Language Evolution, Metasyntactically

First International Workshop on Bidirectional Transformations (BX 2012)

Vadim Zaytsev, SWAT, CWI

Introduction

- Every language document employs its own notation
- We focus on **metalanguage evolution**
  - the language itself does not evolve
  - the notation in which it is written, does
- We limit ourselves to grammarware technical space
- Working prototypes are a part of SLPS

Metalanguage evolution

- Notation correction
  - Misused notation
  - Overused notation
- Notation evolution
  - Notations are languages, they evolve
- Mapping between notations
  - Many notations are equivalent
  - Most are (almost EBNF) + (little extra)

Metasyntactic evolution megamodel

Megamodel

Syntactic notation specification

EBNF Dialect Definition

```
program::=
            function+;
function::=
            name argument* "=" expr?;
```

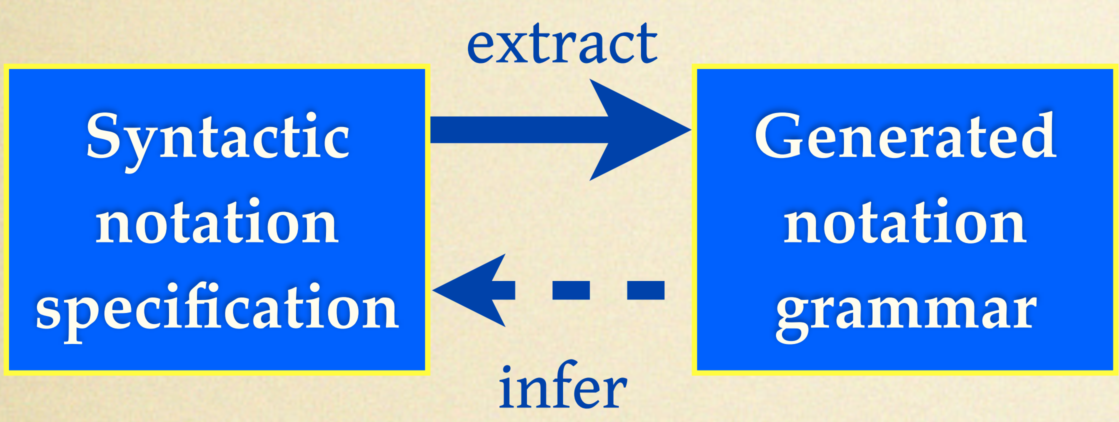
- List of indicators
- Together form a notation specification

Zaytsev, <u>*What Have We Done About the Unnecessary Diversity of Notation for Syntactic Definitions*</u>, SAC/PL 2012.

EDD example

defining metasymbol	:	definition separator metasymbol	|
terminator metasymbol	;	postfix optionality metasymbol	?
postfix star metasymbol	*	postfix plus metasymbol	+
start terminal metasymbol	"	end terminal metasymbol	"

Megamodel


extract
Syntactic notation specification
Generated notation grammar
infer

Generated notation grammar (in Rascal)

```
module LLL
import util::IDE; // needed only for advanced IDE support (see last two lines)
start syntax LLLGrammar = LLLLayoutList LLLProduction* LLLLayoutList;
syntax LLLProduction = LLLNonterminal ":" {LLLDefinition "|"}+ ";";
syntax LLLDefinition = LLLSymbol+;
syntax LLLSymbol
 = @category="Identifier" nonterminal: LLLNonterminal
  | @category="Constant" terminal: LLLTerminal
  | group: "(" LLLDefinition ")"
  | optional: LLLSymbol "?"
  | star: LLLSymbol "*"
  | plus: LLLSymbol "+"
  | sepliststar: "{" LLLSymbol LLLSymbol "}  *"
  | seplistplus: "{" LLLSymbol LLLSymbol "} +";
lexical LLLTerminal = "\"" LLLTerminalSymbol* "\"";
lexical LLLTerminalSymbol = ![\"];
lexical LLLNonterminal = [A-Za-z_01-9\-/]+ !>> [A-Za-z_01-9\-/];
layout LLLLayoutList = LLLLayout* !>> [\t-\n \r \ ] !>> "#";
lexical LLLLayout = [\t-\n \r \ ] | LLLComment ;
lexical LLLComment = @category="Comment" "#" ![\n]* [\n];
Tree getLLL(str s,loc z) = parse(#LLLGrammar,z);
public void registerLLL() = registerLanguage("LLL","lll",getLLL);
```

Grammar internal representation

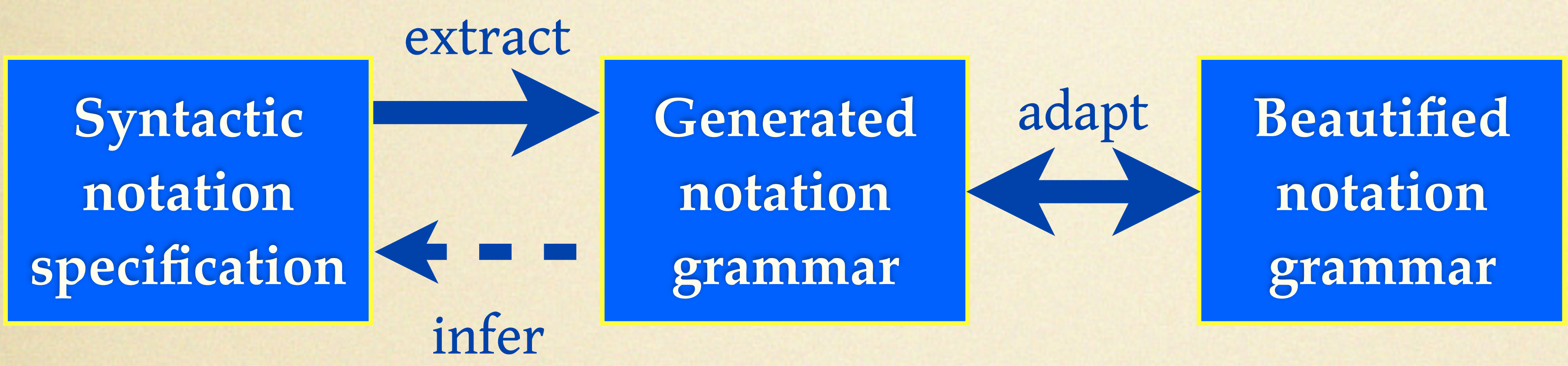
$\mathit{grammar(Rs,Ps)} \Leftarrow \mathit{mapoptlist(n,Rs),\ maplist(prod,Ps).}$	grammar = start symbols + productions
$\mathit{prod(p(L,N,X))} \Leftarrow \mathit{mapopt(label,L),\ atom(N),\ expr(X).}$	production = label + lhs + rhs
$\mathit{label(l(X))} \Leftarrow \mathit{atom(X).}$	production labels
$\mathit{expr(true).}$	ε
$\mathit{expr(fail).}$	empty language
$\mathit{expr(a).}$	universal type
$\mathit{expr(t(T))} \Leftarrow \mathit{atom(T).}$	terminal symbols
$\mathit{expr(n(N))} \Leftarrow \mathit{atom(N).}$	nonterminal symbols
$\mathit{expr(','(Xs))} \Leftarrow \mathit{maplist(expr,Xs).}$	sequential composition
$\mathit{expr(';'(Xs))} \Leftarrow \mathit{maplist(expr,Xs).}$	choice
$\mathit{expr('?'(X))} \Leftarrow \mathit{expr(X).}$	optionality
$\mathit{expr('*'(X))} \Leftarrow \mathit{expr(X).}$	Kleene star
$\mathit{expr('+'(X))} \Leftarrow \mathit{expr(X).}$	transitive closure
$\mathit{expr(slp(X,Y))} \Leftarrow \mathit{expr(X),\ expr(Y).}$	Y-separated list with 1 or more elements
$\mathit{expr(sls(X,Y))} \Leftarrow \mathit{expr(X),\ expr(Y).}$	Y-separated list with 0 or more elements
$\mathit{expr(s(S,X))} \Leftarrow \mathit{atom(S),\ expr(X).}$	selectable expressions

Lämmel, Zaytsev, An Introduction to Grammar Convergence, IFM 2009.

