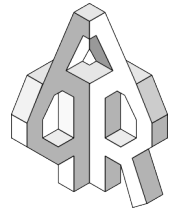
Génération aléatoire d'arbres et applications

Alexis Darrasse

Journée des doctorants des laboratoires LIP6 et LTCI
1er Octobre 2008



Random sampling

Combinatorial structures

- Random input for testing (software validity, server robustness)
- Property visualization (large scale)
- Model adequation - Check conjectures

Boltzmann sampling

$\mathcal{C}$ set of objects γ with size function $|| : \mathcal{C} \to \mathbb{N}$

Boltzmann model, parameter x

$$\mathrm{P}_x(\gamma) = \frac{x^{|\gamma|}}{C(x)} \quad \text{where} \quad C(x) = \sum_{\gamma \in \mathcal{C}} x^{|\gamma|}$$

- Combinatorial classes defined by specification grammars
- Oracle for computing $C(x)$

Properties

- Guarantees uniformity (between objects of the same size)
- Result size is random, but can be tuned with parameter x
- Linear complexity of sampling $\to$ huge objects

Boltzmann sampling

Properties

- Guarantees uniformity (between objects of the same size)
- Result size is random, but can be tuned with parameter x
- Linear complexity of sampling $\longrightarrow$ huge objects

construction	sampler with parameter x
$\mathcal{C} = \mathcal{I}$	$\Gamma C(x) := \varepsilon$
$\mathcal{C} = \mathcal{Z}$	$\Gamma C(x) := z$
$\mathcal{C} = \mathcal{A} + \mathcal{B}$	$\Gamma C(x) := \text{Bern}\,\frac{A(x)}{C(x)} \longrightarrow \Gamma A(x) \mid \Gamma B(x)$
$\mathcal{C} = \mathcal{A} \times \mathcal{B}$	$\Gamma C(x) := \langle\, \Gamma A(x)\ ;\ \Gamma B(x)\, \rangle$
$\mathcal{C} = \textsc{Seq}(\mathcal{A})$	$\Gamma C(x) := \text{Geom}\, A(x) \Longrightarrow \Gamma A(x)$

Specifications

O'Caml

```
type expression =
    Const of float
  | Var of string
  | Sum of expression * expression
  | Prod of expression * expression
```

Combinatorics

$$\mathcal{E} = \mathcal{Z} \cup \mathcal{Z} \cup \mathcal{E} \times \mathcal{E} \cup \mathcal{E} \times \mathcal{E}$$

Generating function

$$E(z) = \sum_{n=0}^{\infty} E_n z^n \text{ with } E_n = \# \text{ expr. with } n \text{ const. or var.}$$

$$E(z) = 2z + 2E^2(z)$$

Expressions Boltzmann sampler

```
type expression =
    Const of float
  | Var of string
  | Sum of expression * expression
  | Prod of expression * expression
```

$$\mathcal{E} = \mathcal{Z} \cup \mathcal{Z} \cup \mathcal{E} \times \mathcal{E} \cup \mathcal{E} \times \mathcal{E}$$

Sampler

```
let rec rand_exp () =
  let r = Random.float 1.0 in
  if      r < 0.25 then Const 1.0
  else if r < 0.5  then Var "foo"
  else if r < 0.75 then Sum (rand_exp (), rand_exp ())
  else                  Prod (rand_exp (), rand_exp ())
;;
```

Result

```
# rand_exp ();;
- : expression =
Prod (Var "foo",
 Sum (Prod (Var "foo", Sum (Sum (Const 1., Var "foo"), Var "foo")),
  Sum
   (Sum (Prod (Prod (Const 1., Var "foo"), Const 1.),
     Prod (Var "foo", Var "foo")),
    Sum
     (Prod
       (Sum
         (Prod
           (Prod
             (Prod
               (Prod
                 (Prod (Sum (Const 1., Prod (Var "foo", Var "foo")), Var "foo"),
                  Sum
                   (Prod
                     (Sum
                       (Prod
                         (Prod (Var "foo",
                           Sum (Var "foo",
                            Sum (Sum (Prod (Const 1., Var "foo"), Var "foo"),
                             Const 1.))),
                          Sum (Sum (Var "foo", Var "foo"),
                           Prod (Var "foo", Prod (Const 1., Var "foo")))),
                        Var "foo"),
                      Sum (Prod (Var "foo", Const 1.),
                       Sum (Const 1.,
                        Sum (Const 1.,
                         Prod (Sum (Const 1., Const 1.),
                          Prod (Prod (Var "foo", Var "foo"), Var "foo")))))),
                   Const 1.)),
               Const 1.),
             Sum (Const 1., Var "foo")),
           Prod (Sum (Var "foo", Sum (Prod (Const 1., Var "foo"), Var "foo")),
            Const 1.)),
         Const 1.),
```

