Guided Grammar Convergence

Full Case Study Report

Generated by `converge::Guided`

Vadim Zaytsev, vadim@grammarware.net,
Software Analysis & Transformation Team (SWAT),
Centrum Wiskunde & Informatica (CWI), The Netherlands

July 30, 2012

Introduction

This report is meant to be used as auxiliary material for the *guided grammar convergence* technique proposed in [18] as problem-specific improvement on [12]. It contains a megamodel renarrated as proposed in [19], as well as full results of the guided grammar convergence experiment on the Factorial Language, with details about each grammar source packaged in a readable form. All formulae used within this document, are generated automatically by the convergence infrastructure in order to avoid any mistakes. The generator source code and the source of the introduction text can be found publicly available in the Software Language Processing Suite repository [21].

Consider the model on Figure 1. It is a *megamodel* in the sense of [1, 6], since it depicts a *linguistic architecture*: all nodes represent software languages and language transformations, and all edges represent relationships between them. MegaL [5] is used as a notation: blue boxes represent tangible *artefacts* (files, programs, modules, directories, collections of other concrete entities), yellow boxes denote software *languages* in the broad sense (from general purpose

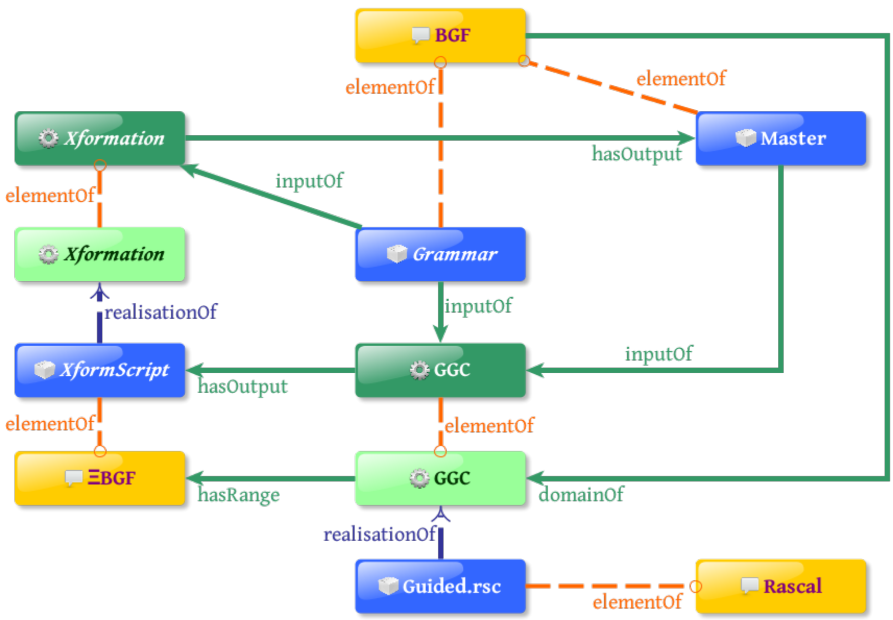

Figure 1: Guided grammar convergence megamodel.

programming languages to data types and protocols), light green boxes are used for *functions* (in fact, model transformations) and dark green boxes are for *function applications*.

As we can see from Figure 1 if we start reading it from the bottom, there is a program Guided.rsc, which was written in Rascal metaprogramming language [11]. It implements the guided grammar convergence process, which input language is BGF (BNF-like Grammar Formalism, a straightforward internal representation format for grammars, introduced in [12]). Its output language is ΞBGF, a bidirectional grammar transformation language introduced in [20]. An application of the guided grammar convergence algorithm to two grammars: one *master grammar* defining the *intended* software language (terminology of [18]) and one servant grammar (its label displayed in italics since it is actually a variable, not a concrete entity) — yields a transformation script that implements a grammar transformation than indeed transforms the servant grammar into the master grammar. The process behind this inference is relatively complicated and involves triggered grammar design mutations, normalisation to Abstract Normal Form, constructing weak prodsig-equivalence ($\Leftrightarrow$) classes and resolving nominal and structural differences, as described on the theoretic level in [18].

The rest of the report presents instantiations of this megamodel for eleven concrete grammar sources:

adt: an algebraic data type[1] in Rascal [10];

antlr: a parser description in the input language of ANTLR [15]. Semantic actions (in Java) are intertwined with EBNF-like productions.

dcg: a logic program written in the style of definite clause grammars [16].

emf: an Ecore model [14], automatically generated by Eclipse [3] from the XML Schema of the **xsd** source;

jaxb: an object model obtained by a data binding framework. Generated automatically by JAXB [7] from the XML schema for FL.

om: a hand-crafted object model (Java classes) for the abstract syntax of FL.

python: a parser specification in a scripting language, using the PyParsing library [13];

rascal: a concrete syntax specification in Rascal metaprogramming language [10, 11];

sdf: a concrete syntax definition in the notation of SDF [9] with scannerless generalized LR [4, 17] as a parsing model.

txl: a concrete syntax definition in the notation of TXL (Turing eXtender Language) transformational framework [2], which, unlike SDF, uses a combination of pattern matching and term rewriting).

xsd: an XML schema [8] for the abstract syntax of FL.

[1] `http://tutor.rascal-mpl.org/Courses/Rascal/Declarations/AlgebraicDataType/AlgebraicDataType.html`.

Contents

Grammar 1

ANTLR

Source name: **antlr**

1.1 Source grammar

- Source artifact: topics/fl/java1/FL.g
- Grammar extractor: topics/extraction/antlr/antlrstrip.py
- Grammar extractor: topics/extraction/antlr/slps/antlr2bgf/StrippedANTLR.g

Production rules
p(‘’, *program*, +(sel (‘f’, *function*)))
p(‘’, *function*, seq ([sel (‘n’, *ID*) , +(sel (‘a’, *ID*)) , ‘=’, sel (‘e’, *expr*) , +(*NEWLINE*)]))
p(‘’, *expr*, choice([sel (‘b’, *binary*) , sel (‘a’, *apply*) , sel (‘i’, *ifThenElse*)]))
p(‘’, *binary*, seq ([sel (‘l’, *atom*) , ∗(seq ([sel (‘o’, *ops*) , sel (‘r’, *atom*)]))]))
p(‘’, *apply*, seq ([sel (‘i’, *ID*) , +(sel (‘a’, *atom*))]))
p(‘’, *ifThenElse*, seq ([‘if’, sel (‘c’, *expr*) , ‘then’, sel (‘e1’, *expr*) , ‘else’, sel (‘e2’, *expr*)]))
p(‘’, *atom*, choice([*ID*, *INT*, seq ([‘(’, sel (‘e’, *expr*) , ‘)’])]))
p(‘’, *ops*, choice([‘==’, ‘+’, ‘-’]))

1.2 Mutations

- **unite-splitN** *expr*
  p (‘’, *atom*, choice ([*ID*, *INT*, seq ([‘(’, sel (‘e’, *expr*) , ‘)’])]))
- **designate-unlabel**
  p (‘tmplabel’, *binary*, seq ([sel (‘l’, *expr*) , ∗(seq ([sel (‘o’, *ops*) , sel (‘r’, *expr*)]))]))
- **anonymize-deanonymize**
  p (‘tmplabel’, *binary*, seq ([sel (‘l’, *expr*), ∗(seq ([sel (‘o’, *ops*), sel (‘r’, *expr*)]))]))
- **assoc-iterate**
  p (‘tmplabel’, *binary*, seq ([*expr*, *ops*, *expr*]))
- **deanonymize-anonymize**
  p (‘tmplabel’, *binary*, seq ([sel (‘l’, *expr*), sel (‘o’, *ops*), sel (‘r’, *expr*)]))
- **unlabel-designate**
  p (‘tmplabel’, *binary*, seq ([sel (‘l’, *expr*) , sel (‘o’, *ops*) , sel (‘r’, *expr*)]))

1.3 Normalizations

- **reroot-reroot** [] to [*program*]
- **anonymize-deanonymize**
  $\mathrm{p}\left(\text{‘’}, \mathit{function}, \mathrm{seq}\left(\left[\boxed{\mathrm{sel}\,(\text{‘n’}, \mathit{ID})}, +\left(\boxed{\mathrm{sel}\,(\text{‘a’}, \mathit{ID})}\right), \text{‘=’}, \boxed{\mathrm{sel}\,(\text{‘e’}, \mathit{expr})}, +(\mathit{NEWLINE})\right]\right)\right)$
- **anonymize-deanonymize**
  $\mathrm{p}\left(\text{‘’}, \mathit{program}, +\left(\boxed{\mathrm{sel}\,(\text{‘f’}, \mathit{function})}\right)\right)$
- **anonymize-deanonymize**
  $\mathrm{p}\left(\text{‘’}, \mathit{ifThenElse}, \mathrm{seq}\left(\left[\text{‘if’}, \boxed{\mathrm{sel}\,(\text{‘c’}, \mathit{expr})}, \text{‘then’}, \boxed{\mathrm{sel}\,(\text{‘e1’}, \mathit{expr})}, \text{‘else’}, \boxed{\mathrm{sel}\,(\text{‘e2’}, \mathit{expr})}\right]\right)\right)$
- **anonymize-deanonymize**
  $\mathrm{p}\left(\text{‘’}, \mathit{binary}, \mathrm{seq}\left(\left[\boxed{\mathrm{sel}\,(\text{‘l’}, \mathit{expr})}, \boxed{\mathrm{sel}\,(\text{‘o’}, \mathit{ops})}, \boxed{\mathrm{sel}\,(\text{‘r’}, \mathit{expr})}\right]\right)\right)$
- **anonymize-deanonymize**
  $\mathrm{p}\left(\text{‘’}, \mathit{expr}, \mathrm{choice}\left(\left[\mathit{ID}, \mathit{INT}, \mathrm{seq}\left(\left[\text{‘(’}, \boxed{\mathrm{sel}\,(\text{‘e’}, \mathit{expr})}, \text{‘)’}\right]\right)\right]\right)\right)$
- **anonymize-deanonymize**
  $\mathrm{p}\left(\text{‘’}, \mathit{apply}, \mathrm{seq}\left(\left[\boxed{\mathrm{sel}\,(\text{‘i’}, \mathit{ID})}, +\left(\boxed{\mathrm{sel}\,(\text{‘a’}, \mathit{expr})}\right)\right]\right)\right)$
- **anonymize-deanonymize**
  $\mathrm{p}\left(\text{‘’}, \mathit{expr}, \mathrm{choice}\left(\left[\boxed{\mathrm{sel}\,(\text{‘b’}, \mathit{binary})}, \boxed{\mathrm{sel}\,(\text{‘a’}, \mathit{apply})}, \boxed{\mathrm{sel}\,(\text{‘i’}, \mathit{ifThenElse})}\right]\right)\right)$
- **abstractize-concretize**
  $\mathrm{p}\left(\text{‘’}, \mathit{ops}, \mathrm{choice}\left(\left[\boxed{\text{‘==’}}, \boxed{\text{‘+’}}, \boxed{\text{‘-’}}\right]\right)\right)$
- **abstractize-concretize**
  $\mathrm{p}\left(\text{‘’}, \mathit{expr}, \mathrm{choice}\left(\left[\mathit{ID}, \mathit{INT}, \mathrm{seq}\left(\left[\boxed{\text{‘(’}}, \mathit{expr}, \boxed{\text{‘)’}}\right]\right)\right]\right)\right)$
- **abstractize-concretize**
  $\mathrm{p}\left(\text{‘’}, \mathit{function}, \mathrm{seq}\left(\left[\mathit{ID}, +(\mathit{ID}), \boxed{\text{‘=’}}, \mathit{expr}, +(\mathit{NEWLINE})\right]\right)\right)$
- **abstractize-concretize**
  $\mathrm{p}\left(\text{‘’}, \mathit{ifThenElse}, \mathrm{seq}\left(\left[\boxed{\text{‘if’}}, \mathit{expr}, \boxed{\text{‘then’}}, \mathit{expr}, \boxed{\text{‘else’}}, \mathit{expr}\right]\right)\right)$
- **vertical-horizontal** in *expr*
- **undefine-define**
  $\mathrm{p}\,(\text{‘’}, \mathit{ops}, \varepsilon)$
- **unchain-chain**
  $\mathrm{p}\,(\text{‘’}, \mathit{expr}, \mathit{binary})$
- **unchain-chain**
  $\mathrm{p}\,(\text{‘’}, \mathit{expr}, \mathit{apply})$
- **unchain-chain**
  $\mathrm{p}\,(\text{‘’}, \mathit{expr}, \mathit{ifThenElse})$
- **abridge-detour**
  $\mathrm{p}\,(\text{‘’}, \mathit{expr}, \mathit{expr})$
- **unlabel-designate**
  $\mathrm{p}\left(\boxed{\text{‘binary’}}, \mathit{expr}, \mathrm{seq}\,([\mathit{expr}, \mathit{ops}, \mathit{expr}])\right)$
- **unlabel-designate**
  $\mathrm{p}\left(\boxed{\text{‘apply’}}, \mathit{expr}, \mathrm{seq}\,([\mathit{ID}, +(\mathit{expr})])\right)$
- **unlabel-designate**
  $\mathrm{p}\left(\boxed{\text{‘ifThenElse’}}, \mathit{expr}, \mathrm{seq}\,([\mathit{expr}, \mathit{expr}, \mathit{expr}])\right)$
- **extract-inline** in *expr*
  $\mathrm{p}\,(\text{‘’}, \mathit{expr_1}, \mathrm{seq}\,([\mathit{expr}, \mathit{ops}, \mathit{expr}]))$
- **extract-inline** in *expr*
  $\mathrm{p}\,(\text{‘’}, \mathit{expr_2}, \mathrm{seq}\,([\mathit{ID}, +(\mathit{expr})]))$
- **extract-inline** in *expr*
  $\mathrm{p}\,(\text{‘’}, \mathit{expr_3}, \mathrm{seq}\,([\mathit{expr}, \mathit{expr}, \mathit{expr}]))$

1.4 Grammar in ANF

Production rule	Production signature
$\mathrm{p}\left(`', \mathit{program}, +(\mathit{function})\right)$	$\{\langle \mathit{function}, +\rangle\}$
$\mathrm{p}\left(`', \mathit{function}, \mathrm{seq}\left([\mathit{ID}, +(\mathit{ID}), \mathit{expr}, +(\mathit{NEWLINE})]\right)\right)$	$\{\langle \mathit{expr}, 1\rangle, \langle \mathit{NEWLINE}, +\rangle, \langle \mathit{ID}, 1{+}\rangle\}$
$\mathrm{p}\left(`', \mathit{expr}, \mathit{ID}\right)$	$\{\langle \mathit{ID}, 1\rangle\}$
$\mathrm{p}\left(`', \mathit{expr}, \mathit{INT}\right)$	$\{\langle \mathit{INT}, 1\rangle\}$
$\mathrm{p}\left(`', \mathit{expr}, \mathit{expr_1}\right)$	$\{\langle \mathit{expr_1}, 1\rangle\}$
$\mathrm{p}\left(`', \mathit{expr}, \mathit{expr_2}\right)$	$\{\langle \mathit{expr_2}, 1\rangle\}$
$\mathrm{p}\left(`', \mathit{expr}, \mathit{expr_3}\right)$	$\{\langle \mathit{expr_3}, 1\rangle\}$
$\mathrm{p}\left(`', \mathit{expr_1}, \mathrm{seq}\left([\mathit{expr}, \mathit{ops}, \mathit{expr}]\right)\right)$	$\{\langle \mathit{ops}, 1\rangle, \langle \mathit{expr}, 11\rangle\}$
$\mathrm{p}\left(`', \mathit{expr_2}, \mathrm{seq}\left([\mathit{ID}, +(\mathit{expr})]\right)\right)$	$\{\langle \mathit{expr}, +\rangle, \langle \mathit{ID}, 1\rangle\}$
$\mathrm{p}\left(`', \mathit{expr_3}, \mathrm{seq}\left([\mathit{expr}, \mathit{expr}, \mathit{expr}]\right)\right)$	$\{\langle \mathit{expr}, 111\rangle\}$

1.5 Nominal resolution

Production rules are matched as follows (ANF on the left, master grammar on the right):

$$
\begin{aligned}
\mathrm{p}\left(`', \mathit{program}, +(\mathit{function})\right) &\bumpeq \mathrm{p}\left(`', \mathit{program}, +(\mathit{function})\right)\\
\mathrm{p}\left(`', \mathit{function}, \mathrm{seq}\left([\mathit{ID}, +(\mathit{ID}), \mathit{expr}, +(\mathit{NEWLINE})]\right)\right) &\Bumpeq \mathrm{p}\left(`', \mathit{function}, \mathrm{seq}\left([\mathit{str}, +(\mathit{str}), \mathit{expression}]\right)\right)\\
\mathrm{p}\left(`', \mathit{expr}, \mathit{ID}\right) &\bumpeq \mathrm{p}\left(`', \mathit{expression}, \mathit{str}\right)\\
\mathrm{p}\left(`', \mathit{expr}, \mathit{INT}\right) &\bumpeq \mathrm{p}\left(`', \mathit{expression}, \mathit{int}\right)\\
\mathrm{p}\left(`', \mathit{expr}, \mathit{expr_1}\right) &\bumpeq \mathrm{p}\left(`', \mathit{expression}, \mathit{binary}\right)\\
\mathrm{p}\left(`', \mathit{expr}, \mathit{expr_2}\right) &\bumpeq \mathrm{p}\left(`', \mathit{expression}, \mathit{apply}\right)\\
\mathrm{p}\left(`', \mathit{expr}, \mathit{expr_3}\right) &\bumpeq \mathrm{p}\left(`', \mathit{expression}, \mathit{conditional}\right)\\
\mathrm{p}\left(`', \mathit{expr_1}, \mathrm{seq}\left([\mathit{expr}, \mathit{ops}, \mathit{expr}]\right)\right) &\bumpeq \mathrm{p}\left(`', \mathit{binary}, \mathrm{seq}\left([\mathit{expression}, \mathit{operator}, \mathit{expression}]\right)\right)\\
\mathrm{p}\left(`', \mathit{expr_2}, \mathrm{seq}\left([\mathit{ID}, +(\mathit{expr})]\right)\right) &\bumpeq \mathrm{p}\left(`', \mathit{apply}, \mathrm{seq}\left([\mathit{str}, +(\mathit{expression})]\right)\right)\\
\mathrm{p}\left(`', \mathit{expr_3}, \mathrm{seq}\left([\mathit{expr}, \mathit{expr}, \mathit{expr}]\right)\right) &\bumpeq \mathrm{p}\left(`', \mathit{conditional}, \mathrm{seq}\left([\mathit{expression}, \mathit{expression}, \mathit{expression}]\right)\right)
\end{aligned}
$$

