SLPS

Phase 1	Phase 2	Phase 3
Visit Koblenz	?	Profit

vadim@grammarware.net

SLPS

Phase 1	Phase 2	Phase 3
Visit Koblenz	converge grammars ?	Profit

vadim@grammarware.net

Grammar Convergence

vadim@grammarware.net

Software Languages Team, Universität Koblenz-Landau

Vadim Zaytsev, SWAT, CWI

What is grammar convergence?

- Grammars in the broad sense
- Distributed grammar knowledge
- Surface and maintain relationships
- Transform grammars until convergence
- Lightweight verification

R. Lämmel, V. Zaytsev, *An Introduction to Grammar Convergence*. IFM, LNCS 5423:246–260. 2009.

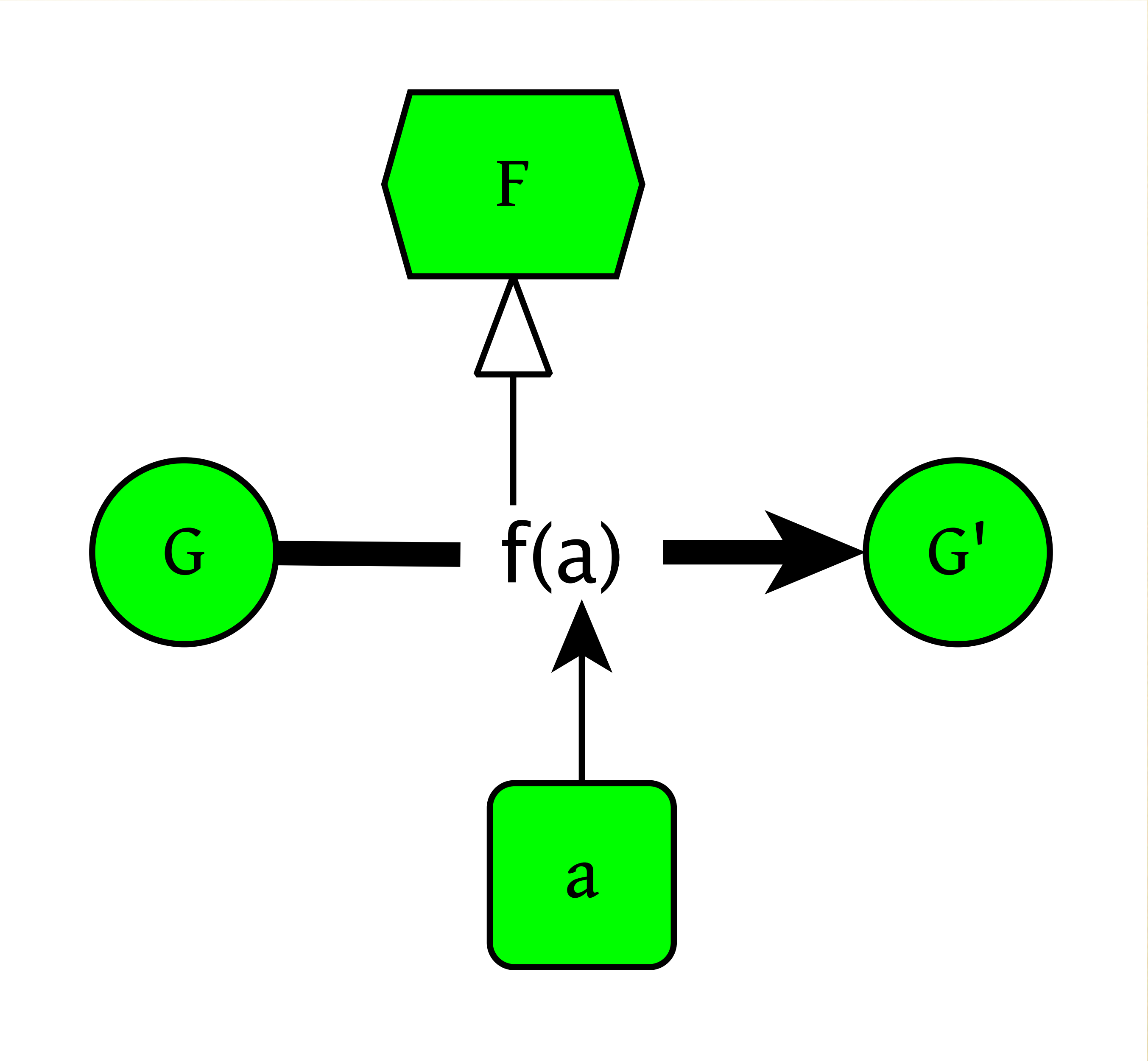
Grammar convergence framewowk

- Grammar format to abstract from idiosyncrasies
- Grammar extraction to feed into the format
- Grammar comparison for spotting grammar deviations
- Grammar transformation:
  - Refactoring
  - Extension / restriction
  - Revision