Constructible classes

	specification	ordinary g.f. (unlabelled)	exponential g.f. (labelled)
ε / atom	$\mathcal{I}$ / $\mathcal{Z}$	1 / x	1 / x
Union	$\mathcal{C} = \mathcal{A} \cup \mathcal{B}$	$C(x) = A(x) + B(x)$	$\hat{C}(x) = A(x) + B(x)$
Product	$\mathcal{C} = \mathcal{A} \times \mathcal{B}$	$C(x) = A(x) \times B(x)$	$\hat{C}(x) = A(x) \times B(x)$
Sequence	$\mathcal{C} = \textsc{Seq}(\mathcal{A})$	$C(x) = \frac{1}{1-A(x)}$	$\hat{C}(x) = \frac{1}{1-A(x)}$
Set	$\mathcal{C} = \textsc{Set}(\mathcal{A})$	$\exp\left(\sum_{k=1}^{\infty} \frac{(-1)^{k-1}}{k} A(x^k)\right)$	$\hat{C}(x) = \exp(A(x))$
Multiset	$\mathcal{C} = \textsc{MSet}(\mathcal{A})$	$\exp\left(\sum_{k=1}^{\infty} \frac{1}{k} A(x^k)\right)$	–
Cycle	$\mathcal{C} = \textsc{Cyc}(\mathcal{A})$	$\sum_{k=1}^{\infty} \frac{\varphi(k)}{k} \log \frac{1}{1 - A(x^k)}$	$\hat{C}(x) = \log \frac{1}{1-A(x)}$

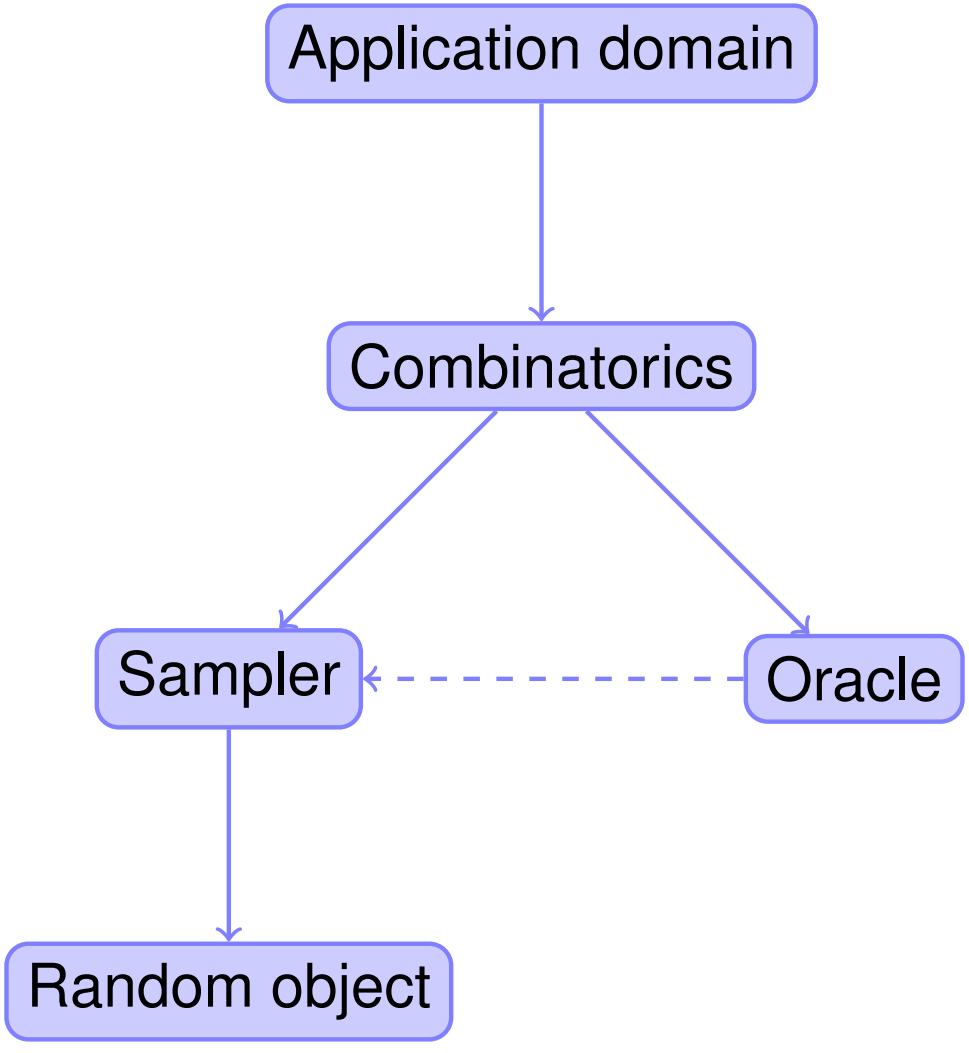
And more to come...

