Grammar Composition & Extension

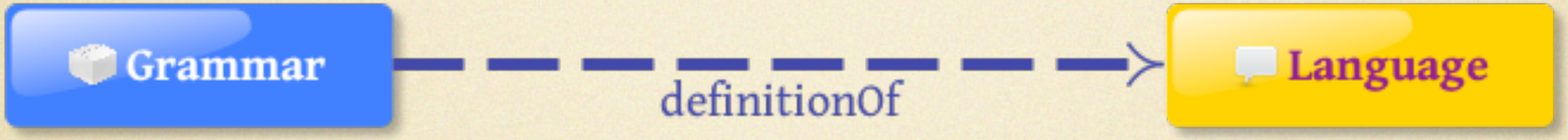
vadim@grammarware.net

Symposium on **La**nguage **C**omposition and **M**odularity

Vadim Zaytsev, SWAT, CWI

Introduction to grammarware

What is a grammar?



- Structural description in software systems
- Description of structures used in software systems

P. Klint, R. Lämmel, C. Verhoef, *Toward an Engineering Discipline for Grammarware*. TOSEM 2005.

Grammar use case

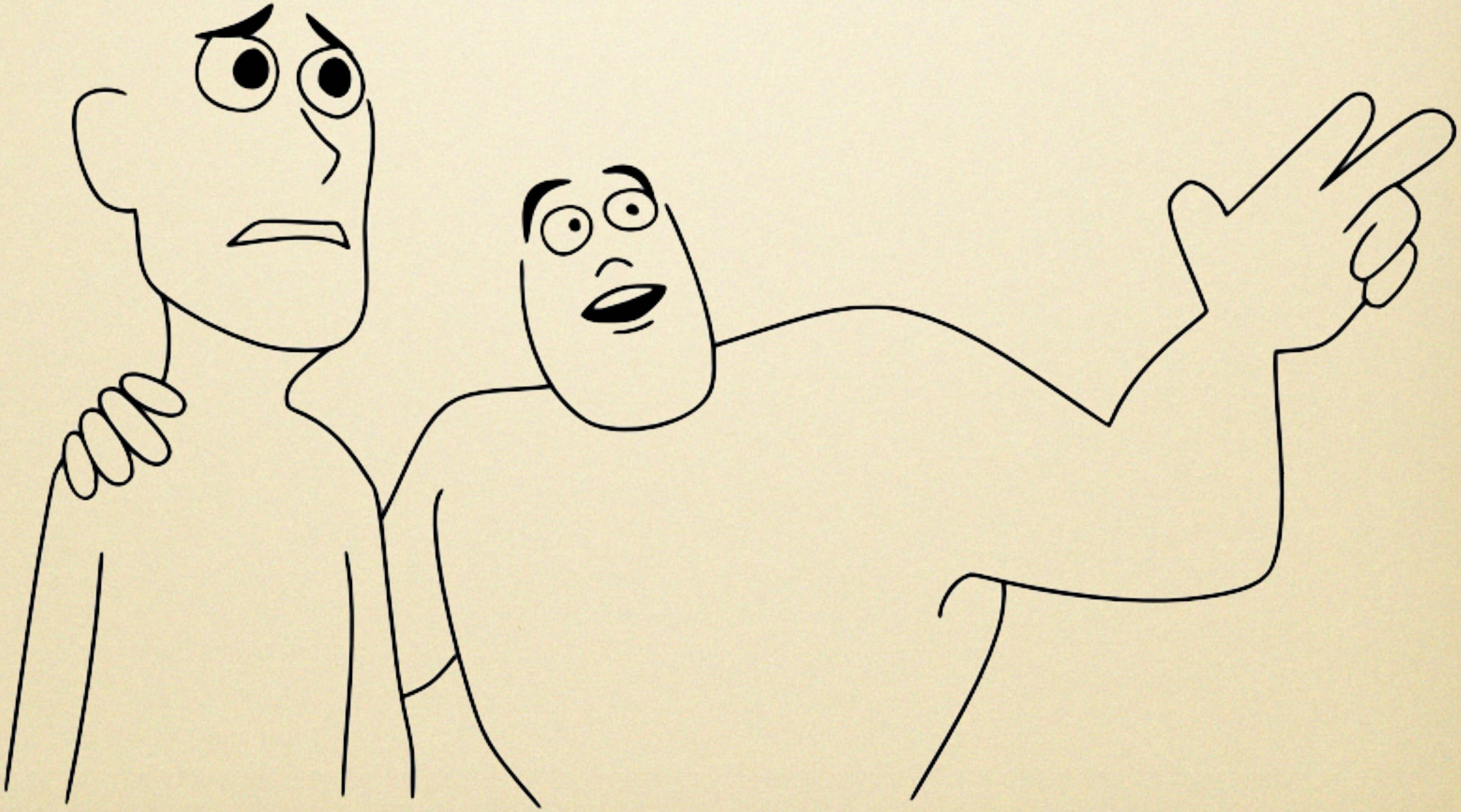
- Abstract use case
  - syntax definition
  - exchange format
  - interaction protocol
  - data model
  - domain model
  - metamodel
  - …
- Concrete use case
  - parsing
  - serialisation
  - renovation
  - refactoring
  - static analysis
  - reengineering
  - …

P. Klint, R. Lämmel, C. Verhoef, *Toward an Engineering Discipline for Grammarware*. TOSEM 2005.

Grammarware

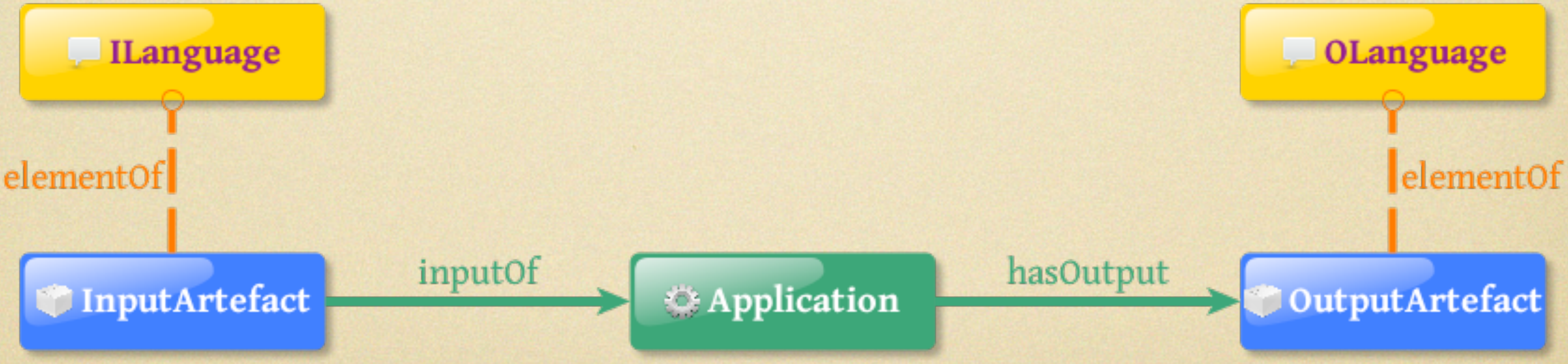
- Parser
- Compiler
- Interpreter
- Pretty-printer
- Scanner
- Browser
- Static checker
- Structural editor
- IDE
- DSL framework
- Preprocessor
- Postprocessor
- Model checker
- Refactorer
- Code slicer
- API
- XMLware
- Modelware
- Language workbench
- Reverse engineering tool
- Benchmark
- Recommender
- Renovation tool