This yields the following nominal mapping:

$$
\begin{aligned}
\mathit{antlr} \diamond \mathit{master} = \{&\langle \mathit{program}, \mathit{program}\rangle,\\
&\langle \mathit{expr_3}, \mathit{conditional}\rangle,\\
&\langle \mathit{expr_1}, \mathit{binary}\rangle,\\
&\langle \mathit{function}, \mathit{function}\rangle,\\
&\langle \mathit{ID}, \mathit{str}\rangle,\\
&\langle \mathit{expr}, \mathit{expression}\rangle,\\
&\langle \mathit{INT}, \mathit{int}\rangle,\\
&\langle \mathit{ops}, \mathit{operator}\rangle,\\
&\langle \mathit{NEWLINE}, \omega\rangle,\\
&\langle \mathit{expr_2}, \mathit{apply}\rangle\}
\end{aligned}
$$

Which is exercised with these grammar transformation steps:

- **renameN-renameN** $\mathit{expr_3}$ to *conditional*
- **renameN-renameN** $\mathit{expr_1}$ to *binary*
- **renameN-renameN** *ID* to *str*
- **renameN-renameN** *expr* to *expression*
- **renameN-renameN** *INT* to *int*
- **renameN-renameN** *ops* to *operator*
- **renameN-renameN** $\mathit{expr_2}$ to *apply*

1.6 Structural resolution

- **project-inject**
  $\mathrm{p}\left(``, \mathit{function}, \mathrm{seq}\left(\left[\mathit{str}, +(\mathit{str}), \mathit{expression}, +\left(\boxed{\mathit{NEWLINE}}\right)\right]\right)\right)$

Grammar 2

Definite Clause Grammar

Source name: **dcg**

2.1 Source grammar

- Source artifact: topics/fl/prolog1/Parser.pro
- Grammar extractor: shared/prolog/cli/dcg2bgf.pro

Production rules
p('', *program*, +(*function*))
p('', *function*, seq ([*name*, +(*name*) , '=', *expr*, +(*newline*)]))
p('binary', *expr*, seq ([*atom*, ∗(seq ([*ops*, *atom*]))]))
p('apply', *expr*, seq ([*name*, +(*atom*)]))
p('ifThenElse', *expr*, seq (['if', *expr*, 'then', *expr*, 'else', *expr*]))
p('literal', *atom*, *int*)
p('argument', *atom*, *name*)
p('', *atom*, seq (['(', *expr*, ')']))
p('equal', *ops*, '==')
p('plus', *ops*, '+')
p('minus', *ops*, '-')

2.2 Mutations

- **unite-splitN** *expr*
  p ('literal', *atom*, *int*)
  p ('argument', *atom*, *name*)
  p ('', *atom*, seq (['(', *expr*, ')']))
- **assoc-iterate**
  p ('binary', *expr*, seq ([*expr*, *ops*, *expr*]))

2.3 Normalizations

- **reroot-reroot** [] to [*program*]
- **unlabel-designate**
  p (['binary'], *expr*, seq ([*expr*, *ops*, *expr*]))
- **unlabel-designate**
  p (['apply'], *expr*, seq ([*name*, +(*expr*)]))
- **unlabel-designate**
  p (['ifThenElse'], *expr*, seq (['if', *expr*, 'then', *expr*, 'else', *expr*]))
- **unlabel-designate**
  p (['literal'], *expr*, *int*)

- **unlabel-designate**
  $\mathrm{p}\left(\boxed{\text{`argument'}}, \mathit{expr}, \mathit{name}\right)$
- **unlabel-designate**
  $\mathrm{p}\left(\boxed{\text{`equal'}}, \mathit{ops}, \text{`=='}\right)$
- **unlabel-designate**
  $\mathrm{p}\left(\boxed{\text{`plus'}}, \mathit{ops}, \text{`+'}\right)$
- **unlabel-designate**
  $\mathrm{p}\left(\boxed{\text{`minus'}}, \mathit{ops}, \text{`-'}\right)$
- **abstractize-concretize**
  $\mathrm{p}\left(\text{`'}, \mathit{expr}, \mathrm{seq}\left(\left[\boxed{\text{`('}}, \mathit{expr}, \boxed{\text{`)'}}\right]\right)\right)$
- **abstractize-concretize**
  $\mathrm{p}\left(\text{`'}, \mathit{function}, \mathrm{seq}\left(\left[\mathit{name}, +(\mathit{name}), \boxed{\text{`='}}, \mathit{expr}, +(\mathit{newline})\right]\right)\right)$
- **abstractize-concretize**
  $\mathrm{p}\left(\text{`'}, \mathit{ops}, \boxed{\text{`=='}}\right)$
- **abstractize-concretize**
  $\mathrm{p}\left(\text{`'}, \mathit{expr}, \mathrm{seq}\left(\left[\boxed{\text{`if'}}, \mathit{expr}, \boxed{\text{`then'}}, \mathit{expr}, \boxed{\text{`else'}}, \mathit{expr}\right]\right)\right)$
- **abstractize-concretize**
  $\mathrm{p}\left(\text{`'}, \mathit{ops}, \boxed{\text{`-'}}\right)$
- **abstractize-concretize**
  $\mathrm{p}\left(\text{`'}, \mathit{ops}, \boxed{\text{`+'}}\right)$
- **undefine-define**
  $\mathrm{p}\left(\text{`'}, \mathit{ops}, \varepsilon\right)$
- **abridge-detour**
  $\mathrm{p}\left(\text{`'}, \mathit{expr}, \mathit{expr}\right)$
- **extract-inline** in *expr*
  $\mathrm{p}\left(\text{`'}, \mathit{expr_1}, \mathrm{seq}\left([\mathit{expr}, \mathit{ops}, \mathit{expr}]\right)\right)$
- **extract-inline** in *expr*
  $\mathrm{p}\left(\text{`'}, \mathit{expr_2}, \mathrm{seq}\left([\mathit{name}, +(\mathit{expr})]\right)\right)$
- **extract-inline** in *expr*
  $\mathrm{p}\left(\text{`'}, \mathit{expr_3}, \mathrm{seq}\left([\mathit{expr}, \mathit{expr}, \mathit{expr}]\right)\right)$

2.4 Grammar in ANF

Production rule	Production signature
$\mathrm{p}(\text{`'}, \mathit{program}, +(\mathit{function}))$	$\{\langle \mathit{function}, + \rangle\}$
$\mathrm{p}(\text{`'}, \mathit{function}, \mathrm{seq}([\mathit{name}, +(\mathit{name}), \mathit{expr}, +(\mathit{newline})]))$	$\{\langle \mathit{expr}, 1\rangle, \langle \mathit{newline}, +\rangle, \langle \mathit{name}, 1+\rangle\}$
$\mathrm{p}(\text{`'}, \mathit{expr}, \mathit{expr_1})$	$\{\langle \mathit{expr_1}, 1\rangle\}$
$\mathrm{p}(\text{`'}, \mathit{expr}, \mathit{expr_2})$	$\{\langle \mathit{expr_2}, 1\rangle\}$
$\mathrm{p}(\text{`'}, \mathit{expr}, \mathit{expr_3})$	$\{\langle \mathit{expr_3}, 1\rangle\}$
$\mathrm{p}(\text{`'}, \mathit{expr}, \mathit{int})$	$\{\langle \mathit{int}, 1\rangle\}$
$\mathrm{p}(\text{`'}, \mathit{expr}, \mathit{name})$	$\{\langle \mathit{name}, 1\rangle\}$
$\mathrm{p}(\text{`'}, \mathit{expr_1}, \mathrm{seq}([\mathit{expr}, \mathit{ops}, \mathit{expr}]))$	$\{\langle \mathit{ops}, 1\rangle, \langle \mathit{expr}, 11\rangle\}$
$\mathrm{p}(\text{`'}, \mathit{expr_2}, \mathrm{seq}([\mathit{name}, +(\mathit{expr})]))$	$\{\langle \mathit{expr}, +\rangle, \langle \mathit{name}, 1\rangle\}$
$\mathrm{p}(\text{`'}, \mathit{expr_3}, \mathrm{seq}([\mathit{expr}, \mathit{expr}, \mathit{expr}]))$	$\{\langle \mathit{expr}, 111\rangle\}$

2.5 Nominal resolution

Production rules are matched as follows (ANF on the left, master grammar on the right):

$$\begin{array}{rcl}
\mathrm{p}\left(`', \mathit{program}, +(\mathit{function})\right) & \simeq & \mathrm{p}\left(`', \mathit{program}, +(\mathit{function})\right) \\
\mathrm{p}\left(`', \mathit{function}, \mathrm{seq}\left(\left[\mathit{name}, +(\mathit{name}), \mathit{expr}, +(\mathit{newline})\right]\right)\right) & \not\simeq & \mathrm{p}\left(`', \mathit{function}, \mathrm{seq}\left(\left[\mathit{str}, +(\mathit{str}), \mathit{expression}\right]\right)\right) \\
\mathrm{p}\left(`', \mathit{expr}, \mathit{expr_1}\right) & \simeq & \mathrm{p}\left(`', \mathit{expression}, \mathit{binary}\right) \\
\mathrm{p}\left(`', \mathit{expr}, \mathit{expr_2}\right) & \simeq & \mathrm{p}\left(`', \mathit{expression}, \mathit{apply}\right) \\
\mathrm{p}\left(`', \mathit{expr}, \mathit{expr_3}\right) & \simeq & \mathrm{p}\left(`', \mathit{expression}, \mathit{conditional}\right) \\
\mathrm{p}\left(`', \mathit{expr}, \mathit{int}\right) & \simeq & \mathrm{p}\left(`', \mathit{expression}, \mathit{int}\right) \\
\mathrm{p}\left(`', \mathit{expr}, \mathit{name}\right) & \simeq & \mathrm{p}\left(`', \mathit{expression}, \mathit{str}\right) \\
\mathrm{p}\left(`', \mathit{expr_1}, \mathrm{seq}\left(\left[\mathit{expr}, \mathit{ops}, \mathit{expr}\right]\right)\right) & \simeq & \mathrm{p}\left(`', \mathit{binary}, \mathrm{seq}\left(\left[\mathit{expression}, \mathit{operator}, \mathit{expression}\right]\right)\right) \\
\mathrm{p}\left(`', \mathit{expr_2}, \mathrm{seq}\left(\left[\mathit{name}, +(\mathit{expr})\right]\right)\right) & \simeq & \mathrm{p}\left(`', \mathit{apply}, \mathrm{seq}\left(\left[\mathit{str}, +(\mathit{expression})\right]\right)\right) \\
\mathrm{p}\left(`', \mathit{expr_3}, \mathrm{seq}\left(\left[\mathit{expr}, \mathit{expr}, \mathit{expr}\right]\right)\right) & \simeq & \mathrm{p}\left(`', \mathit{conditional}, \mathrm{seq}\left(\left[\mathit{expression}, \mathit{expression}, \mathit{expression}\right]\right)\right)
\end{array}$$

This yields the following nominal mapping:

$$\begin{aligned}
\mathit{dcg} \diamond \mathit{master} = \{ & \langle \mathit{program}, \mathit{program} \rangle, \\
& \langle \mathit{expr_3}, \mathit{conditional} \rangle, \\
& \langle \mathit{expr_1}, \mathit{binary} \rangle, \\
& \langle \mathit{function}, \mathit{function} \rangle, \\
& \langle \mathit{expr}, \mathit{expression} \rangle, \\
& \langle \mathit{name}, \mathit{str} \rangle, \\
& \langle \mathit{ops}, \mathit{operator} \rangle, \\
& \langle \mathit{int}, \mathit{int} \rangle, \\
& \langle \mathit{newline}, \omega \rangle, \\
& \langle \mathit{expr_2}, \mathit{apply} \rangle \}
\end{aligned}$$

Which is exercised with these grammar transformation steps:

- **renameN-renameN** $\mathit{expr_3}$ to *conditional*
- **renameN-renameN** $\mathit{expr_1}$ to *binary*
- **renameN-renameN** *expr* to *expression*
- **renameN-renameN** *name* to *str*
- **renameN-renameN** *ops* to *operator*
- **renameN-renameN** *int* to *int*
- **renameN-renameN** $\mathit{expr_2}$ to *apply*

2.6 Structural resolution

- **project-inject**
  $\mathrm{p}\left(`', \mathit{function}, \mathrm{seq}\left(\left[\mathit{str}, +(\mathit{str}), \mathit{expression}, +(\boxed{\mathit{newline}})\right]\right)\right)$

Grammar 3

Eclipse Modeling Framework

Source name: **emf**

3.1 Source grammar

- Source artifact: topics/fl/emf2/model/fl.ecore
- Grammar extractor: topics/extraction/ecore/ecore2bgf.xslt

Production rules
p('', *Apply*, seq ([sel ('name', *str*) , +(sel ('arg', *Expr*))]))
p('', *Argument*, sel ('name', *str*))
p('', *Binary*, seq ([sel ('ops', *Ops*) , sel ('left', *Expr*) , sel ('right', *Expr*)]))
p('', *Expr*, choice([*Apply*, *Argument*, *Binary*, *IfThenElse*, *Literal*]))
p('', *Function*, seq ([sel ('name', *str*) , +(sel ('arg', *str*)) , sel ('rhs', *Expr*)]))
p('', *IfThenElse*, seq ([sel ('ifExpr', *Expr*) , sel ('thenExpr', *Expr*) , sel ('elseExpr', *Expr*)]))
p('', *Literal*, sel ('info', *int*))
p('', *Ops*, choice([sel ('Equal', ε) , sel ('Plus', ε) , sel ('Minus', ε)]))
p('', *ProgramType*, +(sel ('function', *Function*)))

3.2 Normalizations

- **reroot-reroot** [] to [*ProgramType*]
- **unlabel-designate**
  $\mathrm{p}\left(\boxed{\text{`name'}}, \mathit{Argument}, \mathit{str}\right)$
- **unlabel-designate**
  $\mathrm{p}\left(\boxed{\text{`info'}}, \mathit{Literal}, \mathit{int}\right)$
- **anonymize-deanonymize**
  $\mathrm{p}\left(\text{`'}, \mathit{Function}, \mathrm{seq}\left(\left[\boxed{\mathrm{sel}\left(\text{`name'}, \mathit{str}\right)}, +\left(\boxed{\mathrm{sel}\left(\text{`arg'}, \mathit{str}\right)}\right), \boxed{\mathrm{sel}\left(\text{`rhs'}, \mathit{Expr}\right)}\right]\right)\right)$
- **anonymize-deanonymize**
  $\mathrm{p}\left(\text{`'}, \mathit{Apply}, \mathrm{seq}\left(\left[\boxed{\mathrm{sel}\left(\text{`name'}, \mathit{str}\right)}, +\left(\boxed{\mathrm{sel}\left(\text{`arg'}, \mathit{Expr}\right)}\right)\right]\right)\right)$
- **anonymize-deanonymize**
  $\mathrm{p}\left(\text{`'}, \mathit{IfThenElse}, \mathrm{seq}\left(\left[\boxed{\mathrm{sel}\left(\text{`ifExpr'}, \mathit{Expr}\right)}, \boxed{\mathrm{sel}\left(\text{`thenExpr'}, \mathit{Expr}\right)}, \boxed{\mathrm{sel}\left(\text{`elseExpr'}, \mathit{Expr}\right)}\right]\right)\right)$

