Press PgDn to start.

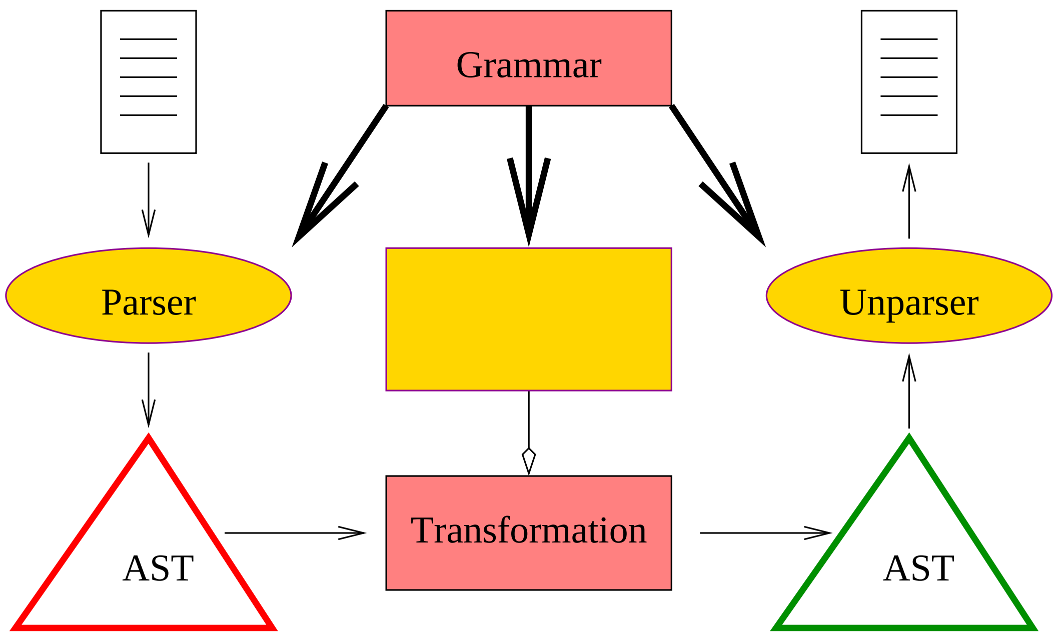
TOWARDS AN ENGINEERING DISCIPLINE FOR GRAMMARWARE

Ralf Lämmel, VU und CWI, Amsterdam

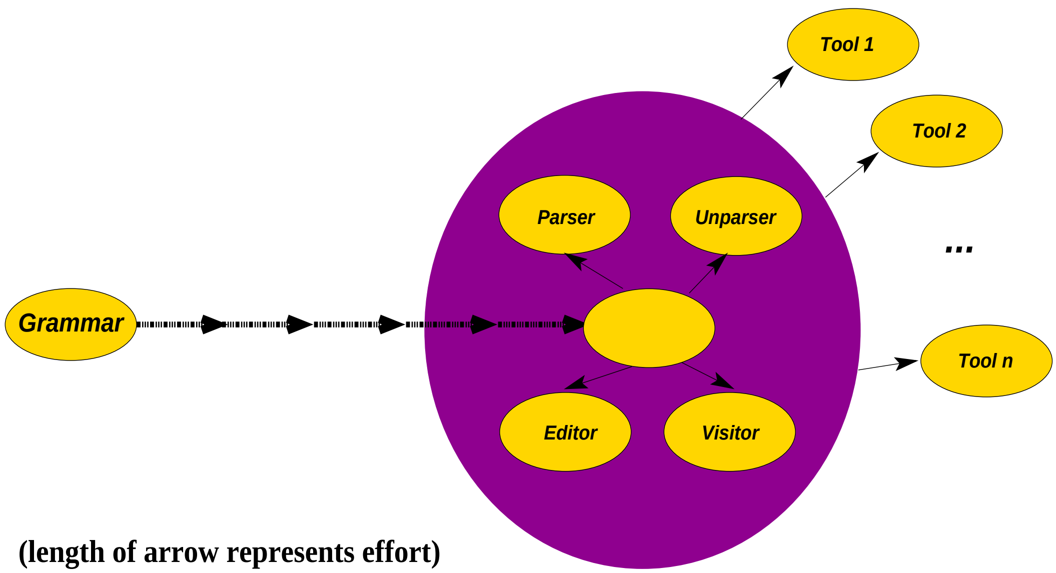
Grammarware — definition

The term ‘grammarware’ comprises grammars and grammar-driven software, i.e., software artifacts that directly involve grammar knowledge.