R. Lämmel, V. Zaytsev, *An Introduction to Grammar Convergence*. IFM, LNCS 5423:246–260. 2009.

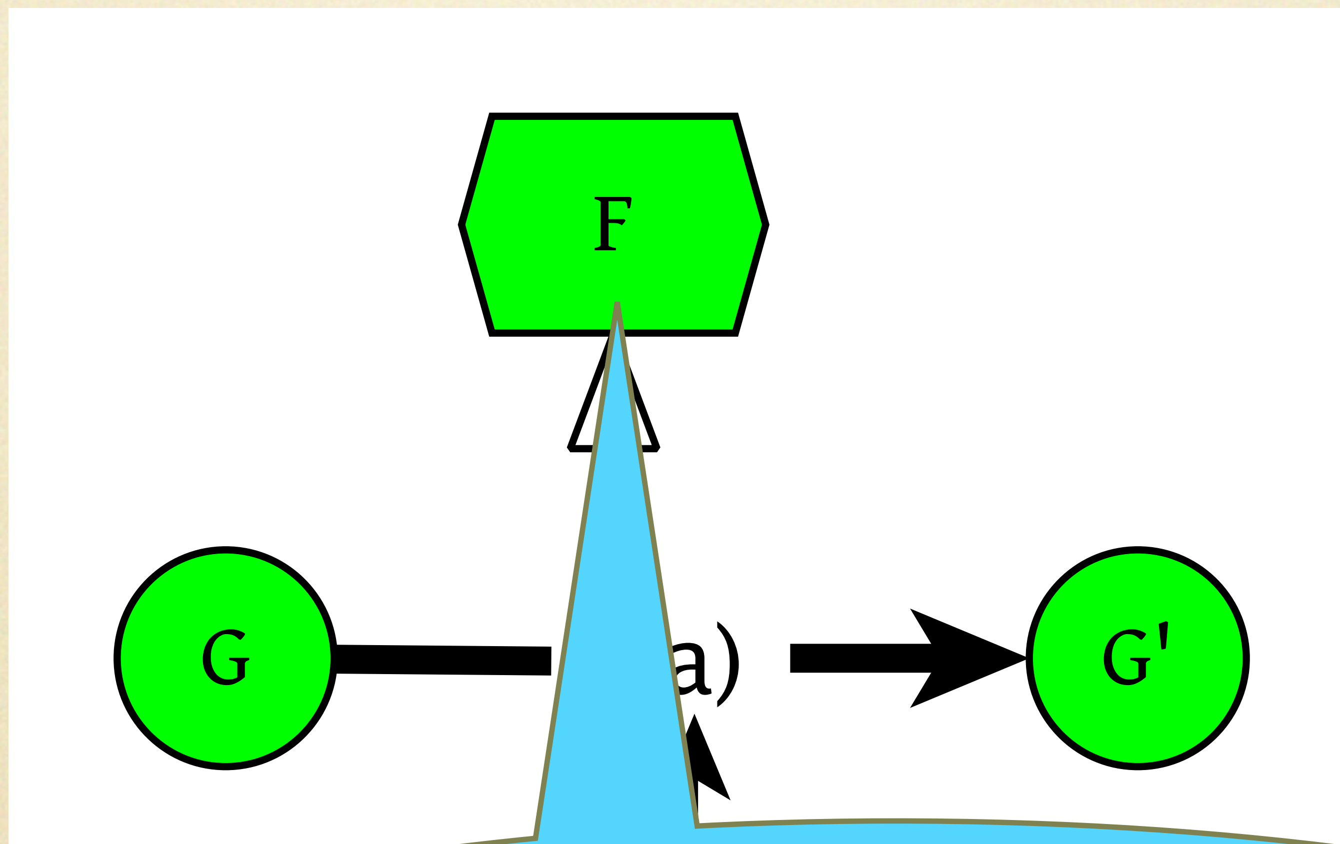
Programmable Grammar Transformations

Programmable G xformation



Ad hoc megamodel shown at IPA Spring Days

Programmable G xformation

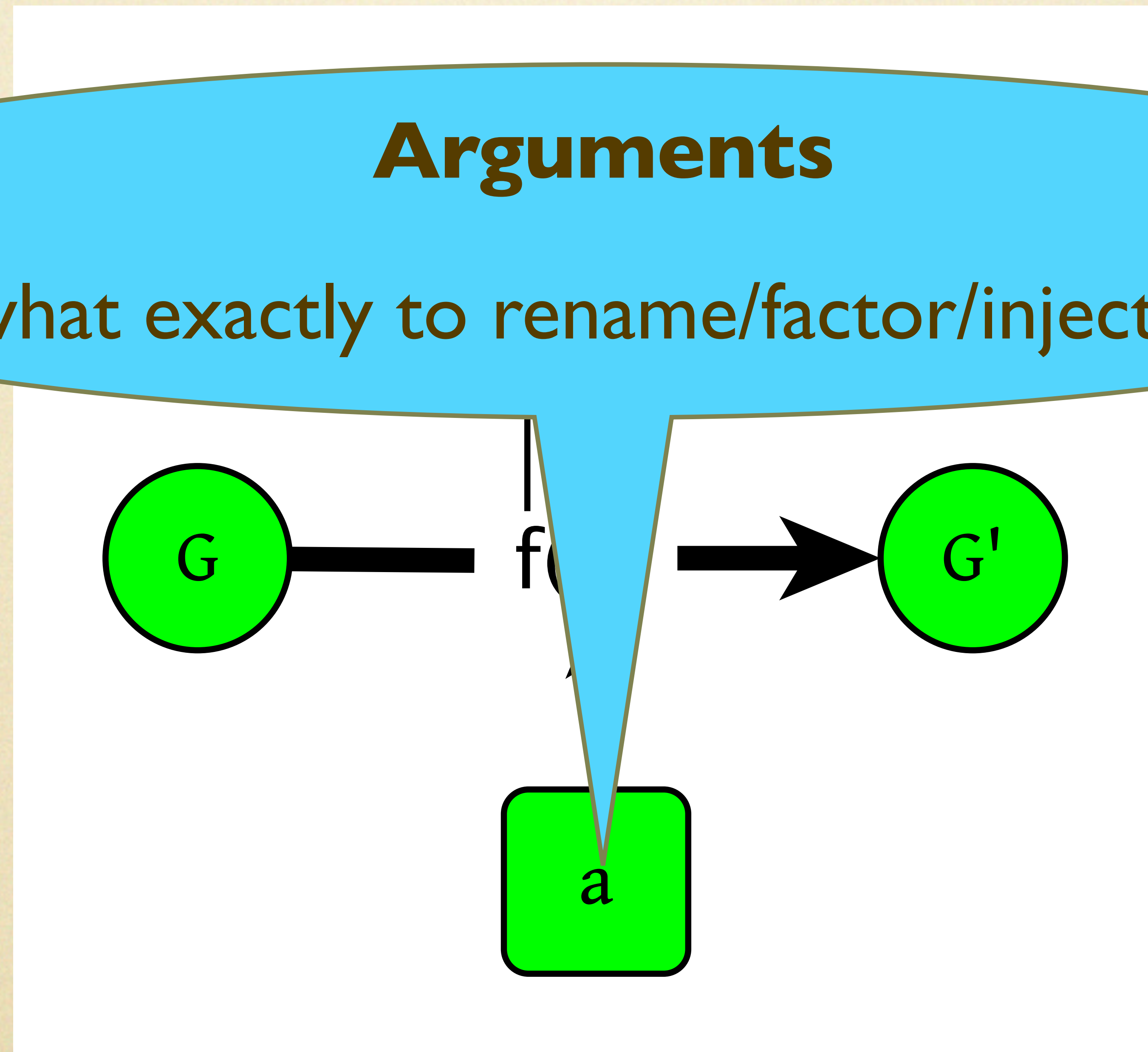


Operator

- known semantics, well-defined algorithm
  - rename, fold, factor, inject, remove, …

Ad hoc megamodel shown at IFA Spring Days

Programmable G xformation



Arguments

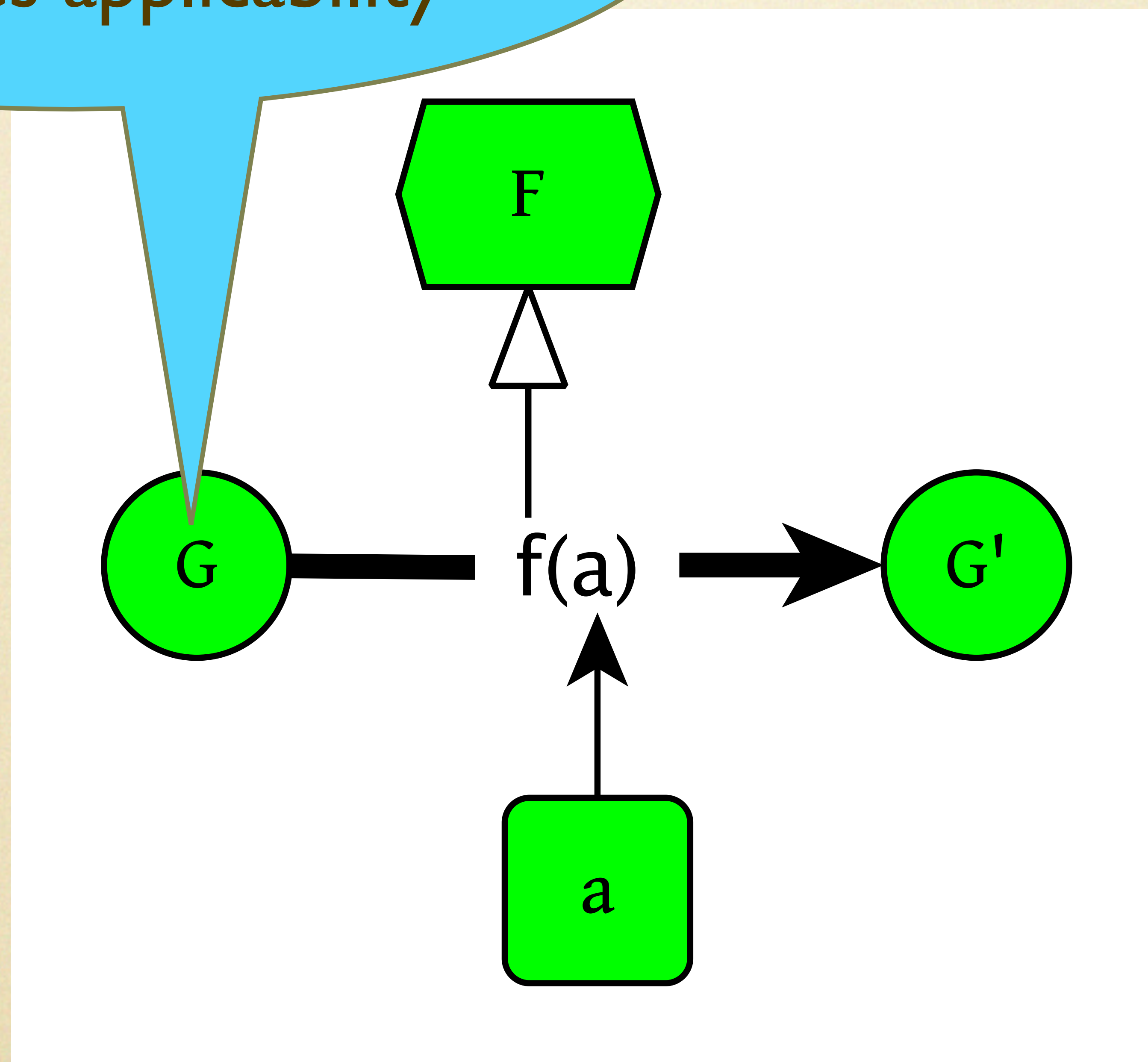
- what exactly to rename/factor/inject/…?