- **anonymize-deanonymize**
  $\mathrm{p}\left(`', \mathit{Ops}, \mathrm{choice}\left(\left[\boxed{\mathrm{sel}\,(`\mathrm{Equal}', \varepsilon)}, \boxed{\mathrm{sel}\,(`\mathrm{Plus}', \varepsilon)}, \boxed{\mathrm{sel}\,(`\mathrm{Minus}', \varepsilon)}\right]\right)\right)$
- **anonymize-deanonymize**
  $\mathrm{p}\left(`', \mathit{ProgramType}, +\left(\boxed{\mathrm{sel}\,(`\mathrm{function}', \mathit{Function})}\right)\right)$
- **anonymize-deanonymize**
  $\mathrm{p}\left(`', \mathit{Binary}, \mathrm{seq}\left(\left[\boxed{\mathrm{sel}\,(`\mathrm{ops}', \mathit{Ops})}, \boxed{\mathrm{sel}\,(`\mathrm{left}', \mathit{Expr})}, \boxed{\mathrm{sel}\,(`\mathrm{right}', \mathit{Expr})}\right]\right)\right)$
- **vertical-horizontal** in *Expr*
- **undefine-define**
  $\mathrm{p}\,(`', \mathit{Ops}, \varepsilon)$
- **unchain-chain**
  $\mathrm{p}\,(`', \mathit{Expr}, \mathit{Apply})$
- **unchain-chain**
  $\mathrm{p}\,(`', \mathit{Expr}, \mathit{Argument})$
- **unchain-chain**
  $\mathrm{p}\,(`', \mathit{Expr}, \mathit{Binary})$
- **unchain-chain**
  $\mathrm{p}\,(`', \mathit{Expr}, \mathit{IfThenElse})$
- **unchain-chain**
  $\mathrm{p}\,(`', \mathit{Expr}, \mathit{Literal})$
- **unlabel-designate**
  $\mathrm{p}\left(\boxed{`\mathrm{Apply}'}, \mathit{Expr}, \mathrm{seq}\,([\mathit{str}, +(\mathit{Expr})])\right)$
- **unlabel-designate**
  $\mathrm{p}\left(\boxed{`\mathrm{Argument}'}, \mathit{Expr}, \mathit{str}\right)$
- **unlabel-designate**
  $\mathrm{p}\left(\boxed{`\mathrm{Binary}'}, \mathit{Expr}, \mathrm{seq}\,([\mathit{Ops}, \mathit{Expr}, \mathit{Expr}])\right)$
- **unlabel-designate**
  $\mathrm{p}\left(\boxed{`\mathrm{IfThenElse}'}, \mathit{Expr}, \mathrm{seq}\,([\mathit{Expr}, \mathit{Expr}, \mathit{Expr}])\right)$
- **unlabel-designate**
  $\mathrm{p}\left(\boxed{`\mathrm{Literal}'}, \mathit{Expr}, \mathit{int}\right)$
- **extract-inline** in *Expr*
  $\mathrm{p}\,(`', \mathit{Expr}_1, \mathrm{seq}\,([\mathit{str}, +(\mathit{Expr})]))$
- **extract-inline** in *Expr*
  $\mathrm{p}\,(`', \mathit{Expr}_2, \mathrm{seq}\,([\mathit{Ops}, \mathit{Expr}, \mathit{Expr}]))$
- **extract-inline** in *Expr*
  $\mathrm{p}\,(`', \mathit{Expr}_3, \mathrm{seq}\,([\mathit{Expr}, \mathit{Expr}, \mathit{Expr}]))$

3.3 Grammar in ANF

Production rule	Production signature
$\mathrm{p}\,(`', \mathit{Expr}, \mathit{Expr}_1)$	$\{\langle \mathit{Expr}_1, 1\rangle\}$
$\mathrm{p}\,(`', \mathit{Expr}, \mathit{str})$	$\{\langle \mathit{str}, 1\rangle\}$
$\mathrm{p}\,(`', \mathit{Expr}, \mathit{Expr}_2)$	$\{\langle \mathit{Expr}_2, 1\rangle\}$
$\mathrm{p}\,(`', \mathit{Expr}, \mathit{Expr}_3)$	$\{\langle \mathit{Expr}_3, 1\rangle\}$
$\mathrm{p}\,(`', \mathit{Expr}, \mathit{int})$	$\{\langle \mathit{int}, 1\rangle\}$
$\mathrm{p}\,(`', \mathit{Function}, \mathrm{seq}\,([\mathit{str}, +(\mathit{str}), \mathit{Expr}]))$	$\{\langle \mathit{str}, 1{+}\rangle, \langle \mathit{Expr}, 1\rangle\}$
$\mathrm{p}\,(`', \mathit{ProgramType}, +(\mathit{Function}))$	$\{\langle \mathit{Function}, +\rangle\}$
$\mathrm{p}\,(`', \mathit{Expr}_1, \mathrm{seq}\,([\mathit{str}, +(\mathit{Expr})]))$	$\{\langle \mathit{str}, 1\rangle, \langle \mathit{Expr}, +\rangle\}$
$\mathrm{p}\,(`', \mathit{Expr}_2, \mathrm{seq}\,([\mathit{Ops}, \mathit{Expr}, \mathit{Expr}]))$	$\{\langle \mathit{Ops}, 1\rangle, \langle \mathit{Expr}, 11\rangle\}$
$\mathrm{p}\,(`', \mathit{Expr}_3, \mathrm{seq}\,([\mathit{Expr}, \mathit{Expr}, \mathit{Expr}]))$	$\{\langle \mathit{Expr}, 111\rangle\}$

3.4 Nominal resolution

Production rules are matched as follows (ANF on the left, master grammar on the right):

$$\begin{aligned}
\mathrm{p}\,(`', \mathit{Expr}, \mathit{Expr_1}) &\bumpeq \mathrm{p}\,(`', \mathit{expression}, \mathit{apply})\\
\mathrm{p}\,(`', \mathit{Expr}, \mathit{str}) &\bumpeq \mathrm{p}\,(`', \mathit{expression}, \mathit{str})\\
\mathrm{p}\,(`', \mathit{Expr}, \mathit{Expr_2}) &\bumpeq \mathrm{p}\,(`', \mathit{expression}, \mathit{binary})\\
\mathrm{p}\,(`', \mathit{Expr}, \mathit{Expr_3}) &\bumpeq \mathrm{p}\,(`', \mathit{expression}, \mathit{conditional})\\
\mathrm{p}\,(`', \mathit{Expr}, \mathit{int}) &\bumpeq \mathrm{p}\,(`', \mathit{expression}, \mathit{int})\\
\mathrm{p}\,(`', \mathit{Function}, \mathrm{seq}\,([\mathit{str}, +(\mathit{str}), \mathit{Expr}])) &\bumpeq \mathrm{p}\,(`', \mathit{function}, \mathrm{seq}\,([\mathit{str}, +(\mathit{str}), \mathit{expression}]))\\
\mathrm{p}\,(`', \mathit{ProgramType}, +(\mathit{Function})) &\bumpeq \mathrm{p}\,(`', \mathit{program}, +(\mathit{function}))\\
\mathrm{p}\,(`', \mathit{Expr_1}, \mathrm{seq}\,([\mathit{str}, +(\mathit{Expr})])) &\bumpeq \mathrm{p}\,(`', \mathit{apply}, \mathrm{seq}\,([\mathit{str}, +(\mathit{expression})]))\\
\mathrm{p}\,(`', \mathit{Expr_2}, \mathrm{seq}\,([\mathit{Ops}, \mathit{Expr}, \mathit{Expr}])) &\Bumpeq \mathrm{p}\,(`', \mathit{binary}, \mathrm{seq}\,([\mathit{expression}, \mathit{operator}, \mathit{expression}]))\\
\mathrm{p}\,(`', \mathit{Expr_3}, \mathrm{seq}\,([\mathit{Expr}, \mathit{Expr}, \mathit{Expr}])) &\bumpeq \mathrm{p}\,(`', \mathit{conditional}, \mathrm{seq}\,([\mathit{expression}, \mathit{expression}, \mathit{expression}]))
\end{aligned}$$

This yields the following nominal mapping:

$$\begin{aligned}
\mathit{emf} \diamond \mathit{master} = \{&\langle \mathit{Expr_2}, \mathit{binary}\rangle,\\
&\langle \mathit{ProgramType}, \mathit{program}\rangle,\\
&\langle \mathit{Expr_3}, \mathit{conditional}\rangle,\\
&\langle \mathit{str}, \mathit{str}\rangle,\\
&\langle \mathit{int}, \mathit{int}\rangle,\\
&\langle \mathit{Function}, \mathit{function}\rangle,\\
&\langle \mathit{Expr}, \mathit{expression}\rangle,\\
&\langle \mathit{Expr_1}, \mathit{apply}\rangle,\\
&\langle \mathit{Ops}, \mathit{operator}\rangle\}
\end{aligned}$$

Which is exercised with these grammar transformation steps:

- **renameN-renameN** $\mathit{Expr_2}$ to *binary*
- **renameN-renameN** *ProgramType* to *program*
- **renameN-renameN** $\mathit{Expr_3}$ to *conditional*
- **renameN-renameN** *Function* to *function*
- **renameN-renameN** *Expr* to *expression*
- **renameN-renameN** $\mathit{Expr_1}$ to *apply*
- **renameN-renameN** *Ops* to *operator*

3.5 Structural resolution

- **permute-permute**
  $\mathrm{p}\,(`', \mathit{binary}, \mathrm{seq}\,([\mathit{operator}, \mathit{expression}, \mathit{expression}]))$
  $\mathrm{p}\,(`', \mathit{binary}, \mathrm{seq}\,([\mathit{expression}, \mathit{operator}, \mathit{expression}]))$

Grammar 4

JAXB Data Binding Framework

Source name: **jaxb**

4.1 Source grammar

- Source artifact: topics/fl/java3/fl/Apply.java
- Source artifact: topics/fl/java3/fl/Argument.java
- Source artifact: topics/fl/java3/fl/Binary.java
- Source artifact: topics/fl/java3/fl/Expr.java
- Source artifact: topics/fl/java3/fl/Function.java
- Source artifact: topics/fl/java3/fl/IfThenElse.java
- Source artifact: topics/fl/java3/fl/Literal.java
- Source artifact: topics/fl/java3/fl/ObjectFactory.java
- Source artifact: topics/fl/java3/fl/Ops.java
- Source artifact: topics/fl/java3/fl/Program.java
- Source artifact: topics/fl/java3/fl/package-info.java
- Grammar extractor: topics/extraction/java2bgf/slps/java2bgf/Tool.java

Production rules
p(‘’, *Apply*, seq ([sel (‘Name’, *str*) , sel (‘Arg’, ∗(*Expr*))]))
p(‘’, *Argument*, sel (‘Name’, *str*))
p(‘’, *Binary*, seq ([sel (‘Ops’, *Ops*) , sel (‘Left’, *Expr*) , sel (‘Right’, *Expr*)]))
p(‘’, *Expr*, choice([*Apply*, *Argument*, *Binary*, *IfThenElse*, *Literal*]))
p(‘’, *Function*, seq ([sel (‘Name’, *str*) , sel (‘Arg’, ∗(*str*)) , sel (‘Rhs’, *Expr*)]))
p(‘’, *IfThenElse*, seq ([sel (‘IfExpr’, *Expr*) , sel (‘ThenExpr’, *Expr*) , sel (‘ElseExpr’, *Expr*)]))
p(‘’, *Literal*, sel (‘Info’, *int*))
p(‘’, *ObjectFactory*, ε)
p(‘’, *Ops*, choice([sel (‘EQUAL’, ε) , sel (‘PLUS’, ε) , sel (‘MINUS’, ε)]))
p(‘’, $package - info$, φ)
p(‘’, *Program*, sel (‘Function’, ∗(*Function*)))

4.2 Normalizations

- **reroot-reroot** [] to [*Program*]
- **unlabel-designate**
  $\mathrm{p}\left(\boxed{\text{‘Name’}}, \mathit{Argument}, \mathit{str}\right)$
- **unlabel-designate**
  $\mathrm{p}\left(\boxed{\text{‘Info’}}, \mathit{Literal}, \mathit{int}\right)$
- **unlabel-designate**
  $\mathrm{p}\left(\boxed{\text{‘Function’}}, \mathit{Program}, *(\mathit{Function})\right)$
- **anonymize-deanonymize**
  $\mathrm{p}\left(\text{‘’}, \mathit{IfThenElse}, \mathrm{seq}\left(\left[\left[\boxed{\mathrm{sel}\,(\text{‘IfExpr’}, \mathit{Expr})}, \boxed{\mathrm{sel}\,(\text{‘ThenExpr’}, \mathit{Expr})}, \boxed{\mathrm{sel}\,(\text{‘ElseExpr’}, \mathit{Expr})}\right]\right]\right)\right)$
- **anonymize-deanonymize**
  $\mathrm{p}\left(\text{‘’}, \mathit{Function}, \mathrm{seq}\left(\left[\left[\boxed{\mathrm{sel}\,(\text{‘Name’}, \mathit{str})}, \boxed{\mathrm{sel}\,(\text{‘Arg’}, *(\mathit{str}))}, \boxed{\mathrm{sel}\,(\text{‘Rhs’}, \mathit{Expr})}\right]\right]\right)\right)$
- **anonymize-deanonymize**
  $\mathrm{p}\left(\text{‘’}, \mathit{Binary}, \mathrm{seq}\left(\left[\left[\boxed{\mathrm{sel}\,(\text{‘Ops’}, \mathit{Ops})}, \boxed{\mathrm{sel}\,(\text{‘Left’}, \mathit{Expr})}, \boxed{\mathrm{sel}\,(\text{‘Right’}, \mathit{Expr})}\right]\right]\right)\right)$
- **anonymize-deanonymize**
  $\mathrm{p}\left(\text{‘’}, \mathit{Apply}, \mathrm{seq}\left(\left[\left[\boxed{\mathrm{sel}\,(\text{‘Name’}, \mathit{str})}, \boxed{\mathrm{sel}\,(\text{‘Arg’}, *(\mathit{Expr}))}\right]\right]\right)\right)$
- **anonymize-deanonymize**
  $\mathrm{p}\left(\text{‘’}, \mathit{Ops}, \mathrm{choice}\left(\left[\left[\boxed{\mathrm{sel}\,(\text{‘EQUAL’}, \varepsilon)}, \boxed{\mathrm{sel}\,(\text{‘PLUS’}, \varepsilon)}, \boxed{\mathrm{sel}\,(\text{‘MINUS’}, \varepsilon)}\right]\right]\right)\right)$
- **vertical-horizontal** in *Expr*
- **undefine-define**
  $\mathrm{p}\,(\text{‘’}, \mathit{Ops}, \varepsilon)$
- **eliminate-introduce**
  $\mathrm{p}\,(\text{‘’}, \mathit{ObjectFactory}, \varepsilon)$
- **eliminate-introduce**
  $\mathrm{p}\,(\text{‘’}, \mathit{package} - \mathit{info}, \varphi)$
- **unchain-chain**
  $\mathrm{p}\,(\text{‘’}, \mathit{Expr}, \mathit{Apply})$
- **unchain-chain**
  $\mathrm{p}\,(\text{‘’}, \mathit{Expr}, \mathit{Argument})$
- **unchain-chain**
  $\mathrm{p}\,(\text{‘’}, \mathit{Expr}, \mathit{Binary})$
- **unchain-chain**
  $\mathrm{p}\,(\text{‘’}, \mathit{Expr}, \mathit{IfThenElse})$
- **unchain-chain**
  $\mathrm{p}\,(\text{‘’}, \mathit{Expr}, \mathit{Literal})$
- **unlabel-designate**
  $\mathrm{p}\left(\boxed{\text{‘Apply’}}, \mathit{Expr}, \mathrm{seq}\,([\mathit{str}, *(\mathit{Expr})])\right)$
- **unlabel-designate**
  $\mathrm{p}\left(\boxed{\text{‘Argument’}}, \mathit{Expr}, \mathit{str}\right)$
- **unlabel-designate**
  $\mathrm{p}\left(\boxed{\text{‘Binary’}}, \mathit{Expr}, \mathrm{seq}\,([\mathit{Ops}, \mathit{Expr}, \mathit{Expr}])\right)$
- **unlabel-designate**
  $\mathrm{p}\left(\boxed{\text{‘IfThenElse’}}, \mathit{Expr}, \mathrm{seq}\,([\mathit{Expr}, \mathit{Expr}, \mathit{Expr}])\right)$
- **unlabel-designate**
  $\mathrm{p}\left(\boxed{\text{‘Literal’}}, \mathit{Expr}, \mathit{int}\right)$
- **extract-inline** in *Expr*
  $\mathrm{p}\,(\text{‘’}, \mathit{Expr}_1, \mathrm{seq}\,([\mathit{str}, *(\mathit{Expr})]))$
- **extract-inline** in *Expr*
  $\mathrm{p}\,(\text{‘’}, \mathit{Expr}_2, \mathrm{seq}\,([\mathit{Ops}, \mathit{Expr}, \mathit{Expr}]))$
- **extract-inline** in *Expr*
  $\mathrm{p}\,(\text{‘’}, \mathit{Expr}_3, \mathrm{seq}\,([\mathit{Expr}, \mathit{Expr}, \mathit{Expr}]))$

4.3 Grammar in ANF

Production rule	Production signature
$\text{p}\,(`', \mathit{Expr}, \mathit{Expr_1})$	$\{\langle \mathit{Expr_1}, 1\rangle\}$
$\text{p}\,(`', \mathit{Expr}, \mathit{str})$	$\{\langle \mathit{str}, 1\rangle\}$
$\text{p}\,(`', \mathit{Expr}, \mathit{Expr_2})$	$\{\langle \mathit{Expr_2}, 1\rangle\}$
$\text{p}\,(`', \mathit{Expr}, \mathit{Expr_3})$	$\{\langle \mathit{Expr_3}, 1\rangle\}$
$\text{p}\,(`', \mathit{Expr}, \mathit{int})$	$\{\langle \mathit{int}, 1\rangle\}$
$\text{p}\,(`', \mathit{Function}, \text{seq}\,([\mathit{str}, *(\mathit{str})\,, \mathit{Expr}]))$	$\{\langle \mathit{Expr}, 1\rangle, \langle \mathit{str}, 1*\rangle\}$
$\text{p}\,(`', \mathit{Program}, *(\mathit{Function}))$	$\{\langle \mathit{Function}, *\rangle\}$
$\text{p}\,(`', \mathit{Expr_1}, \text{seq}\,([\mathit{str}, *(\mathit{Expr})]))$	$\{\langle \mathit{str}, 1\rangle, \langle \mathit{Expr}, *\rangle\}$
$\text{p}\,(`', \mathit{Expr_2}, \text{seq}\,([\mathit{Ops}, \mathit{Expr}, \mathit{Expr}]))$	$\{\langle \mathit{Ops}, 1\rangle, \langle \mathit{Expr}, 11\rangle\}$
$\text{p}\,(`', \mathit{Expr_3}, \text{seq}\,([\mathit{Expr}, \mathit{Expr}, \mathit{Expr}]))$	$\{\langle \mathit{Expr}, 111\rangle\}$