Software re-/reverse engineering needs engineering of grammarware — think of Transformation environments

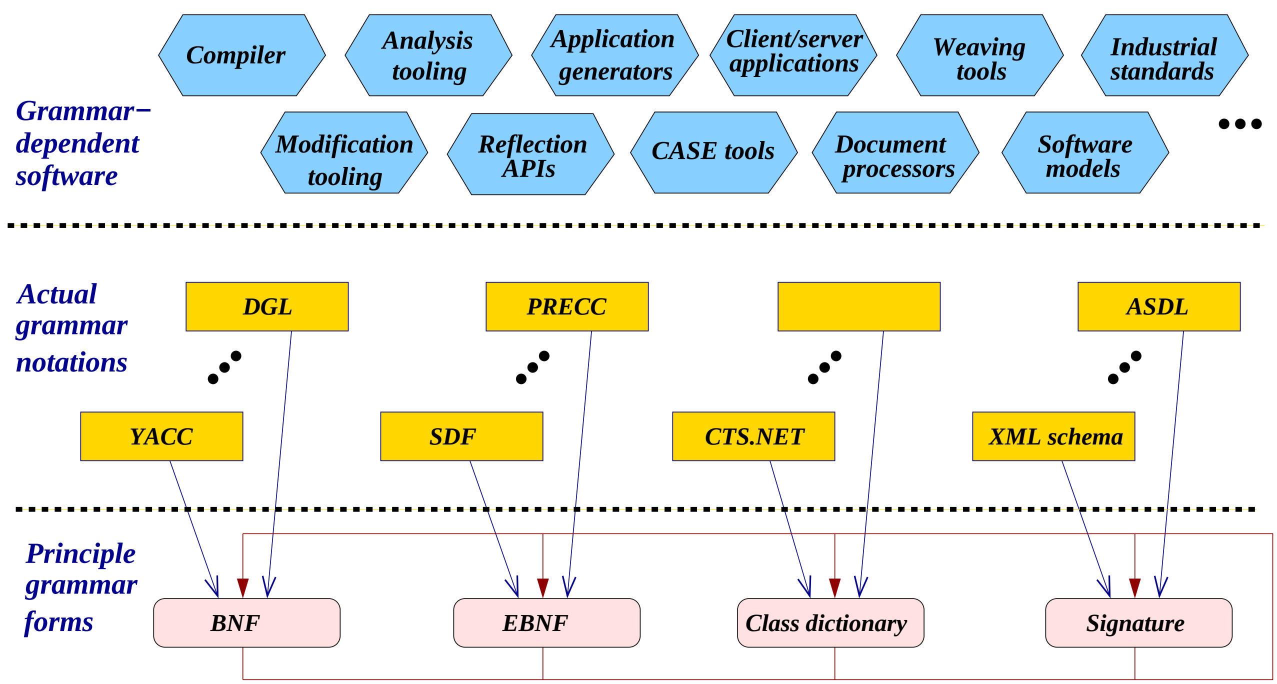


SOFTWARE RE-/REVERSE ENGINEERING NEEDS GRAMMAR ENGINEERING — THE EFFORT DISTRIBUTION



(length of arrow represents effort)

So much grammarware, so little attention



What is engineering of gw? Let's look at scenarios.

- As a developer of database applications, you want to make a transition to a *new screen definition language*.
- As a developer of Commercial Off-The-Shelf software, you want to *import user profiles* used by competing products; think of web browsers.
- As an object-oriented application developer, you want to systematically *migrate to a new release of some API*.
- As a developer of an inhouse domain-specific language (DSL), you want to provide a tool to *convert existing DSL programs* to the evolved DSL.
- As an online service provider, you want to support new import/export formats (maybe using XML schemas) or new protocols for system use.

Grammar forms and notations

- *Definitions of interchange and storage formats*
- *Interface descriptions*
- *Specifications of interaction protocols*
- *Concrete and abstract syntax definitions*
- *Definitions of intermediate and internal representations*