Ad hoc megamodel shown at IPA Spring Days

G xformation

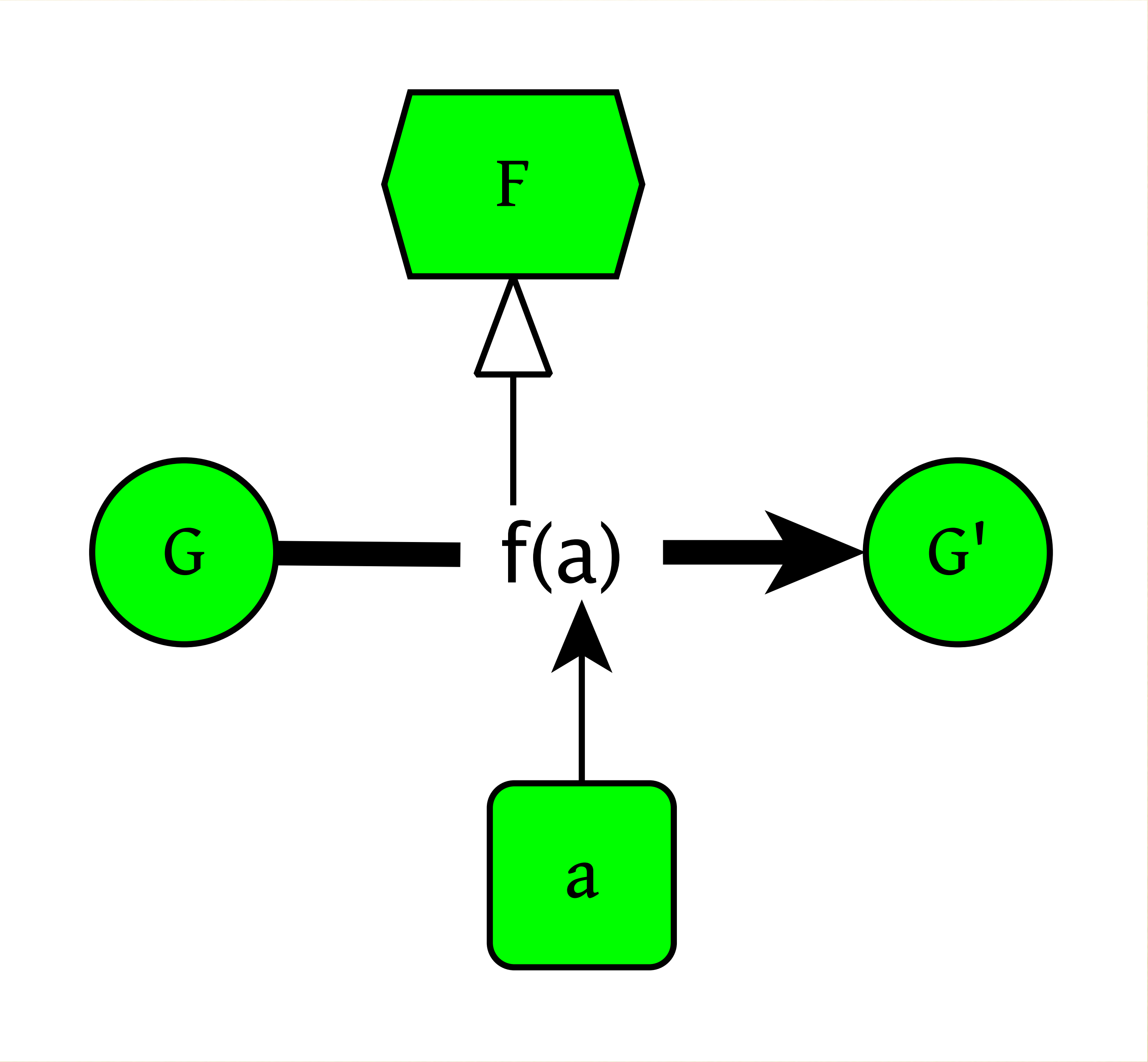
Input grammar

- determines applicability



Ad hoc megamodel shown at IPA Spring Days

Programmable G xformation



Ad hoc megamodel shown at IPA Spring Days

XBGF Operator Suite

- Semantic-preserving operators
  - fold, unfold, extract, inline, massage, factor, deyaccify, …
- (Some) semantic-preserving operators
  - permute, abstractize, concretize, designate, anonymize
- Language-increasing operators
  - add, appear, widen, upgrade, unite
- Language-decreasing operators
  - remove, disappear, narrow, downgrade
- Revising operators
  - redefine, inject, project, replace, …

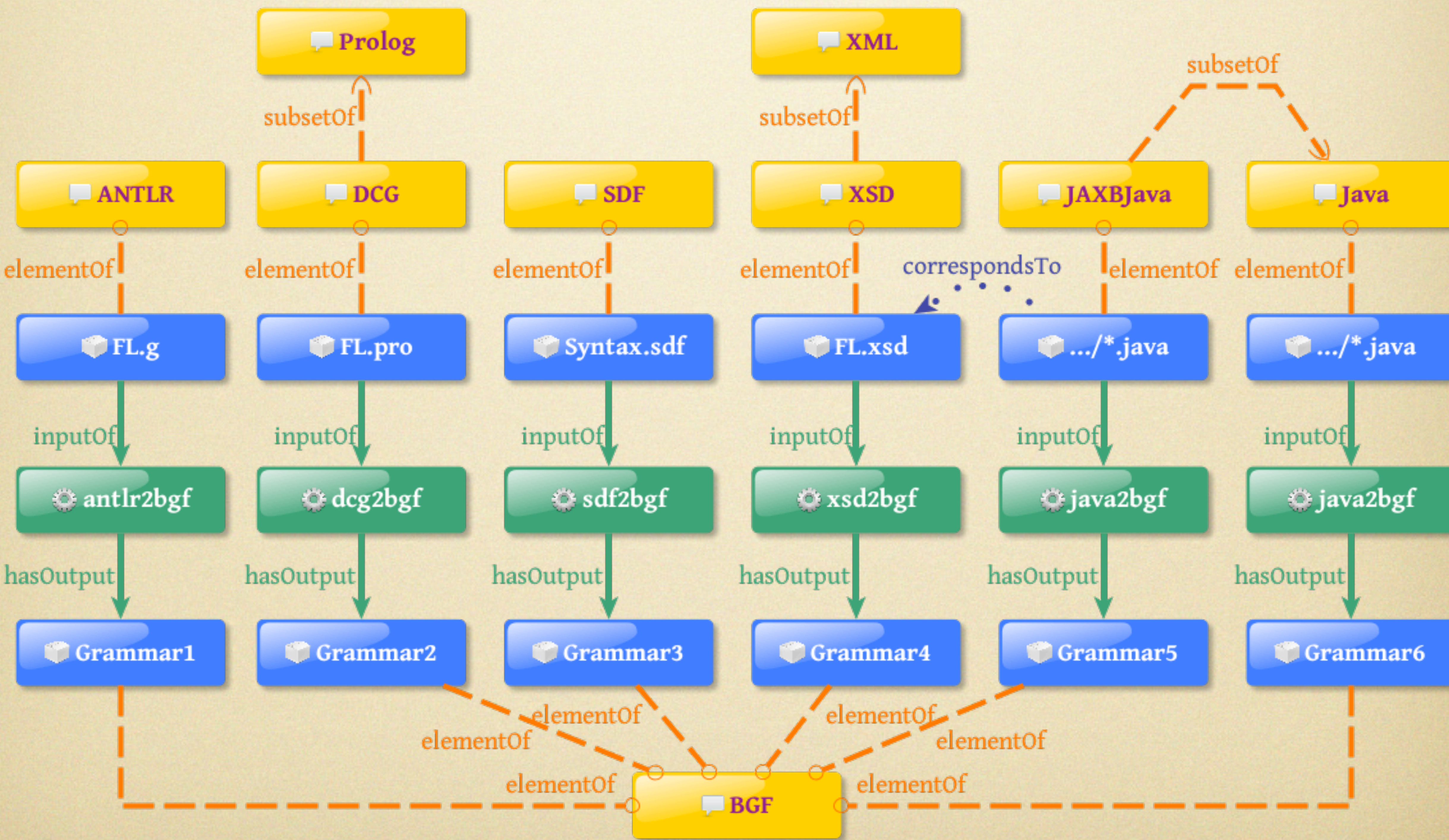
V. Zaytsev, *BGF Transformation Operator Suite v.1.0*, online, 2010.

References

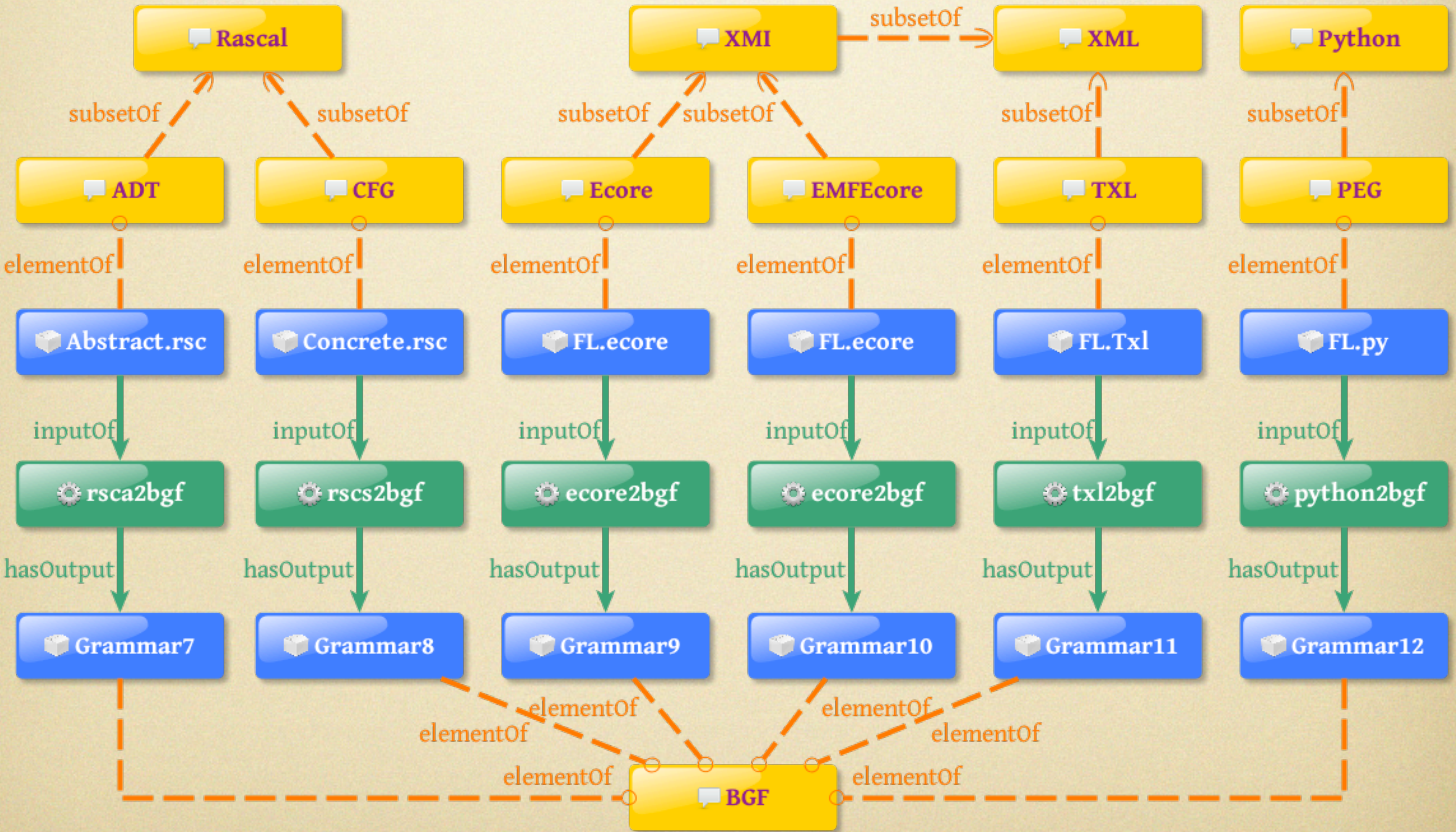
- R. Lämmel. *Grammar Adaptation*. FME, LNCS 2021:550–570. 2001.
- R. Lämmel, G. Wachsmuth. *Transformation of SDF Syntax Definitions in the ASF+SDF Meta-Environment*. LDTA, ENTCS 44. 2001.
- T. R. Dean, J. R. Cordy, A. J. Malton, K. A. Schneider. *Grammar Programming in TXL*. SCAM. 2002.
- R. Lämmel, *Transformations Everywhere*. SCP 52(1–3):1–8, 2004.
- P. Klint, R. Lämmel, C. Verhoef. *Toward an Engineering Discipline for Grammarware*. ACM TOSEM 14(3):331–380, 2005.
- V. Zaytsev, *BGF Transformation Operator Suite v.1.0*, online, 2010.
- V. Zaytsev, *Recovery, Convergence and Documentation of Languages*. PhD, 2010.
- R. Lämmel, V. Zaytsev, *Recovering Grammar Relationships for the Java Language Specification*, SQJ 19(2):333–378. 2011.

