The Application of Grammar Inference to Software Language Engineering

M. Mernik[1][2], D. Hrnčič[1],
B. Bryant[2], A. Sprague[2], Q. Liu[2]
L. Fürst[3], V. Mahnič[3]

[1]University of Maribor, Slovenia
[2]The University of Alabama at Birmingham, USA
[3]University of Ljubljana, Slovenia

Univerza v Mariboru
THE UNIVERSITY OF ALABAMA AT BIRMINGHAM
Univerza v Ljubljani

Outline of the Presentation

- Motivation
- Background
- Context-free grammar inference
- Metamodel inference
- Graph grammar inference
- Semantic inference
- Conclusion

Motivation

What is a grammar of this language?

Try out our newly developed grammar inference algorithm!

Motivation

- Some years ago interesting questions were posted on the Usenet group comp.compilers:

“I am looking for an algorithm that will generate context-free grammar from given set of strings. For example, given a set $L = \{aaabbbbb, aab\}$ one of the grammar is $G \to AB$, $A \to aA \mid a$, $B \to b \mid bB$"

Motivation

“I'm working on a project for which I need information about some reverse engineering method that would help me extract the grammar from a set of programs (written in any language). A sufficient grammar will be the one which is able to parse all the programs ..."

Motivation

- Those questions triggered some interesting responses:

“Unfortunately, there are infinitely many context-free grammars for any given set of strings (Consider for example adding A → C, C → D, ..., Y → Z, Z → A to the above grammar. You can obviously add as many pointless rules as you want this way, and the string set doesn't change) ..."

Motivation

"Within machine learning there is a subfield called Grammatical Inference. They have demonstrated a few practical successes mostly at the level of recognizing regular languages or subsets thereof …"

Motivation

“There are formal theories that address this. However, their results are far from encouraging. The essential problem is that given a finite set of programs, there is a trivial regular expression which recognizes exactly those set of programs and no others …”

Motivation

“There is a way to deal with this issue. Let us assume for the moment that the program is compiled by a compiler. Then the grammar knowledge that you need resides in that compiler. What you do is write a parser that parses the part of the compiler containing the grammar knowledge. If you are lucky this is easy and you recover the BNF in a snippet. If … and it is not possible to obtain the source code of the grammar there is another option. You can extract the grammar from the manual."

Background

- **Grammatical inference** is a process of learning the grammar from positive (and negative) language samples.
- Grammatical inference attracts researchers from different fields such as pattern recognition, computational linguistic, natural language acquisition, software engineering, ...

Background

- Context-Free Grammar G=<N, T, P, S>
- $L(G) = \{w \mid S \Rightarrow^* w, w \in T^*\}$
- Given a sentence ps and CFG G we can tell whether ps belongs to L(G) ($ps \in L(G)$). Such sentence is called positive sample.
- A set of positive samples is denoted with S^+. In similar manner we can defined set of negative samples S^-. Those samples do not belong to L(G) and can no be derived from starting symbol S.

Background

- Given a set S^+ and S^-, which might be also empty, the task of context-free grammar inference is to find at least one context-free grammar G such that $S^+ \subseteq L(G)$ and $S^- \subseteq \overline{L}(G)$.
- A set of positive samples S^+ of a L(G) is structurally complete if each grammar production is used in the generation of at least one sentence in S^+.

Background

- Gold Theorem (1967) - it is impossible to identify any of the four classes of languages in the Chomsky hierarchy in the limit using only positive samples. Using both negative and positive samples, the Chomsky hierarchy languages can be identified in the limit.

Background

- Intuitively, Gold's theorem can be explained by recognizing the fact that the final generalization of positive samples would be an automation that accept all strings.
- Singular use of positive samples results in an uncertainty as to when the generalization steps should be stopped. This implies the need for some restrictions or background knowledge on the generalization process.

Background

- A lot of research has been done on extraction of context-free grammars, but the problem is still not solved sufficiently mainly due to immense search space.

Background

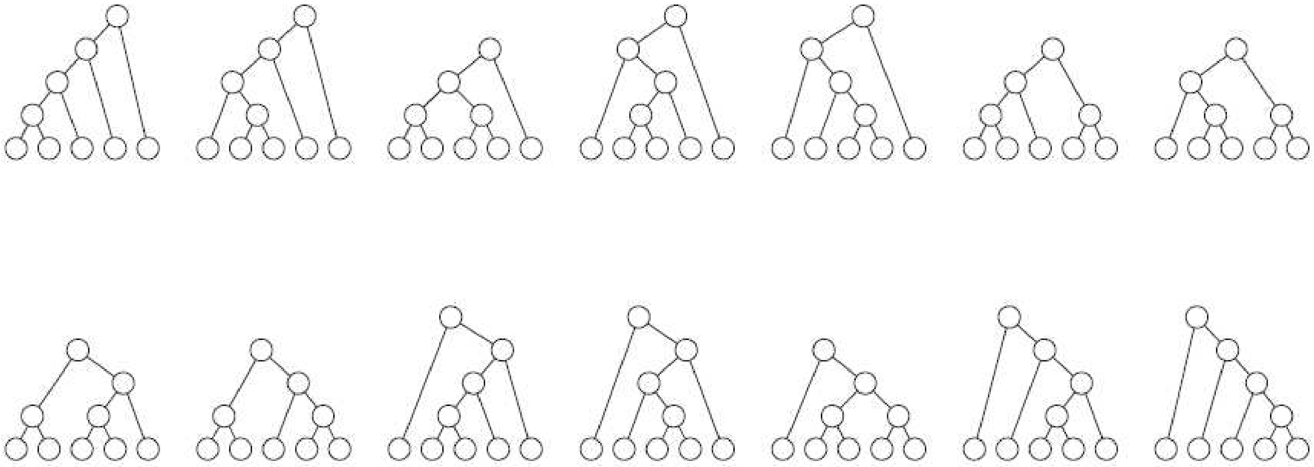


Fig. 3. All full binary trees for $l = 5$ ($n = 4$)

n	Number of full binary trees (Catalan numbers)
1	1
2	2
3	5
4	14
5	42
6	132
7	429
8	1430
9	4862
10	16796
11	58786
12	208012
13	742900
14	2674440