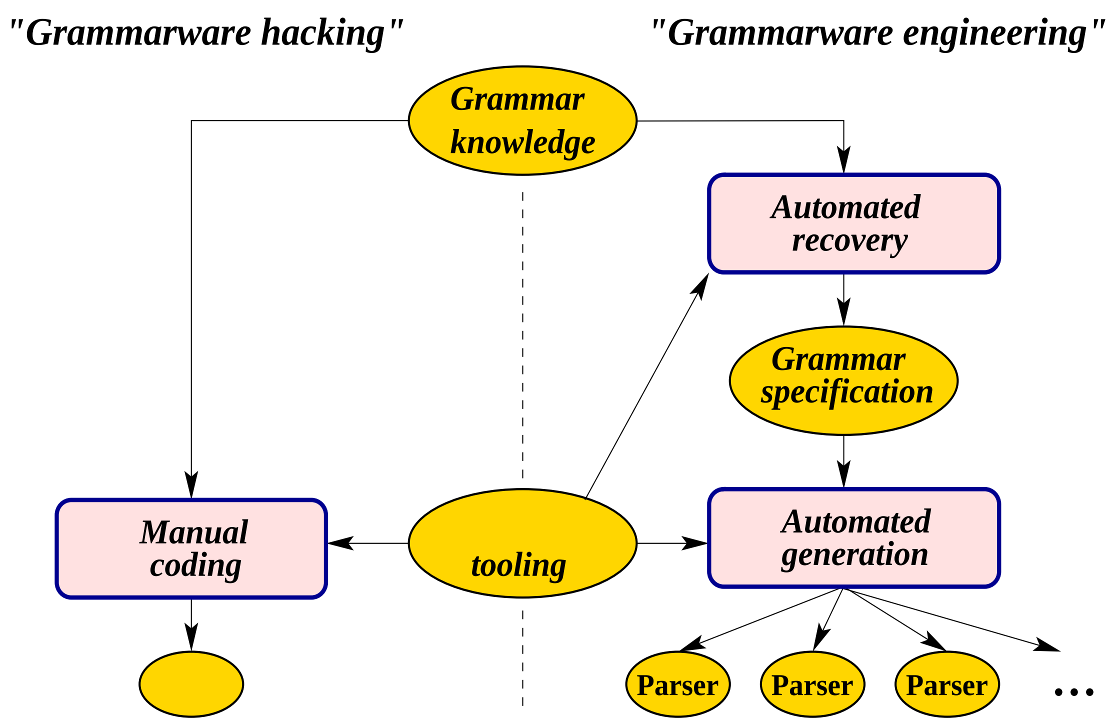
Grammar-driven software

- Application generators, tools for generative programming, aspect-oriented weavers, tools for automated software engineering
- Distributed or component-based applications where grammars occur in the sense of interfaces and I/O formats
- Functionality in language implementations, e.g., compilers, animators, documentation generators, profilers, debuggers, ...
- Functionality for automated software re-documentation, analysis and modification, pre- and post-processors
- Reference manuals, style guides, and industrial standards

GRAMMARWARE REALITY — HACKING

Given the omnipresence of grammarware, one may expect that grammarware is treated as an engineering artifact — subject to reasonable common or best practices. In reality, grammarware is predominantly treated in an *ad-hoc* manner.

EXAMPLE PARSER DEVELOPMENT



LACK OF BEST PRACTICES

- There is no established approach for performing grammar evolution in a traceable manner — not to mention the even more difficult problem of co-evolution of grammar-driven software.

- There is no established approach for maintaining consistency between the incarnations of conceptually the same grammar.

- Grammars are immediately implemented using specific technology, which implies the use of idiosyncratic notations.

LACK OF COMPREHENSIVE FOUNDATIONS

- no comprehensive theory of testing grammarware
- ... of transforming ...
- version management?
- quality assessment?
- design patterns?
- debugging?

THE GRAMMAR DILEMMA

Improving on hacking sounds like such a good idea!
Why did it not happen?

Myth “Grammars are a buried subject”

Myth “Engineering of grammarware = parser generation”]

Myth “XML is the answer”

Myth “Grammarware is all about programming languages”

In need of a paradigm shift

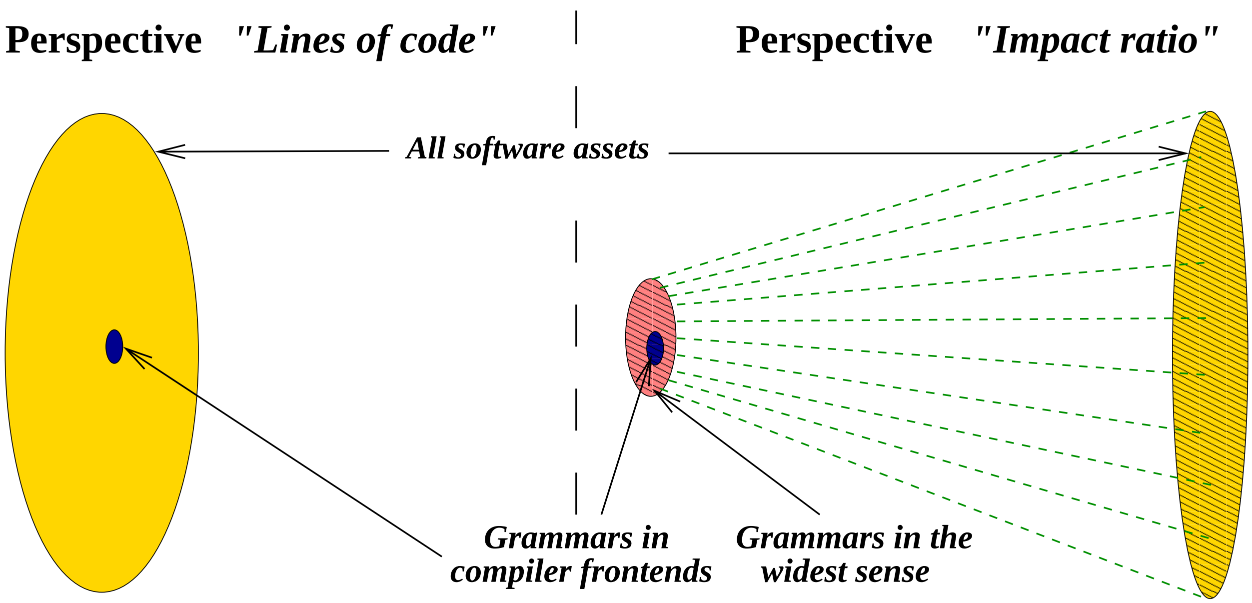
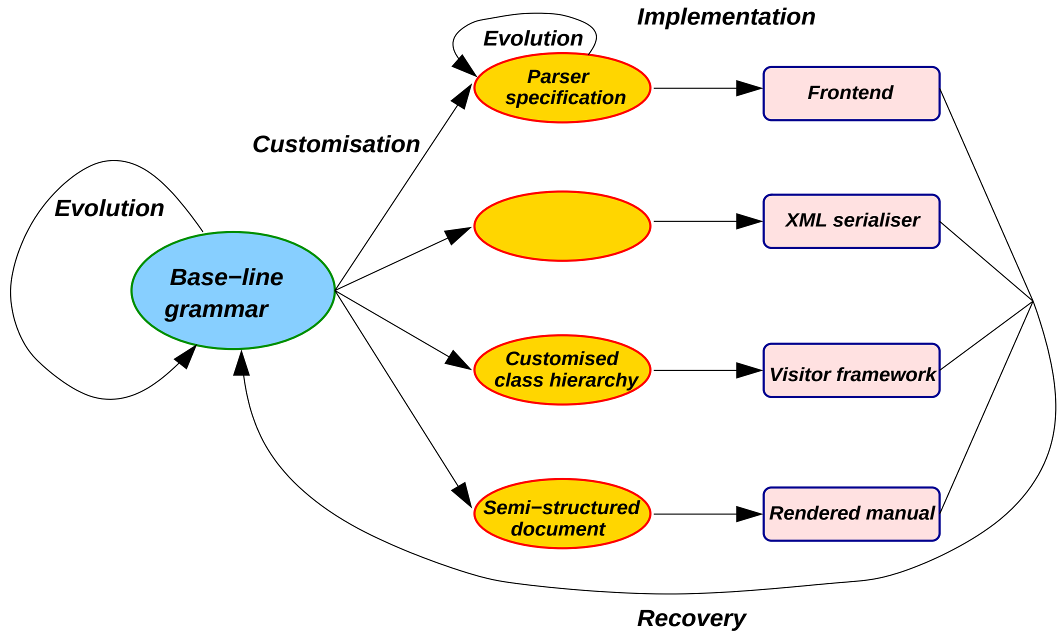


