UNIVERSITÄT
BERN

Mastering Embedded Languages

Lukas Renggli
Software Composition Group
University of Bern, Switzerland

Smalltalk is a highly dynamic language

Smalltalk syntax is locked down

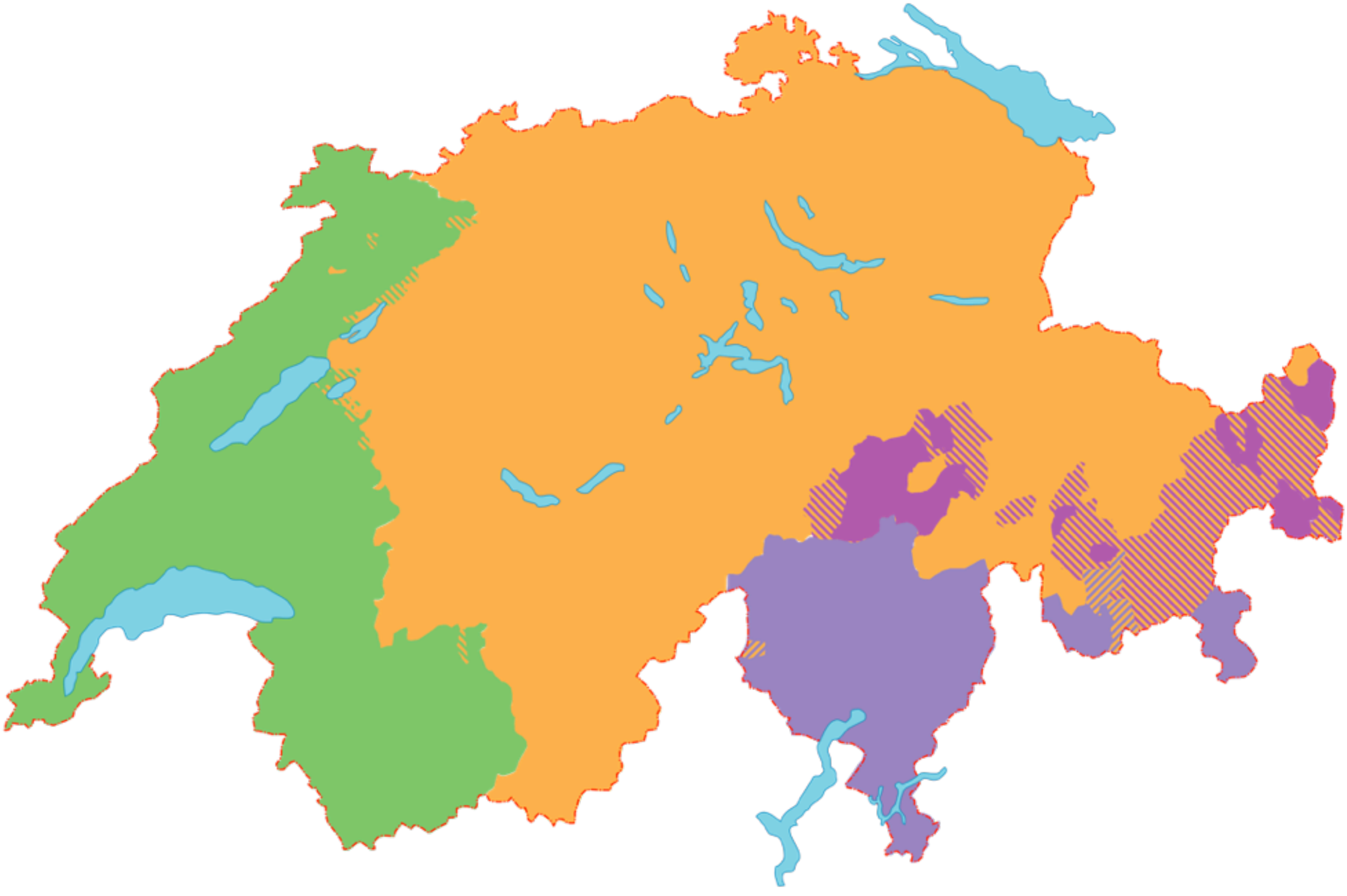
Photo by Simeon Eichmann, sxc.hu

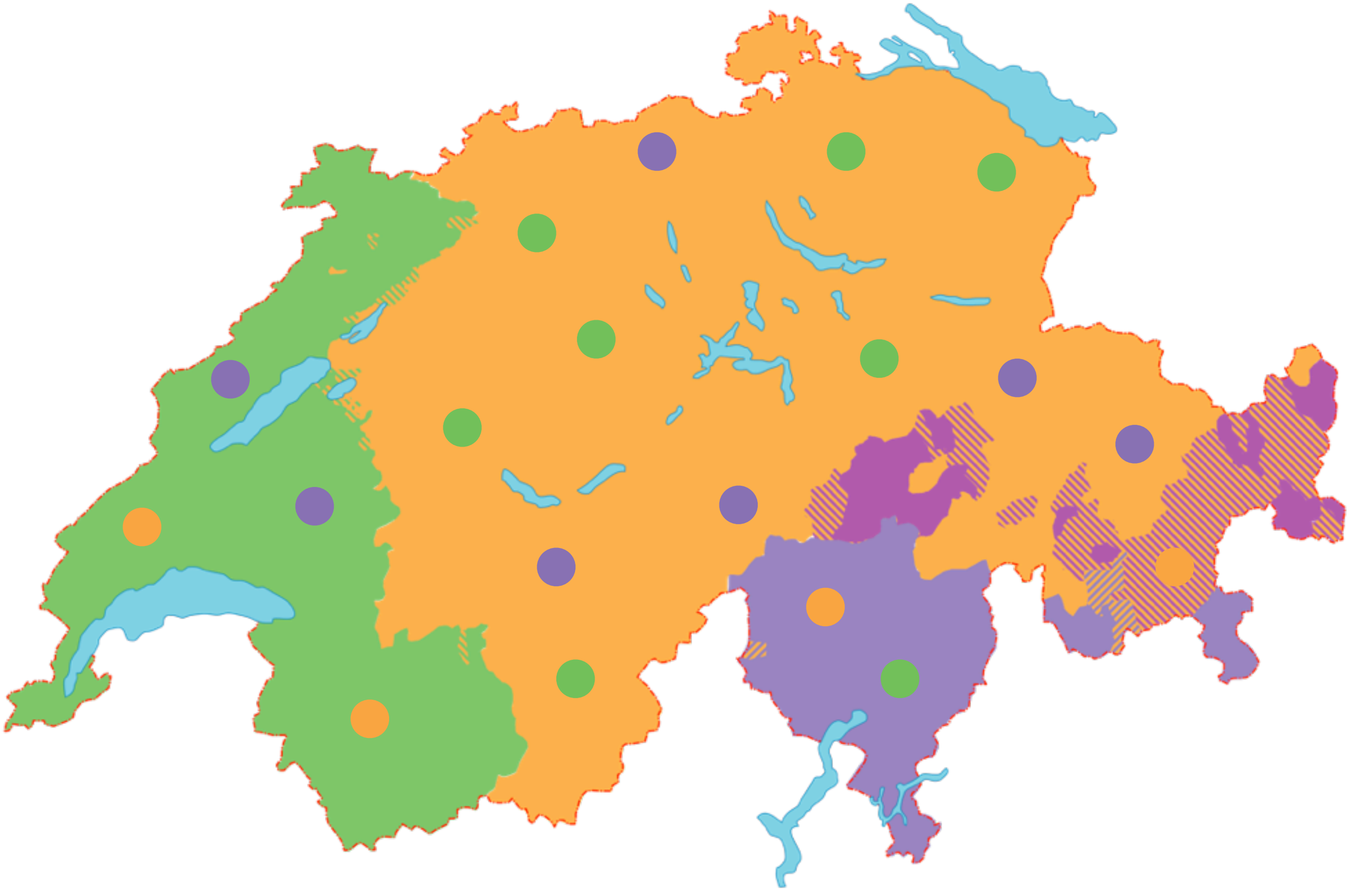
Host-language compiler can be patched

Patching wrecks your development tools

Helvetia

Mastering Embedded Languages





MIGROS

coop

§

§

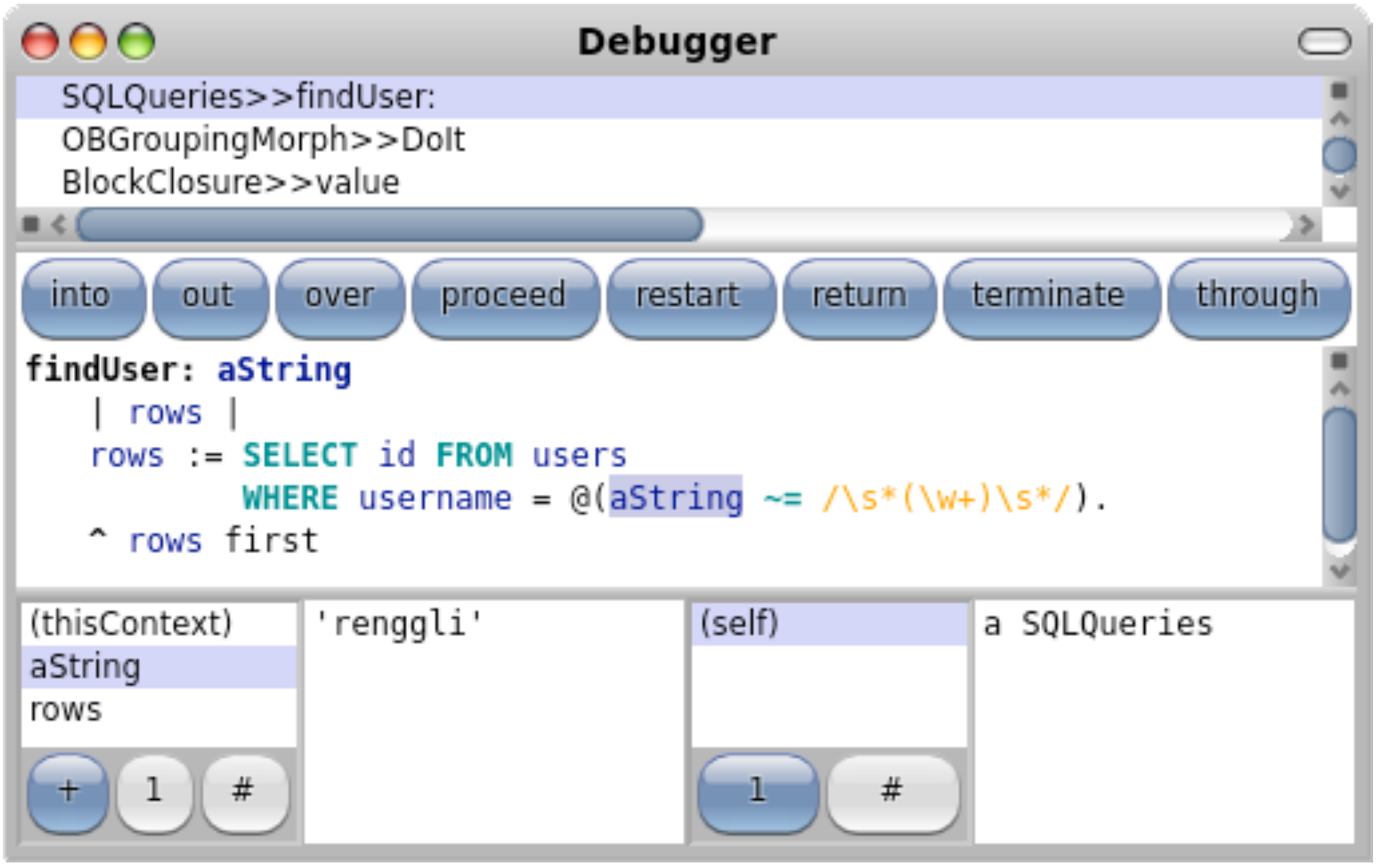
MIGROS

MIGROS

coop

§

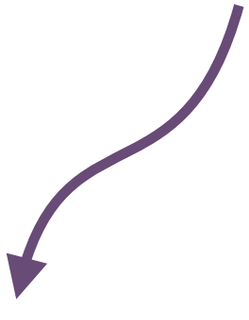
coop


Debugger
SQLQueries>>findUser:
OBGroupingMorph>>DoIt
BlockClosure>>value
into
out
over
proceed
restart
return
terminate
through
findUser: aString
| rows |
rows := SELECT id FROM users
WHERE username = @(aString ~= /\s*(\w+)\s*/).
^ rows first
(thisContext)
aString
rows
'renggli'
(self)
a SQLQueries
+
1
#
1
#