Grammar Extraction

Grammar extraction



Grammar extraction!



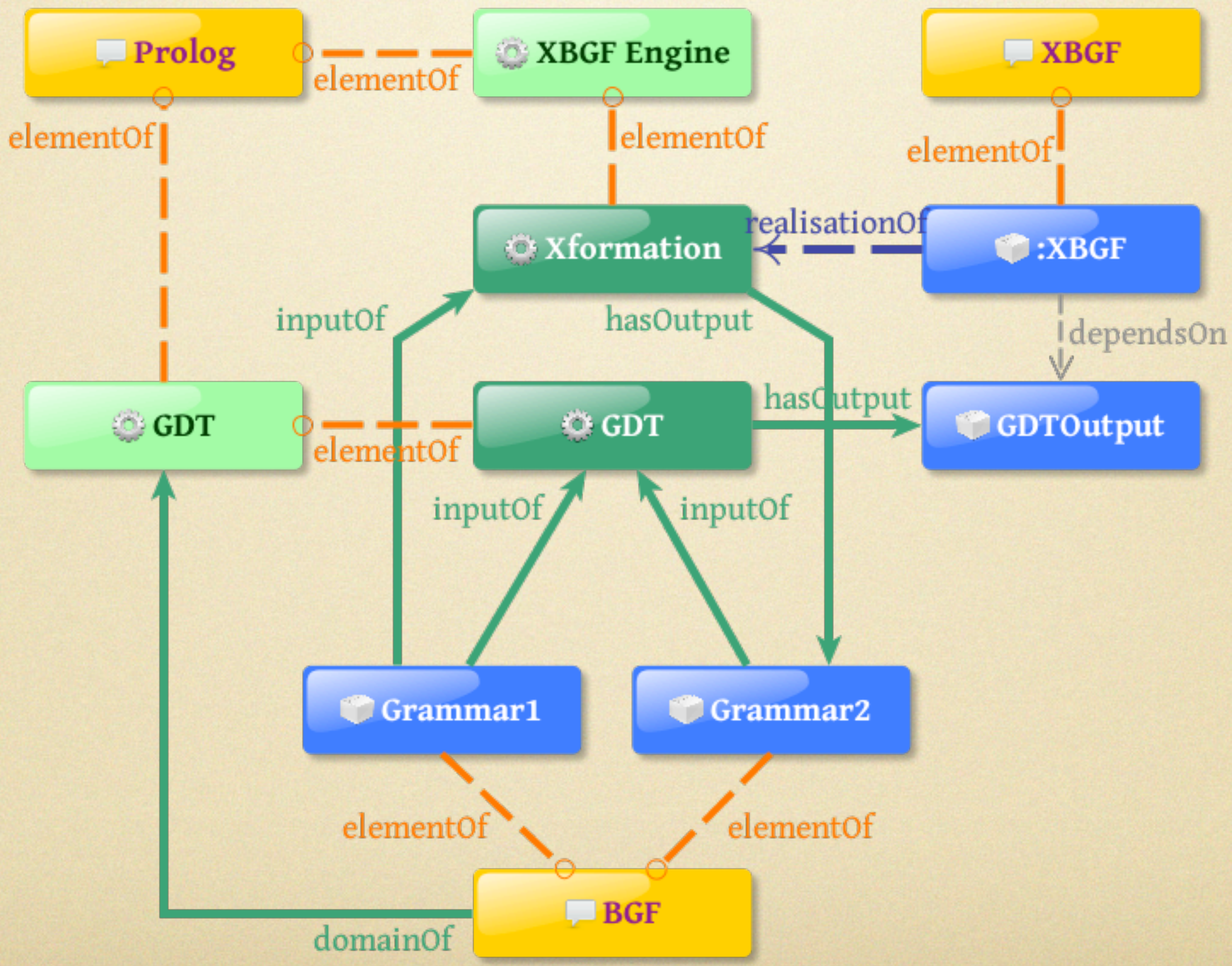
CWI

References

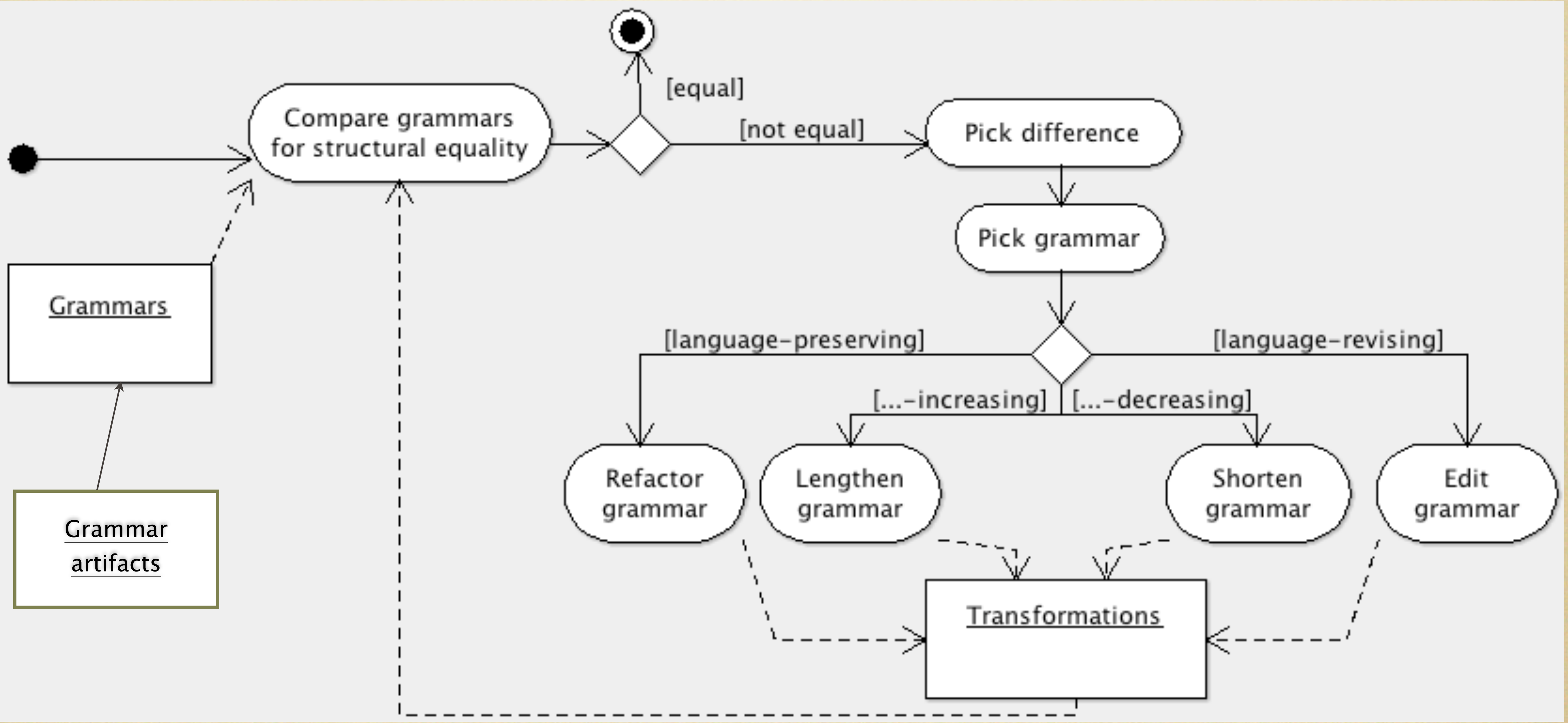
- M. v/d Brand, M. P.A. Sellink, C. Verhoef, Obtaining a COBOL Grammar from Legacy Code for Reengineering Purposes, TaPAS, 1997.
- M. P.A. Sellink, C. Verhoef, *Development, Assessment, and Reengineering of Language Descriptions*, CSMR. 2000.
- R. Lämmel, C. Verhoef, *Cracking the 500-Language Problem*, IEEE Software, 2001.
- R. Lämmel, C. Verhoef, *Semi-automatic Grammar Recovery*, SCP, 2001.
- V. Zaytsev, *Recovery, Convergence and Documentation of Languages*. PhD, 2010.
- R. Lämmel, V. Zaytsev, *Recovering Grammar Relationships for the Java Language Specification*, CoRR, SQJ 19(2):333–378. 2011.
- V. Zaytsev, *MediaWiki Grammar Recovery*, CoRR abs/1107.4661:1–47. 2011.
- V. Zaytsev, *Notation-Parametric Grammar Recovery*. LDTA. 2012.
- V. Zaytsev, *BNF WAS HERE: What Have We Done About the Unnecessary Diversity of Notation for Syntactic Definitions*, SAC/PL:1910–1915. 2012.

