Software Engineering and Enumerative Combinatorics

Alain Giorgetti[1] Richard Genestier[1] Valerio Senni[2]

[1]FEMTO-ST Institute, Univ. of Franche-Comté
[2]IASI-CNR, Roma

MAP 2014

Motivations

Cross-fertilization between software engineering and enumerative combinatorics

- Enumerative Combinatorics (EC)
  - Branch of mathematics
  - $\rightsquigarrow$ Counting discrete structures of given size
  - Also, exhibiting non-trivial structural bijections
- Software Engineering (SE)
  - methods for the rational design, dev^{t} and maintenance of software
  - validation, mainly by testing (around 50% of software dev^{t})
- EC for SE
  - Analysis of algorithm complexity
  - Bounded exhaustive testing with structured data
- SE for EC
  - Methods for guessing and proving conjectures in combinatorics
  - Focus on rooted map enumeration

Outline

Bounded exhaustive testing

- **Motivation:** test cases for programs manipulating structured data (lists, arrays, trees, etc.) with complex invariants (e.g. red-black trees, Dyck words)
- Exhaustive generation of combinatorial structures up to some given (small) size
- Naive solution: Rejection (not efficient)
- Test case generators based on constraint logic programming (CLP)

Example: Dyck words

- A Dyck word over the alphabet $\{(,)\}$ is a balanced parenthesis word
- A Dyck word of length $2n$ (size n) contains n pairs of parentheses (possibly nested) which correctly match
- Example: (( )) ((( ) ( )))
- Grammar: $D ::= \varepsilon \mid (D) D$

CLP-based test case generators [SF12]

- Motivation: test cases for programs manipulating structured data (lists, arrays, trees, etc.) with complex invariants
- Logic programs provide declarative specifications of test cases
- Filter promotion techniques optimize specifications

Logic programming

- Programs are sets of rules (Horn clauses) of the form
  $\mathrm{C} \;\texttt{:-}\; \mathrm{H}_1 \wedge \ldots \wedge \mathrm{H}_n$
  (meaning, C holds if H_i holds for $i = 1, \ldots, n$)
- Example

```
ordered([]).
ordered([X]).
ordered([X1,X2|L]) :- X1 ≤ X2 ∧ ordered([X2|L]).
```

- Query evaluation
  1. Pick leftmost atom in current query: $\mathrm{Q} = \mathrm{H} \wedge \mathrm{R}$
  2. Find unifying head: $\mathrm{C}\,\sigma = \mathrm{H}\,\sigma$
  3. Rewrite to get a new query: $(\mathrm{H}_1 \wedge \ldots \wedge \mathrm{H}_n \wedge \mathrm{R})\,\sigma$

LP-based generation

```
ordered([]).
ordered([X]).
ordered([X1,X2|L]) :- X1 ≤ X2 ∧ ordered([X2|L]).
```

as a generator:

`ordered(L).`

`L = []`

`L = [X]`

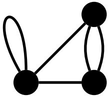
`L = [X1,X2]` with $X_1 \leq X_2$

... `L = [X1,X2,X3]` with $X_1 \leq X_2 \land X_2 \leq X_3$

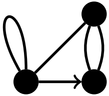
Outline

Planar topological map

- A planar topological map is a connected graph (loops and multiple edges allowed) drawn on the sphere so that each connected component of the complement of the graph (face) is homeomorphic to an open disc
- Maps are studied (generated, counted, etc.) up to isomorphism (orientation-preserving surface isomorphism $+$ underlying graph isomorphism)



- A rooted map is a map with a distinguished dart (half an edge), its root
- Rooted maps have no non-trivial (root-preserving) automorphism $\rightarrow$ easier to study than maps



- A combinatorial map is a triple (D, R, L) where D is a finite set, R is a permutation of D and L is a fixpoint-free involution of D such that the group $\langle R, L\rangle$ generated by R and L acts transitively on D

Correspondence between two map encodings

Encodings of rooted planar maps

- By words: Canonical parenthesis-bracket systems [Walsh & Lehman 72], named p-words here
- By trees
  - Former proposals: well labeled trees [Cori & Vauquelin 81], balanced blossom trees [Schaeffer 03]
  - New family (conjecture): p-trees
- New theorem: p-words and p-trees of the same size are in one-to-one correspondence

p-words

- A p-word is any shuffle of a Dyck word on the alphabet $\{(,)\}$ and a Dyck word on the alphabet $\{[,]\}$, which does not contain any subword [( ]) composed of two pairs [ ] and ( ) matching in the Dyck words (canonicity property)
- Forbidden pattern $\ldots[\ldots(\ldots]\ldots)\ldots$
- Example
  - 9 p-words with 4 letters
    (( ))   ([ ])   ([ )]   ( ) ( )   ( ) [ ]   [( )]   [[ ]]   [ ] ( )   [ ] [ ]
  - One non-canonical p-word with 4 letters: [( ])
- The size of a p-word is half its length

Design of efficient p-word generators [GS12]

Exploiting the resolution-based computation mechanism of Prolog

1. First declarative version in logic programming (specification, correct)
   - Dyck words, two kinds of parentheses
   - Shuffling
   - Inefficient: Several computation branches leading to failure
2. Second (more operational) version
   - Based on word extension from left to right + a stack of counters
   - More efficient
3. Third version (optimized)
   - Pruning failing computations in the second version
   - Even more efficient

How to ensure correctness of (2) and (3) w.r.t. (1)?

Correctness of p-word generators

How to ensure correctness of (2) and (3) w.r.t. (1)?