Generated notation grammar (in BGF)

```
LLL1Grammar:
    LLL1Production*
LLL1Production:
    LLL1Nonterminal ":" {LLL1Definition "|"}+ ";"
LLL1Definition:
    LLL1Symbol+
[nonterminal] LLL1Symbol:
    LLL1Nonterminal
[terminal] LLL1Symbol:
    LLL1Terminal
[optional] LLL1Symbol:
    LLL1Symbol "?"
[star] LLL1Symbol:
    LLL1Symbol "*"
[plus] LLL1Symbol:
    LLL1Symbol "+"
```

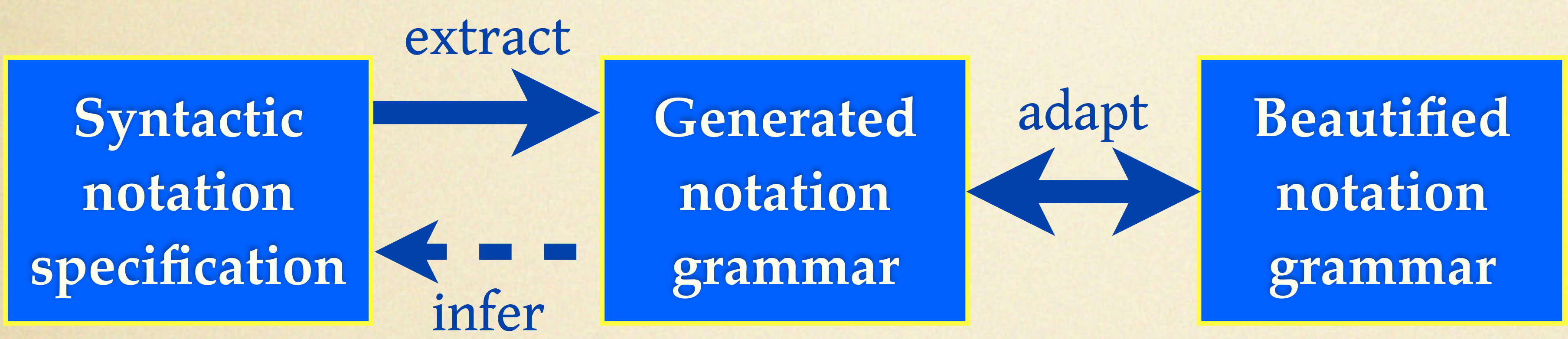
Megamodel


extract
Syntactic notation specification
Generated notation grammar
adapt
Beautified notation grammar
infer

Beautified notation grammar (in BGF)

```
grammar:
    rule+
rule:
    sort ":" alts ";"
alts:
    alt alts-tail*
alts-tail:
    "|" alt
alt:
    term*
term:
    basis repetition?
basis:
    literal
    sort
repetition:
    "*"
    "+"
    "?"
```

Megamodel



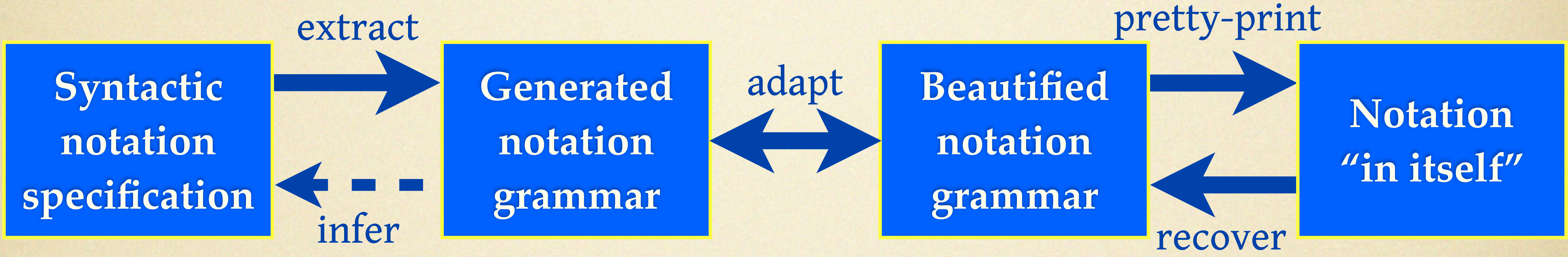
extract
Syntactic notation specification
Generated notation grammar
adapt
Beautified notation grammar
infer

Bidirectional grammar adaptation

- XBGF ⇒ ΞBGF:
  - renameN, factor, etc: flip arguments
  - addV/removeV, narrow/widen: form pairs
  - extract/inline, unlabel/designate: asymmetry
  - distribute: removed from the language
  - unite, equate: tricky, superposition of others
- BX is a stable way to represent grammar relationship

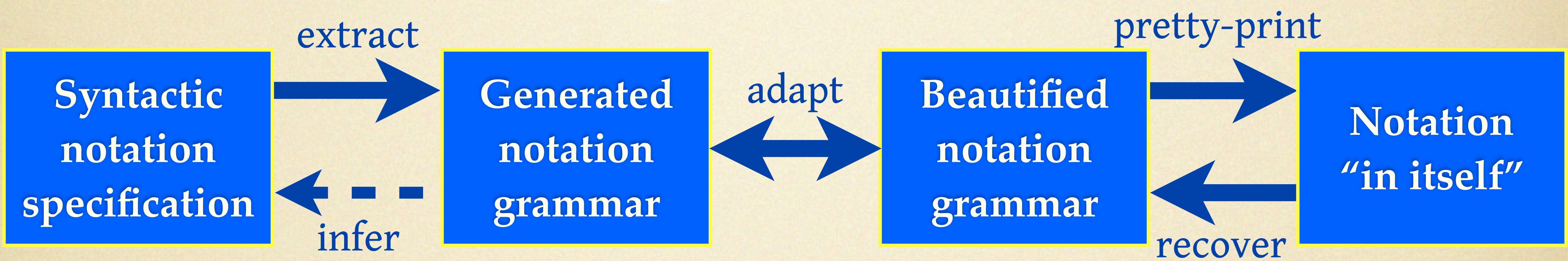
Lämmel, Zaytsev, *Recovering Grammar Relationships for the Java Language Specification*, SQJ 19:2, 2011.

