COUVERTURE
La certification libre

UPMC PARISUNIVERSITAS

LIP6

cnrs dépasser les frontières

Couverture non-intrusive de code Objective Caml

http://www.algo-prog.info/zamcov/

Groupe de Travail “Programmation”
Laboratoire PPS
15 Avril 2010

Emmanuel Chailloux,
Adrien Jonquet, Alexis Darrasse,
Mathias Bourgoin, **Philippe Wang**

Plan

- Motivations
- Couverture de code OCaml
- Machine Virtuelle OCaml en OCaml
- Génération de rapports de couverture
  à l'aide d'informations (statiques) pour le *debug*
- Génération de rapports de couverture
  avec mesure MC/DC
- Conclusion

COUVERTURE
La certification libre

Projet «Couverture»

http://www.projet-couverture.com

- logiciels critiques
  ne doivent pas mettre des vies en danger
- normes de développement, traçabilité, tests
  DO-178B, IEC 61508, FDA Class III, ...
- projet «couverture»
  - *logiciels **libres** pour le développement de logiciels **critiques***
  - ***couverture** de code **non-intrusive***

SYSTEM@TIC PARIS-REGION Pôle de compétitivité
AdaCore The GNAT Pro Company
LIP6
Open Wide Architecte Open Source
TELECOM ParisTech

Traçabilité et Tests

DO-178B
EN 50128
IEC 61508
IEC 60880
FDA Class III
etc.

- Le logiciel ne doit pas mettre de vies en danger
- Traçabilité entre les différentes étapes du développement logiciel
- Montrer l'équivalence entre les spécifications de haut-niveau et le code de bas-niveau
- Le logiciel doit être testé !

Les activités de tests

- Test fonctionnel
  - comportement du programme conforme aux attentes
  - robustesse du programme
- Test de couverture de code :
  analyse d'activation de code
  - couverture structurelle
  - couverture décisionnelle
  - couverture conditionnelle
  - MC/DC (*modified condition/decision coverage*)

Les activités de tests

- Test fonctionnel
  - comportement du programme conforme aux attentes
  - robustesse du programme
- Test de couverture de code :
  analyse d'activation de code
  - couverture structurelle
  - couverture décisionnelle
  - couverture conditionnelle
  - MC/DC (*modified condition/decision coverage*)

Les tests

- Test fonct
  - comport orme aux attentes
  - robustes

définitions

décision
expression booléenne

condition
expression booléenne
qui ne contient pas
de sous-expression booléenne

- Test de couverture de code :
  analyse d'activation de code
  - couverture structurelle
  - couverture décisionnelle
  - couverture conditionnelle
  - MC/DC (*modified condition/decision coverage*)

Mlcov

- outil libre de couverture de code OCaml
- développé par Esterel Technologies
  (prototype développé à PPS)
- technique : instrumentation du code source
  - branchement sur le *frontend* OCaml pour réécriture du programme avec injection d'instructions de génération de traces d'exécution
    - le binaire généré est plus gros...

couverture de code

Mlcov

technique de réécriture du programme source,
par l'exemple, sur un mini langage,
en injectant des instructions de génération de traces

intrusive instrumentation for a tiny language

$$
\begin{aligned}
\mathcal{I}(\texttt{atom}) &= \textit{mark}();\ \texttt{atom} \\
\mathcal{I}(\textbf{let}\ \texttt{v} = \texttt{e1}\ \textbf{in}\ \texttt{e2}) &= \textbf{let}\ \texttt{v} = \mathcal{I}(\texttt{e1})\ \textbf{in}\ \mathcal{I}(\texttt{e2}) \\
\mathcal{I}(\texttt{f}\ \texttt{x}) &= \textbf{let}\ \texttt{r} = \mathcal{I}(\texttt{f})\ \mathcal{I}(\texttt{x})\ \textbf{in}\ \textit{mark}();\ \texttt{r} \\
\mathcal{I}(\textbf{fun}\ \texttt{v} \texttt{->}\ \texttt{e}) &= \textbf{fun}\ \texttt{v} \texttt{->}\ \mathcal{I}(\texttt{e}) \\
\mathcal{I}(\textbf{if}\ \texttt{e1}\ \textbf{then}\ \texttt{e2}\ \textbf{else}\ \texttt{e3}) &= \textbf{if}\ \mathcal{I}(\texttt{e1})\ \textbf{then}\ \mathcal{I}(\texttt{e2})\ \textbf{else}\ \mathcal{I}(\texttt{e3})
\end{aligned}
$$

couverture de code

Mlcov

exemple

de rapport pour un code OCaml
(slide suivant)

exemple

de rapport pour un code OCaml

MLcov — Source Code Report (file queens.ml)