Background

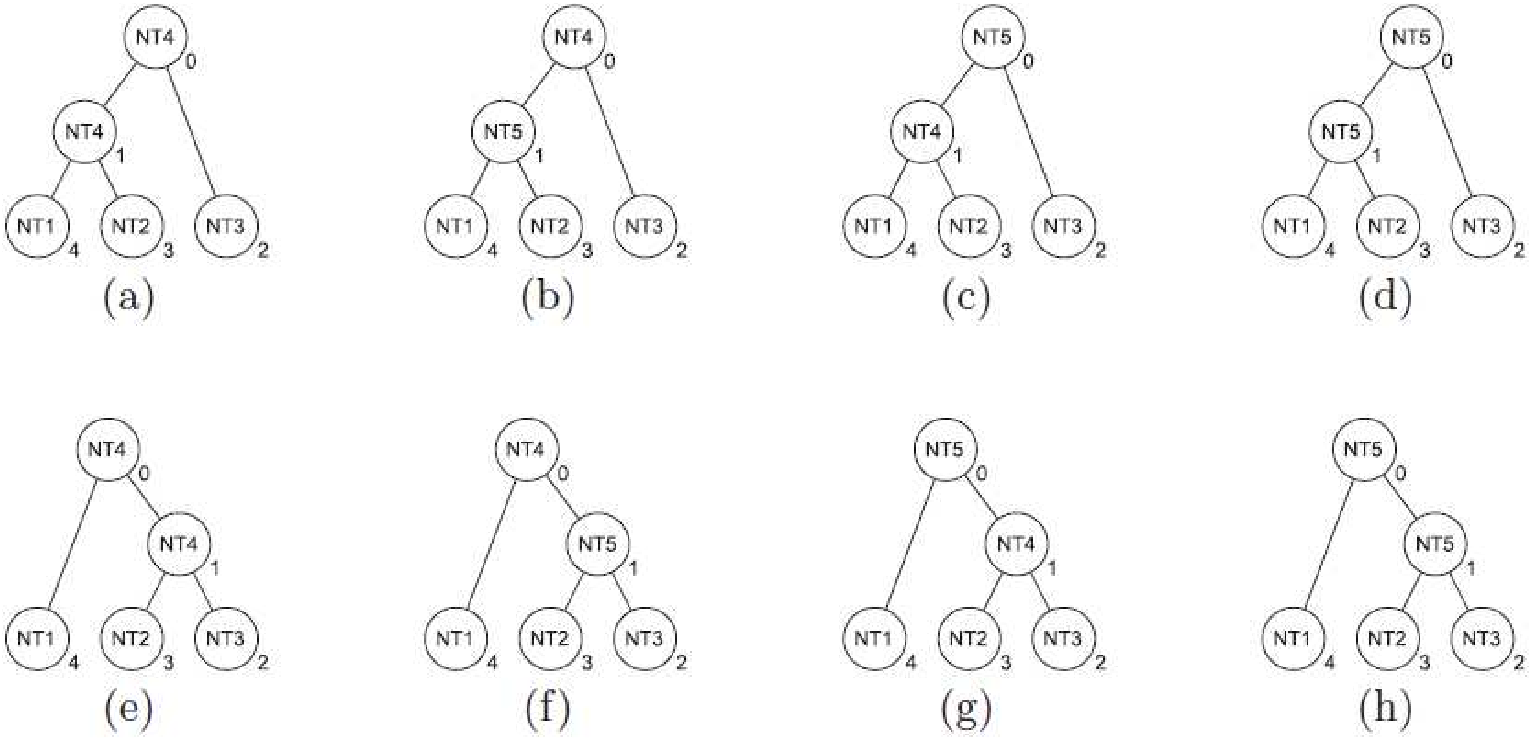
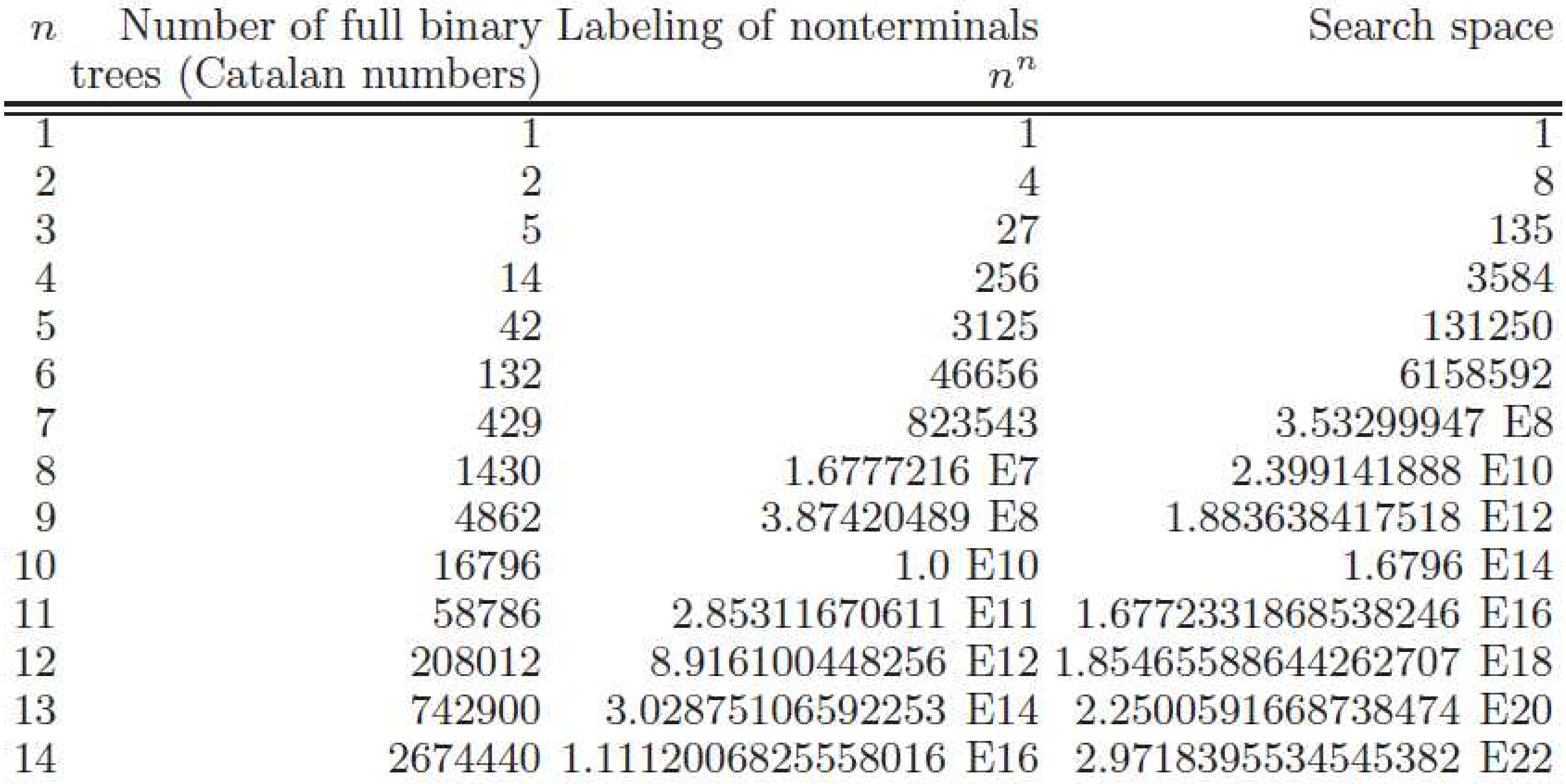


Fig. 4. All possible labeling's of nonterminals for $l = 3 (n = 2)$

Background



n	Number of full binary trees (Catalan numbers)	Labeling of nonterminals n^n	Search space
1	1	1	1
2	2	4	8
3	5	27	135
4	14	256	3584
5	42	3125	131250
6	132	46656	6158592
7	429	823543	3.53299947 E8
8	1430	1.6777216 E7	2.399141888 E10
9	4862	3.87420489 E8	1.883638417518 E12
10	16796	1.0 E10	1.6796 E14
11	58786	2.85311670611 E11	1.6772331868538246 E16
12	208012	8.916100448256 E12	1.85465588644262707 E18
13	742900	3.02875106592253 E14	2.2500591668738474 E20
14	2674440	1.1112006825558016 E16	2.9718395534545382 E22

