Adocs : a Drawing System for Generic Combinatorial Structures

François Bertault

INRIA Lorraine, 615 rue du Jardin Botanique, BP 101,
F-54600 Villers-les-Nancy, France

Abstract. Existing graph drawing systems imply the use of specific algorithms for each kind of data structures. This paper provide a description of the `Adocs` program. The program is based on a generic description of the structures, and thus allows to draw structures of an infinite number of classes. It can be used in order to produce graphical output for Gaïa, an uniform random generator of combinatorial structures. It could be used with other programs and is well suited for drawing compound objects.

1 Introduction

The theory of decomposable structures [1], used in order to describe combinatorial objects, allows to describe large classes of data structures. With only a small set of constructors, we can describe structures frequently used in software, like permutations, trees, partitions, or functionals graphs. The `Adocs` [1] program produces a graphical representation for such objects.

2 Description of Combinatorial Structures

Definition 1. The objects classes we can deal with are specified by a set of productions of the form `A = <rhs>`, where `A` is the name of the class being defined, and `<rhs>` is an expression involving elementary classes, constructors and other classes specifications.

Elementary classes are :

- `Epsilon` : object of size 0
- `Atom` : object of size 1

Constructors available are :

- `Union(A,B,...)` : disjoint union of the classes A,B,...
- `Prod(A,B, ...)` : product of the classes A, B,...
- `Set(A)` : sets (with repetitions) whose elements are in A
- `Sequence(A)` : sequences of elements of A
- `Cycle(A)` : directed cycles of elements of A
- `Subst(A,B)` : B-objects whose atoms are replaced by A-objects

[1] available via anonymous ftp from `ftp.loria.fr`, in `pub/loria/eureca/ADOCS/`

Example 1. The class `N` of functional digraphs can be defined as a set of cycles of binary trees. A binary tree (class `Tree`) can be defined as a leaf or the product of a node with two trees :

```
{N=Set(Cycle(Tree)), Tree=Union(Leaf,Prod(Node,Tree,Tree)),
Node=Atom, Leaf=Atom}
```

Definition 2. Inputs of `Adocs` are objects described with the following initial objects and lowercase operators :

Initial objects are :
- `Identifiers` : a word with alpha-numerical characters, or a set of characters between double quotes.

Operators are :
- `prod(a,b,...)` : product of the objects a, b,...
- `set(a,b,..)` : set of objects a, b,...
- `sequence(a,b,...)` : sequence of objects a,b,...
- `cycle(a,b,...)` : directed cycle of objects a,b,...

Example 2. `set(cycle(prod(A,a1,a2)),cycle(prod(B,"f(3)","*%$")))` is a valid functional digraph object. Figure 1 shows an object described using all the operators.

3 Representation

The default settings for each operator representation are :

- `prod(o1,o2,..,om)` draws a rooted tree, where `o1` is the father of the objects `o2 ... om`, with non-oriented edges between father and sons.
- `sequence(o1,o2,..,om)` places `o1 .. om` on a straight line, starting with `o1` on the left, with oriented edges between adjacent elements.
- `cycle(o1,o2,..,om)` places `o1 .. om` along a circle, with oriented edges.
- `set(o1,o2,..,om)` places `o1 .. om` along a spiral, with no edges.

We can add informations to each operator in order to constraint the representation. Thus, we can have different representations of a same structure (Fig. 2). The instructions are as follows, where `<operator>` is either `prod`, `sequence`, `cycle` or `set` :

- `<operator>[Tree](o1,o2,..om)` to represent a rooted tree (default for `prod`).
- `<operator>[Radial](o1,o2,..om)` to represent a free-tree.
- `<operator>[Line](o1, o2, ... om)` to place elements `o1 .. om` on an horizontal line (default for `sequence`).
- `<operator>[Circle](o1, o2, ... om)` to place elements `o1 .. om` on a circle (default for `cycle`).
- `<operator>[Spiral](o1, o2, ... om)` to place elements `o1 .. om` on a spiral (default for `set`).

