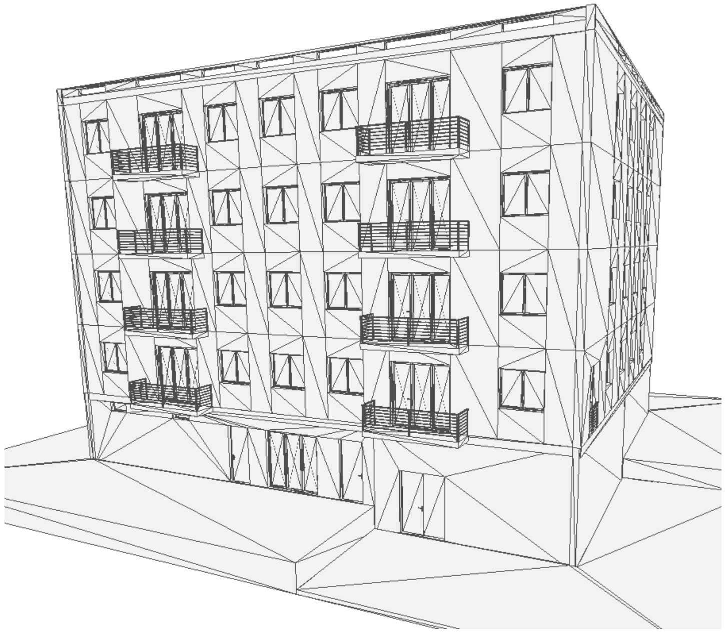
Semantizing Complex 3D Scenes using Constrained Attribute Grammars

A. Boulch, S. Houllier,
R. Marlet and O. Tournaire

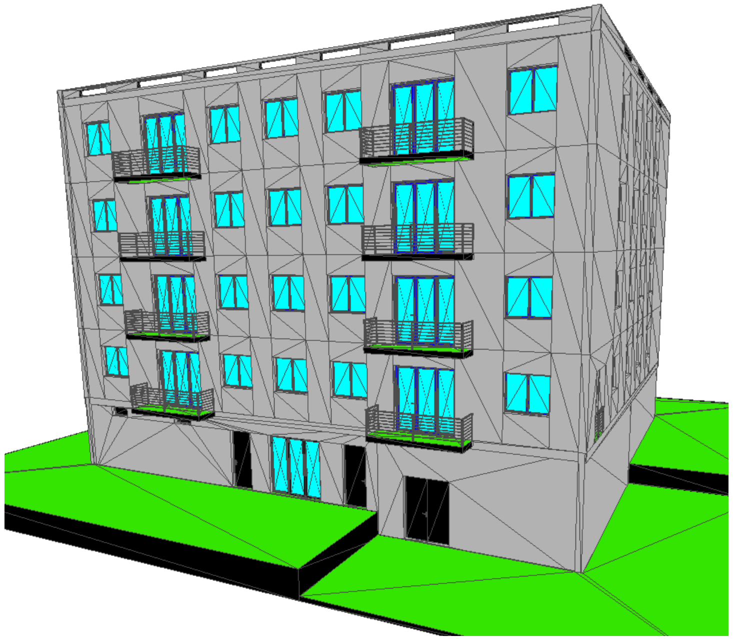
slight variant of the material
presented at SGP 2013

Motivation

Semantizing complex objects in 3D scenes



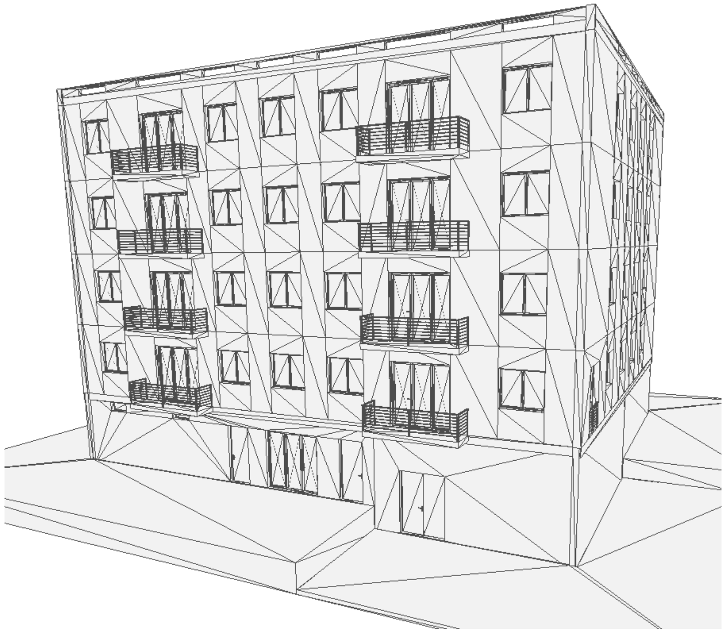
Bare geometry



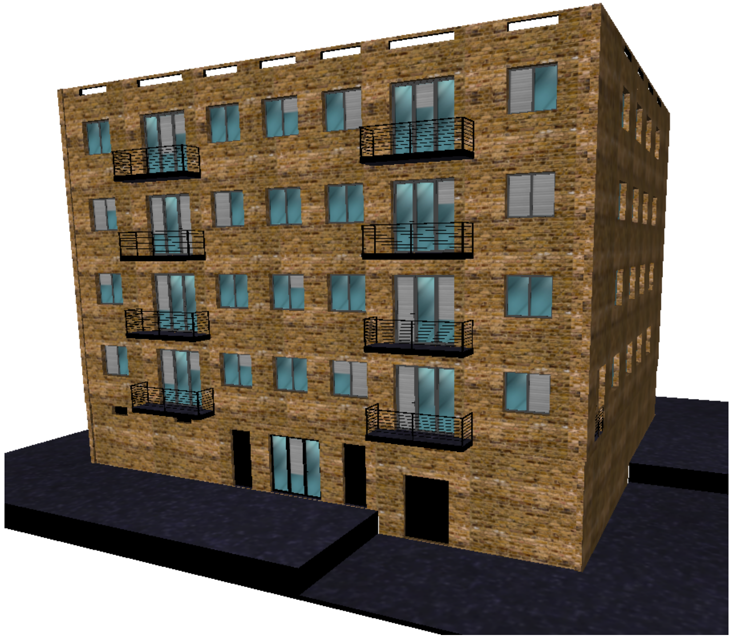
Semantized geometry

Motivation

Semantizing complex objects in 3D scenes



Bare geometry

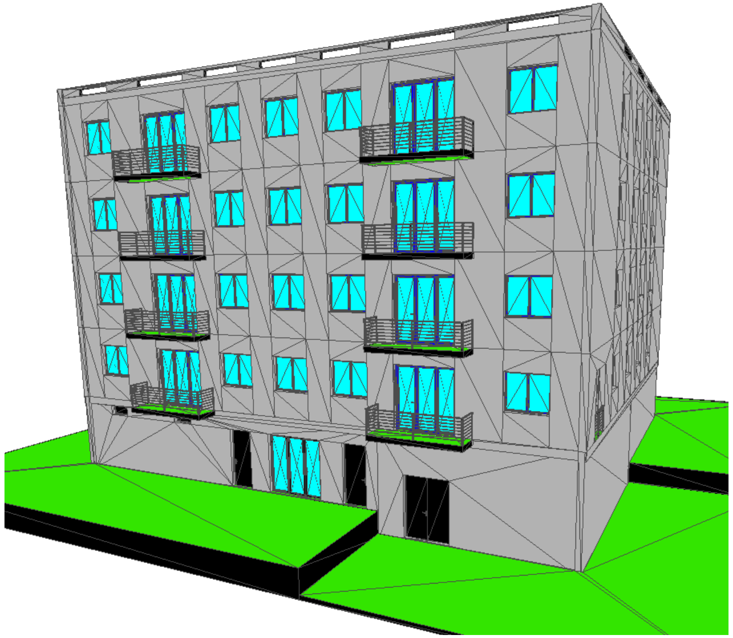


Semantized geometry (rendered)

Motivation

Semantizing complex objects in 3D scenes

- Building industry
  - for the renovation market
    - point cloud → building model

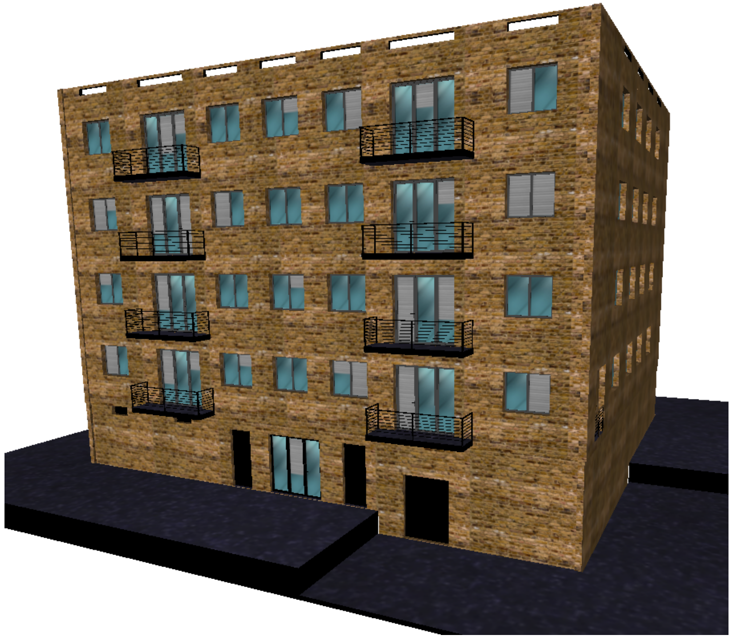


Semantized geometry

Motivation

Semantizing complex objects in 3D scenes

- Building industry
  - for the renovation market
    - point cloud → building model
  - for architects
    - building sketch → rendering

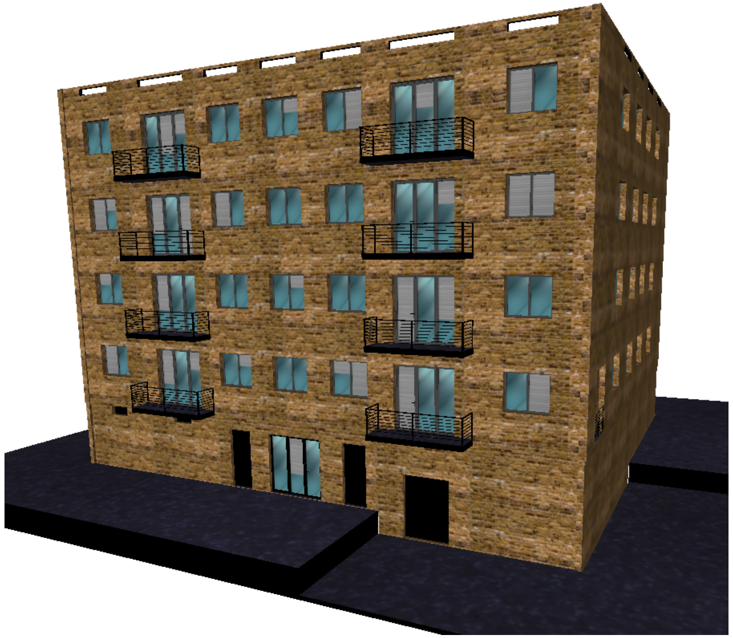


Semantized geometry (rendered)