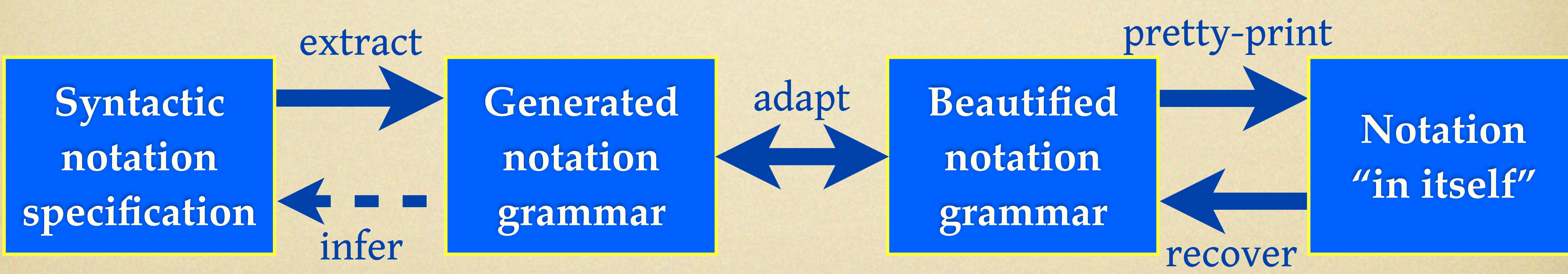
Megamodel



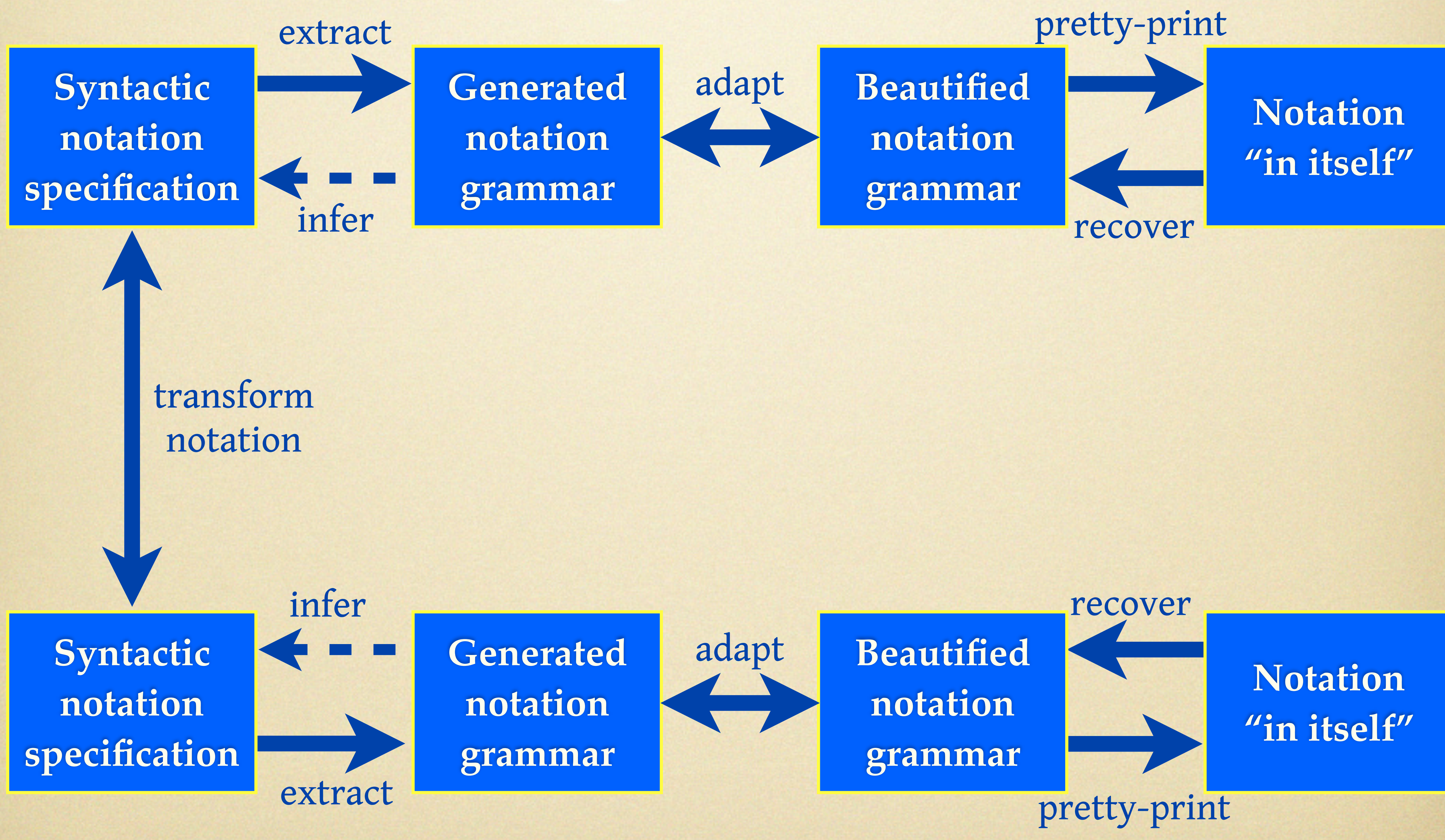
extract
Syntactic notation specification
Generated notation grammar
infer
adapt
Beautified notation grammar
pretty-print
Notation "in itself"
recover

Megamodel


Syntactic notation specification
extract
infer
Generated notation grammar
adapt
Beautified notation grammar
pretty-print
recover
Notation "in itself"


Syntactic notation specification
extract
infer
Generated notation grammar
adapt
Beautified notation grammar
pretty-print
recover
Notation "in itself"