Requirements

Adopt, Extend, Overload

change semantics



Adopt, Extend, Overload

change semantics

Adopt, Extend, Overload

introduce new syntax

change semantics

Adopt, Extend, Overload

introduce new syntax

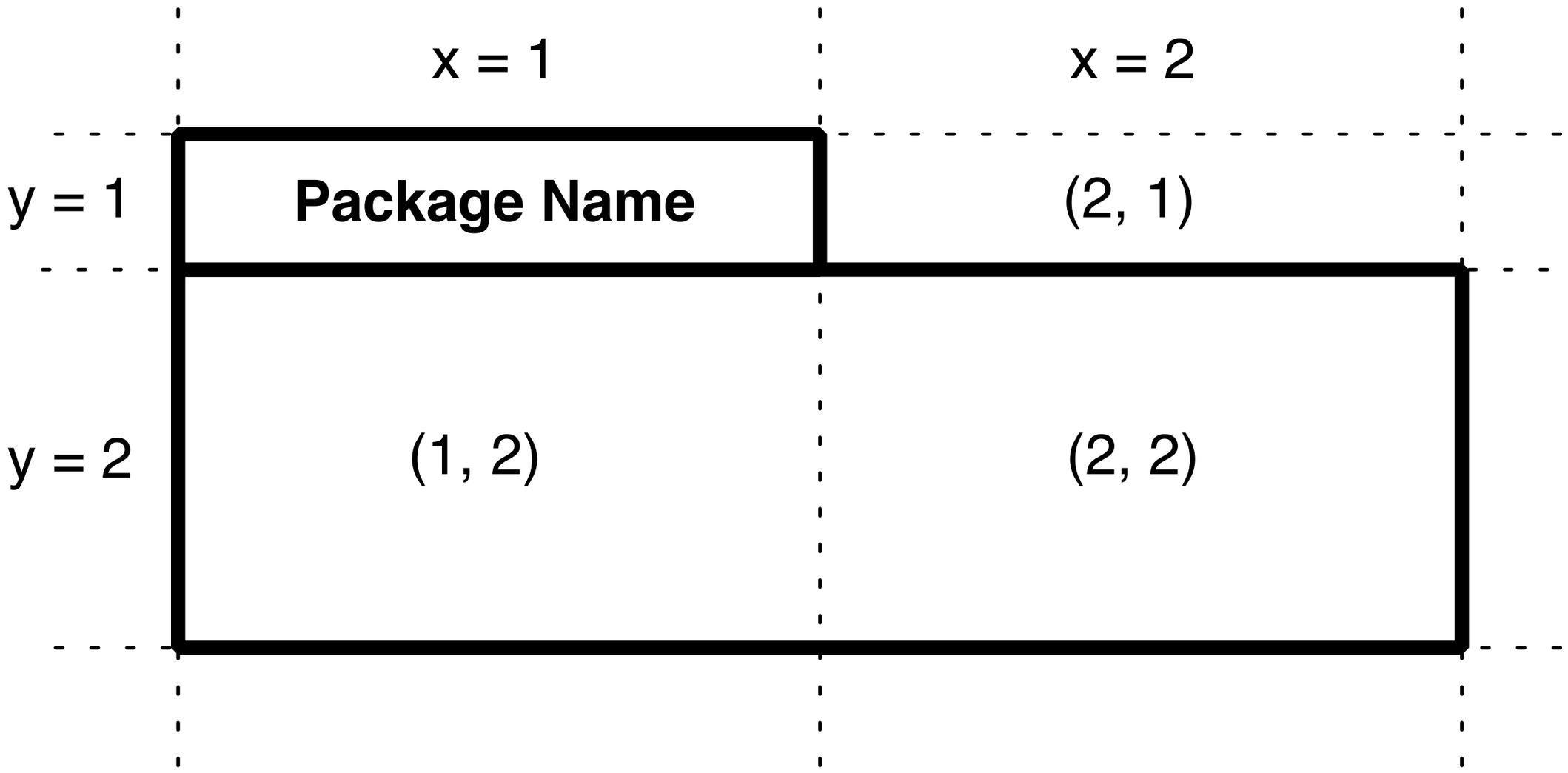
change syntax & semantics

Multiple Context Dependent Languages

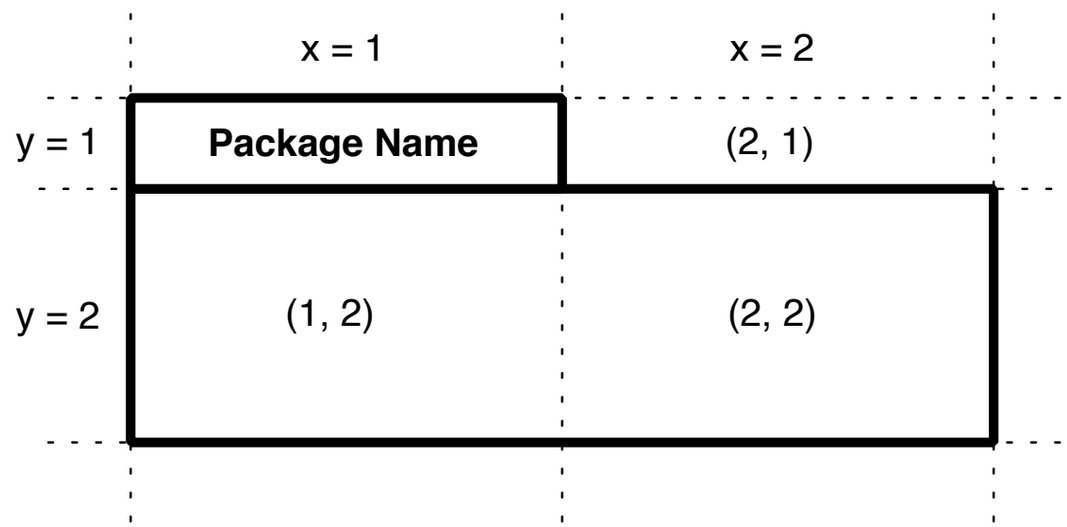
Homogeneous Data and Code Abstraction

Homogeneous Tool Support

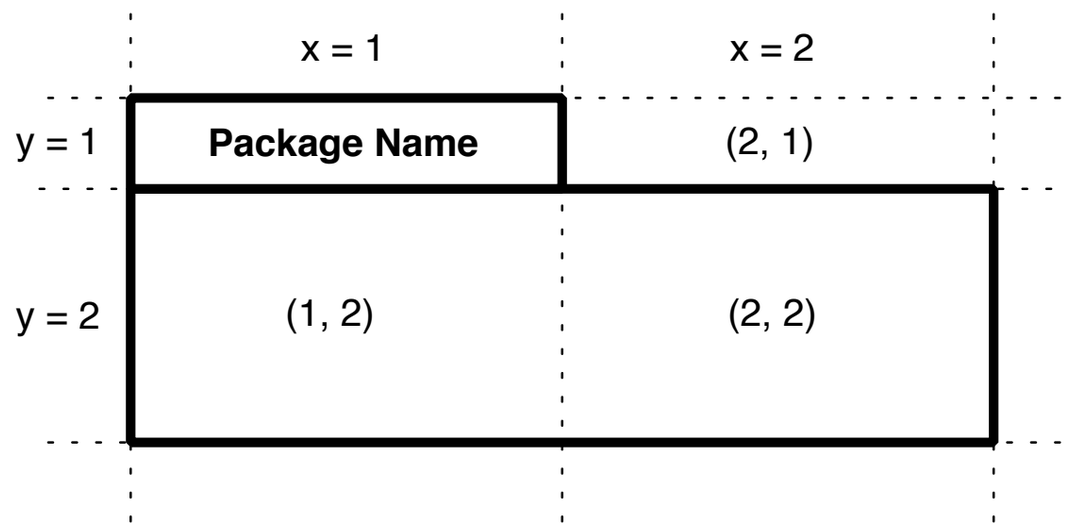
Context Specific Languages with Homogeneous Tool Integration


x = 1
x = 2
y = 1
Package Name
(2, 1)
y = 2
(1, 2)
(2, 2)

```
aBuilder row grow.
aBuilder row fill.

aBuilder column grow.
aBuilder column fill.

aBuilder x: 1 y: 1 add: (LabelShape new
      text: [ :each | each name ];
      borderColor: #black;
      borderWidth: 1;
      yourself).
aBuilder x: 1 y: 2 w: 2 h: 1 add: (RectangleShape new
      borderColor: #black;
      borderWidth: 1;
      width: 200;
      height: 100;
      yourself)
```

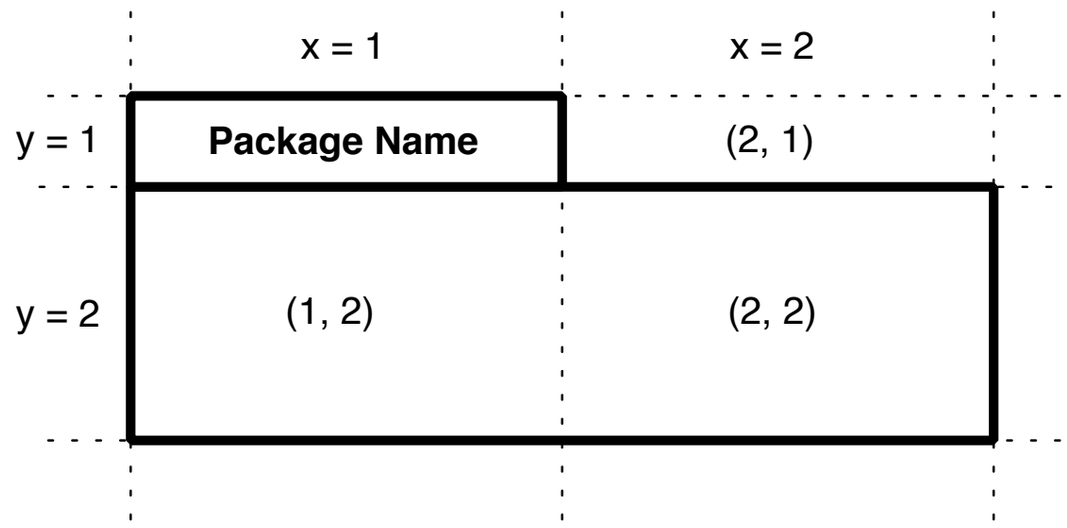


```
row = grow.
row = fill.

column = grow.
column = fill.

(l , l) = label
        text: [ :each | each name ];
        borderColor: #black;
        borderWidth: l.

(l , 2) - (2 , l) = rectangle
        borderColor: #black;
        borderWidth: l;
        width: 200;
        height: 100.
```



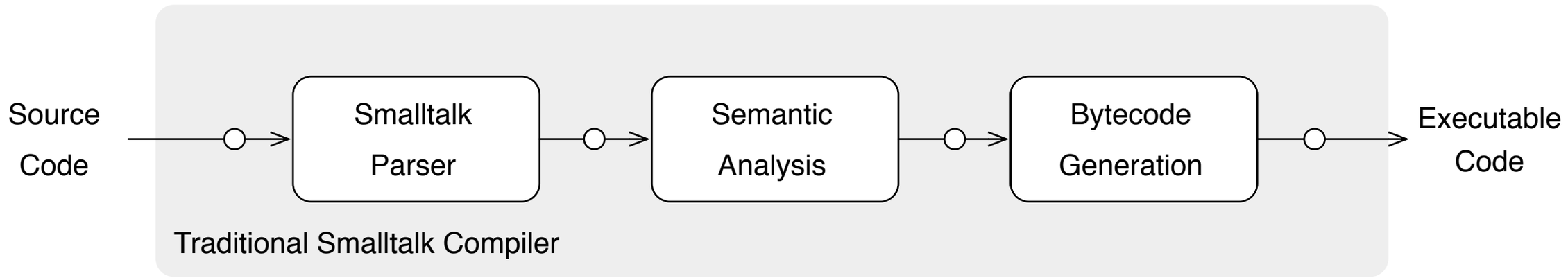
A *pidgin* is a simplified form of the host language. It introduces new vocabulary and semantic meaning.

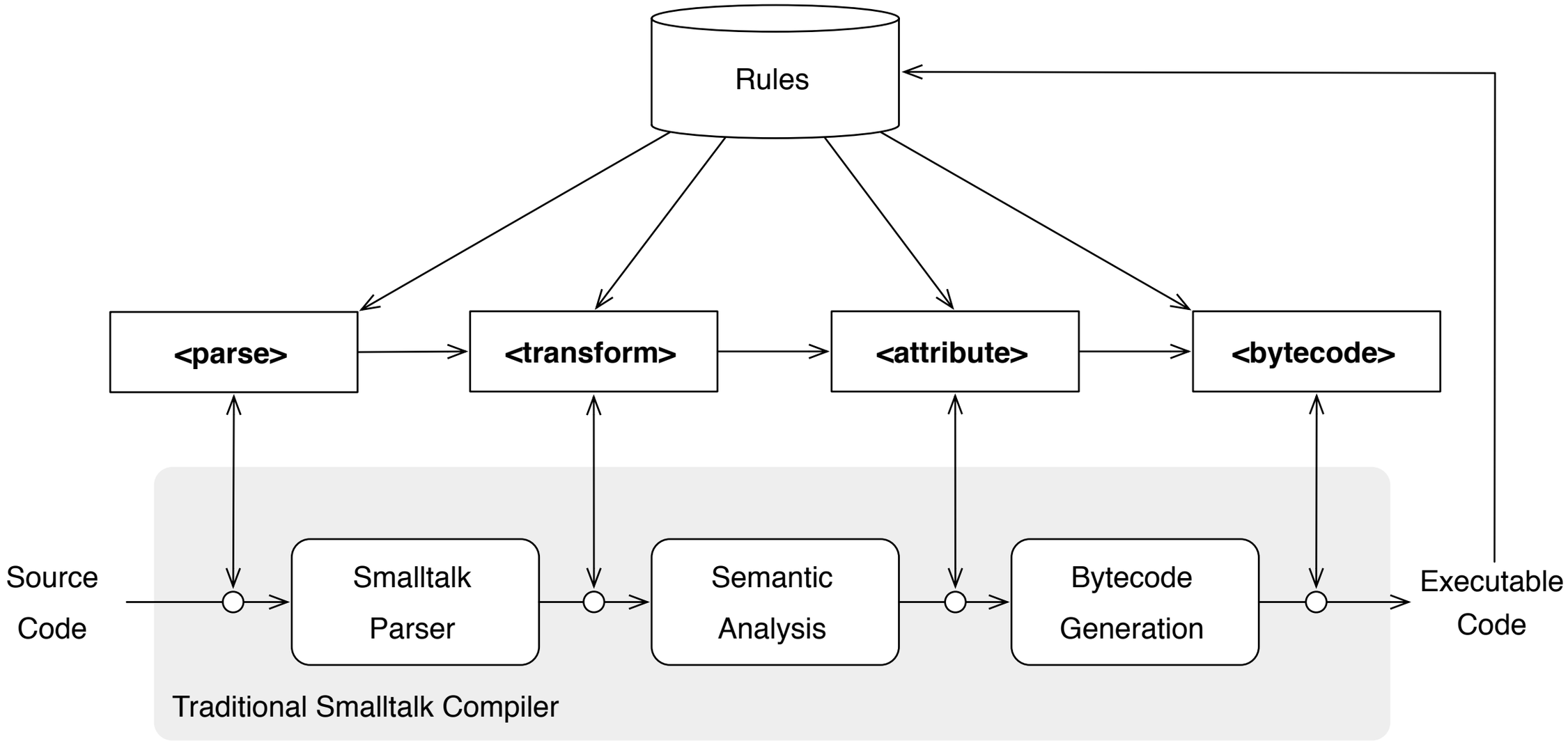
```
shape {
        cols: #grow, #fill;
        rows: #grow, #fill;
}
label {
        position: 1 , 1;
        text: [ :each | each name ];
        borderColor: #black;
        borderWidth: 1;
}
rectangle {
        position: 1 , 2;
        colspan: 2;
        borderColor: #black;
        borderWidth: 1;
        width: 200;
        height: 100;
}
```