Motivation

Semantizing complex objects in 3D scenes

- Building industry
  - for the renovation market
    - point cloud → building model
  - for architects
    - building sketch → rendering
- Game industry
  - for graphic designers
    - basic level design → rendering

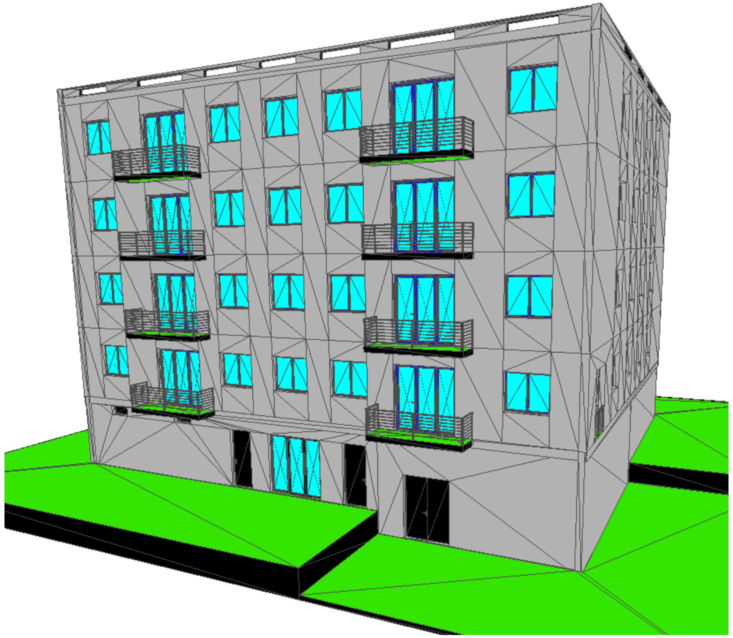


Semantized geometry (rendered)

Motivation

Semantizing complex objects in 3D scenes

- Building industry
  - for the renovation market
    - point cloud → building model
  - for architects
    - building sketch → rendering
- Game industry
  - for graphic designers
    - basic level design → rendering
- Object mining in shape databases
  - for semantic queries
    - 3D object → semantic labeling



Semantized geometry

Outline

Constrained attribute grammars

Scene interpretation

Bottom-up parsing

Experiments

Outline

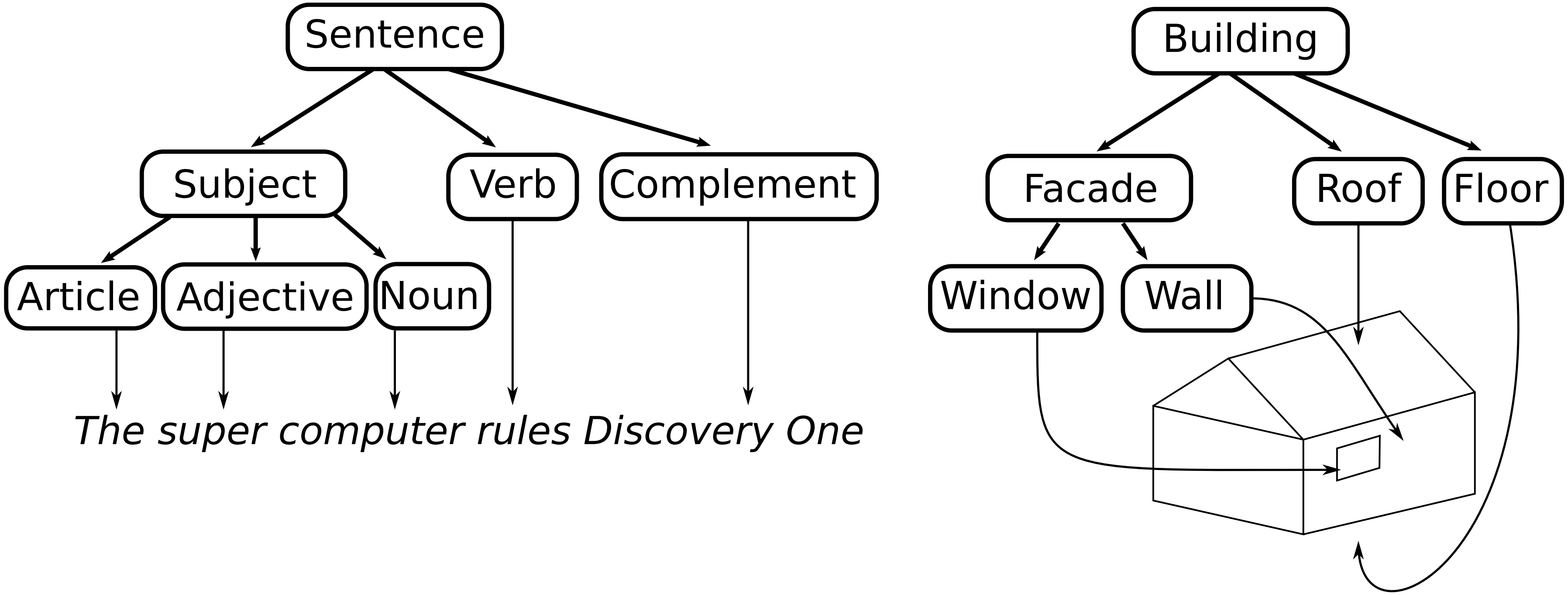
Constrained attribute grammars

Scene interpretation

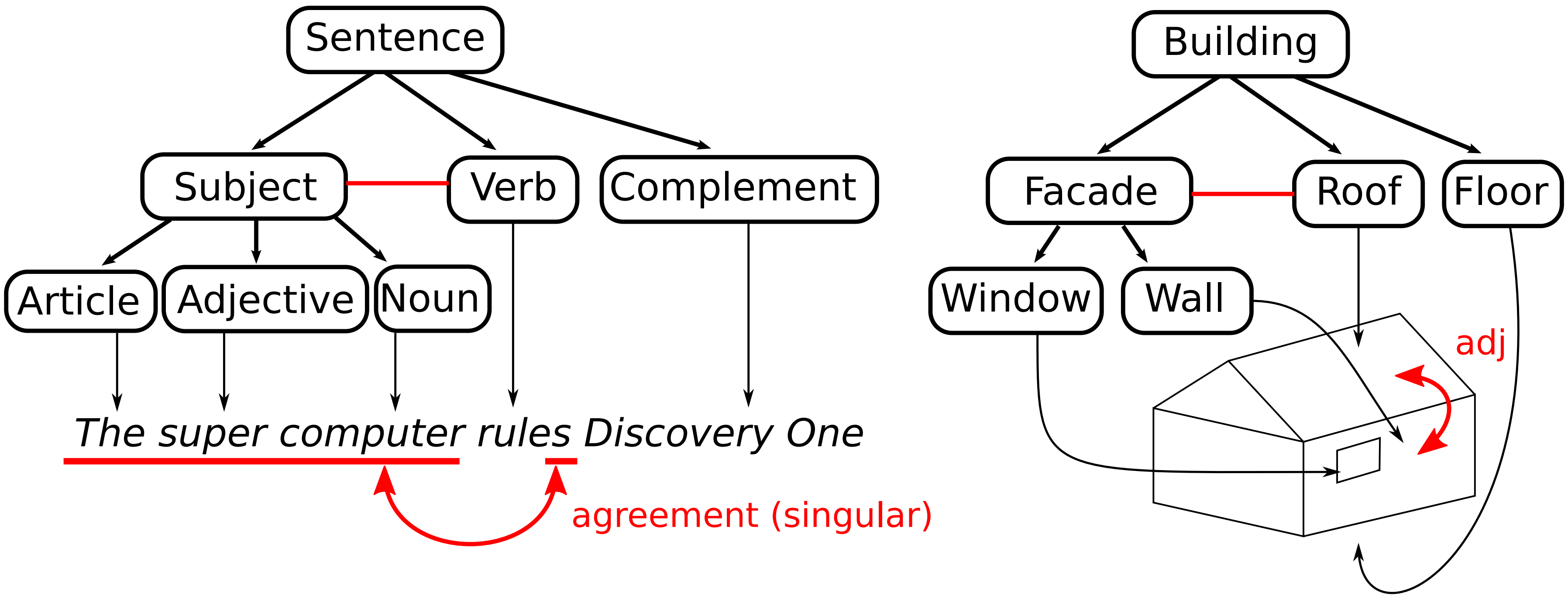
Bottom-up parsing

Experiments

Grammars to express hierarchical decomposition



Complex, non-hierarchic relations between components



Constrained attribute grammars

$G = (N, T, P, S)$

- N: nonterminals ($\leftrightarrow$ complex forms) *e.g., window, wall*
- T: terminals ($\leftrightarrow$ geometric primitives) *e.g., polygon, cylinder*
- P: production rules ($\leftrightarrow$ hierarchical decomposition and constraints)
- S: start symbols ($\leftrightarrow$ root shapes) *e.g., building*

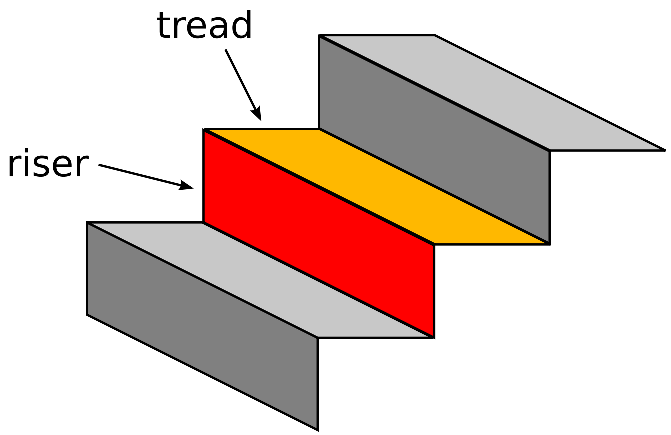
Basic rules

Decomposition of a complex object y of type Y into its constituents x_i of type X_i:

$$Y\ y \longrightarrow X_1\ x_1, \ldots, X_n\ x_n$$

Example:

$$\mathit{step}\ s \longrightarrow \mathit{riser}\ r, \mathit{tread}\ t$$



Rule application:

- top-down view:
  y decomposes into $x_1, \ldots, x_n$
- bottom-up view:
  given some $x_1, \ldots, x_n$, create a new object y