Convergence Process

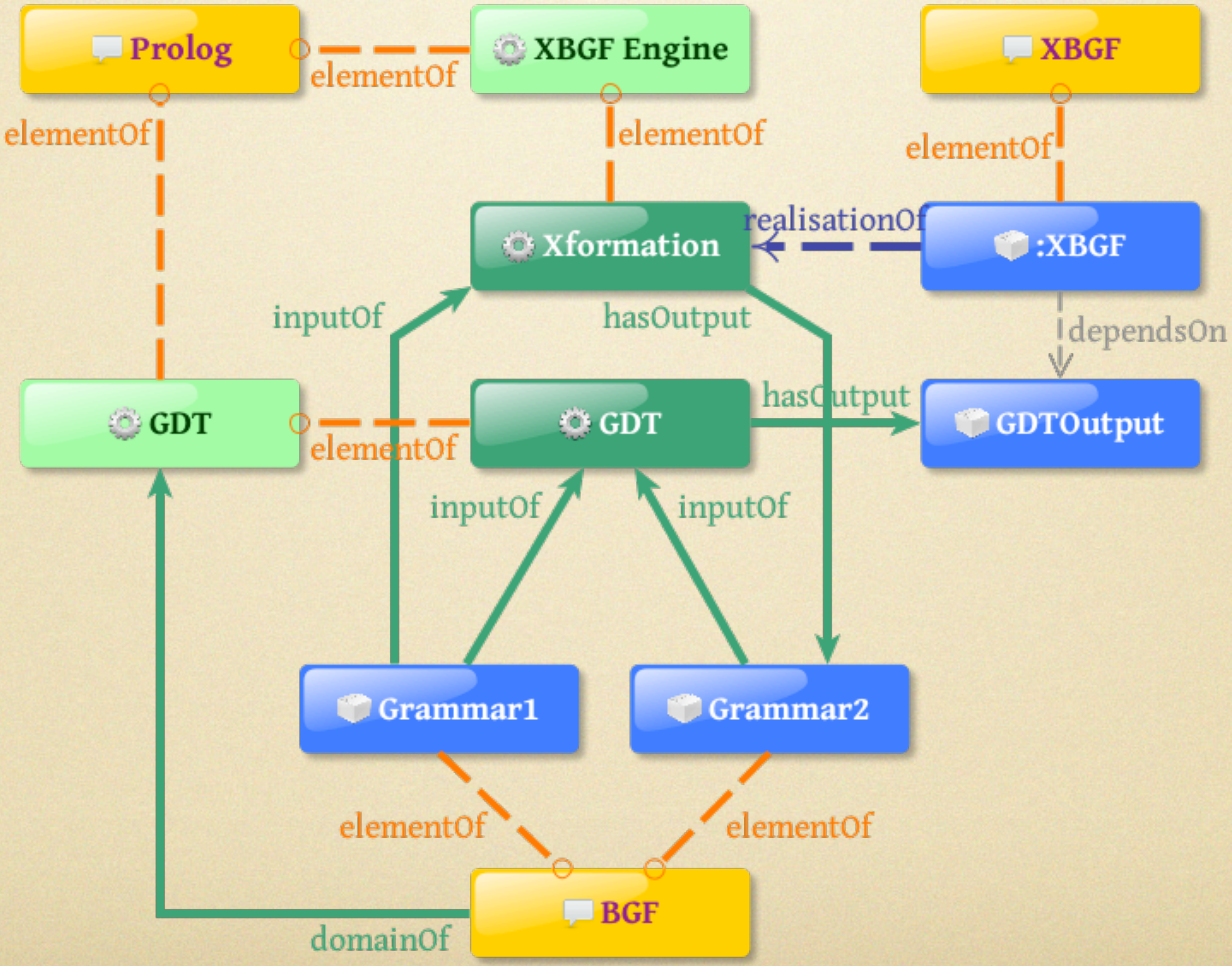
Grammar convergence



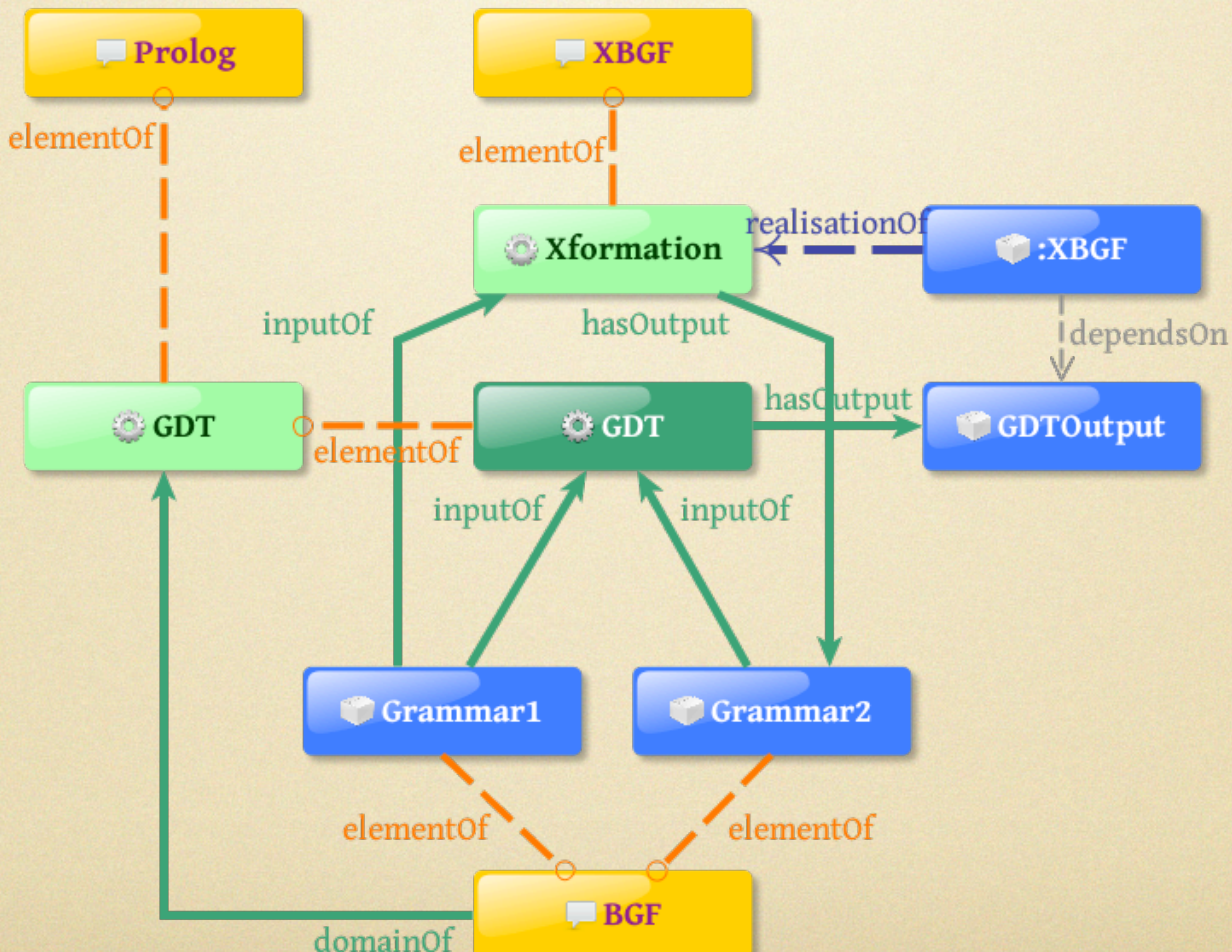
How convergence works



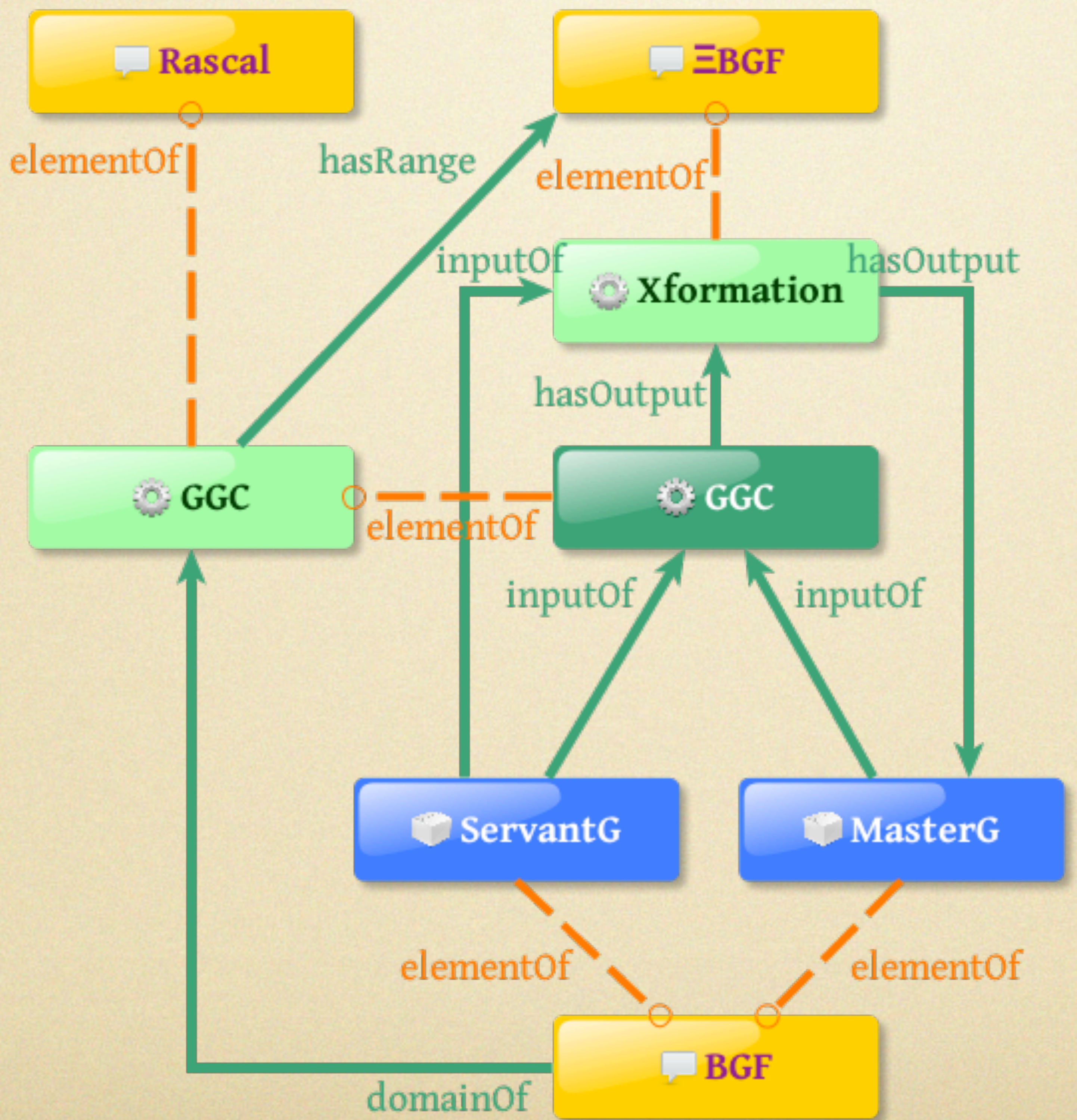
Grammar convergence



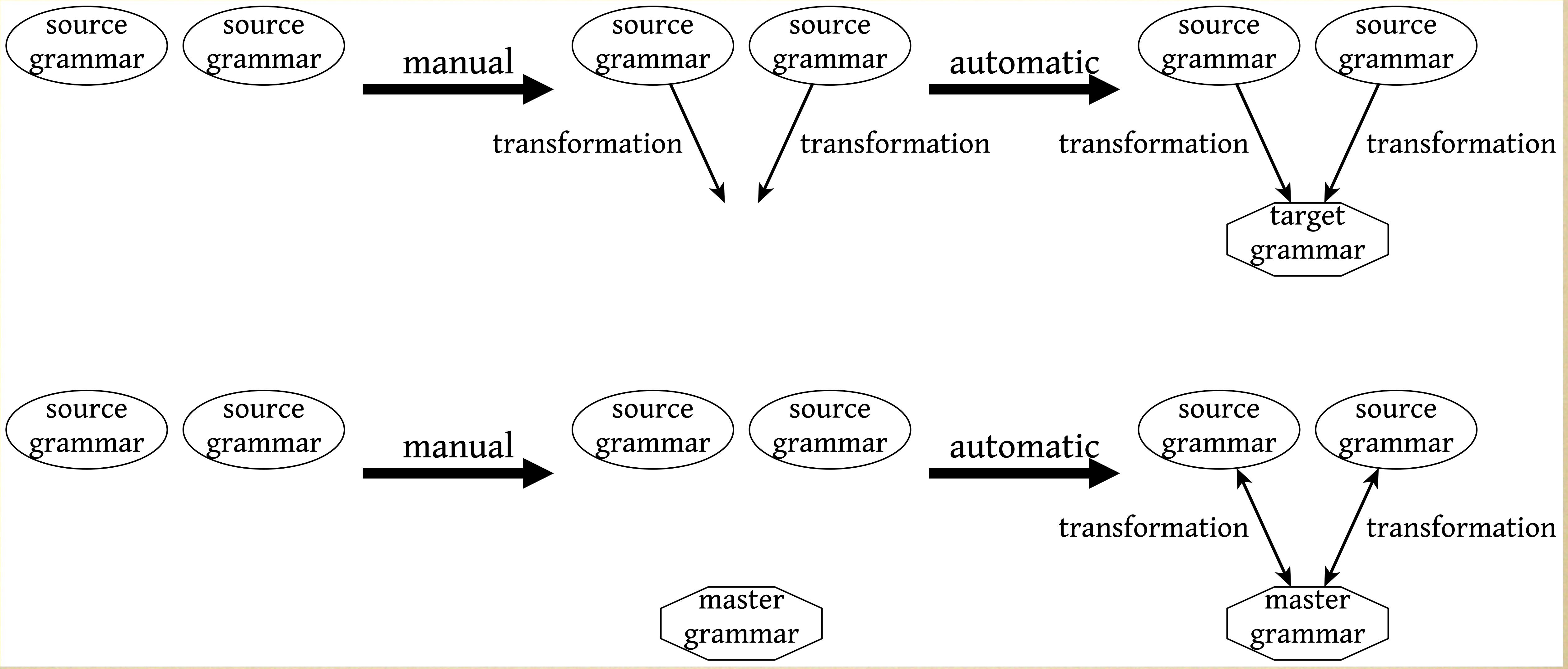
Grammar convergence



Grammar convergence



Guided convergence



V. Zaytsev, Guided Grammar Convergence, POPL draft, 2013.

CWI

References

- R. Lämmel, V. Zaytsev, *An Introduction to Grammar Convergence*. IFM, LNCS 5423:246–260. 2009.
- V. Zaytsev, *Language Convergence Infrastructure*. GTTSE, LNCS 6491:481–497. 2011.
- V. Zaytsev, *Recovery, Convergence and Documentation of Languages*. PhD, 2010.
- R. Lämmel, V. Zaytsev, *Recovering Grammar Relationships for the Java Language Specification*, SCAM, CoRR, SQJ 19(2):333–378. 2011.
- V. Zaytsev, *Guided Grammar Convergence. Full Case Study Report. Generated by* converge::Guided, CoRR?
- V. Zaytsev, *Guided Grammar Convergence*, ~~ECMFA~~, ~~ICSM~~, POPL?

Guided Grammar Convergence

The most trivial case

- Equal grammars
- Algebraically equivalent grammars
- Nothing to do here

V. Zaytsev, *Guided Grammar Convergence*, POPL draft, 2013.