http://www.algo-prog.info/mlcov/

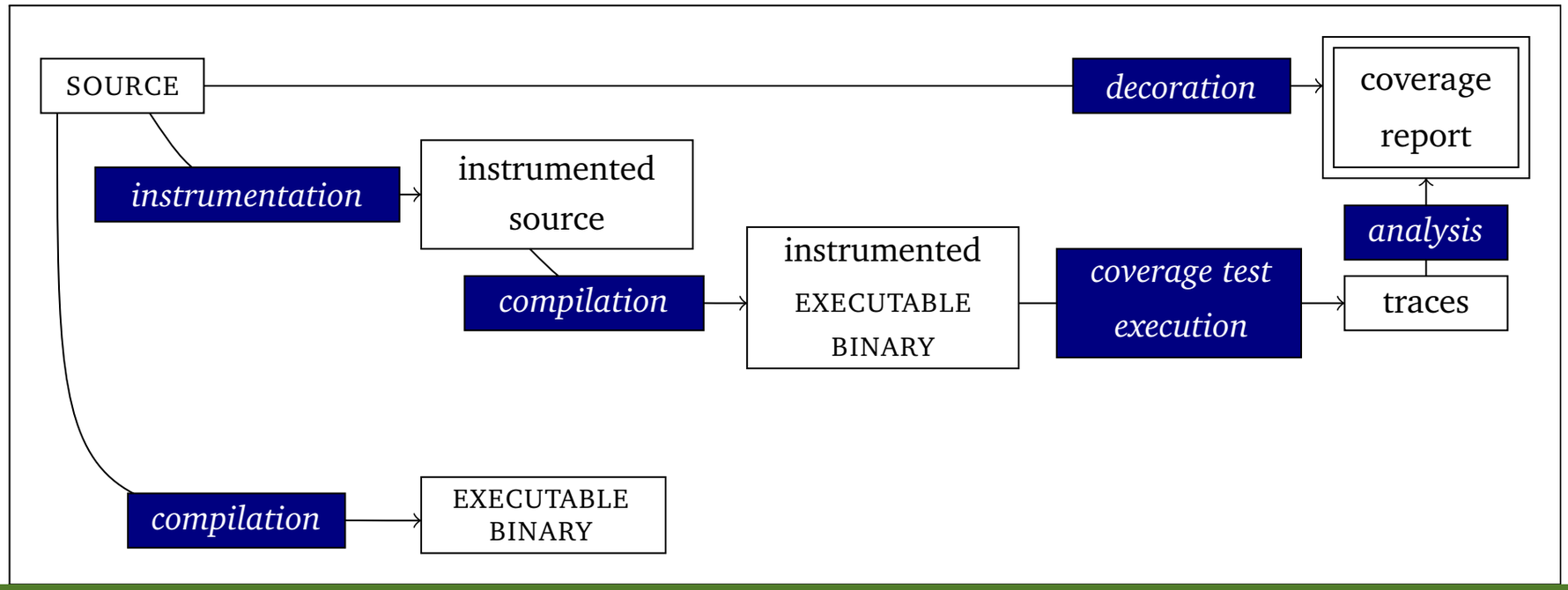
```
(* Benchmark on list allocation and manipulation *)

let rec append l1 l2 = match l1 with
  [] -> l2
| a::q -> a::(append q l2);;

let rec map f l = match l with [] -> [] | h::t -> (f h)::(map f t);;

let rec iter f l = match l with [] -> () | h::t -> f h ; iter f t ;;

let rec interval n m =
    if n > m then [] else (n :: interval (succ n) m);;

let rec concmap f = function
    [] -> []
  | x :: l -> append (f x) (concmap f l ) (*f x @ concmap f l*);;

let rec list_length = function
    [] -> 0
  | _::l -> 1 + list_length l;;

let rec safe d x = function
    [] -> true
  | q::l -> (not (x = q)) & ((not (x = q+d)) & ((not (x = q-d)) &
            safe (d+1) x l));;

let ok = function [] -> true | x::l -> safe 1 x l;;

let rec filter p = function
   [] -> []
 | x::l -> if p x then x::filter p l else filter p l;;
let range = interval 1;;

let queens n =
 let qs = range n in
 let testcol = function b -> filter ok (map (fun q -> q::b) qs) in
 let rec gen = function
    0 -> [[]]
  | n -> print_string "\n** : ";  print_int n;print_newline();
           let r = concmap testcol (gen (n - 1)) in
           (*print_bll r;*) r  in
 let r =  (gen n) in
   print_string "nb sols ";  print_int (list_length r); print_newline();;

queens 5 ;;
```