4.4 Nominal resolution

Production rules are matched as follows (ANF on the left, master grammar on the right):

$$\begin{aligned}
\text{p}\,(`', \mathit{Expr}, \mathit{Expr_1}) &\doteqdot \text{p}\,(`', \mathit{expression}, \mathit{apply})\\
\text{p}\,(`', \mathit{Expr}, \mathit{str}) &\doteqdot \text{p}\,(`', \mathit{expression}, \mathit{str})\\
\text{p}\,(`', \mathit{Expr}, \mathit{Expr_2}) &\doteqdot \text{p}\,(`', \mathit{expression}, \mathit{binary})\\
\text{p}\,(`', \mathit{Expr}, \mathit{Expr_3}) &\doteqdot \text{p}\,(`', \mathit{expression}, \mathit{conditional})\\
\text{p}\,(`', \mathit{Expr}, \mathit{int}) &\doteqdot \text{p}\,(`', \mathit{expression}, \mathit{int})\\
\text{p}\,(`', \mathit{Function}, \text{seq}\,([\mathit{str}, *(\mathit{str})\,, \mathit{Expr}])) &\Doteq \text{p}\,(`', \mathit{function}, \text{seq}\,([\mathit{str}, +(\mathit{str})\,, \mathit{expression}]))\\
\text{p}\,(`', \mathit{Program}, *(\mathit{Function})) &\Doteq \text{p}\,(`', \mathit{program}, +(\mathit{function}))\\
\text{p}\,(`', \mathit{Expr_1}, \text{seq}\,([\mathit{str}, *(\mathit{Expr})])) &\Doteq \text{p}\,(`', \mathit{apply}, \text{seq}\,([\mathit{str}, +(\mathit{expression})]))\\
\text{p}\,(`', \mathit{Expr_2}, \text{seq}\,([\mathit{Ops}, \mathit{Expr}, \mathit{Expr}])) &\Doteq \text{p}\,(`', \mathit{binary}, \text{seq}\,([\mathit{expression}, \mathit{operator}, \mathit{expression}]))\\
\text{p}\,(`', \mathit{Expr_3}, \text{seq}\,([\mathit{Expr}, \mathit{Expr}, \mathit{Expr}])) &\doteqdot \text{p}\,(`', \mathit{conditional}, \text{seq}\,([\mathit{expression}, \mathit{expression}, \mathit{expression}]))
\end{aligned}$$

This yields the following nominal mapping:

$$\begin{aligned}
\mathit{jaxb} \diamond \mathit{master} = \{&\langle \mathit{Expr_2}, \mathit{binary}\rangle,\\
&\langle \mathit{Expr_3}, \mathit{conditional}\rangle,\\
&\langle \mathit{int}, \mathit{int}\rangle,\\
&\langle \mathit{Function}, \mathit{function}\rangle,\\
&\langle \mathit{str}, \mathit{str}\rangle,\\
&\langle \mathit{Program}, \mathit{program}\rangle,\\
&\langle \mathit{Expr}, \mathit{expression}\rangle,\\
&\langle \mathit{Expr_1}, \mathit{apply}\rangle,\\
&\langle \mathit{Ops}, \mathit{operator}\rangle\}
\end{aligned}$$

Which is exercised with these grammar transformation steps:

- **renameN-renameN** $\mathit{Expr_2}$ to *binary*
- **renameN-renameN** $\mathit{Expr_3}$ to *conditional*
- **renameN-renameN** *Function* to *function*
- **renameN-renameN** *Program* to *program*
- **renameN-renameN** *Expr* to *expression*
- **renameN-renameN** $\mathit{Expr_1}$ to *apply*
- **renameN-renameN** *Ops* to *operator*

4.5 Structural resolution

- **narrow-widen** in *function*
  $*(\mathit{str})$
  $+(\mathit{str})$

- **narrow-widen** in *program*
  $*(\mathit{function})$
  $+(\mathit{function})$
- **narrow-widen** in *apply*
  $*(\mathit{expression})$
  $+(\mathit{expression})$
- **permute-permute**
  $\mathrm{p}\,(``, \mathit{binary}, \mathrm{seq}\,([\mathit{operator}, \mathit{expression}, \mathit{expression}]))$
  $\mathrm{p}\,(``, \mathit{binary}, \mathrm{seq}\,([\mathit{expression}, \mathit{operator}, \mathit{expression}]))$

Grammar 5

Java Object Model

Source name: **om**

5.1 Source grammar

- Source artifact: topics/fl/java1/types/Apply.java
- Source artifact: topics/fl/java1/types/Argument.java
- Source artifact: topics/fl/java1/types/Binary.java
- Source artifact: topics/fl/java1/types/Expr.java
- Source artifact: topics/fl/java1/types/Function.java
- Source artifact: topics/fl/java1/types/IfThenElse.java
- Source artifact: topics/fl/java1/types/Literal.java
- Source artifact: topics/fl/java1/types/Ops.java
- Source artifact: topics/fl/java1/types/Program.java
- Source artifact: topics/fl/java1/types/Visitor.java
- Grammar extractor: topics/extraction/java2bgf/slps/java2bgf/Tool.java

Production rules
p(‘’, *Apply*, seq ([sel (‘name’, *str*) , sel (‘args’, ∗(*Expr*))]))
p(‘’, *Argument*, sel (‘name’, *str*))
p(‘’, *Binary*, seq ([sel (‘ops’, *Ops*) , sel (‘left’, *Expr*) , sel (‘right’, *Expr*)]))
p(‘’, *Expr*, choice([*Apply*, *Argument*, *Binary*, *IfThenElse*, *Literal*]))
p(‘’, *Function*, seq ([sel (‘name’, *str*) , sel (‘args’, ∗(*str*)) , sel (‘rhs’, *Expr*)]))
p(‘’, *IfThenElse*, seq ([sel (‘ifExpr’, *Expr*) , sel (‘thenExpr’, *Expr*) , sel (‘elseExpr’, *Expr*)]))
p(‘’, *Literal*, sel (‘info’, *int*))
p(‘’, *Ops*, choice([sel (‘Equal’, ε) , sel (‘Plus’, ε) , sel (‘Minus’, ε)]))
p(‘’, *Program*, sel (‘functions’, ∗(*Function*)))
p(‘’, *Visitor*, φ)

5.2 Normalizations

- **reroot-reroot** [] to [*Program*]
- **unlabel-designate**
  p ([‘name’], *Argument*, *str*)

- **unlabel-designate**
  $\mathrm{p}\left(\boxed{\text{‘info’}}, \mathit{Literal}, \mathit{int}\right)$
- **unlabel-designate**
  $\mathrm{p}\left(\boxed{\text{‘functions’}}, \mathit{Program}, *(\mathit{Function})\right)$
- **anonymize-deanonymize**
  $\mathrm{p}\left(\text{‘’}, \mathit{IfThenElse}, \mathrm{seq}\left(\left[\boxed{\mathrm{sel}\,(\text{‘ifExpr’}, \mathit{Expr})}, \boxed{\mathrm{sel}\,(\text{‘thenExpr’}, \mathit{Expr})}, \boxed{\mathrm{sel}\,(\text{‘elseExpr’}, \mathit{Expr})}\right]\right)\right)$
- **anonymize-deanonymize**
  $\mathrm{p}\left(\text{‘’}, \mathit{Ops}, \mathrm{choice}\left(\left[\boxed{\mathrm{sel}\,(\text{‘Equal’}, \varepsilon)}, \boxed{\mathrm{sel}\,(\text{‘Plus’}, \varepsilon)}, \boxed{\mathrm{sel}\,(\text{‘Minus’}, \varepsilon)}\right]\right)\right)$
- **anonymize-deanonymize**
  $\mathrm{p}\left(\text{‘’}, \mathit{Apply}, \mathrm{seq}\left(\left[\boxed{\mathrm{sel}\,(\text{‘name’}, \mathit{str})}, \boxed{\mathrm{sel}\,(\text{‘args’}, *(\mathit{Expr}))}\right]\right)\right)$
- **anonymize-deanonymize**
  $\mathrm{p}\left(\text{‘’}, \mathit{Function}, \mathrm{seq}\left(\left[\boxed{\mathrm{sel}\,(\text{‘name’}, \mathit{str})}, \boxed{\mathrm{sel}\,(\text{‘args’}, *(\mathit{str}))}, \boxed{\mathrm{sel}\,(\text{‘rhs’}, \mathit{Expr})}\right]\right)\right)$
- **anonymize-deanonymize**
  $\mathrm{p}\left(\text{‘’}, \mathit{Binary}, \mathrm{seq}\left(\left[\boxed{\mathrm{sel}\,(\text{‘ops’}, \mathit{Ops})}, \boxed{\mathrm{sel}\,(\text{‘left’}, \mathit{Expr})}, \boxed{\mathrm{sel}\,(\text{‘right’}, \mathit{Expr})}\right]\right)\right)$
- **vertical-horizontal** in *Expr*
- **eliminate-introduce**
  $\mathrm{p}\,(\text{‘’}, \mathit{Visitor}, \varphi)$
- **undefine-define**
  $\mathrm{p}\,(\text{‘’}, \mathit{Ops}, \varepsilon)$
- **unchain-chain**
  $\mathrm{p}\,(\text{‘’}, \mathit{Expr}, \mathit{Apply})$
- **unchain-chain**
  $\mathrm{p}\,(\text{‘’}, \mathit{Expr}, \mathit{Argument})$
- **unchain-chain**
  $\mathrm{p}\,(\text{‘’}, \mathit{Expr}, \mathit{Binary})$
- **unchain-chain**
  $\mathrm{p}\,(\text{‘’}, \mathit{Expr}, \mathit{IfThenElse})$
- **unchain-chain**
  $\mathrm{p}\,(\text{‘’}, \mathit{Expr}, \mathit{Literal})$
- **unlabel-designate**
  $\mathrm{p}\left(\boxed{\text{‘Apply’}}, \mathit{Expr}, \mathrm{seq}\,([\mathit{str}, *(\mathit{Expr})])\right)$
- **unlabel-designate**
  $\mathrm{p}\left(\boxed{\text{‘Argument’}}, \mathit{Expr}, \mathit{str}\right)$
- **unlabel-designate**
  $\mathrm{p}\left(\boxed{\text{‘Binary’}}, \mathit{Expr}, \mathrm{seq}\,([\mathit{Ops}, \mathit{Expr}, \mathit{Expr}])\right)$
- **unlabel-designate**
  $\mathrm{p}\left(\boxed{\text{‘IfThenElse’}}, \mathit{Expr}, \mathrm{seq}\,([\mathit{Expr}, \mathit{Expr}, \mathit{Expr}])\right)$
- **unlabel-designate**
  $\mathrm{p}\left(\boxed{\text{‘Literal’}}, \mathit{Expr}, \mathit{int}\right)$
- **extract-inline** in *Expr*
  $\mathrm{p}\,(\text{‘’}, \mathit{Expr}_{\mathit{1}}, \mathrm{seq}\,([\mathit{str}, *(\mathit{Expr})]))$
- **extract-inline** in *Expr*
  $\mathrm{p}\,(\text{‘’}, \mathit{Expr}_{\mathit{2}}, \mathrm{seq}\,([\mathit{Ops}, \mathit{Expr}, \mathit{Expr}]))$
- **extract-inline** in *Expr*
  $\mathrm{p}\,(\text{‘’}, \mathit{Expr}_{\mathit{3}}, \mathrm{seq}\,([\mathit{Expr}, \mathit{Expr}, \mathit{Expr}]))$

5.3 Grammar in ANF

Production rule	Production signature
$\text{p}\,(`', \mathit{Expr}, \mathit{Expr_1})$	$\{\langle \mathit{Expr_1}, 1\rangle\}$
$\text{p}\,(`', \mathit{Expr}, \mathit{str})$	$\{\langle \mathit{str}, 1\rangle\}$
$\text{p}\,(`', \mathit{Expr}, \mathit{Expr_2})$	$\{\langle \mathit{Expr_2}, 1\rangle\}$
$\text{p}\,(`', \mathit{Expr}, \mathit{Expr_3})$	$\{\langle \mathit{Expr_3}, 1\rangle\}$
$\text{p}\,(`', \mathit{Expr}, \mathit{int})$	$\{\langle \mathit{int}, 1\rangle\}$
$\text{p}\,(`', \mathit{Function}, \text{seq}\,([\mathit{str}, *(\mathit{str}), \mathit{Expr}]))$	$\{\langle \mathit{Expr}, 1\rangle, \langle \mathit{str}, 1*\rangle\}$
$\text{p}\,(`', \mathit{Program}, *(\mathit{Function}))$	$\{\langle \mathit{Function}, *\rangle\}$
$\text{p}\,(`', \mathit{Expr_1}, \text{seq}\,([\mathit{str}, *(\mathit{Expr})]))$	$\{\langle \mathit{str}, 1\rangle, \langle \mathit{Expr}, *\rangle\}$
$\text{p}\,(`', \mathit{Expr_2}, \text{seq}\,([\mathit{Ops}, \mathit{Expr}, \mathit{Expr}]))$	$\{\langle \mathit{Ops}, 1\rangle, \langle \mathit{Expr}, 11\rangle\}$
$\text{p}\,(`', \mathit{Expr_3}, \text{seq}\,([\mathit{Expr}, \mathit{Expr}, \mathit{Expr}]))$	$\{\langle \mathit{Expr}, 111\rangle\}$

5.4 Nominal resolution

Production rules are matched as follows (ANF on the left, master grammar on the right):

$$\begin{aligned}
\text{p}\,(`', \mathit{Expr}, \mathit{Expr_1}) &\doteqdot \text{p}\,(`', \mathit{expression}, \mathit{apply})\\
\text{p}\,(`', \mathit{Expr}, \mathit{str}) &\doteqdot \text{p}\,(`', \mathit{expression}, \mathit{str})\\
\text{p}\,(`', \mathit{Expr}, \mathit{Expr_2}) &\doteqdot \text{p}\,(`', \mathit{expression}, \mathit{binary})\\
\text{p}\,(`', \mathit{Expr}, \mathit{Expr_3}) &\doteqdot \text{p}\,(`', \mathit{expression}, \mathit{conditional})\\
\text{p}\,(`', \mathit{Expr}, \mathit{int}) &\doteqdot \text{p}\,(`', \mathit{expression}, \mathit{int})\\
\text{p}\,(`', \mathit{Function}, \text{seq}\,([\mathit{str}, *(\mathit{str}), \mathit{Expr}])) &\Leftrightarrow \text{p}\,(`', \mathit{function}, \text{seq}\,([\mathit{str}, +(\mathit{str}), \mathit{expression}]))\\
\text{p}\,(`', \mathit{Program}, *(\mathit{Function})) &\Leftrightarrow \text{p}\,(`', \mathit{program}, +(\mathit{function}))\\
\text{p}\,(`', \mathit{Expr_1}, \text{seq}\,([\mathit{str}, *(\mathit{Expr})])) &\Leftrightarrow \text{p}\,(`', \mathit{apply}, \text{seq}\,([\mathit{str}, +(\mathit{expression})]))\\
\text{p}\,(`', \mathit{Expr_2}, \text{seq}\,([\mathit{Ops}, \mathit{Expr}, \mathit{Expr}])) &\Leftrightarrow \text{p}\,(`', \mathit{binary}, \text{seq}\,([\mathit{expression}, \mathit{operator}, \mathit{expression}]))\\
\text{p}\,(`', \mathit{Expr_3}, \text{seq}\,([\mathit{Expr}, \mathit{Expr}, \mathit{Expr}])) &\doteqdot \text{p}\,(`', \mathit{conditional}, \text{seq}\,([\mathit{expression}, \mathit{expression}, \mathit{expression}]))
\end{aligned}$$

This yields the following nominal mapping:

$$\begin{aligned}
om \diamond master = \{&\langle \mathit{Expr_2}, \mathit{binary}\rangle,\\
&\langle \mathit{Expr_3}, \mathit{conditional}\rangle,\\
&\langle \mathit{int}, \mathit{int}\rangle,\\
&\langle \mathit{Function}, \mathit{function}\rangle,\\
&\langle \mathit{str}, \mathit{str}\rangle,\\
&\langle \mathit{Program}, \mathit{program}\rangle,\\
&\langle \mathit{Expr}, \mathit{expression}\rangle,\\
&\langle \mathit{Expr_1}, \mathit{apply}\rangle,\\
&\langle \mathit{Ops}, \mathit{operator}\rangle\}
\end{aligned}$$

Which is exercised with these grammar transformation steps:

- **renameN-renameN** $\mathit{Expr_2}$ to *binary*
- **renameN-renameN** $\mathit{Expr_3}$ to *conditional*
- **renameN-renameN** *Function* to *function*
- **renameN-renameN** *Program* to *program*
- **renameN-renameN** *Expr* to *expression*
- **renameN-renameN** $\mathit{Expr_1}$ to *apply*
- **renameN-renameN** *Ops* to *operator*