Background

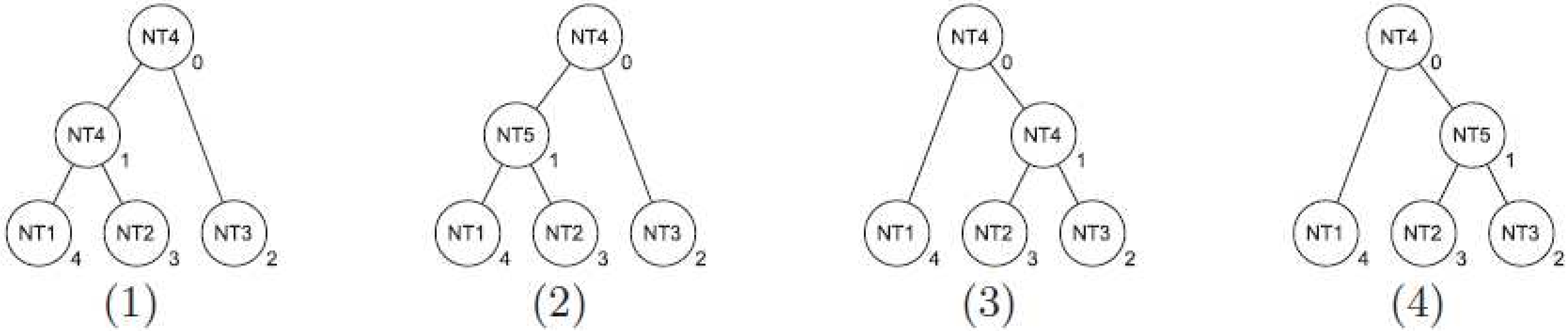
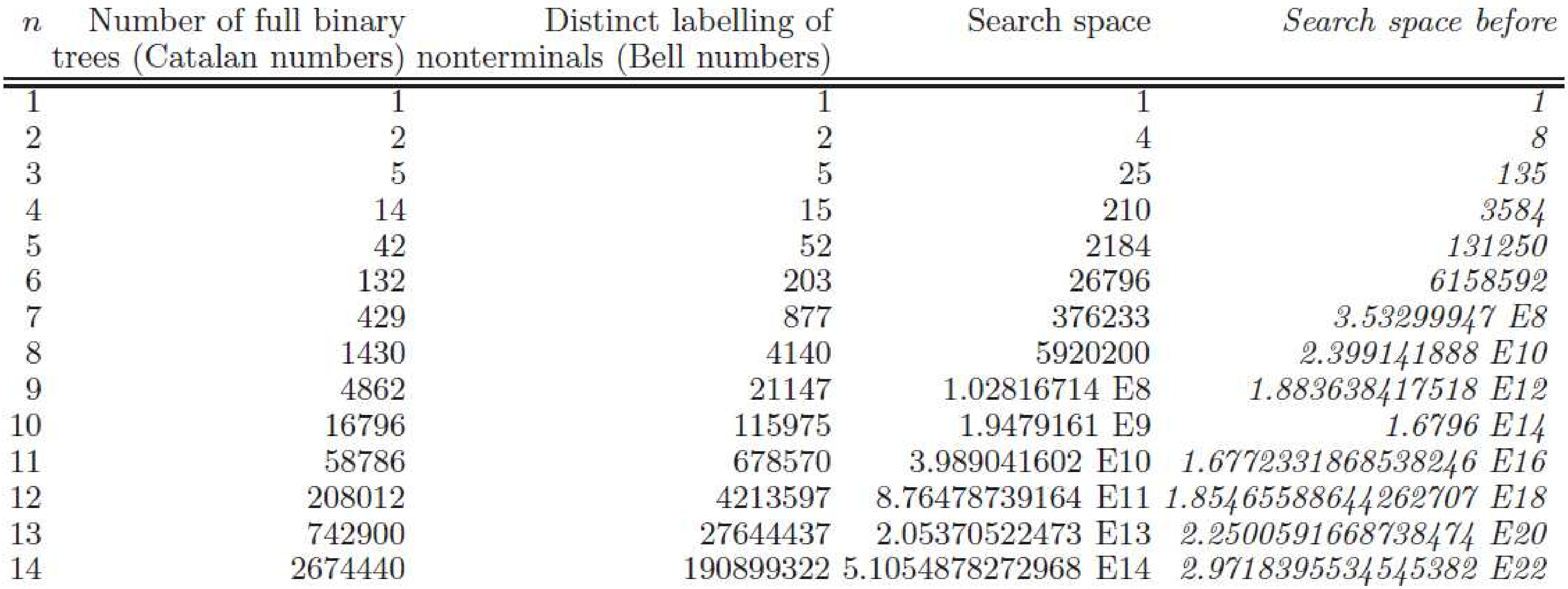


Fig. 5. All distinct labeling's of nonterminals when $l = 3(n = 2)$

Background



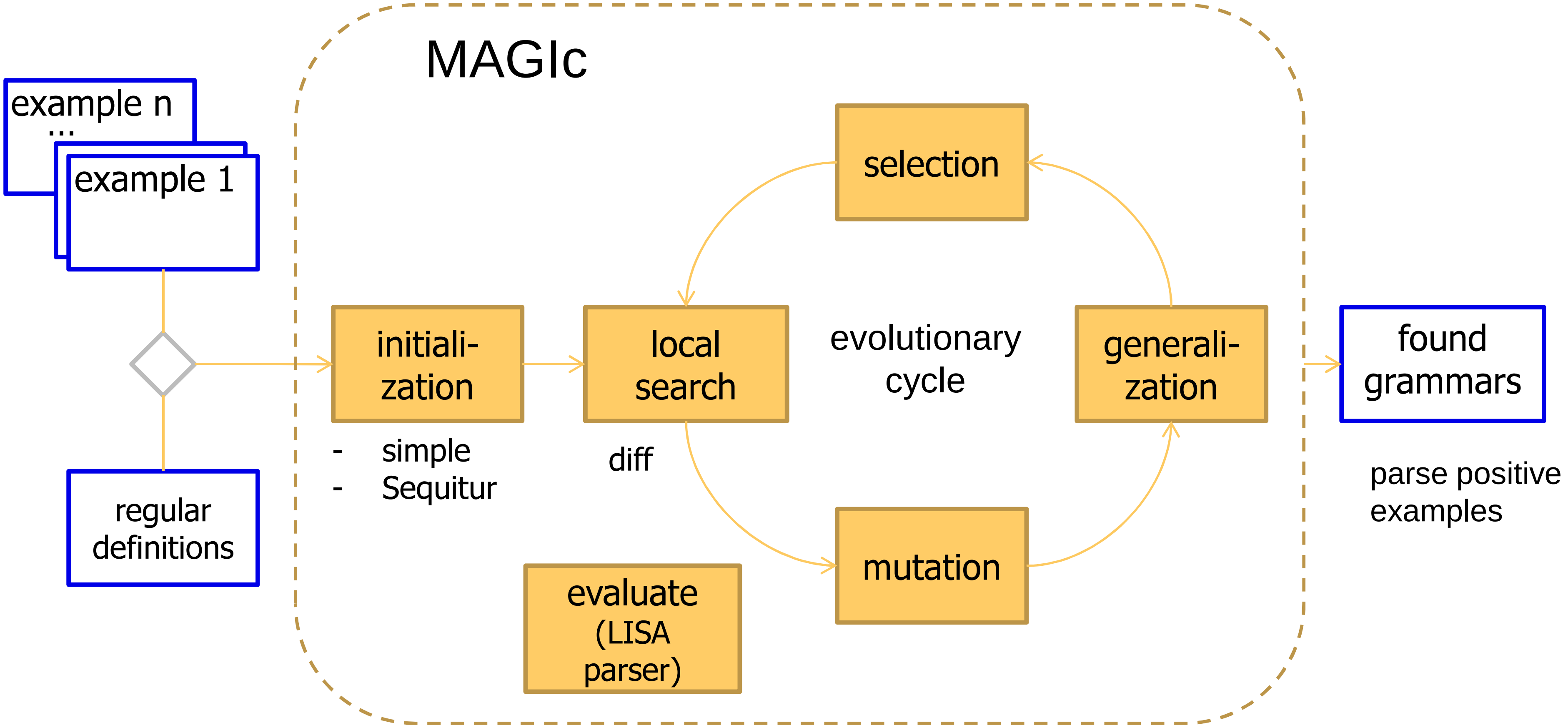
n	Number of full binary trees (Catalan numbers)	Distinct labelling of nonterminals (Bell numbers)	Search space	*Search space before*
1	1	1	1	*1*
2	2	2	4	*8*
3	5	5	25	*135*
4	14	15	210	*3584*
5	42	52	2184	*131250*
6	132	203	26796	*6158592*
7	429	877	376233	*3.53299947 E8*
8	1430	4140	5920200	*2.399141888 E10*
9	4862	21147	1.02816714 E8	*1.883638417518 E12*
10	16796	115975	1.9479161 E9	*1.6796 E14*
11	58786	678570	3.989041602 E10	*1.6772331868538246 E16*
12	208012	4213597	8.76478739164 E11	*1.854655886442627O7 E18*
13	742900	27644437	2.05370522473 E13	*2.2500591668738474 E20*
14	2674440	190899322	5.1054878272968 E14	*2.9718395534545382 E22*

Background

- Memetic algorithms are evolutionary algorithms with local search operator
  - use of evolutionary concepts (population, evolutionary operators)
  - improves the search for solutions with local search.