Couverture non-intrusive

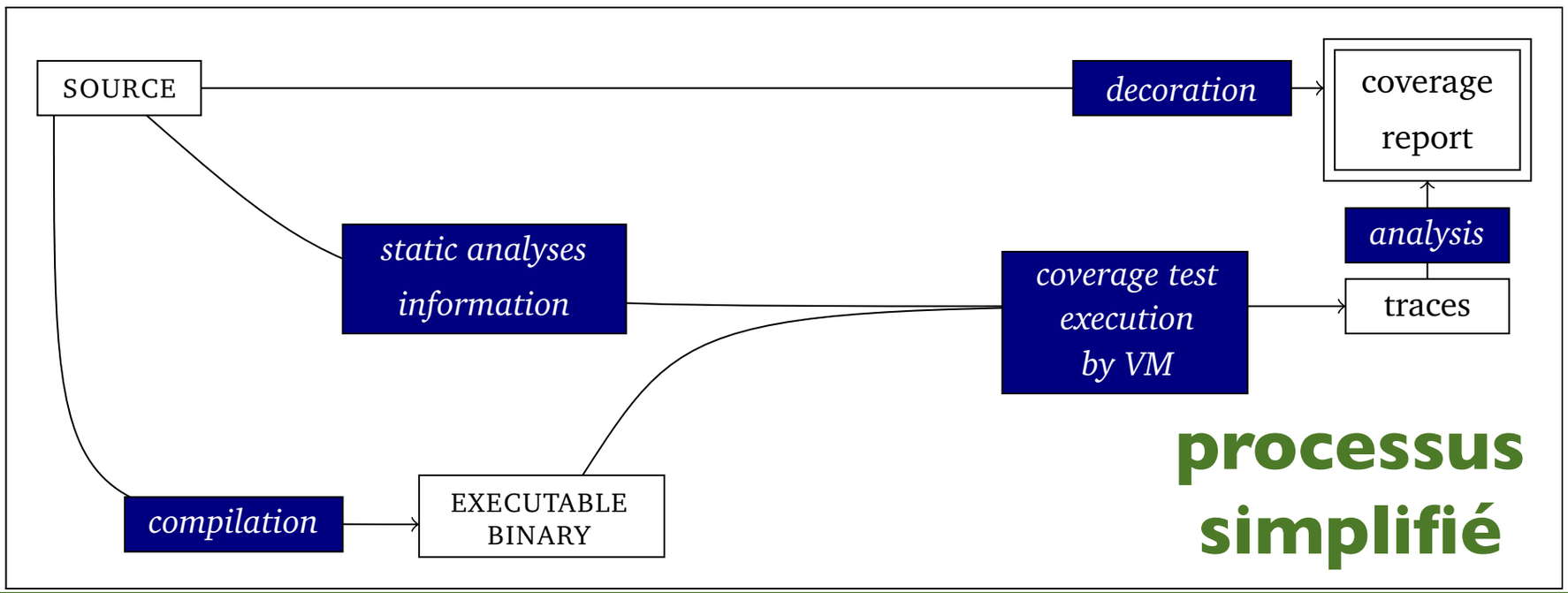
- **Ne pas changer le code source**
- Changer l'environnement d'exécution
- Générer des traces d'exécution
- Analyse des traces et génération de rapports

Comparaison

Intrusive code coverage obtention method



Non-Intrusive code coverage obtention method



Implantation

- Ingrédients
  - Langage source : OCaml
  - Environnement d'exécution : OCaml VM
- Réalisations
  - OCaml VM en OCaml
  - Greffon de génération de traces d'exécution

Implantation

- Ingrédients
  - Langage source : OCaml
  - Environnement d'exécution : OCaml VM
- Réalisations
  - OCaml VM en OCaml
  - Greffon de génération de traces d'exécution