- Compare their outputs incrementally (by the size of the structure)
- Number of generated structures
- Sets of generated structures
- Programs validated up to size 11 (constructing around $1.60x10^9$ structures)
- Also for a translation of the optimized program (3) into C
- Our C program is more efficient than any other C program in the literature
- Incremental comparison improves confidence of correctness
- Logic programming-supported method for the design of combinatorial algorithms

What are the key ingredients of the proof?

- Bijection between two encodings of rooted planar maps
  - p-words
  - p-trees (see next slide)
- Computer-assisted discovery of bijections w2t and t2w between both families
- With a validation tool (LP-based) and a proof assistant (Coq/SSReflect)

Definition of p-trees

- An mtree is a (rooted plane) binary-unary tree in which each unary node is labelled by a natural number

```
Inductive mtree :=
| mty : mtree
| bnode : mtree → mtree → mtree
| unode : ℕ → mtree → mtree.
```

- The degree of an mtree is defined by

```
Function deg (t : mtree) : ℕ :=
 match t with
 | mty ⇒ 0
 | bnode u v ⇒ 2 + deg u + deg v
 | unode n _ ⇒ n + 1
 end.
```

- A ptree is an mtree where each unary node label does not exceed the degree of its child
- The size of a tree is the total number of its nodes

p-trees in Coq/SSReflect

- A ptree is an mtree where each unary node label does not exceed the degree of its child
- Characteristic property of p-trees among m-trees

```
Function isPtree (t : mtree) : bool :=
 match t with
 | mty ⇒ true
 | bnode u v ⇒ isPtree u && isPtree v
 | unode n w ⇒ isPtree w && (n <= deg w)
 end.
```

- ptrees are mtrees with this property

```
Structure ptree : Type := mkPtree {
 pval :> mtree;
 _ : isPtree pval
}.
```

p-words in Coq/SSReflect

- Letters: [ ] ( ) : lett
- Words: Definition word := seq lett .
- Dyck words on parentheses

```
Inductive dwp : word → Prop :=
| mtyP : dwp nil
| decompP : ∀ u v : word,
   dwp u → dwp v → dwp (( :: u ++ ) :: v).
```

- Characterization of p-words (adapted from [Cor75, Property II.7])

```
Inductive pword : word → Prop :=
| pwordmty : pword nil
| pwordbracket : ∀ u v : word,
   pword u → pword v → pword ([ :: u ++ ] :: v)
| pwordparen : ∀ u v : word, dwp (rmb u) →
   pword (u ++ v) → pword (( :: u ++ ) :: v).
```

where rmb removes brackets

Validation

- Similar definitions in Prolog
- Same number of generated structures up to size 6
- Sequence 1, 2, 9, 54, 378, 2916, 24057 (`https://oeis.org/A000168`)
- Same set of generated structures up to size 5
- Guess inductive functions
  t2w : mtree $\rightarrow$word
  and
  w2t : word $\rightarrow$tree
  whose restrictions to ptrees and pwords are bijective

From p-trees to p-words

```
Fixpoint t2w (t : mtree) {struct t} : word := ???
```

- Source of inspiration: Binary trees → Dyck words

```
match t with
| mty ⇒ nil
| bnode u v ⇒ [ :: t2w u ++ ] :: t2w v
| unode n s ⇒ let w := t2w s in ( :: insertCP w n
```

- Ideas for the insertion function
  - n is sometimes less than the length of w
  - Add a) before the first n letters of w?
    - Invalidated by generation of words of size 3
    - ([( ))] twice, ([( )]) missing
  - Add a) after the n-th Dyck word in rmb w?
    - Invalidated, but works with deg s − n instead of n

From p-words to p-trees

```
Fixpoint w2t (w : word) {struct w} : mtree :=
 match w with
 | nil ⇒ mty
 | [ :: u ⇒ ??
 | ( :: u ⇒ ???
 end.
```

- For [, similar to parsing of Dyck words
- For free in LP

```
w2t([], mty).
w2t([b|W], b(T1, T2)) :- append(U, [r|V], W),
  pword(U), pword(V), w2t(U, T1), w2t(V, T2).
```

- For (, discovery in Prolog

```
w2t([p|W], u(N, T)) :- append(U, [a|V], W),
  rmb(U, P), dwp(P), append(U, V, S), w2t(S, T),
  cn(V, Np1), N is Np1-1.
```

- Last line guessed, comparing T with the antecedent of W by t2w.

Outline

Conclusion

- Software engineering methods to
  - Assist the discovery and proof of new results in combinatorics
  - Design and validate generators of structured data/combinatorial objects
- Giving more confidence in scientific results and programs
- Testing works as an accelerator, formal proving as a brake
- Thanks to Reynald Affeldt, Cyril Cohen and Enrico Tassi for their help on SSReflect, to Timothy R. S. Walsh for helpful comments and to Noam Zeilberger for exciting discussions
- Work in progress... Join the team!

References

Robert Cori.
Un code pour les graphes planaires et ses applications.
Société mathématique de France, 1975.

Alain Giorgetti and Valerio Senni.
Specification and Validation of Algorithms Generating Planar Lehman Words.
In GASCom'12, Bordeaux, France, June 2012.

Valerio Senni and Fabio Fioravanti.
Generation of test data structures using constraint logic programming.
In Achim D. Brucker and Jacques Julliand, editors, TAP, volume 7305 of Lecture Notes in Computer Science, pages 115–131. Springer, 2012.