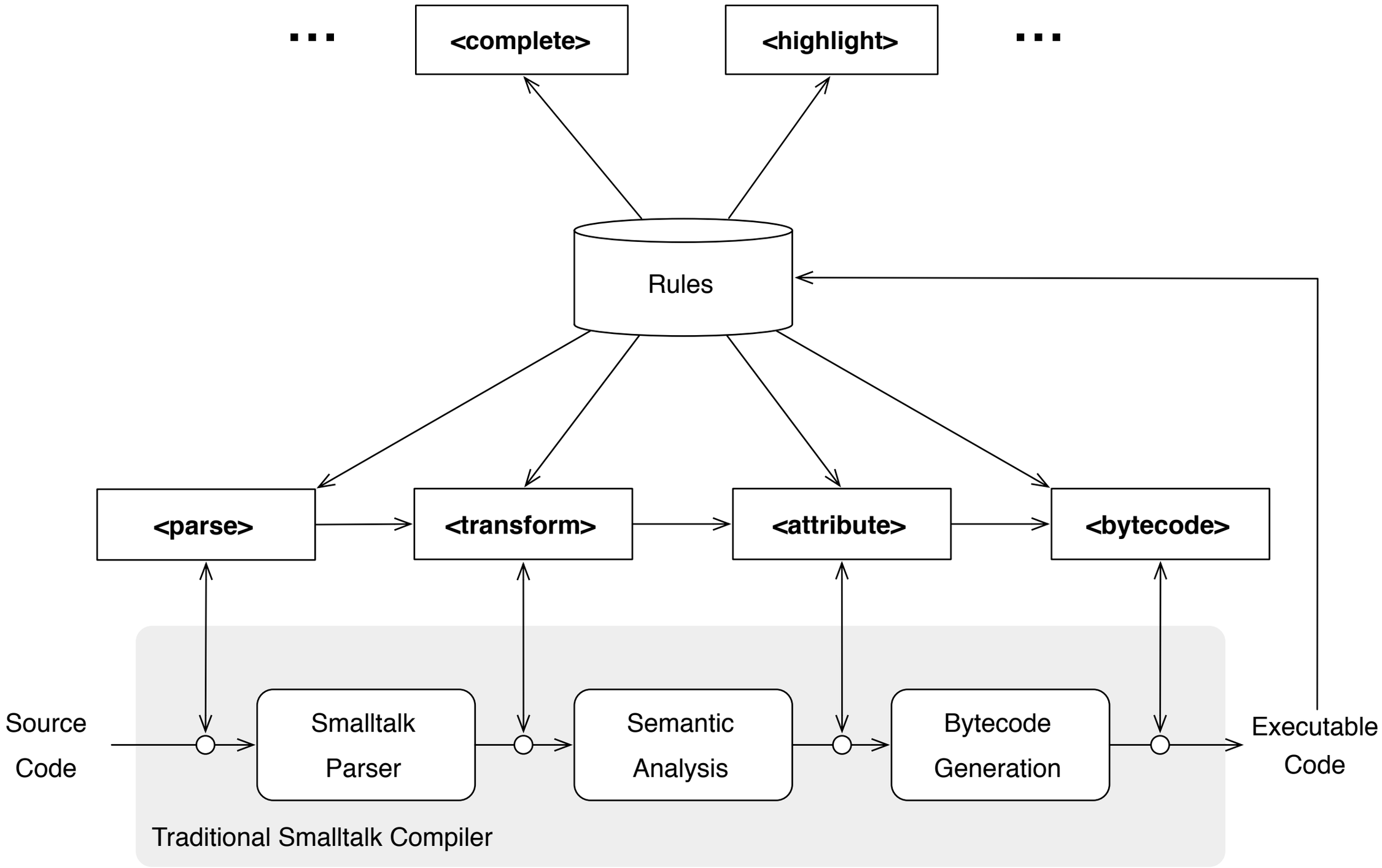


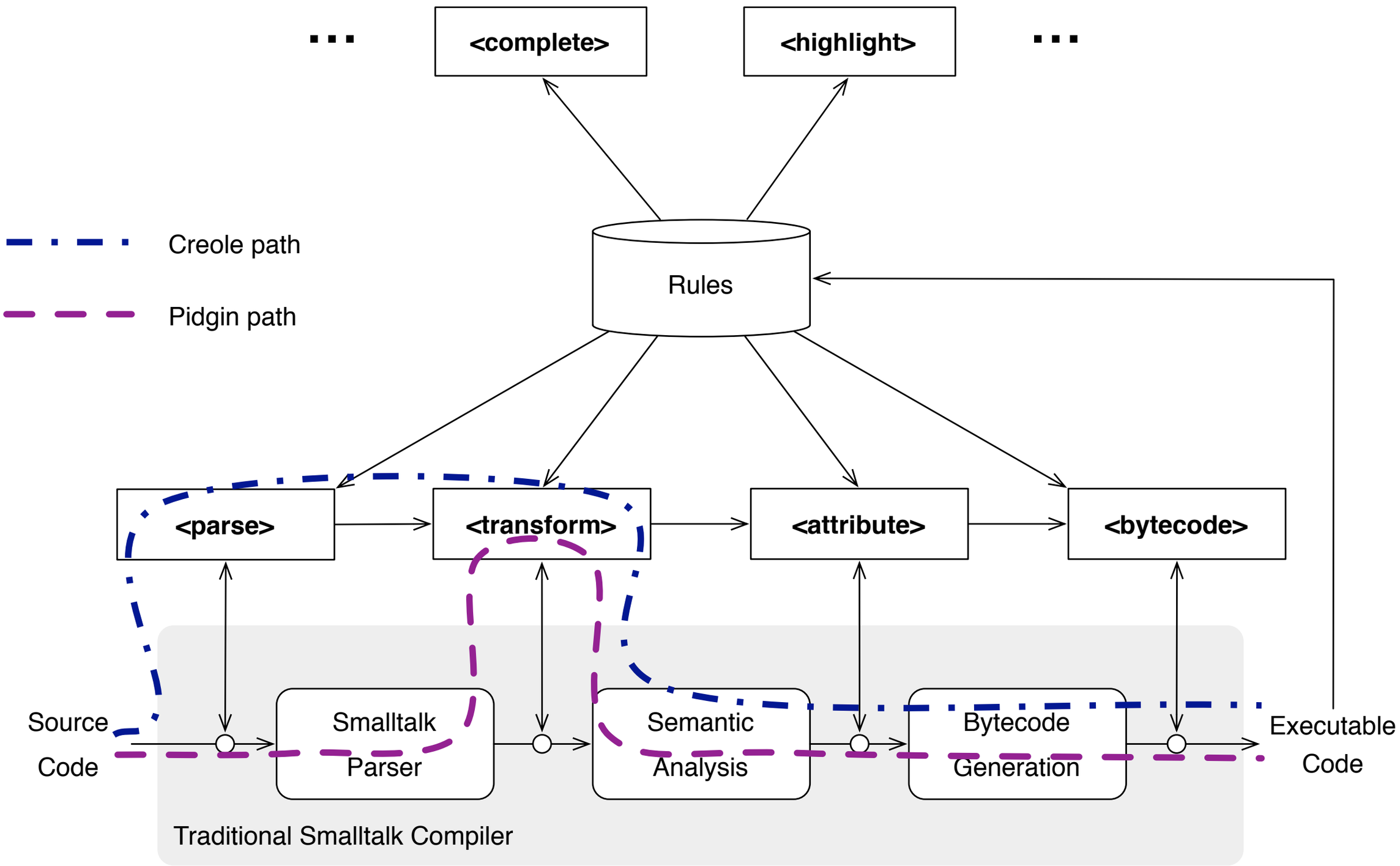
A *creole* a is a new language formed from the contact of multiple languages.

The Helvetia Model


Source
Code
Smalltalk
Parser
Semantic
Analysis
Bytecode
Generation
Executable
Code
Traditional Smalltalk Compiler

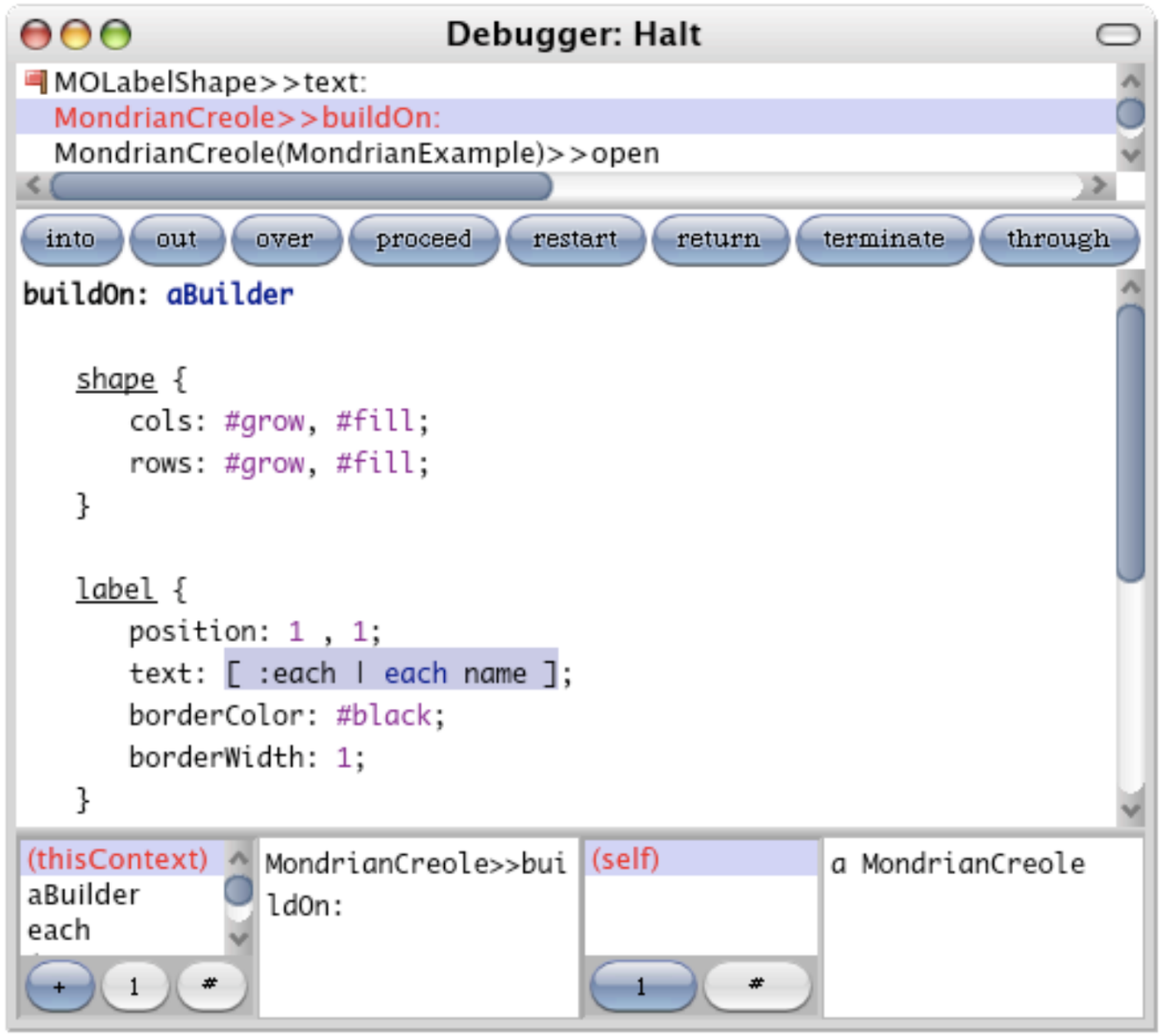

Rules
<parse>
<transform>
<attribute>
<bytecode>
Source
Code
Smalltalk
Parser
Semantic
Analysis
Bytecode
Generation
Executable
Code
Traditional Smalltalk Compiler


...
<complete>
<highlight>
...
Rules
<parse>
<transform>
<attribute>
<bytecode>
Source
Code
Smalltalk
Parser
Semantic
Analysis
Bytecode
Generation
Executable
Code
Traditional Smalltalk Compiler


...
<complete>
<highlight>
...
Creole path
Pidgin path
Rules
<parse>
<transform>
<attribute>
<bytecode>
Source
Code
Smalltalk
Parser
Semantic
Analysis
Bytecode
Generation
Executable
Code
Traditional Smalltalk Compiler