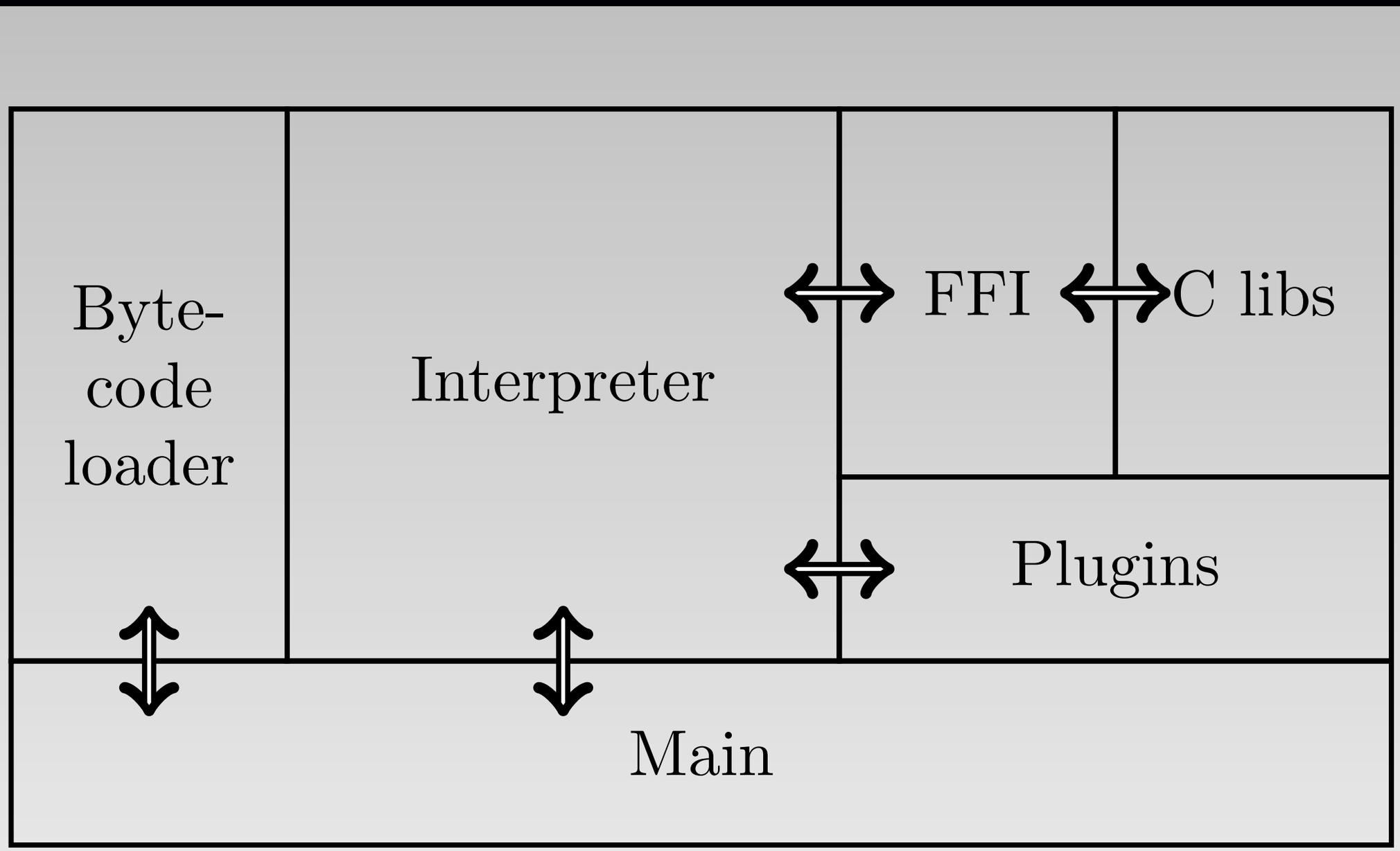
OCaml VM en OCaml

- Caractéristiques du bytecode
  - non typé, pas de vérification de type à l'exécution
  - 146 instructions
- Caractéristiques de la machine
  - machine fonctionnelle à pile
  - ~ 3300 loc (interprète, bibliothèque d'exécution)
  - compilé en bytecode avec `ocamlc`
  - ou compilé en code natif avec `ocamlopt`

→ **Prêt pour le bootstrap**

OCaml VM en OCaml

architecture de `zamcov-run`



VMs OCaml

Plusieurs implémentations de VM OCaml

OCAMLRUN
C
(X. Leroy)

(g)cc / many archs

CADMIUM
JAVA
(X. Clerc)

javac / JVM / gcj /
many archs

O'Browser
JavaScript
(B. Canou)

javascript /
firefox et al.

ZAMCOV
OCaml

ocamlc / ocamlrun
ocamlopt / many archs

Benchmarks

(slowdowns)

	ocamlrun	zamcov	cadmium
Knuth-Bendix	0.51s (x1)	4.64s (x9.1)	24.70s (x48.4)
Peg Solitaire	0.46s (x1)	3.59s (x7.8)	7.02s (x15.3)
Nucleic	0.55s (x1)	3.19s (x5.8)	89.23s (x162)
`ocamlc oAvl.ml`	0.44s (x1)	7.18s (x16.3)	26.30s (x59.8)