Grammars everywhere

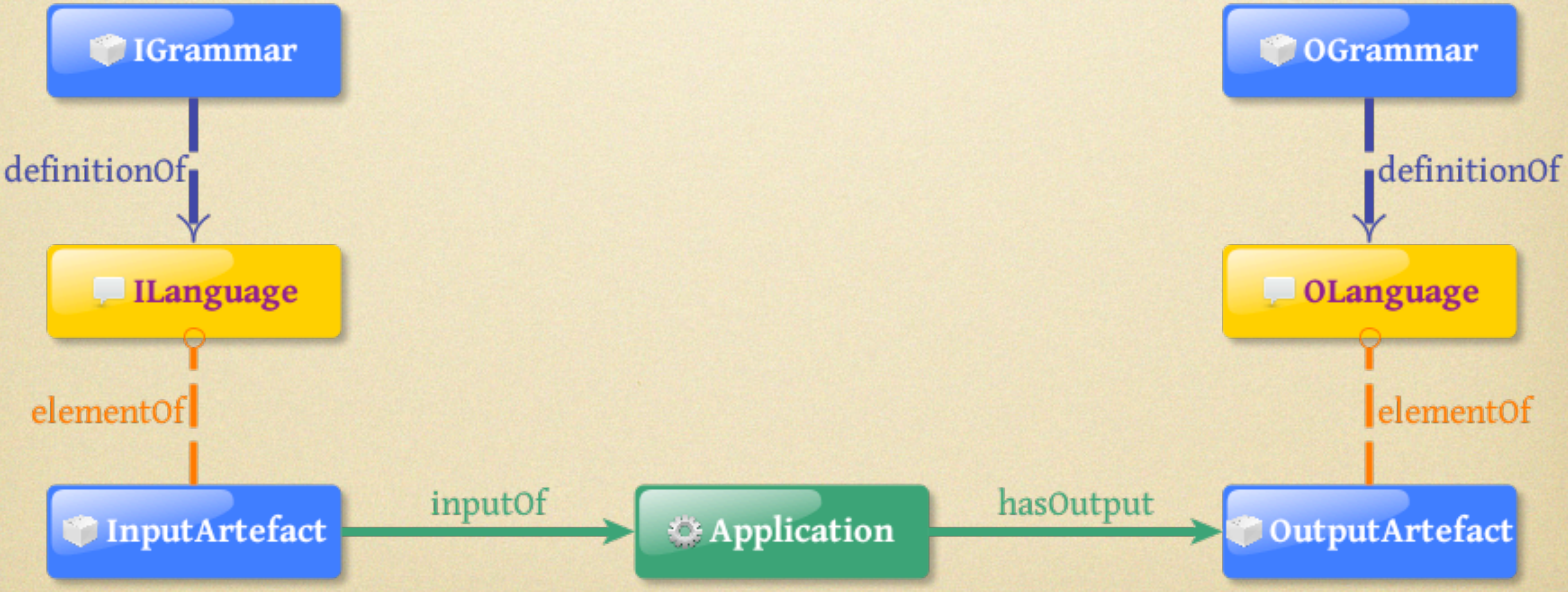


Grammarware

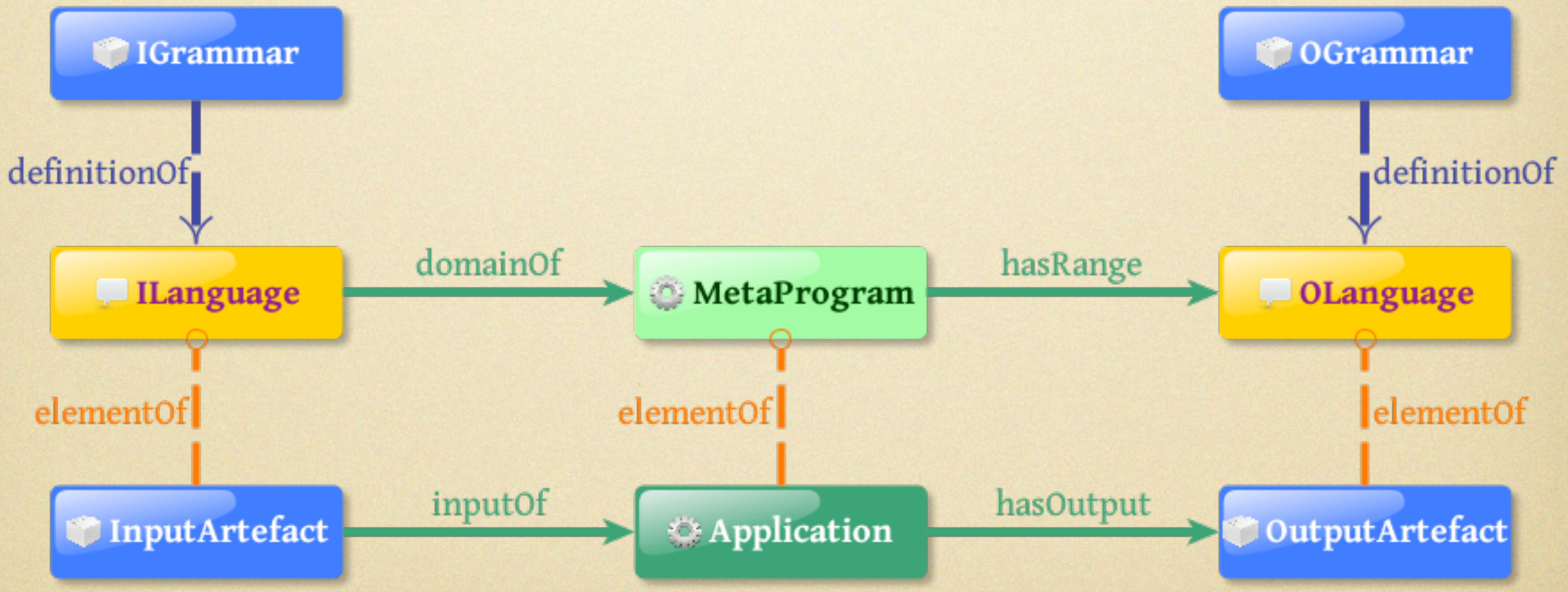
Grammarware



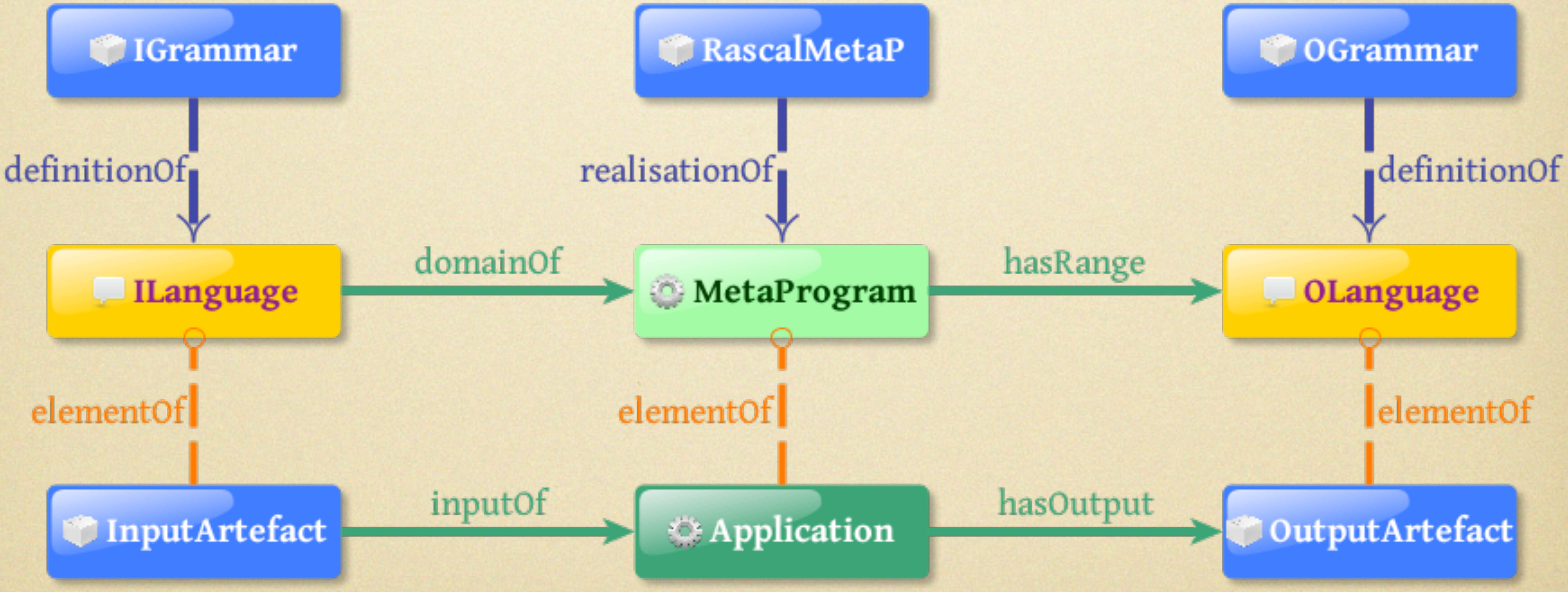
Grammarware



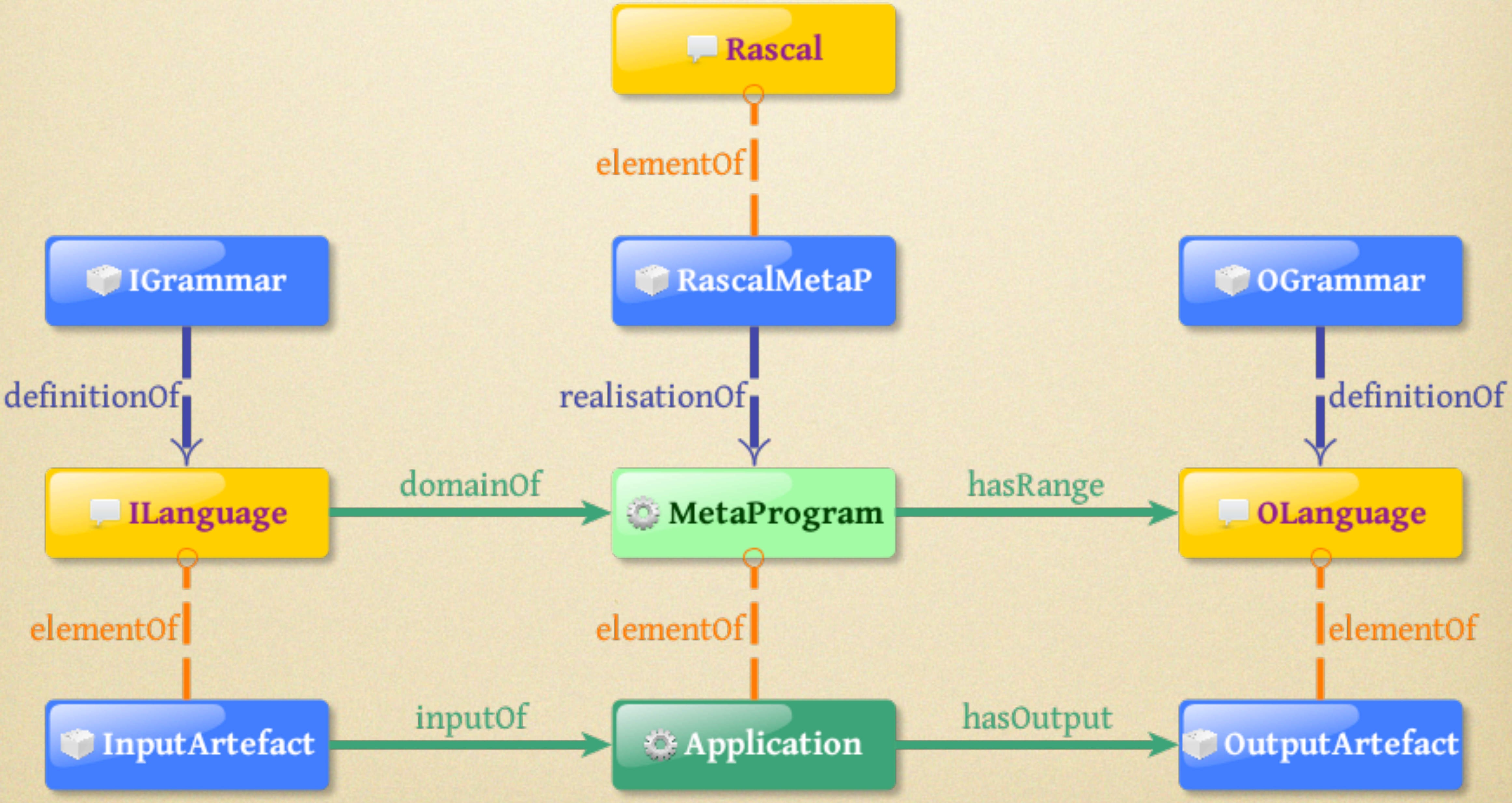
Grammarware



Grammarware

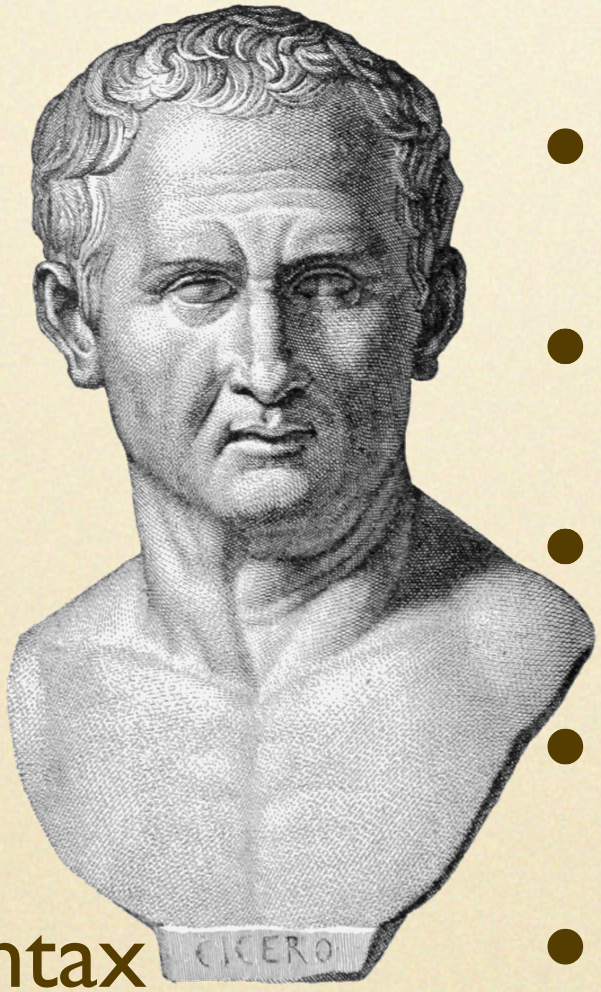


Grammarware



Grammar decomposition

Quo usque tandem?



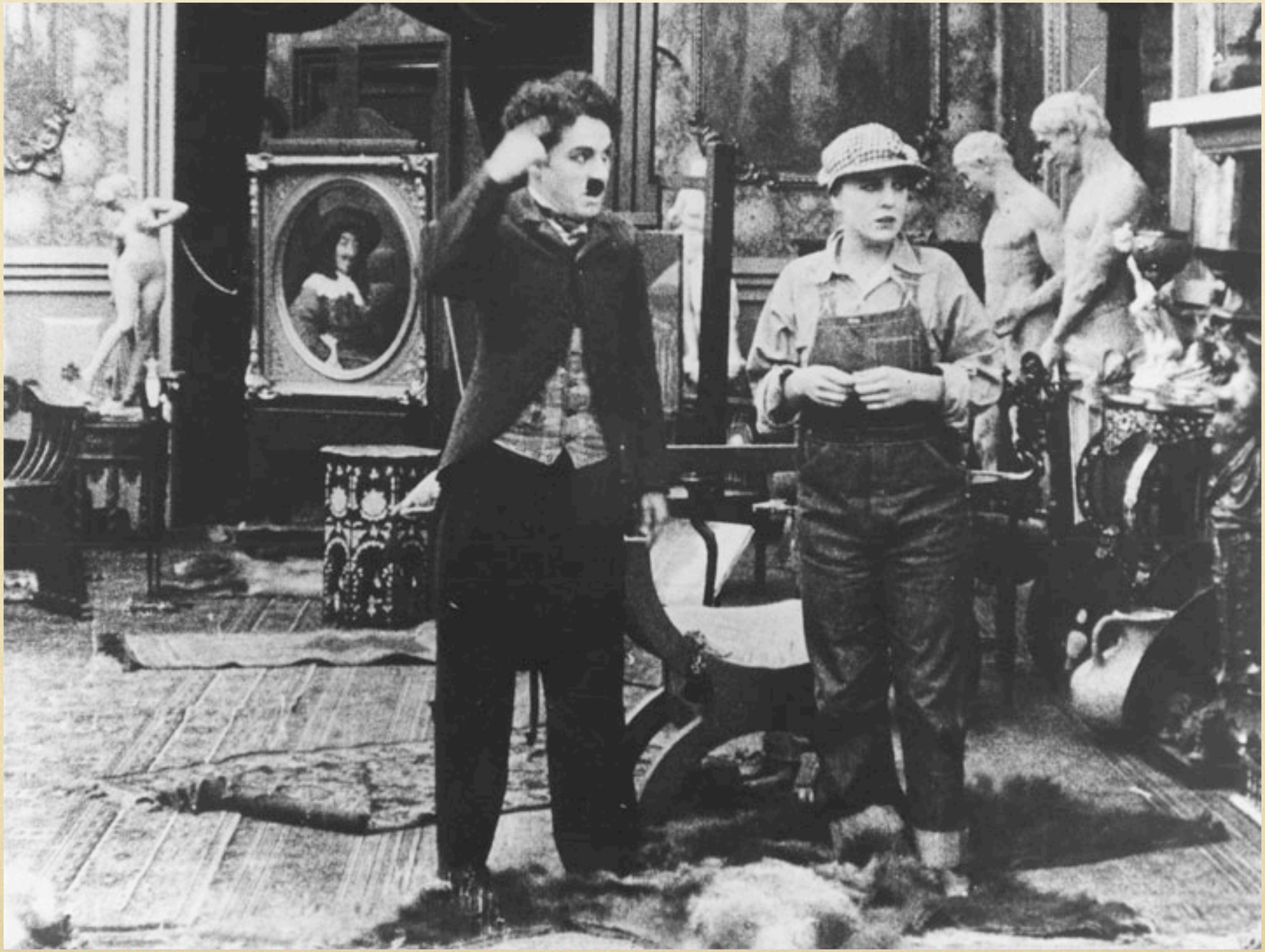
- Lexical syntax:
  - character level (tokenisation)
  - block level (indentation)
- Preprocessing syntax
  - comments
  - directives

- Base syntax
- Syntax highlighting
- Processing order
- Filtering / disambiguation
- Error handling
- Tree construction
- AST format

http://commons.wikimedia.org/wiki/File:Cicero.PNG

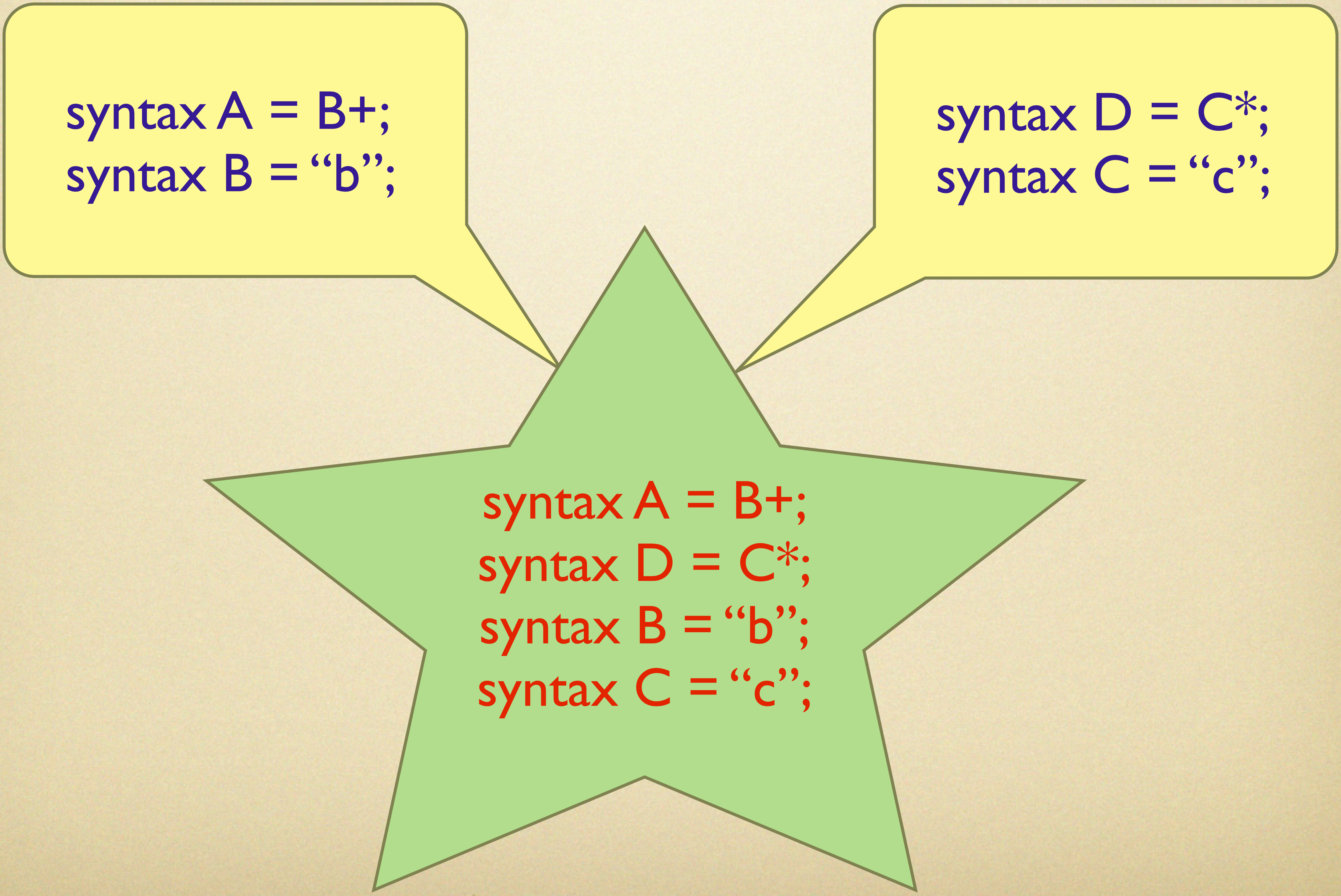
Grammar composition

Behind the Screen / Unknown Chaplin



Fair (re)use, picture is the courtesy of chaplin.bfi.org.uk.

Grammar composition

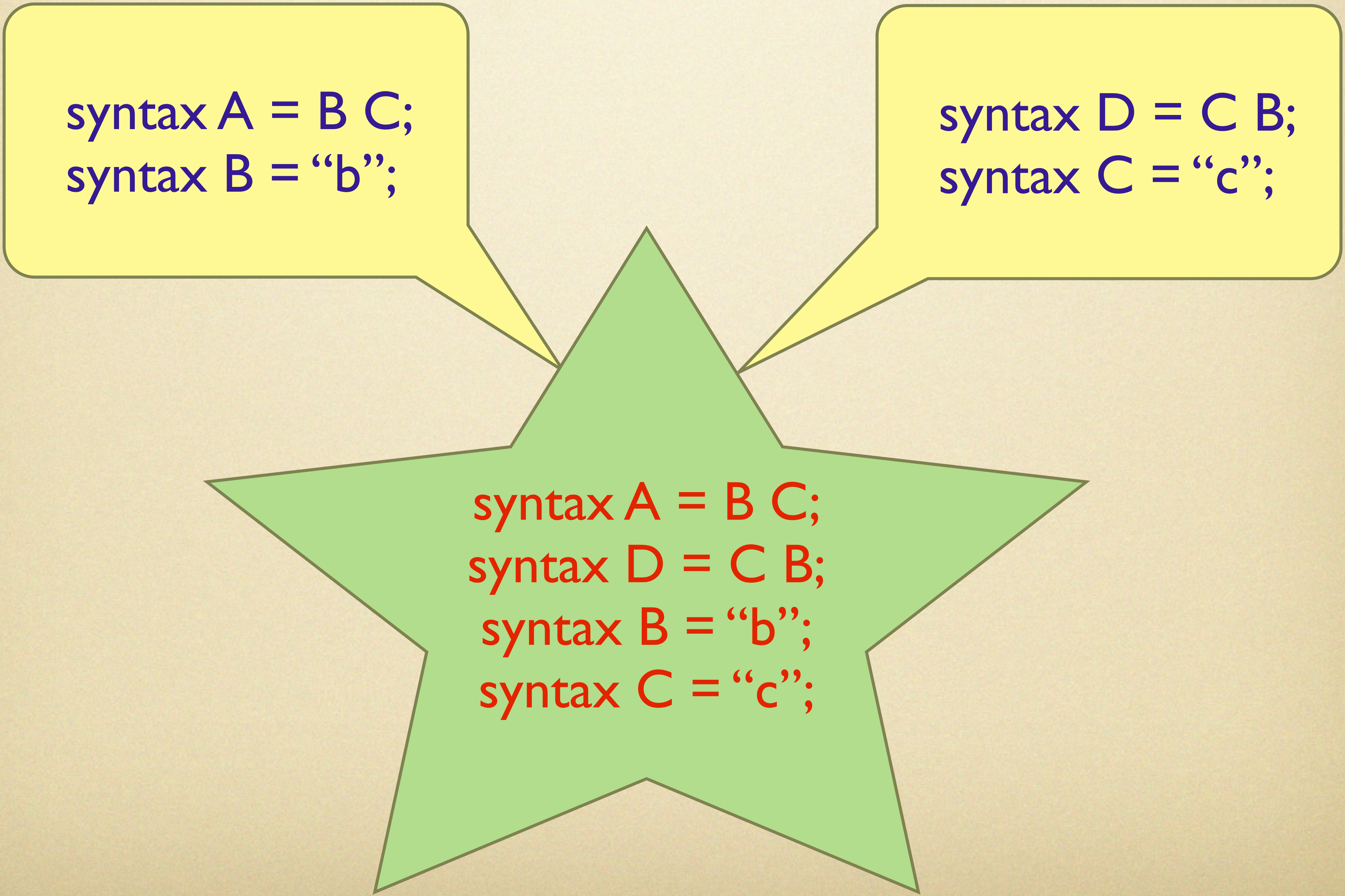


```
syntax A = B+;
syntax B = “b”;
```

```
syntax D = C*;
syntax C = “c”;
```

```
syntax A = B+;
syntax D = C*;
syntax B = “b”;
syntax C = “c”;
```