Homogenous Embedding

Homogeneous Tools



IV + VII = XI

```
,>++++++[<---------->-],[<+>-]<.
```

Helvetia

Host Language

Helvetia

Renggli *et al.*
ECOOP 2010

Host Language

Language Boxes

Modular Language Changes

Language 1

Language 2

Helvetia

Syntax

Editor

Compiler

Debugger

Language 1

Language 2

Helvetia

Syntax

Editor

Compiler

Debugger

Language 1

Language 2

Helvetia

Syntax

Editor

Compiler

Debugger

Language 1

Language 2

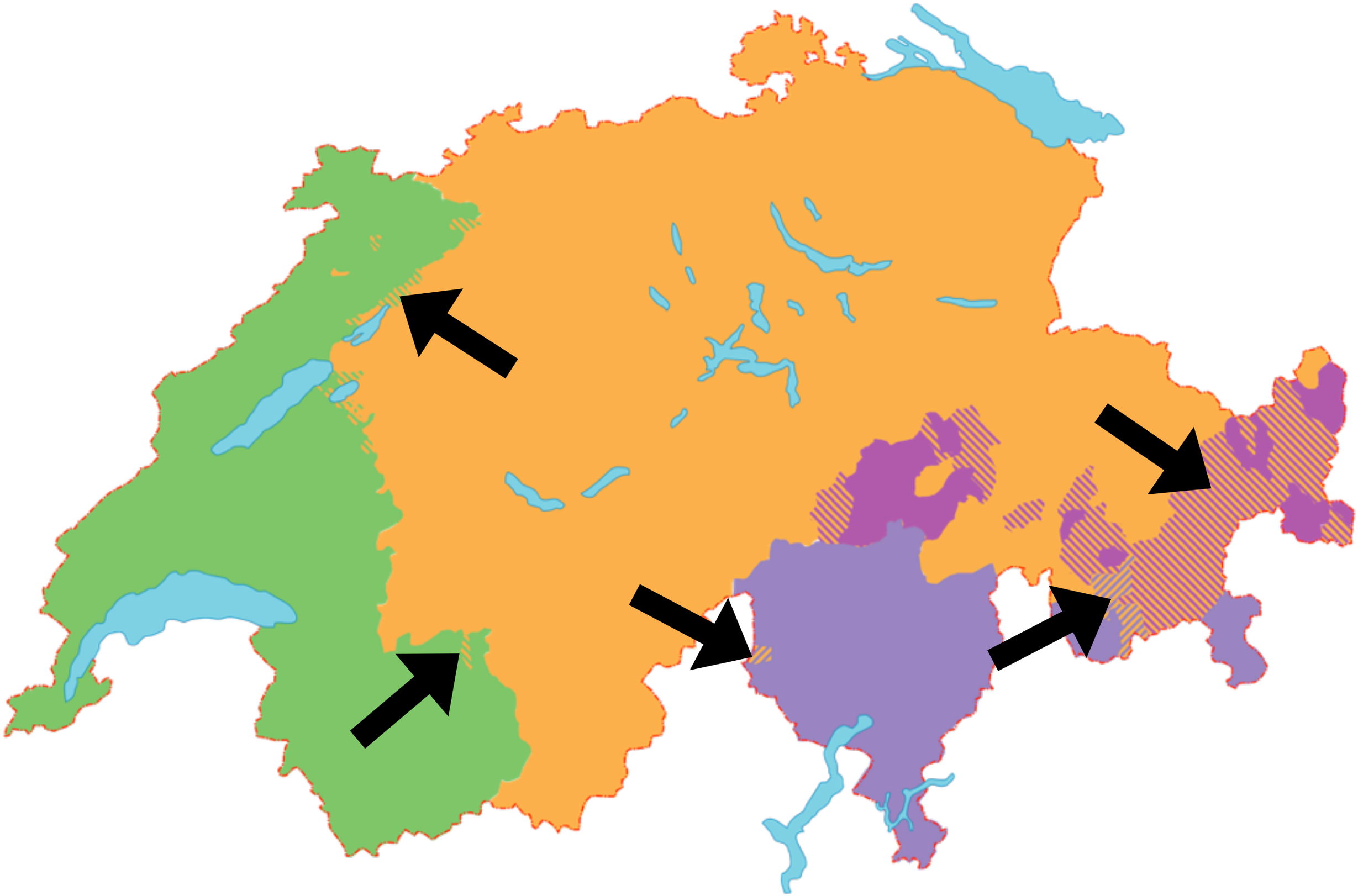
Syntax

Editor

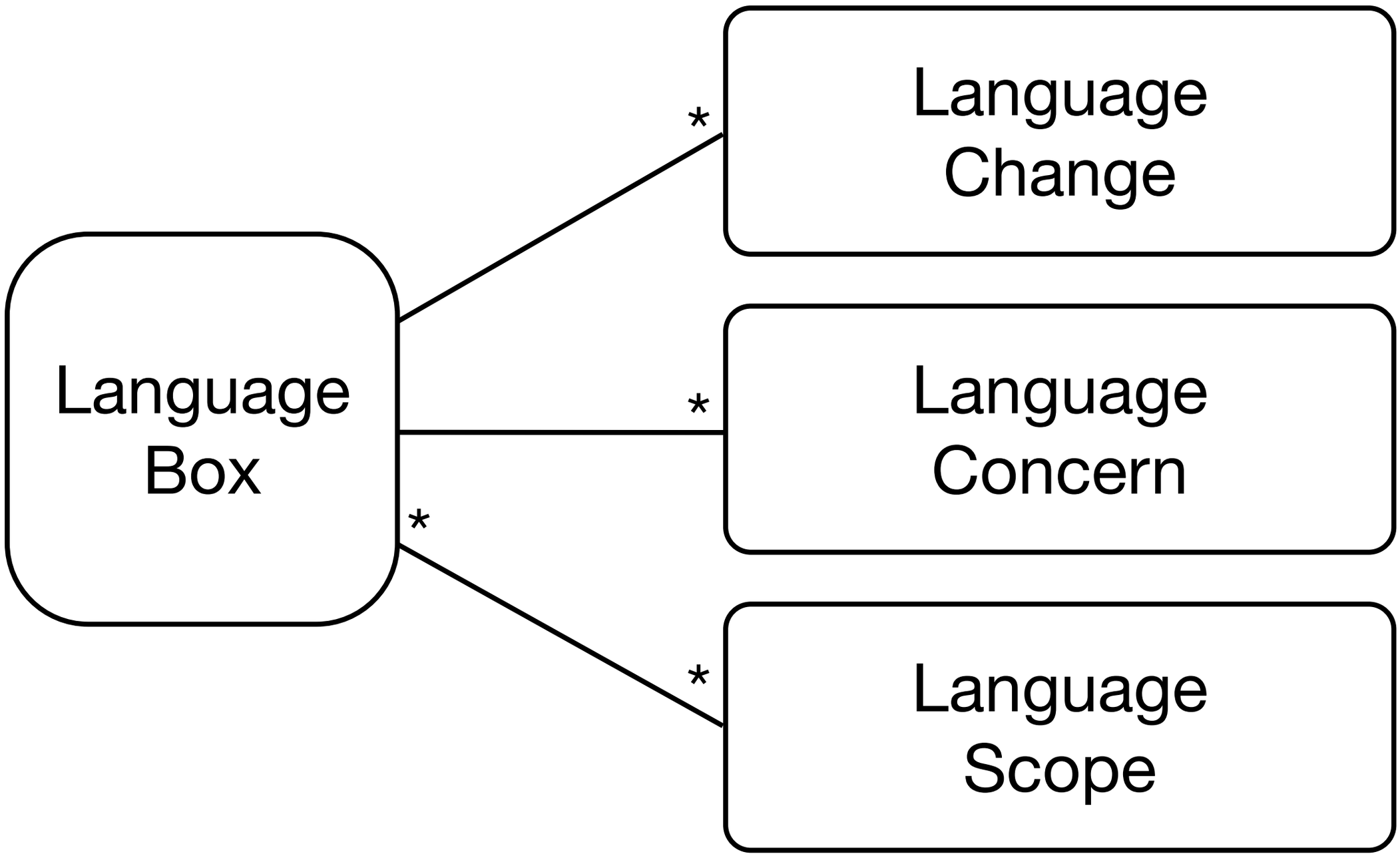
Compiler

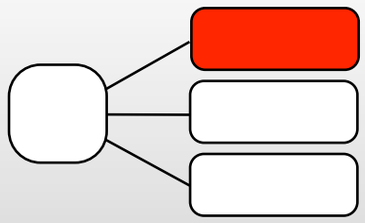
Debugger

Helvetia



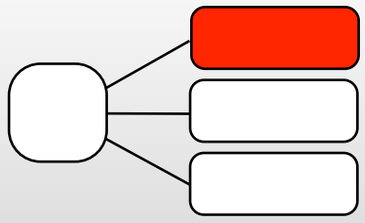
```
/\w+@\w+\.\w+/
```


Language Box
*
Language Change
*
Language Concern
*
Language Scope



Language Change

High-level Description

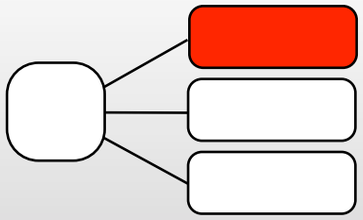


Language Change

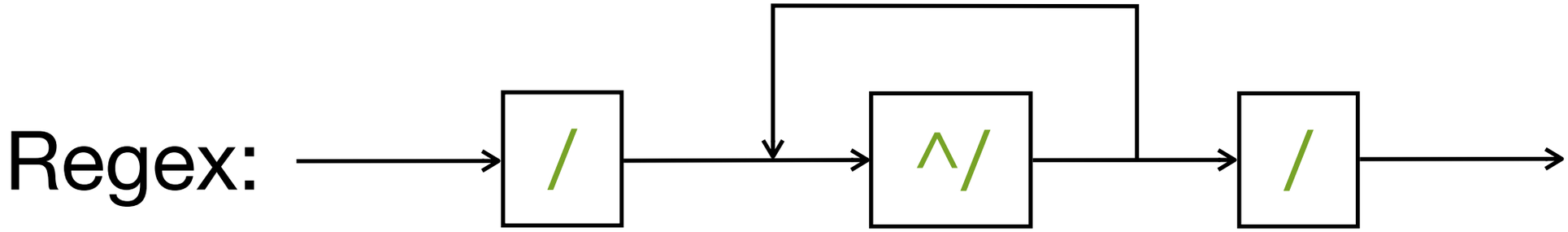
Add Regex

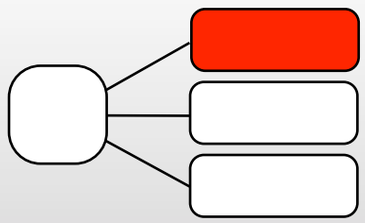
as an additional choice

to Primary.

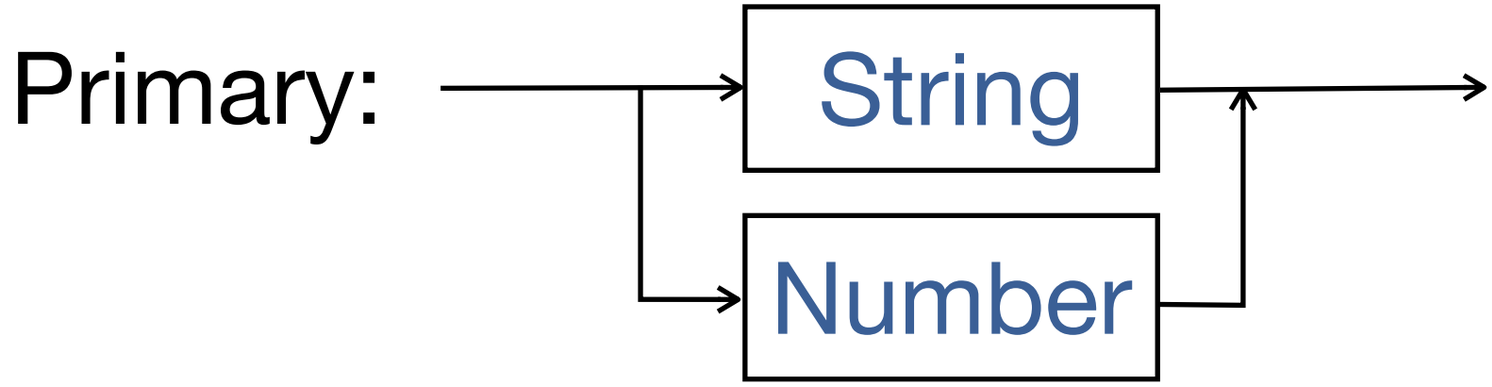


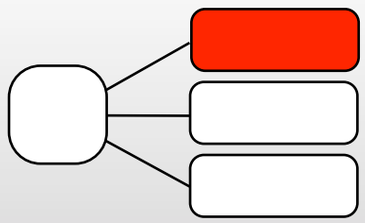
Language Change



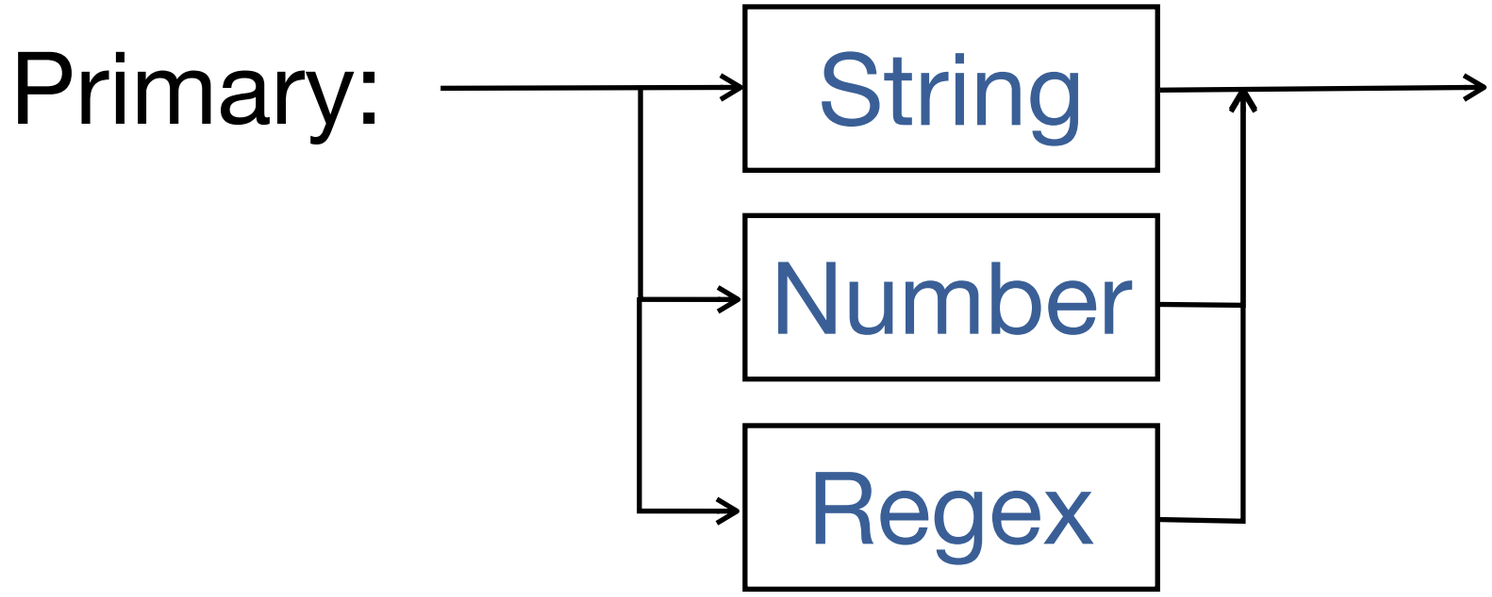


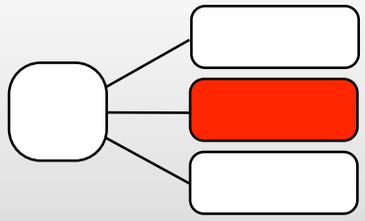
Language Change





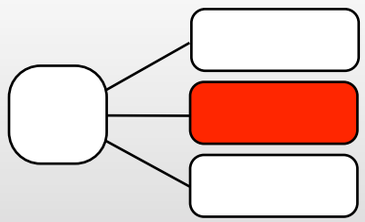
Language Change





Language Concern

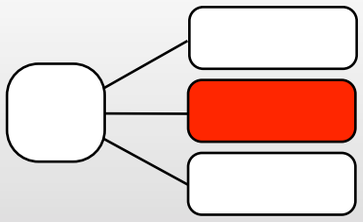
Production Action



Language Concern

Compiler

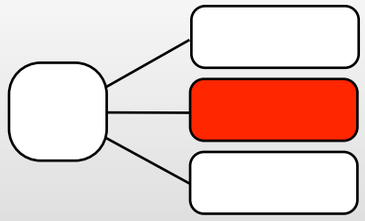
aToken asRegex lift



Language Concern

Highlighting

aToken -> Color orange



Language Concern

Custom Inspector

Context Menus

Search

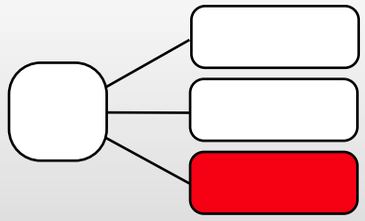
Navigation

Other Concerns

Error Correction

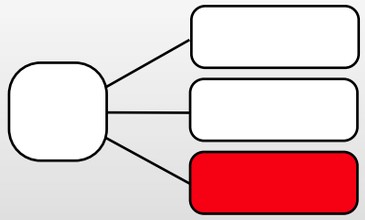
Code Expansion

Code Completion



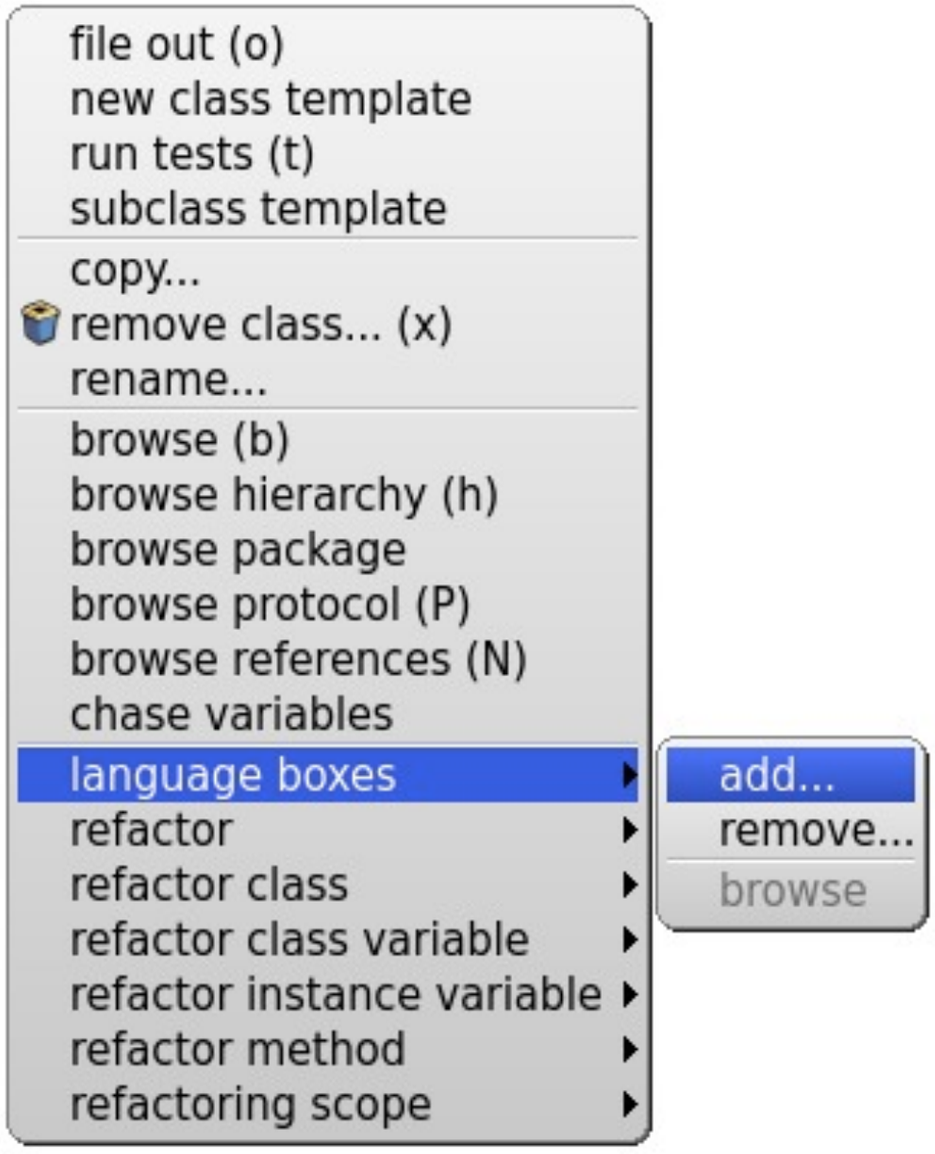
Language Scope

Active?