ocamlrun — implantation en C

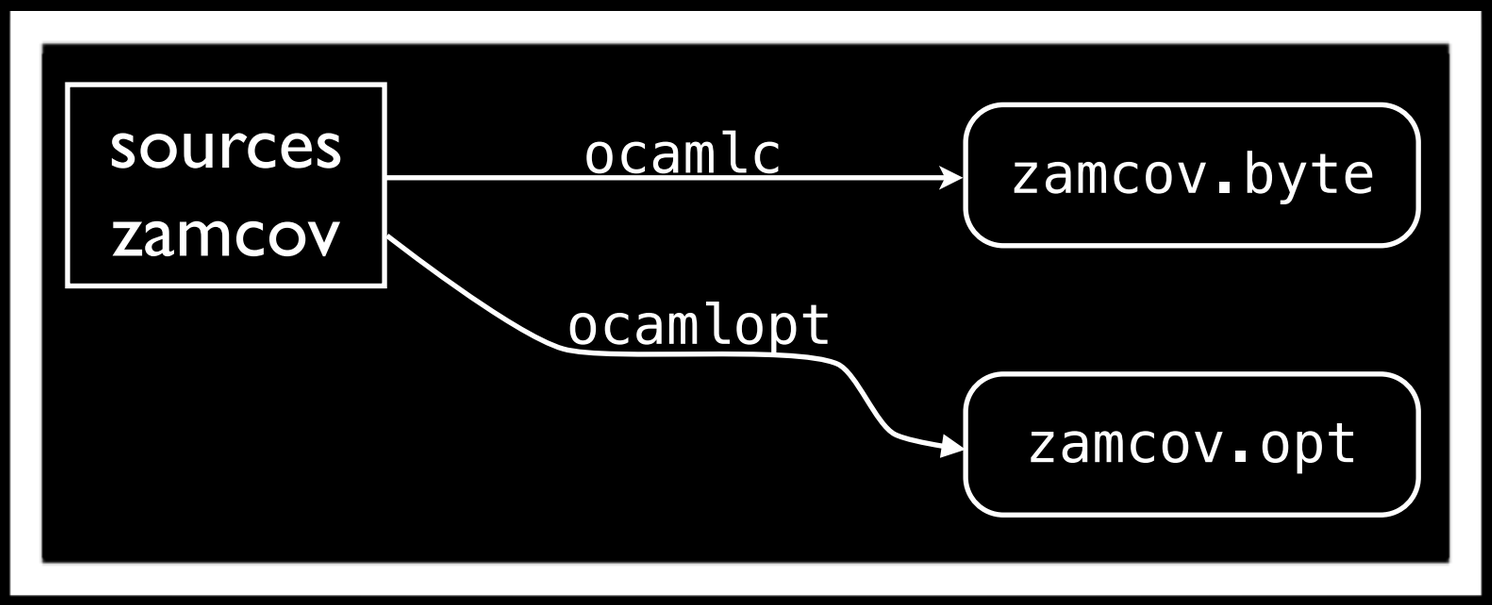
zamcov — implantation en OCaml (`ocamlopt`)

cadmium — implantation en Java (Sun JVM)

“foreign function interface”

“Bootstrap” & F.F.I.

- On dispose d’une OCaml VM en OCaml
- On peut donc imaginer l’interpréter par n’importe OCaml VM, pour peu qu’on la compile en bytecode... :-)
  Voyons ce que ça donne...



zamcov
interprète n’importe quel programme OCaml compilé par `ocamlc`, dont `zamcov.byte`, pour peu que l’interfaçage avec les bibliothèques soit complet

F.F.I.