Megamodel

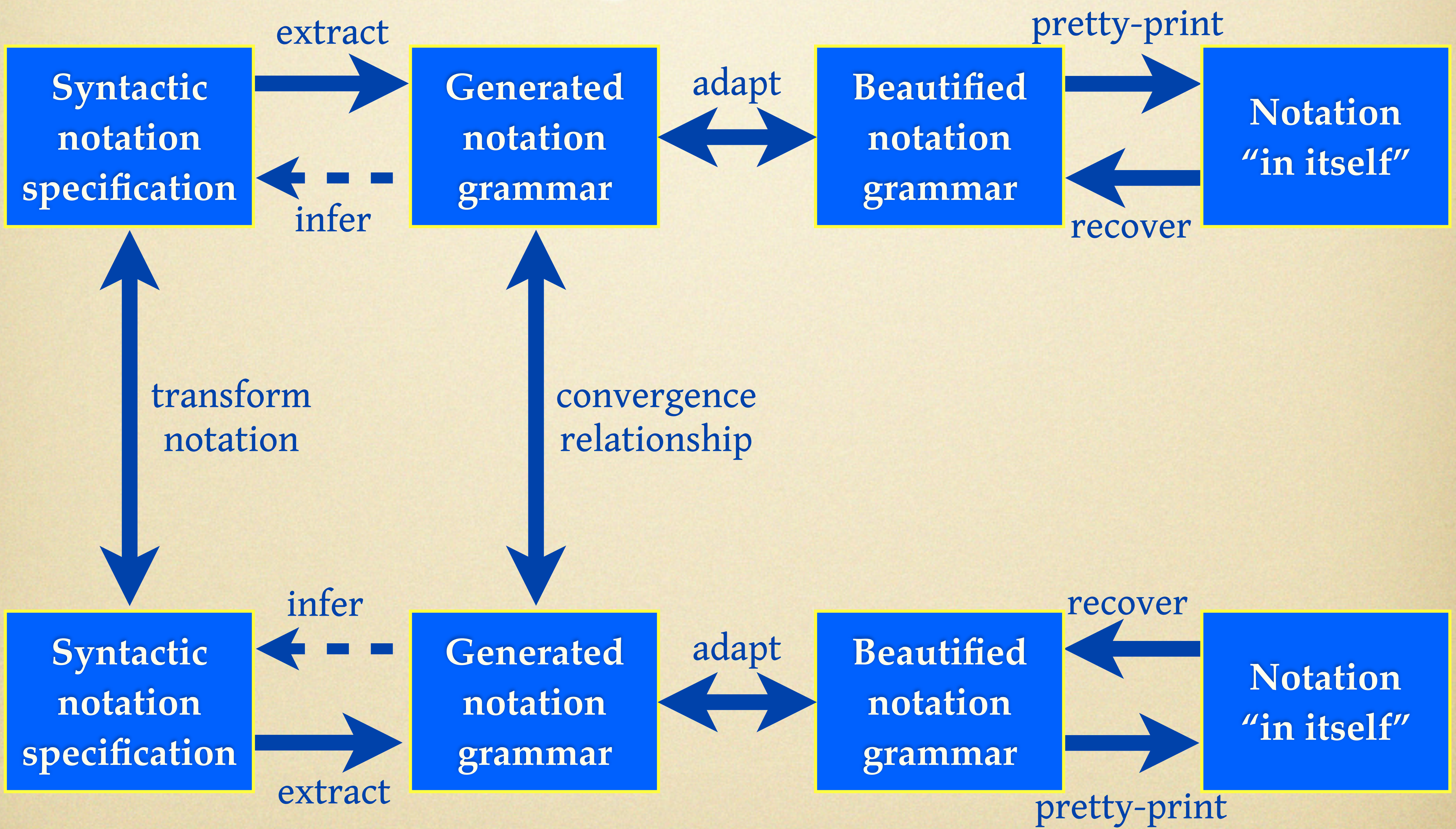


extract
pretty-print
Syntactic notation specification
Generated notation grammar
adapt
Beautified notation grammar
Notation "in itself"
infer
recover
transform notation
infer
recover
Syntactic notation specification
Generated notation grammar
adapt
Beautified notation grammar
Notation "in itself"
extract
pretty-print

Notation transformation

- EDD — notation, consists of metasymbols
- XEDD — transformation language
  - **rename-metasymbol**(*s, v1, v2*)
    - e.g., change defining metasymbol from ":" to "::="
  - **introduce-metasymbol**(*s, v*)
    - e.g., bring a terminator metasymbol to a notation
  - **eliminate-metasymbol**(*s, v*)

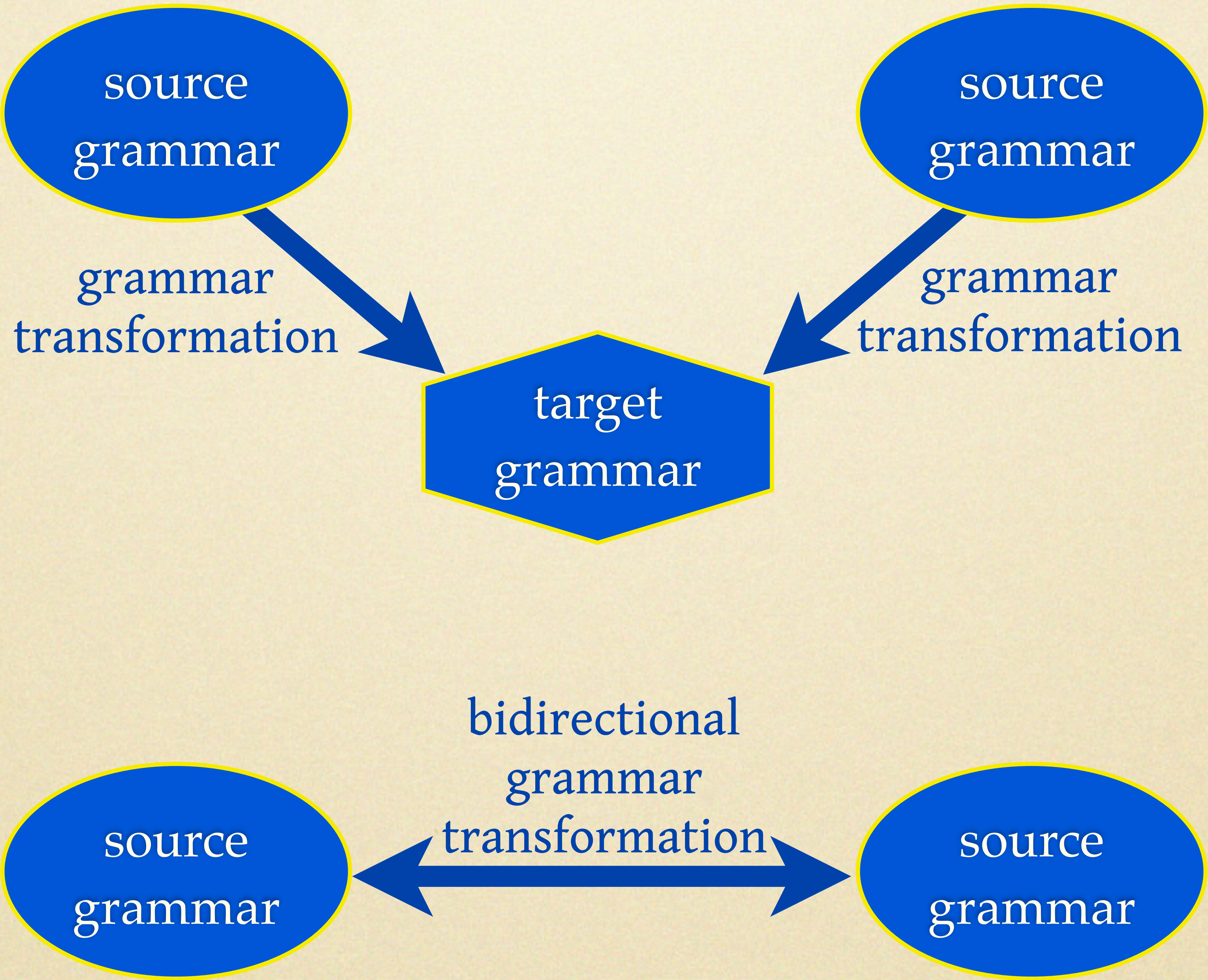
Megamodel


extract
Syntactic notation specification
Generated notation grammar
adapt
Beautified notation grammar
pretty-print
Notation “in itself”
infer
recover
transform notation
convergence relationship
infer
recover
Syntactic notation specification
Generated notation grammar
adapt
Beautified notation grammar
Notation “in itself”
extract
pretty-print