5.5 Structural resolution

- **narrow-widen** in *function*
  $*(\mathit{str})$
  $+(\mathit{str})$

- **narrow-widen** in *program*
  $*(\mathit{function})$
  $+(\mathit{function})$
- **narrow-widen** in *apply*
  $*(\mathit{expression})$
  $+(\mathit{expression})$
- **permute-permute**
  $\mathrm{p}\,(`', \mathit{binary}, \mathrm{seq}\,([\mathit{operator}, \mathit{expression}, \mathit{expression}]))$
  $\mathrm{p}\,(`', \mathit{binary}, \mathrm{seq}\,([\mathit{expression}, \mathit{operator}, \mathit{expression}]))$

Grammar 6

PyParsing in Python

Source name: **python**

6.1 Source grammar

- Source artifact: topics/fl/python/parser.py
- Grammar extractor: shared/rascal/src/extract/Python2BGF.rsc

Production rules
p('', *_Literal*, *Literal*)
p('', *_IF*, 'if')
p('', *_THEN*, 'then')
p('', *_ELSE*, 'else')
p('', *name*, *str*)
p('', *literal*, seq ([?('-') , *int*]))
p('', *atom*, choice([*name*, *literal*, seq (['(', *expr*, ')'])]))
p('', *ifThenElse*, seq ([*_IF*, *expr*, *_THEN*, *expr*, *_ELSE*, *expr*]))
p('', *operators*, choice(['==', '+', '-']))
p('', *binary*, seq ([*atom*, ∗(seq ([*operators*, *atom*]))]))
p('', *apply*, seq ([*name*, +(*atom*)]))
p('', *expr*, choice([*binary*, *apply*, *ifThenElse*]))
p('', *function*, seq ([*name*, +(*name*) , '=', *expr*]))
p('', *program*, seq ([+(*function*) , *StringEnd*]))

6.2 Mutations

- **unite-splitN** *expr*
  p ('', *atom*, choice ([*name*, *literal*, seq (['(', *expr*, ')'])]))
- **designate-unlabel**
  p ('tmplabel', *binary*, seq ([*expr*, ∗(seq ([*operators*, *expr*]))]))
- **assoc-iterate**
  p ('tmplabel', *binary*, seq ([*expr*, *operators*, *expr*]))
- **unlabel-designate**
  p ('tmplabel', *binary*, seq ([*expr*, *operators*, *expr*]))

6.3 Normalizations

- **reroot-reroot** [] to [*program*]
- **abstractize-concretize**
  $\mathrm{p}\left(`', \mathit{literal}, \mathrm{seq}\left(\left[?\left(\boxed{\text{`-'}}\right), \mathit{int}\right]\right)\right)$
- **abstractize-concretize**
  $\mathrm{p}\left(`', \mathit{operators}, \mathrm{choice}\left(\left[\boxed{\text{`=='}}, \boxed{\text{`+'}}, \boxed{\text{`-'}}\right]\right)\right)$
- **abstractize-concretize**
  $\mathrm{p}\left(`', \_\mathit{IF}, \boxed{\text{`if'}}\right)$
- **abstractize-concretize**
  $\mathrm{p}\left(`', \mathit{expr}, \mathrm{choice}\left(\left[\mathit{name}, \mathit{literal}, \mathrm{seq}\left(\left[\boxed{\text{`('}}, \mathit{expr}, \boxed{\text{`)'}}\right]\right)\right]\right)\right)$
- **abstractize-concretize**
  $\mathrm{p}\left(`', \_\mathit{ELSE}, \boxed{\text{`else'}}\right)$
- **abstractize-concretize**
  $\mathrm{p}\left(`', \_\mathit{THEN}, \boxed{\text{`then'}}\right)$
- **abstractize-concretize**
  $\mathrm{p}\left(`', \mathit{function}, \mathrm{seq}\left(\left[\mathit{name}, +(\mathit{name}), \boxed{\text{`='}}, \mathit{expr}\right]\right)\right)$
- **vertical-horizontal** in *expr*
- **undefine-define**
  $\mathrm{p}(`', \_\mathit{IF}, \varepsilon)$
- **undefine-define**
  $\mathrm{p}(`', \_\mathit{THEN}, \varepsilon)$
- **undefine-define**
  $\mathrm{p}(`', \_\mathit{ELSE}, \varepsilon)$
- **undefine-define**
  $\mathrm{p}(`', \mathit{operators}, \varepsilon)$
- **unchain-chain**
  $\mathrm{p}(`', \mathit{expr}, \mathit{literal})$
- **abridge-detour**
  $\mathrm{p}(`', \mathit{expr}, \mathit{expr})$
- **unchain-chain**
  $\mathrm{p}(`', \mathit{expr}, \mathit{binary})$
- **unchain-chain**
  $\mathrm{p}(`', \mathit{expr}, \mathit{apply})$
- **unchain-chain**
  $\mathrm{p}(`', \mathit{expr}, \mathit{ifThenElse})$
- **inline-extract**
  $\mathrm{p}(`', \mathit{name}, \mathit{str})$
- **unlabel-designate**
  $\mathrm{p}\left(\boxed{\text{`literal'}}, \mathit{expr}, \mathit{int}\right)$
- **unlabel-designate**
  $\mathrm{p}\left(\boxed{\text{`ifThenElse'}}, \mathit{expr}, \mathrm{seq}\left([\_\mathit{IF}, \mathit{expr}, \_\mathit{THEN}, \mathit{expr}, \_\mathit{ELSE}, \mathit{expr}]\right)\right)$
- **unlabel-designate**
  $\mathrm{p}\left(\boxed{\text{`binary'}}, \mathit{expr}, \mathrm{seq}\left([\mathit{expr}, \mathit{operators}, \mathit{expr}]\right)\right)$
- **unlabel-designate**
  $\mathrm{p}\left(\boxed{\text{`apply'}}, \mathit{expr}, \mathrm{seq}\left([\mathit{str}, +(\mathit{expr})]\right)\right)$
- **extract-inline** in *expr*
  $\mathrm{p}\left(`', \mathit{expr}_{1}, \mathrm{seq}\left([\_\mathit{IF}, \mathit{expr}, \_\mathit{THEN}, \mathit{expr}, \_\mathit{ELSE}, \mathit{expr}]\right)\right)$
- **extract-inline** in *expr*
  $\mathrm{p}\left(`', \mathit{expr}_{2}, \mathrm{seq}\left([\mathit{expr}, \mathit{operators}, \mathit{expr}]\right)\right)$
- **extract-inline** in *expr*
  $\mathrm{p}\left(`', \mathit{expr}_{3}, \mathrm{seq}\left([\mathit{str}, +(\mathit{expr})]\right)\right)$

6.4 Grammar in ANF

Production rule	Production signature
$p(``, \_Literal, Literal)$	$\{\langle Literal, 1\rangle\}$
$p(``, expr, int)$	$\{\langle int, 1\rangle\}$
$p(``, expr, str)$	$\{\langle str, 1\rangle\}$
$p(``, expr, expr_1)$	$\{\langle expr_1, 1\rangle\}$
$p(``, expr, expr_2)$	$\{\langle expr_2, 1\rangle\}$
$p(``, expr, expr_3)$	$\{\langle expr_3, 1\rangle\}$
$p(``, function, seq([str, +(str), expr]))$	$\{\langle str, 1+\rangle, \langle expr, 1\rangle\}$
$p(``, program, seq([+(function), StringEnd]))$	$\{\langle function, +\rangle, \langle StringEnd, 1\rangle\}$
$p(``, expr_1, seq([\_IF, expr, \_THEN, expr, \_ELSE, expr]))$	$\{\langle \_IF, 1\rangle, \langle \_THEN, 1\rangle, \langle expr, 111\rangle, \langle \_ELSE, 1\rangle\}$
$p(``, expr_2, seq([expr, operators, expr]))$	$\{\langle expr, 11\rangle, \langle operators, 1\rangle\}$
$p(``, expr_3, seq([str, +(expr)]))$	$\{\langle str, 1\rangle, \langle expr, +\rangle\}$

6.5 Nominal resolution

Production rules are matched as follows (ANF on the left, master grammar on the right):

$$
\begin{array}{rcl}
p(``, \_Literal, Literal) & & \varnothing \\
p(``, expr, int) & \doteqdot & p(``, expression, int) \\
p(``, expr, str) & \doteqdot & p(``, expression, str) \\
p(``, expr, expr_1) & \doteqdot & p(``, expression, conditional) \\
p(``, expr, expr_2) & \doteqdot & p(``, expression, binary) \\
p(``, expr, expr_3) & \doteqdot & p(``, expression, apply) \\
p(``, function, seq([str, +(str), expr])) & \doteqdot & p(``, function, seq([str, +(str), expression])) \\
p(``, program, seq([+(function), StringEnd])) & \doteqdot & p(``, program, +(function)) \\
p(``, expr_1, seq([\_IF, expr, \_THEN, expr, \_ELSE, expr])) & \doteqdot & p(``, conditional, seq([expression, expression, expression])) \\
p(``, expr_2, seq([expr, operators, expr])) & \doteqdot & p(``, binary, seq([expression, operator, expression])) \\
p(``, expr_3, seq([str, +(expr)])) & \doteqdot & p(``, apply, seq([str, +(expression)]))
\end{array}
$$

This yields the following nominal mapping:

$$
\begin{aligned}
python \diamond master = \{ & \langle expr_2, binary\rangle, \\
& \langle program, program\rangle, \\
& \langle function, function\rangle, \\
& \langle expr_1, conditional\rangle, \\
& \langle expr, expression\rangle, \\
& \langle str, str\rangle, \\
& \langle int, int\rangle, \\
& \langle StringEnd, \omega\rangle, \\
& \langle \_ELSE, \omega\rangle, \\
& \langle \_IF, \omega\rangle, \\
& \langle expr_3, apply\rangle, \\
& \langle \_THEN, \omega\rangle, \\
& \langle operators, operator\rangle\}
\end{aligned}
$$

Which is exercised with these grammar transformation steps:

- **renameN-renameN** $expr_2$ to *binary*
- **renameN-renameN** $expr_1$ to *conditional*
- **renameN-renameN** *expr* to *expression*
- **renameN-renameN** $expr_3$ to *apply*
- **renameN-renameN** *operators* to *operator*

6.6 Structural resolution

- **project-inject**
  $\mathrm{p}\left(`', \mathit{program}, \mathrm{seq}\left(\left[+(\mathit{function}), \boxed{\mathit{StringEnd}}\right]\right)\right)$
- **project-inject**
  $\mathrm{p}\left(`', \mathit{conditional}, \mathrm{seq}\left(\left[\_\mathit{IF}, \mathit{expression}, \_\mathit{THEN}, \mathit{expression}, \boxed{\_\mathit{ELSE}}, \mathit{expression}\right]\right)\right)$
- **project-inject**
  $\mathrm{p}\left(`', \mathit{conditional}, \mathrm{seq}\left(\left[\boxed{\_\mathit{IF}}, \mathit{expression}, \_\mathit{THEN}, \mathit{expression}, \mathit{expression}\right]\right)\right)$
- **project-inject**
  $\mathrm{p}\left(`', \mathit{conditional}, \mathrm{seq}\left(\left[\mathit{expression}, \boxed{\_\mathit{THEN}}, \mathit{expression}, \mathit{expression}\right]\right)\right)$
- **eliminate-introduce**
  $\mathrm{p}\left(`', \_\mathit{Literal}, \mathit{Literal}\right)$

Grammar 7

Rascal Algebraic Data Type

Source name: **rascal-a**

7.1 Source grammar

- Source artifact: topics/fl/rascal/Abstract.rsc
- Grammar extractor: shared/rascal/src/extract/RascalADT2BGF.rsc

Production rules
p('prg', $FLPrg$, sel ('fs', $*(FLFun)$))
p('fun', $FLFun$, seq ([sel ('f', str) , sel ('args', $*(str)$) , sel ('body', $FLExpr$)]))
p('', $FLExpr$, choice([sel ('binary', seq ([sel ('e1', $FLExpr$) , sel ('op', $FLOp$) , sel ('e2', $FLExpr$)])) ,
sel ('apply', seq ([sel ('f', str) , sel ('vargs', $*(FLExpr)$)])) ,
sel ('ifThenElse', seq ([sel ('c', $FLExpr$) , sel ('t', $FLExpr$) , sel ('e', $FLExpr$)])) ,
sel ('argument', sel ('a', str)) ,
sel ('literal', sel ('i', int))]))
p('', $FLOp$, choice([sel ('minus', ε) ,
sel ('plus', ε) ,
sel ('equal', ε)]))

7.2 Normalizations

- **reroot-reroot** [] to [$FLPrg$]
- **unlabel-designate**
  p ($\boxed{\text{'prg'}}$, $FLPrg$, sel ('fs', $*(FLFun)$))
- **unlabel-designate**
  p ($\boxed{\text{'fun'}}$, $FLFun$, seq ([sel ('f', str) , sel ('args', $*(str)$) , sel ('body', $FLExpr$)]))
- **anonymize-deanonymize**
  p ('', $FLOp$, choice ([$\boxed{\text{sel ('minus', }\varepsilon\text{)}}$, $\boxed{\text{sel ('plus', }\varepsilon\text{)}}$, $\boxed{\text{sel ('equal', }\varepsilon\text{)}}$]))
- **anonymize-deanonymize**
  p ('', $FLExpr$, choice ([$\boxed{\text{sel ('binary', seq ([}\boxed{\text{sel ('e1', } FLExpr\text{)}}, \boxed{\text{sel ('op', } FLOp\text{)}}, \boxed{\text{sel ('e2', } FLExpr\text{)}}\text{]))}}$, $\boxed{\text{sel ('apply', seq (}}$
- **anonymize-deanonymize**
  p ('', $FLFun$, seq ([$\boxed{\text{sel ('f', } str\text{)}}$, $\boxed{\text{sel ('args', } *(str)\text{)}}$, $\boxed{\text{sel ('body', } FLExpr\text{)}}$]))
- **vertical-horizontal** in $FLExpr$
- **undefine-define**
  p ('', $FLOp$, ε)
- **unlabel-designate**
  p ($\boxed{\text{'fs'}}$, $FLPrg$, $*(FLFun)$)

- **extract-inline** in $FLExpr$
  $\mathrm{p}\,(`', FLExpr_1, \mathrm{seq}\,([FLExpr, FLOp, FLExpr]))$
- **extract-inline** in $FLExpr$
  $\mathrm{p}\,(`', FLExpr_2, \mathrm{seq}\,([str, *(FLExpr)]))$
- **extract-inline** in $FLExpr$
  $\mathrm{p}\,(`', FLExpr_3, \mathrm{seq}\,([FLExpr, FLExpr, FLExpr]))$

7.3 Grammar in ANF

Production rule	Production signature
$\mathrm{p}\,(`', FLPrg, *(FLFun))$	$\{\langle FLFun, *\rangle\}$
$\mathrm{p}\,(`', FLFun, \mathrm{seq}\,([str, *(str)\,, FLExpr]))$	$\{\langle str, 1*\rangle, \langle FLExpr, 1\rangle\}$
$\mathrm{p}\,(`', FLExpr, FLExpr_1)$	$\{\langle FLExpr_1, 1\rangle\}$
$\mathrm{p}\,(`', FLExpr, FLExpr_2)$	$\{\langle FLExpr_2, 1\rangle\}$
$\mathrm{p}\,(`', FLExpr, FLExpr_3)$	$\{\langle FLExpr_3, 1\rangle\}$
$\mathrm{p}\,(`', FLExpr, str)$	$\{\langle str, 1\rangle\}$
$\mathrm{p}\,(`', FLExpr, int)$	$\{\langle int, 1\rangle\}$
$\mathrm{p}\,(`', FLExpr_1, \mathrm{seq}\,([FLExpr, FLOp, FLExpr]))$	$\{\langle FLOp, 1\rangle, \langle FLExpr, 11\rangle\}$
$\mathrm{p}\,(`', FLExpr_2, \mathrm{seq}\,([str, *(FLExpr)]))$	$\{\langle str, 1\rangle, \langle FLExpr, *\rangle\}$
$\mathrm{p}\,(`', FLExpr_3, \mathrm{seq}\,([FLExpr, FLExpr, FLExpr]))$	$\{\langle FLExpr, 111\rangle\}$

7.4 Nominal resolution

Production rules are matched as follows (ANF on the left, master grammar on the right):

$$\begin{aligned}
\mathrm{p}\,(`', FLPrg, *(FLFun)) &\approx \mathrm{p}\,(`', program, +(function))\\
\mathrm{p}\,(`', FLFun, \mathrm{seq}\,([str, *(str)\,, FLExpr])) &\approx \mathrm{p}\,(`', function, \mathrm{seq}\,([str, +(str)\,, expression]))\\
\mathrm{p}\,(`', FLExpr, FLExpr_1) &\doteq \mathrm{p}\,(`', expression, binary)\\
\mathrm{p}\,(`', FLExpr, FLExpr_2) &\doteq \mathrm{p}\,(`', expression, apply)\\
\mathrm{p}\,(`', FLExpr, FLExpr_3) &\doteq \mathrm{p}\,(`', expression, conditional)\\
\mathrm{p}\,(`', FLExpr, str) &\doteq \mathrm{p}\,(`', expression, str)\\
\mathrm{p}\,(`', FLExpr, int) &\doteq \mathrm{p}\,(`', expression, int)\\
\mathrm{p}\,(`', FLExpr_1, \mathrm{seq}\,([FLExpr, FLOp, FLExpr])) &\doteq \mathrm{p}\,(`', binary, \mathrm{seq}\,([expression, operator, expression]))\\
\mathrm{p}\,(`', FLExpr_2, \mathrm{seq}\,([str, *(FLExpr)])) &\approx \mathrm{p}\,(`', apply, \mathrm{seq}\,([str, +(expression)]))\\
\mathrm{p}\,(`', FLExpr_3, \mathrm{seq}\,([FLExpr, FLExpr, FLExpr])) &\doteq \mathrm{p}\,(`', conditional, \mathrm{seq}\,([expression, expression, expression]))
\end{aligned}$$