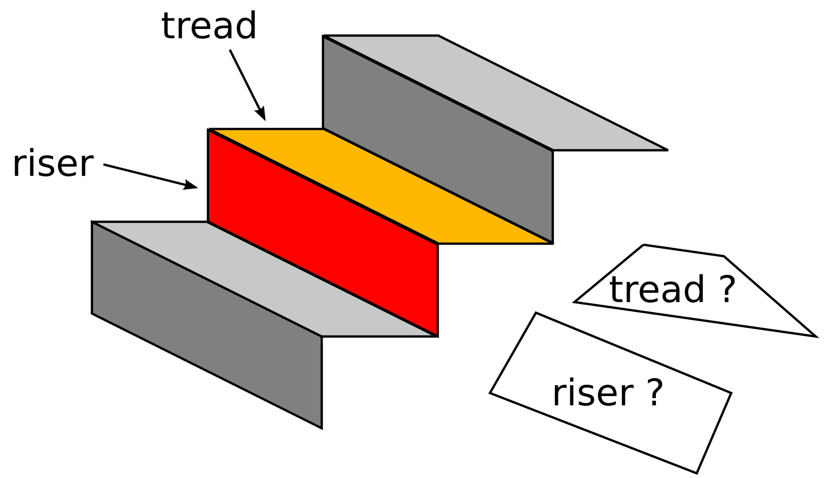
Basic rules

Decomposition of a complex object y of type Y into its constituents x_i of type X_i:

$$Y\ y \longrightarrow X_1\ x_1, \ldots, X_n\ x_n$$

Example:

$$\textit{step}\ s \longrightarrow \textit{riser}\ r, \textit{tread}\ t$$



Rule application:

- top-down view:
  y decomposes into $x_1, \ldots, x_n$
- bottom-up view:
  given some $x_1, \ldots, x_n$, create a new object y

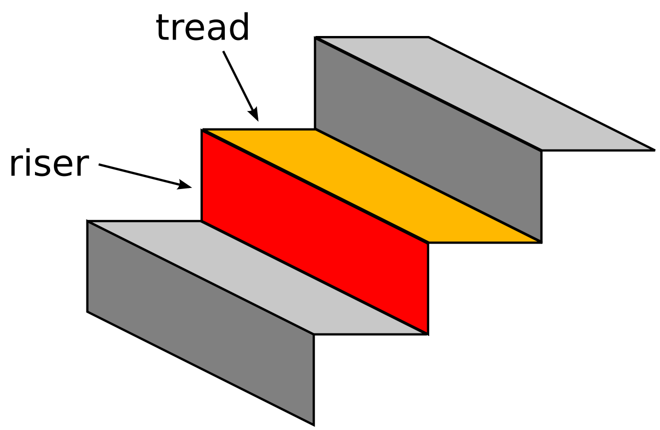
Constraints

Conditional rule application (conjonction of predicates):

$$Y\ y \longrightarrow X_1\ x_1, X_2\ x_2, \ldots\ \langle\, cstr_1(x_1), cstr_2(x_1, x_2), \ldots \rangle$$

Example:

$$\begin{array}{rcl} riser\ r & \longrightarrow & polygon\ p\ \langle\, vertical(p)\, \rangle \\ tread\ t & \longrightarrow & polygon\ p\ \langle\, horizontal(p)\, \rangle \\ step\ s & \longrightarrow & riser\ r, tread\ t \\ & & \langle\, edgeAdj(r,t), above(t,r)\, \rangle \end{array}$$



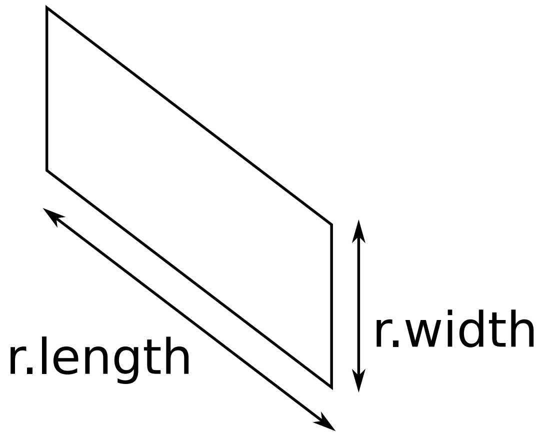
Attributes

Features attached to each grammar element:

- at creation time (primitives)
- at rule application (synthetized attributes)

Examples:

- length, width
- bounding box
- ...



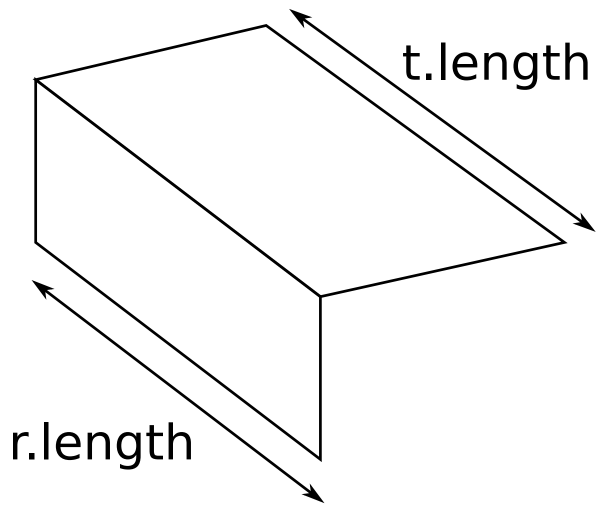
Predicates

Predicates on grammar elements:

- adj, edgeAdj
- orthogonal, parallel
- vertical, horizontal
- ...

Predicates on attributes:

- $\geq, >, \leq, \ldots$
- $=, \neq$
- ...



Example: $r.length == t.length$

Collections of similar elements

Grouping elements via recursion (Y as set of Xs):

$$\begin{aligned} Y\ y &\longrightarrow X\ x \\ Y\ y &\longrightarrow X\ x, Y\ y_2 \end{aligned}$$

Grouping elements via specific collection operators:

$$Y\ \ y \longrightarrow \mathit{coll}(X)\ xs$$

Useful operators: maximal collections

- maxconn: maximal set of connected components
- maxseq: maximal sequence
- ...

Example:

$$\mathit{stairway}\ \ \mathit{sw} \longrightarrow \mathrm{maxseq}(\mathit{step}, \mathit{adjEdge})\ \ \mathit{ss}$$

Collections of similar elements

Grouping elements via recursion (Y as set of Xs):

$$Y\ y \longrightarrow X\ x$$
$$Y\ y \longrightarrow X\ x, Y\ y_2$$

Grouping elements via specific collection operators:

$$Y\ y \longrightarrow coll(X)\ xs$$

Useful operators: maximal collections

- maxconn: maximal set of connected components
- maxseq: maximal sequence
- ... [see complete stairway grammar]

Example:

$$stairway\ sw \longrightarrow \text{maxseq}(step, adjEdge)\ ss$$

Outline

Constrained attribute grammars

Scene interpretation

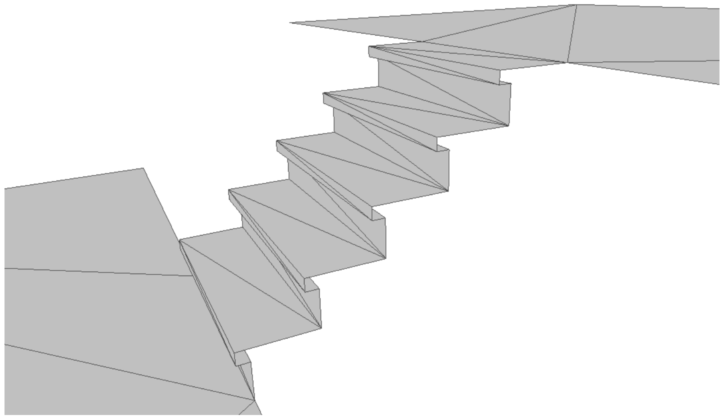
Bottom-up parsing

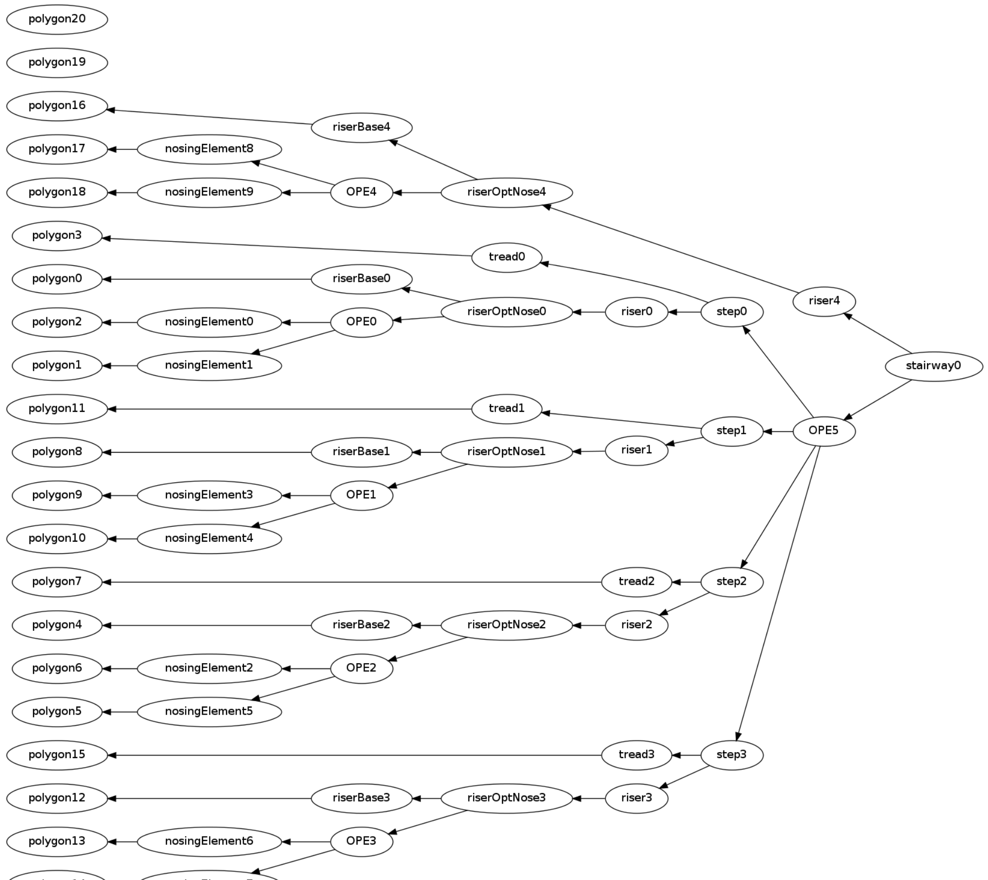
Experiments

Scene interpretation: parse tree

Tree representation of a grammatical analysis of the scene:

- leaves: terminals representing primitives
- non-leaf nodes: instanciations of grammar rules