Applications

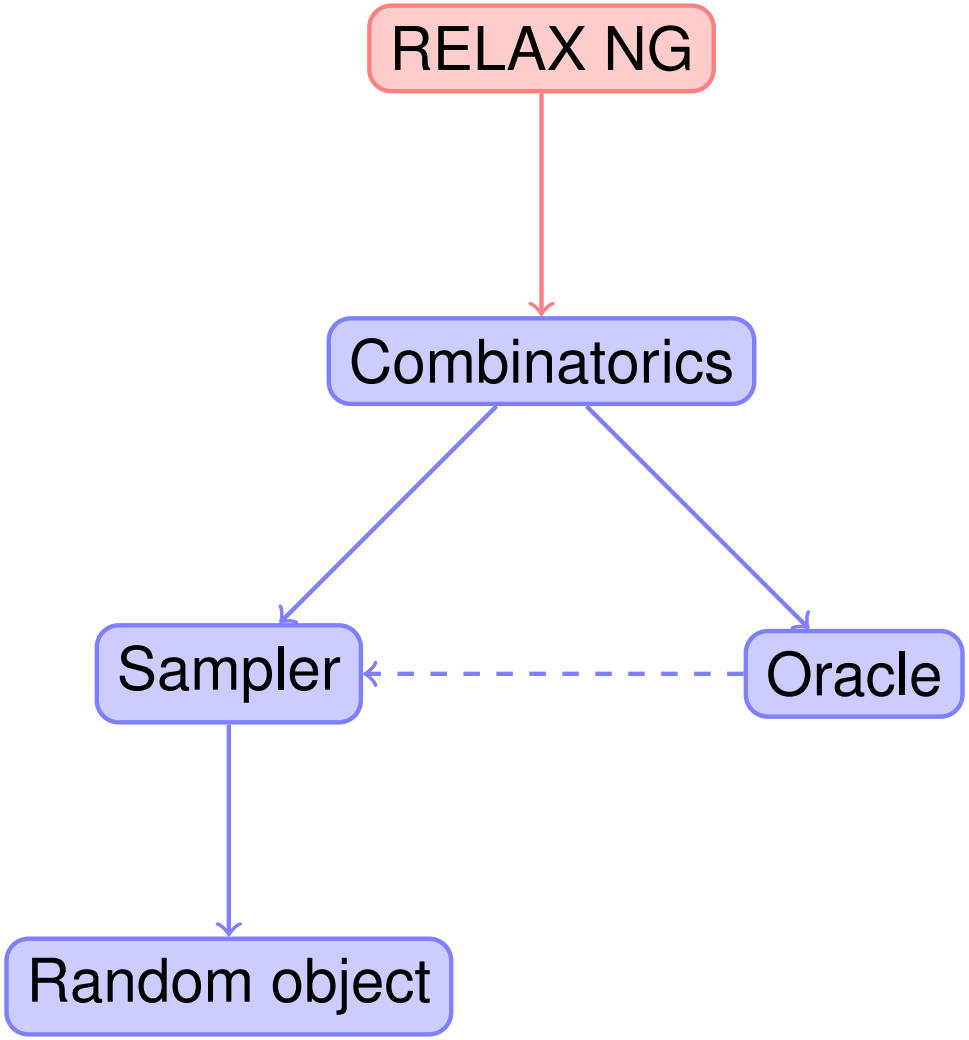
- Planar and other graphs
- Automata
- Boolean expressions
- RNA structures
- Software testing
- Plane partitions
- ...