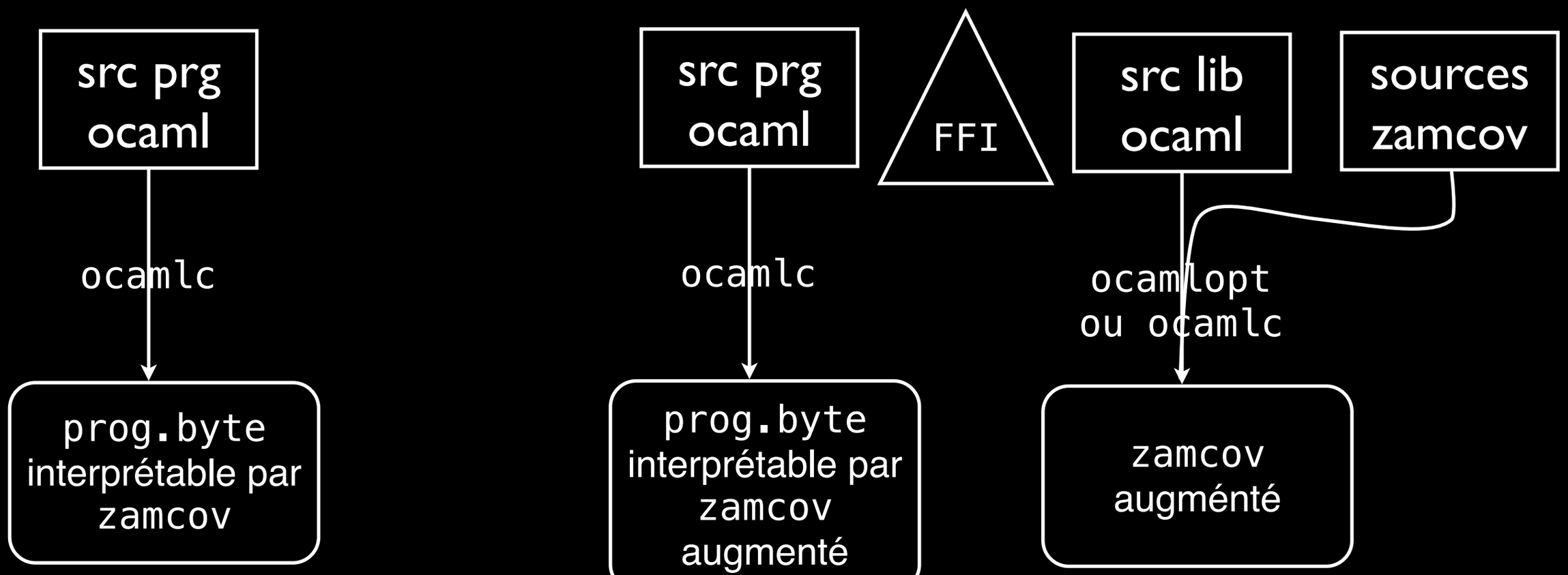
`zamcov` *high-level view*

- impl. : OCaml
- runtime lib. : OCaml
- `VM CCALL` ~> OCaml call
- foreign : OCaml

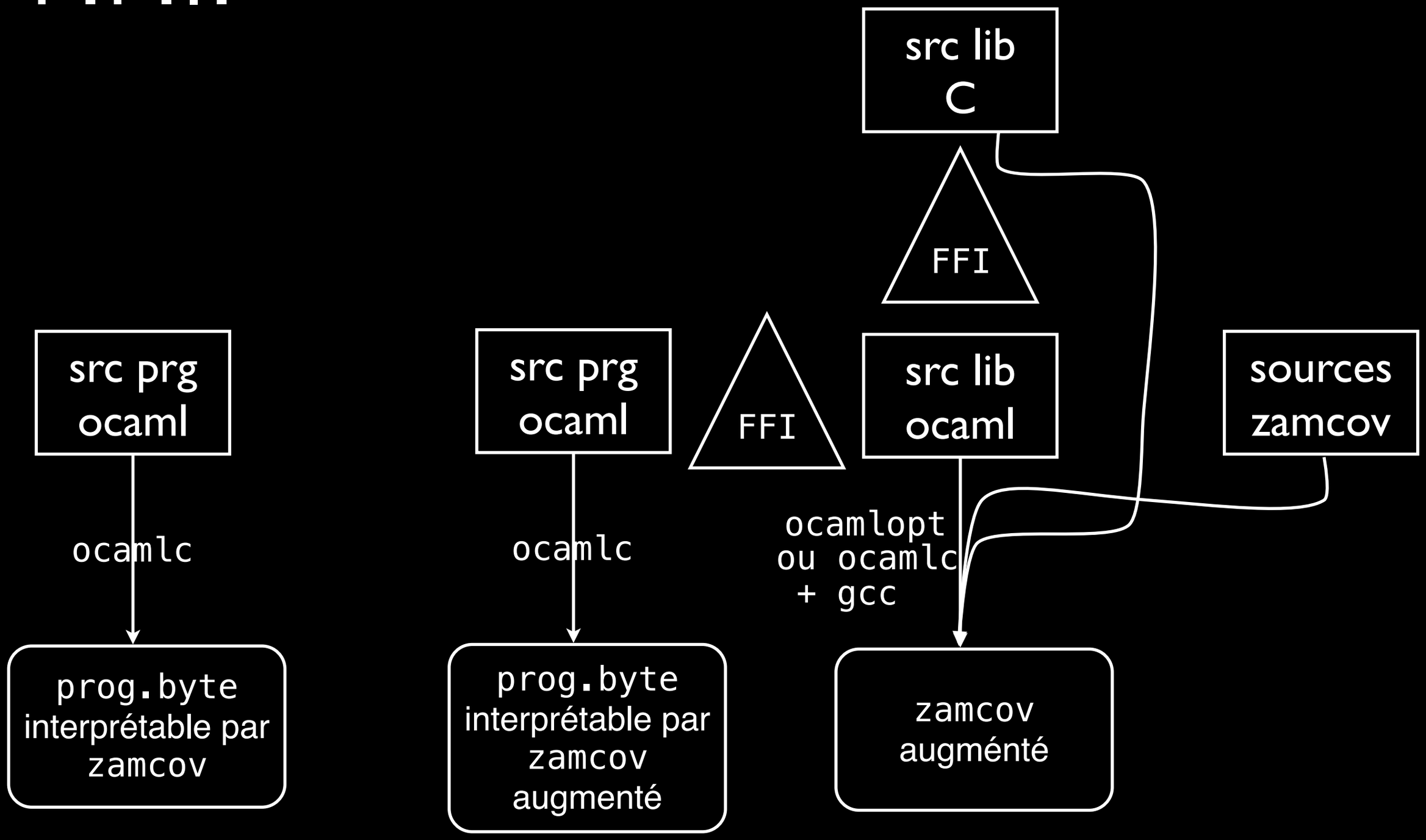
`zamcov.byte` *low-level view* `zamcov.opt`