Structural resolution

- Nonterminal vs. value
  - A vs. *string*
- Sequence permutations
  - A B B A vs. B B A A
- Lists of symbols
  - A* vs. A+
- Separator lists… irrelevant

V. Zaytsev, *Guided Grammar Convergence*, POPL draft, 2013.

Nominal resolution

Production rule in the master grammar	Production signature
p_1=p(‘’, *program*, +(*function*))	$\{\langle$*function*, +$\rangle\}$
p_2=p(‘’, *function*, seq([str, +(str), *expr*]))	$\{\langle$*expr*, 1$\rangle$, $\langle str, 1+\rangle\}$
p_3=p(‘’, *expr*, str)	$\{\langle str, 1\rangle\}$
p_4=p(‘’, *expr*, int)	$\{\langle int, 1\rangle\}$
p_5=p(‘’, *expr*, $apply$)	$\{\langle apply, 1\rangle\}$
p_6=p(‘’, *expr*, $binary$)	$\{\langle binary, 1\rangle\}$
p_7=p(‘’, *expr*, $cond$)	$\{\langle cond, 1\rangle\}$
p_8=p(‘’, $apply$, seq([str, +(*expr*)]))	$\{\langle$*expr*, +$\rangle$, $\langle str, 1\rangle\}$
p_9=p(‘’, $binary$, seq([*expr*, *operator*, *expr*]))	$\{\langle$*expr*, 11$\rangle$, $\langle$*operator*, 1$\rangle\}$
p_{10}=p(‘’, $cond$, seq([*expr*, *expr*, *expr*]))	$\{\langle$*expr*, 111$\rangle\}$

Table 1. Production rules of the master grammar for FL, with their production signatures.