Context-free grammar inference

- **M**emetic **A**lgorithm for **G**rammatical **I**nferen**c**e



Context-free grammar inference

- Sequitur: http://sequitur.info/
- abcabdabcabd

0 → 1 1
1 → 2 c 2 d
2 → a b

- p i w i=n, i=n   // print id where id=n, id=n

0 → p 1 w 2, 2
1 → i
2 → 1 = n

Context-free grammar inference

print id where id=num
print num+id where id=num

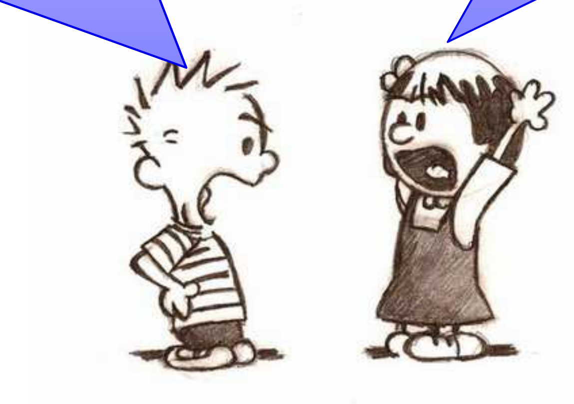
print a where c=2
print 5+b where b = 10

Context-free grammar inference

print id where id=num
print num+id where id=num

But where to change the grammar?

Apply diff command!
1a2,3
> num
> +



Context-free grammar inference

Configurations returned from the LR(1) parser:

$Nx \rightarrow \alpha_1 \bullet \alpha_2$
$Ny \rightarrow \beta \bullet$
$Nz \rightarrow \bullet \gamma$

Use information from LR(1) parsing on 2nd sample.

Context-free grammar inference

- Input samples:

  $s_1,s_2,\ldots,s_n$ (true positive)

  $s_1,s_2,\ldots,s_k,a_1,\ldots,a_m,s_{k+1},\ldots s_n$ (false negative)

  – difference: $a_1,\ldots,a_m$

Context-free grammar inference

- $Nx \rightarrow \alpha_1 \bullet \alpha_2$
  - if $s_{k+1} \in \mathrm{FIRST}(\alpha_2)$

    $Nx ::= \alpha_1\ N1\ \alpha_2$

    $N1 ::= a_{i+1} \ldots a_m$

    $N1 ::= \varepsilon$
  - if $s_{k+1} \notin \mathrm{FIRST}(\alpha_2) \wedge s_{k+1} \in \mathrm{FOLLOW}(Nx)$

    $Nx ::= \alpha_1\ N1$

    $N1 ::= \alpha_2$

    $N1 ::= a_{i+1} \ldots a_m$
  - if $s_{k+1} \notin \mathrm{FIRST}(\alpha_2) \wedge s_{k+1} \notin \mathrm{FOLLOW}(Nx)$

    change in this configuration can’t be made

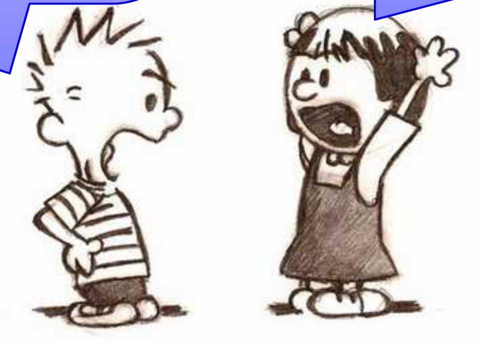
Context-free grammar inference

print a where c=2
print 5+b where b = 10

N1 → print • N2 where id = num

N1 ::= print N2 where id = num
N2 ::= id

N1 ::= print N3 N2 where id = num
N2 ::= id
N3 ::= num +
N3 ::= ε



Context-free grammar inference

But, how mutation is done?

Production: **Nx ::= α1 Ny α2**

Option

Nx ::= α1 Nz α2
Nz ::= Ny
Nz ::= ε

Context-free grammar inference

What about generalization step?

Nx ::= Ny Ny
Ny ::= α
Ny ::= β

→

Nx ::= Ny
Ny ::= α Ny
Ny ::= β Ny
Ny ::= ε

Context-free grammar inference

- 12 input samples of DESK language on which the algorithm was tested:

1. print a
2. print 3
3. print b + 14
4. print a + b + c
5. print a where b = 14
6. print 10 where d = 15
7. print 9 + b where b = 16
8. print 1 + 2 where id = 1
9. print a where b = 5, c = 4
10. print 21 where a = 6, b = 5
11. print 5 + 6 where a = 3, c = 14
12. print a + b + c where a = 4, b = 3, c = 2

Context-free grammar inference

Original grammar:

1. DESK ::= print E C
2. E ::= E + F
3. E ::= F
4. F ::= id
5. F ::= num
6. C ::= where Ds
7. C ::= ε
8. Ds ::= D
9. Ds ::= Ds , D
10. D ::= id = num

Inferred grammar:

1: NT1 -> print NT3 NT5
2: NT2 -> + NT3
3: NT2 -> ε
4: NT3 -> num NT2
5: NT3 -> id NT2
6: NT4 -> , id = num NT4
7: NT4 -> ε
8: NT5 -> where id = num NT4
9: NT5 -> ε

Context-free grammar inference

DSL for hypertree description

```
RESOLUTION 300 400 300
ITERATIONS 3000000
POINTINIT 0 0 0
TREEDEPTH 5
BRANCHDEPTH 1
HYPERVOLUME -0.6 0.6 -1 0.6 -0.6 0.6

DEPTHCOLOR 0-1 0.7+/-0.0 0.7+/-0.0 0.5+/-0.0
DEPTHCOLOR 2-5 0.25+/-0.25 0.75+/-0.25
     0.25+/-0.25
TRANSFORM 1 0
TRANSLATE (0,0,0) (1,1,1) (0,0,0)
SHEAR (0,0,0) (0.5,0.5,0.5) (2,2,2) SHEAR_XZ
SCALE (0.3,0.3,0.3) (0.4,0.4,0.4) (0.3,0.3,0.3)
ROTATE (-80,-80,-80) (0,0,0) (0,0,0)
ROTATE (0,0,0) (45,45,45) (0,0,0)
TRANSLATE (0,0,0) (-0.72,-0.72,-0.72) (0,0,0)

TRANSFORM 1 0
TRANSLATE (0,0,0) (1,1,1) (0,0,0)
SCALE (0.6,0.6,0.6) (0.6,0.6,0.6) (0.6,0.6,0.6)
ROTATE (0,0,0) (50,50,50) (0,0,0)
TRANSLATE (0,0,0) (-0.4,-0.4,-0.4) (0,0,0)

TRANSFORM 1 0
TRANSLATE (0,0,0) (1,1,1) (0,0,0)
SCALE (0.8,0.8,0.8) (0.8,0.8,0.8) (0.8,0.8,0.8)
ROTATE (0,0,0) (150,150,150) (0,0,0)
TRANSLATE (0,0,0) (-0.8,-0.8,-0.8) (0,0,0)

CONDENSATION 1
CONE -1.0 0.5 0.02 0.0 CONE_Y
```

Context-free grammar inference

Inferred grammar for hypertree description DSL

```
NT1 -> #resolution NT2 #iterations #num NT3 NT2 #treedepth #num #branchdepth #num
       #hypervolume NT2 NT2 #condensation #num #cone NT2 #num #coney
NT2 -> #num #num #num NT4
NT3 -> #pointinit
NT3 -> #lineinit #num #num #num #num
NT4 -> #depthcolor #range #bpp #bpp #bpp NT4
NT4 -> epsilon
NT4 -> #name #progname NT4
NT4 -> #scale #lpar #num #comma #num #comma #num #rpar
       #lpar #num #comma #num #comma #num #rpar
       #lpar #num #comma #num #comma #num #rpar NT4
NT4 -> #rotate #lpar #num #comma #num #comma #num #rpar
       #lpar #num #comma #num #comma #num #rpar
       #lpar #num #comma #num #comma #num #rpar NT4
NT4 -> #translate #lpar #num #comma #num #comma #num #rpar
       #lpar #num #comma #num #comma #num #rpar
       #lpar #num #comma #num #comma #num #rpar NT4
NT4 -> #transform #num #num NT4
NT4 -> #shear #lpar #num #comma #num #comma #num #rpar
       #lpar #num #comma #num #comma #num #rpar
       #lpar #num #comma #num #comma #num #rpar #shearxz NT4
NT4 -> #perturb #lpar #num #comma #num #comma #num #comma #num #rpar
       #lpar #num #comma #num #comma #num #comma #num #rpar
       #lpar #num #comma #num #comma #num #comma #num #rpar
       #lpar #num #comma #num #comma #num #comma #num #rpar NT4
```