Application of random sampling of trees: XML document sampling

The framework



The framework



RELAX NG - Compact syntax

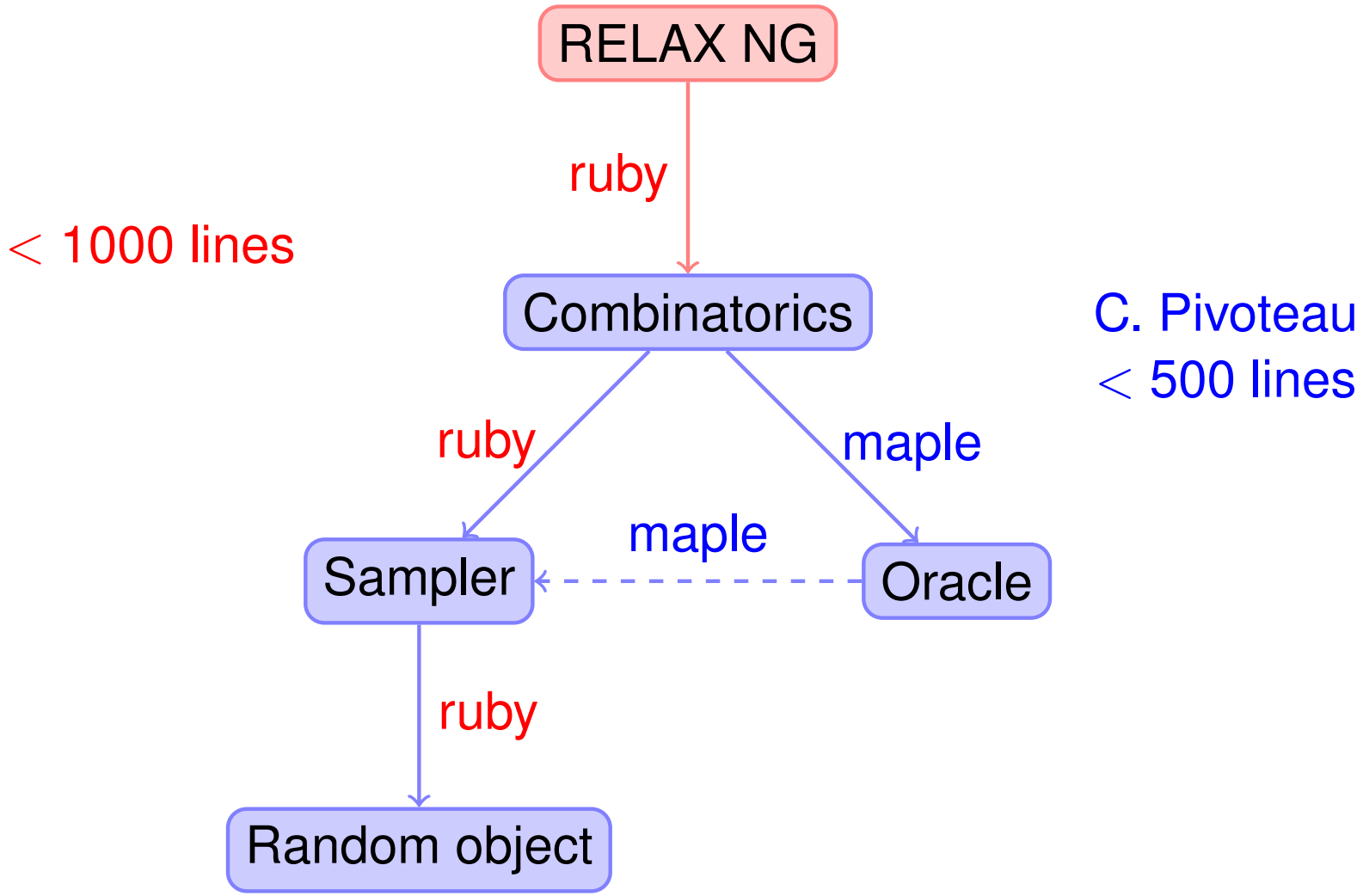
An example

```
start = expr

expr = element const {
         attribute val { xsd:integer },
         empty }
     | element var   {
         attribute val { xsd:string },
         empty }
     | element sum   { expr, expr }
     | element prod  { expr, expr }
```

Operator correspondance

RELAX NG	Combinatorics
`empty`	$\mathcal{I}$
`data/value/text`	$\mathcal{Z}$
`choice`	$\cup$
`group`	$\times$
`oneOrMore`	SEQ
`define`(X)	X = ...
`ref`(X)	X
`element/attribute`	$\mathcal{Z} \times \ldots$

