Graph Grammars Applied to Metamodels and Flowcharts

Luka Fürst[1], Marjan Mernik[2], Viljan Mahnič[1], Barrett R. Bryant[3], Jeff Gray[3]
[1]University of Ljubljana, Slovenia; [2]University of Maribor, Slovenia; [3]University of Alabama at Birmingham, USA

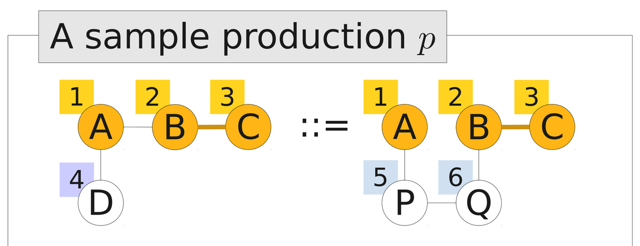
Graph grammars (as defined by Rekers & Schürr)

Introduction

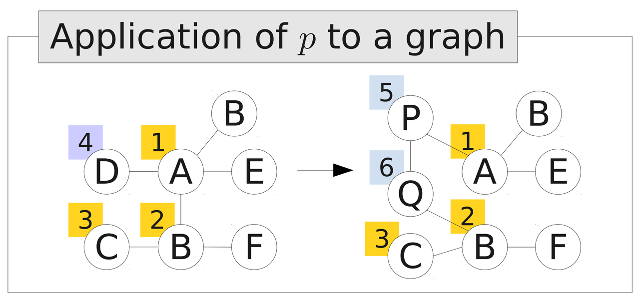
- Source: J. Rekers, A. Schürr: *Defining and parsing visual languages with Layered Graph Grammars.* Journal of Visual Languages and Computing, 1997.
- Rekers & Schürr defined graph grammars as a generalization of context-sensitive string grammars.
- Primary use: Defining the syntax of visual languages.
- A graph grammar consists of rules (*productions*) for replacing subgraphs in an arbitrary graph.
- Every production takes the form $L ::= R$, where L, R, and $L \cap R$ (the context) are proper graphs.

A production and its application

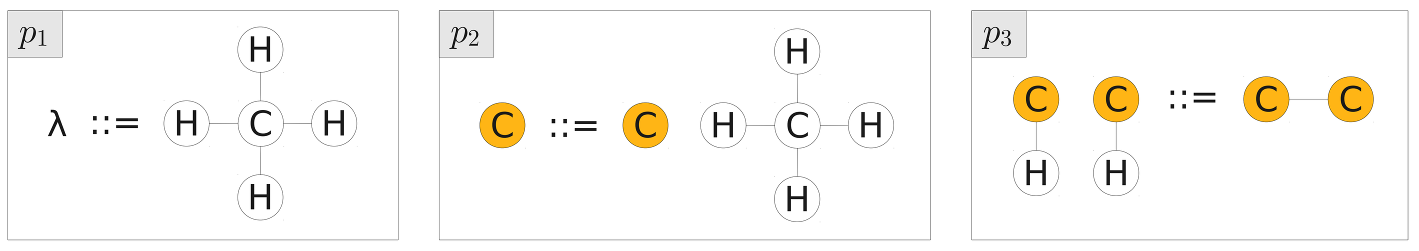
A sample production p



Application of p to a graph

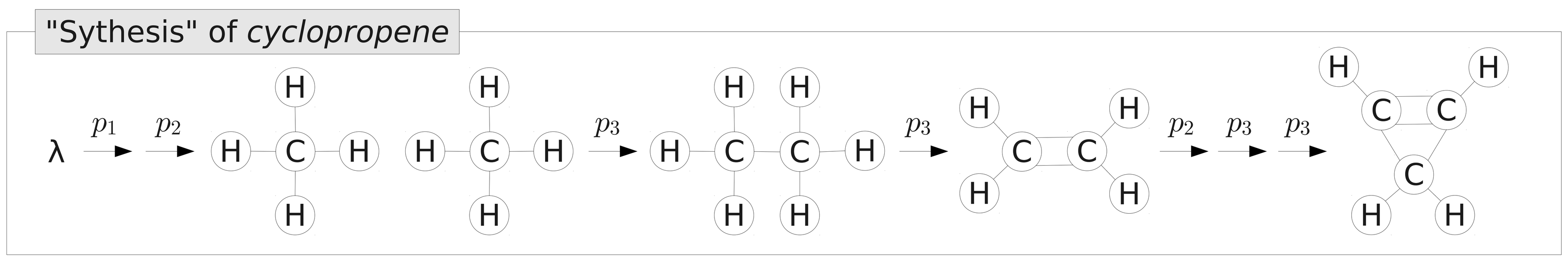


Example: a grammar of hydrocarbon graphs



This simple grammar can generate any hydrocarbon graph, acyclic or cyclic.