Grammar composition

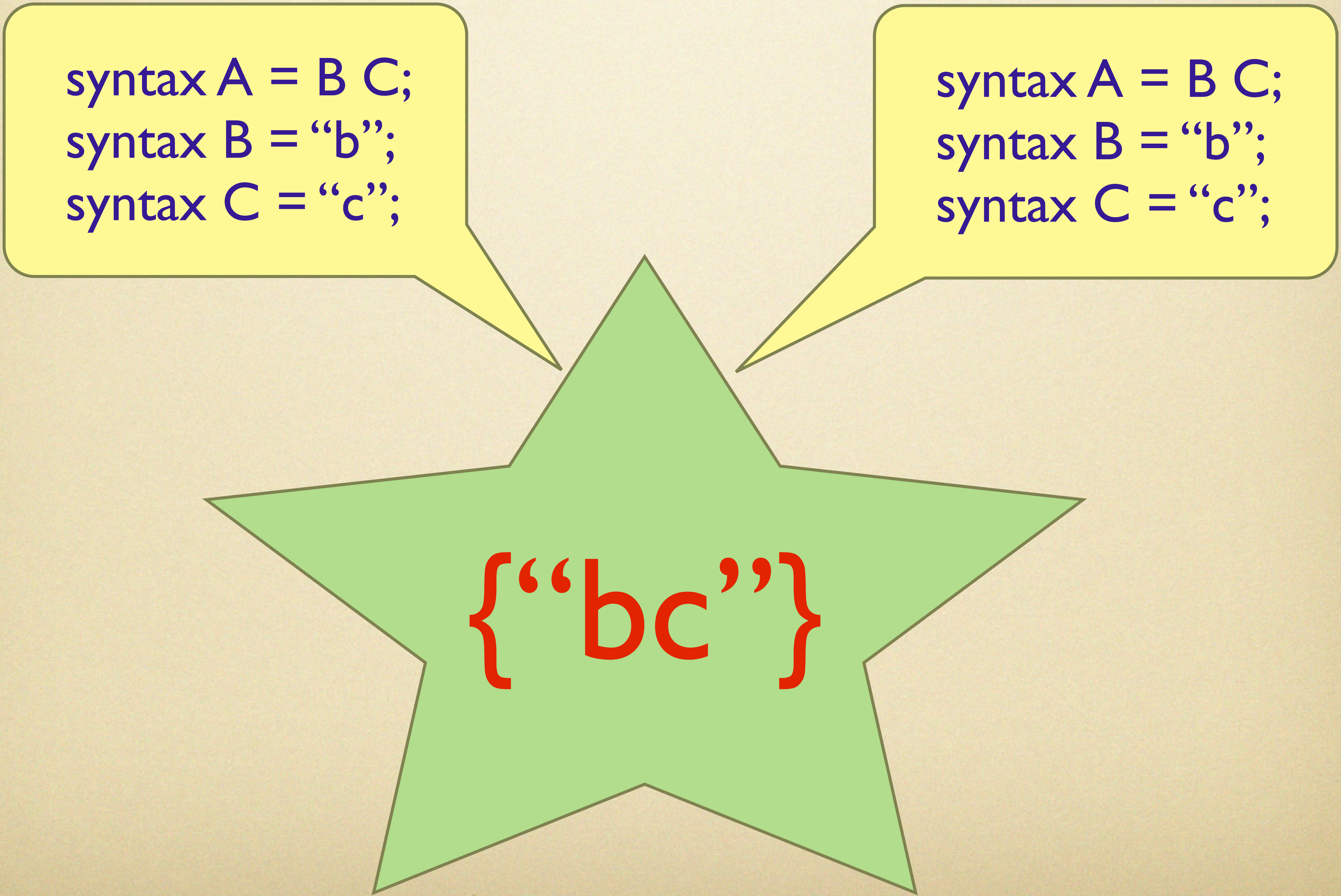


```
syntax A = B C;
syntax B = “b”;
```

```
syntax D = C B;
syntax C = “c”;
```

```
syntax A = B C;
syntax D = C B;
syntax B = “b”;
syntax C = “c”;
```

Grammar composition



```
syntax A = B C;
syntax B = “b”;
syntax C = “c”;
```

```
syntax A = B C;
syntax B = “b”;
syntax C = “c”;
```

{“bc”}

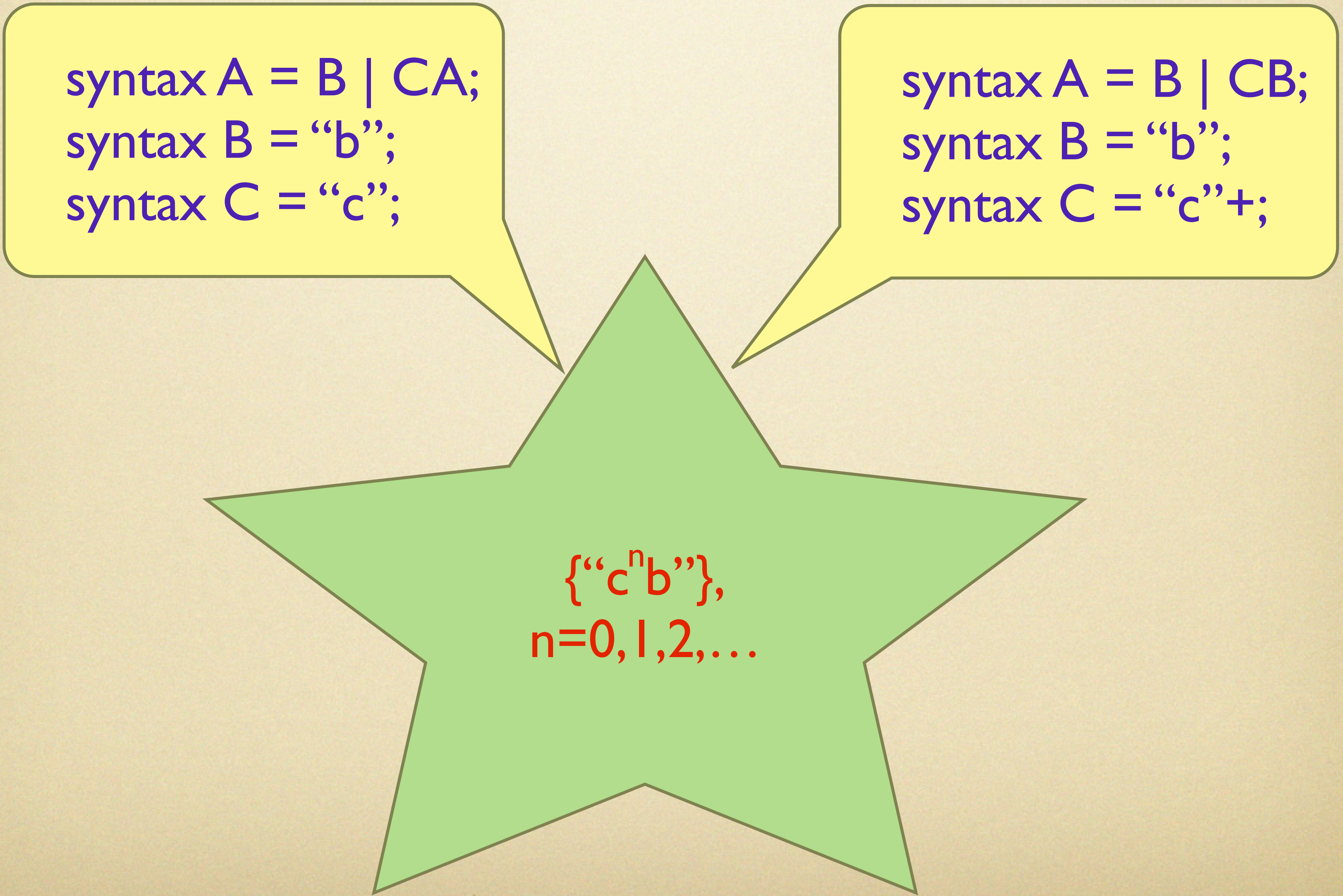
Grammar composition

```
syntax A = B C;
syntax B = “b”;
syntax C = “c”;
```

```
syntax A = C B;
syntax B = “c”;
syntax C = “b”;
```

{“bc”,
~~“cb”~~}

Grammar composition

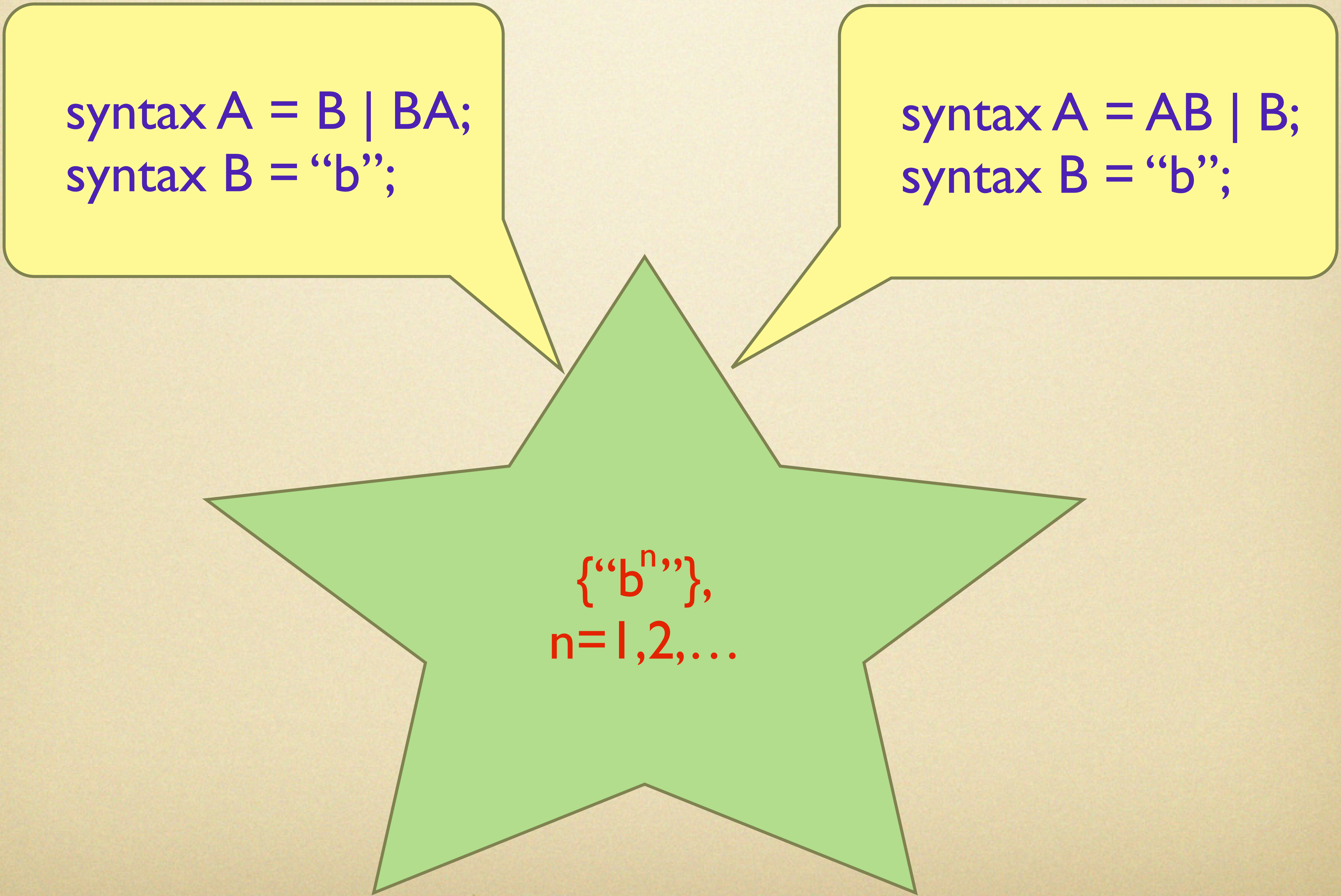


```
syntax A = B | CA;
syntax B = “b”;
syntax C = “c”;
```

```
syntax A = B | CB;
syntax B = “b”;
syntax C = “c”+;
```

{“c^nb”},
n=0,1,2,…

Grammar composition

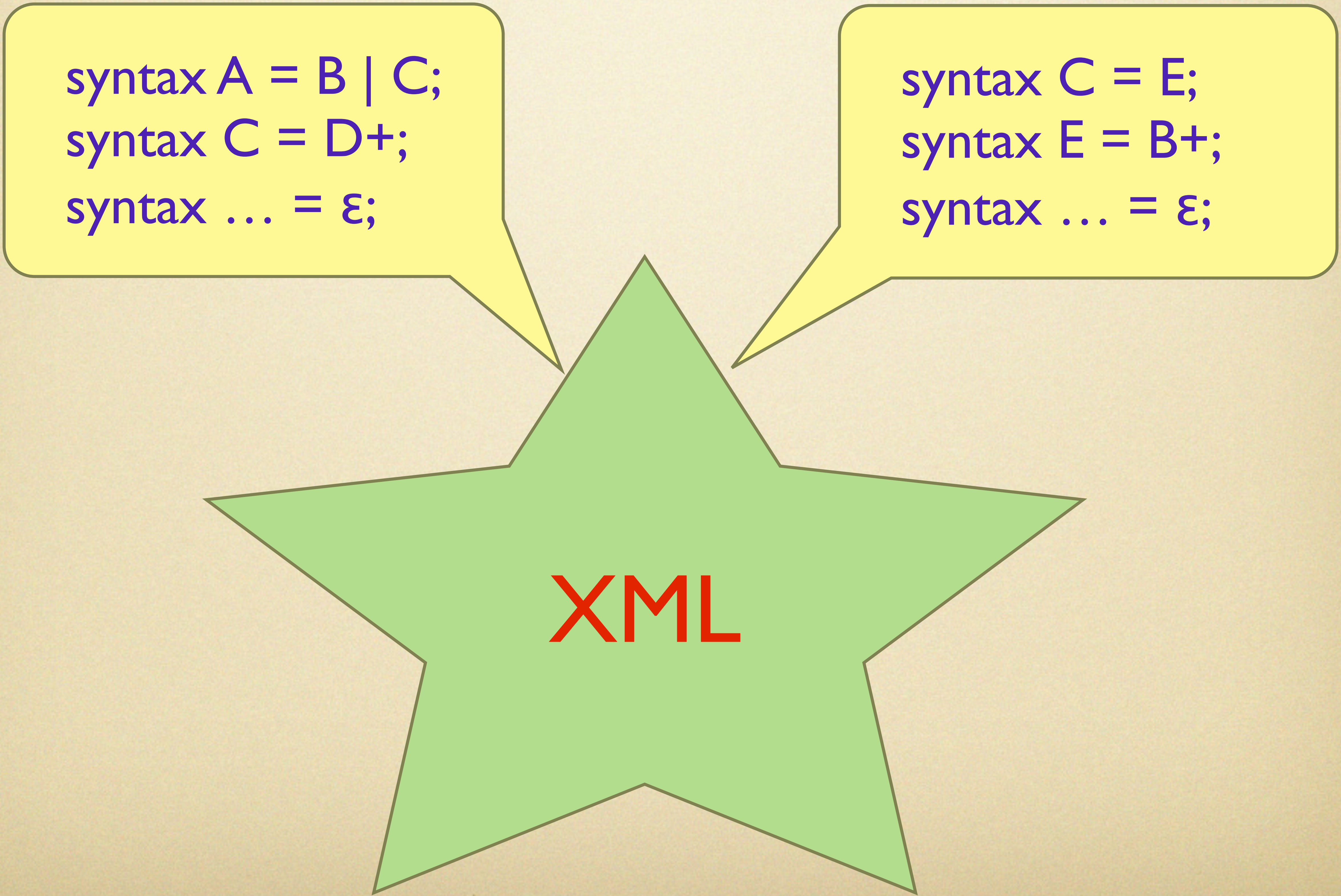


syntax A = B | BA;
syntax B = “b”;

syntax A = AB | B;
syntax B = “b”;