Language Scope

- System
- Packages
- Classes
- Methods



Multiple Language Extensions

Host Language

Grammar

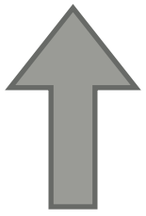
Host Language Grammar

Language Box 1

Language Box 2

La

Host Language Grammar



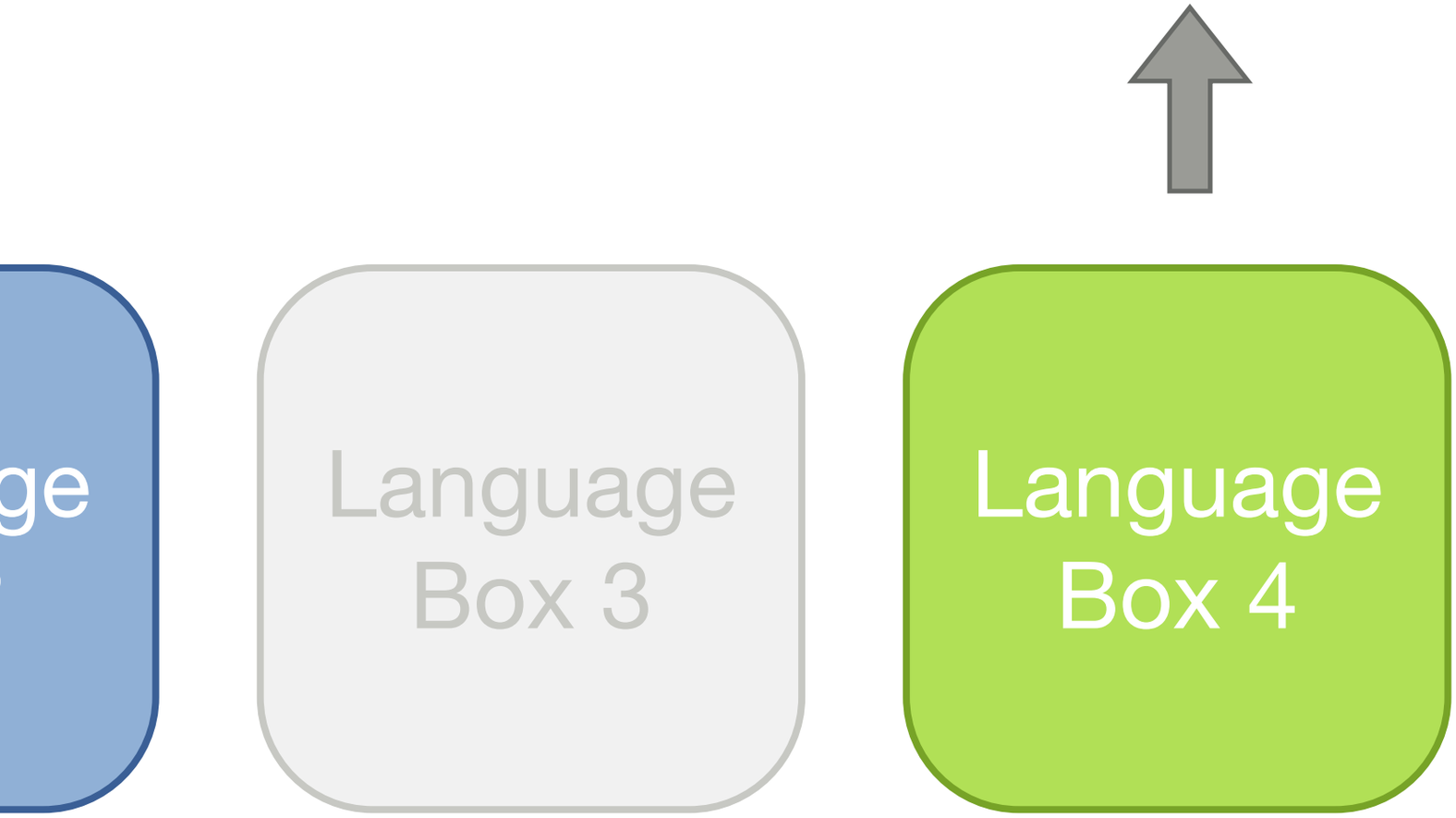
Language Box 1

Language Box 2

Language Box 3

La

Host Language Grammar



Custom Host Language Grammar

- Fine-grained language changes
- Fine-grained language scoping
- Composable and reusable model
- Tool integration (Editor, Debugger)

Language Boxes

Helvetia

Host Language

Language Boxes

Renggli *et al.*
SLE 2009

Helvetia

Host Language

Language Boxes

Helvetia

Dynamic Grammars

Host Language

grammar

a set of rules governing what strings are valid or allowable in a [formal] language.

[Adapted from *The New Oxford American Dictionary on Grammar*]

dynamic grammar

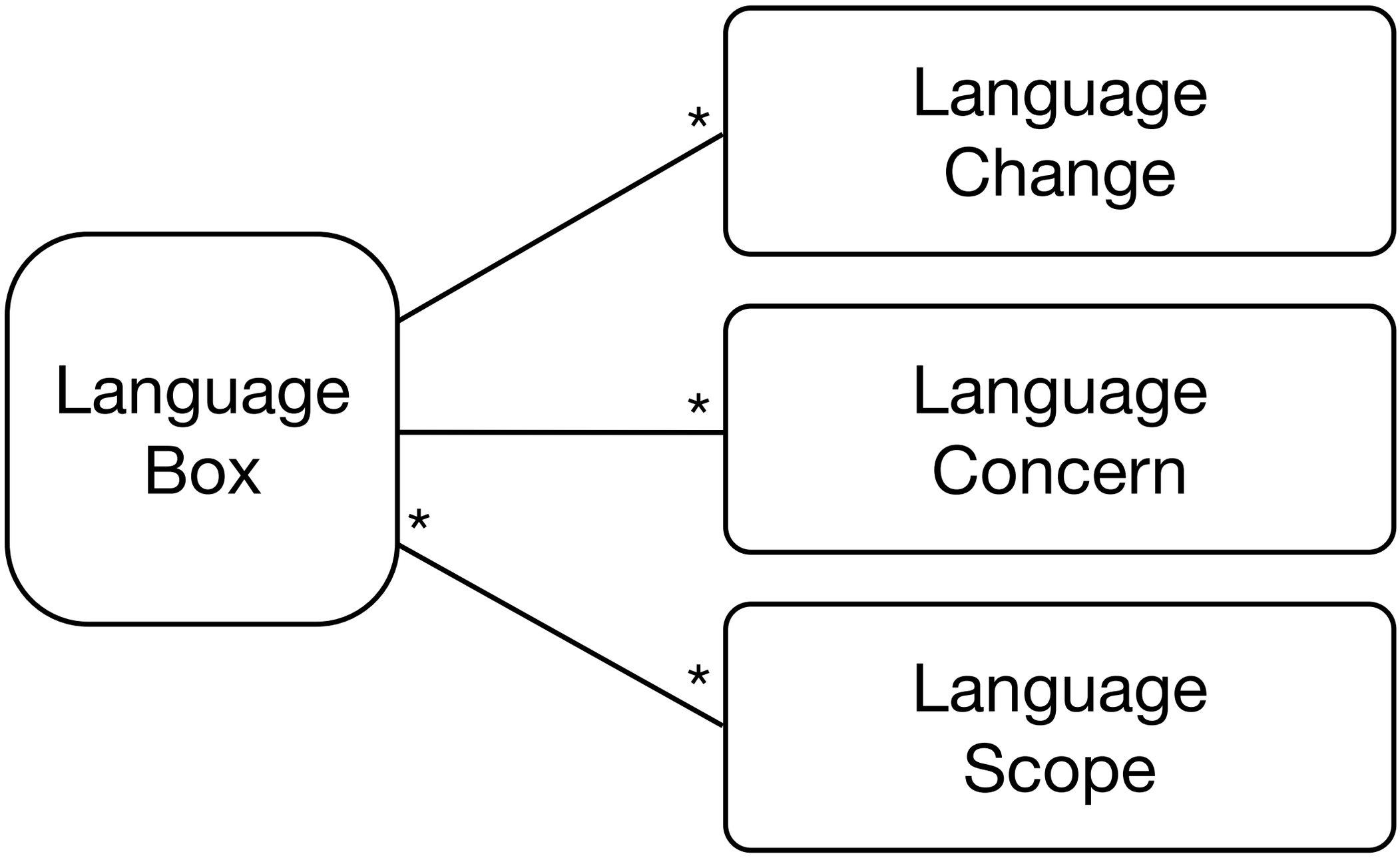
a high-level grammar that executes at runtime behaviors that other grammars perform during compilation, see *dynamic languages*.

[Adapted from *Wikipedia on Dynamic Programming Languages*]

Dynamic Grammars
Dynamic Languages

- Late-bound behavior
- First-class representation
- On-the-fly transformation
- Introspection and reflection

Why would we want that?


Language Box
*
Language Change
*
Language Concern
*
Language Scope

Language Change

=

δ

Host Language Grammar

Language Change

=

Grammar Transformation

```
scanIdentifier
   self step.
   ((currentCharacter between: $A and: $Z) or:
[ currentCharacter between: $a and: $z ]) ifTrue: [
      [ self recordMatch: #IDENTIFIER.
      self step.
      (currentCharacter between: $0 and: $9)
or: [ (currentCharacter between: $A and: $Z) or:
[ currentCharacter between: $a and: $z ] ] ]
            whileTrue.
       ^ self reportLastMatch ]
```

```
scanIdentifier
    self step.
    ((currentC                    tw                    $Z) or:
[ currentCha                                              $z ]) ifTrue: [
        [ self recordI                                    u.
        self step.
        (currentChara                                     and: $9)
    or: [ (currentC                                       and: $Z) or:
    [ currentCh                                           z ] ] ]
            whileT
        ^ self repor                Match ]
```

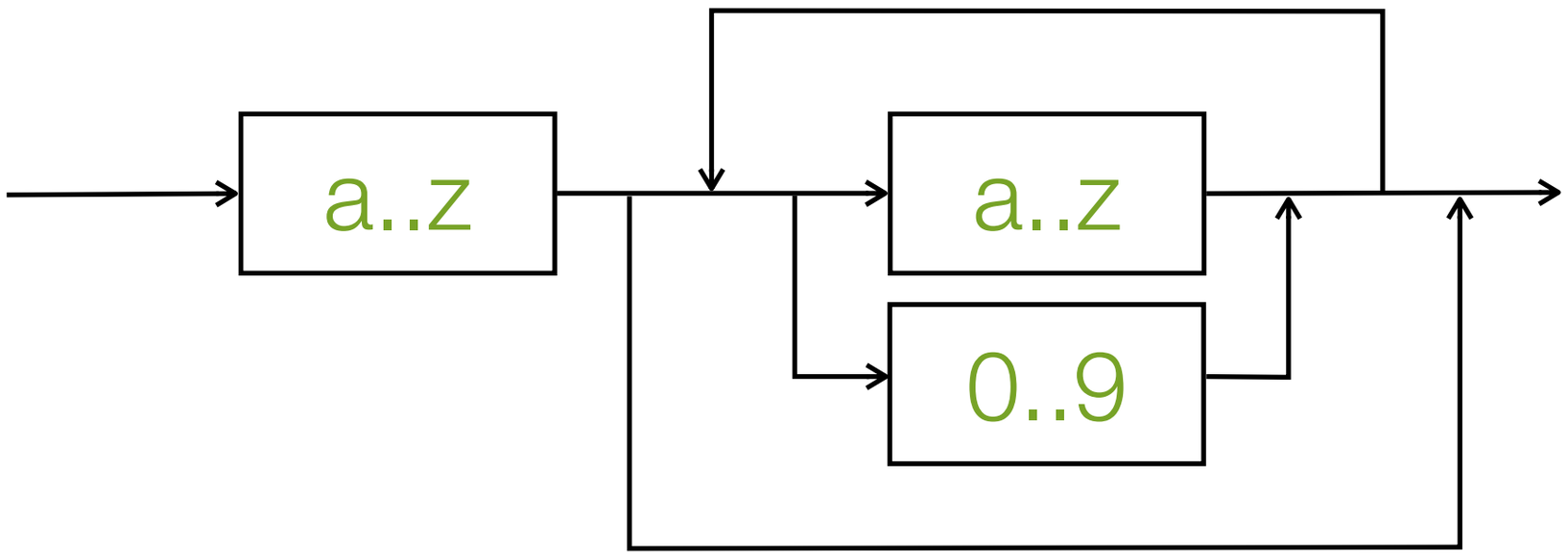
```
#(#[1 0 9 0 25 0 13 0 34 0 17 0 40 0 21 0 41]
  #[1 0 9 0 25 0 13 0 34 0 93 0 76 0 157 0 112]
  #[1 2 38 0 21 2 38 0 25 2 38 0 26 0 13 0 34]
  #[0 1 154 0 16 0 21 0 25 0 26 0 34 0 40 0 41]
  #[0 1 210 0 76 0 81]
  #[0 1 214 0 76 0 81]
  #[1 0 173 0 76 0 177 0 81]
  #[0 1 134 0 16 0 21 0 25 0 26 0 34 0 40 0 41]
  #[1 1 46 0 21 1 46 0 25 1 46 0 26 1 69]
  #[1 1 54 0 21 1 54 0 25 1 54 0 26 1 54 0 34]
  #[0 2 102 0 21 0 25 0 26 0 34 0 40 0 41 0 76]
  #[0 2 50 0 21 0 25 0 26 0 76 0 79]
  #[1 1 13 0 76 2 85 0 124 1 21 0 125]
  #[1 2 89 0 17 2 30 0 21 2 30 0 82]
  #[1 2 93 0 21 2 97 0 82])
```

```
#(#[1 0 9 0 25 0 13 0 34 0 17 0 40 0 21 0 41]
  #[1 0 9 0 25 0 13 0 34 0 93 0 76 0 157 0 112]
  #[1 2 38 0 21 [redacted] 0 25 2 38 [redacted] 6 0 13 0 34]
  #[0 1 154 0 [redacted] 0 25 0 [redacted] 0 40 0 41]
  #[0 1 210 [redacted]
  #[0 1 214 [redacted]
  #[1 0 173 0 [redacted]
  #[0 1 134 0 16 [redacted] 0 34 0 40 0 41]
  #[1 1 46 0 21 [redacted] 26 1 69]
  #[1 1 54 0 2[redacted] 1 54 0 34]
  #[0 2 102 [redacted] 41 0 76]
  #[0 2 50 0 [redacted] 26 0 [redacted]
  #[1 1 13 0 7[redacted] 0 124 1 [redacted] 25]
  #[1 2 89 0 17 [redacted] 0 21 2 30 0 [redacted] 2]
  #[1 2 93 0 21 2 97 0 82])
```