This yields the following nominal mapping:

$$\begin{aligned}
rascal - a \diamond master = \{&\langle FLFun, function\rangle,\\
&\langle FLExpr_2, apply\rangle,\\
&\langle FLPrg, program\rangle,\\
&\langle FLExpr, expression\rangle,\\
&\langle int, int\rangle,\\
&\langle str, str\rangle,\\
&\langle FLExpr_3, conditional\rangle,\\
&\langle FLOp, operator\rangle,\\
&\langle FLExpr_1, binary\rangle\}
\end{aligned}$$

Which is exercised with these grammar transformation steps:

- **renameN-renameN** $FLFun$ to $function$
- **renameN-renameN** $FLExpr_2$ to $apply$
- **renameN-renameN** $FLPrg$ to $program$
- **renameN-renameN** $FLExpr$ to $expression$
- **renameN-renameN** $FLExpr_3$ to $conditional$

- **renameN-renameN** *FLOp* to *operator*
- **renameN-renameN** $FLExpr_1$ to *binary*

7.5 Structural resolution

- **narrow-widen** in *program*
  ∗(*function*)
  +(*function*)
- **narrow-widen** in *function*
  ∗(*str*)
  +(*str*)
- **narrow-widen** in *apply*
  ∗(*expression*)
  +(*expression*)

Grammar 8

Rascal Concrete Syntax Definition

Source name: **rascal-c**

8.1 Source grammar

- Source artifact: topics/fl/rascal/Concrete.rsc
- Grammar extractor: shared/rascal/src/extract/RascalSyntax2BGF.rsc

Production rules
p('prg', *Program*, sel ('functions', s+ (*Function*, ↙)))
p('ifThenElse', *Expr*, seq (['if', sel ('cond', *Expr*) , 'then', sel ('thenbranch', *Expr*) , 'else', sel ('elsebranch', *Expr*)]))
p('', *Expr*, seq (['(', sel ('e', *Expr*) , ')']))
p('literal', *Expr*, sel ('i', *Int*))
p('argument', *Expr*, sel ('a', *Name*))
p('binary', *Expr*, seq ([sel ('lexpr', *Expr*) , sel ('op', *Ops*) , sel ('rexpr', *Expr*)]))
p('apply', *Expr*, seq ([sel ('f', *Name*) , sel ('vargs', +(*Expr*))]))
p('plus', *Ops*, '+')
p('equal', *Ops*, '==')
p('minus', *Ops*, '-')
p('fun', *Function*, seq ([sel ('f', *Name*) , sel ('args', +(*Name*)) , '=', sel ('body', *Expr*)]))

8.2 Normalizations

- **reroot-reroot** [] to [*Program*]
- **unlabel-designate**
  p ('prg', *Program*, sel ('functions', s+ (*Function*, ↙)))
- **unlabel-designate**
  p ('ifThenElse', *Expr*, seq (['if', sel ('cond', *Expr*) , 'then', sel ('thenbranch', *Expr*) , 'else', sel ('elsebranch', *Expr*)]))
- **unlabel-designate**
  p ('literal', *Expr*, sel ('i', *Int*))
- **unlabel-designate**
  p ('argument', *Expr*, sel ('a', *Name*))
- **unlabel-designate**
  p ('binary', *Expr*, seq ([sel ('lexpr', *Expr*) , sel ('op', *Ops*) , sel ('rexpr', *Expr*)]))
- **unlabel-designate**
  p ('apply', *Expr*, seq ([sel ('f', *Name*) , sel ('vargs', +(*Expr*))]))
- **unlabel-designate**
  p ('plus', *Ops*, '+')

- **unlabel-designate**
  $\mathrm{p}\left(\boxed{\text{‘equal’}}, \mathit{Ops}, \text{‘==’}\right)$
- **unlabel-designate**
  $\mathrm{p}\left(\boxed{\text{‘minus’}}, \mathit{Ops}, \text{‘-’}\right)$
- **unlabel-designate**
  $\mathrm{p}\left(\boxed{\text{‘fun’}}, \mathit{Function}, \mathrm{seq}\left([\mathrm{sel}\left(\text{‘f’}, \mathit{Name}\right), \mathrm{sel}\left(\text{‘args’}, +(\mathit{Name})\right), \text{‘=’}, \mathrm{sel}\left(\text{‘body’}, \mathit{Expr}\right)]\right)\right)$
- **anonymize-deanonymize**
  $\mathrm{p}\left(\text{‘’}, \mathit{Expr}, \mathrm{seq}\left(\left[\boxed{\mathrm{sel}\left(\text{‘f’}, \mathit{Name}\right)}, \boxed{\mathrm{sel}\left(\text{‘vargs’}, +(\mathit{Expr})\right)}\right]\right)\right)$
- **anonymize-deanonymize**
  $\mathrm{p}\left(\text{‘’}, \mathit{Function}, \mathrm{seq}\left(\left[\boxed{\mathrm{sel}\left(\text{‘f’}, \mathit{Name}\right)}, \boxed{\mathrm{sel}\left(\text{‘args’}, +(\mathit{Name})\right)}, \text{‘=’}, \boxed{\mathrm{sel}\left(\text{‘body’}, \mathit{Expr}\right)}\right]\right)\right)$
- **anonymize-deanonymize**
  $\mathrm{p}\left(\text{‘’}, \mathit{Expr}, \mathrm{seq}\left(\left[\text{‘if’}, \boxed{\mathrm{sel}\left(\text{‘cond’}, \mathit{Expr}\right)}, \text{‘then’}, \boxed{\mathrm{sel}\left(\text{‘thenbranch’}, \mathit{Expr}\right)}, \text{‘else’}, \boxed{\mathrm{sel}\left(\text{‘elsebranch’}, \mathit{Expr}\right)}\right]\right)\right)$
- **anonymize-deanonymize**
  $\mathrm{p}\left(\text{‘’}, \mathit{Expr}, \mathrm{seq}\left(\left[\boxed{\mathrm{sel}\left(\text{‘lexpr’}, \mathit{Expr}\right)}, \boxed{\mathrm{sel}\left(\text{‘op’}, \mathit{Ops}\right)}, \boxed{\mathrm{sel}\left(\text{‘rexpr’}, \mathit{Expr}\right)}\right]\right)\right)$
- **anonymize-deanonymize**
  $\mathrm{p}\left(\text{‘’}, \mathit{Expr}, \mathrm{seq}\left(\left[\text{‘(’}, \boxed{\mathrm{sel}\left(\text{‘e’}, \mathit{Expr}\right)}, \text{‘)’}\right]\right)\right)$
- **abstractize-concretize**
  $\mathrm{p}\left(\text{‘’}, \mathit{Expr}, \mathrm{seq}\left(\left[\boxed{\text{‘(’}}, \mathit{Expr}, \boxed{\text{‘)’}}\right]\right)\right)$
- **abstractize-concretize**
  $\mathrm{p}\left(\text{‘’}, \mathit{Function}, \mathrm{seq}\left(\left[\mathit{Name}, +(\mathit{Name}), \boxed{\text{‘=’}}, \mathit{Expr}\right]\right)\right)$
- **abstractize-concretize**
  $\mathrm{p}\left(\text{‘’}, \mathit{Ops}, \boxed{\text{‘-’}}\right)$
- **abstractize-concretize**
  $\mathrm{p}\left(\text{‘’}, \mathit{Ops}, \boxed{\text{‘+’}}\right)$
- **abstractize-concretize**
  $\mathrm{p}\left(\text{‘’}, \mathit{Ops}, \boxed{\text{‘==’}}\right)$
- **abstractize-concretize**
  $\mathrm{p}\left(\text{‘functions’}, \mathit{Program}, \mathrm{s}+\left(\mathit{Function}, \boxed{\swarrow}\right)\right)$
- **abstractize-concretize**
  $\mathrm{p}\left(\text{‘’}, \mathit{Expr}, \mathrm{seq}\left(\left[\boxed{\text{‘if’}}, \mathit{Expr}, \boxed{\text{‘then’}}, \mathit{Expr}, \boxed{\text{‘else’}}, \mathit{Expr}\right]\right)\right)$
- **undefine-define**
  $\mathrm{p}\left(\text{‘’}, \mathit{Ops}, \varepsilon\right)$
- **abridge-detour**
  $\mathrm{p}\left(\text{‘’}, \mathit{Expr}, \mathit{Expr}\right)$
- **unlabel-designate**
  $\mathrm{p}\left(\boxed{\text{‘functions’}}, \mathit{Program}, +(\mathit{Function})\right)$
- **unlabel-designate**
  $\mathrm{p}\left(\boxed{\text{‘i’}}, \mathit{Expr}, \mathit{Int}\right)$
- **unlabel-designate**
  $\mathrm{p}\left(\boxed{\text{‘a’}}, \mathit{Expr}, \mathit{Name}\right)$
- **extract-inline** in *Expr*
  $\mathrm{p}\left(\text{‘’}, \mathit{Expr}_{\mathit{1}}, \mathrm{seq}\left([\mathit{Expr}, \mathit{Expr}, \mathit{Expr}]\right)\right)$
- **extract-inline** in *Expr*
  $\mathrm{p}\left(\text{‘’}, \mathit{Expr}_{\mathit{2}}, \mathrm{seq}\left([\mathit{Expr}, \mathit{Ops}, \mathit{Expr}]\right)\right)$
- **extract-inline** in *Expr*
  $\mathrm{p}\left(\text{‘’}, \mathit{Expr}_{\mathit{3}}, \mathrm{seq}\left([\mathit{Name}, +(\mathit{Expr})]\right)\right)$

8.3 Grammar in ANF

Production rule	Production signature
$\mathrm{p}\,(\text{`'}, \mathit{Program}, +(\mathit{Function}))$	$\{\langle \mathit{Function}, +\rangle\}$
$\mathrm{p}\,(\text{`'}, \mathit{Expr}, \mathit{Expr_1})$	$\{\langle \mathit{Expr_1}, 1\rangle\}$
$\mathrm{p}\,(\text{`'}, \mathit{Expr}, \mathit{Int})$	$\{\langle \mathit{Int}, 1\rangle\}$
$\mathrm{p}\,(\text{`'}, \mathit{Expr}, \mathit{Name})$	$\{\langle \mathit{Name}, 1\rangle\}$
$\mathrm{p}\,(\text{`'}, \mathit{Expr}, \mathit{Expr_2})$	$\{\langle \mathit{Expr_2}, 1\rangle\}$
$\mathrm{p}\,(\text{`'}, \mathit{Expr}, \mathit{Expr_3})$	$\{\langle \mathit{Expr_3}, 1\rangle\}$
$\mathrm{p}\,(\text{`'}, \mathit{Function}, \mathrm{seq}\,([\mathit{Name}, +(\mathit{Name}), \mathit{Expr}]))$	$\{\langle \mathit{Expr}, 1\rangle, \langle \mathit{Name}, 1+\rangle\}$
$\mathrm{p}\,(\text{`'}, \mathit{Expr_1}, \mathrm{seq}\,([\mathit{Expr}, \mathit{Expr}, \mathit{Expr}]))$	$\{\langle \mathit{Expr}, 111\rangle\}$
$\mathrm{p}\,(\text{`'}, \mathit{Expr_2}, \mathrm{seq}\,([\mathit{Expr}, \mathit{Ops}, \mathit{Expr}]))$	$\{\langle \mathit{Ops}, 1\rangle, \langle \mathit{Expr}, 11\rangle\}$
$\mathrm{p}\,(\text{`'}, \mathit{Expr_3}, \mathrm{seq}\,([\mathit{Name}, +(\mathit{Expr})]))$	$\{\langle \mathit{Expr}, +\rangle, \langle \mathit{Name}, 1\rangle\}$

8.4 Nominal resolution

Production rules are matched as follows (ANF on the left, master grammar on the right):

$$
\begin{aligned}
\mathrm{p}\,(\text{`'}, \mathit{Program}, +(\mathit{Function})) &\simeq \mathrm{p}\,(\text{`'}, \mathit{program}, +(\mathit{function}))\\
\mathrm{p}\,(\text{`'}, \mathit{Expr}, \mathit{Expr_1}) &\simeq \mathrm{p}\,(\text{`'}, \mathit{expression}, \mathit{conditional})\\
\mathrm{p}\,(\text{`'}, \mathit{Expr}, \mathit{Int}) &\simeq \mathrm{p}\,(\text{`'}, \mathit{expression}, \mathit{int})\\
\mathrm{p}\,(\text{`'}, \mathit{Expr}, \mathit{Name}) &\simeq \mathrm{p}\,(\text{`'}, \mathit{expression}, \mathit{str})\\
\mathrm{p}\,(\text{`'}, \mathit{Expr}, \mathit{Expr_2}) &\simeq \mathrm{p}\,(\text{`'}, \mathit{expression}, \mathit{binary})\\
\mathrm{p}\,(\text{`'}, \mathit{Expr}, \mathit{Expr_3}) &\simeq \mathrm{p}\,(\text{`'}, \mathit{expression}, \mathit{apply})\\
\mathrm{p}\,(\text{`'}, \mathit{Function}, \mathrm{seq}\,([\mathit{Name}, +(\mathit{Name}), \mathit{Expr}])) &\simeq \mathrm{p}\,(\text{`'}, \mathit{function}, \mathrm{seq}\,([\mathit{str}, +(\mathit{str}), \mathit{expression}]))\\
\mathrm{p}\,(\text{`'}, \mathit{Expr_1}, \mathrm{seq}\,([\mathit{Expr}, \mathit{Expr}, \mathit{Expr}])) &\simeq \mathrm{p}\,(\text{`'}, \mathit{conditional}, \mathrm{seq}\,([\mathit{expression}, \mathit{expression}, \mathit{expression}]))\\
\mathrm{p}\,(\text{`'}, \mathit{Expr_2}, \mathrm{seq}\,([\mathit{Expr}, \mathit{Ops}, \mathit{Expr}])) &\simeq \mathrm{p}\,(\text{`'}, \mathit{binary}, \mathrm{seq}\,([\mathit{expression}, \mathit{operator}, \mathit{expression}]))\\
\mathrm{p}\,(\text{`'}, \mathit{Expr_3}, \mathrm{seq}\,([\mathit{Name}, +(\mathit{Expr})])) &\simeq \mathrm{p}\,(\text{`'}, \mathit{apply}, \mathrm{seq}\,([\mathit{str}, +(\mathit{expression})]))
\end{aligned}
$$

This yields the following nominal mapping:

$$
\begin{aligned}
\mathit{rascal} - c \diamond \mathit{master} = \{&\langle \mathit{Expr_2}, \mathit{binary}\rangle,\\
&\langle \mathit{Int}, \mathit{int}\rangle,\\
&\langle \mathit{Expr_1}, \mathit{conditional}\rangle,\\
&\langle \mathit{Function}, \mathit{function}\rangle,\\
&\langle \mathit{Program}, \mathit{program}\rangle,\\
&\langle \mathit{Name}, \mathit{str}\rangle,\\
&\langle \mathit{Expr_3}, \mathit{apply}\rangle,\\
&\langle \mathit{Expr}, \mathit{expression}\rangle,\\
&\langle \mathit{Ops}, \mathit{operator}\rangle\}
\end{aligned}
$$

Which is exercised with these grammar transformation steps:

- **renameN-renameN** $\mathit{Expr_2}$ to *binary*
- **renameN-renameN** *Int* to *int*
- **renameN-renameN** $\mathit{Expr_1}$ to *conditional*
- **renameN-renameN** *Function* to *function*
- **renameN-renameN** *Program* to *program*
- **renameN-renameN** *Name* to *str*
- **renameN-renameN** $\mathit{Expr_3}$ to *apply*
- **renameN-renameN** *Expr* to *expression*
- **renameN-renameN** *Ops* to *operator*

Grammar 9

Syntax Definition Formalism

Source name: **sdf**

9.1 Source grammar

- Source artifact: topics/fl/asfsdf/Syntax.sdf
- Grammar extractor: topics/extraction/sdf/Main.sdf
- Grammar extractor: topics/extraction/sdf/Main.asf
- Grammar extractor: topics/extraction/sdf/Tokens.sdf
- Grammar extractor: topics/extraction/sdf/Tokens.asf

Production rules
p(‘’, *Program*, +(*Function*))
p(‘’, *Function*, seq ([*Name*, +(*Name*) , ‘=’, *Expr*, +(*Newline*)]))
p(‘binary’, *Expr*, seq ([*Expr*, *Ops*, *Expr*]))
p(‘apply’, *Expr*, seq ([*Name*, +(*Expr*)]))
p(‘ifThenElse’, *Expr*, seq ([‘if’, *Expr*, ‘then’, *Expr*, ‘else’, *Expr*]))
p(‘’, *Expr*, seq ([‘(’, *Expr*, ‘)’]))
p(‘argument’, *Expr*, *Name*)
p(‘literal’, *Expr*, *Int*)
p(‘minus’, *Ops*, ‘-’)
p(‘plus’, *Ops*, ‘+’)
p(‘equal’, *Ops*, ‘==’)