{“b^{n}”},
n=1,2,…

Grammar composition

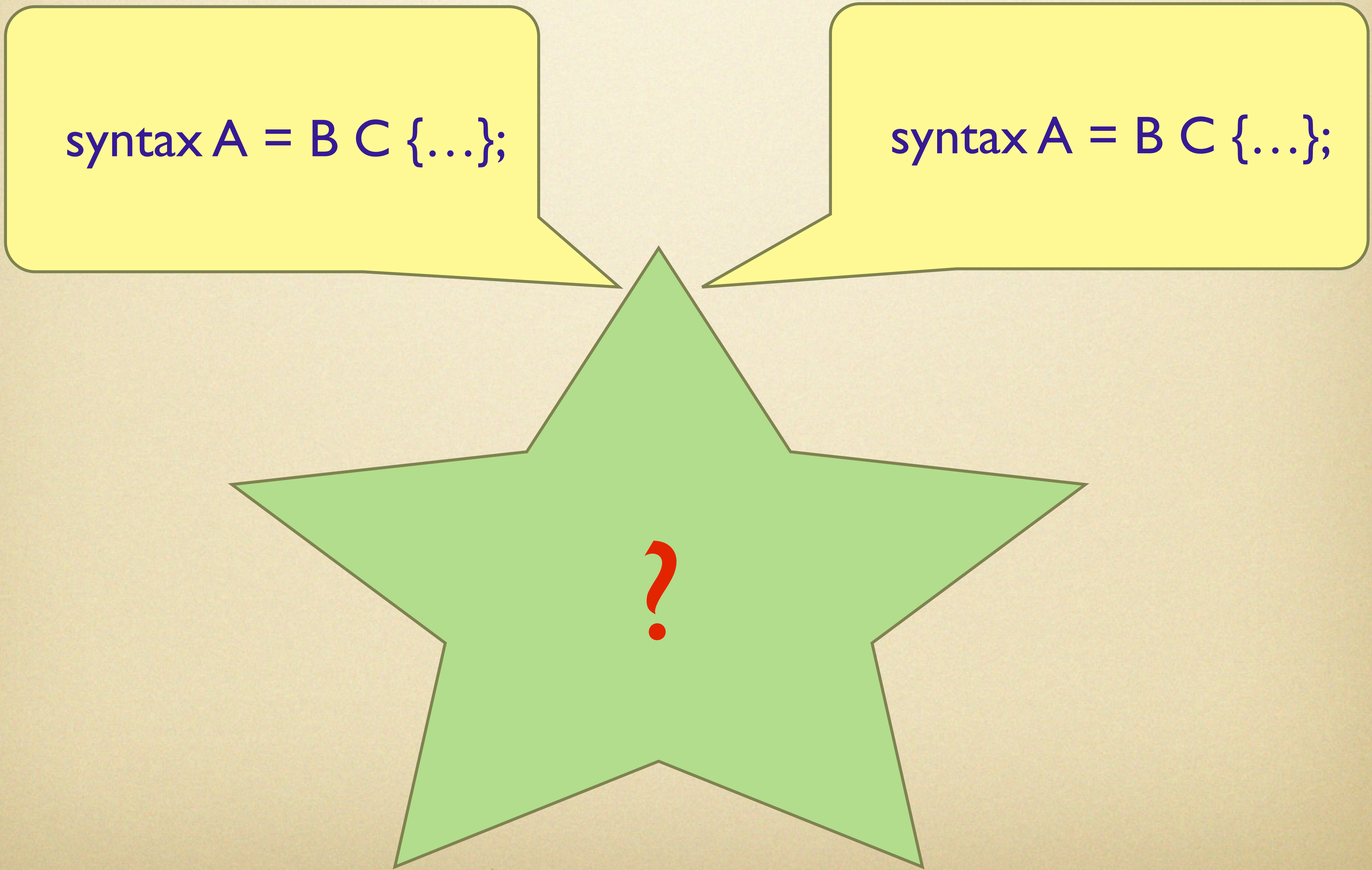


```
syntax A = B | C;
syntax C = D+;
syntax … = ε;
```

```
syntax C = E;
syntax E = B+;
syntax … = ε;
```

XML

Grammar composition

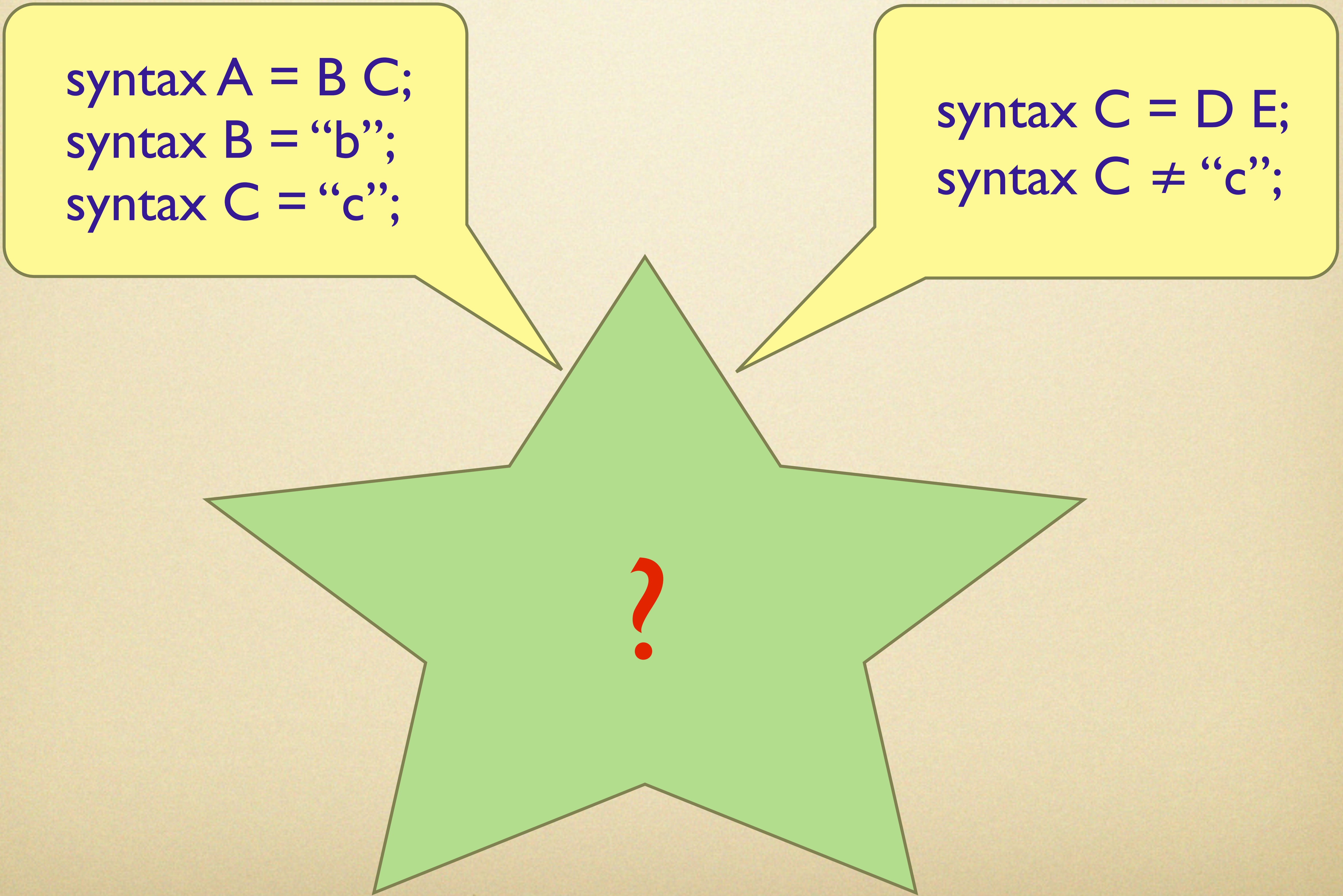


syntax A = B C {…};

syntax A = B C {…};

?

Grammar composition



```
syntax A = B C;
syntax B = “b”;
syntax C = “c”;
```

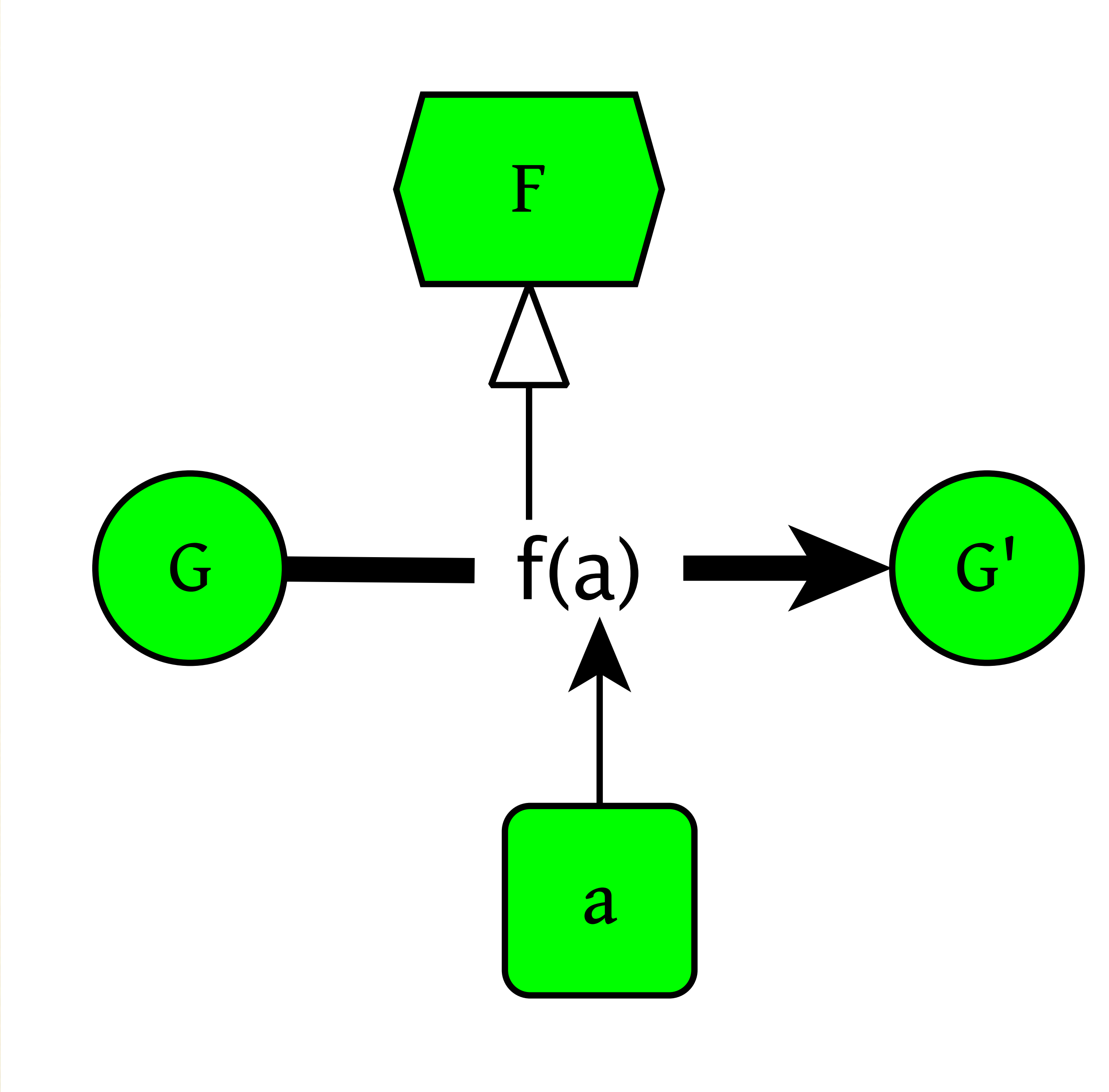
```
syntax C = D E;
syntax C ≠ “c”;
```

Adjacent topics:

pgt gc ngt no

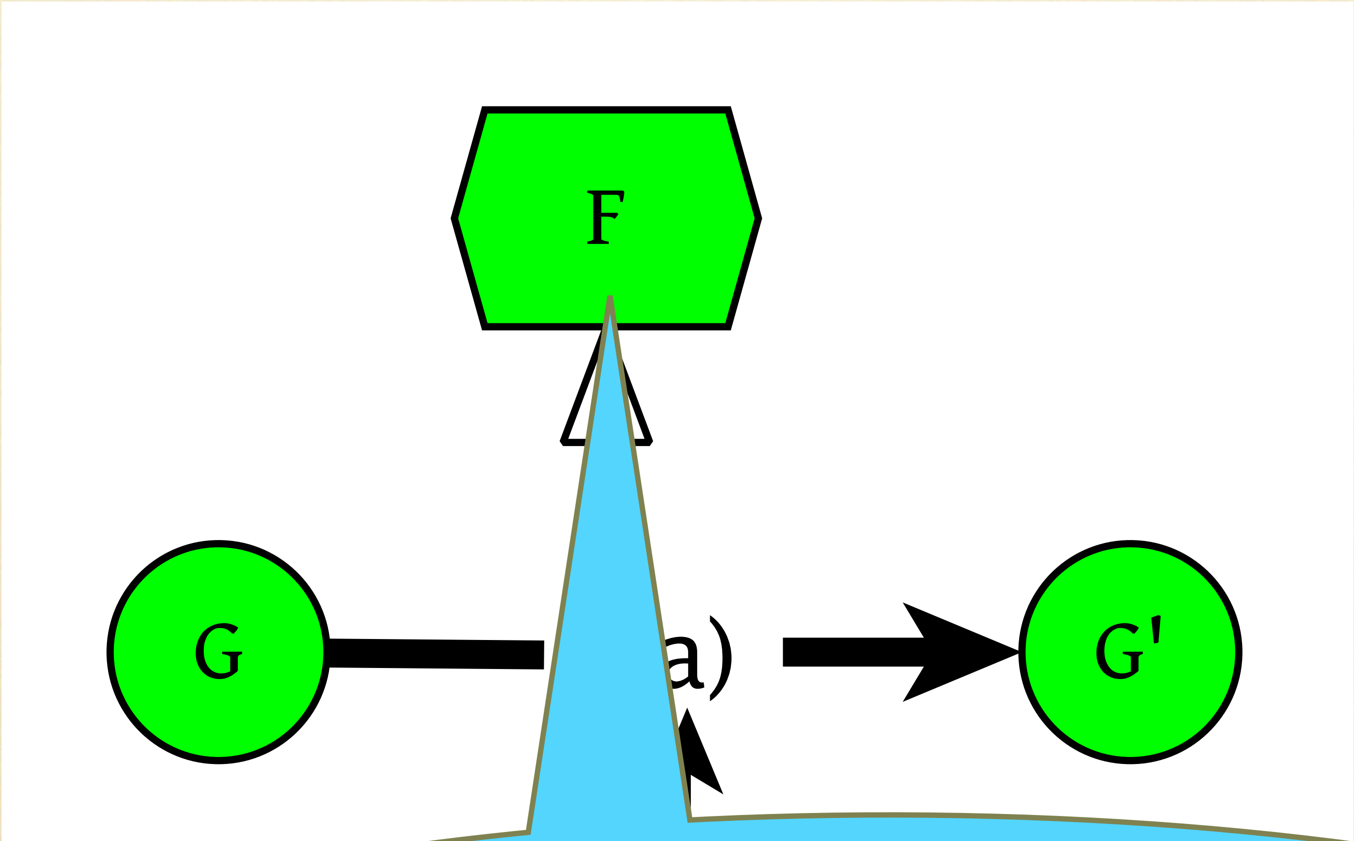
Programmable Grammar Transformations

Programmable G xformation



Ad hoc megamodel shown at IPA Spring Days

Programmable G xformation

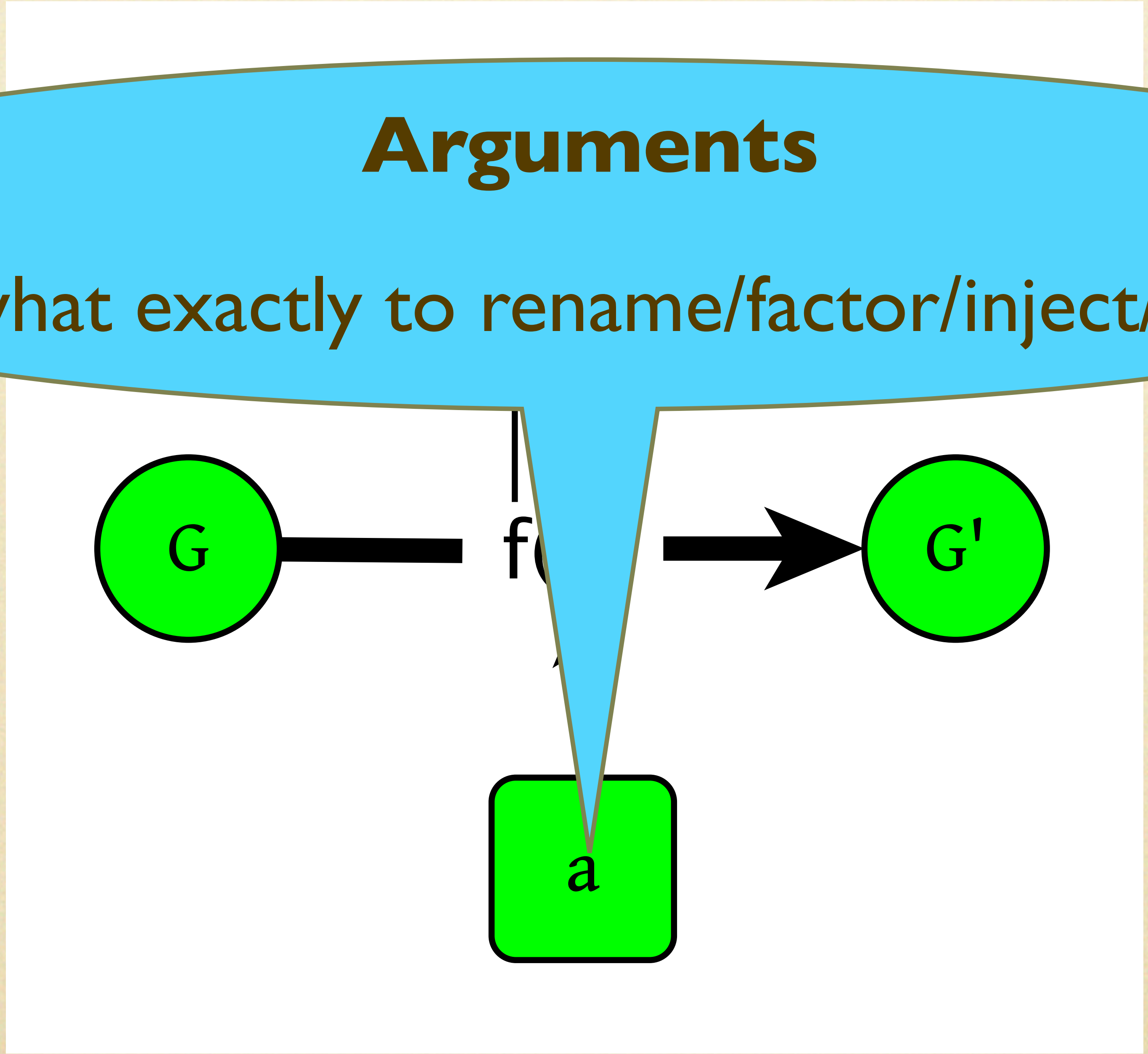


Operator

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

Ad hoc megamodel shown at IFA Spring Days

Programmable G xformation



Arguments

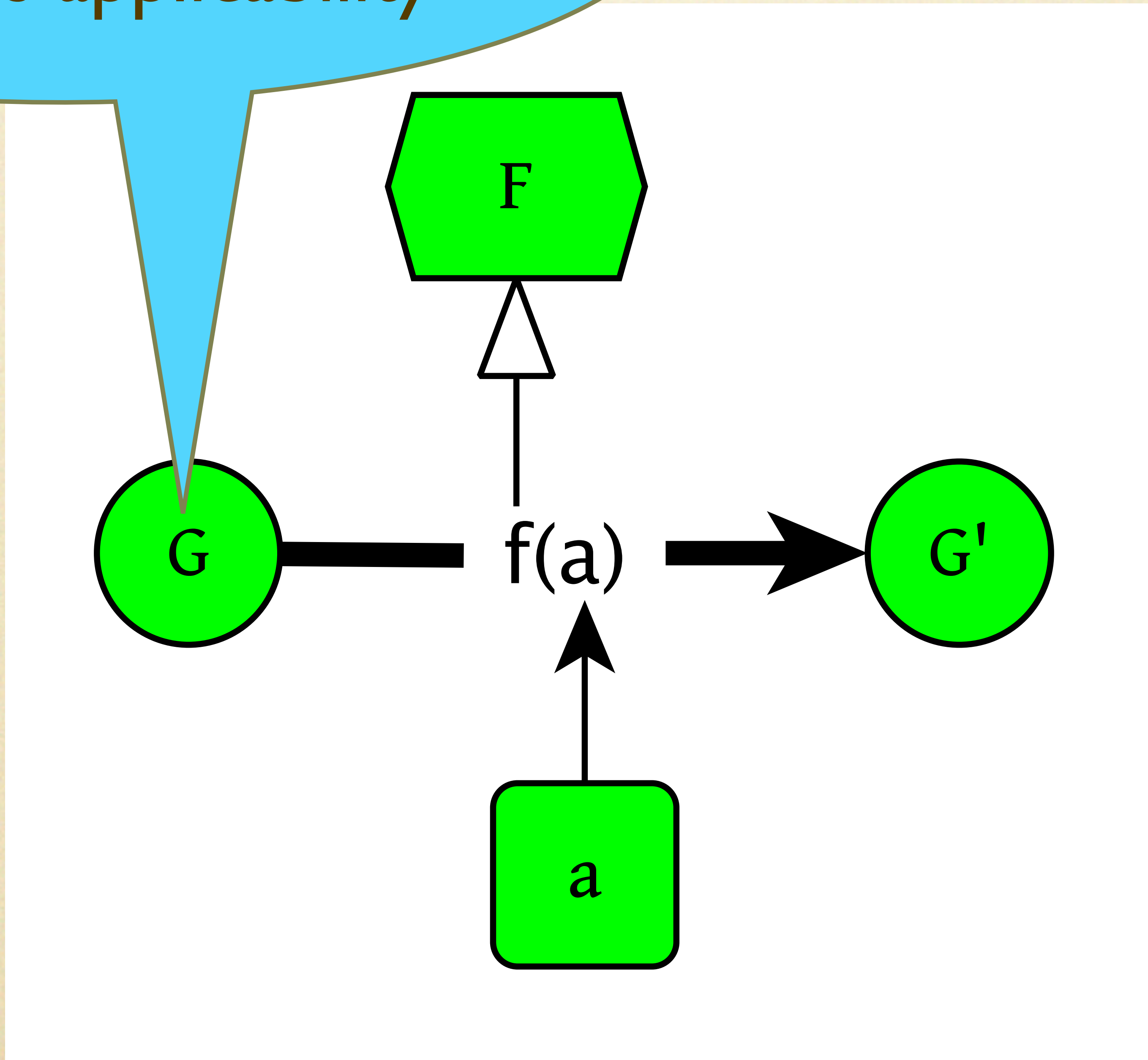
- what exactly to rename/factor/inject/…?

Ad hoc megamodel shown at IPA Spring Days

P le G xformation

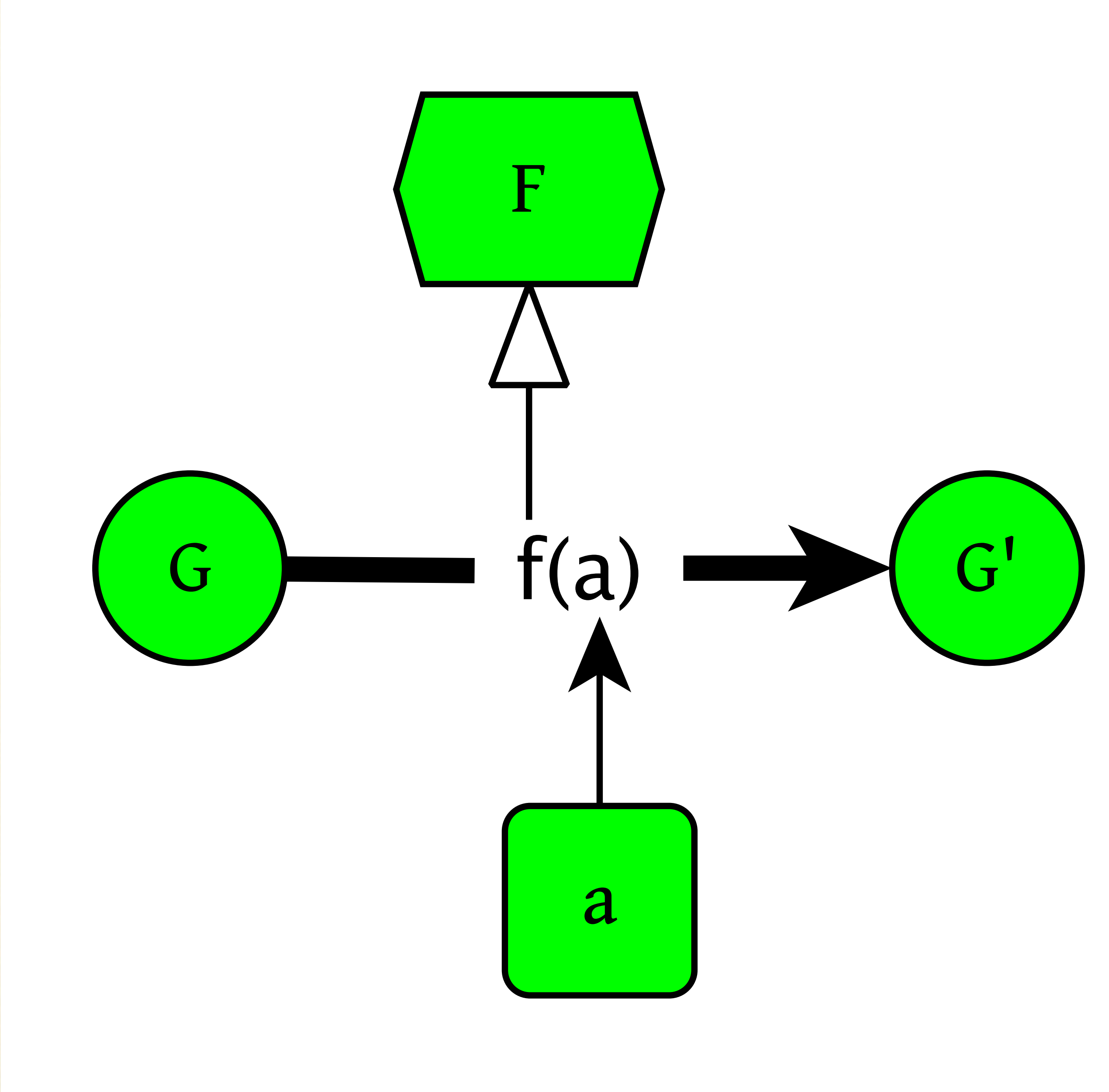
Input grammar

- determines applicability



Ad hoc megamodel shown at IPA Spring Days

Programmable G xformation



Ad hoc megamodel shown at IPA Spring Days

XBGF Operator Suite

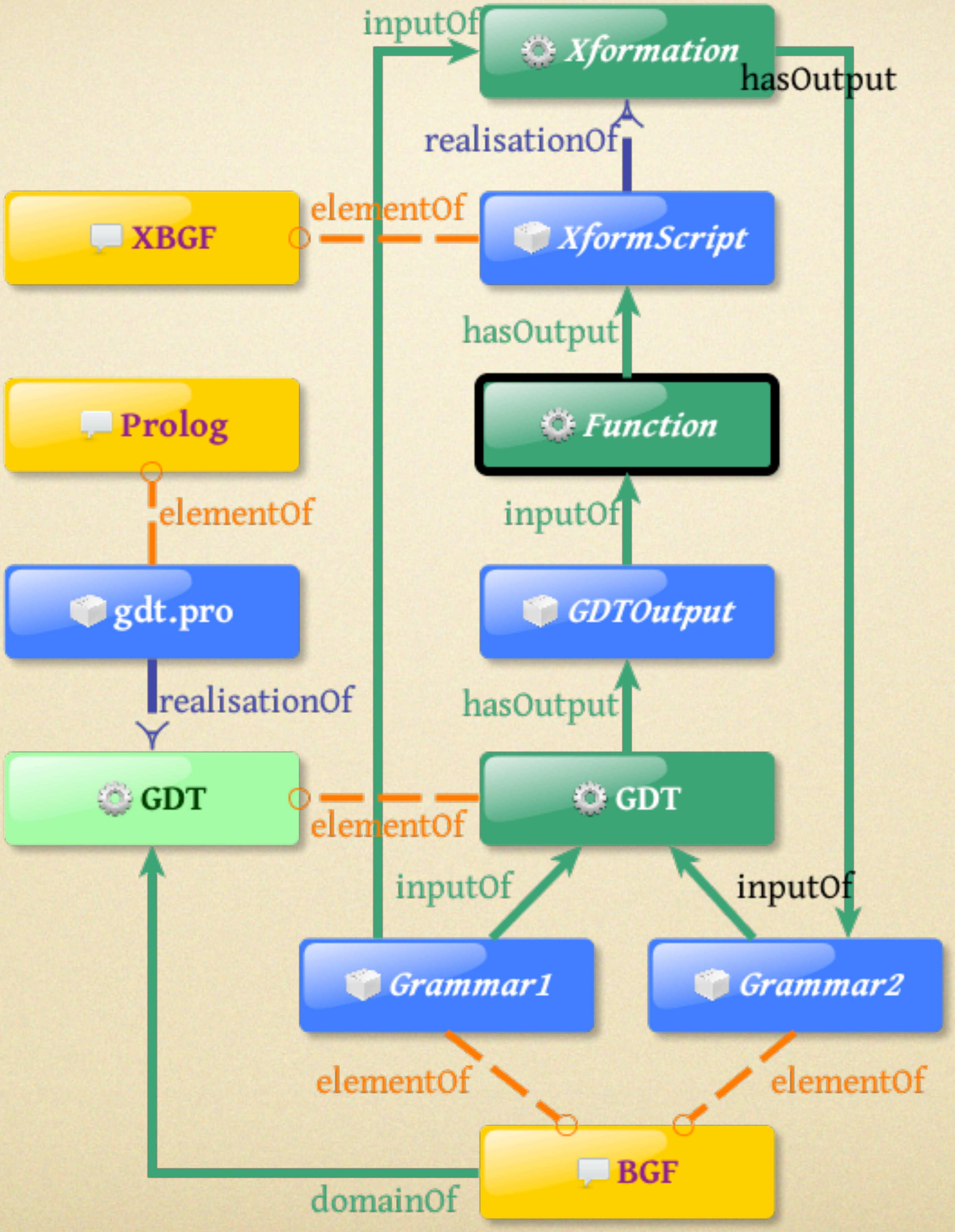
- Semantic-preserving operators
  - fold, unfold, extract, inline, massage, factor, deyaccify, …
- (Some) semantic-preserving operators
  - permute, abstractize, concretize, designate, anonymize
- Language-increasing operators
  - add, appear, widen, upgrade, unite
- Language-decreasing operators
  -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