Grammar convergence

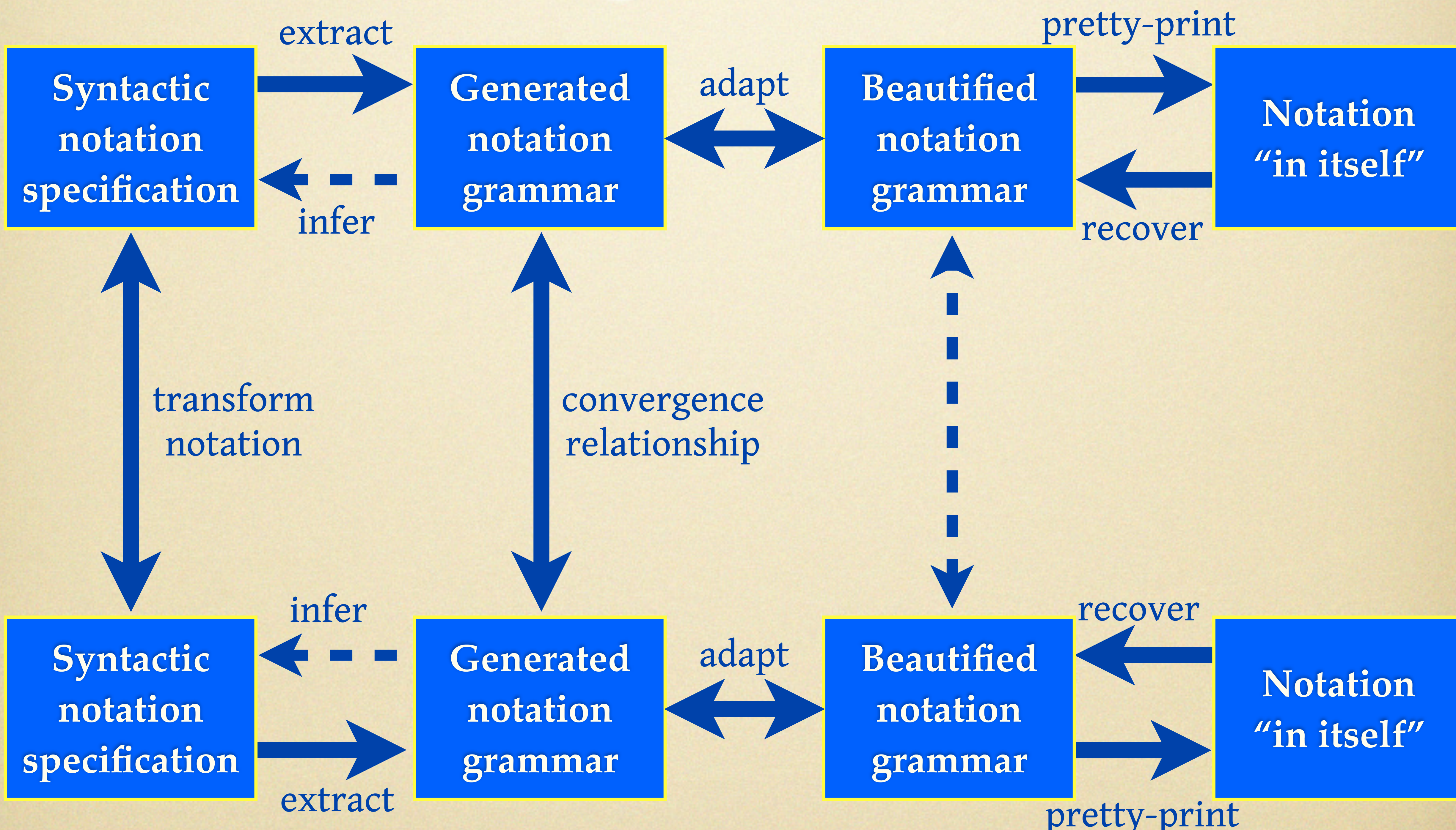
XBGF

ΞBGF



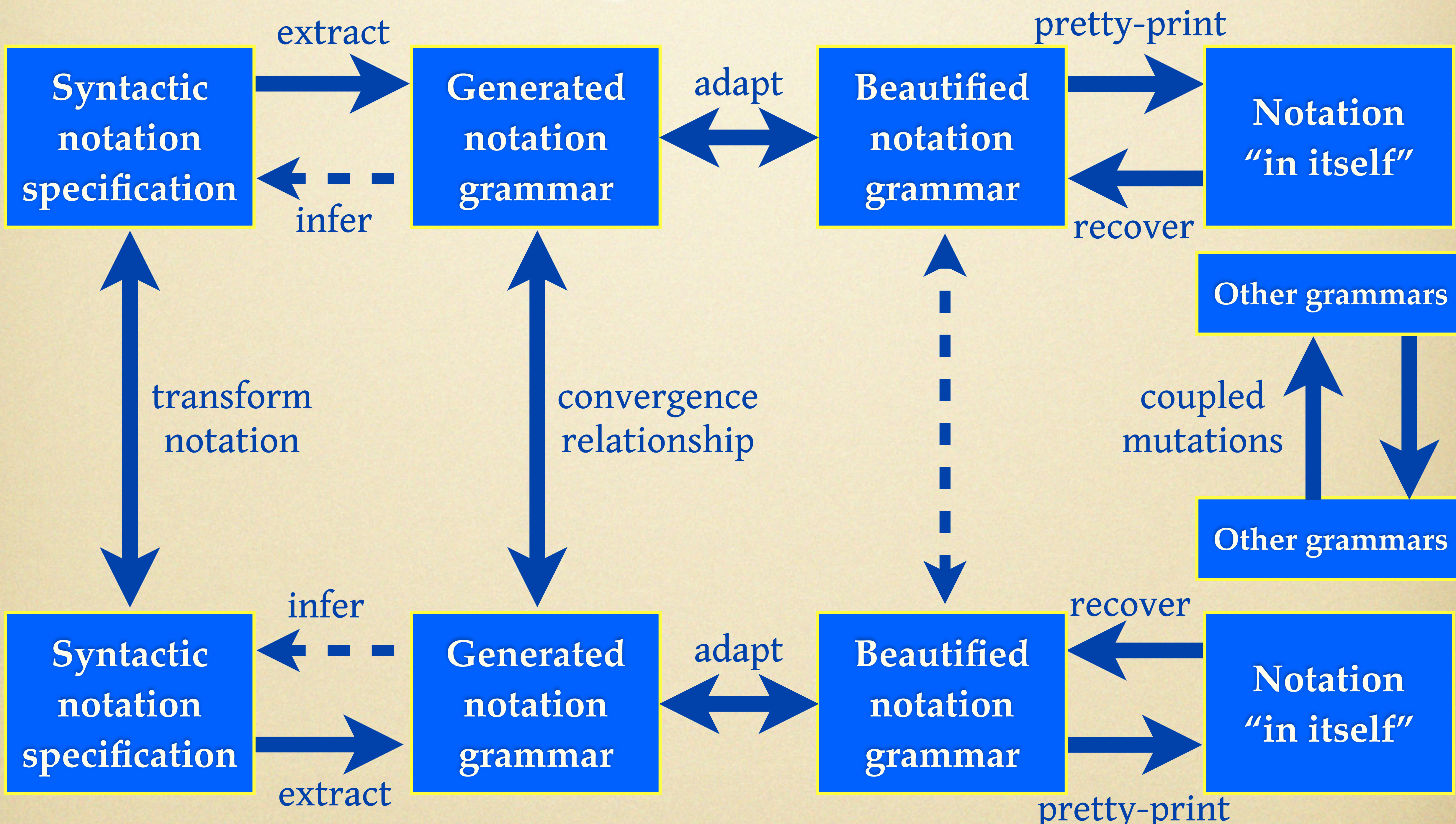
= relationship

Megamodel



extract
Syntactic notation specification
Generated notation grammar
adapt
Beautified notation grammar
pretty-print
Notation "in itself"
infer
recover
transform notation
convergence relationship
infer
recover
Syntactic notation specification
Generated notation grammar
adapt
Beautified notation grammar
Notation "in itself"
extract
pretty-print