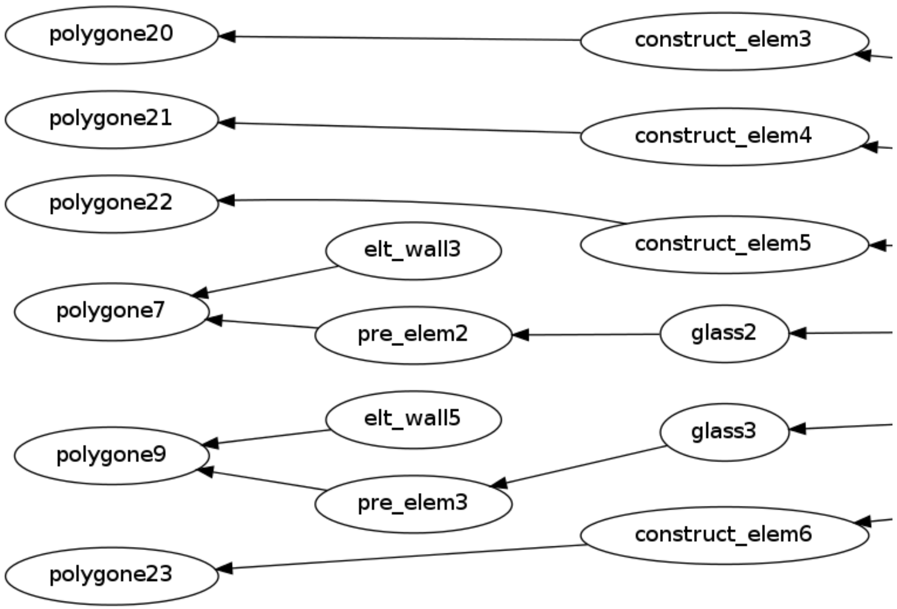




Scene interpretations: parse forest

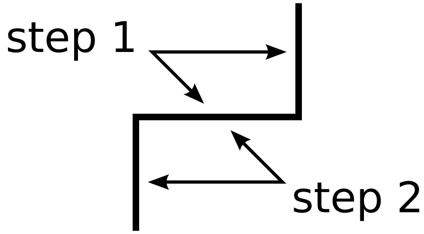
Set of parse trees
with systematic sharing (DAG)

Compact representation
of all possible interpretations



Ambiguity and the exclusivity constraint

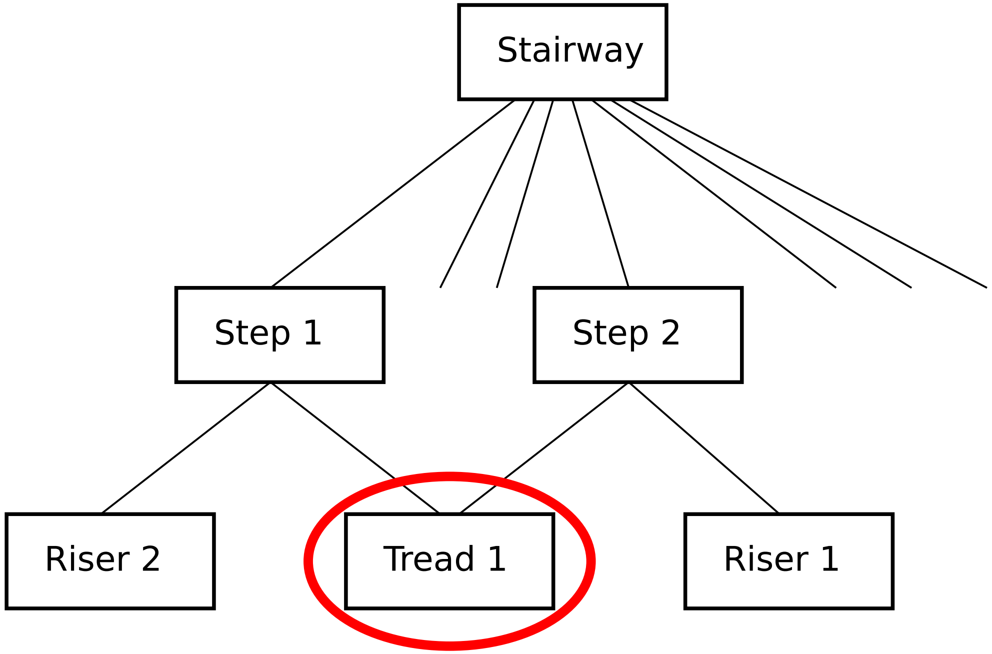
Example (assuming no height ordering constraint):



$$\textit{step } s \longrightarrow \textit{riser } r, \textit{tread } t \,\langle\, \textit{edgeAdj}(r,t) \,\rangle$$

Exclusivity constraint:

at most 1 occurrence of a grammar element per interpretation



Outline

Constrained attribute grammars

Scene interpretation

Bottom-up parsing

Experiments

Parse forest computation

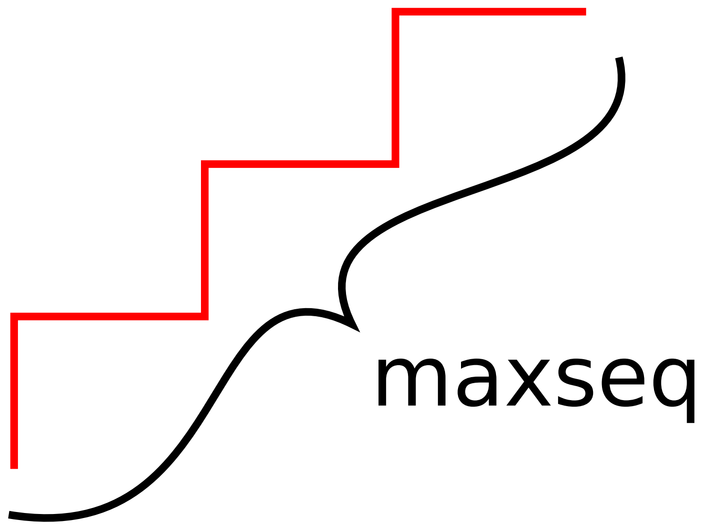
Bottom-up parsing:

construction of the parse forest
from leaves (terminals) to roots (start symbols)

- create one terminal for each primitives
- iteratively create new grammar elements from existing ones
  - e.g., given grammar rule $\textit{step}\ s \longrightarrow \textit{riser}\ r, \textit{tread}\ t$
    given existing instances $\textit{riser}\ r_{23}, \textit{tread}\ t_{18}$
    create new instance $\textit{step}\ s_5$
- merge identical trees on the fly
- stop iterating when no rule applies

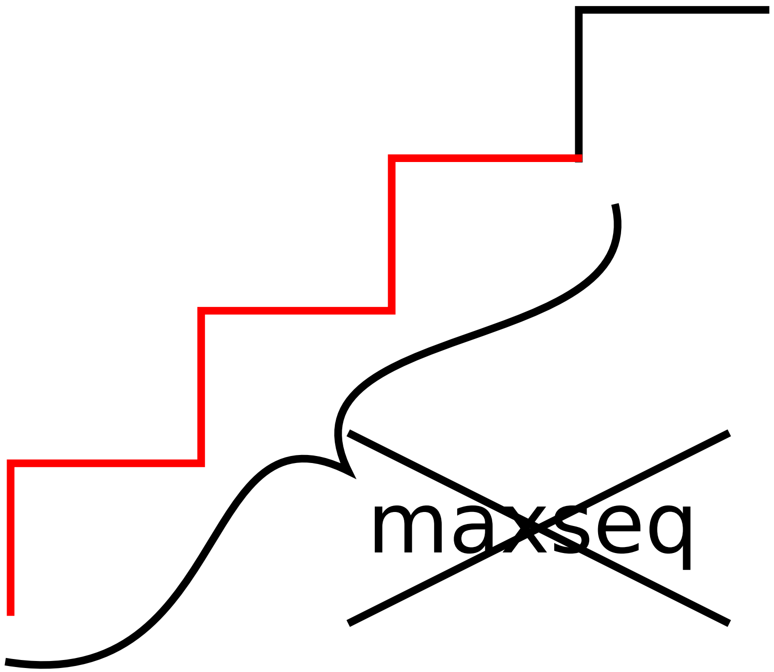
Rule application order

- Simple rules: use any order
- Maximal operators: wait for all subelements to ensure maximality



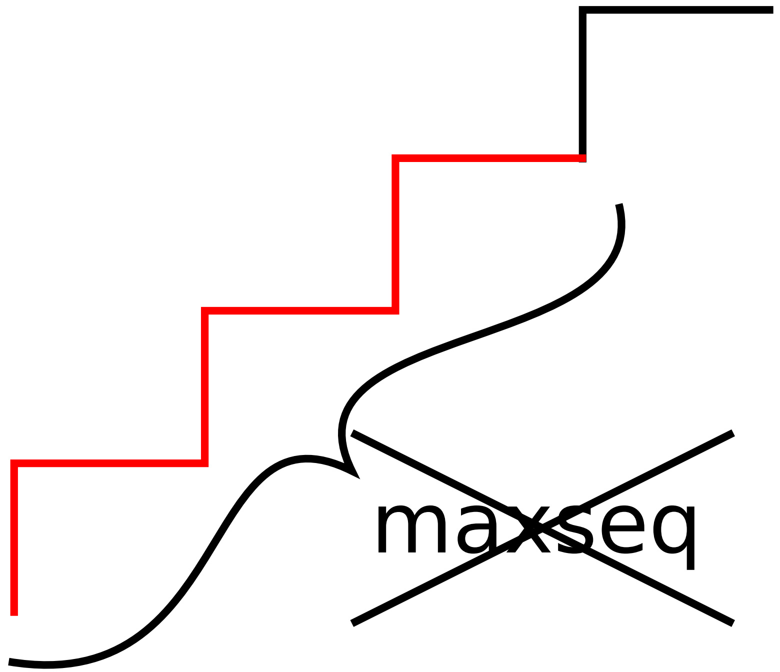
Rule application order

- Simple rules: use any order
- Maximal operators: wait for all subelements to ensure maximality



Rule application order

- Simple rules: use any order
- Maximal operators: wait for all subelements to ensure maximality
  - reverse topological sort of nonterminal dependency graph



Mastering the combinatorial explosion

Usual drawback of bottom-up analysis: combinatorial explosion

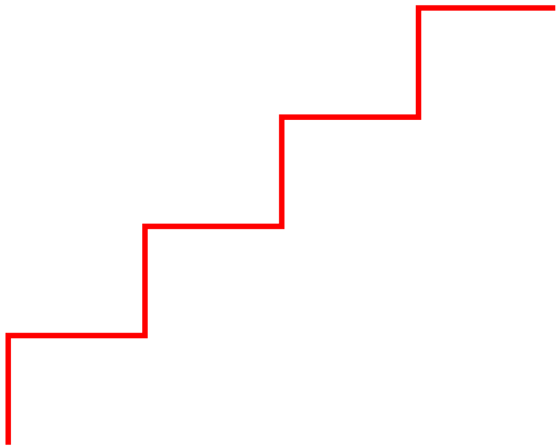
- all trees
- all sets, all sequences, ...
- all combinations

Our solution:

- tree sharing: construction of a parse forest (exp. $\rightarrow$ lin.)
- maximal operators, with efficient implementation ($>$ exp. $\rightarrow$ polyn.)
- constraint propagation: predicate ordering $\Rightarrow$ early pruning

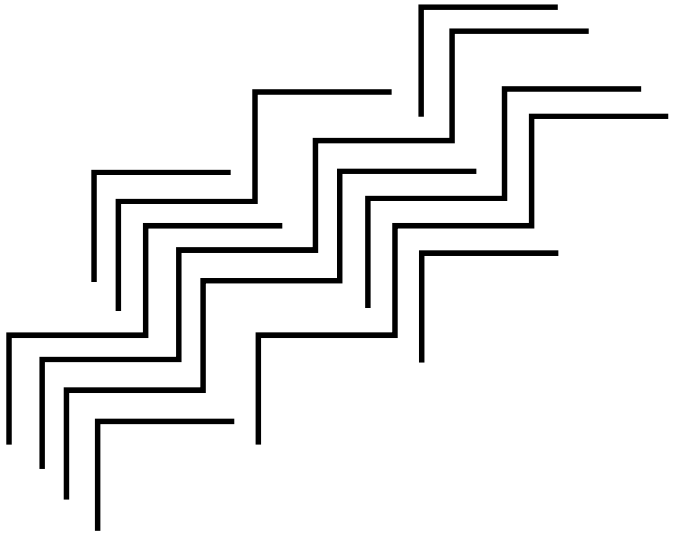
Maximal operators

maxseq



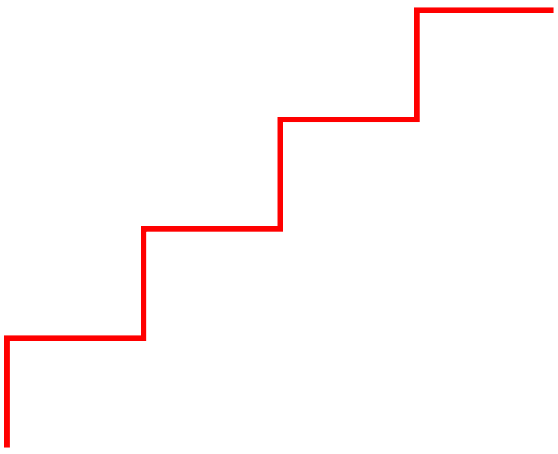
vs

allseq



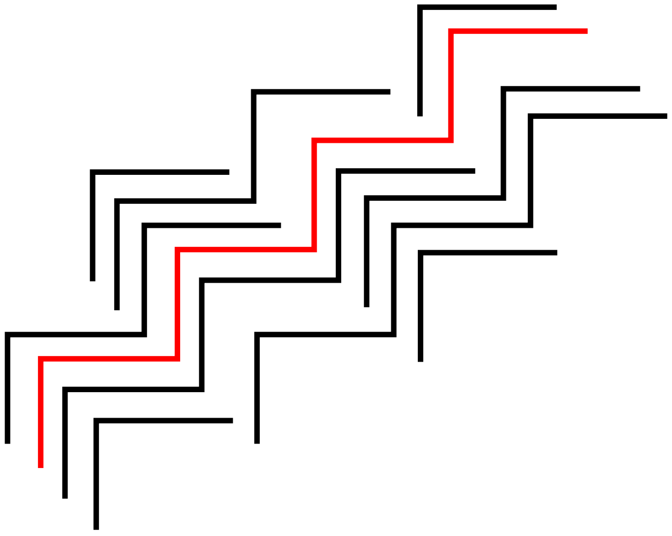
Maximal operators

maxseq



vs

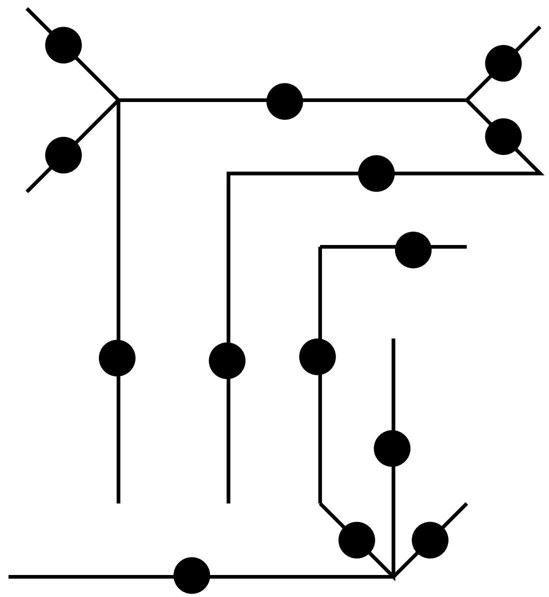
max(allseq)



Constraint propagation as predicate ordering

A simple 2D example

$$pair\ p \longrightarrow seg\ s_1, seg\ s_2,\ \langle\, orthogonal(s_1, s_2), adj(s_1, s_2), vertical(s_1)\,\rangle$$



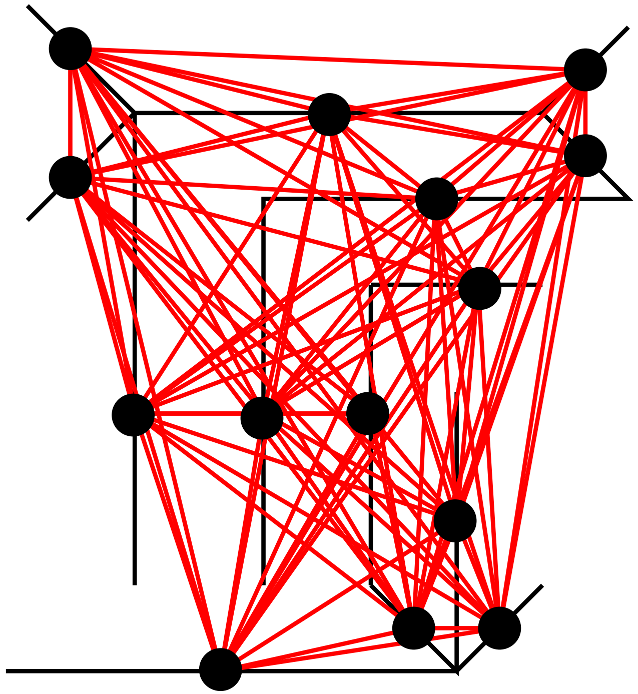
Constraint propagation as predicate ordering

Example of poor ordering

$pair\ p \longrightarrow seg\ s_1, seg\ s_2,\ \langle\, \underline{orthogonal(s_1, s_2)}, adj(s_1, s_2), vertical(s_1)\, \rangle$

1. orthogonality

Complexity:
$O(\#seg^2)$



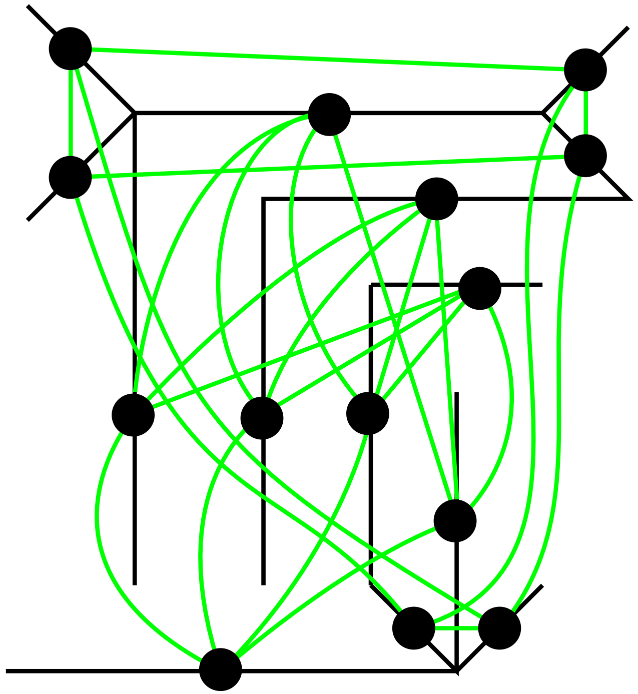
constraints to test

Constraint propagation as predicate ordering

Example of poor ordering

$pair\ p \longrightarrow seg\ s_1, seg\ s_2,\ \langle\ \underline{orthogonal(s_1, s_2)}, adj(s_1, s_2), vertical(s_1)\ \rangle$

1. orthogonality



satisfied constraints

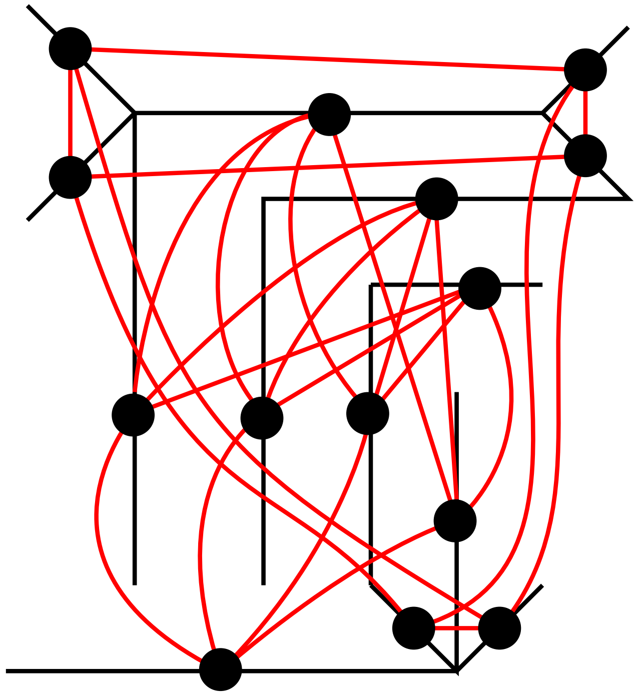
Complexity:
$O(\#seg^2)$

Constraint propagation as predicate ordering

Example of poor ordering

$$pair\ p \longrightarrow seg\ s_1, seg\ s_2,\ \langle\, \underline{orthogonal(s_1, s_2)}, \underline{adj(s_1, s_2)}, vertical(s_1)\,\rangle$$