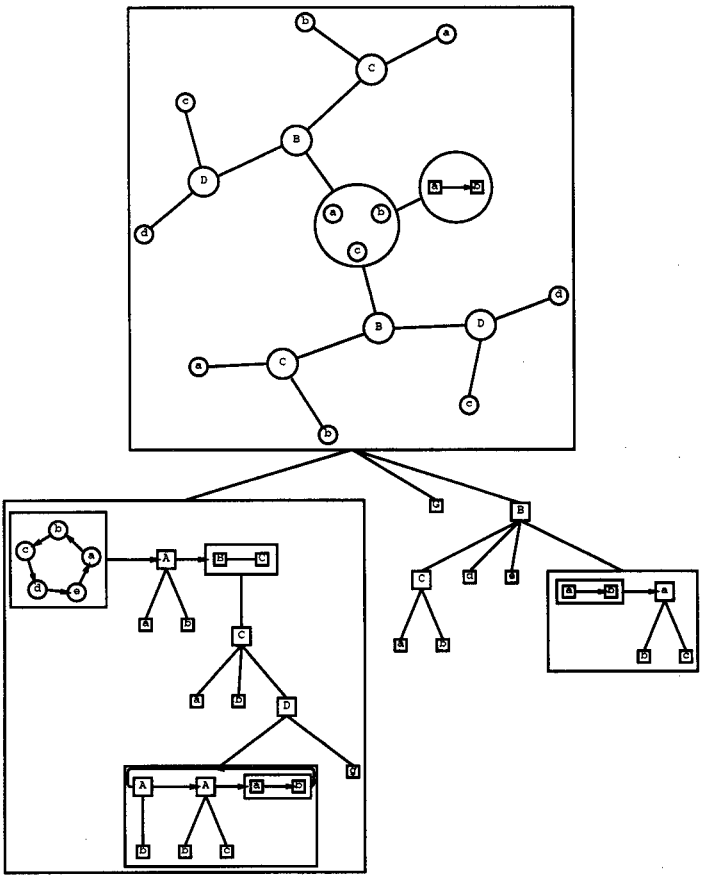

Fig. 1. Rooted tree with free-tree, sequence of trees, cycle and set

4 Implementation

Different layout algorithms are used in the **Adocs** program. The algorithm for drawing rooted trees is based on the Reingold and Tilford algorithm [3]. The algorithm has been extended to m-ary trees with nodes of different sizes. For drawing free-trees, we use an algorithm proposed by P. Eades [2]. The spiral algorithm is heuristic and tries to place a set of circles in the smallest possible enclosing circle. All algorithms used need improvement and other algorithms could be included, but **Adocs** can already draw quite large structures (more than 10000 nodes) in a few seconds.

The **Adocs** interface is very simple. The description of the object we want to draw is written into a file (for example **descr**). The Unix command **Adocs descr** produces the **descr.ps** Postscript file containing the drawing of the structure. Other outputs can also be provided, like LaTeX pictures or a specific simple graphical description (involve **AdocsTex** or **AdocsGr** instead of **Adocs**). **Adocs**

```
prod(prod[Radial](C,H,H,H),
     cycle(a,b,c,d,e),
     sequence(A,
              prod(B,a,b),
              C))
```

```
prod[Line](prod(C,H,H,H),
     cycle[Line](a,b,c,d,e),
     sequence[Tree](A,
              prod(B,a,b),
              C))
```

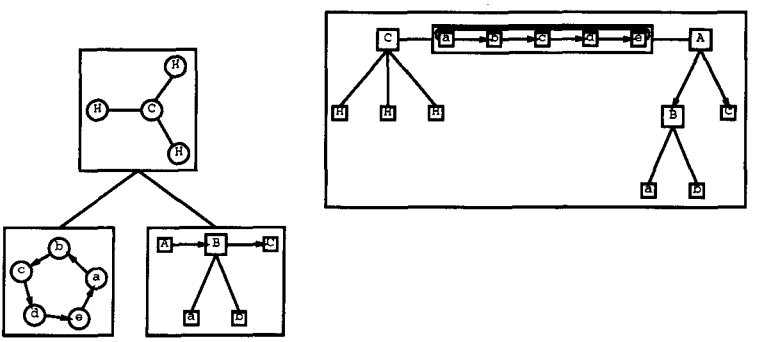

Fig. 2. Two different drawing of a same structure

can then easily be connected to other programs, like structure generators or other drawing programs.

For example **Adocs** can directly draw the structures calculated by the random structure generator Gaïa [2] [4]. Given a structure class specification and a size of the object we want to obtain, Gaïa generates with uniform probability an object of the class, that we can represent with **Adocs**.

5 Conclusion

Adocs could also be used with other programs, like data structures browsers. In particular, the generic description proposed for data structures is well suited for compound drawing of data structures.

References

1. Philippe Flajolet, Paul Zimmermann, and Bernard Van Cutsem. A calculus for the random generation of labelled combinatorial structures. *Theoretical Computer Science*, 132(1-2):1–35, 1994.
2. Eades Peter D. Drawing free trees. *Bulletin of the Institute for Combinatorics and its Applications*, 5(2):10–36, 1992.
3. Edward M. Reingold and John S. Tilford. Tidier drawings of trees. *IEEE Transactions on Software Engineering*, SE-7(2):223–228, March 1981.
4. Paul Zimmermann. Gaïa: a package for the random generation of combinatorial structures. *MapleTech*, 1(1):38–46, 1994.

[2] available via anonymous ftp from `ftp.inria.fr`, in `lang/maple/INRIA/combstruct`