Megamodel


Syntactic notation specification
extract
Generated notation grammar
adapt
Beautified notation grammar
pretty-print
Notation "in itself"
infer
recover
transform notation
convergence relationship
Other grammars
coupled mutations
Other grammars
infer
recover
Syntactic notation specification
Generated notation grammar
adapt
Beautified notation grammar
Notation "in itself"
extract
pretty-print

Grammar transformation vs. grammar mutation

- A **grammar transformation operator** τ can be formalised as a triplet: $\tau = \langle c\_pre, t, c\_post \rangle$.
- A **grammar transformation** then is τ_a_i (G), resulting in G'.
  - if a_i are of incorrect types and quantity than expected by t ⇒ τ is **incorrectly called**;
  - if the constraint c_pre does not hold on G ⇒ τ_a_i is **inapplicable** to G;
  - if the constraint c_post holds on G ⇒ τ_a_i is **vacuous** on G;
  - if the constraint c_pre holds on G and c_post does not hold on G' ⇒ t is **incorrectly implemented**;
  - if c_pre holds on G, c_post holds on G' ⇒ τ has been **applied correctly** with arguments a_i to G resulting in G'.

Grammar transformation vs. grammar mutation

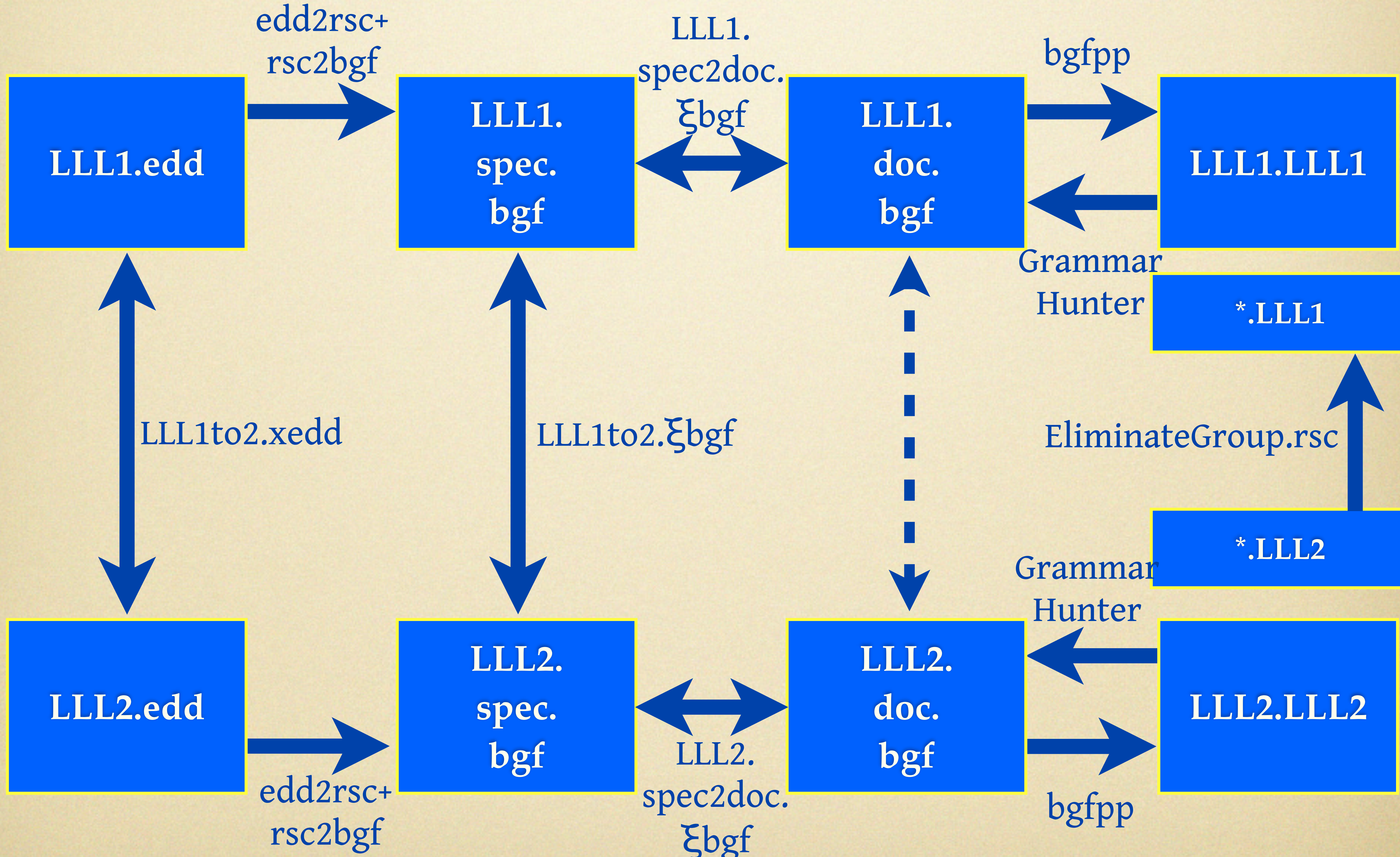
- A **grammar mutation** does not have a single precondition
- It has a set of preconditions that serve as **triggers**: $\mu = \langle \{c\_i\}, \{t\_i\}, c\_post \rangle$.
- The mutation **terminates** once no trigger c_i holds and the postcondition c_post is met.
- A **bidirectional** grammar mutation: $\mu\_bx = \langle c\_pre, \{c\_i\}, \{t\_i\}, c\_post \rangle$ will be an instantiation of a grammar mutation
- The family of spawned BMs does **not** define the original: i.e., $\forall \mu\ \exists G\ \exists G'\ \nexists \mu\_bx,\ G' = \mu(G) \wedge G' = \mu\_bx(G) \wedge G = \mu^{-1}(G')$.