V. Zaytsev, Guided Grammar Convergence, POPL draft, 2013.

CWI

Definitions

- Nonterminal footprint
- Production signature
- Prodsig-equivalence
- Weak prodsig-equivalence
- Nominal resolution

V. Zaytsev, *Guided Grammar Convergence*, POPL draft, 2013.

Nominal resolution example

Production rule	Production signature
q_1=p $\left(`', \underline{Fragment}, Expr\right)$	$\{\langle Expr, 1\rangle\}$
q_2=p $\left(`', \underline{Program}, +(Function)\right)$	$\{\langle Function, +\rangle\}$
q_3=p $(`', Function, \text{seq}([str, +(str), Expr]))$	$\{\langle str, 1+\rangle, \langle Expr, 1\rangle\}$
q_4=p $(`', Expr, int)$	$\{\langle int, 1\rangle\}$
q_5=p $(`', Expr, str)$	$\{\langle str, 1\rangle\}$
q_6=p $(`', Expr, Expr_1)$	$\{\langle Expr_1, 1\rangle\}$
q_7=p $(`', Expr, Expr_2)$	$\{\langle Expr_2, 1\rangle\}$
q_8=p $(`', Expr, Expr_3)$	$\{\langle Expr_3, 1\rangle\}$
q_9=p $(`', Expr_1, \text{seq}([Ops, Expr, Expr]))$	$\{\langle Ops, 1\rangle, \langle Expr, 11\rangle\}$
q_{10}=p $(`', Expr_2, \text{seq}([Expr, Expr, Expr]))$	$\{\langle Expr, 111\rangle\}$
q_{11}=p $(`', Expr_3, \text{seq}([str, +(Expr)]))$	$\{\langle str, 1\rangle, \langle Expr, +\rangle\}$

Prerequisite	Match	$p_i \diamond q_j$
roots		$\{\langle program, Fragment\rangle\}$
	$p_1 \not\simeq q_1$	$\{\langle \omega, Fragment\rangle\}$
roots		$\{\langle program, Program\rangle\}$
	$p_1 \simeq q_2$	$\{\langle function, Function\rangle\}$
	$p_2 \simeq q_3$	$\{\langle str, str\rangle, \langle expr, Expr\rangle\}$
$\{\langle str, str\rangle\}$	$p_3 \simeq q_5$	
	$p_4 \simeq q_4$	$\{\langle int, int\rangle\}$
$\{\langle expr, Expr\rangle, \langle str, str\rangle\}$	$p_5 \simeq q_8$	$\{\langle apply, Expr_3\rangle\}$
$\{\langle expr, Expr\rangle, \langle str, str\rangle\}$	$p_8 \simeq q_{11}$	
$\{\langle expr, Expr\rangle\}$	$p_6 \simeq q_6$	$\{\langle binary, Expr_1\rangle\}$
$\{\langle expr, Expr\rangle\}$	$p_9 \simeq q_9$	$\{\langle operator, Ops\rangle\}$
$\{\langle expr, Expr\rangle\}$	$p_7 \simeq q_7$	$\{\langle cond, Expr_2\rangle\}$
$\{\langle expr, Expr\rangle\}$	$p_{10} \simeq q_{10}$	

Table 2. On the left: production rules of the servant grammar for FL, derived from the XML schema, with their production signatures. On the right: the process of derivation of the nominal resolution relation $p_i \diamond q_j$. Note how two hypotheses must be formed and one of them rejected, because this servant grammar has two roots and both need to be checked for prodsig-equivalence with the root of the master grammar. Other than that, all production rules are matched with strong equivalence.

V. Zaytsev, Guided Grammar Convergence, POPL draft, 2013.

Nominal resolution example

Production rule	Production signature
r_1=p $\left(``, \underline{Program}, +(Function)\right)$	$\{\langle Function, +\rangle\}$
r_2=p$(``, Function, \mathrm{seq}([Name, +(Name), Expr, +(CR)]))$	$\{\langle CR, +\rangle, \langle Expr, 1\rangle, \langle Name, 1{+}\rangle\}$
r_3=p $(``, Expr, Expr_1)$	$\{\langle Expr_1, 1\rangle\}$
r_4=p $(``, Expr, Expr_2)$	$\{\langle Expr_2, 1\rangle\}$
r_5=p $(``, Expr, Expr_3)$	$\{\langle Expr_3, 1\rangle\}$
r_6=p $(``, Expr, Name)$	$\{\langle Name, 1\rangle\}$
r_7=p $(``, Expr, Int)$	$\{\langle Int, 1\rangle\}$
r_8=p $(``, Expr_1, \mathrm{seq}([Expr, Ops, Expr]))$	$\{\langle Ops, 1\rangle, \langle Expr, 11\rangle\}$
r_9=p $(``, Expr_2, \mathrm{seq}([Name, +(Expr)]))$	$\{\langle Expr, +\rangle, \langle Name, 1\rangle\}$
r_{10}=p $(``, Expr_3, \mathrm{seq}([Expr, Expr, Expr]))$	$\{\langle Expr, 111\rangle\}$

Prerequisite	Match	$p_i \diamond r_j$
roots		$\{\langle program, Program\rangle\}$
	$p_1 \bumpeq r_1$	$\{\langle function, Function\rangle\}$
	$p_2 \Bumpeq r_2$	$\{\langle \omega, CR\rangle, \langle str, Name\rangle, \langle expr, Expr\rangle\}$
$\{\langle str, Name\rangle\}$	$p_3 \bumpeq r_6$	
	$p_4 \bumpeq r_7$	$\{\langle int, Int\rangle\}$
$\{\langle expr, Expr\rangle, \langle str, Name\rangle\}$	$p_5 \bumpeq r_4$	$\{\langle apply, Expr_2\rangle\}$
$\{\langle expr, Expr\rangle, \langle str, Name\rangle\}$	$p_8 \bumpeq r_9$	
$\{\langle expr, Expr\rangle\}$	$p_7 \bumpeq r_5$	$\{\langle cond, Expr_3\rangle\}$
$\{\langle expr, Expr\rangle\}$	$p_{10} \bumpeq r_{10}$	
$\{\langle expr, Expr\rangle\}$	$p_6 \bumpeq r_3$	$\{\langle binary, Expr_1\rangle\}$
$\{\langle expr, Expr\rangle\}$	$p_9 \bumpeq r_8$	$\{\langle operator, Ops\rangle\}$

Table 3. On the left: production rules of the servant grammar for FL, derived from a corresponding SDF syntax definition, with their production signatures. On the right: the process of derivation of the nominal resolution relation $p_i \diamond r_j$. Note how a special lexical nonterminal for CR nonterminal remains unmatched due to weak equivalence of production rules that contain it.

V. Zaytsev, Guided Grammar Convergence, POPL draft, 2013.

CWI

Abstract Normal Form

(1) lack of labels for production rules

(2) lack of named subexpressions

(3) lack of terminal symbols

(4) maximal outward factoring of inner choices

(5) lack of horizontal production rules

(6) lack of separator lists

(7) lack of trivially defined nonterminals (with α, ε or φ)

(8) no mixing of chain and non-chain production rules

(9) the nonterminal call graph is connected, and its top nonterminals are the starting symbols of the grammar

V. Zaytsev, Guided Grammar Convergence, POPL draft, 2013.

Grammar design mutation

- Deyaccification
  - B = C B | C vs. B = C+
- Layers vs. priorities
  - X = … |Y; Y = … | X;   vs   X = … | …;
- Associativity
  - A O A vs. A (O A)*

Unresolved

- Aggregation

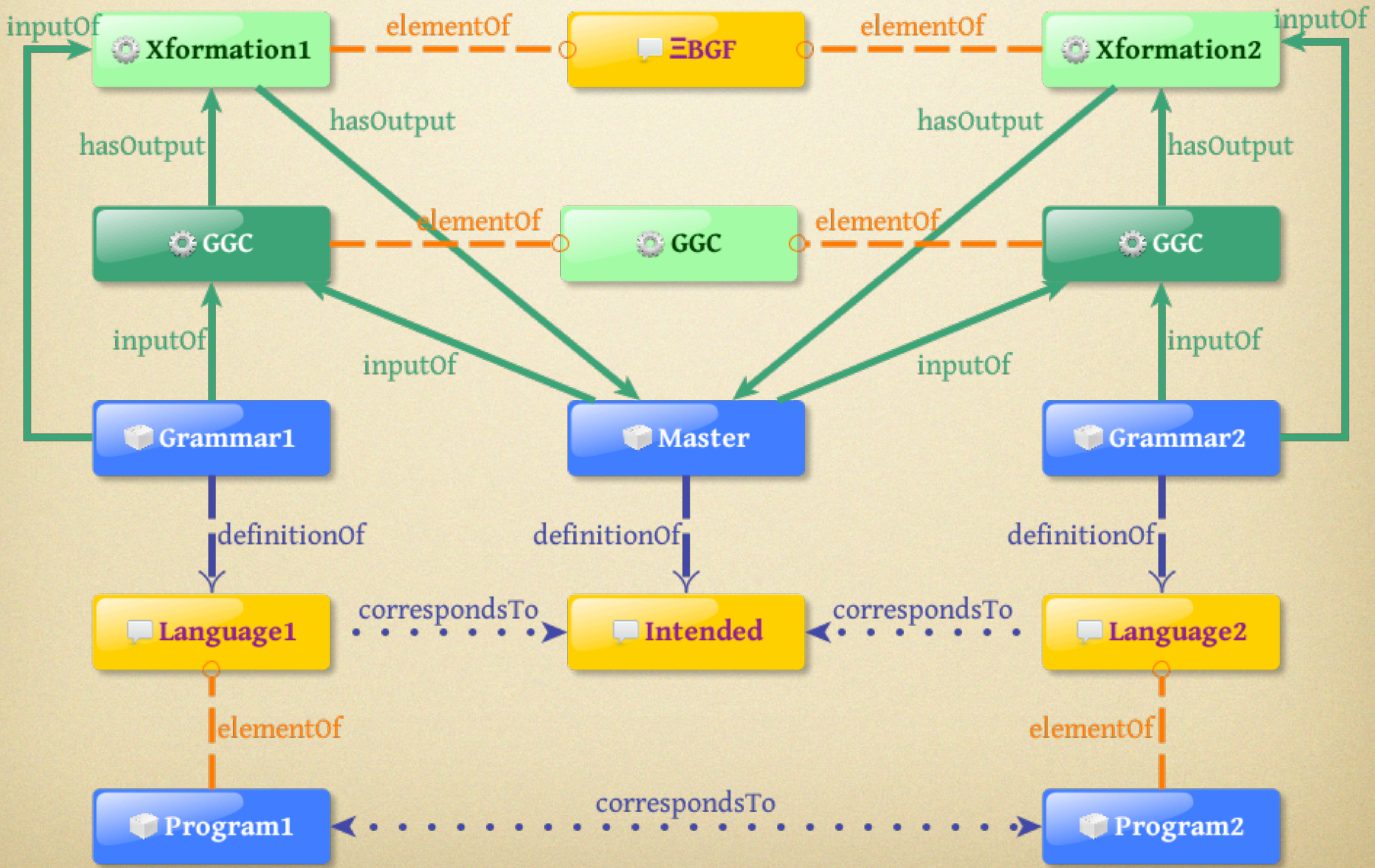
Master grammar	Ecore
`exp:` `STR exp`$^+$	`ApplyExp:` `Function Exp`$^+$