"Sythesis" of *cyclopropene*



Converting a metamodel into an equivalent graph grammar

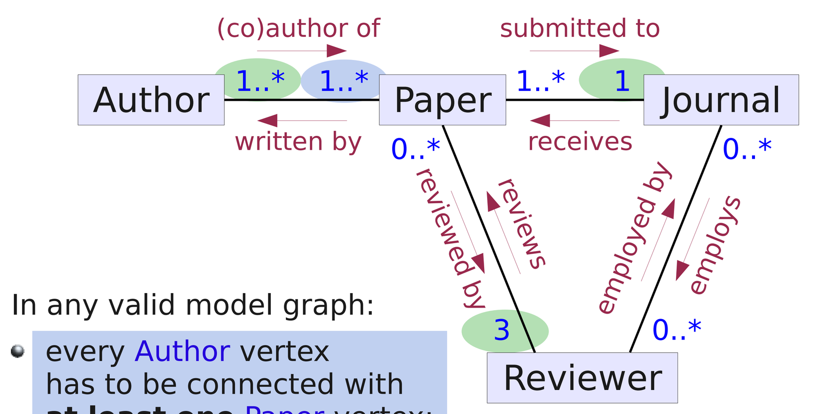
Introduction

- Problem statement: For a given metamodel, construct a graph grammar that generates precisely the graphs of those models that conform to the metamodel.
- Motivation:
  - A metamodel does not provide a means to generate its models.
  - Graph grammars can be augmented with semantic information.
- Related work: K. Ehrig et al., Softw. & System Modeling, 2009.

Properties of the grammar construction process

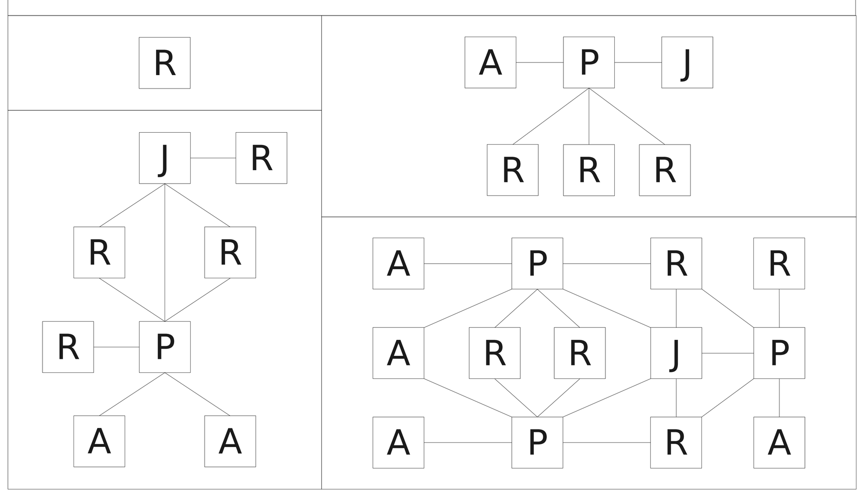
- The output grammar consists of both terminal and nonterminal symbols.
- A nonterminal $V_{l(1)\,A(1),\ldots,l(k)\,A(k)}$ represents a vertex V that still has to be connected with $l(1)$ vertices $A(1)$, $l(2)$ vertices $A(2)$, ..., and $l(k)$ vertices $A(k)$.
- A terminal V represents a vertex V for which all connection requirements are satisfied.
- For each nonterminal, the grammar contains a set of productions that (immediately or gradually) convert it to a terminal.
- Every sentential form that the grammar can generate represents a (*potentially*) *valid* model graph:
  - If a sentential form contains only terminals, then it *is* a valid model graph.
  - Otherwise, the nonterminals in a sentential form determine how the sentential form can be completed to a valid model graph.