9.2 Normalizations

- **reroot-reroot** [] to [*Program*]
- **unlabel-designate**
  p (‘binary’, *Expr*, seq ([*Expr*, *Ops*, *Expr*]))
- **unlabel-designate**
  p (‘apply’, *Expr*, seq ([*Name*, +(*Expr*)]))
- **unlabel-designate**
  p (‘ifThenElse’, *Expr*, seq ([‘if’, *Expr*, ‘then’, *Expr*, ‘else’, *Expr*]))
- **unlabel-designate**
  p (‘argument’, *Expr*, *Name*)
- **unlabel-designate**
  p (‘literal’, *Expr*, *Int*)

- **unlabel-designate**
  $\mathrm{p}\left(\boxed{\text{‘minus’}}, \mathit{Ops}, \text{‘-’}\right)$
- **unlabel-designate**
  $\mathrm{p}\left(\boxed{\text{‘plus’}}, \mathit{Ops}, \text{‘+’}\right)$
- **unlabel-designate**
  $\mathrm{p}\left(\boxed{\text{‘equal’}}, \mathit{Ops}, \text{‘==’}\right)$
- **abstractize-concretize**
  $\mathrm{p}\left(\text{‘’}, \mathit{Expr}, \mathrm{seq}\left(\left[\boxed{\text{‘(’}}, \mathit{Expr}, \boxed{\text{‘)’}}\right]\right)\right)$
- **abstractize-concretize**
  $\mathrm{p}\left(\text{‘’}, \mathit{Ops}, \boxed{\text{‘+’}}\right)$
- **abstractize-concretize**
  $\mathrm{p}\left(\text{‘’}, \mathit{Ops}, \boxed{\text{‘-’}}\right)$
- **abstractize-concretize**
  $\mathrm{p}\left(\text{‘’}, \mathit{Ops}, \boxed{\text{‘==’}}\right)$
- **abstractize-concretize**
  $\mathrm{p}\left(\text{‘’}, \mathit{Expr}, \mathrm{seq}\left(\left[\boxed{\text{‘if’}}, \mathit{Expr}, \boxed{\text{‘then’}}, \mathit{Expr}, \boxed{\text{‘else’}}, \mathit{Expr}\right]\right)\right)$
- **abstractize-concretize**
  $\mathrm{p}\left(\text{‘’}, \mathit{Function}, \mathrm{seq}\left(\left[\mathit{Name}, +(\mathit{Name}), \boxed{\text{‘=’}}, \mathit{Expr}, +(\mathit{Newline})\right]\right)\right)$
- **undefine-define**
  $\mathrm{p}\left(\text{‘’}, \mathit{Ops}, \varepsilon\right)$
- **abridge-detour**
  $\mathrm{p}\left(\text{‘’}, \mathit{Expr}, \mathit{Expr}\right)$
- **extract-inline** in *Expr*
  $\mathrm{p}\left(\text{‘’}, \mathit{Expr_1}, \mathrm{seq}\left([\mathit{Expr}, \mathit{Ops}, \mathit{Expr}]\right)\right)$
- **extract-inline** in *Expr*
  $\mathrm{p}\left(\text{‘’}, \mathit{Expr_2}, \mathrm{seq}\left([\mathit{Name}, +(\mathit{Expr})]\right)\right)$
- **extract-inline** in *Expr*
  $\mathrm{p}\left(\text{‘’}, \mathit{Expr_3}, \mathrm{seq}\left([\mathit{Expr}, \mathit{Expr}, \mathit{Expr}]\right)\right)$

9.3 Grammar in ANF

Production rule	Production signature
$\mathrm{p}(\text{‘’}, \mathit{Program}, +(\mathit{Function}))$	$\{\langle \mathit{Function}, + \rangle\}$
$\mathrm{p}(\text{‘’}, \mathit{Function}, \mathrm{seq}([\mathit{Name}, +(\mathit{Name}), \mathit{Expr}, +(\mathit{Newline})]))$	$\{\langle \mathit{Expr}, 1 \rangle, \langle \mathit{Newline}, + \rangle, \langle \mathit{Name}, 1+ \rangle\}$
$\mathrm{p}(\text{‘’}, \mathit{Expr}, \mathit{Expr_1})$	$\{\langle \mathit{Expr_1}, 1 \rangle\}$
$\mathrm{p}(\text{‘’}, \mathit{Expr}, \mathit{Expr_2})$	$\{\langle \mathit{Expr_2}, 1 \rangle\}$
$\mathrm{p}(\text{‘’}, \mathit{Expr}, \mathit{Expr_3})$	$\{\langle \mathit{Expr_3}, 1 \rangle\}$
$\mathrm{p}(\text{‘’}, \mathit{Expr}, \mathit{Name})$	$\{\langle \mathit{Name}, 1 \rangle\}$
$\mathrm{p}(\text{‘’}, \mathit{Expr}, \mathit{Int})$	$\{\langle \mathit{Int}, 1 \rangle\}$
$\mathrm{p}(\text{‘’}, \mathit{Expr_1}, \mathrm{seq}([\mathit{Expr}, \mathit{Ops}, \mathit{Expr}]))$	$\{\langle \mathit{Ops}, 1 \rangle, \langle \mathit{Expr}, 11 \rangle\}$
$\mathrm{p}(\text{‘’}, \mathit{Expr_2}, \mathrm{seq}([\mathit{Name}, +(\mathit{Expr})]))$	$\{\langle \mathit{Expr}, + \rangle, \langle \mathit{Name}, 1 \rangle\}$
$\mathrm{p}(\text{‘’}, \mathit{Expr_3}, \mathrm{seq}([\mathit{Expr}, \mathit{Expr}, \mathit{Expr}]))$	$\{\langle \mathit{Expr}, 111 \rangle\}$

9.4 Nominal resolution

Production rules are matched as follows (ANF on the left, master grammar on the right):

$$
\begin{aligned}
\mathrm{p}\left(``, \mathit{Program}, +(\mathit{Function})\right) &\quad\bumpeq\quad \mathrm{p}\left(``, \mathit{program}, +(\mathit{function})\right)\\
\mathrm{p}\left(``, \mathit{Function}, \mathrm{seq}\left([\mathit{Name}, +(\mathit{Name}), \mathit{Expr}, +(\mathit{Newline})]\right)\right) &\quad\Bumpeq\quad \mathrm{p}\left(``, \mathit{function}, \mathrm{seq}\left([\mathit{str}, +(\mathit{str}), \mathit{expression}]\right)\right)\\
\mathrm{p}\left(``, \mathit{Expr}, \mathit{Expr_1}\right) &\quad\bumpeq\quad \mathrm{p}\left(``, \mathit{expression}, \mathit{binary}\right)\\
\mathrm{p}\left(``, \mathit{Expr}, \mathit{Expr_2}\right) &\quad\bumpeq\quad \mathrm{p}\left(``, \mathit{expression}, \mathit{apply}\right)\\
\mathrm{p}\left(``, \mathit{Expr}, \mathit{Expr_3}\right) &\quad\bumpeq\quad \mathrm{p}\left(``, \mathit{expression}, \mathit{conditional}\right)\\
\mathrm{p}\left(``, \mathit{Expr}, \mathit{Name}\right) &\quad\bumpeq\quad \mathrm{p}\left(``, \mathit{expression}, \mathit{str}\right)\\
\mathrm{p}\left(``, \mathit{Expr}, \mathit{Int}\right) &\quad\bumpeq\quad \mathrm{p}\left(``, \mathit{expression}, \mathit{int}\right)\\
\mathrm{p}\left(``, \mathit{Expr_1}, \mathrm{seq}\left([\mathit{Expr}, \mathit{Ops}, \mathit{Expr}]\right)\right) &\quad\bumpeq\quad \mathrm{p}\left(``, \mathit{binary}, \mathrm{seq}\left([\mathit{expression}, \mathit{operator}, \mathit{expression}]\right)\right)\\
\mathrm{p}\left(``, \mathit{Expr_2}, \mathrm{seq}\left([\mathit{Name}, +(\mathit{Expr})]\right)\right) &\quad\bumpeq\quad \mathrm{p}\left(``, \mathit{apply}, \mathrm{seq}\left([\mathit{str}, +(\mathit{expression})]\right)\right)\\
\mathrm{p}\left(``, \mathit{Expr_3}, \mathrm{seq}\left([\mathit{Expr}, \mathit{Expr}, \mathit{Expr}]\right)\right) &\quad\bumpeq\quad \mathrm{p}\left(``, \mathit{conditional}, \mathrm{seq}\left([\mathit{expression}, \mathit{expression}, \mathit{expression}]\right)\right)
\end{aligned}
$$

This yields the following nominal mapping:

$$
\begin{aligned}
\mathit{sdf} \diamond \mathit{master} = \{&\langle \mathit{Expr_3}, \mathit{conditional}\rangle,\\
&\langle \mathit{Int}, \mathit{int}\rangle,\\
&\langle \mathit{Expr_1}, \mathit{binary}\rangle,\\
&\langle \mathit{Newline}, \omega\rangle,\\
&\langle \mathit{Function}, \mathit{function}\rangle,\\
&\langle \mathit{Program}, \mathit{program}\rangle,\\
&\langle \mathit{Name}, \mathit{str}\rangle,\\
&\langle \mathit{Expr}, \mathit{expression}\rangle,\\
&\langle \mathit{Ops}, \mathit{operator}\rangle,\\
&\langle \mathit{Expr_2}, \mathit{apply}\rangle\}
\end{aligned}
$$

Which is exercised with these grammar transformation steps:

- **renameN-renameN** $\mathit{Expr_3}$ to *conditional*
- **renameN-renameN** *Int* to *int*
- **renameN-renameN** $\mathit{Expr_1}$ to *binary*
- **renameN-renameN** *Function* to *function*
- **renameN-renameN** *Program* to *program*
- **renameN-renameN** *Name* to *str*
- **renameN-renameN** *Expr* to *expression*
- **renameN-renameN** *Ops* to *operator*
- **renameN-renameN** $\mathit{Expr_2}$ to *apply*

9.5 Structural resolution

- **project-inject**
  $\mathrm{p}\left(``, \mathit{function}, \mathrm{seq}\left(\left[\mathit{str}, +(\mathit{str}), \mathit{expression}, +(\boxed{\mathit{Newline}})\right]\right)\right)$

Grammar 10

TXL

Source name: **txl**

10.1 Source grammar

- Source artifact: topics/fl/txl/FL.Txl
- Grammar extractor: topics/extraction/txl/txl2bgf.xslt

Production rules
p(‘’, *program*, +(*fun*))
p(‘’, *fun*, seq ([*id*, +(*id*) , ‘=’, *expression*, *newline*]))
p(‘’, *expression*, choice([seq ([*expression*, *op*, *expression*]) , seq ([*id*, +(*expression*)]) , seq ([‘if’, *expression*, ‘then’, *expression*, ‘else’, *expression*]) , seq ([‘(’, *expression*, ‘)’]) , *id*, *number*]))
p(‘’, *op*, choice([‘+’, ‘-’, ‘==’]))

10.2 Normalizations

- **abstractize-concretize**
  p (‘’, *fun*, seq ([*id*, +(*id*) , ‘=’, *expression*, *newline*]))
- **abstractize-concretize**
  p (‘’, *op*, choice ([‘+’, ‘-’, ‘==’]))
- **abstractize-concretize**
  p (‘’, *expression*, choice ([seq ([*expression*, *op*, *expression*]) , seq ([*id*, +(*expression*)]) , seq ([‘if’, *expression*, ‘then’, *expressi*
- **vertical-horizontal** in *expression*
- **undefine-define**
  p (‘’, *op*, ε)
- **abridge-detour**
  p (‘’, *expression*, *expression*)
- **extract-inline** in *expression*
  p (‘’, $expression_1$, seq ([*expression*, *op*, *expression*]))
- **extract-inline** in *expression*
  p (‘’, $expression_2$, seq ([*id*, +(*expression*)]))
- **extract-inline** in *expression*
  p (‘’, $expression_3$, seq ([*expression*, *expression*, *expression*]))

10.3 Grammar in ANF

Production rule	Production signature
$\mathrm{p}(``, \mathit{program}, +(\mathit{fun}))$	$\{\langle \mathit{fun}, +\rangle\}$
$\mathrm{p}(``, \mathit{fun}, \mathrm{seq}([\mathit{id}, +(\mathit{id}), \mathit{expression}, \mathit{newline}]))$	$\{\langle \mathit{newline}, 1\rangle, \langle \mathit{id}, 1+\rangle, \langle \mathit{expression}, 1\rangle\}$
$\mathrm{p}(``, \mathit{expression}, \mathit{expression}_1)$	$\{\langle \mathit{expression}_1, 1\rangle\}$
$\mathrm{p}(``, \mathit{expression}, \mathit{expression}_2)$	$\{\langle \mathit{expression}_2, 1\rangle\}$
$\mathrm{p}(``, \mathit{expression}, \mathit{expression}_3)$	$\{\langle \mathit{expression}_3, 1\rangle\}$
$\mathrm{p}(``, \mathit{expression}, \mathit{id})$	$\{\langle \mathit{id}, 1\rangle\}$
$\mathrm{p}(``, \mathit{expression}, \mathit{number})$	$\{\langle \mathit{number}, 1\rangle\}$
$\mathrm{p}(``, \mathit{expression}_1, \mathrm{seq}([\mathit{expression}, \mathit{op}, \mathit{expression}]))$	$\{\langle \mathit{op}, 1\rangle, \langle \mathit{expression}, 11\rangle\}$
$\mathrm{p}(``, \mathit{expression}_2, \mathrm{seq}([\mathit{id}, +(\mathit{expression})]))$	$\{\langle \mathit{expression}, +\rangle, \langle \mathit{id}, 1\rangle\}$
$\mathrm{p}(``, \mathit{expression}_3, \mathrm{seq}([\mathit{expression}, \mathit{expression}, \mathit{expression}]))$	$\{\langle \mathit{expression}, 111\rangle\}$

10.4 Nominal resolution

Production rules are matched as follows (ANF on the left, master grammar on the right):

$$
\begin{array}{rcl}
\mathrm{p}(``, \mathit{program}, +(\mathit{fun})) & \simeq & \mathrm{p}(``, \mathit{program}, +(\mathit{function})) \\
\mathrm{p}(``, \mathit{fun}, \mathrm{seq}([\mathit{id}, +(\mathit{id}), \mathit{expression}, \mathit{newline}])) & \not\simeq & \mathrm{p}(``, \mathit{function}, \mathrm{seq}([\mathit{str}, +(\mathit{str}), \mathit{expression}])) \\
\mathrm{p}(``, \mathit{expression}, \mathit{expression}_1) & \simeq & \mathrm{p}(``, \mathit{expression}, \mathit{binary}) \\
\mathrm{p}(``, \mathit{expression}, \mathit{expression}_2) & \simeq & \mathrm{p}(``, \mathit{expression}, \mathit{apply}) \\
\mathrm{p}(``, \mathit{expression}, \mathit{expression}_3) & \simeq & \mathrm{p}(``, \mathit{expression}, \mathit{conditional}) \\
\mathrm{p}(``, \mathit{expression}, \mathit{id}) & \simeq & \mathrm{p}(``, \mathit{expression}, \mathit{str}) \\
\mathrm{p}(``, \mathit{expression}, \mathit{number}) & \simeq & \mathrm{p}(``, \mathit{expression}, \mathit{int}) \\
\mathrm{p}(``, \mathit{expression}_1, \mathrm{seq}([\mathit{expression}, \mathit{op}, \mathit{expression}])) & \simeq & \mathrm{p}(``, \mathit{binary}, \mathrm{seq}([\mathit{expression}, \mathit{operator}, \mathit{expression}])) \\
\mathrm{p}(``, \mathit{expression}_2, \mathrm{seq}([\mathit{id}, +(\mathit{expression})])) & \simeq & \mathrm{p}(``, \mathit{apply}, \mathrm{seq}([\mathit{str}, +(\mathit{expression})])) \\
\mathrm{p}(``, \mathit{expression}_3, \mathrm{seq}([\mathit{expression}, \mathit{expression}, \mathit{expression}])) & \simeq & \mathrm{p}(``, \mathit{conditional}, \mathrm{seq}([\mathit{expression}, \mathit{expression}, \mathit{expression}]))
\end{array}
$$

This yields the following nominal mapping:

$$
\begin{aligned}
\mathit{txl} \diamond \mathit{master} = \{ & \langle \mathit{program}, \mathit{program}\rangle, \\
& \langle \mathit{expression}_2, \mathit{apply}\rangle, \\
& \langle \mathit{fun}, \mathit{function}\rangle, \\
& \langle \mathit{expression}, \mathit{expression}\rangle, \\
& \langle \mathit{id}, \mathit{str}\rangle, \\
& \langle \mathit{expression}_1, \mathit{binary}\rangle, \\
& \langle \mathit{op}, \mathit{operator}\rangle, \\
& \langle \mathit{number}, \mathit{int}\rangle, \\
& \langle \mathit{newline}, \omega\rangle, \\
& \langle \mathit{expression}_3, \mathit{conditional}\rangle\}
\end{aligned}
$$

Which is exercised with these grammar transformation steps:

- **renameN-renameN** $\mathit{expression}_2$ to *apply*
- **renameN-renameN** *fun* to *function*
- **renameN-renameN** *id* to *str*
- **renameN-renameN** $\mathit{expression}_1$ to *binary*
- **renameN-renameN** *op* to *operator*
- **renameN-renameN** *number* to *int*
- **renameN-renameN** $\mathit{expression}_3$ to *conditional*

10.5 Structural resolution

- **project-inject**
  $\mathrm{p}\left(`',\mathit{function},\mathrm{seq}\left(\left[\mathit{str},+(\mathit{str}),\mathit{expression},\boxed{\mathit{newline}}\right]\right)\right)$