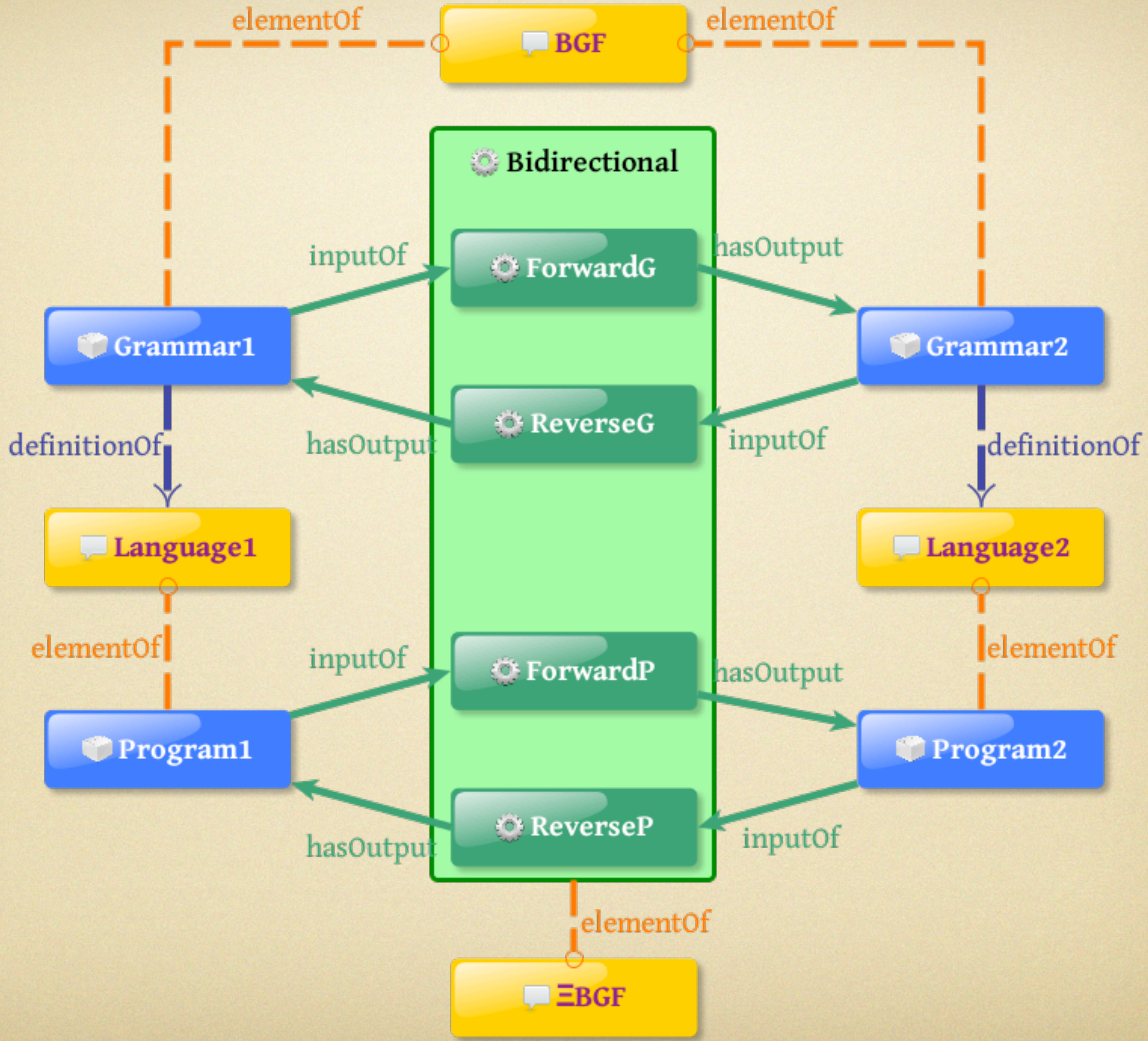
- Meaningful chain rules

Master grammar	Ecore
`exp:` `exp op exp`	`BinaryExp:` `PlusExp` `BinaryExp:` `MinusExp` `BinaryExp:` `EqualExp` `PlusExp:` `Exp Exp` `MinusExp:` `Exp Exp` `EqualExp:` `Exp Exp`

CWI

Final megamodel

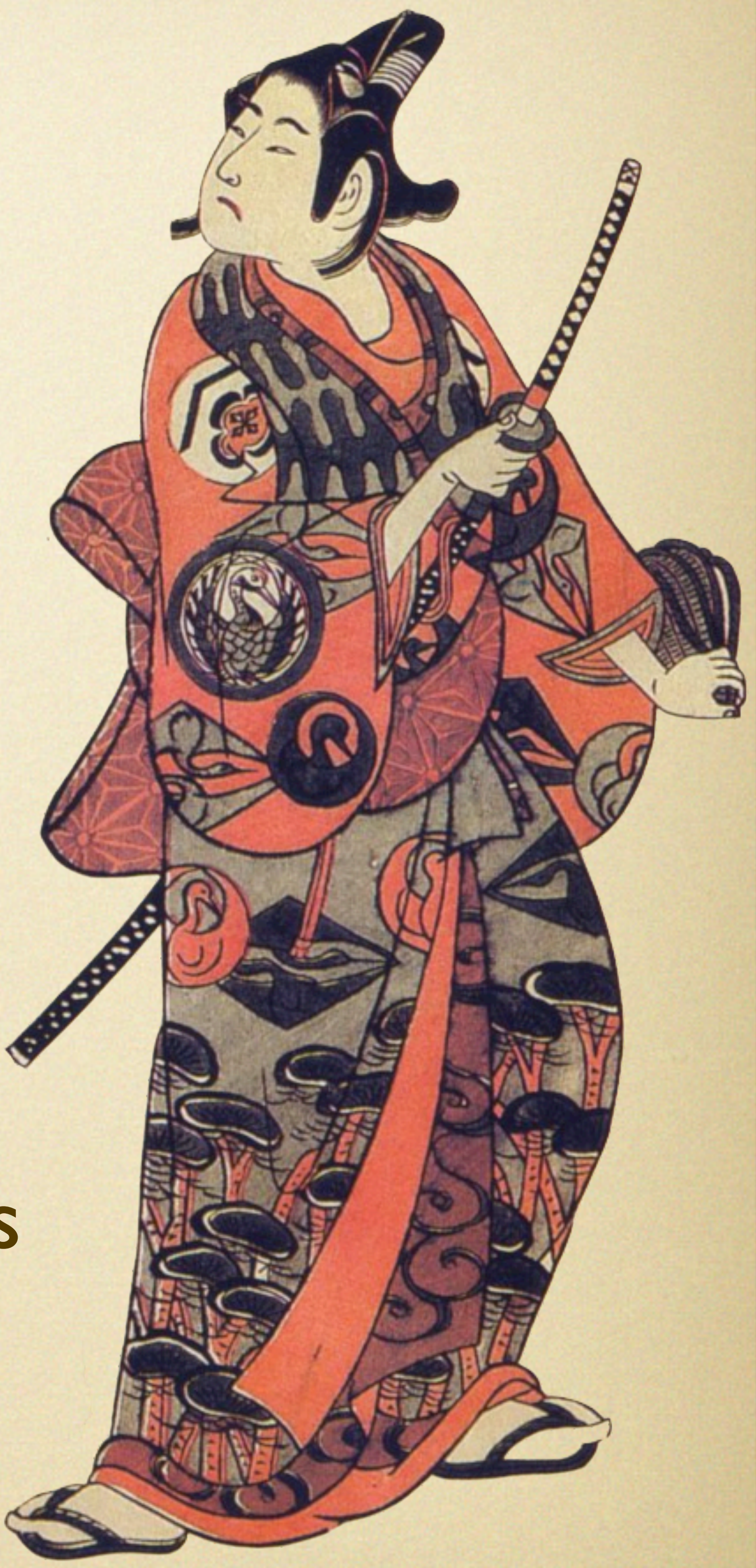



BGF
elementOf
elementOf
Bidirectional
inputOf
ForwardG
hasOutput
Grammar1
Grammar2
ReverseG
definitionOf
hasOutput
inputOf
definitionOf
Language1
Language2
elementOf
elementOf
inputOf
ForwardP
hasOutput
Program1
Program2
ReverseP
hasOutput
inputOf
elementOf
ΞBGF

CWI

To summarise

- Convergence reverse engineers relationships
- We need:
  - extractor
  - transformer
- Guided convergence infers relationships between different grammars of one ~~*intended*~~ problem language



http://commons.wikimedia.org/wiki/File:Torii_kiyoshige_bando_hikosaburo_ii.jpg

Feedback?

vadim@grammarware.net