The framework



Some results

grammar	file	# el.	s.c.c.	oracle
ternary trees	1024	2	1	0.260s
RSS	9.5K	10	1	0.320s
PNML	23K	22	1	0.322s
ILP 1	21K	20	6	0.416s
ILP 6	99K	51	31	0.828s
RELAX NG	124K	33	18	0.696s
XSLT	168K	40	17	0.469s
XHTML	289K	47	32	0.932s
XML Schema	237K	59	9	0.528s
XHTML basic	284K	53	38	1.080s
XHTML strict	1.2M	80	58	2.414s
XHTML	1.5M	93	66	3.798s
SVG tiny	1.6M	49	7	0.371s
SVG full	6.3M	118	27	0.718s
MathML	2.2M	182	48	2.432s
OpenDocument	2.8M	500	101	6.544s
DocBook	11M	407	295	143.411s

Current state

Sampling performance

In 2 minutes:

100 ILP 1 documents of size $>$ 1000 (64MB)

In 1 minute:

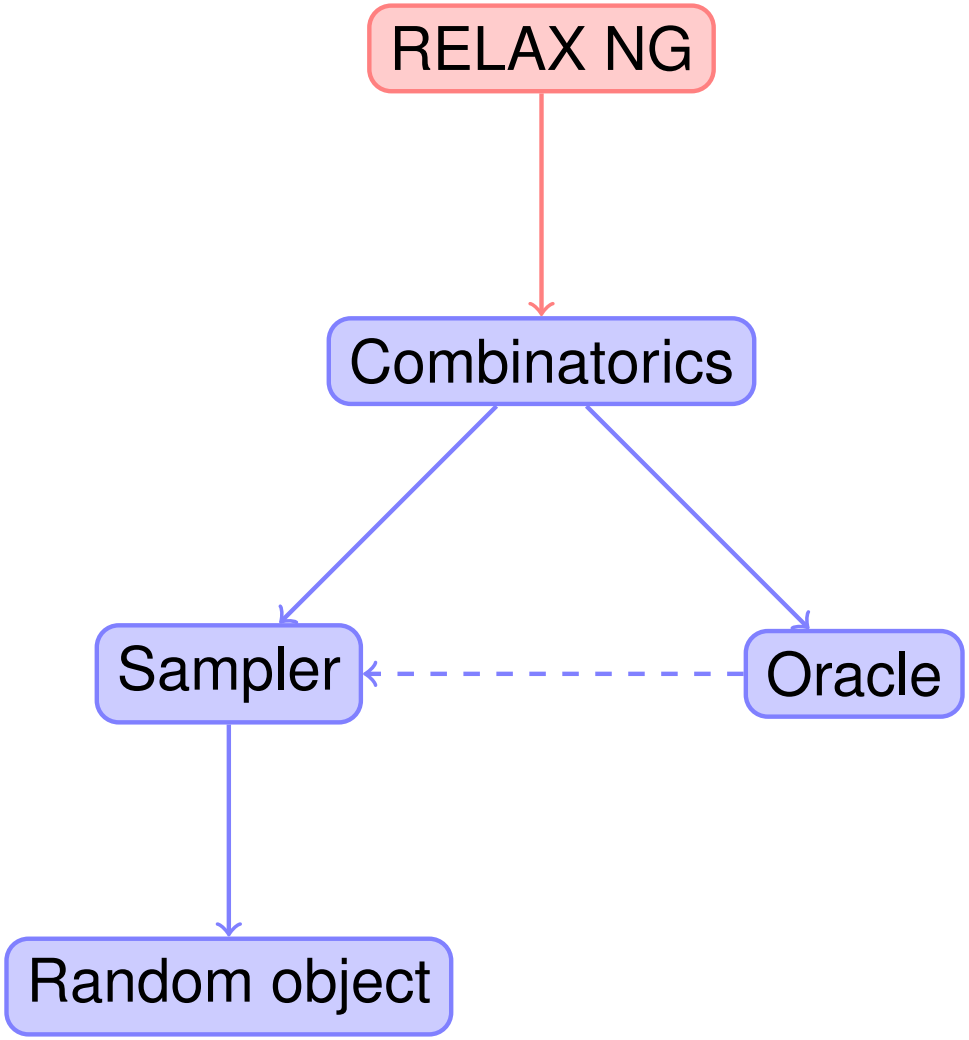
1 ILP 1 document of size $>$ 100000 (61MB)