Context-free grammar inference

- Our approach can be used also for syntax extensions and for DSL embedding
  - To embed domain-specific language (e.g, SQL) into another programming language (GPL or DSL)

Context-free grammar inference

- Initial grammar (ANSI C):

1. **translation unit** ::= **external decl**
2. **translation unit** ::= **translation unit external decl**
3. **external decl** ::= **function definition**
4. **external decl** ::= **decl**
6. **function definition** ::= **declarator decl list compound stat**
9. **decl** ::= **decl specs init declarator list** ;
10. **decl** ::= **decl specs** ;
11. **decl list** ::= **decl**
12. **decl list** ::= **decl list decl**
15. **decl specs** ::= **type spec decl specs**
27. **type spec** ::= int / long / ...
45. **init declarator list** ::= **init declarator**
46. **init declarator list** ::= **init declarator list** , **init declarator**
47. **init declarator** ::= **declarator**
64. **enumerator** ::= id
65. **enumerator** ::= id = **const exp**
67. **declarator** ::= **direct declarator**
68. **direct declarator** ::= id
69. **direct declarator** ::= (**declarator**)
70. **direct declarator** ::= **direct declarator** [**const exp**]
71. **direct declarator** ::= **direct declarator** [ ]
72. **direct declarator** ::= **direct declarator** (**param type list**)
73. **direct declarator** ::= **direct declarator** (**id list**)
74. **direct declarator** ::= **direct declarator** ( )
88. **id list** ::= id
89. **id list** ::= **id list** , id
90. **initializer** ::= **assignment exp**
91. **initializer** ::= **initializer list**
93. **initializer list** ::= **initializer**
94. **initializer list** ::= **initializer list** , **initializer**
110. **stat** ::= **labeled stat** / **exp stat** / **compound stat** / **selection stat**
114. **stat** ::= **iteration stat** / **jump stat**
116. **labeled stat** ::= id : **stat**
117. **labeled stat** ::= case **const exp** : **stat**
118. **labeled stat** ::= default : **stat**
119. **exp stat** ::= **exp** ;
120. **exp stat** ::= ;
121. **compound stat** ::= **decl list stat list**
125. **stat list** ::= **stat**
126. **stat list** ::= **stat list stat**
127. **selection stat** ::= if (**exp**) **stat**
129. **selection stat** ::= switch (**exp**) **stat**
130. **iteration stat** ::= while (**exp**) **stat**
131. **iteration stat** ::= do **stat** while (**exp**) ;
132. **iteration stat** ::= for (**exp** ; **exp** ; **exp**) **stat**
140. **jump stat** ::= goto id ; / continue ; / break ; / return **exp** ;
145. **exp** ::= **assignment exp**
146. **exp** ::= **exp** , **assignment exp**
147. **assignment exp** ::= **conditional exp**
148. **assignment exp** ::= **conditional exp assignment operator assignment exp**
205. **const** ::= int const / char const / float const

Context-free grammar inference

- Initial grammar (ANSI C):

true positive sample

```
int main() {
    char str[][];
    int i;
    printf("Students:");
    for(i = 0; i < str.length; i++) {
        printf(str[i]);
    }
    return 0;
}
```

false negative samples:

```
int main() {
    char str[][] = { SELECT Name FROM
        Students };
    int i;
    printf("Students:");
    for(i = 0; i < str.length; i++) {
        printf(str[i]);
    }
    return 0;
}
```

```
int main() {
    char str[][] = { SELECT Name, Surname
        FROM Students, Professors };
    int i;
    printf("Students and Professors:");
    for(i = 0; i < str.length; i++) {
        printf(str[i]);
    }
    return 0;
}
```

Context-free grammar inference

- Inferred Grammar:

1. **translation unit** ::= **external decl**
2. **translation unit** ::= **translation unit external decl**
3. **external decl** ::= **function definition**
4. **external decl** ::= **decl**
6. **function definition** ::= **declarator decl list compound stat**
9. **decl** ::= **decl specs init declarator list** ;
10. **decl** ::= **decl specs** ;
11. **decl list** ::= **decl**
12. **decl list** ::= **decl list decl**
15. **decl specs** ::= **type spec decl specs**
27. **type spec** ::= int / long / ...
45. **init declarator list** ::= **init declarator**
46. **init declarator list** ::= **init declarator list** , **init declarator**
47. **init declarator** ::= **declarator**
64. **enumerator** ::= id
65. **enumerator** ::= id = **const exp**
67. **declarator** ::= **direct declarator** ***NT1***
68. **direct declarator** ::= id
69. **direct declarator** ::= (**declarator**)
70. **direct declarator** ::= **direct declarator** [**const exp**]
71. **direct declarator** ::= **direct declarator** [ ]
72. **direct declarator** ::= **direct declarator** (**param type list**)
73. **direct declarator** ::= **direct declarator** (**id list**)
74. **direct declarator** ::= **direct declarator** ( )
88. **id list** ::= id
89. **id list** ::= **id list** , id
90. **initializer** ::= **assignment exp**
91. **initializer** ::= **initializer list**
93. **initializer list** ::= **initializer**
94. **initializer list** ::= **initializer list** , **initializer**
110. **stat** ::= **labeled stat** / **exp stat** / **compound stat** / **selection stat**
114. **stat** ::= **iteration stat** / **jump stat**
116. **labeled stat** ::= id : **stat**
117. **labeled stat** ::= case **const exp** : **stat**
118. **labeled stat** ::= default : **stat**
119. **exp stat** ::= **exp** ;
120. **exp stat** ::= ;
121. **compound stat** ::= **decl list stat list**
125. **stat list** ::= **stat**
126. **stat list** ::= **stat list stat**
127. **selection stat** ::= if (**exp**) **stat**
129. **selection stat** ::= switch (**exp**) **stat**
130. **iteration stat** ::= while (**exp**) **stat**
131. **iteration stat** ::= do **stat** while (**exp**) ;
132. **iteration stat** ::= for (**exp** ; **exp** ; **exp**) **stat**
140. **jump stat** ::= goto id ; / continue ; / break ; / return **exp** ;
145. **exp** ::= **assignment exp**
146. **exp** ::= **exp** , **assignment exp**
147. **assignment exp** ::= **conditional exp**
148. **assignment exp** ::= **conditional exp assignment operator assignment exp**
205. **const** ::= int const / char const / float const
208. ***NT1*** ::= = SELECT id **NT2** FROM id **NT2** / ϵ
210. ***NT2*** ::= , id **NT2** / ϵ

Metamodel inference

- As a model conforms to a metamodel in a similar manner to how a program conforms to a grammar, the metamodel inference can be defined as follows.
- The set of all models that conform to a given metamodel MM will be called the language of the metamodel and denoted L(MM). Given a model instance m and a metamodel MM we can tell whether m conforms to MM ($m \in L(MM)$).