Sample metamodel and some of its models



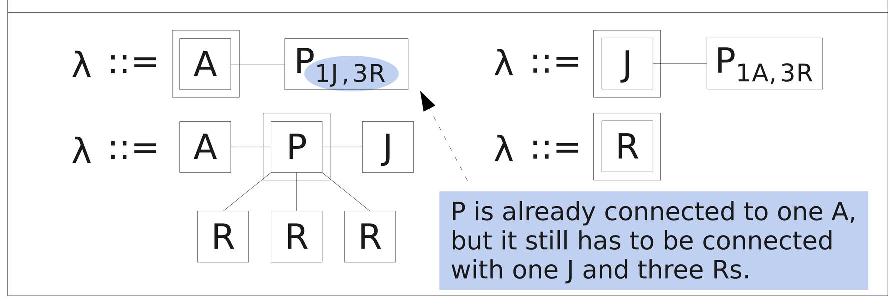
In any valid model graph:

- every Author vertex has to be connected with **at least one** Paper vertex;
- every Paper vertex has to be connected with **at least one** Author vertex, **exactly one** Journal vertex, and **exactly three** Reviewer vertices;
- ...



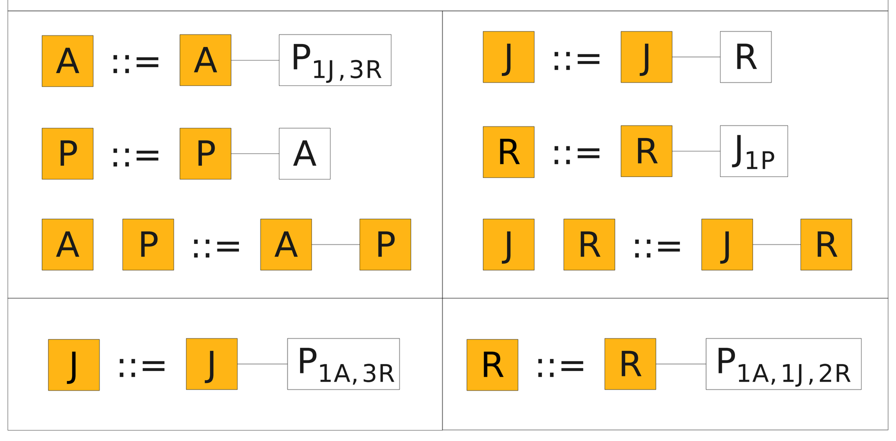
1 Initial productions

For each class C, add a production $\lambda ::= C - \{Required\ connections\ of\ C\}$.



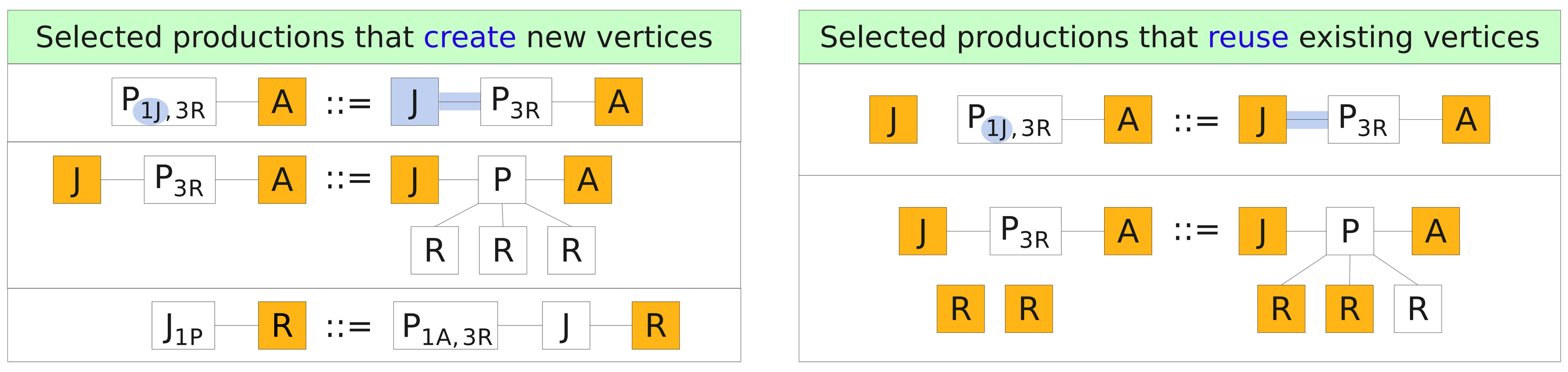
2 Productions for unbounded multiplicities