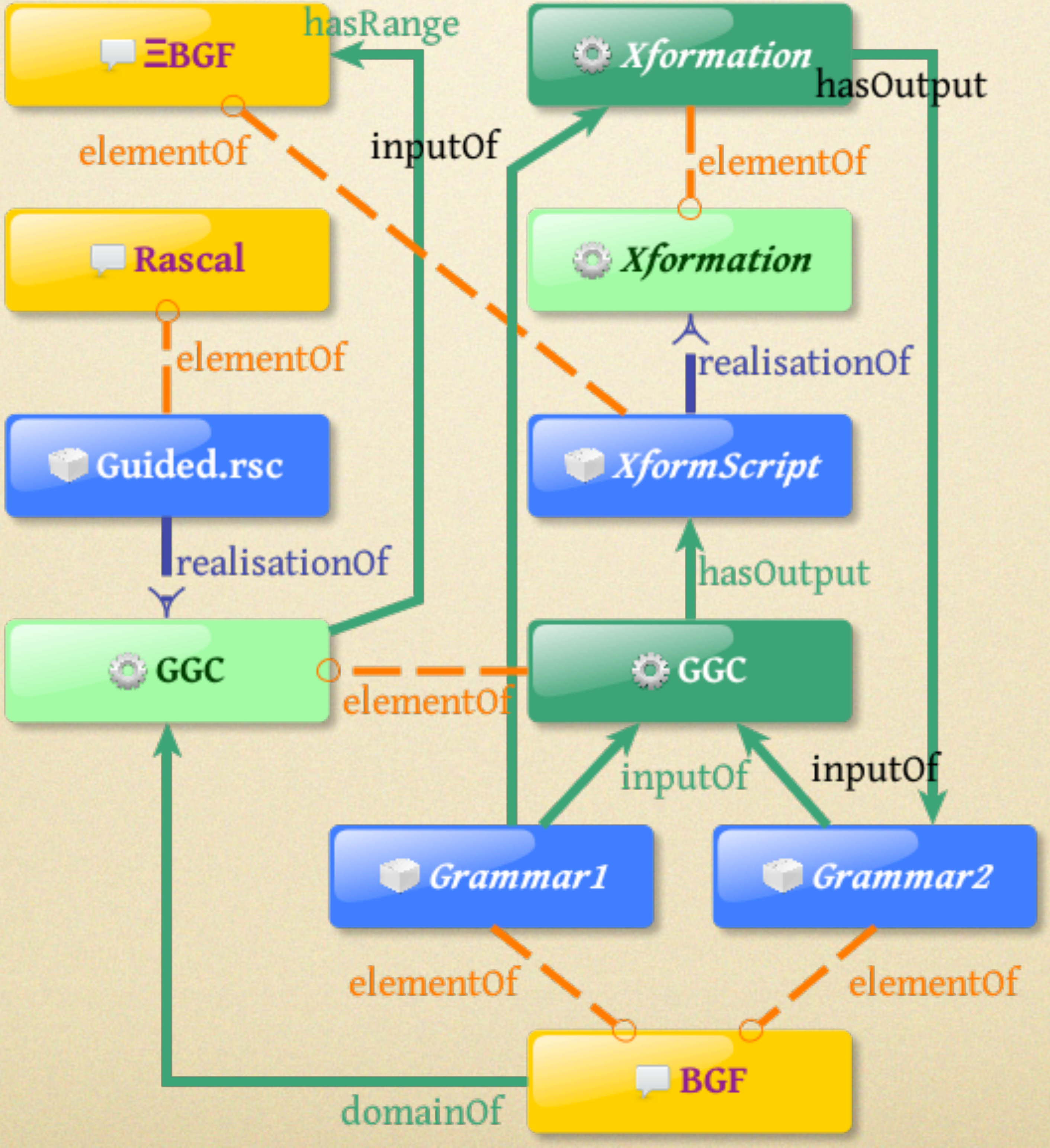
V. Zaytsev, *BGF Transformation Operator Suite v.1.0*, online, 2010.

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.

Guided Grammar Convergence


inputOf
Xformation
hasOutput
realisationOf
XBGF
elementOf
XformScript
hasOutput
Prolog
Function
elementOf
inputOf
gdt.pro
GDTOutput
realisationOf
hasOutput
GDT
GDT
elementOf
inputOf
inputOf
Grammar1
Grammar2
elementOf
elementOf
BGF
domainOf



ΞBGF
hasRange
Xformation
hasOutput
elementOf
inputOf
elementOf
Rascal
Xformation
elementOf
realisationOf
Guided.rsc
XformScript
realisationOf
hasOutput
GGC
elementOf
GGC
inputOf
inputOf
Grammar1
Grammar2
elementOf
elementOf
BGF
domainOf

The most trivial case

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

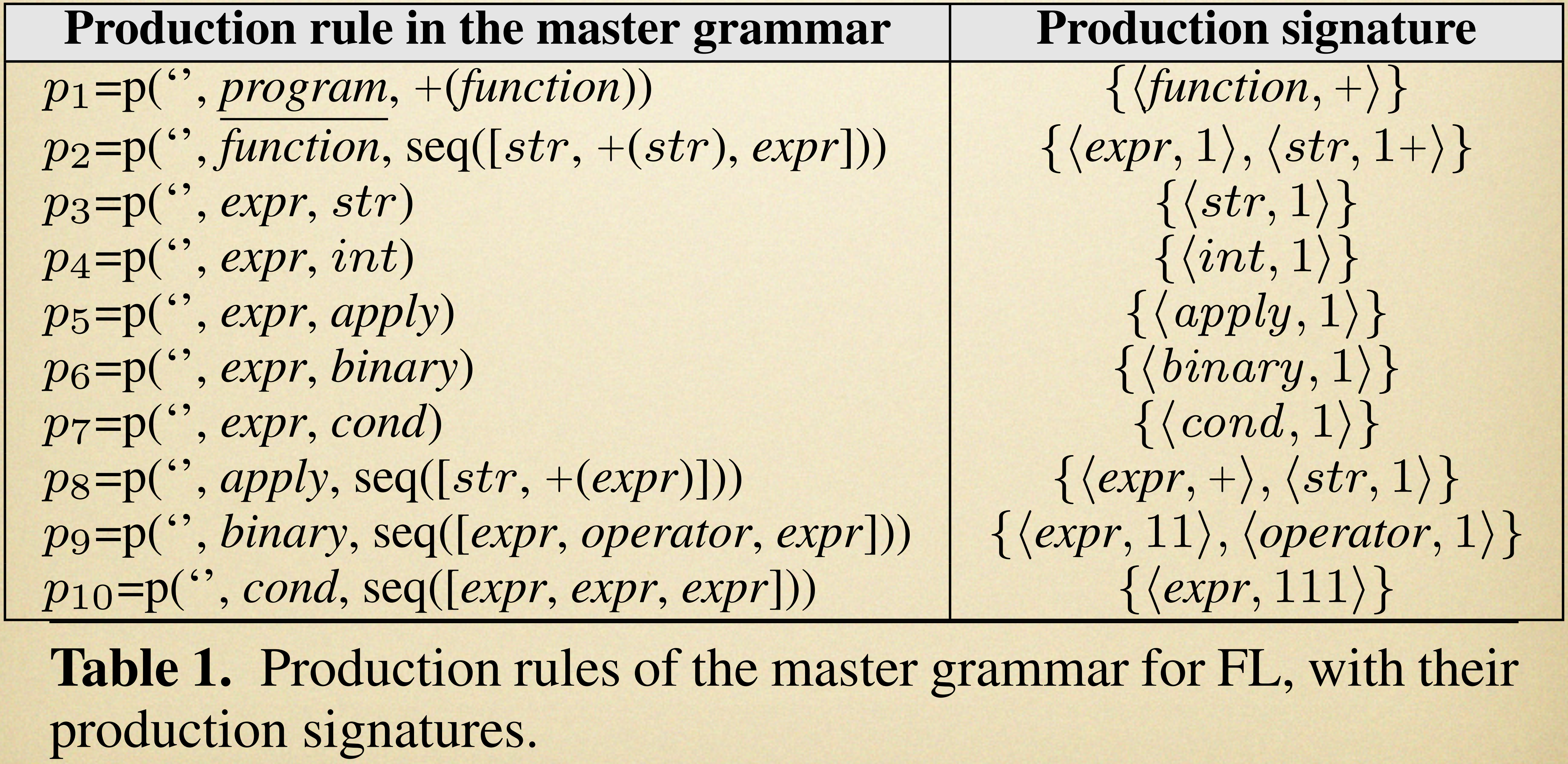
V. Zaytsev, *Guided Grammar Convergence*, CoRR abs/1207.6541, draft for 2013.

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*, CoRR abs/1207.6541, draft for 2013.

Nominal resolution



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

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

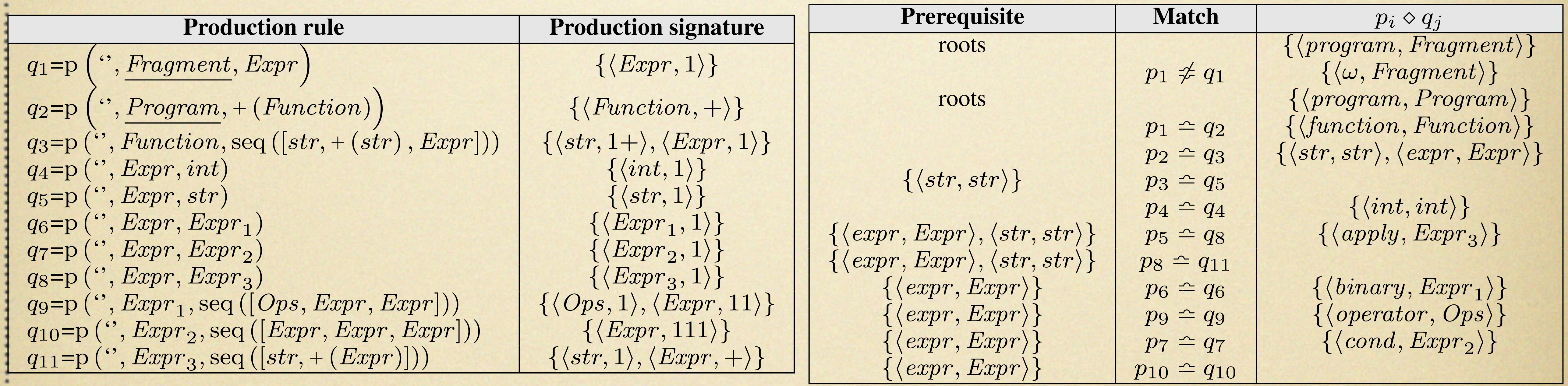
V. Zaytsev, Guided Grammar Convergence, CoRR abs/1207.6541, draft for 2013.

Definitions

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

V. Zaytsev, *Guided Grammar Convergence*, CoRR abs/1207.6541, draft for 2013.

Nominal resolution example



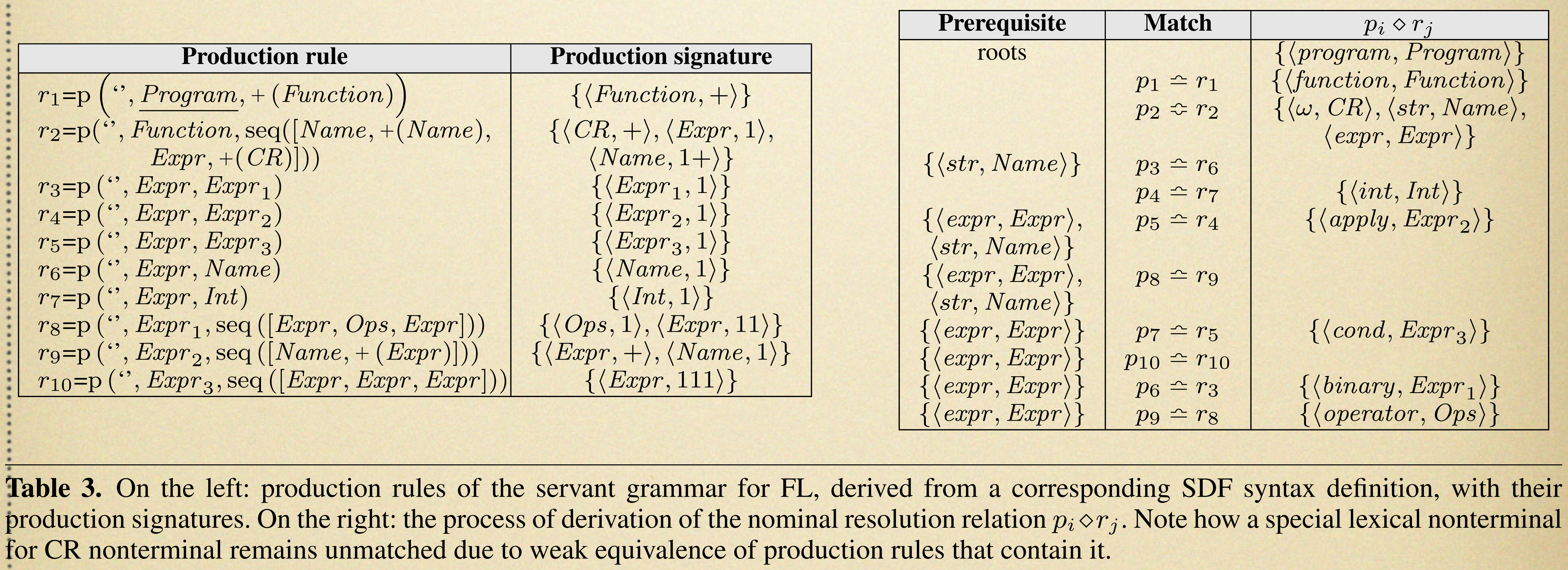
Production rule	**Production signature**
q_1=p $\left(``", \underline{Fragment}, Expr\right)$	$\{\langle Expr, 1\rangle\}$
q_2=p $\left(``", \underline{Program}, +(Function)\right)$	$\{\langle Function, +\rangle\}$
q_3=p $(``", Function, \mathrm{seq}([str, +(str), Expr]))$	$\{\langle str, 1+\rangle, \langle Expr, 1\rangle\}$
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 $(``", Expr_1, \mathrm{seq}([Ops, Expr, Expr]))$	$\{\langle Ops, 1\rangle, \langle Expr, 11\rangle\}$
q_{10}=p $(``", Expr_2, \mathrm{seq}([Expr, Expr, Expr]))$	$\{\langle Expr, 111\rangle\}$
q_{11}=p $(``", Expr_3, \mathrm{seq}([str, +(Expr)]))$	$\{\langle str, 1\rangle, \langle Expr, +\rangle\}$

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

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, CoRR abs/1207.6541, draft for 2013.

Nominal resolution example



Production rule	Production signature
r_1=p$\left(\text{‘’}, \underline{Program}, +(Function)\right)$	$\{\langle Function, +\rangle\}$
r_2=p(‘’, $Function$, 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 (‘’, $Expr_1$, seq ([$Expr$, Ops, $Expr$]))	$\{\langle Ops, 1\rangle, \langle Expr, 11\rangle\}$
r_9=p (‘’, $Expr_2$, seq ([$Name$, + ($Expr$)]))	$\{\langle Expr, +\rangle, \langle Name, 1\rangle\}$
r_{10}=p (‘’, $Expr_3$, 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, CoRR abs/1207.6541, draft for 2013.