Metamodel inference

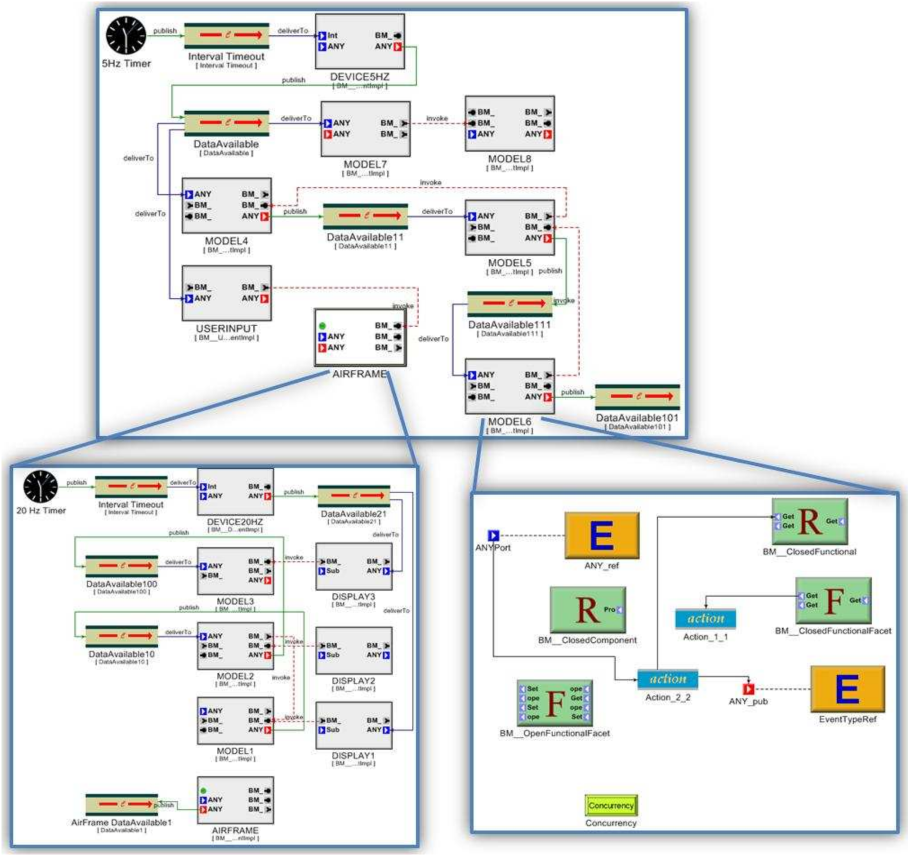
- A set of positive samples is denoted with S^+. Conversely, a negative sample belongs to $\overline{L}(MM)$, which denotes a set of all models that do not conform to metamodel MM. A set of negative samples is denoted with S^-.
- A set of positive samples S^+ of a metamodel MM is structurally complete if each metamodel element appears in at least one model in S^+.

Metamodel inference

- Given a set of positive samples S^+ and set of negative samples S^-, which might be also empty, the task of metamodel inference is to find at least one metamodel MM such that $S^+ \subseteq L(MM)$ and $S^- \subseteq \overline{L}(MM)$.

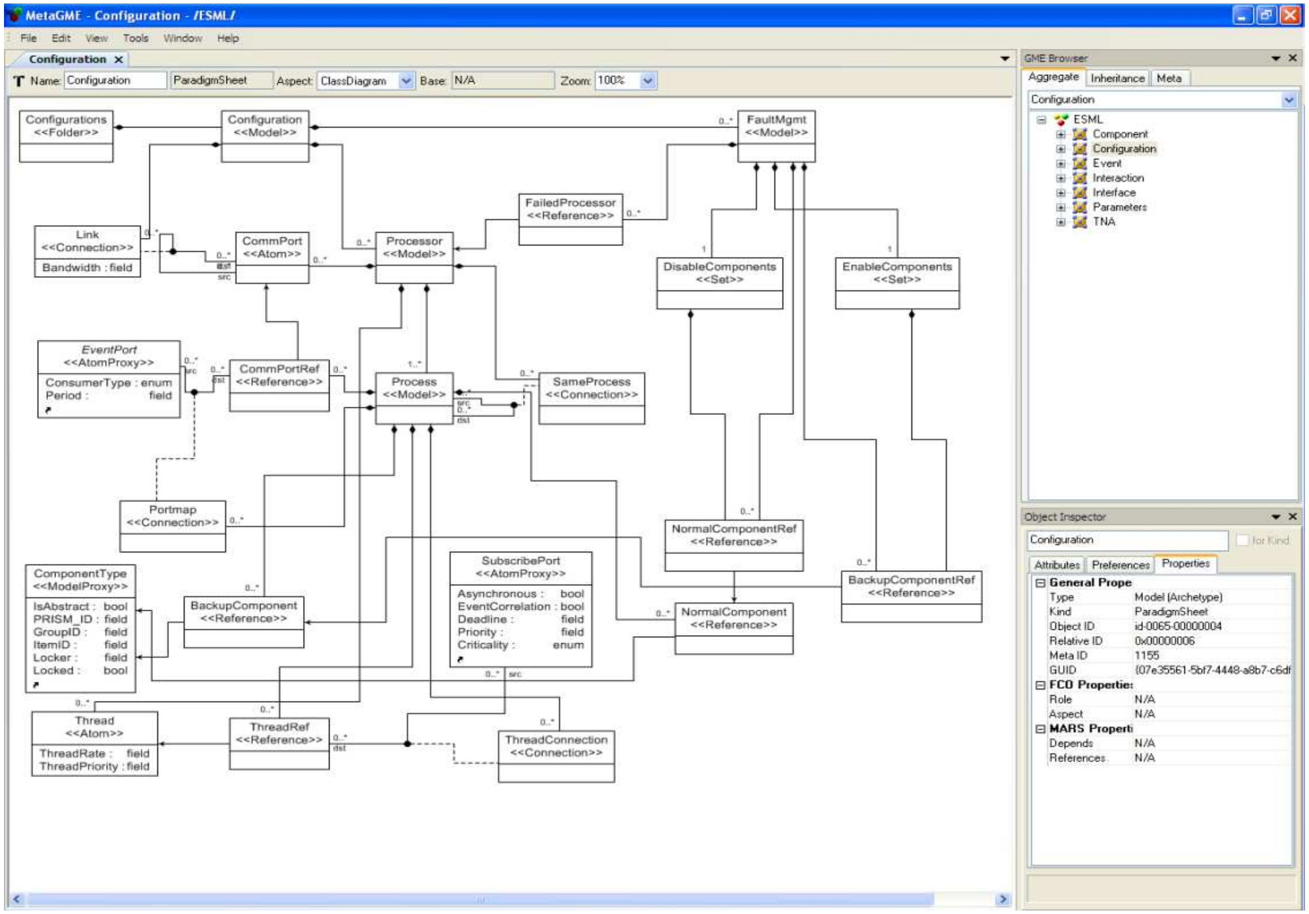
Metamodel inference

- ESML (Embedded System Modeling Language)



Metamodel inference

- Original ESML metamodel - Configuration viewpoint



Name: Configuration
ParadigmSheet
Aspect: ClassDiagram
Base: N/A
Zoom: 100%

Configuration

General Prope

Type	Model (Archetype)
Kind	ParadigmSheet
Object ID	id-0065-00000004
Relative ID	0x00000006
Meta ID	1155
GUID	{07e35561-5bf7-4448-a8b7-c6df