1. orthogonality
2. adjacency



constraints to test

Complexity:

$$O(\#seg^2 + \#seg \times maxDeg) = O(\#seg^2)$$

Constraint propagation as predicate ordering

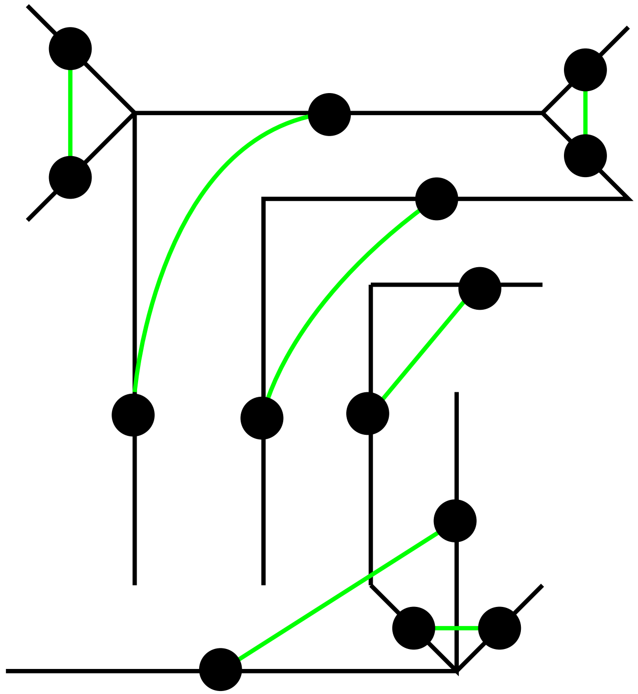
Example of poor ordering

$$\mathit{pair}\ p \longrightarrow \mathit{seg}\ s_1, \mathit{seg}\ s_2,\ \langle\, \underline{\mathit{orthogonal}(s_1, s_2)}, \underline{\mathit{adj}(s_1, s_2)}, \mathit{vertical}(s_1)\,\rangle$$

1. orthogonality
2. adjacency

Complexity:

$$O(\#\mathit{seg}^2 + \#\mathit{seg} \times \mathit{maxDeg}) = O(\#\mathit{seg}^2)$$



satisfied constraints

Constraint propagation as predicate ordering

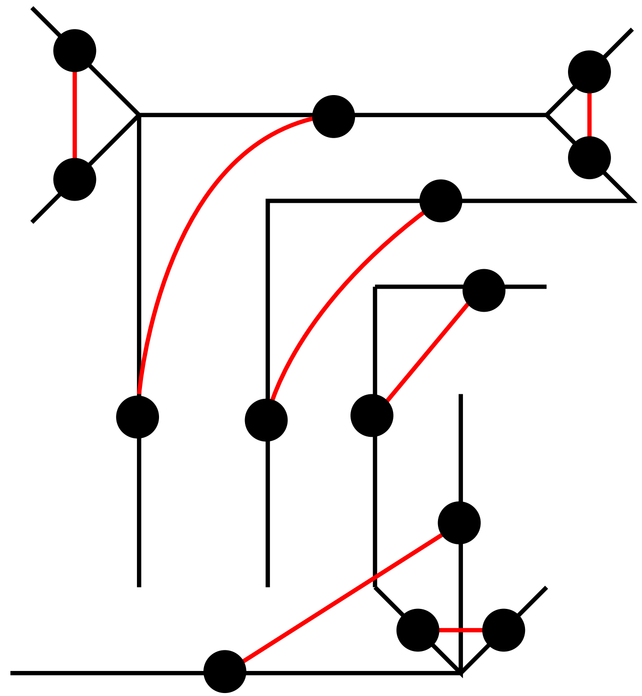
Example of poor ordering

$$\textit{pair } p \longrightarrow \textit{seg } s_1, \textit{seg } s_2, \; \langle\, \underline{\textit{orthogonal}(s_1, s_2)}, \underline{\textit{adj}(s_1, s_2)}, \underline{\textit{vertical}(s_1)} \,\rangle$$

1. orthogonality
2. adjacency
3. verticality

Complexity:

$$O(\#\textit{seg}^2 + \#\textit{seg} \times \textit{maxDeg} + \#\textit{seg}) = O(\#\textit{seg}^2)$$



constraints to test

Constraint propagation as predicate ordering

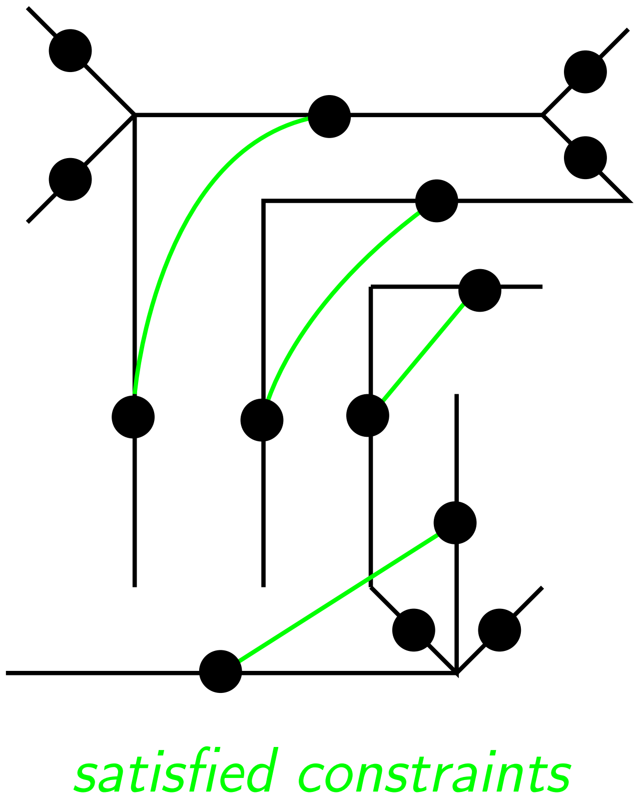
Example of poor ordering

$$\textit{pair } p \longrightarrow \textit{seg } s_1, \textit{seg } s_2, \ \langle \underline{\textit{orthogonal}(s_1, s_2)}, \underline{\textit{adj}(s_1, s_2)}, \underline{\textit{vertical}(s_1)} \rangle$$

1. orthogonality
2. adjacency
3. verticality

Complexity:

$$O(\#seg^2 + \#seg \times maxDeg + \#seg) = O(\#seg^2)$$



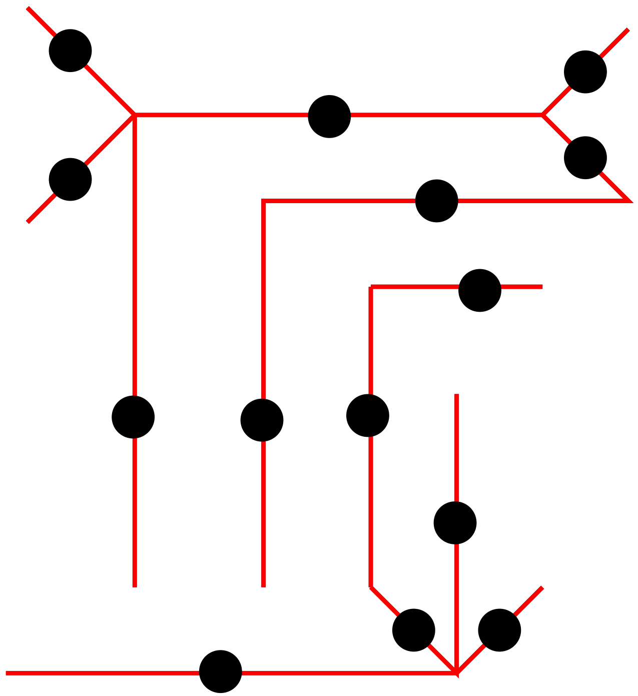
Constraint propagation as predicate ordering

Example of good ordering

$pair\ p \longrightarrow seg\ s_1, seg\ s_2,\ \langle\, orthogonal(s_1, s_2), adj(s_1, s_2), \underline{vertical(s_1)}\,\rangle$

1. verticality

Complexity :
$O(\#seg)$



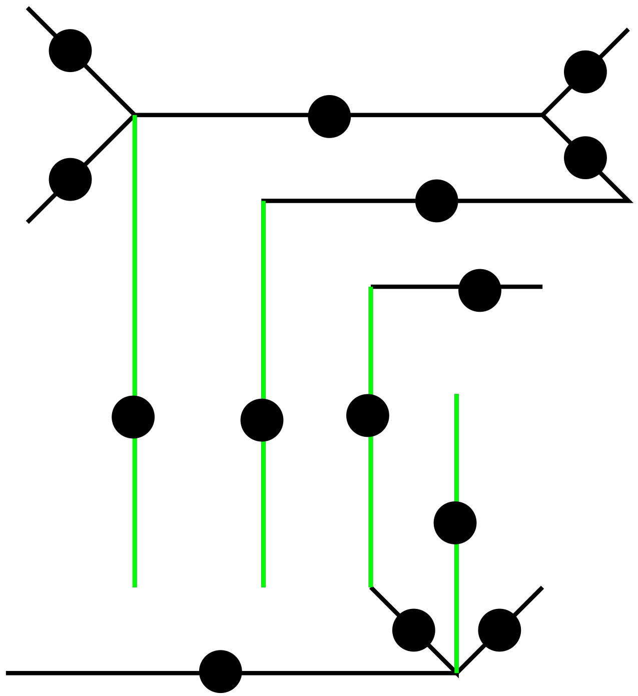
constraints to test

Constraint propagation as predicate ordering

Example of good ordering

$$pair\ p \longrightarrow seg\ s_1, seg\ s_2,\ \langle\, orthogonal(s_1, s_2), adj(s_1, s_2), \underline{vertical(s_1)}\,\rangle$$

1. verticality



satisfied constraints

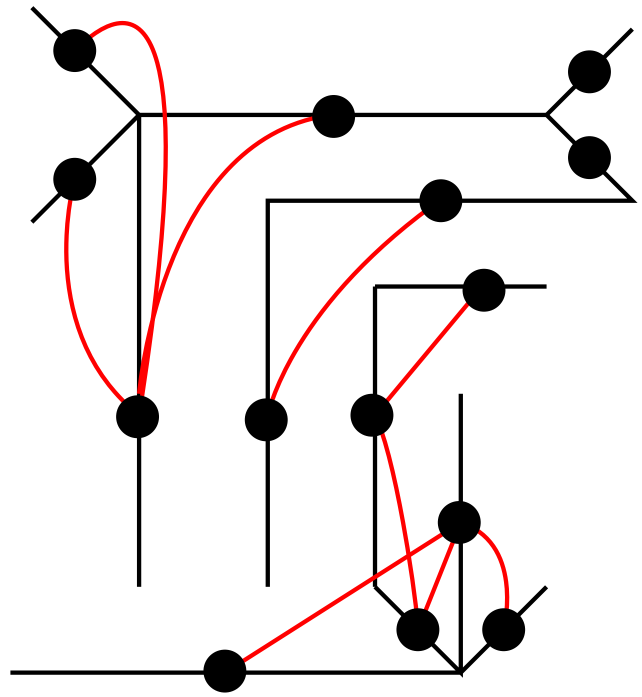
Complexity :
$O(\#seg)$

Constraint propagation as predicate ordering

Example of good ordering

$pair\ p \longrightarrow seg\ s_1, seg\ s_2,\ \langle\, orthogonal(s_1, s_2), \underline{adj(s_1, s_2)}, \underline{vertical(s_1)}\,\rangle$

1. verticality
2. adjacency



Complexity :