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*, CoRR abs/1207.6541, draft for 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

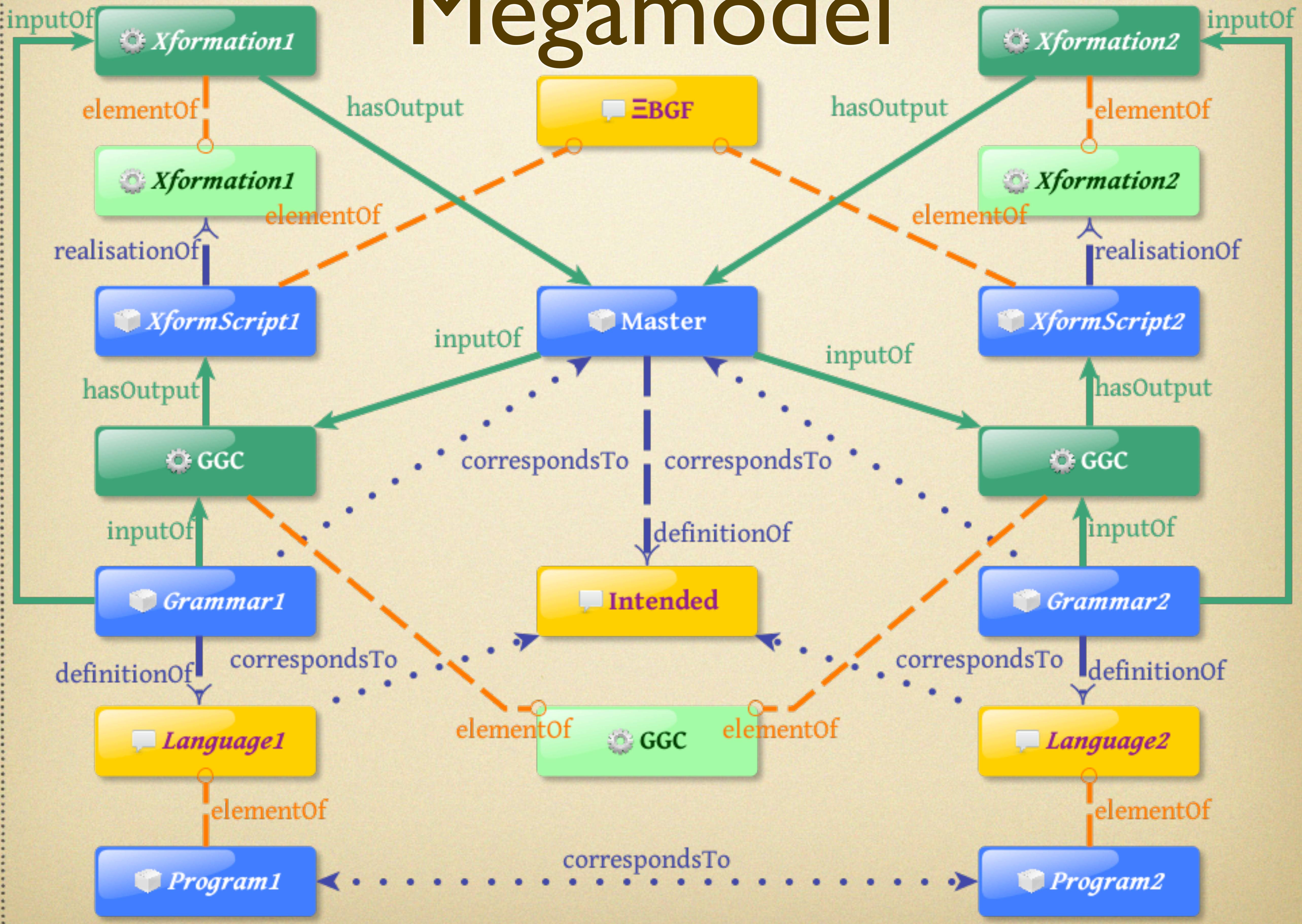
Master grammar	**Ecore**
exp: STR exp^{+}	ApplyExp: Function Exp^{+}

- Meaningful chain rules

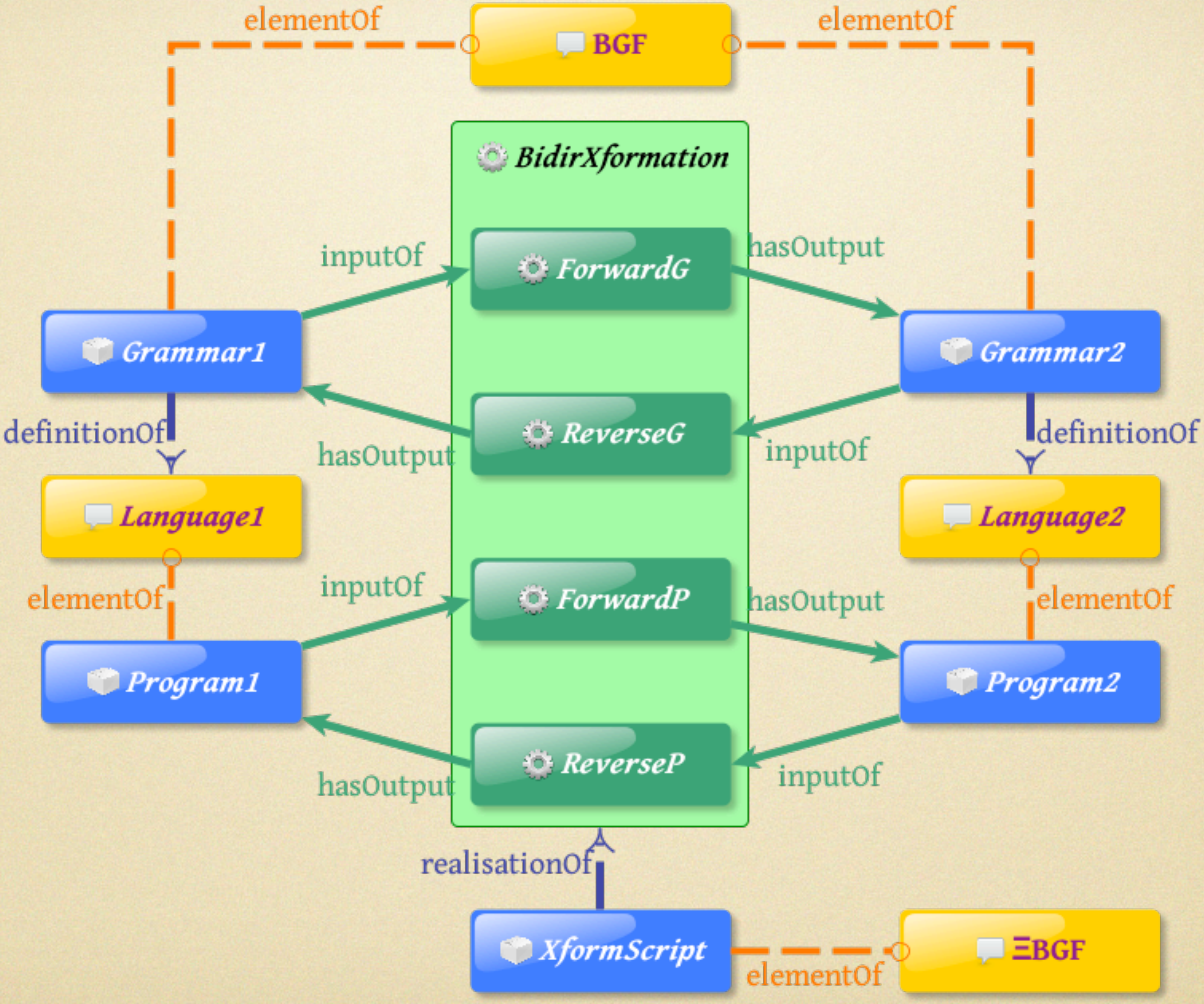
Master grammar	**Ecore**
exp: exp op exp	BinaryExp: PlusExp BinaryExp: MinusExp BinaryExp: EqualExp PlusExp: Exp Exp MinusExp: Exp Exp EqualExp: Exp Exp

CWI

Megamodel

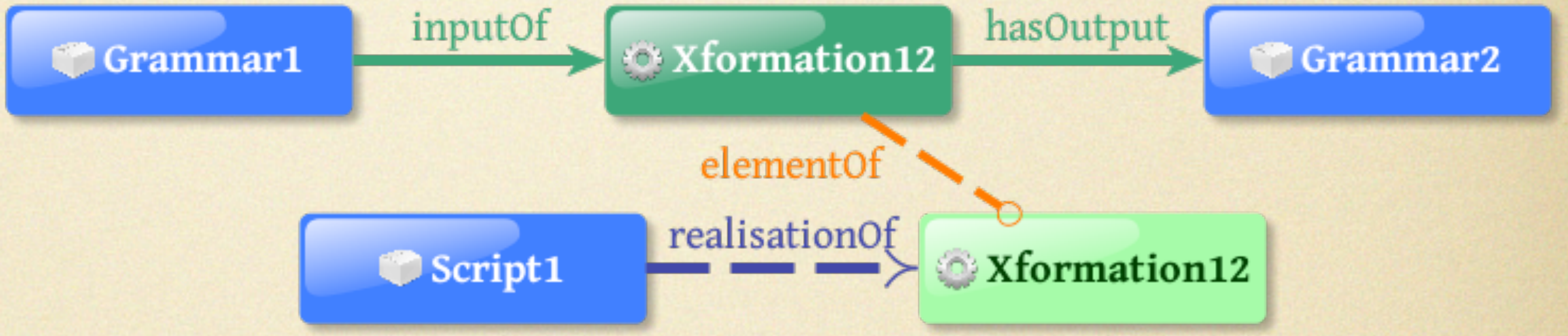


inputOf
Xformation1
hasOutput
ΞBGF
hasOutput
Xformation2
inputOf
elementOf
elementOf
Xformation1
Xformation2
elementOf
elementOf
realisationOf
realisationOf
XformScript1
Master
XformScript2
inputOf
inputOf
hasOutput
hasOutput
GGC
correspondsTo
correspondsTo
GGC
inputOf
definitionOf
inputOf
Grammar1
Intended
Grammar2
correspondsTo
correspondsTo
definitionOf
definitionOf
Language1
elementOf
GGC
elementOf
Language2
elementOf
elementOf
correspondsTo
Program1
Program2


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

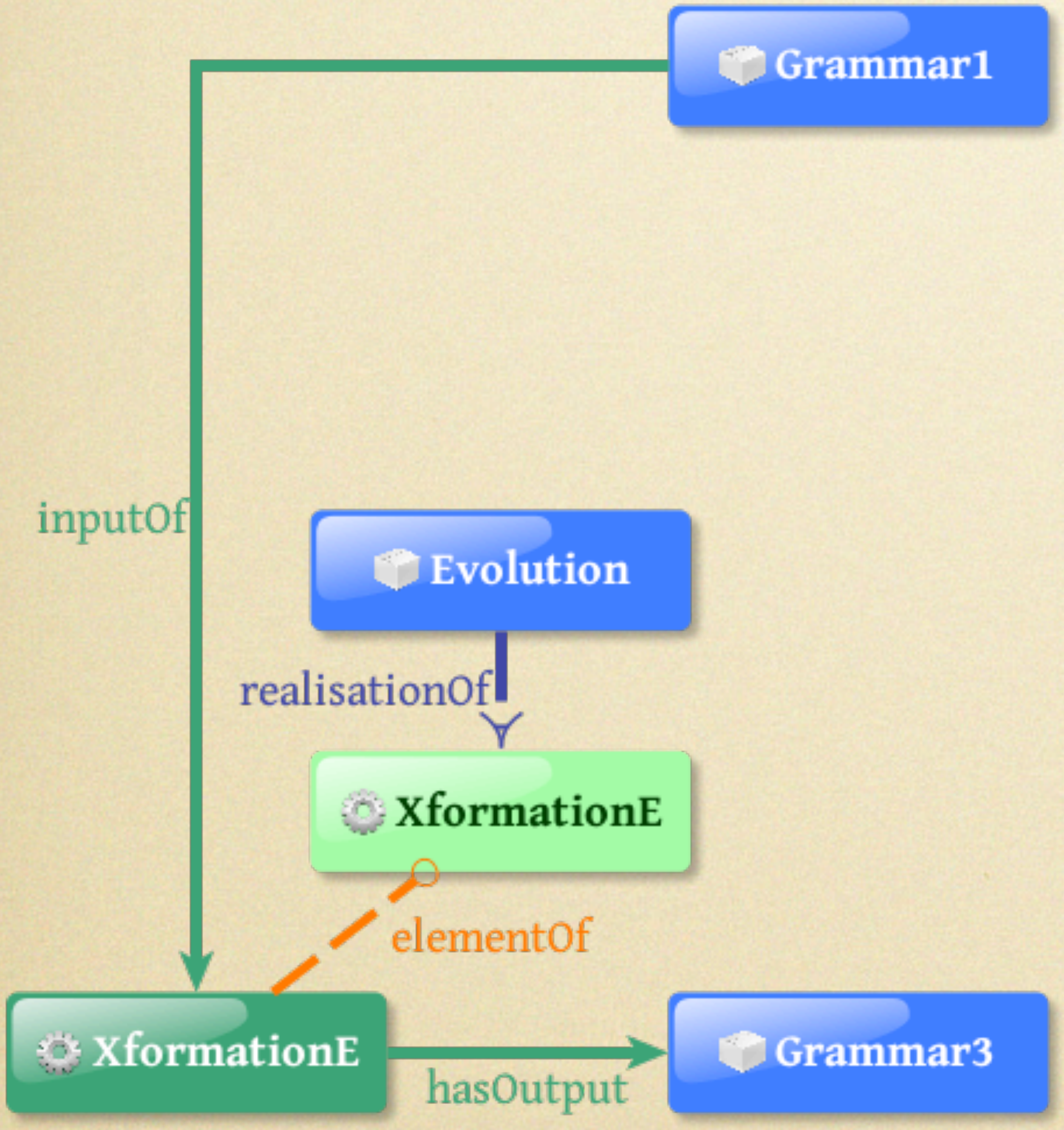
Grammar transformation composition

Adaptation through tolerance



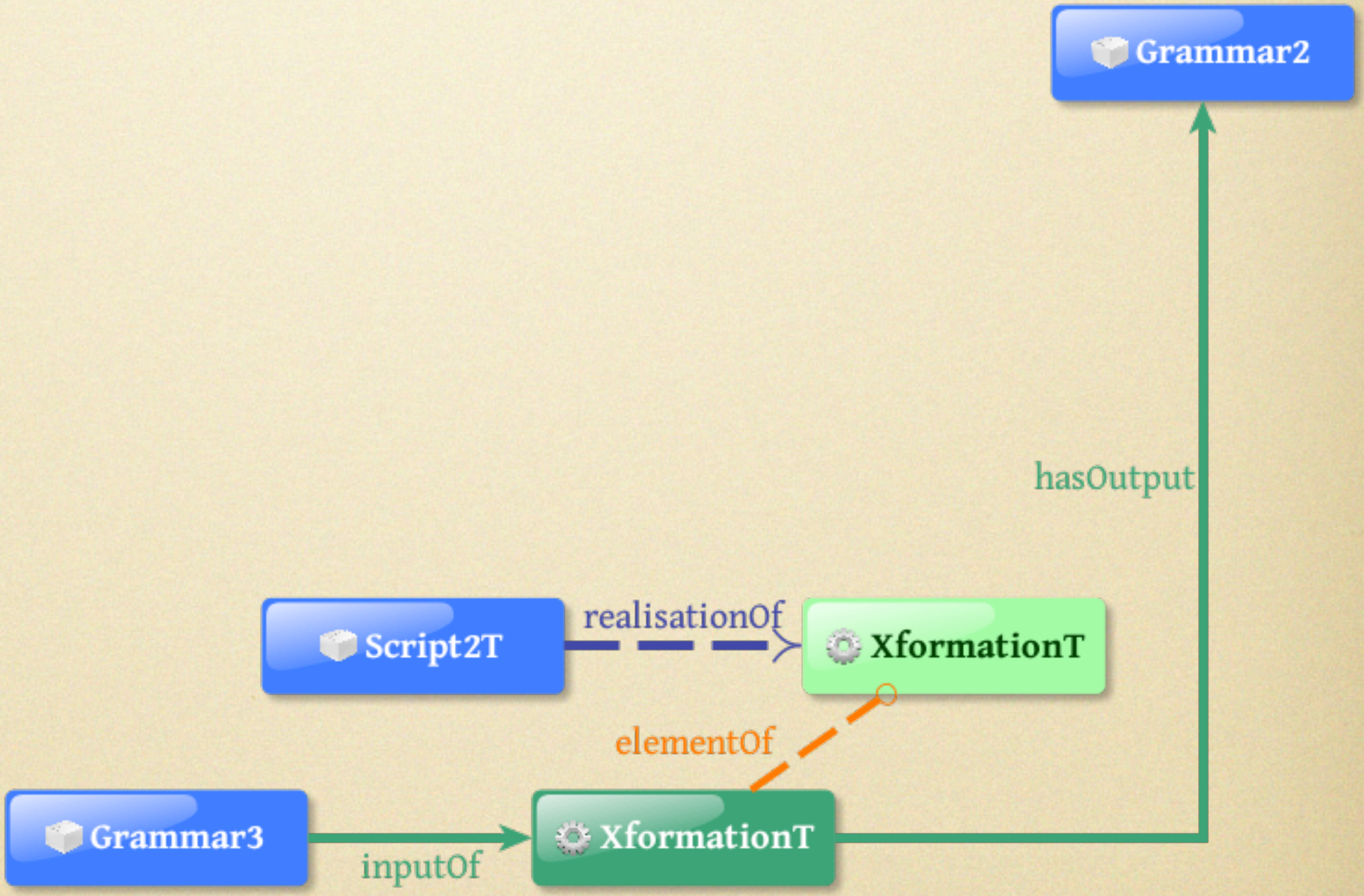
V. Zaytsev, Negotiated Grammar Transformation. Draft for XM 2012.