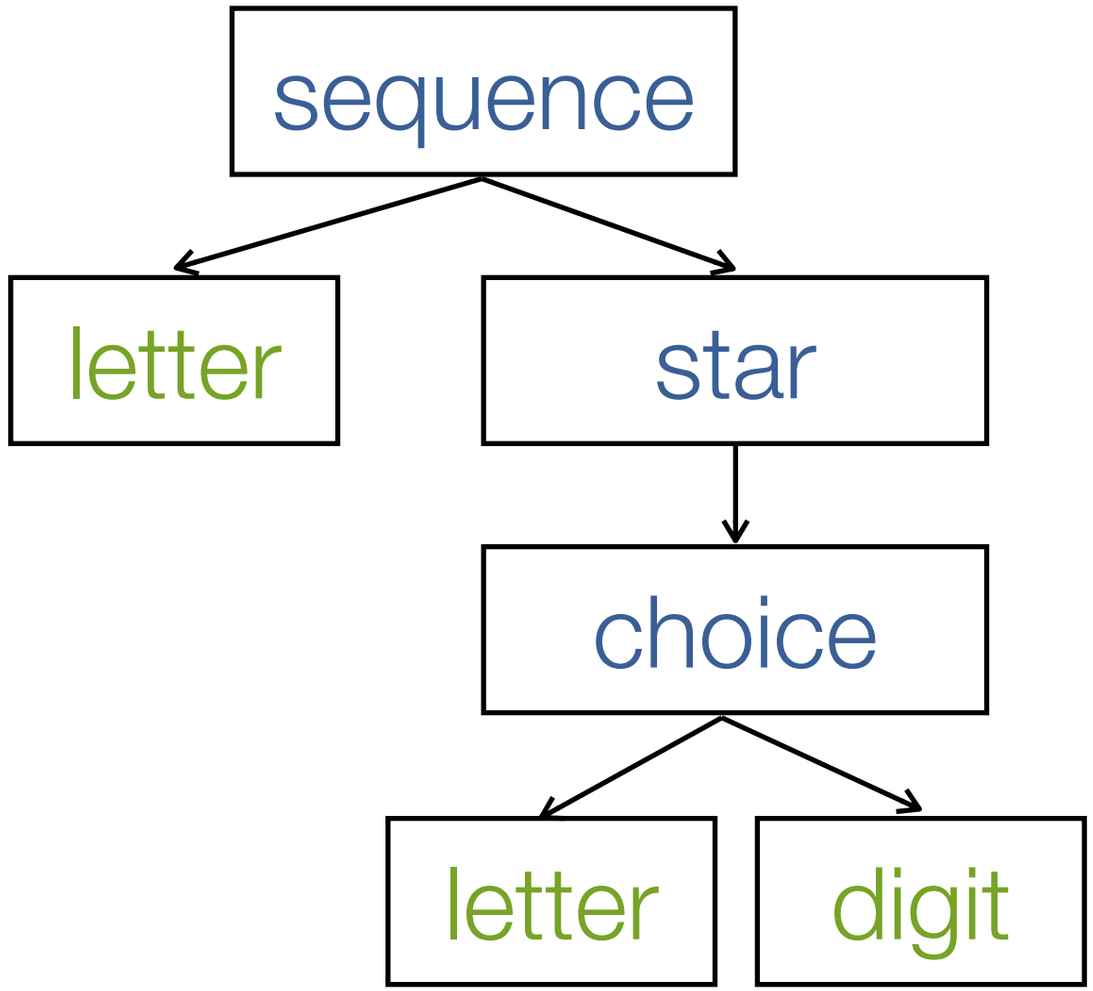
PetitParser

Dynamic Grammars in Smalltalk

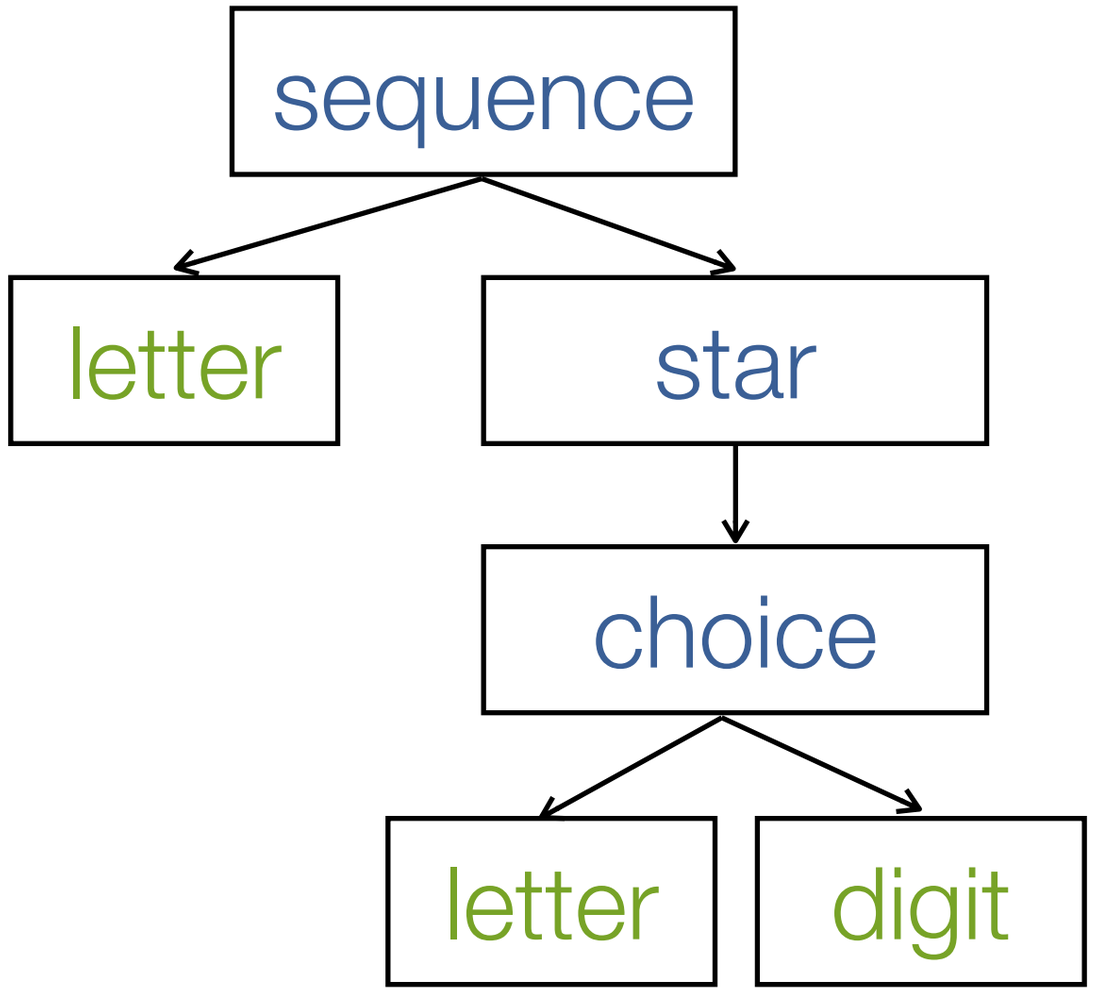
ID ::= letter { letter | digit } ;



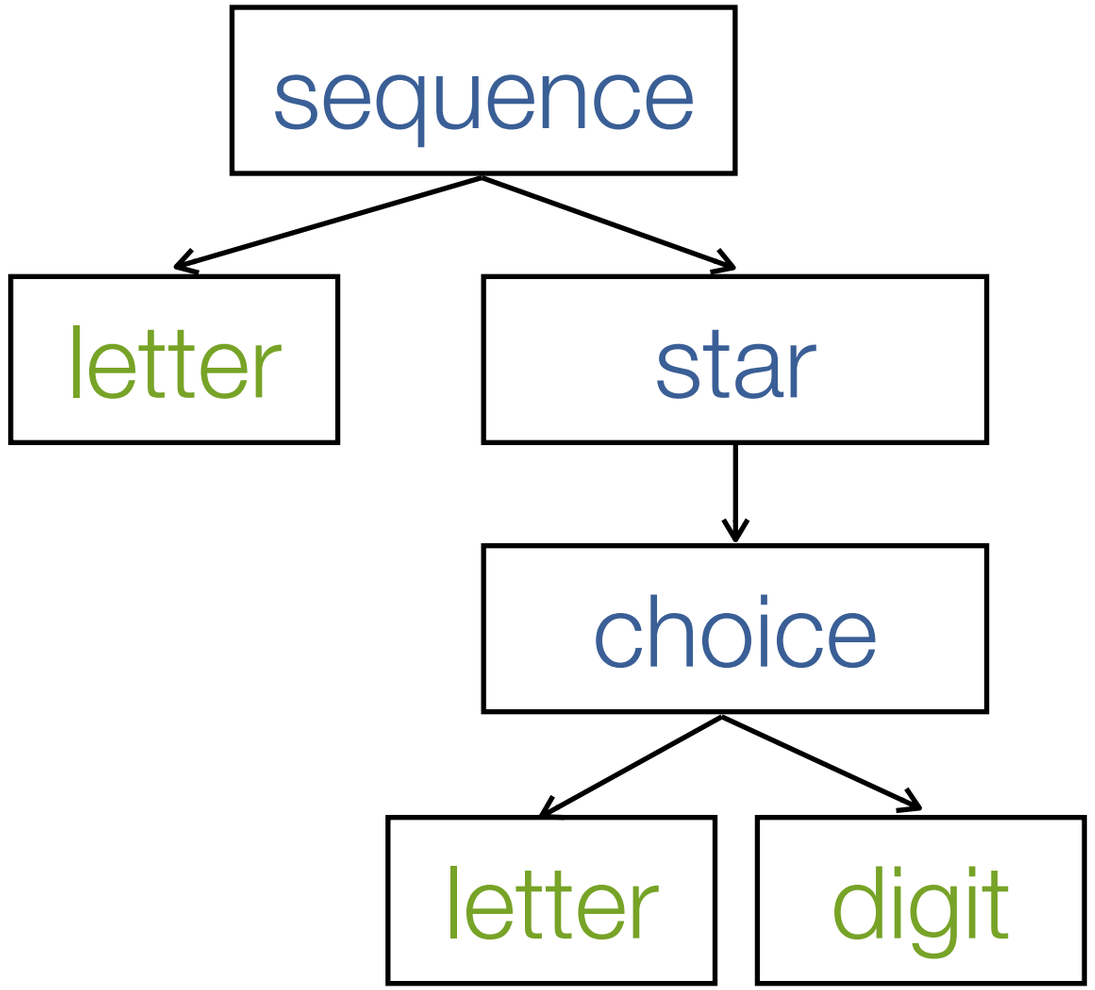
```
id := #letter , (#letter / #digit) star
```



```
id := #letter asParser ,
(#letter asParser / #digit asParser) star
```



ID ::= letter { letter | digit } ;

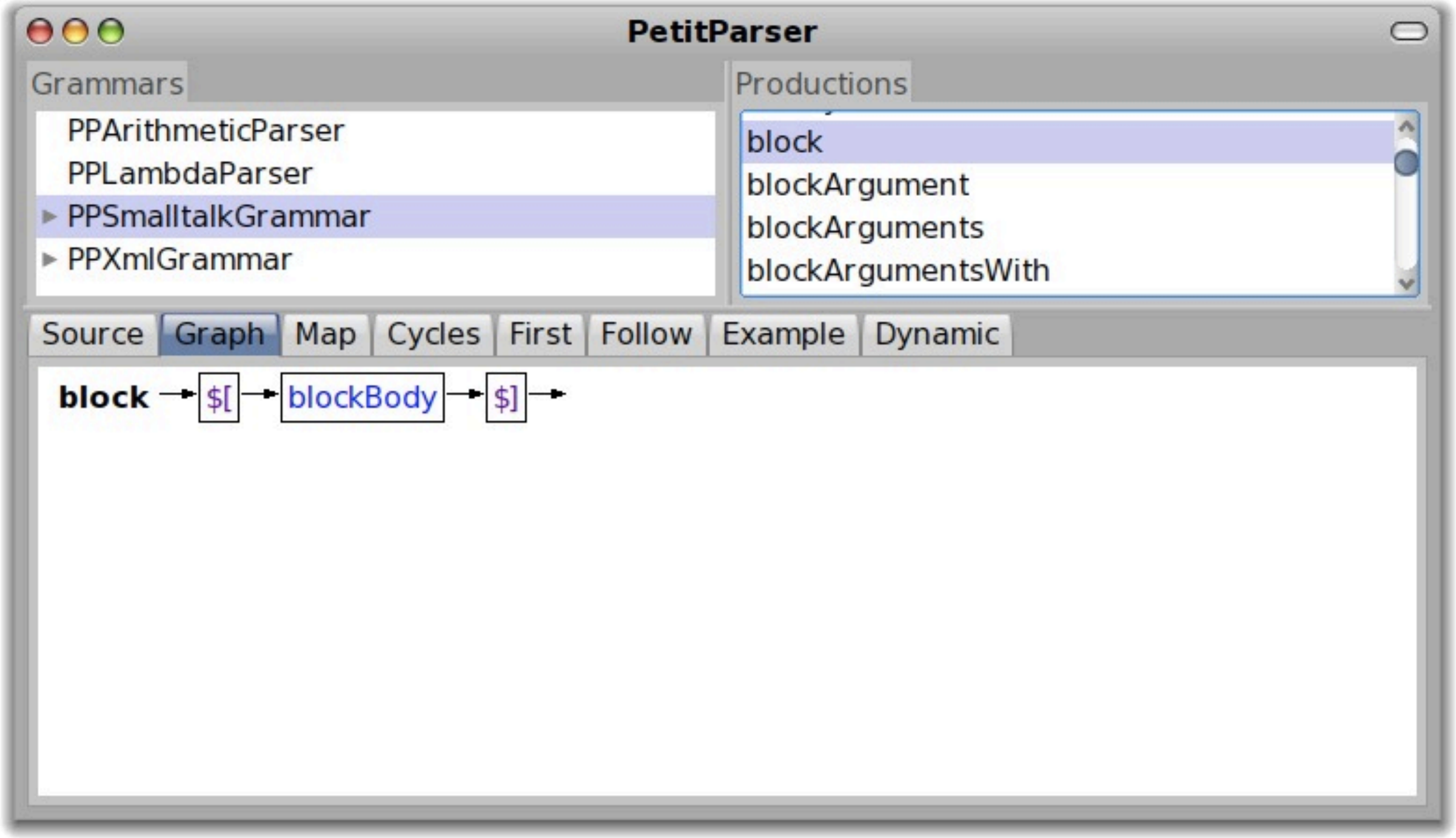


Scannerless Parser

Parsing Expression Grammar

Packrat Parser

Parser Combinator

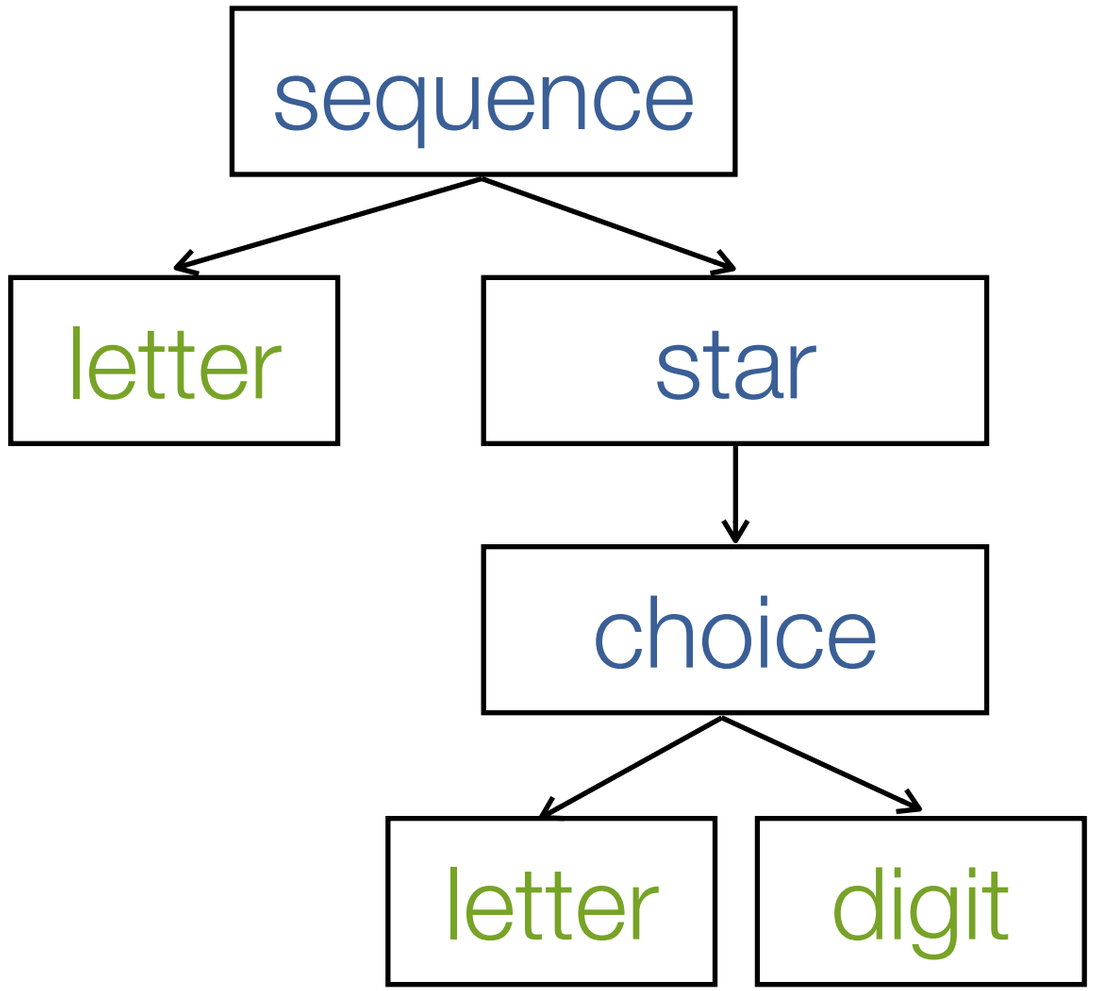

PetitParser
Grammars
PPArithmeticParser
PPLambdaParser
PPSmalltalkGrammar
PPXmlGrammar
Productions
block
blockArgument
blockArguments
blockArgumentsWith
Source
Graph
Map
Cycles
First
Follow
Example
Dynamic
block
$[
blockBody
$]