on a Core 2 Duo with 1GB of RAM

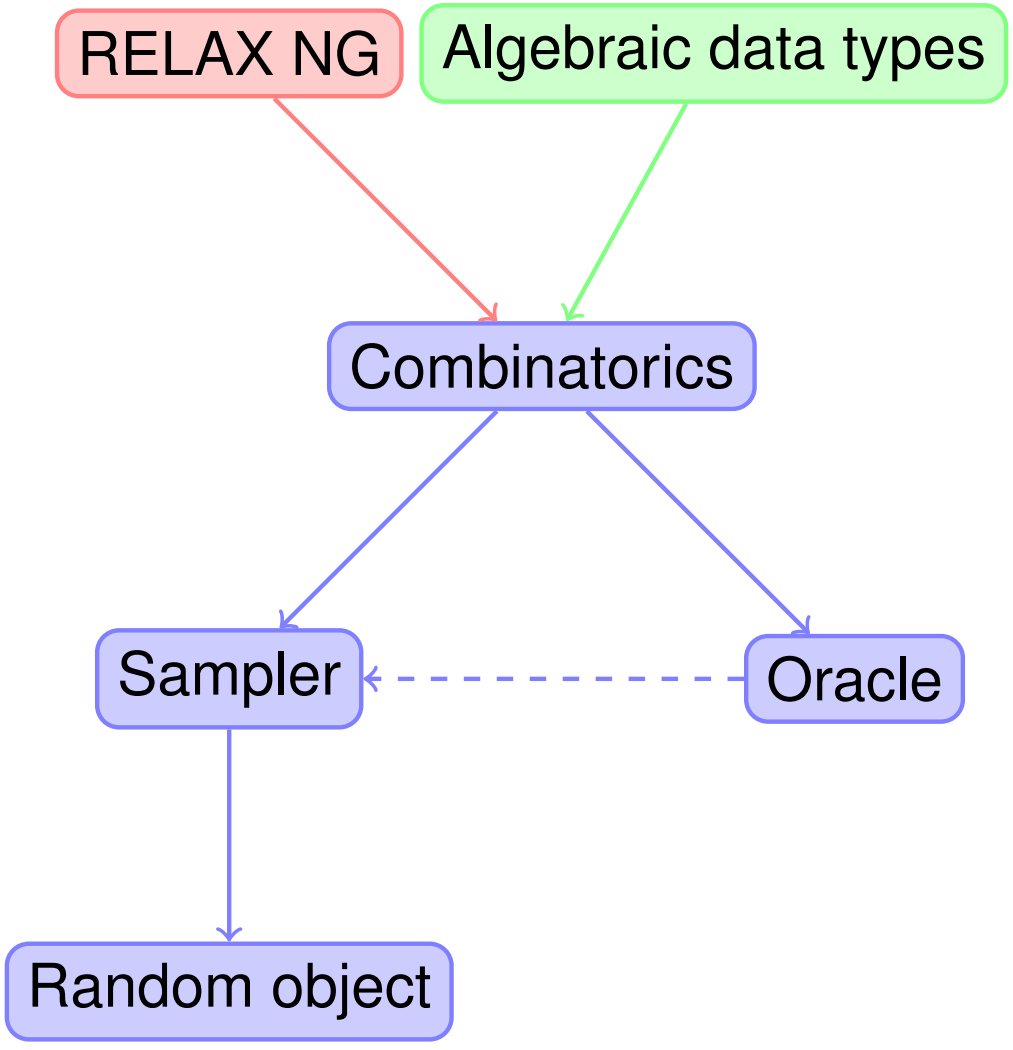
Work in progress

- Namespaces not yet implemented
- `interleave` not yet treated correctly
- Attributes should be sets, not sequences
- Samplers for `data` must be provided by the user
- The document is not serialized until the sampling is over