- For each unbounded X-to-Y multiplicity, add a production $\underline{X} ::= \underline{X} - Y_{Req(Y)}$
- For each two-way unbounded multiplicity, also add a production $\underline{X}\ \underline{Y} ::= \underline{X} - \underline{Y}$



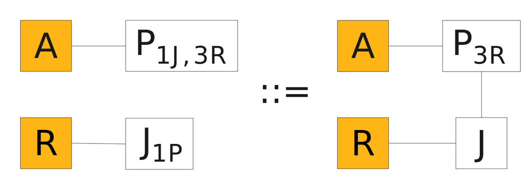
3 Expanding nonterminal forms

- Nonterminal form: A sentential form composed of a nonterminal vertex and all vertices connected to it.
- Expansion: $V_{Req1,\ Req2,\ \ldots,\ ReqN} - \{\underline{Conns}\} ::= V_{Req2,\ \ldots,\ ReqN} - \{\underline{Conns}, Connections\ from\ Req1\}$
- Every nonterminal form has to be eventually expanded.
- Expansion may create new nonterminal forms, but the process is guaranteed to terminate.

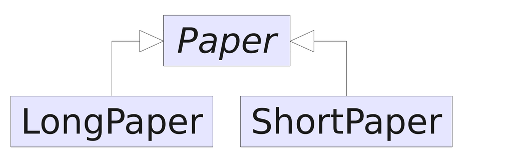


4 Complementary nonterminal form pairs

- In some cases, two nonterminal forms may mutually satisfy their connection requirements.
- Example: P needs a connection with one J, and J needs a connection with one P.



5 Dealing with inheritance



- A simple solution: The terminal P is re-interpreted as a wildcard that can be replaced either with LP or with SP.

Translating flowcharts into Java programs using graph grammar parsing

Introduction

Problem statement: Convert a given flowchart into an equivalent Java program.

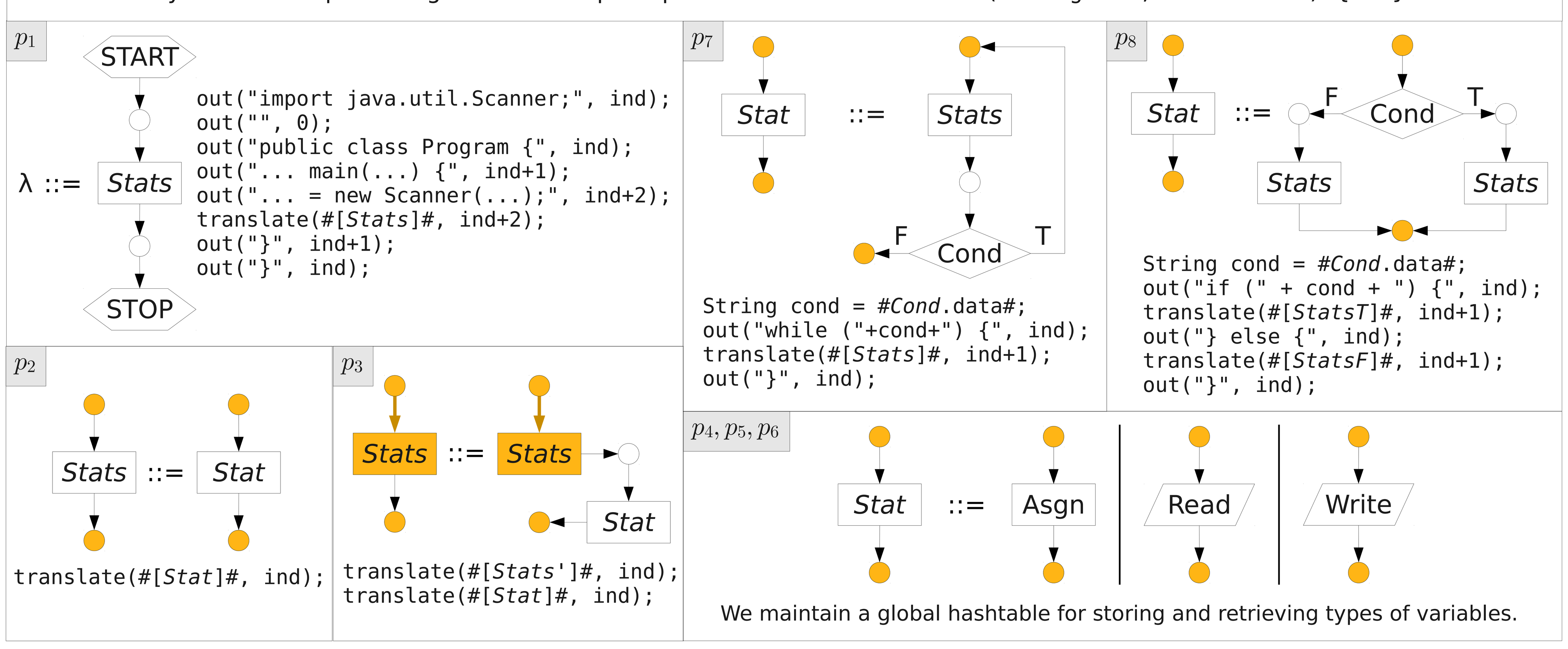
Motivation: This is a simple example, but a similar approach could be used for parsing and translating real-world visual languages.