Grammar

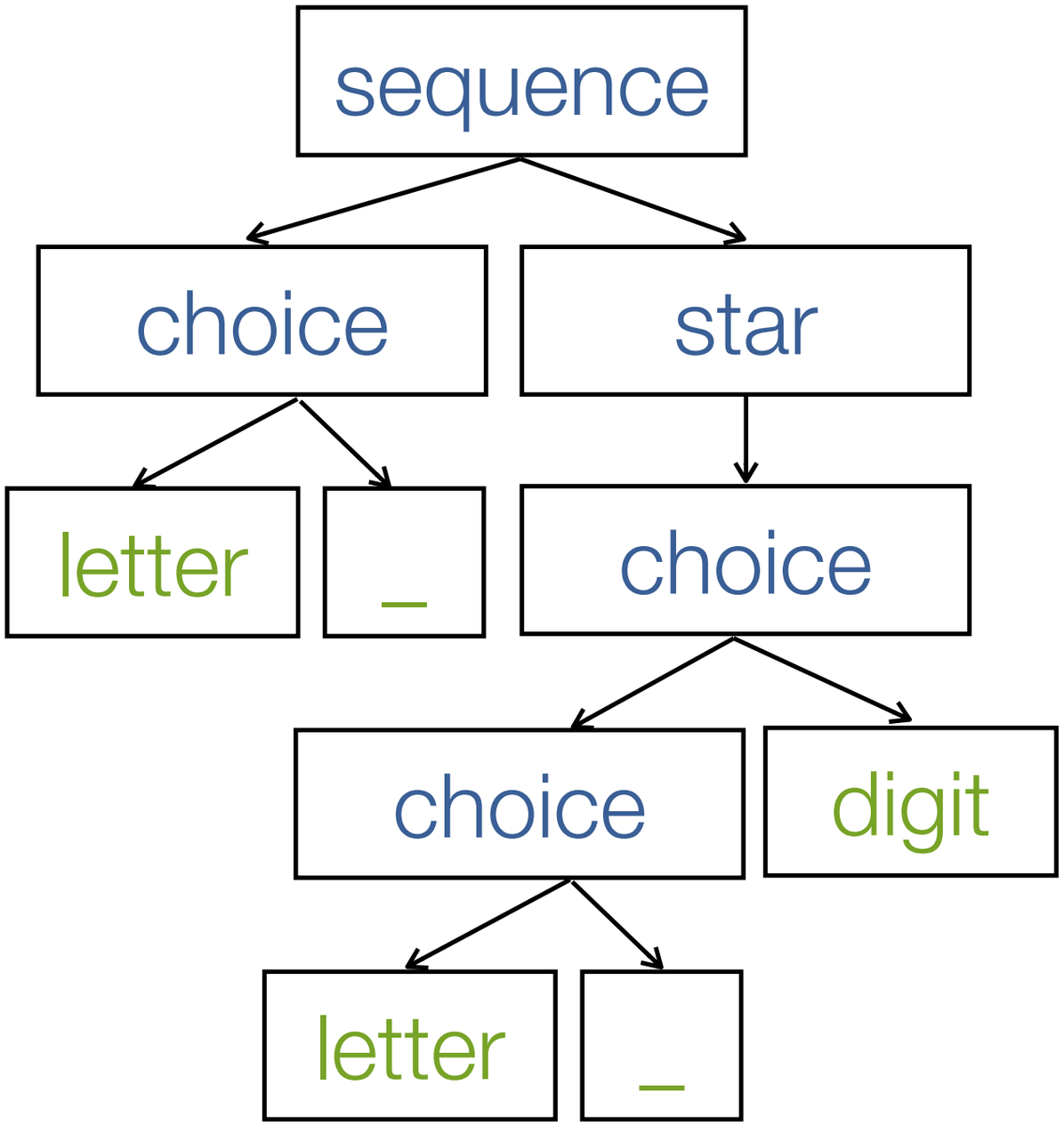
Transformation

Graph Rewriting

#letter → #letter / $_

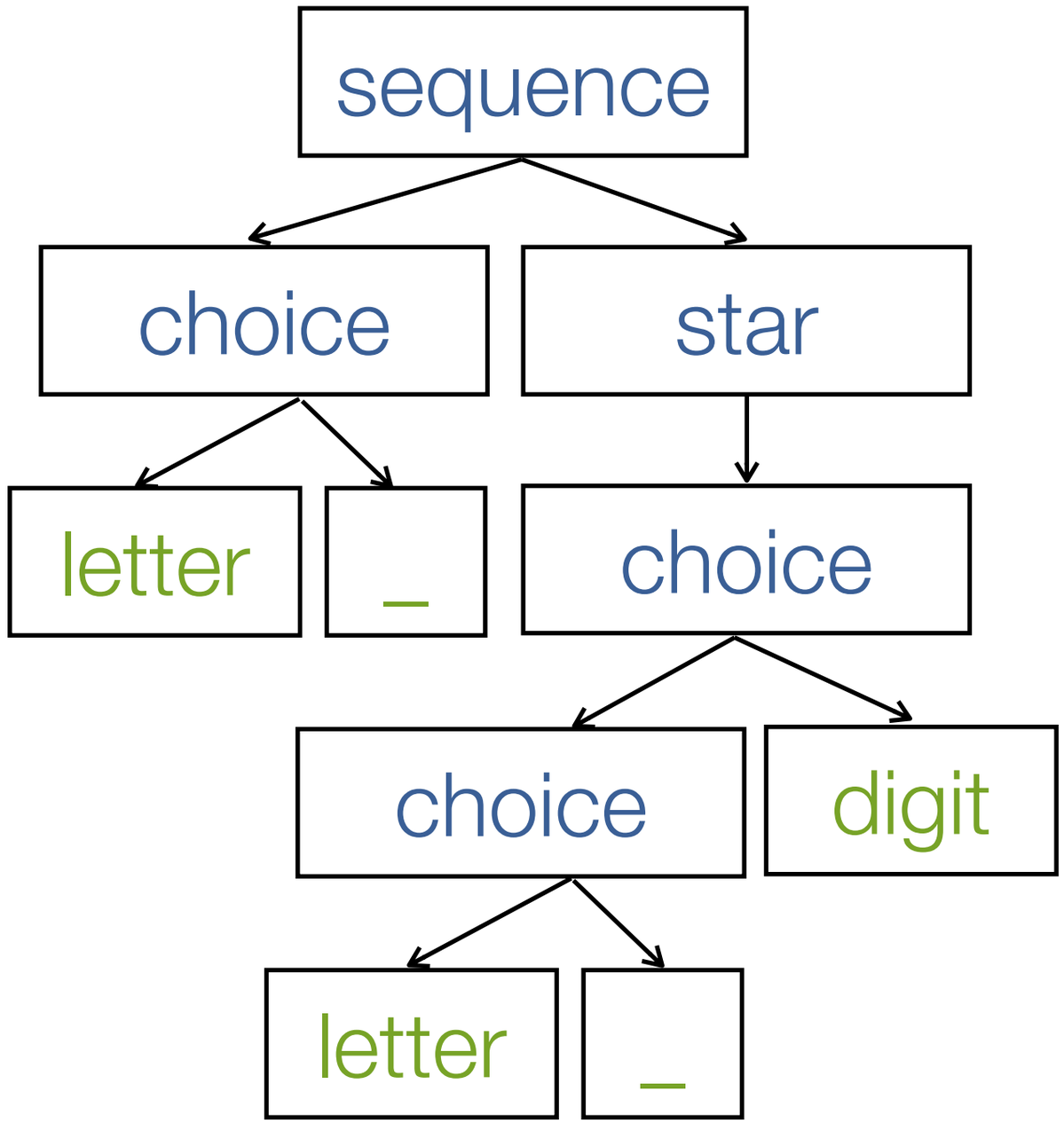


#letter → #letter / $_

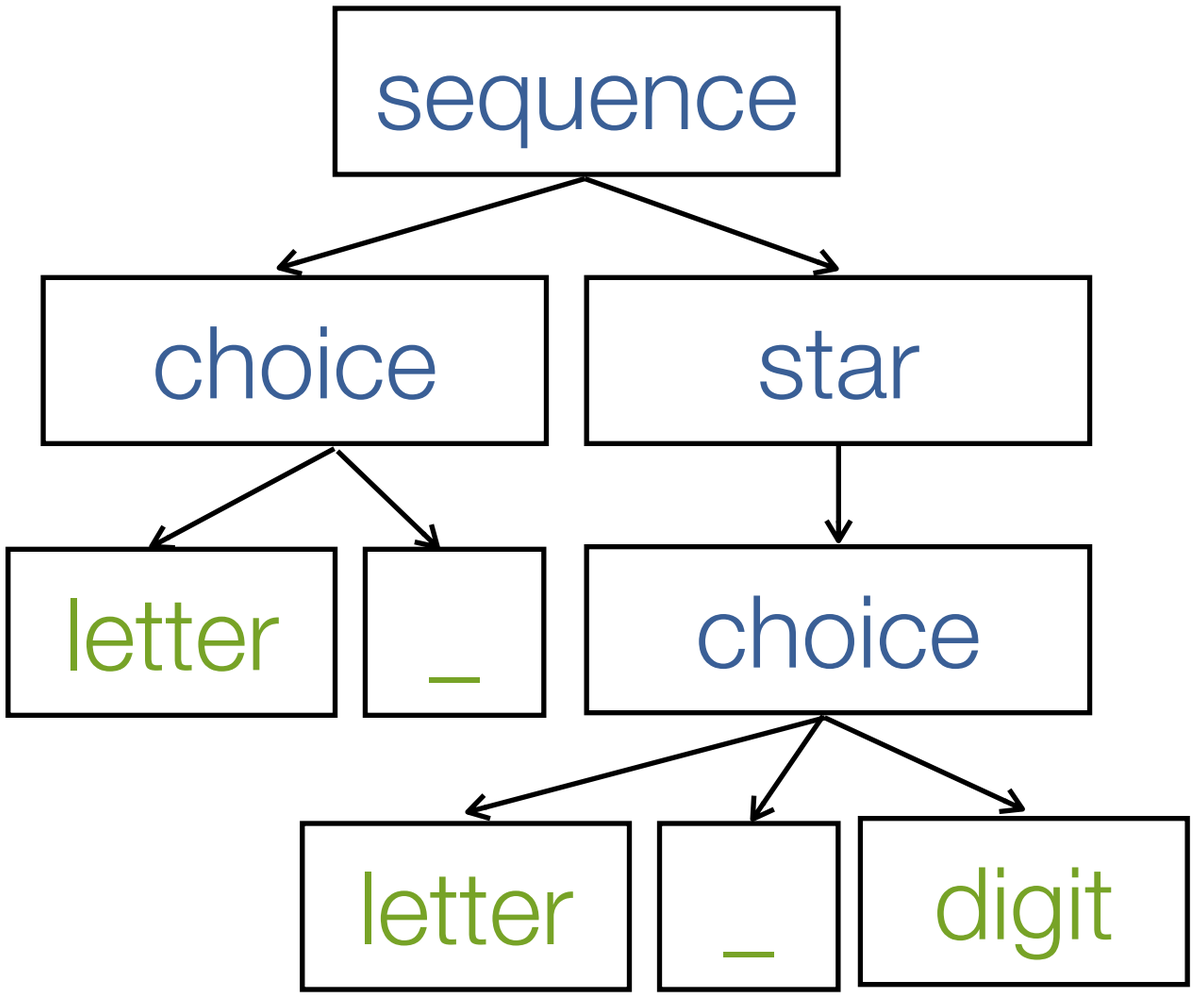


Optimizations

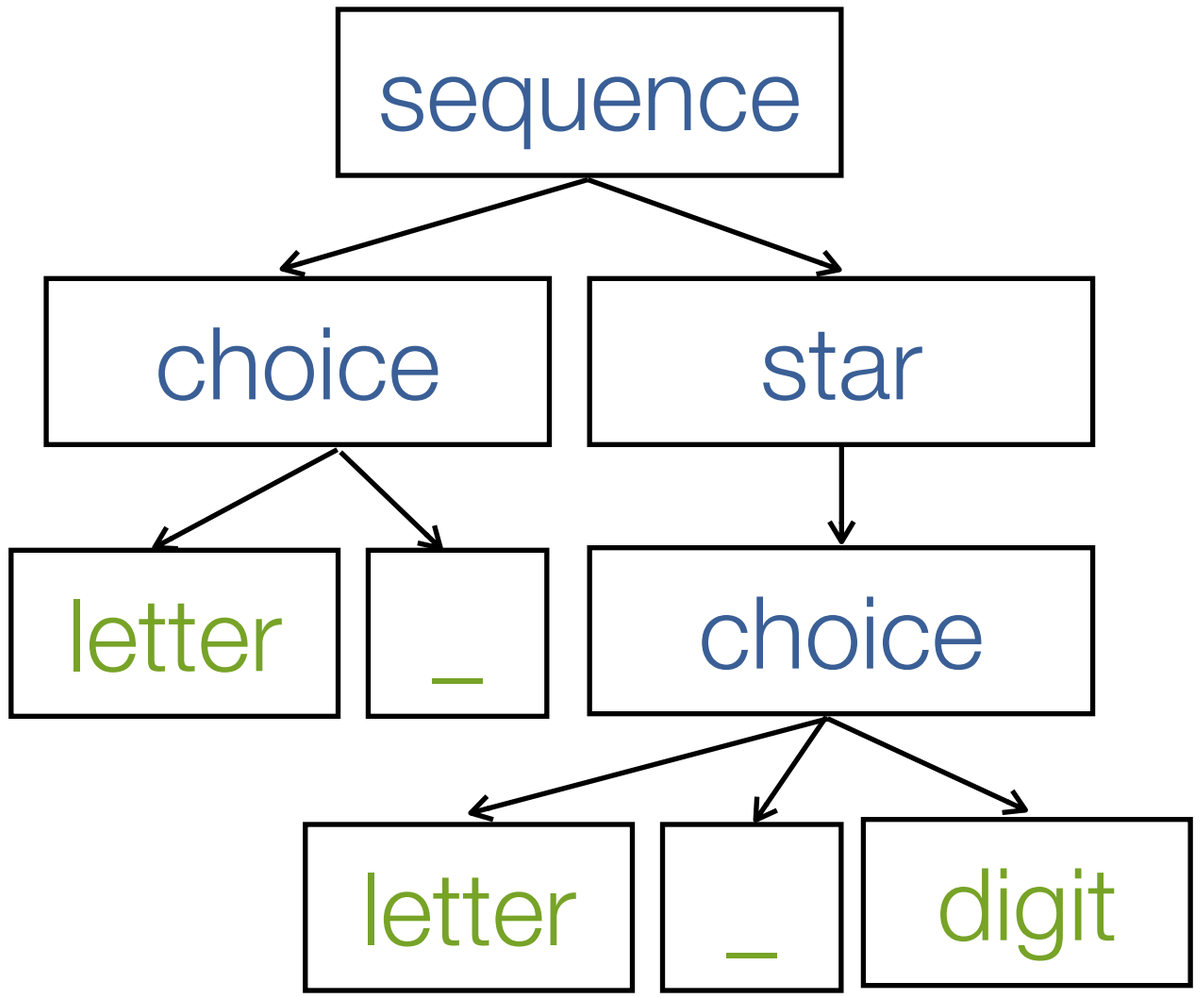
(`a/`b)/`c → `a/`b/`c



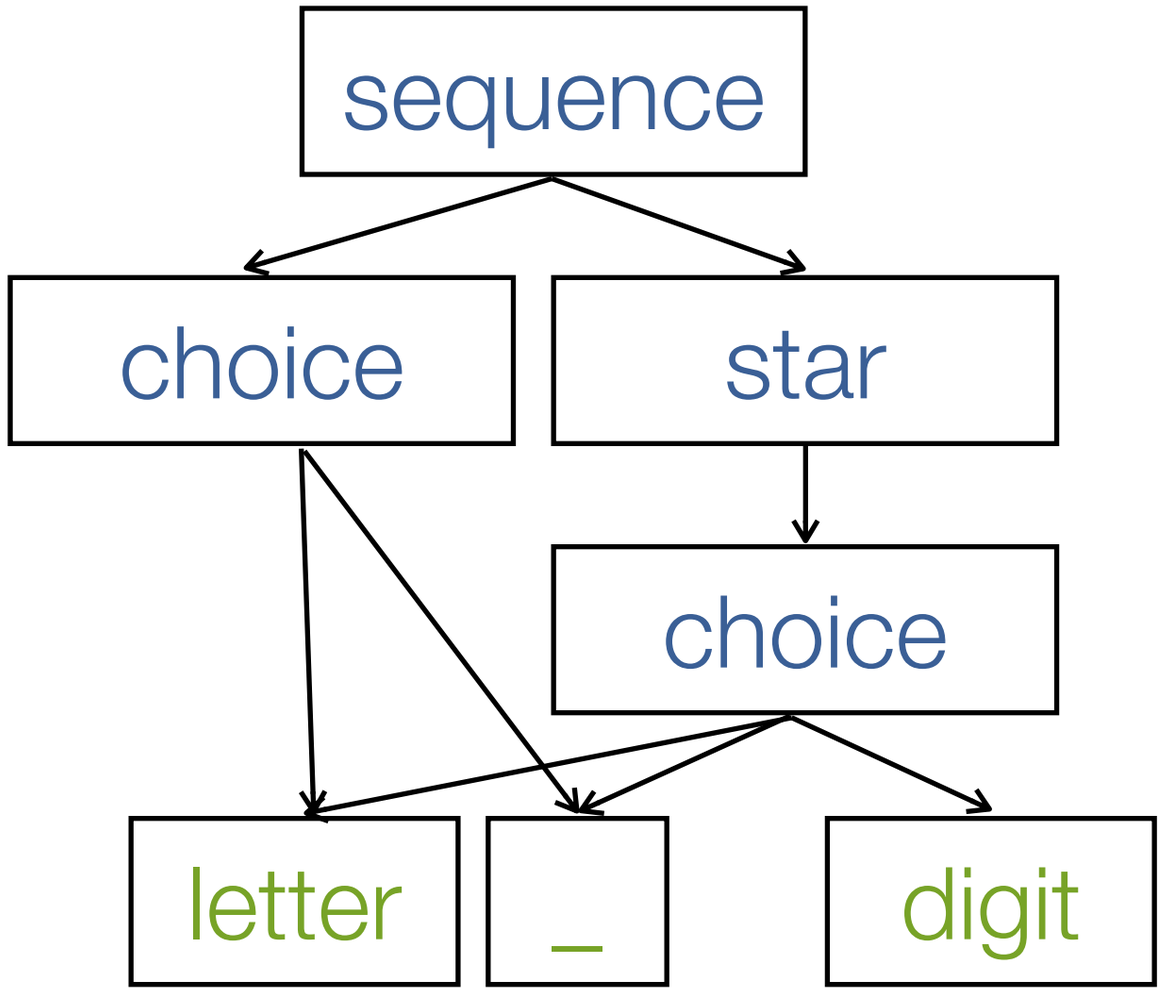
(`a/`b)/`c → `a/`b/`c



remove duplicates

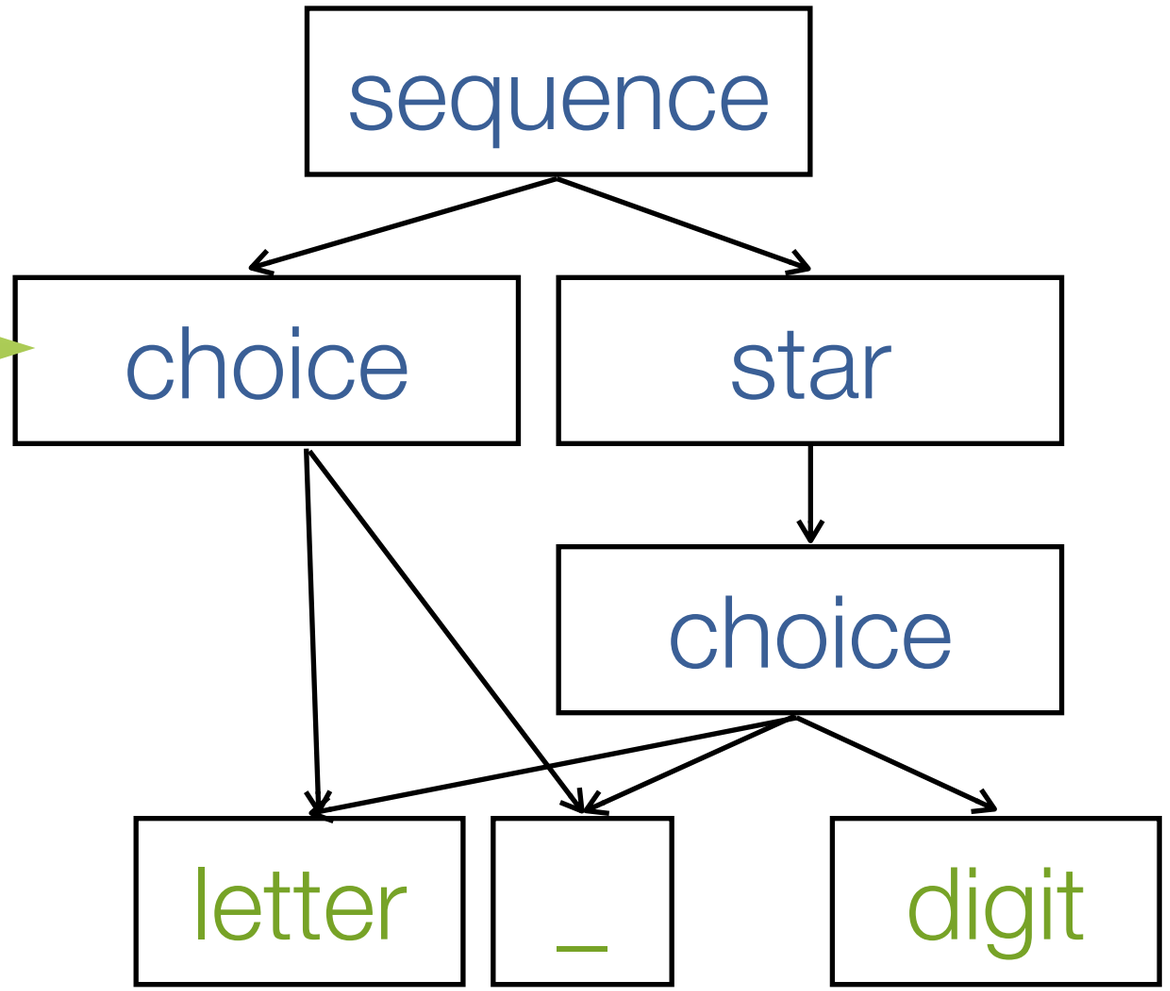


remove duplicates



remove duplicates

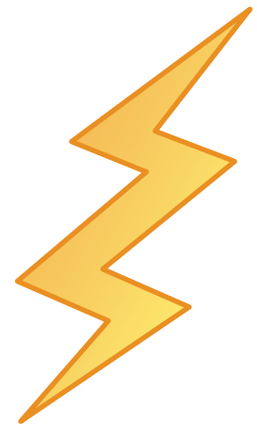
in general
a graph



Conflicts?

```
SELECT * FROM users
```

```
| r |
r := SELECT * FROM users.
^ User fromRow: r
```



```
expr | sql
```

<SQL: SELECT * FROM users>

ordered

expr / sql

No Conflicts

No Ambiguities

surprise

expr / sql

surprise

sql / expr