Notation evolution summary

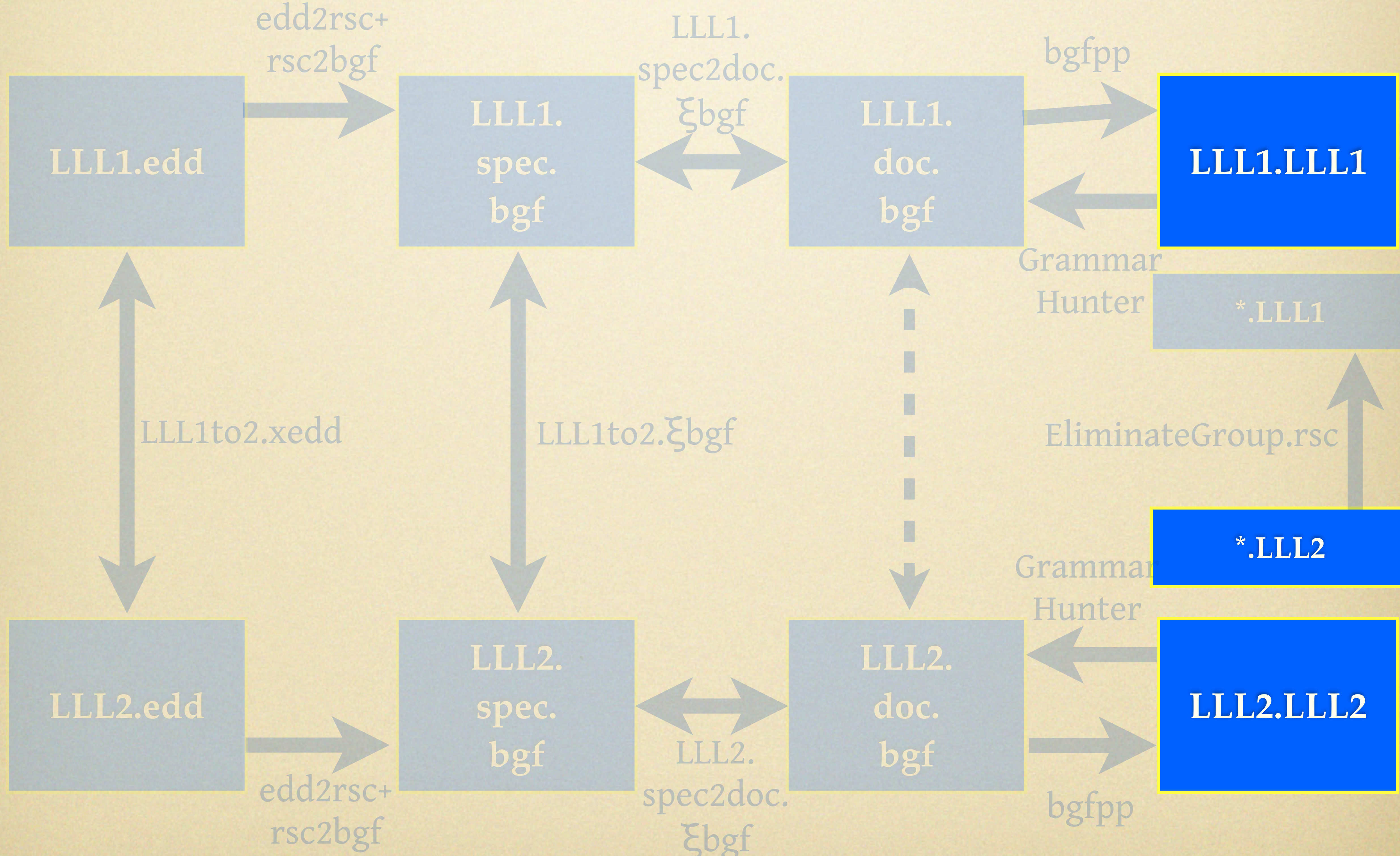
- A notation evolution step Δ consists of the following coupled components:
  - σ, a **bidirectional notation transformation** that changes the notation itself
  - δ, a **convergence relationship** that can transform the notation grammars
  - γ, a **bidirectional grammar adaptation** that prepares a beautified readable version of N'
  - μ, an unidirectional **coupled grammar mutation** that migrates the grammarbase according to notation changes
  - possibly μ', an unidirectional **coupled grammar mutation** that migrates the grammarbase according to the inverse of the intended notation changes

Evaluation

Megamodel: case study



Megamodel: previously



LLL1 in itself

```
grammar     : rule+;
rule        : sort ":" alts ";";
alts        : alt alts-tail*;
alts-tail   : "|" alt;
alt         : term*;
term        : basis repetition?;
basis       : literal | sort;
repetition : "*" | "+" | "?";
```

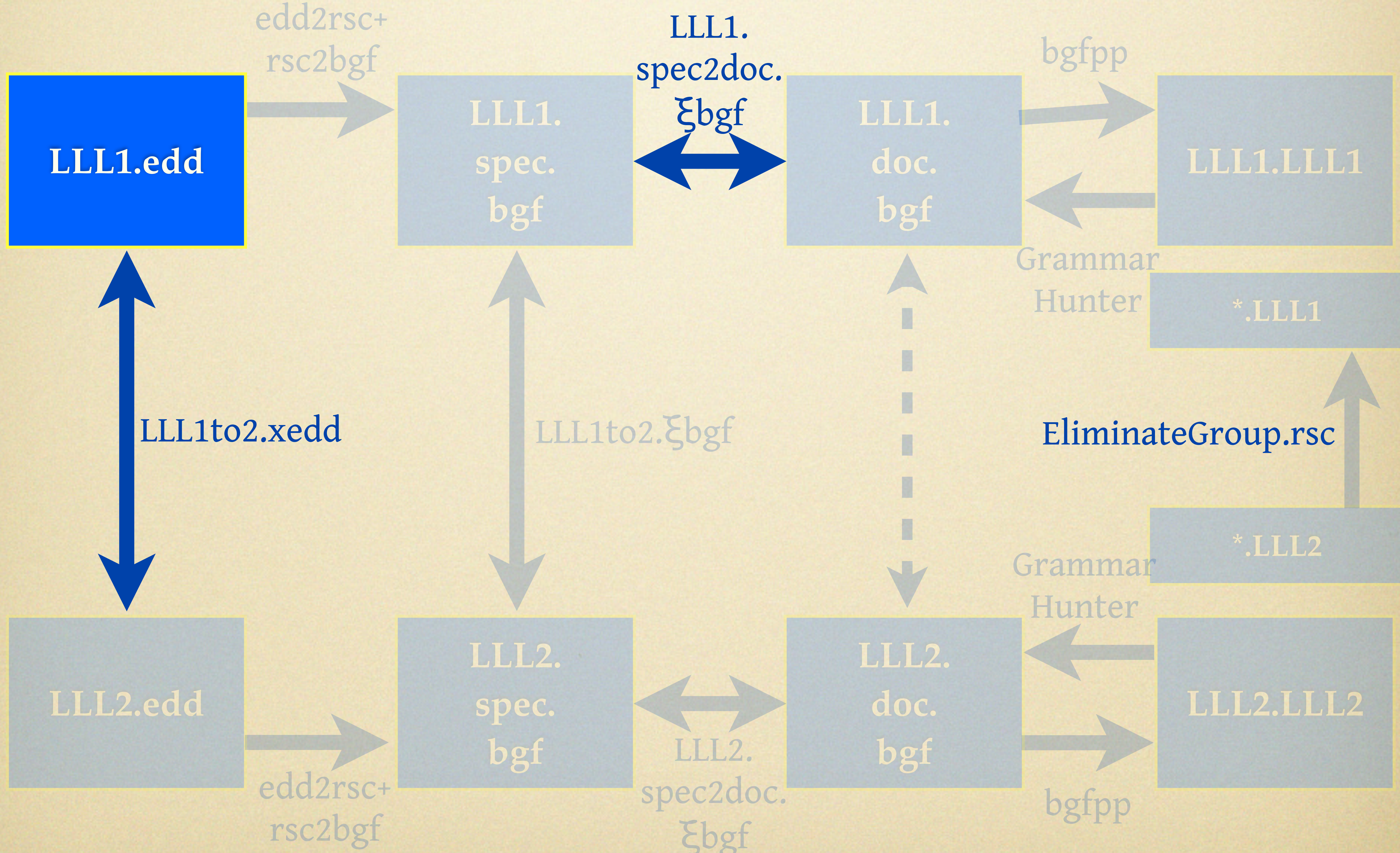
Kort, Lämmel, Verhoef, *The Grammar Deployment Kit: System Demonstration*, LDTA 2002.