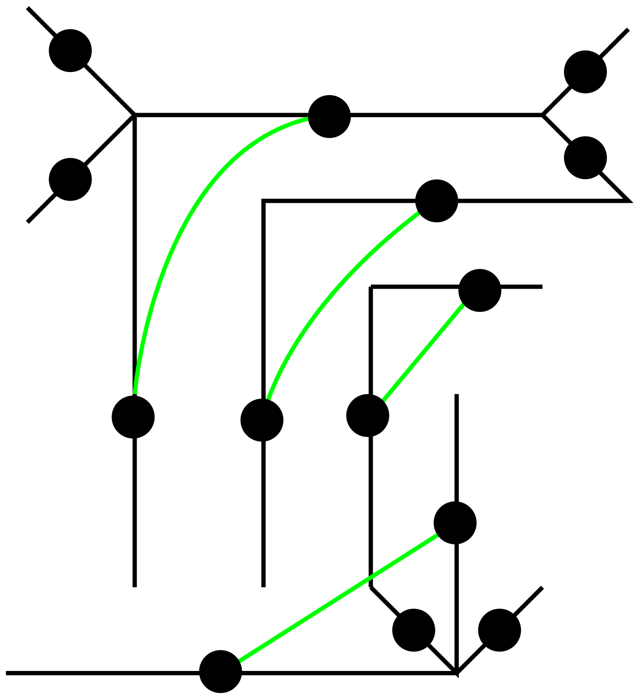
$$O(\#seg + \#seg \times maxDeg) = O(\#seg \times maxDeg)$$

Constraint propagation as predicate ordering

Example of good ordering

$$pair\ p \longrightarrow seg\ s_1, seg\ s_2,\ \langle\, \underline{orthogonal(s_1, s_2)}, \underline{adj(s_1, s_2)}, \underline{vertical(s_1)}\,\rangle$$

1. verticality
2. adjacency
3. orthogonality



Complexity :

$$O(\#seg + \#seg \times maxDeg + \#seg \times maxDeg)$$
$$= O(\#seg \times maxDeg) \ll O(\#seg^2)$$

Constraint propagation as predicate ordering

1. Unary predicates

= constraints implying only 1 element: complexity $O(\#element)$

Example:

$$riser\ r \longrightarrow polygon\ p\ \langle\ vertical(p)\ \rangle$$

Constraint propagation as predicate ordering

1. Unary predicates
2. Invertible predicates that are partially instantiated

= constraints with small cardinality when some arguments are fixed

Example:

$$\mathit{step}\ s \longrightarrow \mathit{riser}\ r, \mathit{tread}\ t\ \langle\, \mathit{edgeAdj}(r, t)\,\rangle$$

Constraint propagation as predicate ordering

1. Unary predicates
2. Invertible predicates that are partially instantiated
3. General predicates

$=$ remaining relations

Example:

$$\mathit{step}\ s \longrightarrow \mathit{riser}\ \ r, \mathit{tread}\ \ t\ \langle\, \mathit{orthogonal}(r, t)\,\rangle$$

Outline

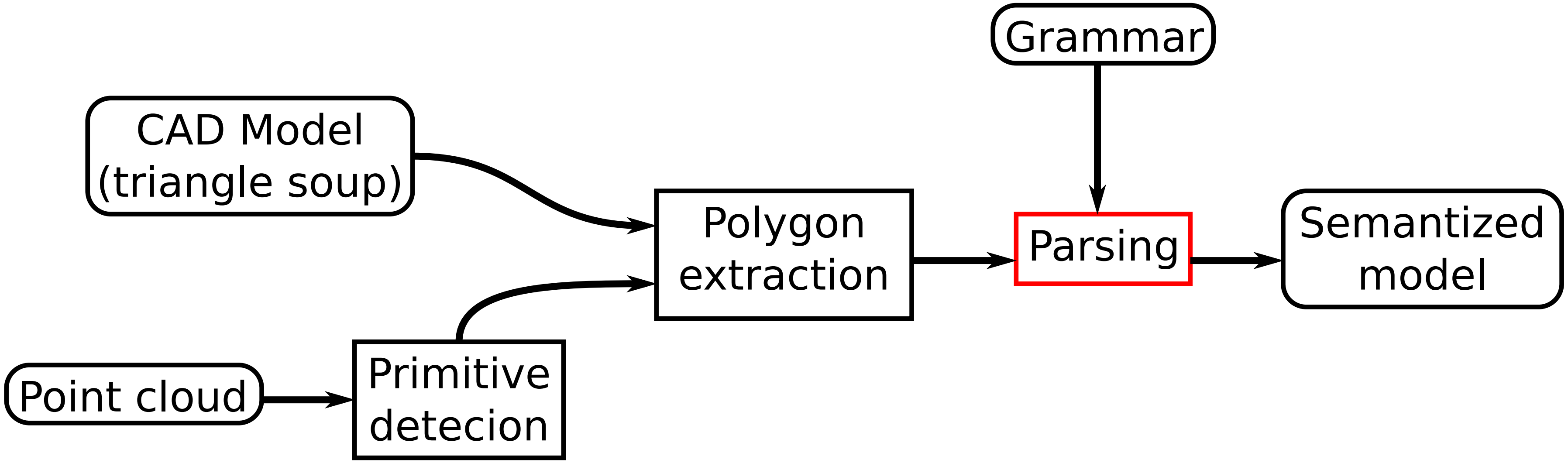
Constrained attribute grammars

Scene interpretation

Bottom-up parsing

Experiments

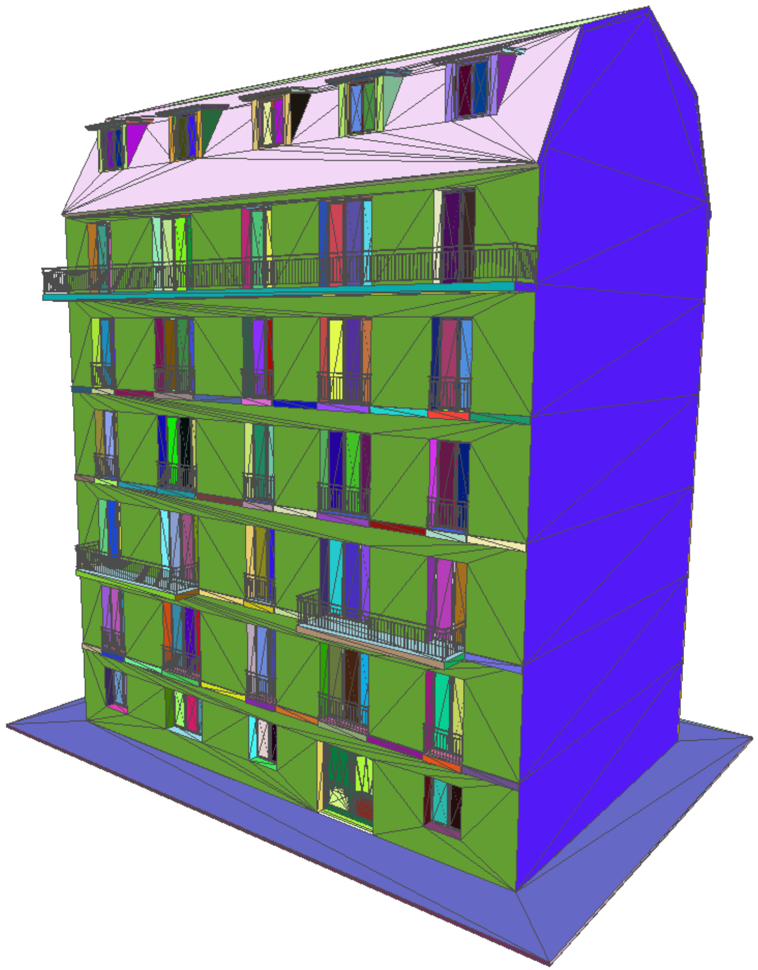
Semantization pipeline



CAD models

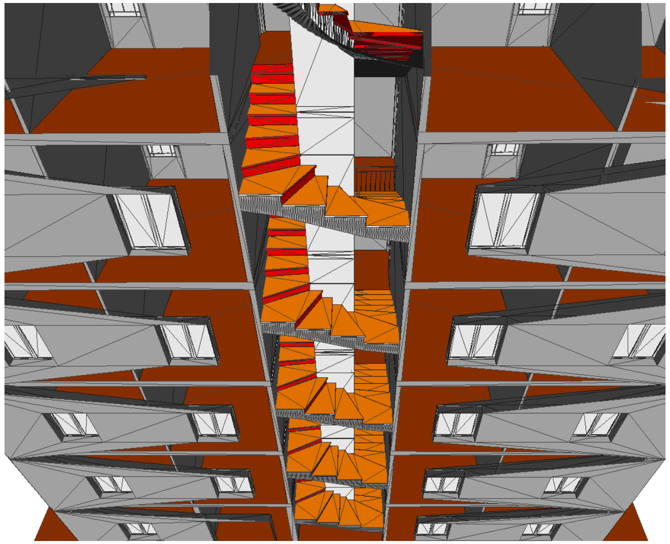
Preprocessing

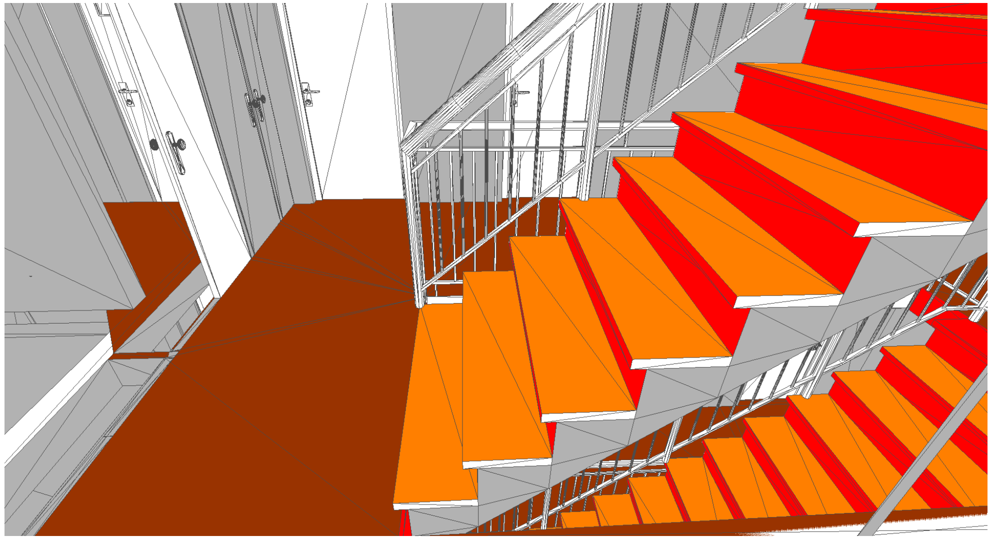
- Region growing over triangles for polygon creation
- Computation of exact and approximate adjacency graphs