!expr sql / !sql expr

expr | sql

!expr sql / !sql expr / ui

```
expr $ sql
```

Language Boxes

Helvetia

Dynamic Grammars

Host Language

Language Boxes

Helvetia

Host Language

Renggli et al.
DYLA 2010

Dynamic
Grammars

Embedded Languages

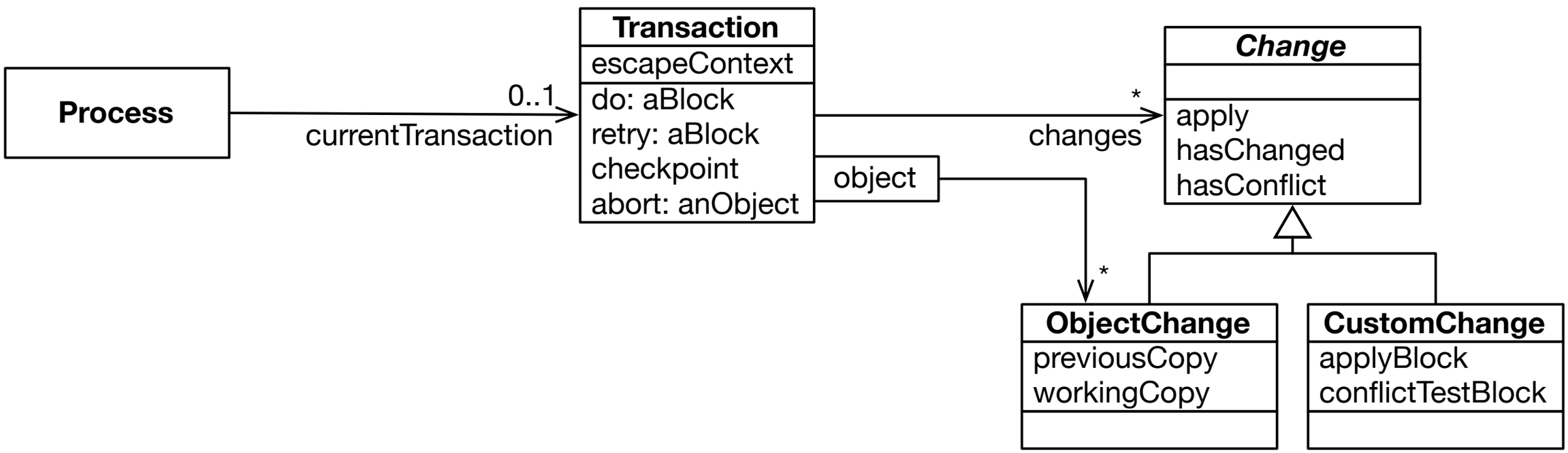
Language Boxes

Helvetia

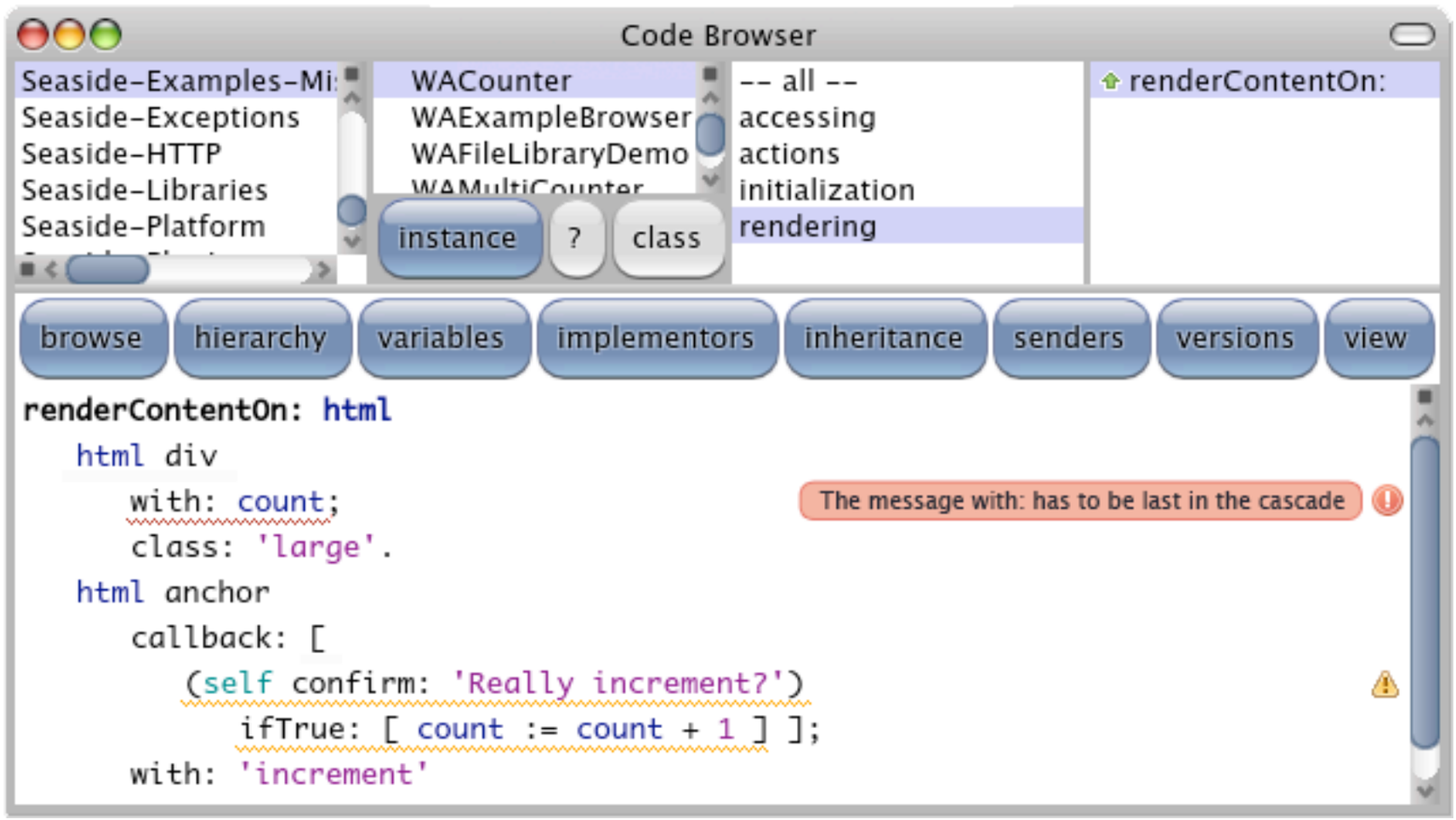
Dynamic
Grammars

Host Language

Transactional Memory



Domain-Specific Program Checking



Meta-Programming Facilities

```
``(`,(aString) asRegex)
```

The LISP stuff,
for a cool language

Renggli et al.
CLSS 2009

Renggli et al.
IWST 2009

Nierstrasz et al.
LNCS 2009

Renggli et al.
TOOLS 2010

Language Extensions

Language Boxes

Helvetia

Dynamic
Grammars

Host Language

Helvetia

scg.unibe.ch/research/helvetia