- impl. : OCaml
- runtime lib. : C
- impl. : OCaml
- runtime lib. : C + ASM

F.F.I.


src prg
ocaml
ocamlc
prog.byte
interprétable par
zamcov
src prg
ocaml
ocamlc
prog.byte
interprétable par
zamcov
augmenté
FFI
src lib
ocaml
sources
zamcov
ocamlopt
ou ocamlc
zamcov
augménté

F.F.I.


src lib
C
FFI
src prg
ocaml
src prg
ocaml
FFI
src lib
ocaml
sources
zamcov
ocamlc
ocamlc
ocamlopt
ou ocamlc
+ gcc
prog.byte
interprétable par
zamcov
prog.byte
interprétable par
zamcov
augmenté
zamcov
augménté

Implantation

- Ingrédients
  - Langage source : OCaml
  - Environnement d'exécution : OCaml VM
- Réalisations
  - OCaml VM en OCaml
  - Greffon de génération de traces d'exécution

Couverture structurelle
du code machine

- Génération de la trace d'exécution
- Génération du rapport de couverture
  - désassemblage de l'exécutable
  - union des informations avec la trace d'exécution

```
00000000 BRANCH 14
00000002 CONST0
00000003 PUSHACC1
00000004 GTINT
00000005 BRANCHIFNOT 5
00000007 CONSTINT 42
00000009 RETURN 1
00000011 CONSTINT 69
00000013 RETURN 1
00000015 CLOSURE 0 -15
00000018 PUSH
00000019 CONST3
00000020 PUSHACC1
00000021 APPLY1
00000022 ACC0
00000023 MAKEBLOCK1 0
00000025 POP 1
00000027 SETGLOBAL 12
00000029 STOP
```

Couverture structurelle
du code machine

- Génération de la trace d'exécution
- Génération du rapport de couverture
  - désassemblage de l'exécutable
  - union des informations avec la trace d'exécution

```
external (>) : 'a -> 'a -> bool = "%greaterthan"

let pos x =
  if x > 0 then 42
  else 69
;;

pos 3;;
```

(compilé avec `-nopervasives` pour simplifier)

```
00000000 BRANCH 14
00000002 CONST0
00000003 PUSHACC1
00000004 GTINT
00000005 BRANCHIFNOT 5
00000007 CONSTINT 42
00000009 RETURN 1
00000011 CONSTINT 69
00000013 RETURN 1
00000015 CLOSURE 0 -15
00000018 PUSH
00000019 CONST3
00000020 PUSHACC1
00000021 APPLY1
00000022 ACC0
00000023 MAKEBLOCK1 0
00000025 POP 1
00000027 SETGLOBAL 12
00000029 STOP
```

Couverture structurelle
du code source

- Génération de la trace d'exécution
- Génération du rapport de couverture
  - besoin de faire le lien entre une expression dans le source et le bytecode
  - utilisation des informations de debug

les informations de debug sont utilisées par `ocamldebug`

Informations de debug

- Générées par le compilateur standard
- Placées dans une entête à l'extérieur du bytecode

  => technique **non-intrusive**
- Encadre une expression OCaml
- Une info de debug contient
  - l'adresse dans le bytecode de la **première** ou **dernière** instruction de l'expression
  - la position dans le source de l'expression