Implementation:

1. Define a flowchart graph grammar and a translation scheme.
2. Parse a given flowchart against this grammar using the Rekers-Schürr parser.
3. Build a syntax graph from the derivation produced by the parser.
4. Recursively execute the translation scheme on the syntax graph.

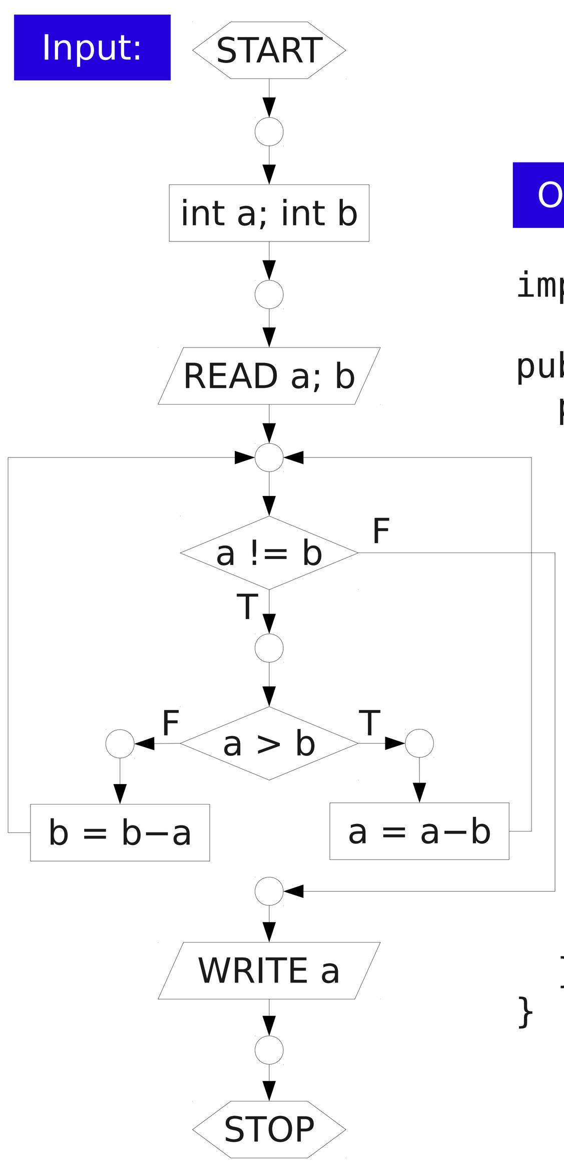
Graph grammar with a translation scheme

Translation method header: `private void translate(#root#, int indent)`
Invocation of the `translate` method on the λ node: `translate(#[lambda]#, 0);`
An auxiliary method for producing indented output: `private static void out(String str, int indent) {...}`



Sample input and output

Input:



Output:

```
import java.util.Scanner;

public class Program {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    int a;
    int b;
    a = sc.nextInt();
    b = sc.nextInt();
    while (a != b) {
      if (a > b) {
        a = a–b;
      } else {
        b = b–a;
      }
    }
    System.out.print(a);
  }
}
```

The simplified syntax graph for the sample input

- The `translate` method is invoked on the root (λ) node.
- After the initial invocation, the translation is guided only by the syntax graph and by the translation rules for individual productions.