FCO Propertie

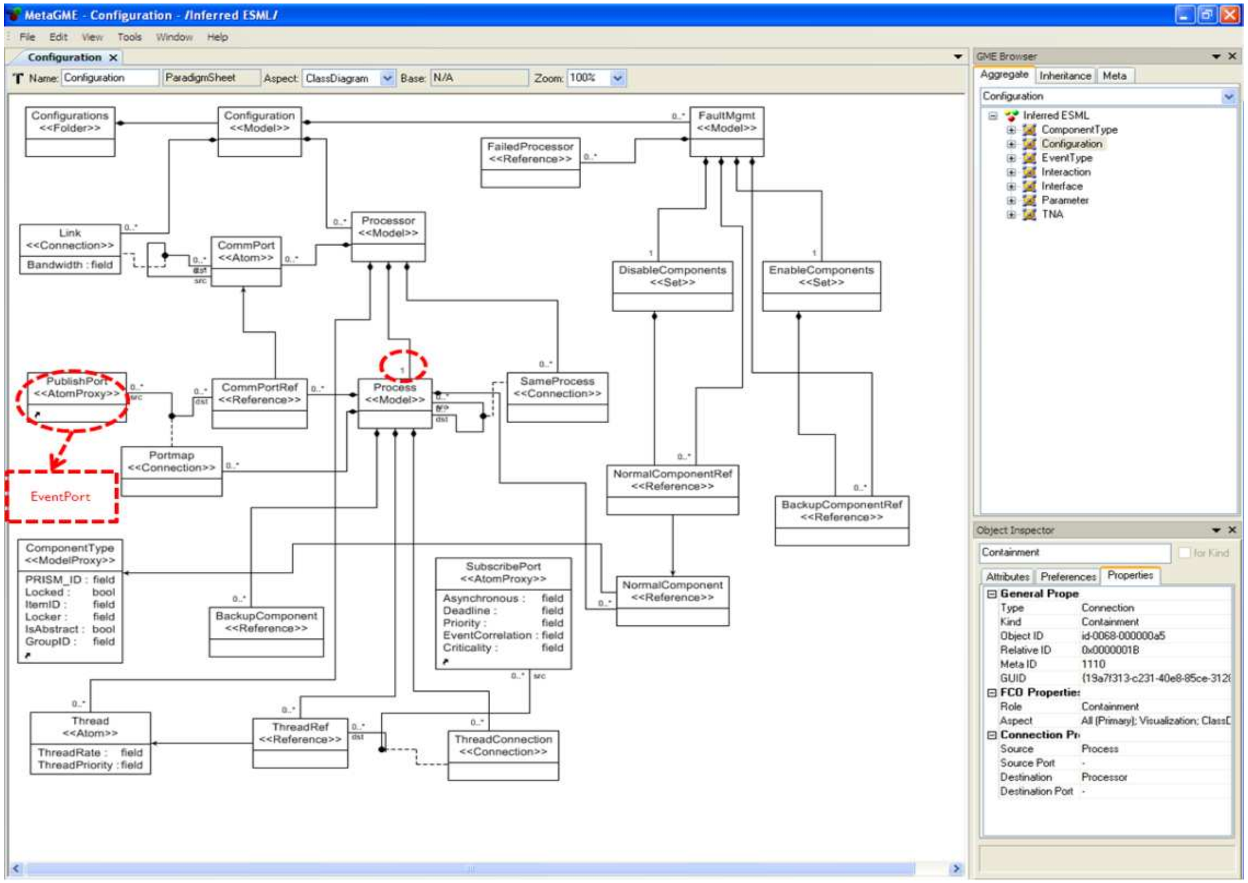
Role	N/A
Aspect	N/A

MARS Properti

Depends	N/A
References	N/A

Metamodel inference

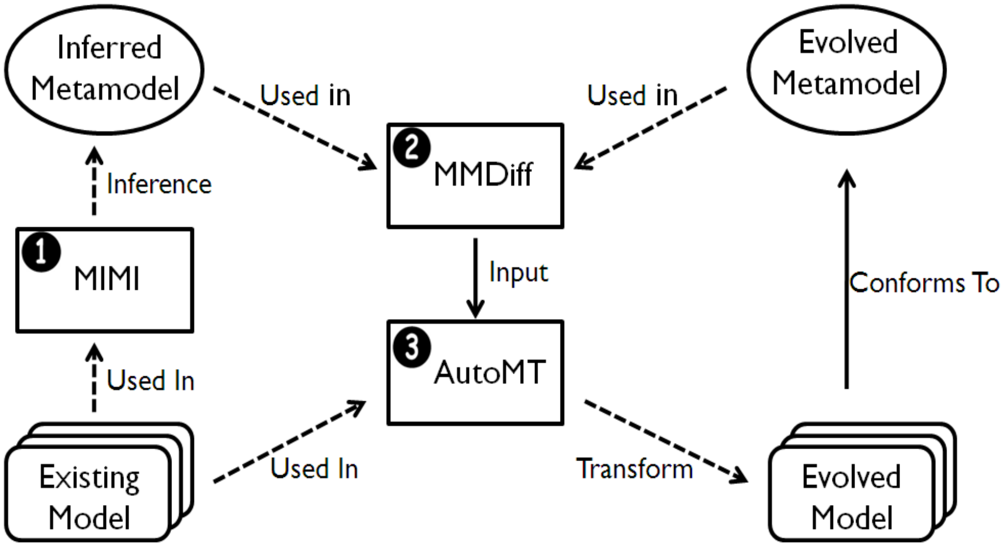
- Inferred ESML metamodel - Configuration viewpoint



General Prope	
Type	Connection
Kind	Containment
Object ID	id-0068-000000a5
Relative ID	0x0000001B
Meta ID	1110
GUID	{19a7f313-c231-40e8-85ce-312
FCO Propertie	
Role	Containment
Aspect	All (Primary); Visualization; Class
Connection Pr	
Source	Process
Source Port	-
Destination	Processor
Destination Port	-