Fig.: The mythical view and the proposed view on grammarware

Engineering of grammarware — principles

- *Abstraction*
- *Customisation*
- *Modularity*
- *Evolution*
- *Assessment*
- *Automation*

ENGINEERING OF GRAMMARWARE — A LIFE CYCLE



A CASE STUDY IN GRAMMAR RECOVERY AND DEPLOYMENT — VS COBOL II

- Input: IBM's industrial standard for VS COBOL II
- Result: public, normative COBOL grammar

Time for a demo.

GRAMMAR TRANSFORMATIONS AS A PARADIGM

- Grammar re-/reverse engineering
  - Correction
  - Completion
  - Restructuring
  - Style conversion

Correction by transformations

Informal rule in IBM's reference: "Level-number, data-name-1, and FILLER are not part of the REDEFINES clause itself, and are included in the format only for clarity."

Transformation

```
focus on REDEFINES-clause do
 delete level-number (data-name | FILLER)
```

COMPLETION BY TRANSFORMATIONS

Informal rule in IBM's Reference: "A series of imperative statements can be specified whenever an imperative statement is allowed."

Transformation

generalise `imperative-statement`
to {`imperative-statement`}+

Grammar transformations as a paradigm

- Grammar implementation
  - Style conversion
  - Disambiguation
  - Conflict resolution
  - Preprocessing
  - Grammar minimalisation

GRAMMAR TRANSFORMATIONS AS A PARADIGM

- Grammar maintenance
  - Evolution
  - Optimisation
  - Migration

A FRAMEWORK FOR GRAMMAR TRANSFORMATIONS

- Suitable notion of grammars
- Primitive operators
- Combinators
- Conditions
- A suite of defined operators

SOME OPERATORS FOR GRAMMAR CONSTRUCTION

generalise P **in** F **to** P'	$\equiv$	P' **covers** $P\,?$; **replace** P **by** $P'\,!_{/F}$; P' **covers** $P\,?$
include P **for** N	$\equiv$	**defined** $N\,?$; **add** $N \rightarrow P\,!$
resolve N **as** P	$\equiv$	**bottom** $N\,?$; **add** $N \rightarrow P$
unify N **to** N'	$\equiv$	**bottom** $N\,?$; $\neg$ **fresh** $N'\,?$; **replace** N **by** N'

TRANSFORMATION OPERATORS; THEIR PROPERTIES

Operator	Preservation	Inverse
Refactoring		
preserve	strict	**preserve**
fold	introducing	**unfold**
unfold	strict	**fold**
introduce	introducing	**eliminate**
eliminate	eliminating	**introduce**
rename	modulo renaming	**rename**
Construction		
generalise	increasing	**restrict**
include	increasing	**exclude**
resolve	resolving	**reject**
unify	essentially resolving	**separate**
Destruction		
restrict	decreasing	**generalise**
exclude	decreasing	**include**
reject	rejecting	**resolve**
separate	essentially rejecting	**unify**
delete	"zig-zag"	

A PROCESS FOR GRAMMAR RECOVERY

- Lack of a reference grammar for some intended language L.
- Availability of an approximative grammar γ_0 for L.
- Availability of representative portfolio C for L.
- Derive the ultimate grammar γ_n for L from γ_0.
- γ_n is (absolutely) complete if $L \subseteq \mathcal{L}(\gamma_n)$.
- γ_n is (absolutely) correct if $\mathcal{L}(\gamma_n) \subseteq L$.
- γ_i is derived from γ_{i-1} by transformations t_i for correction and completion of γ_0.