Grammar 11

XML Schema

Source name: **xsd**

11.1 Source grammar

- Source artifact: topics/fl/xsd/fl.xsd
- Grammar extractor: shared/prolog/xsd2bgf.pro

Production rules
p(‘’, *Program*, +(sel (‘function’, *Function*)))
p(‘’, *Fragment*, *Expr*)
p(‘’, *Function*, seq ([sel (‘name’, *str*) , +(sel (‘arg’, *str*)) , sel (‘rhs’, *Expr*)]))
p(‘’, *Expr*, choice([*Literal*, *Argument*, *Binary*, *IfThenElse*, *Apply*]))
p(‘’, *Literal*, sel (‘info’, *int*))
p(‘’, *Argument*, sel (‘name’, *str*))
p(‘’, *Binary*, seq ([sel (‘ops’, *Ops*) , sel (‘left’, *Expr*) , sel (‘right’, *Expr*)]))
p(‘’, *Ops*, choice([sel (‘Equal’, ε) , sel (‘Plus’, ε) , sel (‘Minus’, ε)]))
p(‘’, *IfThenElse*, seq ([sel (‘ifExpr’, *Expr*) , sel (‘thenExpr’, *Expr*) , sel (‘elseExpr’, *Expr*)]))
p(‘’, *Apply*, seq ([sel (‘name’, *str*) , +(sel (‘arg’, *Expr*))]))

11.2 Normalizations

- **unlabel-designate**
  p ([‘info’], *Literal*, *int*)
- **unlabel-designate**
  p ([‘name’], *Argument*, *str*)
- **anonymize-deanonymize**
  p (‘’, *Apply*, seq ([[sel (‘name’, *str*)], +([sel (‘arg’, *Expr*)])]))
- **anonymize-deanonymize**
  p (‘’, *Function*, seq ([[sel (‘name’, *str*)], +([sel (‘arg’, *str*)]) , [sel (‘rhs’, *Expr*)]]))
- **anonymize-deanonymize**
  p (‘’, *IfThenElse*, seq ([[sel (‘ifExpr’, *Expr*)], [sel (‘thenExpr’, *Expr*)], [sel (‘elseExpr’, *Expr*)]]))
- **anonymize-deanonymize**
  p (‘’, *Program*, +([sel (‘function’, *Function*)]))

- **anonymize-deanonymize**
  $\mathrm{p}\left(\text{`'}, \mathit{Ops}, \mathrm{choice}\left(\left[\boxed{\mathrm{sel}\,(\text{`Equal'}, \varepsilon)}, \boxed{\mathrm{sel}\,(\text{`Plus'}, \varepsilon)}, \boxed{\mathrm{sel}\,(\text{`Minus'}, \varepsilon)}\right]\right)\right)$
- **anonymize-deanonymize**
  $\mathrm{p}\left(\text{`'}, \mathit{Binary}, \mathrm{seq}\left(\left[\boxed{\mathrm{sel}\,(\text{`ops'}, \mathit{Ops})}, \boxed{\mathrm{sel}\,(\text{`left'}, \mathit{Expr})}, \boxed{\mathrm{sel}\,(\text{`right'}, \mathit{Expr})}\right]\right)\right)$
- **vertical-horizontal** in *Expr*
- **undefine-define**
  $\mathrm{p}\,(\text{`'}, \mathit{Ops}, \varepsilon)$
- **unchain-chain**
  $\mathrm{p}\,(\text{`'}, \mathit{Expr}, \mathit{Literal})$
- **unchain-chain**
  $\mathrm{p}\,(\text{`'}, \mathit{Expr}, \mathit{Argument})$
- **unchain-chain**
  $\mathrm{p}\,(\text{`'}, \mathit{Expr}, \mathit{Binary})$
- **unchain-chain**
  $\mathrm{p}\,(\text{`'}, \mathit{Expr}, \mathit{IfThenElse})$
- **unchain-chain**
  $\mathrm{p}\,(\text{`'}, \mathit{Expr}, \mathit{Apply})$
- **unlabel-designate**
  $\mathrm{p}\,(\boxed{\text{`Literal'}}, \mathit{Expr}, \mathit{int})$
- **unlabel-designate**
  $\mathrm{p}\,(\boxed{\text{`Argument'}}, \mathit{Expr}, \mathit{str})$
- **unlabel-designate**
  $\mathrm{p}\,(\boxed{\text{`Binary'}}, \mathit{Expr}, \mathrm{seq}\,([\mathit{Ops}, \mathit{Expr}, \mathit{Expr}]))$
- **unlabel-designate**
  $\mathrm{p}\,(\boxed{\text{`IfThenElse'}}, \mathit{Expr}, \mathrm{seq}\,([\mathit{Expr}, \mathit{Expr}, \mathit{Expr}]))$
- **unlabel-designate**
  $\mathrm{p}\,(\boxed{\text{`Apply'}}, \mathit{Expr}, \mathrm{seq}\,([\mathit{str}, +(\mathit{Expr})]))$
- **extract-inline** in *Expr*
  $\mathrm{p}\,(\text{`'}, \mathit{Expr}_1, \mathrm{seq}\,([\mathit{Ops}, \mathit{Expr}, \mathit{Expr}]))$
- **extract-inline** in *Expr*
  $\mathrm{p}\,(\text{`'}, \mathit{Expr}_2, \mathrm{seq}\,([\mathit{Expr}, \mathit{Expr}, \mathit{Expr}]))$
- **extract-inline** in *Expr*
  $\mathrm{p}\,(\text{`'}, \mathit{Expr}_3, \mathrm{seq}\,([\mathit{str}, +(\mathit{Expr})]))$

11.3 Grammar in ANF

Production rule	Production signature
$\mathrm{p}\,(\text{`'}, \mathit{Program}, +(\mathit{Function}))$	$\{\langle \mathit{Function}, + \rangle\}$
$\mathrm{p}\,(\text{`'}, \mathit{Fragment}, \mathit{Expr})$	$\{\langle \mathit{Expr}, 1 \rangle\}$
$\mathrm{p}\,(\text{`'}, \mathit{Function}, \mathrm{seq}\,([\mathit{str}, +(\mathit{str}), \mathit{Expr}]))$	$\{\langle \mathit{str}, 1+ \rangle, \langle \mathit{Expr}, 1 \rangle\}$
$\mathrm{p}\,(\text{`'}, \mathit{Expr}, \mathit{int})$	$\{\langle \mathit{int}, 1 \rangle\}$
$\mathrm{p}\,(\text{`'}, \mathit{Expr}, \mathit{str})$	$\{\langle \mathit{str}, 1 \rangle\}$
$\mathrm{p}\,(\text{`'}, \mathit{Expr}, \mathit{Expr}_1)$	$\{\langle \mathit{Expr}_1, 1 \rangle\}$
$\mathrm{p}\,(\text{`'}, \mathit{Expr}, \mathit{Expr}_2)$	$\{\langle \mathit{Expr}_2, 1 \rangle\}$
$\mathrm{p}\,(\text{`'}, \mathit{Expr}, \mathit{Expr}_3)$	$\{\langle \mathit{Expr}_3, 1 \rangle\}$
$\mathrm{p}\,(\text{`'}, \mathit{Expr}_1, \mathrm{seq}\,([\mathit{Ops}, \mathit{Expr}, \mathit{Expr}]))$	$\{\langle \mathit{Ops}, 1 \rangle, \langle \mathit{Expr}, 11 \rangle\}$
$\mathrm{p}\,(\text{`'}, \mathit{Expr}_2, \mathrm{seq}\,([\mathit{Expr}, \mathit{Expr}, \mathit{Expr}]))$	$\{\langle \mathit{Expr}, 111 \rangle\}$
$\mathrm{p}\,(\text{`'}, \mathit{Expr}_3, \mathrm{seq}\,([\mathit{str}, +(\mathit{Expr})]))$	$\{\langle \mathit{str}, 1 \rangle, \langle \mathit{Expr}, + \rangle\}$

11.4 Nominal resolution

Production rules are matched as follows (ANF on the left, master grammar on the right):

$$
\begin{array}{rcl}
\mathrm{p}(\text{‘’}, \mathit{Program}, +(\mathit{Function})) & \bumpeq & \mathrm{p}(\text{‘’}, \mathit{program}, +(\mathit{function})) \\
\mathrm{p}(\text{‘’}, \mathit{Fragment}, \mathit{Expr}) & & \varnothing \\
\mathrm{p}(\text{‘’}, \mathit{Function}, \mathrm{seq}([\mathit{str}, +(\mathit{str}), \mathit{Expr}])) & \bumpeq & \mathrm{p}(\text{‘’}, \mathit{function}, \mathrm{seq}([\mathit{str}, +(\mathit{str}), \mathit{expression}])) \\
\mathrm{p}(\text{‘’}, \mathit{Expr}, \mathit{int}) & \bumpeq & \mathrm{p}(\text{‘’}, \mathit{expression}, \mathit{int}) \\
\mathrm{p}(\text{‘’}, \mathit{Expr}, \mathit{str}) & \bumpeq & \mathrm{p}(\text{‘’}, \mathit{expression}, \mathit{str}) \\
\mathrm{p}(\text{‘’}, \mathit{Expr}, \mathit{Expr_1}) & \bumpeq & \mathrm{p}(\text{‘’}, \mathit{expression}, \mathit{binary}) \\
\mathrm{p}(\text{‘’}, \mathit{Expr}, \mathit{Expr_2}) & \bumpeq & \mathrm{p}(\text{‘’}, \mathit{expression}, \mathit{conditional}) \\
\mathrm{p}(\text{‘’}, \mathit{Expr}, \mathit{Expr_3}) & \bumpeq & \mathrm{p}(\text{‘’}, \mathit{expression}, \mathit{apply}) \\
\mathrm{p}(\text{‘’}, \mathit{Expr_1}, \mathrm{seq}([\mathit{Ops}, \mathit{Expr}, \mathit{Expr}])) & \Bumpeq & \mathrm{p}(\text{‘’}, \mathit{binary}, \mathrm{seq}([\mathit{expression}, \mathit{operator}, \mathit{expression}])) \\
\mathrm{p}(\text{‘’}, \mathit{Expr_2}, \mathrm{seq}([\mathit{Expr}, \mathit{Expr}, \mathit{Expr}])) & \bumpeq & \mathrm{p}(\text{‘’}, \mathit{conditional}, \mathrm{seq}([\mathit{expression}, \mathit{expression}, \mathit{expression}])) \\
\mathrm{p}(\text{‘’}, \mathit{Expr_3}, \mathrm{seq}([\mathit{str}, +(\mathit{Expr})])) & \bumpeq & \mathrm{p}(\text{‘’}, \mathit{apply}, \mathrm{seq}([\mathit{str}, +(\mathit{expression})]))
\end{array}
$$

This yields the following nominal mapping:

$$
\begin{aligned}
\mathit{xsd} \diamond \mathit{master} = \{ & \langle \mathit{Expr_1}, \mathit{binary} \rangle, \\
& \langle \mathit{str}, \mathit{str} \rangle, \\
& \langle \mathit{int}, \mathit{int} \rangle, \\
& \langle \mathit{Expr_2}, \mathit{conditional} \rangle, \\
& \langle \mathit{Function}, \mathit{function} \rangle, \\
& \langle \mathit{Program}, \mathit{program} \rangle, \\
& \langle \mathit{Expr_3}, \mathit{apply} \rangle, \\
& \langle \mathit{Expr}, \mathit{expression} \rangle, \\
& \langle \mathit{Ops}, \mathit{operator} \rangle \}
\end{aligned}
$$

Which is exercised with these grammar transformation steps:

- **renameN-renameN** $\mathit{Expr_1}$ to *binary*
- **renameN-renameN** $\mathit{Expr_2}$ to *conditional*
- **renameN-renameN** *Function* to *function*
- **renameN-renameN** *Program* to *program*
- **renameN-renameN** $\mathit{Expr_3}$ to *apply*
- **renameN-renameN** *Expr* to *expression*
- **renameN-renameN** *Ops* to *operator*

11.5 Structural resolution

- **reroot-reroot** [*program*, *Fragment*] to [*program*]
- **eliminate-introduce**
  $\mathrm{p}(\text{‘’}, \mathit{Fragment}, \mathit{expression})$
- **permute-permute**
  $\mathrm{p}(\text{‘’}, \mathit{binary}, \mathrm{seq}([\mathit{operator}, \mathit{expression}, \mathit{expression}]))$
  $\mathrm{p}(\text{‘’}, \mathit{binary}, \mathrm{seq}([\mathit{expression}, \mathit{operator}, \mathit{expression}]))$

Bibliography

[1] J. Bézivin, F. Jouault, and P. Valduriez. On the Need for Megamodels. *OOPSLA & GPCE, Workshop on best MDSD practices*, 2004.
Publicly available via http://www.softmetaware.com/oopsla2004/bezivin-megamodel.pdf.

[2] T. R. Dean, J. R. Cordy, A. J. Malton, and K. A. Schneider. Grammar Programming in TXL. In *Proceedings of SCAM 2002*. IEEE, 2002.
Publicly available via http://plg1.uwaterloo.ca/~ajmalton/ajmalton/Papers/SCAM02_GP.pdf.

[3] Eclipse. Eclipse Modeling Framework Project (EMF 2.4), 2008. http://www.eclipse.org/modeling/emf/.

[4] G. Economopoulos, P. Klint, and J. J. Vinju. Faster scannerless GLR parsing. In O. de Moor and M. I. Schwartzbach, editors, *Proceedings of CC 2009*, volume 5501 of *LNCS*, pages 126–141. Springer, 2009.
Publicly available via http://oai.cwi.nl/oai/asset/15095/15095B.pdf.

[5] J.-M. Favre, R. Lämmel, and A. Varanovich. Modeling the Linguistic Architecture of Software Products. In *Proceedings of MODELS 2012*, LNCS. Springer, 2012.
Publicly available via http://softlang.uni-koblenz.de/mega.

[6] J.-M. Favre and T. NGuyen. Towards a Megamodel to Model Software Evolution through Transformations. In *Proceedings of SETra*, volume 127 of *ENTCS*, 2004.
Publicly available via http://adele.imag.fr/Les.Publications/intConferences/SETRAa2004Fav.pdf.

[7] J. Fialli and S. Vajjhala. *Java Specification Request 31: XML Data Binding Specification*, 1999.
Publicly available via http://jcp.org/en/jsr/detail?id=031.

[8] S. Gao, C. M. Sperberg-McQueen, and H. S. Thompson. W3C XML Schema Definition Language (XSD) 1.1 Part 1: Structures. *W3C Recommendation*, Apr. 2012.
Publicly available via http://www.w3.org/TR/2012/REC-xmlschema11-1-20120405.

[9] P. Klint. A Meta-Environment for Generating Programming Environments. *ACM TOSEM*, 2(2):176–201, 1993.
Publicly available via http://dare.uva.nl/document/28101.

[10] P. Klint et al. *Rascal Tutor*. SWAT, CWI, 2012. http://tutor.rascal-mpl.org.

[11] P. Klint, T. van der Storm, and J. Vinju. EASY Meta-programming with Rascal. In J. M. Fernandes, R. Lämmel, J. Visser, and J. Saraiva, editors, *Post-proceedings of GTTSE 2009*, volume 6491 of *LNCS*, pages 222–289. Springer-Verlag, January 2011.
Publicly available via http://homepages.cwi.nl/~paulk/publications/rascal-gttse-final.pdf.

[12] R. Lämmel and V. Zaytsev. An Introduction to Grammar Convergence. In M. Leuschel and H. Wehrheim, editors, *Proceedings of iFM 2009*, volume 5423 of *LNCS*, pages 246–260. Springer-Verlag, February 2009.
Publicly available via http://grammarware.net/writes#Convergence2009.

[13] P. McGuire. *Getting Started with Pyparsing*. O'Reilly, first edition, 2007.

[14] Object Management Group. *Meta-Object Facility (MOF^{TM}) Core Specification*, 2.0 edition, Jan. 2006.
Publicly available via http://www.omg.org/spec/MOF/2.0.

[15] T. Parr. ANTLR—ANother Tool for Language Recognition, 2008. http://antlr.org.

[16] F. C. N. Pereira and D. H. D. Warren. Definite Clause Grammars for Language Analysis — A Survey of the Formalism and a Comparison with Augmented Transition Networks. *Artificial Intelligence*, 13:231–278, 1980.
Publicly available via http://cgi.di.uoa.gr/~takis/pereira-warren.pdf.

[17] E. Visser. Scannerless Generalized-LR Parsing. Technical Report P9707, Programming Research Group, Universiteit van Amsterdam, July 1997.
Publicly available via http://www.science.uva.nl/pub/programming-research/reports/1997/P9707.ps.Z.

[18] V. Zaytsev. Guided Grammar Convergence. Submitted to POPL 2013. Pending notification. June 2012.
Publicly available via http://grammarware.net/writes#Guided2013.

[19] V. Zaytsev. Renarrating Linguistic Architecture: A Case Study. Submitted to MPM 2012. Pending notification. July 2012.
Publicly available via http://grammarware.net/writes#Renarration2012.

[20] V. Zaytsev. Language Evolution, Metasyntactically. In *Proceedings of BX 2012*, volume 49 of *EC-EASST*. EASST, 2012.
Publicly available via http://grammarware.net/writes#Metasyntactically2012.

[21] V. Zaytsev, R. Lämmel, T. van der Storm, L. Renggli, and G. Wachsmuth. Software Language Processing Suite[1], 2008–2012. http://grammarware.github.com.

[1]The authors are given according to the statistics at http://github.com/grammarware/slps/graphs/contributors.