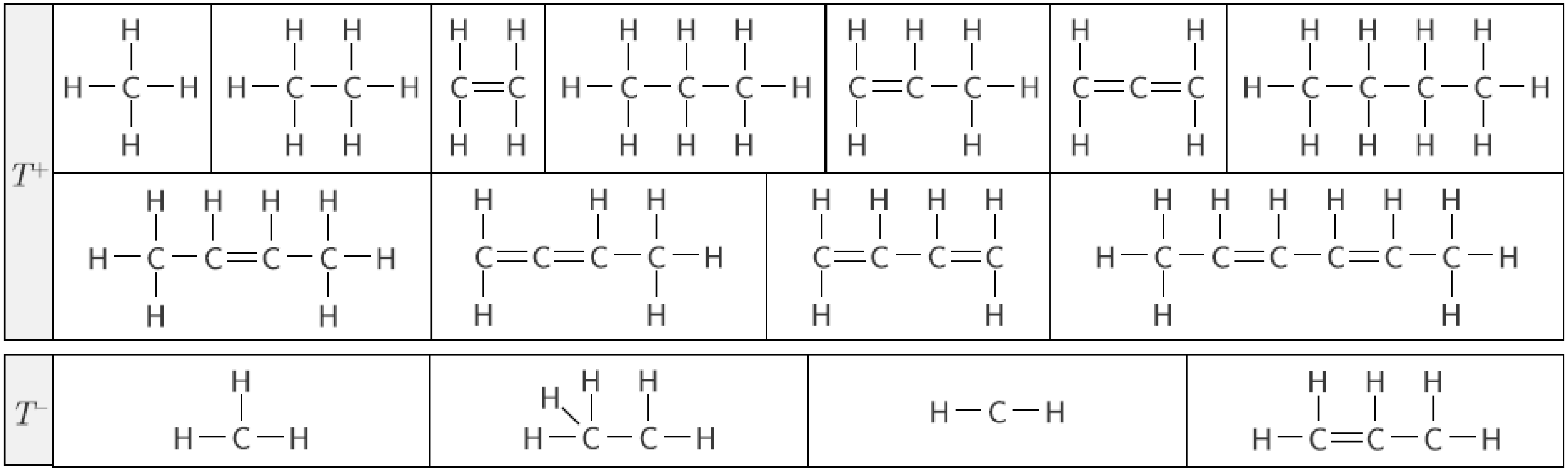
Metamodel inference

- Our approach to model evolution using metamodel inference



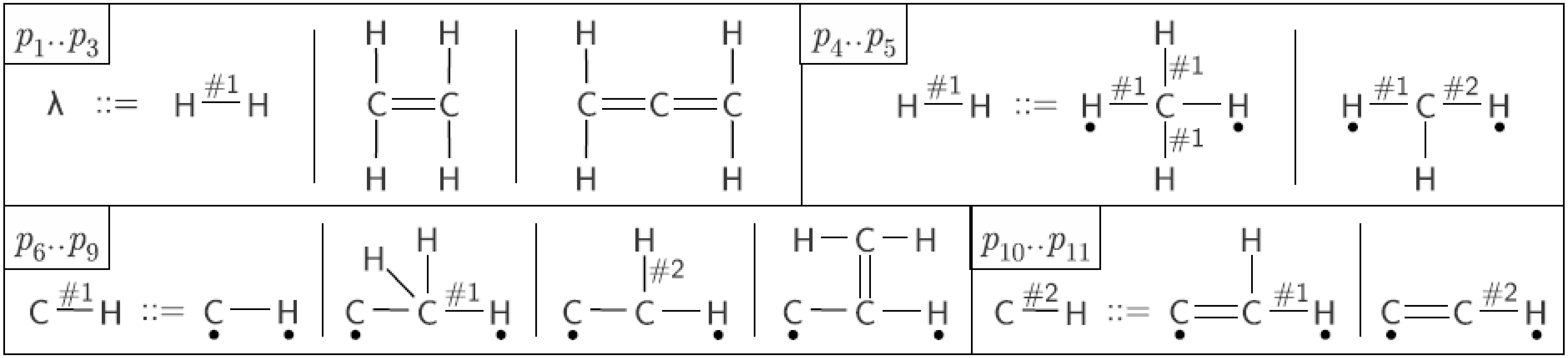
Graph grammar inference

Positive and negative samples for hydrocarbons with single and double bonds



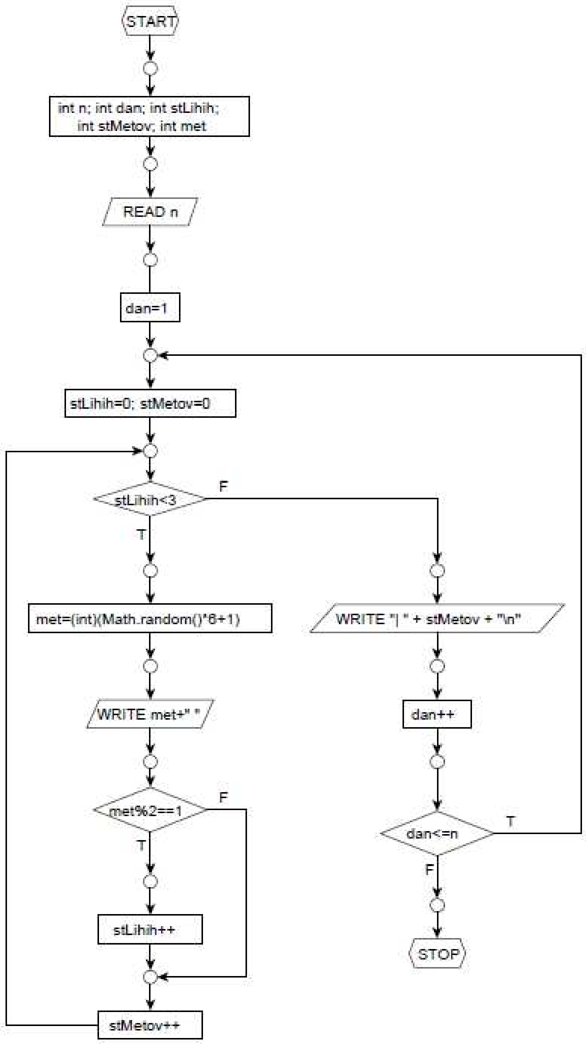
Graph grammar inference

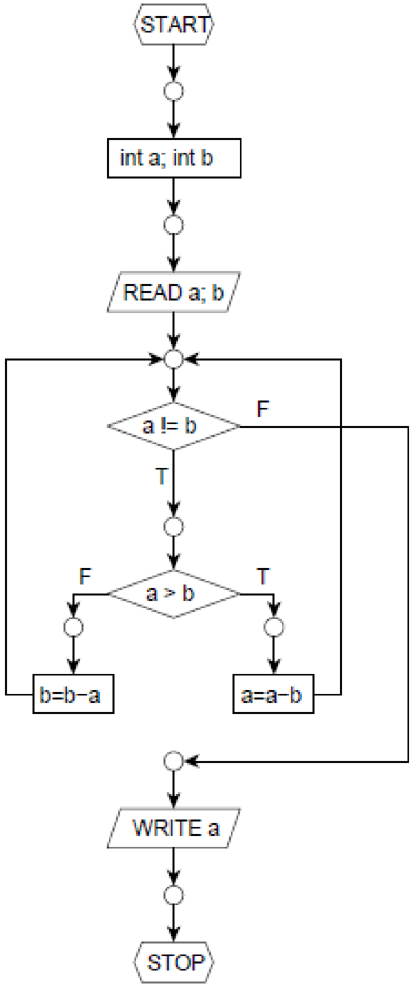
Inferred graph grammar



Graph grammar inference

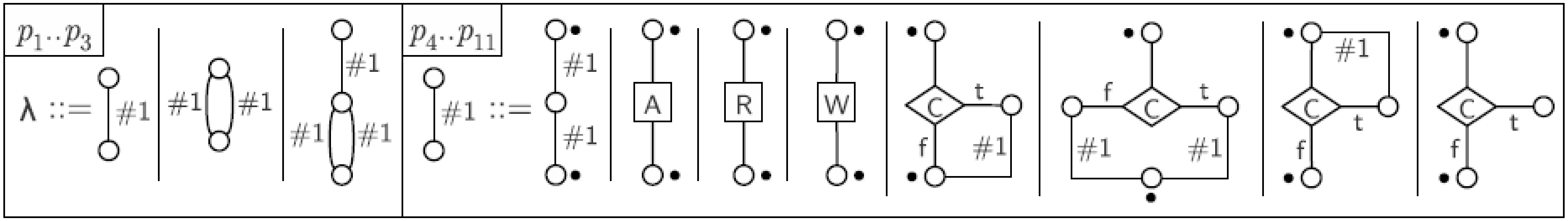
Positive samples
for flowcharts





Graph grammar inference

Inferred graph grammar



Semantic inference

$L(G) = \{a^n b^n c^n | n \geq 1\}$

$S \rightarrow A\ B\ C$
{S.ok = (A.val == B.val) && (B.val == C.val);}
$A \rightarrow a\ A$
{A[0].val = 1 + A[1],val;}
$A \rightarrow a$
{A.val=1;}
$B \rightarrow b\ B$
{B[0].val=1+B[1].val;}
$B \rightarrow b$
{B.val=1;}
$C \rightarrow c\ C$
{C[0].val=1+C[1].val;}
$C \rightarrow c$
{C.val=1;}

Set of positive programs with associated meanings:

(abc, true)
(aabbcc, true)
(aaabbbccc, true)
(aabc, false)
(abcc, false)
(abbbc, false)
(abbccc, false)

Conclusion

Yes, I will used in my current project on business process mining.

Hope that I convinced you that grammatical inference is interesting and useful.

Conclusion

1. HRNČIČ, Dejan, MERNIK, Marjan, BRYANT, Barrett Richard, JAVED, Faizan. A memetic grammar inference algorithm for language learning. *Applied Soft Computing*, 2012, vol. 12, iss. 3, pp. 1006-1020.
2. HRNČIČ, Dejan, MERNIK, Marjan, BRYANT, Barrett Richard. Improving grammar inference by a memetic algorithm. *IEEE Transactions on Systems, Man, and Cybernetics - Part* C, 2012, vol. 42, no. 5, pp. 692-703.
3. FÜRST, Luka, MERNIK, Marjan, MAHNIČ, Viljan. Graph grammar induction as a parser-controlled heuristic search process. *AGTIVE'12,* pp. 121-136.
4. HRNČIČ, Dejan, MERNIK, Marjan, BRYANT, Barrett Richard. Embedding DSLS into GPLS: A Grammatical Inference Approach. *Information Technology and Control* , 2011, vol. 40, no. 4, pp. 307-315.
5. JAVED, Faizan, MERNIK, Marjan, GRAY, Jeffrey G., BRYANT, Barrett Richard. MARS: A Metamodel Recovery System Using Grammar Inference. *Information and Software Technology*, 2008, vol. 50, iss. 9-10, pp. 948-968.
6. FÜRST, Luka, MERNIK, Marjan, MAHNIČ, Viljan. Converting metamodels to graph grammars: doing without advanced graph grammar features. *Software and System Modeling (SoSym)*, 2013, Article in Press.

Conclusion

More information at:
http://www.cis.uab.edu/softcom/GrammarInference/

Sent comments/questions to:
marjan.mernik@uni-mb.si; mernik@cis.uab.edu

Javna agencija
za raziskovalno dejavnost
Republike Slovenije

This work was supported in part by NSF award CCF-0811630
and by ARRS bilateral project BI-US/11-12-031