Engineered recovery

Role of portfolio C
The γ_i accept more and more of C.

Completeness
γ_n parses all of C.

Correctness
γ_n is covered by C.

Preservation
γ_n preserves γ_0 as much as possible.

Operators
The t_i are weakly semantics-preserving.

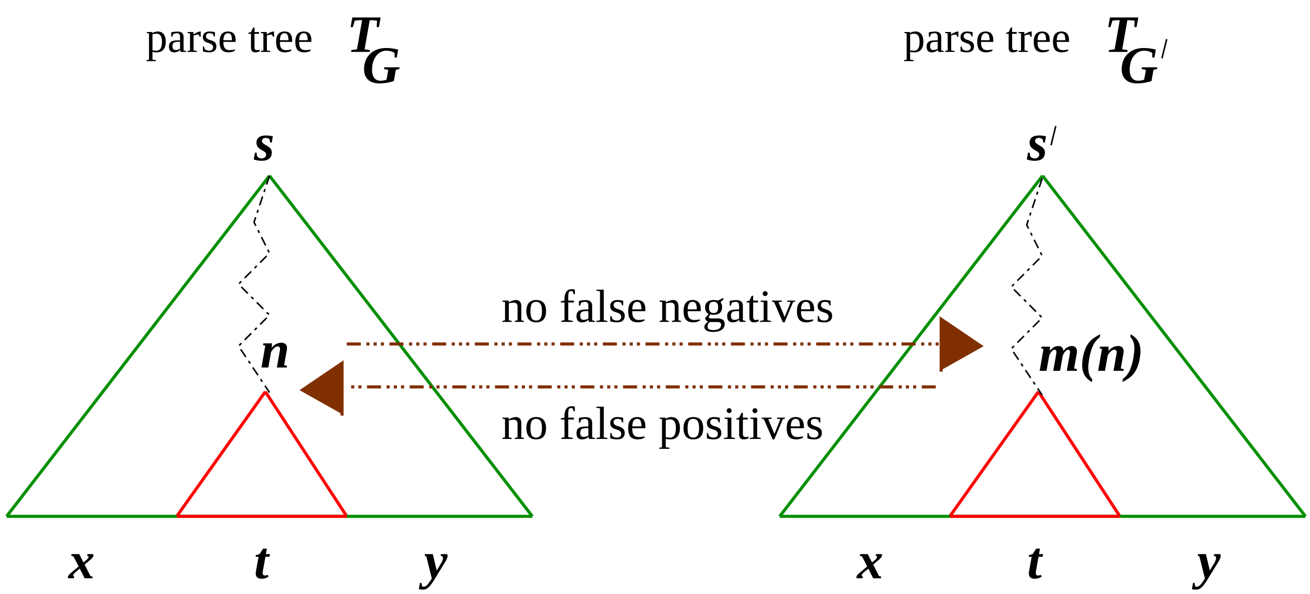
Completion
Missing branches are added.

Correction
Invalid branches are removed.

