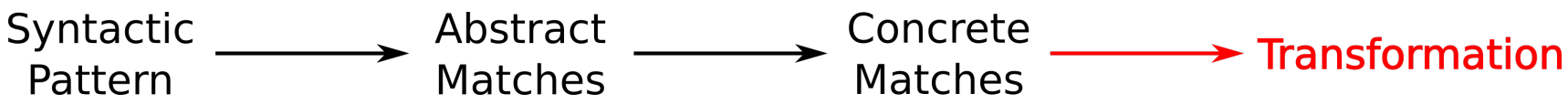
Makes it as easy as parsing a regular language with GLR

MATCHING THE SOURCE

Syntactic Pattern → Abstract Matches → **Concrete Matches** → Transformation



TRANSFORMING THE MATCH




```
1 ‘esug‘ ^ 2 -> ( ‘esug‘ * ‘esug‘ )
```

Automated source rewriting

- the transformation part is not parsed
- metavariables are replaced by the source of their matched subtree

Node rewrite

- play with the subtrees in Smalltalk
- replace nodes
- not automated unlike source transforms

APPLICATIONS

Applications of the Rewrite Engine

- Code migration
- Refactorings
- Search engine
- Code specialization
- ...

MORE?

Upcoming features in my thesis:

- Search Engine
- Refactoring Engine
- Refactoring DSL

Tentative features:

- Code critics for grammars
- Code example generalizer

SEARCH ENGINE

Syntactic pattern based search engine:

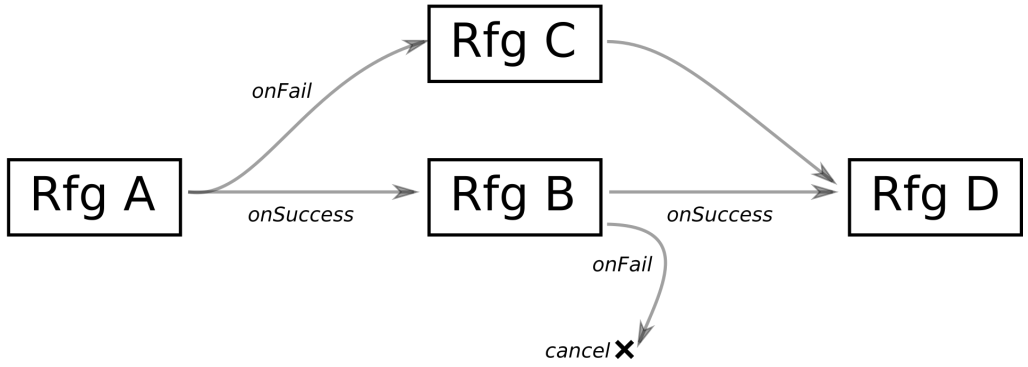
- Pattern matching and transformation are closely related
- Decoupling: proper syntactic search engine

Example generalizer:

- Extension of a search engine
- Generalize a code example to a pattern with metavariable
- Progressively introduce generality and see its effects on the matches

REFACTORING ENGINE AND DSL

Refactoring Sequence



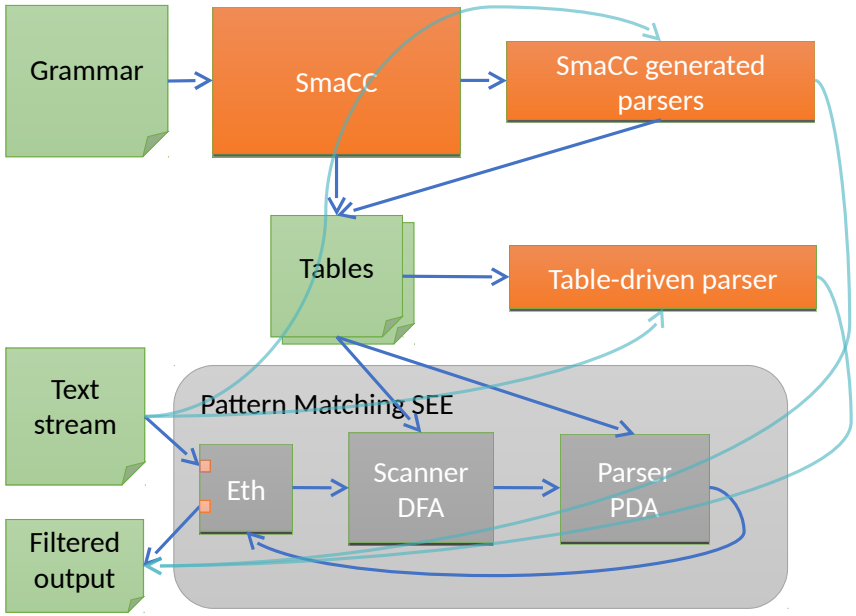
Principle = patterns and matches as first-class citizens

```
1 ast := MyParser parse: program.
2 pattern := [ :power | 'esug' ^ 'power' ].
3 pattern2 := ['Number:2' in: pattern ].
4 match1 := ast select: pattern2.
5 match2 := match1 select: [ :esug | esug := '
    2019' ]
```

FUN FACT: HARDWARE GENERATION FROM SMACC

The M2DC(H2020) Pattern Matching SEE

- Write grammar in SmaCC
- Export grammar objects and parse tables as simplified parse tables
- Load reconfigurable hardware with Ethernet IP (FPGA)
- Run... Extract patterns in SLURM events stream over UDP
- Table-driven scanner is 3 times faster than SmaCC's generated



LAST WORDS

To try:

- SmaCC's repository: `https://github.com/j-brant/SmaCC`
- My development fork: `https://github.com/SmaCCRefactoring/SmaCC`

To know more:

- The SmaCC booklet: `https://github.com/SquareBracketAssociates/Booklet-Smacc`
- Paper on the Rewrite Engine: `https://hal.archives-ouvertes.fr/hal-01851857`
- Paper presentation @ICSME, 09/28/2018, Madrid

To discuss:

- jason.lecerf@cea.fr
- brant@refactoryworkers.com
- thierry.goubier@cea.fr

Thanks!

Any questions?

Commissariat à l'énergie atomique et aux énergies alternatives
Campus Saclay Nano-Innov | F-91191 Gif-sur-Yvette Cedex
www-list.cea.fr

Établissement public à caractère industriel et commercial | RCS Paris B 775 685 019