Adaptation through tolerance



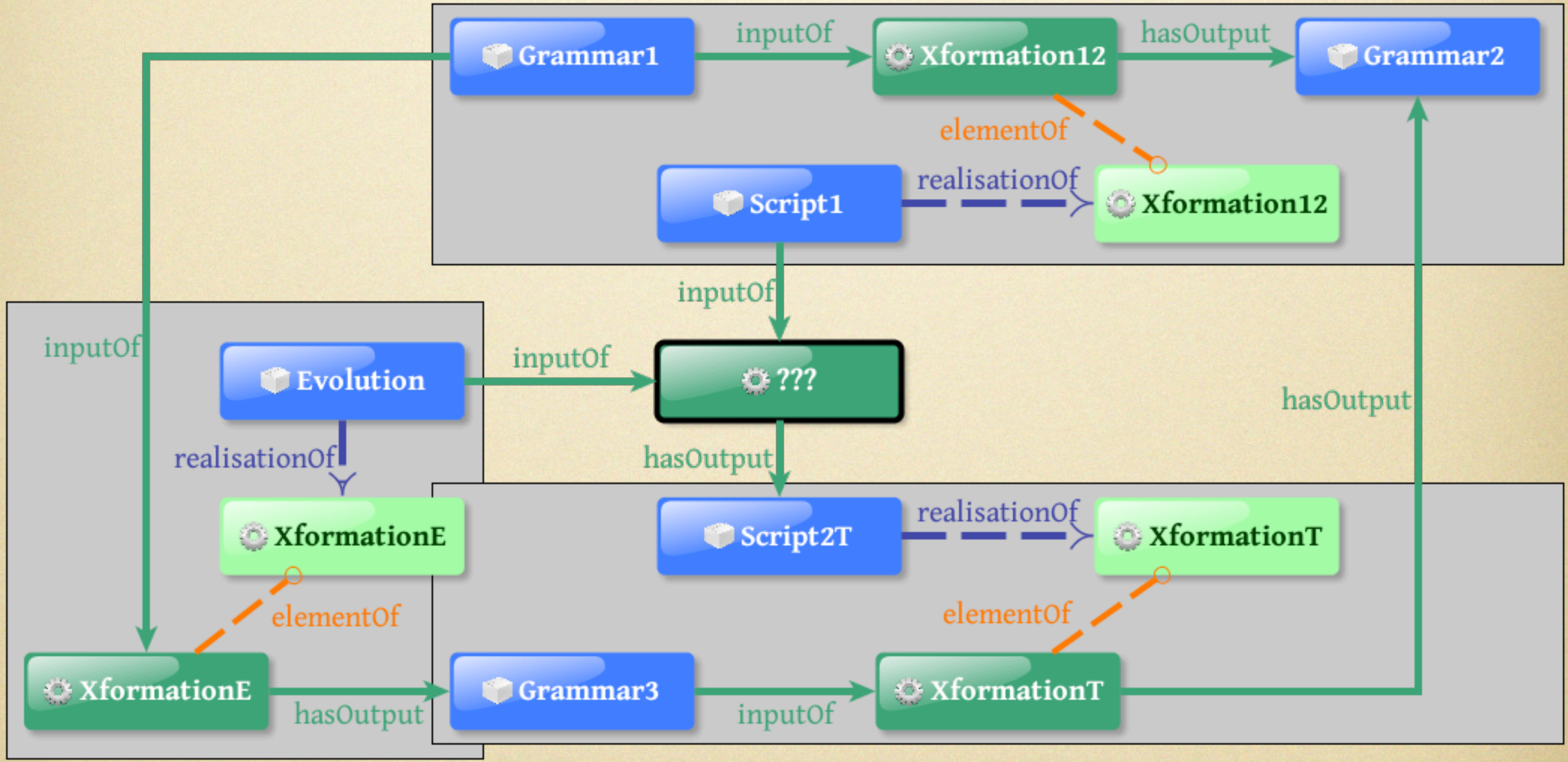
V. Zaytsev, _Negotiated Grammar Transformation_. Draft for XM 2012.

Adaptation through tolerance



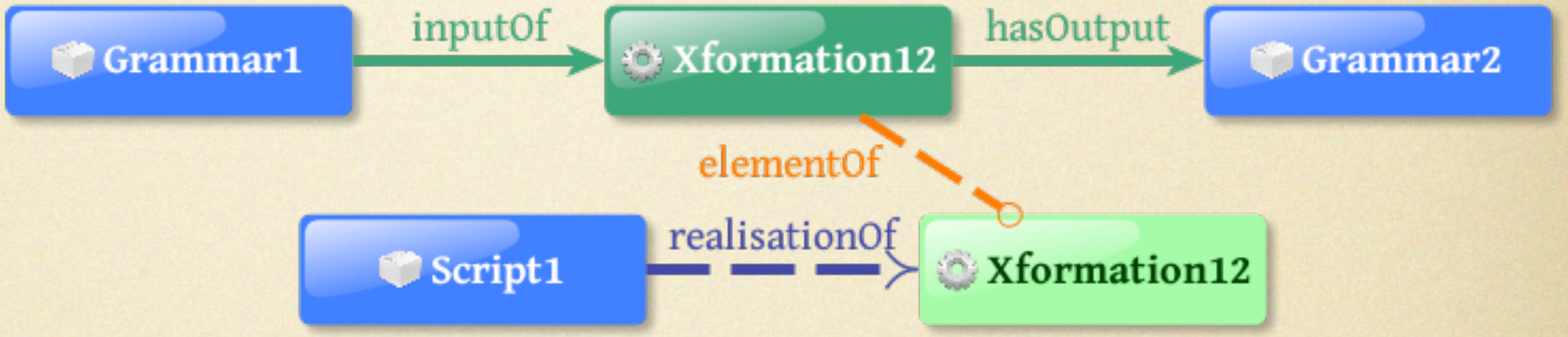
V. Zaytsev, *Negotiated Grammar Transformation*. Draft for XM 2012.

Adaptation through tolerance



V. Zaytsev, <u>*Negotiated Grammar Transformation*</u>. Draft for XM 2012.

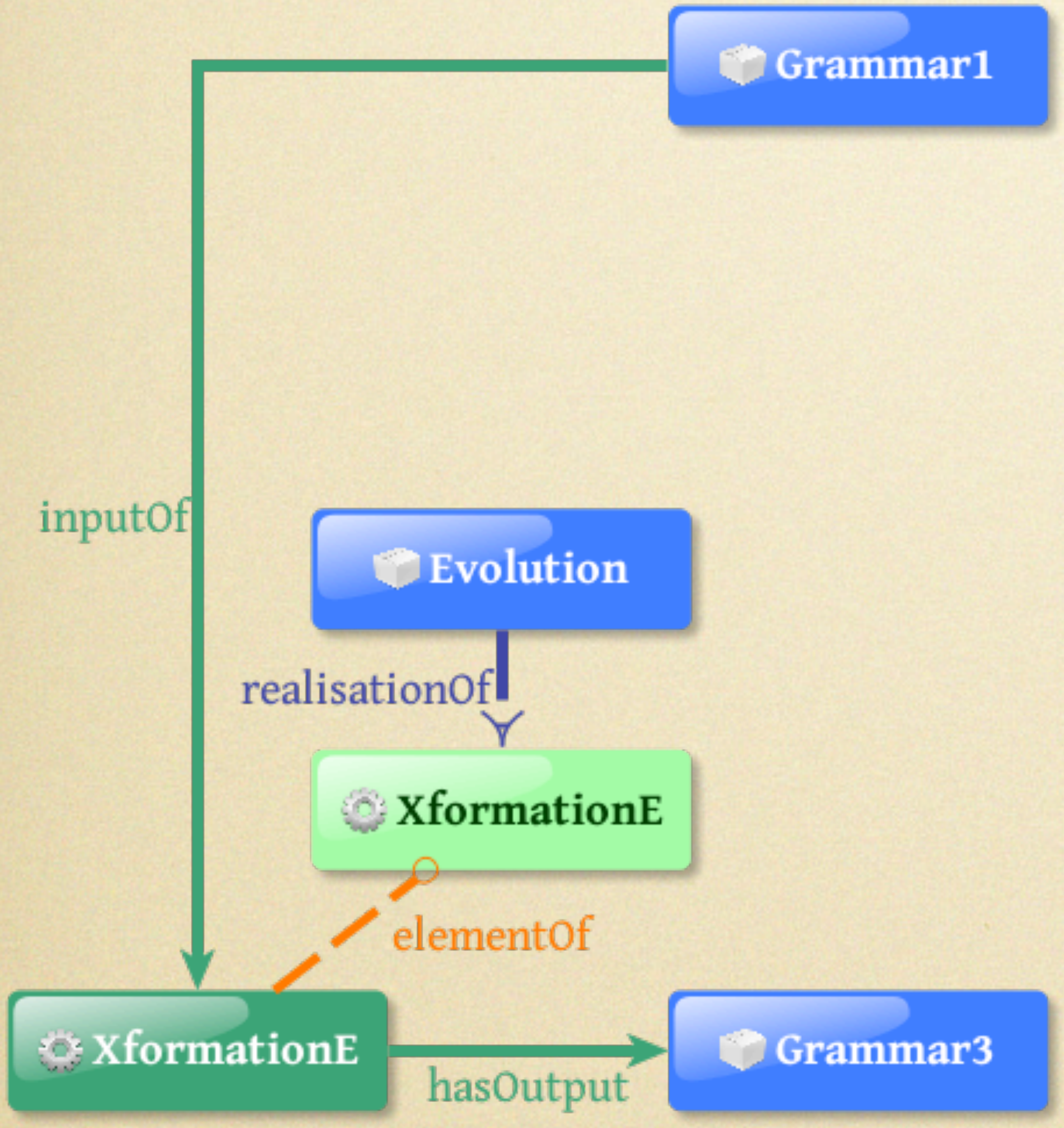
Adaptation through adjustment



V. Zaytsev, Negotiated Grammar Transformation. Draft for XM 2012.

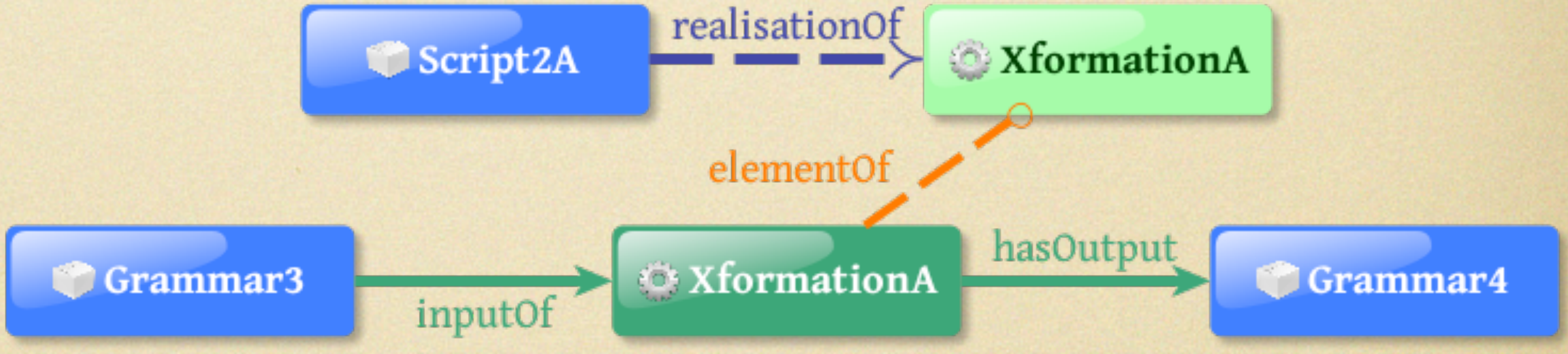
CWI

Adaptation through adjustment



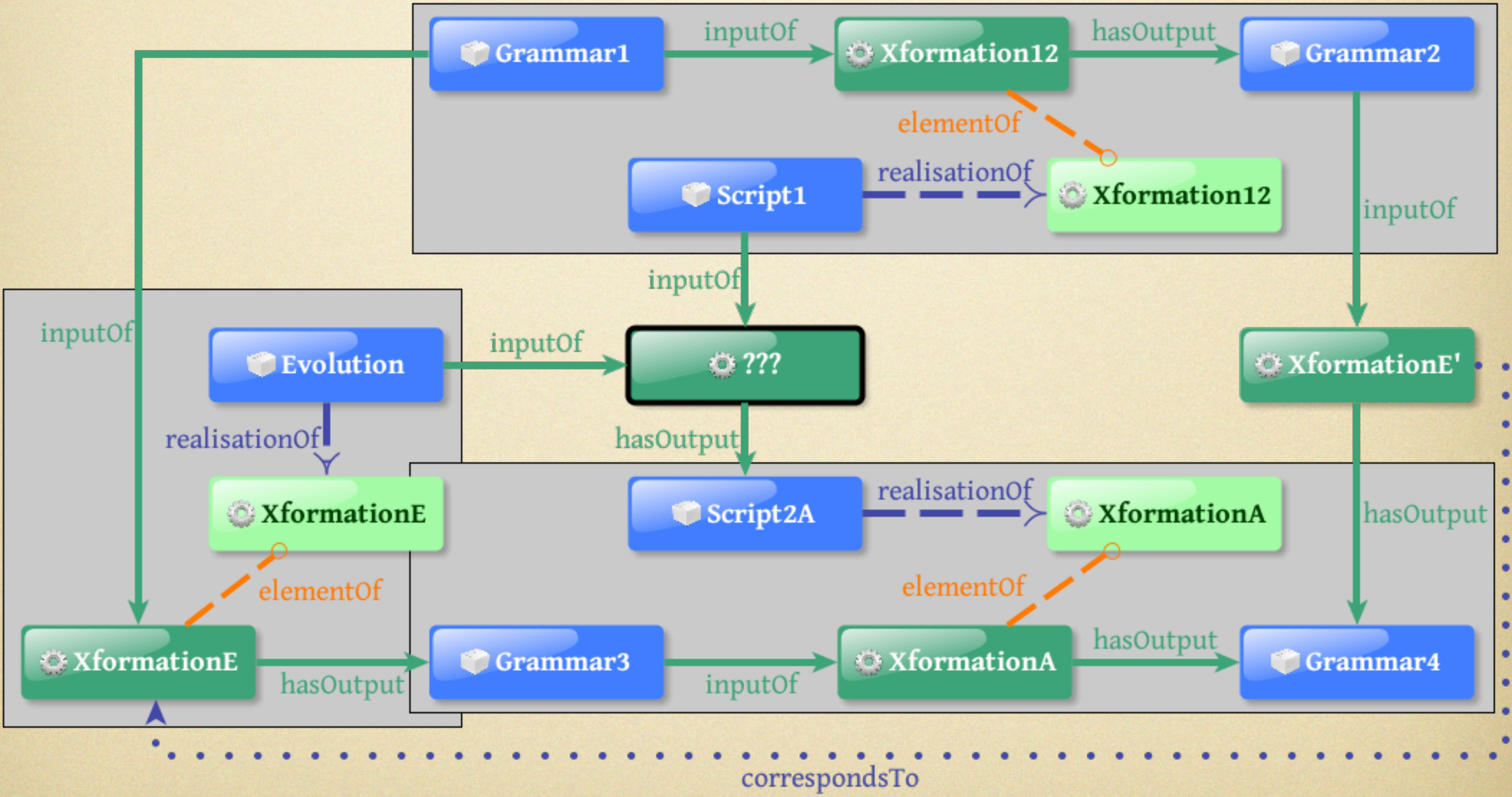
V. Zaytsev, *Negotiated Grammar Transformation*. Draft for XM 2012.

Adaptation through adjustment



V. Zaytsev, *Negotiated Grammar Transformation*. Draft for XM 2012.

Adaptation through adjustment



V. Zaytsev, *Negotiated Grammar Transformation*. Draft for XM 2012.

Grammar transformations

- Suite of well-defined well-studied operators
- Partial evaluation of transformation operators
- Classic grammar transformation:
  - inapplicable? error! halt!
  - vacuous? error! halt!
  - transform! next!

Negotiated transformations

- Negotiated grammar transformation:
  - applicable & non-vacuous? transform! next!
  - vacuous? suggest to do nothing!
  - not applicable? suggest better arguments!
  - keep negotiating until applicability or surrender

V. Zaytsev, *Negotiated Grammar Transformation*. Draft for XM 2012.

Negotiated transformations

- Variability limits as a part of transformation command
- Interactive transformation (ask the user)
- Display a warning and proceed with minimal adjustment
- Proceed with one, save other options for fallback
- Halt and recomment
- …

V. Zaytsev, *Negotiated Grammar Transformation*. Draft for XM 2012.

To summarise

- Grammars define structure
- Grammarware works on grammars & languages
- Too much stuff in the grammar
- Decomposition
- Composition
- Adjacent topics?

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

Questions?

vadim@grammarware.net