CAD models

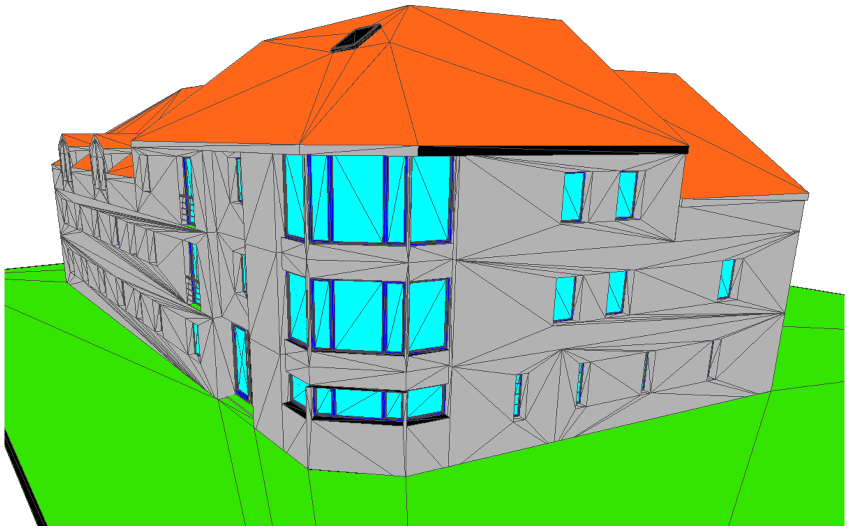
Detection of stairs (more examples in supplem. material)

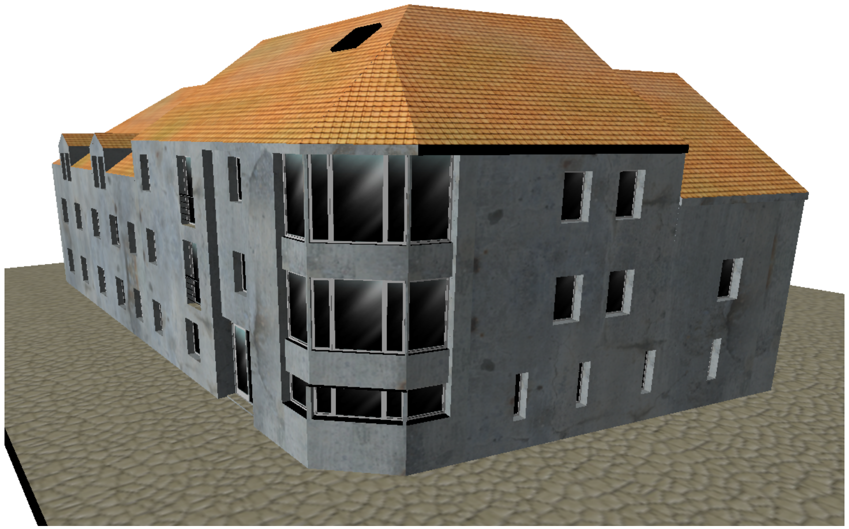




CAD models

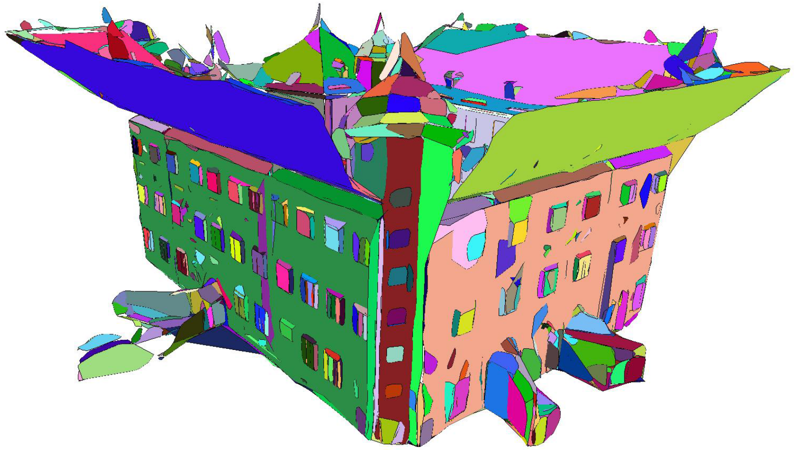
Detection of walls, roofs and openings (more examples in supplem. material)

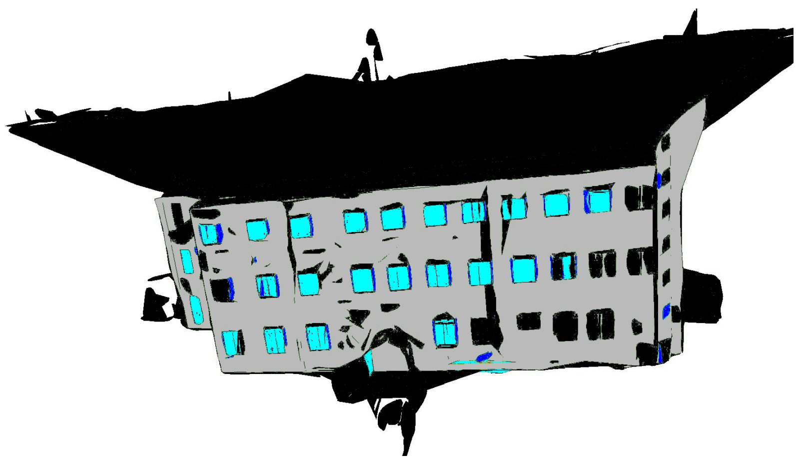




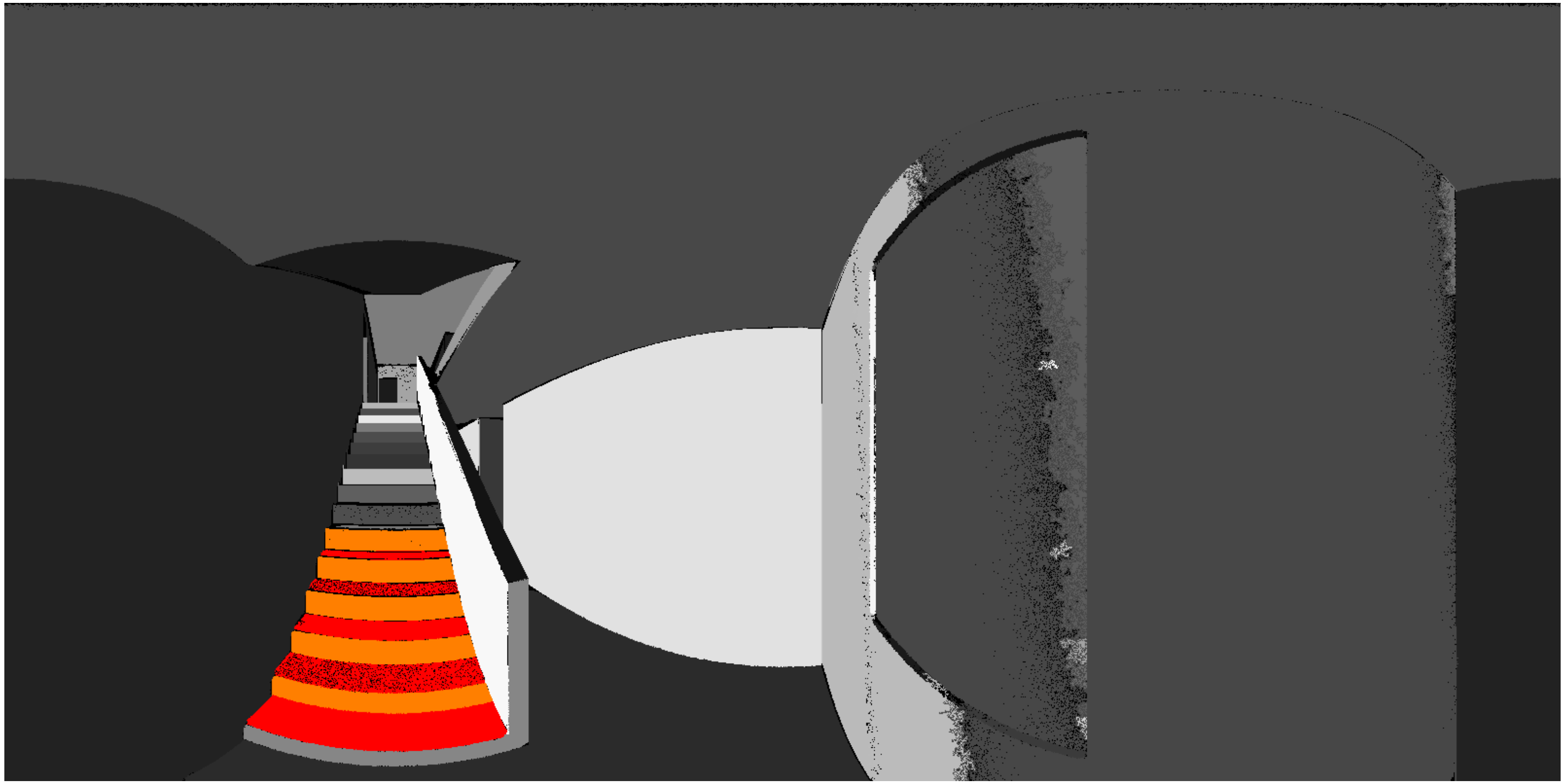
Real data: photogrammetry

- Preprocessing (point cloud)
  - clustering using RANSAC (or region growing)
  - polygons bounded by alpha shapes
- Problems:
  - missing primitives
  - false primitives
  - wrong adjacencies
- Solution:
  - use of a relaxed grammar
    - looser bounds
    - 1-2 missing items OK
  - 22 openings out of 31





Quasi-real data: simulated LIDAR



- Planes by region growing in depth image
- Polygons as oriented bounding rectangles
- Adjacency based on pixels in depth image

Size and parsing time (CAD models)

Name	# of triangles	# of polygons	Parsing time (s) stairs	Parsing time (s) openings
LcG	48332	9705	5	15
LcA	111979	26585	14	42
LcC	385541	111732	33	306
LcD	313012	75257	25	111
LcF	286996	84347	39	322

Precision and recall (%, CAD models)

Name	# of stairs	# of steps	Stairs Prec.	Stairs Rec.	Openings #	Openings Prec.	Openings Rec.
LcG	3	45	100	93	83	100	90
LcA	6	84	100	100	62	98	83
LcC	30	210	100	100	196	100	98
LcD	5	61	93	100	74	100	93
LcF	7	98	100	50	99	100	96

Future work

- Principled way to deal with partial or missing primitives
- Exploitation of occlusion/visibility information
- Scoring of interpretations: pick best tree(s)

Conclusion

Constrained attribute grammars:

- appropriate to semantize complex objects
- high-level specification language
  - being expert is enough, computer scientist not required
- efficient even on large models

This work:

- well-delimited first step: perfect data
- extensions required for incomplete/noisy data

On the web

- http://imagine.enpc.fr/
- sites.google.com/site/boulchalexandre/