ENGINEERING OF GRAMMARWARE — SHOWCASES OTHER THAN GRAMMAR RECOVERY AND PARSER DEVELOPMENT

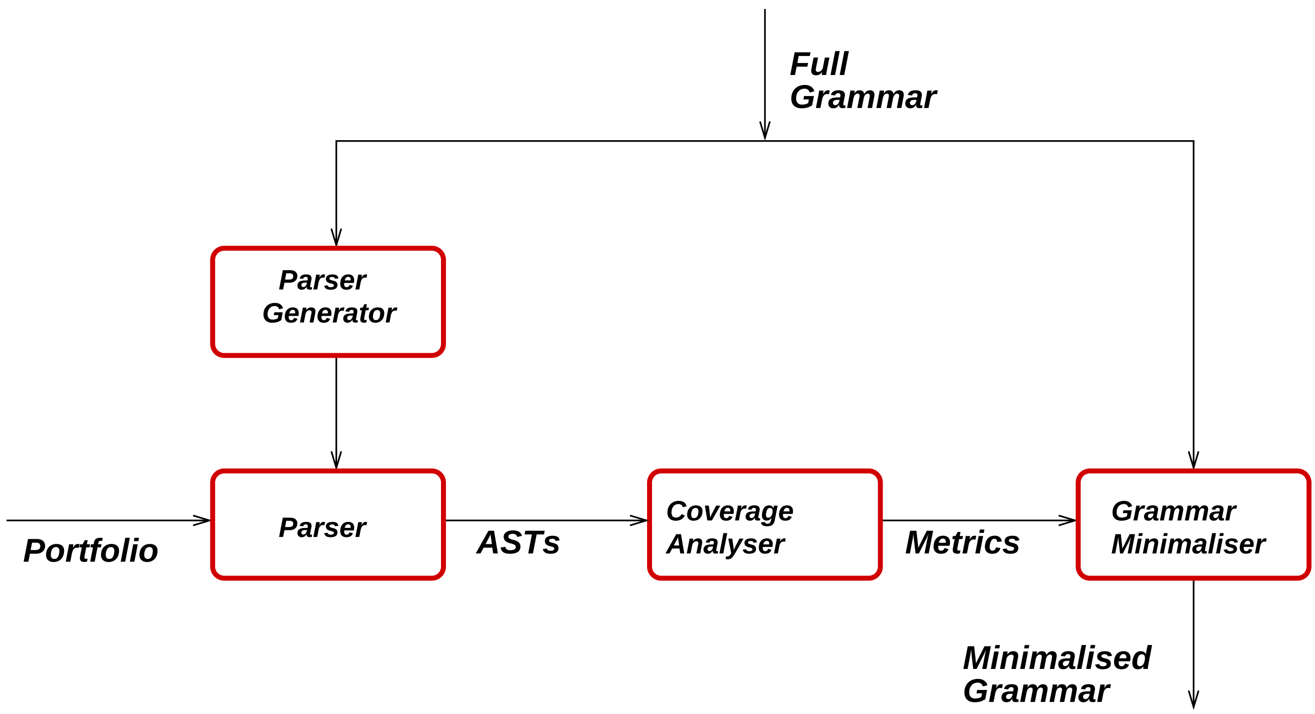
- Component-specific, tolerant parsers
- Portfolio-based grammar specialisation
- Transposition to XML: format evolution
- API-fication
- ...

