UNIVERSITÄT KOBLENZ • LANDAU

Institut für Informatik

Software Languages Team

FB4

Prof.Dr.
Ralf Lämmel

Dipl.-Math.
Vadim Zaytsev

Semi-Automated Language Convergence

Grammar knowledge is ingrained in different kinds of software artifacts.

We want to establish and maintain correspondence among them.

Universal, not language-specific methodology is needed.

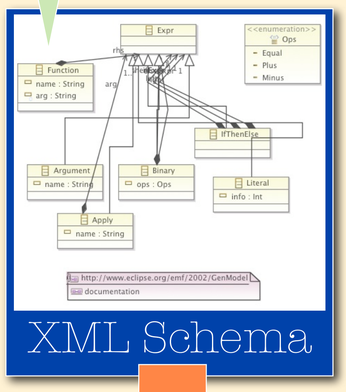
Grammar convergence is lightweight verification for mapping, binding, implementations, dialects, etc.

XML Schema:
* data model
* domain representation

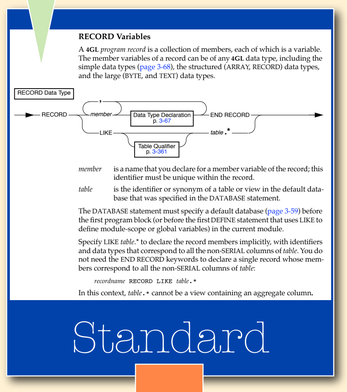
Documentation:
* language standard
* coder's manual
* formal specification

Grammarware:
* parser source code
* compiler sources
* pretty-printer
* IDE, ...

Syntax definitions:
* specific formalisms
* usually (E)BNF
* railroad tracks



XML Schema



RECORD Variables

A 4GL *program record* is a collection of members, each of which is a variable. The member variables of a record can be of any 4GL data type, including the simple data types (page 3-88), the structured (ARRAY, RECORD) data types, and the large (BYTE, and TEXT) data types.

RECORD Data Type

RECORD — member — Data Type Declaration p. 3-87 — END RECORD

LIKE — Table Qualifier p. 3-901 — table.*

member is a name that you declare for a member variable of the record; this identifier must be unique within the record.

table is the identifier or synonym of a table or view in the default database that was specified in the DATABASE statement.

The DATABASE statement must specify a default database (page 3-59) before the first program block (or before the first DEFINE statement that uses LIKE to define module-scope or global variables) in the current module.

Specify LIKE *table*.* to declare the record members implicitly, with identifiers and data types that correspond to all the non-SERIAL columns of *table*. You do not need the END RECORD keywords to declare a single record whose members correspond to all the non-SERIAL columns of *table*:

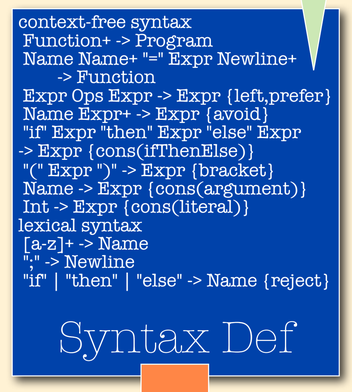
recordname RECORD LIKE *table*.*

In this context, *table*.* cannot be a view containing an aggregate column.

Standard

```
import fl.*;
import java.io.*;
import javax.xml.bind.*;
public class TestIO {
public static void main (String[]
args) throws Exception {
JAXBContext jaxbContext =
JAXBContext.newInstance("fl");
Unmarshaller unMarshaller =
jaxbContext.createUnmarshaller();
Program program = (Program)
unMarshaller.unmarshal(new
File(args[0]));
String s = program.prettyPrint();
FileOutputStream output = new ...
```

Parser



```
context-free syntax
 Function+ -> Program
 Name Name+ "=" Expr Newline+
      -> Function
 Expr Ops Expr -> Expr {left,prefer}
 Name Expr+ -> Expr {avoid}
 "if" Expr "then" Expr "else" Expr
-> Expr {cons(ifThenElse)}
 "(" Expr ")" -> Expr {bracket}
 Name -> Expr {cons(argument)}
 Int -> Expr {cons(literal)}
lexical syntax
 [a-z]+ -> Name
 "," -> Newline
 "if" | "then" | "else" -> Name {reject}
```

Syntax Def

Sources:
*extraction
*parsing
*evalutation

Extractor:
* mapping
* abstraction
* unified output

grammar

grammar

grammar

grammar

BGF:
unified grammar format

Transformations:
*semi-automated
*programmable
*functional
*refactorings
*language increase
*language decrease
*editing

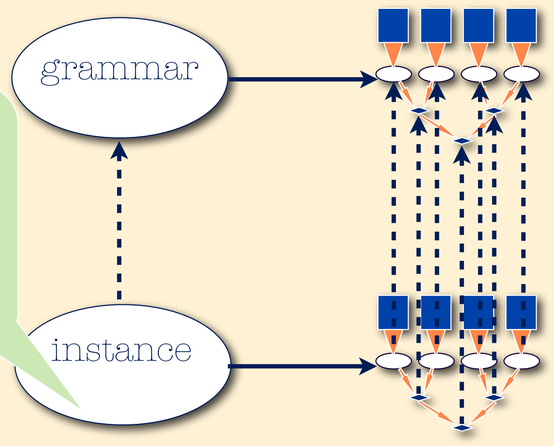
Targets:
* convergence point
* branches converge
* grammar comparator

Phases:
* preparation
* nominal matching
* structural matching
* extension
* relaxation
* correction

Related research topics for student projects:

* IDE support for interactive grammar transformation
* Full XML Schema support for grammar extraction
* Optimization of transformations by deforestation
* Proof of correctness for coupled transformations
* Model-based grammar comparison

Coupled transformations:
* repeat with instances
* use as test cases:
* parse & evaluate



Universität Koblenz-Landau • Campus Koblenz • Universitätsstraße 1 • 56070 Koblenz
Telefon +49 261 287-2711 • Fax +49 261 287-2721