LLL2 in itself

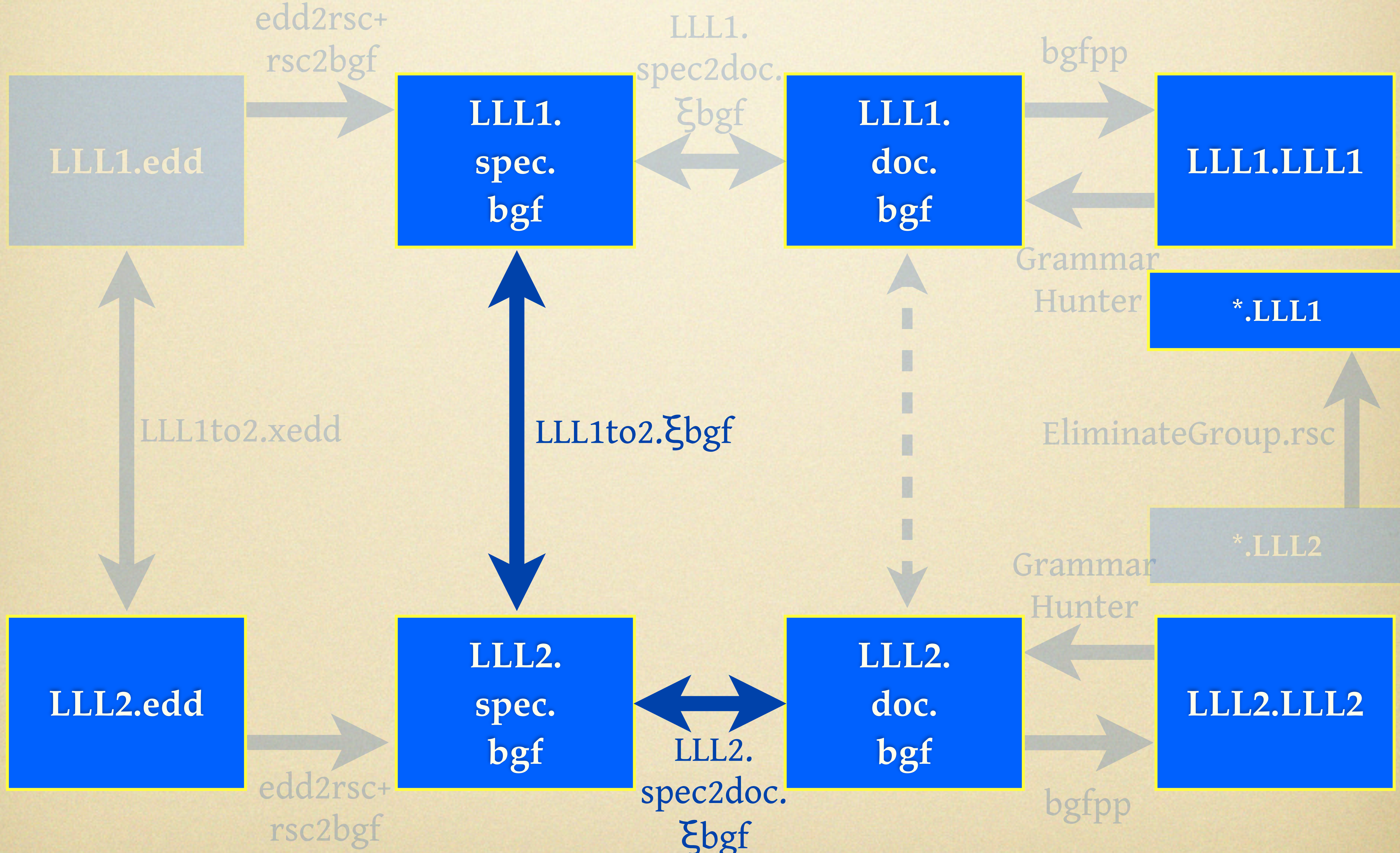
```
specification : rule+;
rule          : ident ":" disjunction ";";
disjunction   : {conjunction "|"}+;
conjunction   : term*;
term          : basis repetition?;
basis         : ident | literal
              | alternation | group;
repetition    :"+"|"*"|"?";
alternation   : "{"basis basis "}" repetition;
group         : "(" disjunction ")" ;
```

Kort, _Grammar Deployment Kit Reference Manual_, UvA/SourceForge 2003.

Megamodel: manually



Megamodel: automated



Applying coupled mutation **eliminate-metasymbol(group)** to Grammar Zoo

ada-kellogg	108	csharp-iso-23270-2003	0	java-1-jls-read	0
ada-kempe	89	csharp-iso-23270-2006	0	java-2-jls-impl	36
ada-laemmel-verhoef	79	csharp-msft-ls-1.0	0	java-2-jls-read	0
ada-lncs-2219	89	csharp-msft-ls-1.2	0	java-5-habelitz	65
ada-lncs-4348	109	csharp-msft-ls-3.0	0	java-5-jls-impl	60
c-iso-9899-1999	0	csharp-msft-ls-4.0	0	java-5-jls-read	1
c-iso-9899-tc2	0	csharp-zaytsev	23	java-5-parr	95
c-iso-9899-tc3	0	dart-google	58	java-5-stahl	92
cpp-iso-14882-1998	0	dart-spec-0.01	56	java-5-studman	91
cpp-iso-n2723	0	dart-spec-0.05	62	mediawiki-bnf	32
csharp-ecma-334-1	0	eiffel-bezault	45	mediawiki-ebnf	30
csharp-ecma-334-2	0	eiffel-iso-25436-2006	345	modula-sdf	50
csharp-ecma-334-3	0	fortran-derricks	101	modula-src-052	65
csharp-ecma-334-4	0	java-1-jls-impl	0	w3c-xpath1	3

Conclusion

Conclusion

- Extended XBGF to bidirectionality, resulting in ΞBGF.
- Proposed EDD and XEDD for notation & its evolution.
- Presented a case study of LLL evolution (GDK).
- Generalised transformers and generators to transformations and mutations; also formalised them.
- Implemented an XEDD processor for evolution, coevolution, change propagation and mutation.

Discussion

grammarware.net

slps.sf.net