A generic framework



A generic framework



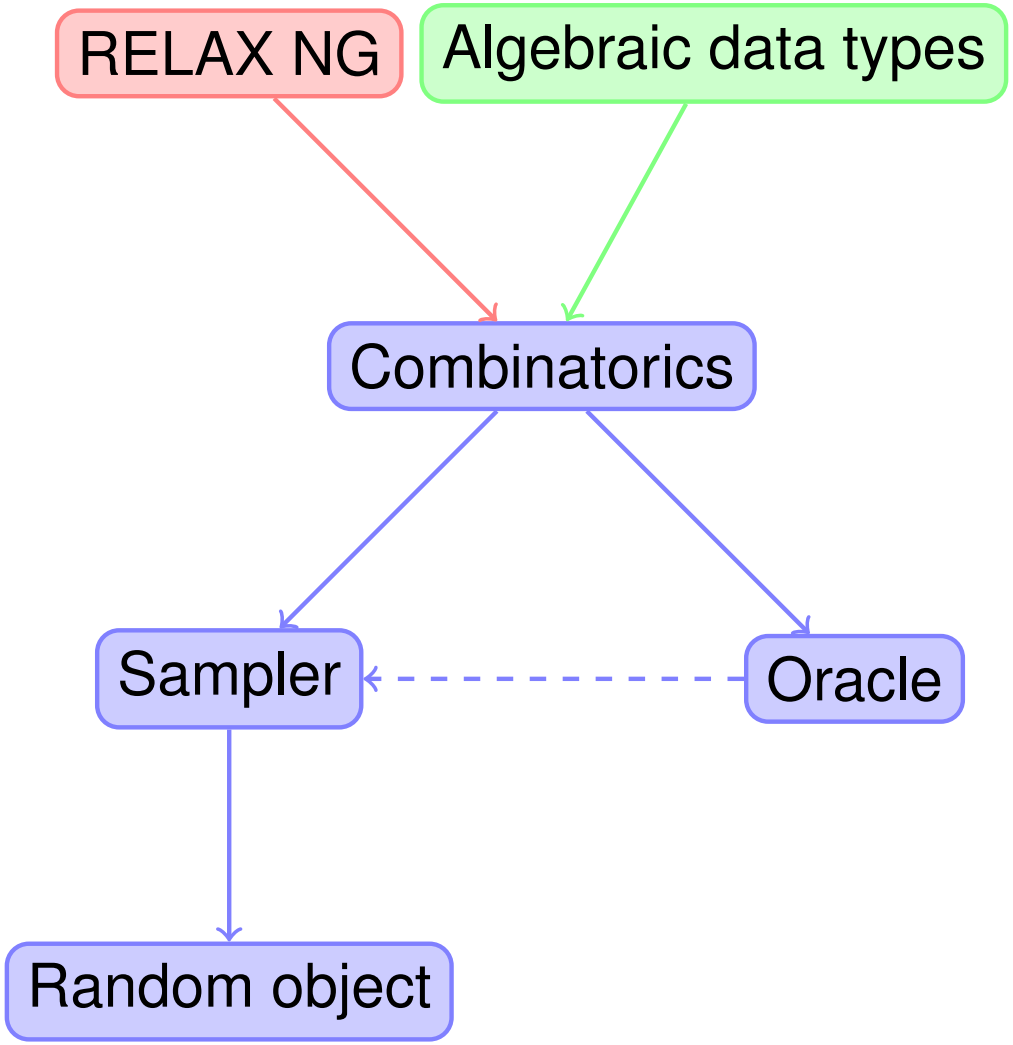
Our prototypes

- Extension for algebraic datatypes in QuickCheck (Haskell)
- Algebraic datatype sampler for O'Caml (with B. Canou)