COMPONENT-SPECIFIC, TOLERANT PARSERS



for all $x\,t\,y$ ***in*** $L(G)$

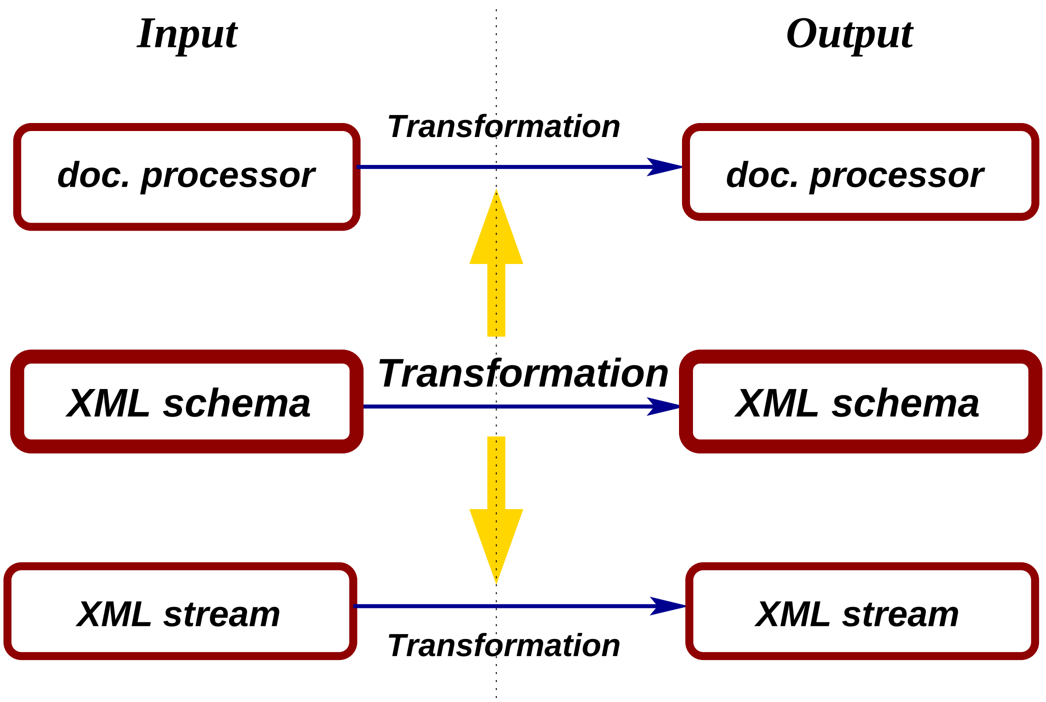
PORTFOLIO-BASED GRAMMAR SPECIALISATION



XML Engineering

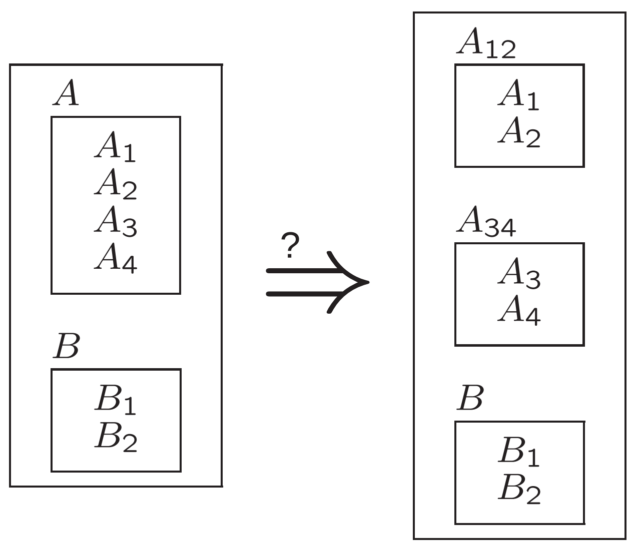
- XML increasingly used
  - Storage formats
  - Exchange formats
  - in component-based, distributed applications.
- DTD/XML-Schemas are grammars.
- XML schemas do evolve.
- XML challenges grammar engineering.
- Format evolution $\approx$ DB scheme evolution.

Two levels of XML transformation

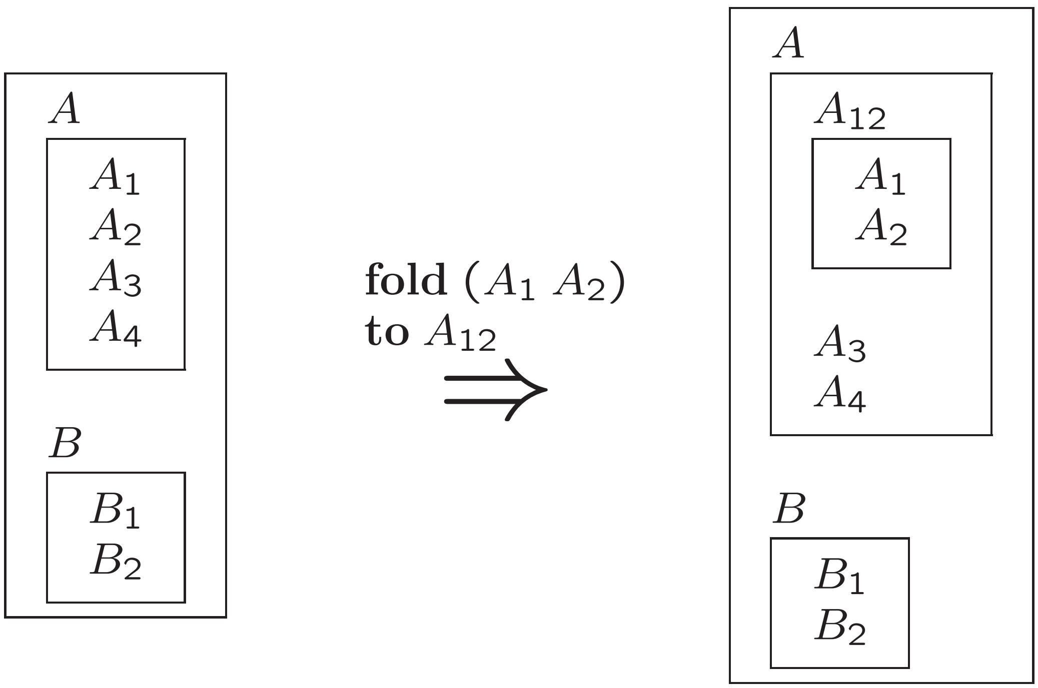


XML demo:

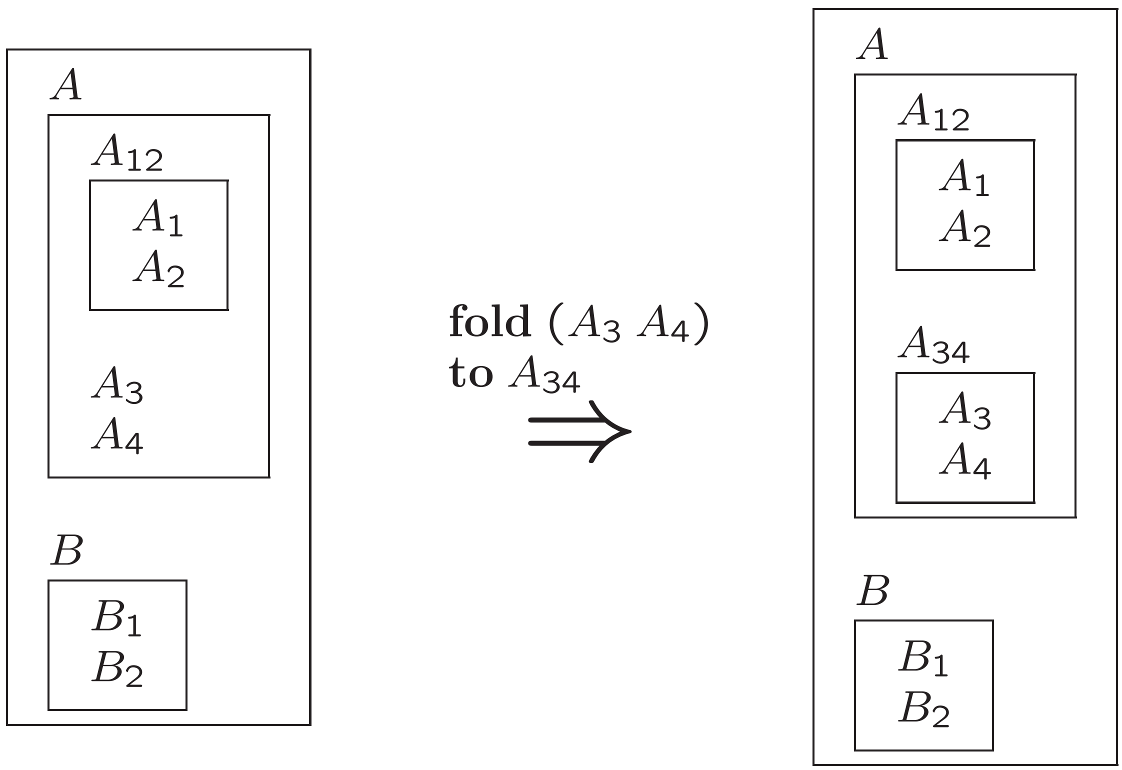
Stepwise transformation of a form



Step 1

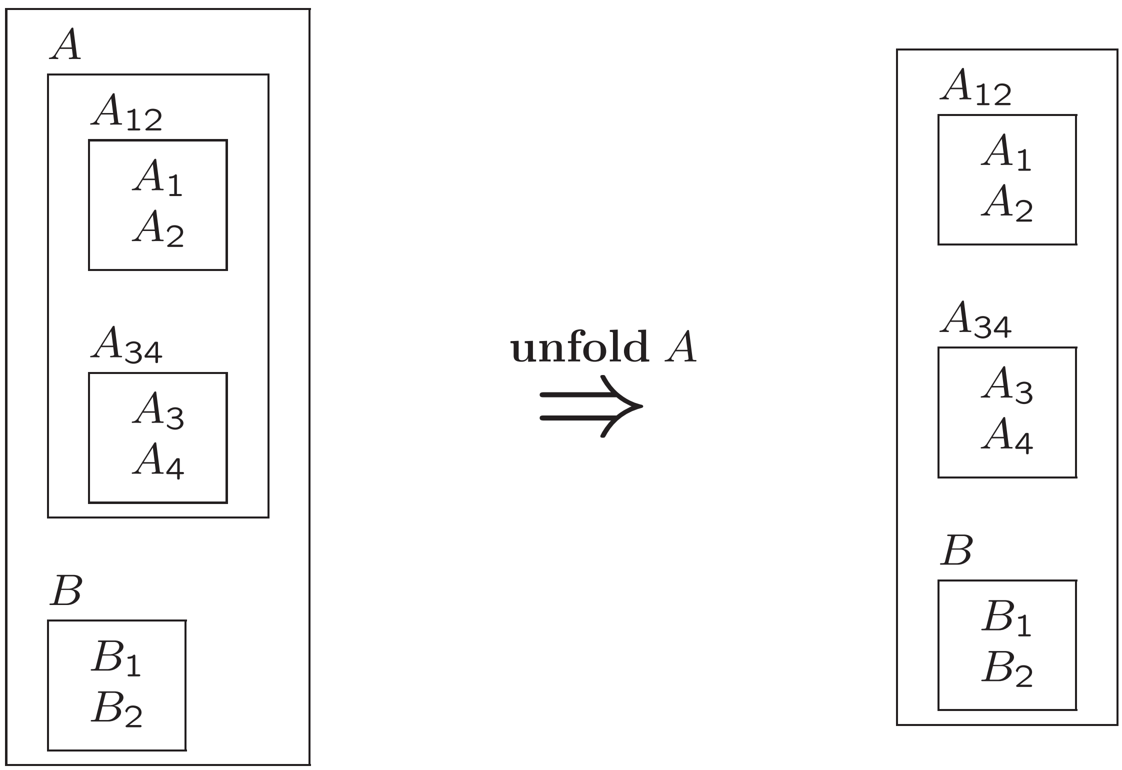


Step 2



A
A_{12}
A_1
A_2
A_3
A_4
B
B_1
B_2
fold $(A_3\ A_4)$
to A_{34}
A
A_{12}
A_1
A_2
A_{34}
A_3
A_4
B
B_1
B_2

Step 3



A
A_{12}
A_1
A_2
A_{34}
A_3
A_4
B
B_1
B_2
unfold A
$\Rightarrow$
A_{12}
A_1
A_2
A_{34}
A_3
A_4
B
B_1
B_2

THINGS TO REMEMBER

- Recover base-line grammars
- Derive grammar uses
- Assess quality of grammars
- Keep track of grammar adaptation
- Co-evolve grammar-based software
- Treat grammars as contracts

Last slide: Cornerstones of the software evolution age

- *Component–based*
- *Executable specifications*
- *Alg. for program analysis*
- *Generative programming*
- *Generic programming*
- *Grammar–based*
- *Pragmatics*
- *Methodology*

End of slide show.