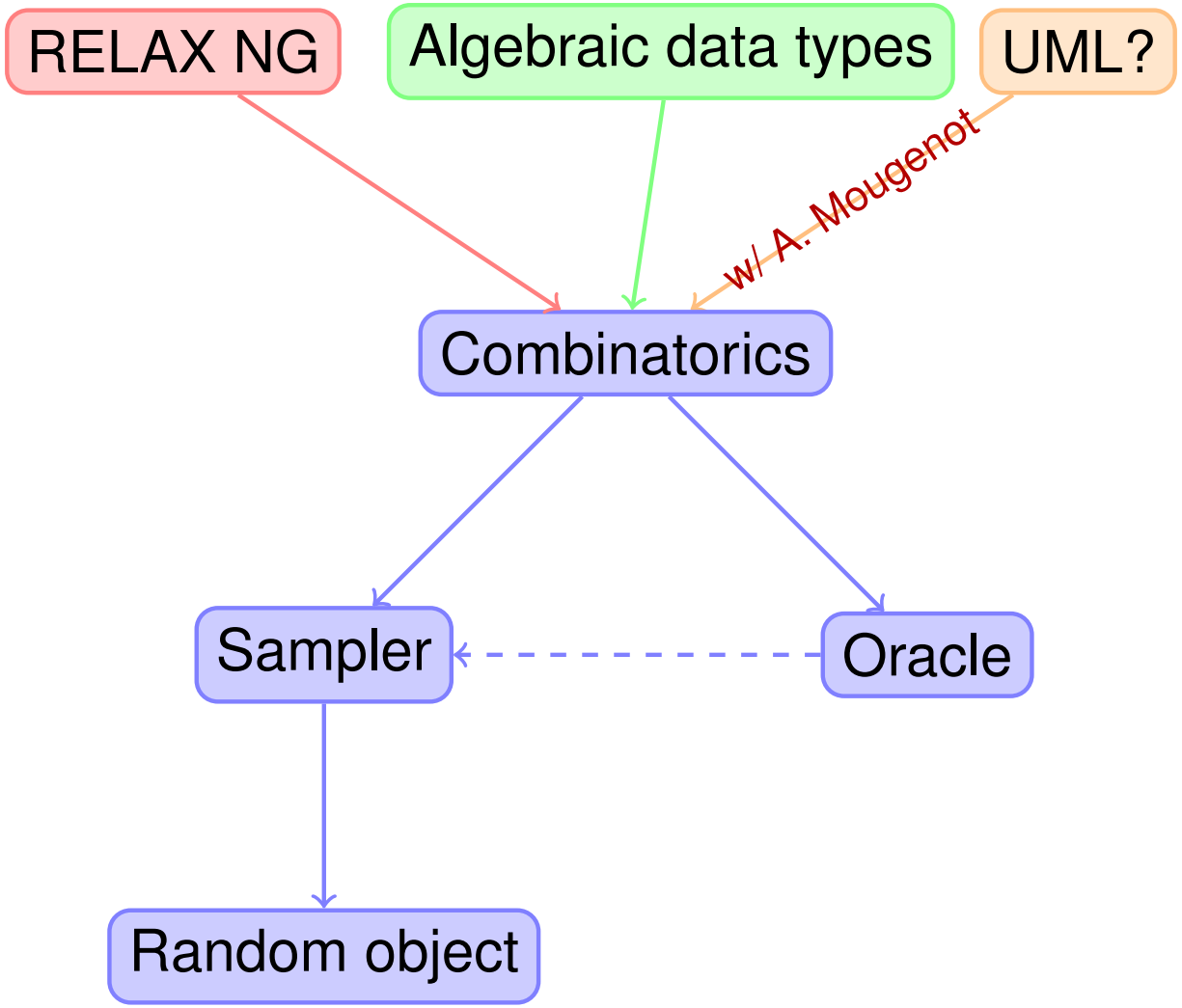
```
data Prog = Prog Dec Stat deriving (Typeable, Data, Show)
data Dec  = Nodec | Ondec Id Type | Manydecs Dec Dec deriving (Typeable, Data, Show)
data Id   = A | B deriving (Typeable, Data, Show)
data Type = Int | Bool deriving (Typeable, Data, Show)
data Stat = Noop | Assign Id Exp | Seq Stat Stat deriving (Typeable, Data, Show)
data Exp  = Zero | Succ Exp deriving (Typeable, Data, Show)

main = do gen <- getStdGen
          g <- toGenIO (undefined::Prog)
          putStrLn . show $ generate 1 gen g
```

```
$ ./prog
Prog (Manydecs (Manydecs (Manydecs Nodec Nodec) (Manydecs Nodec (Manydecs Nodec
(Manydecs (Manydecs (Manydecs (Ondec B Bool) (Manydecs (Manydecs (Manydecs (Manydecs
(Manydecs (Ondec B Int) (Manydecs (Ondec A Int) (Manydecs (Manydecs (Ondec A Bool)
(Manydecs (Ondec B Int) Nodec)) (Manydecs Nodec (Ondec A Int))))) (Ondec A Int))
(Manydecs (Manydecs (Manydecs (Manydecs (Manydecs Nodec (Manydecs Nodec (Manydecs
(Manydecs (Manydecs (Ondec A Int) Nodec) Nodec) Nodec))) (Manydecs (Manydecs
Nodec Nodec) Nodec)) (Manydecs Nodec (Manydecs (Manydecs (Manydecs (Manydecs
(Ondec A Bool) (Manydecs (Manydecs Nodec (Manydecs (Ondec B Int) (Ondec A
Bool))) Nodec)) (Manydecs (Manydecs (Manydecs (Ondec B Bool) Nodec) (Ondec B
Bool)) (Manydecs (Ondec A Int) (Manydecs (Ondec B Bool) (Ondec A Bool)))))
Nodec) Nodec))) (Ondec B Int)) (Manydecs Nodec (Manydecs (Manydecs (Manydecs
(Manydecs (Ondec B Int) Nodec) (Manydecs Nodec (Manydecs Nodec (Manydecs
(Manydecs (Manydecs (Manydecs (Manydecs (Manydecs (Ondec A Int) Nodec) Nodec)
Nodec) Nodec) (Manydecs (Manydecs (Manydecs Nodec (Ondec B Bool)) Nodec)
```

A generic framework



A generic framework



A generic framework