→ **liaison entre le bytecode et les sources**

Informations de debug

$$
\begin{aligned}
F(atom) &= atom \text{ (constant value or identifier)} \\
F(e_0\ e_1) &= (F(e_0))\ (F(e_1))\ \$\$ \\
F(\texttt{let}\ [\texttt{rec}]\ p\ =\ e_0\ \texttt{in}\ e_1) &= \texttt{let}\ [\texttt{rec}]\ p\ =\ F(e_0)\ \texttt{in}\ F(e_1) \\
F(\texttt{fun}\ p_0\ ...\ p_n\ \rightarrow\ e) &= \texttt{fun}\ p_0\ ...\ p_n\ \rightarrow\ \$\$\ F(e) \\
F(\texttt{function}\ P) &= \texttt{function}\ F(P) \\
F(\texttt{match}\ e\ \texttt{with}\ P) &= \texttt{match}\ F(e)\ \texttt{with}\ F(P) \\
F(\texttt{try}\ e\ \texttt{with}\ P) &= \texttt{try}\ F(e)\ \texttt{with}\ F(P) \\
F(P) &= F(p_i\ [\texttt{when}\ c_i] \rightarrow\ e_i) \\
F(p_i\ [\texttt{when}\ c_i]\ \rightarrow\ e_i) &= p_i\ [\texttt{when}\ F(c_i)\ ]\ \rightarrow\ \$\$F(e_i) \\
F(e_0;\ e_1) &= (F(e_0);\ \$\$\ F(e_1)) \\
F(\texttt{if}\ e_0\ \texttt{then}\ e_1\ [\texttt{else}\ e_2]) &= \texttt{if}\ F(e_0)\ \texttt{then}\ \$\$\ F(e_1)\ [\texttt{else}\ \$\$\ F(e_2)] \\
F(\texttt{while}\ e_0\ \texttt{do}\ e_1\ \texttt{done}) &= \texttt{while}\ F(e_0)\ \texttt{do}\ \$\$\ F(e_1)\ \texttt{done} \\
F(\texttt{for}\ i\ =\ e_0\ \texttt{to}\ e_1\ \texttt{do}\ e_2\ \texttt{done}) &= \texttt{for}\ i\ =\ F(e_0)\ \texttt{to}\ F(e_1)\ \texttt{do}\ \$\$\ F(e_2)\ \texttt{done} \\
F(e_0\#m\ [e_1...e_n]) &= F(e_0)\#m\ [(F(e_1))...(F(e_n))]\ \$\$ \\
&\text{where } m \text{ represents the name of a method}
\end{aligned}
$$

couverture de code

Zamcov

exemple

de rapport pour un code OCaml
utilisant les informations de debug
(slide suivant)

exemple
de rapport avec Zamcov

ZamCov: Expression Coverage (queens.ml)

http://www.algo-prog.info/zamcov/

```
(* Benchmark on list allocation and manipulation *)

let rec append l1 l2 = match l1 with
  [] -> l2
| a::q -> a::(append q l2);;

let rec map f l = match l with [] -> [] | h::t -> (f h)::(map f t);;

let rec iter f l = match l with [] -> () | h::t -> f h ; iter f t ;;

let rec interval n m =
    if n > m then [] else (n :: interval (succ n) m);;

let rec concmap f = function
    [] -> []
  | x :: l -> append (f x) (concmap f l ) (*f x @ concmap f l*);;

let rec list_length = function
    [] -> 0
  | _::l -> 1 + list_length l;;

let rec safe d x = function
    [] -> true
  | q::l -> (not (x = q)) & ((not (x = q+d)) & ((not (x = q-d)) &
            safe (d+1) x l));;

let ok = function [] -> true | x::l -> safe 1 x l;;

let rec filter p = function
   [] -> []
 | x::l -> if p x then x::filter p l else filter p l;;
let range = interval 1;;

let queens n =
 let qs = range n in
 let testcol = function b -> filter ok (map (fun q -> q::b) qs) in
 let rec gen = function
    0 -> [[]]
  | n -> print_string "\n** : ";  print_int n;print_newline();
           let r = concmap testcol (gen (n - 1)) in
           (*print_bll r;*) r  in
 let r =  (gen n) in
   print_string "nb sols ";  print_int (list_length r); print_newline();;

queens 5 ;;
```

Informations de debug

- Permet de générer un rapport de couverture structurelle équivalent à mlcov
- Ne permet pas de traiter la couverture MC/DC
  - pas d'identification des expressions booléennes
  - manque d'informations pour identifier les conditions

→ Génération de nouvelles informations

Couverture MC/DC

"Modified Condition/Decision Coverage"

- Motivation 1 : on veut activer toutes les conditions à `true` et à `false`
- Motivation 2 : on veut que les sous-conditions soient assez indépendantes
- Motivation 3 : "pas possible" de faire 2^n tests

=>

- Technique/Compromis de test pour les expressions booléennes complexes (i.e., composées avec &&, ||, not)
- *n+1* tests au lieu de 2^n tests

Couverture MC/DC

“Modified Condition/Decision Coverage”

exemple

```
let all_positive1 a b c =
   (a > 0) && (b > 0) && (c > 0) ;;

    (* all_positive1 1 1 1 ;; *)
    (* all_positive1 1 1 0 ;; *)

let all_positive2 a b c =
   (a > 0) && (b > 0) && (c > 0) ;;

    (* all_positive2 1 1 1 ;; *)
    (* all_positive2 1 0 1 ;; *)
    (* all_positive2 1 1 0 ;; *)
let all_positive3 a b c =
   (a > 0) && (b > 0) && (c > 0) ;;

    (* all_positive3 1 1 1 ;; *)
    (* all_positive3 0 1 1 ;; *)
    (* all_positive3 1 0 1 ;; *)
    (* all_positive3 1 1 0 ;; *)
```

Couverture structurelle

avec “informations de couverture” pour MC/DC

- Modification du générateur de code
  pour générer de nouvelles informations
  de liaison entre code source et code machine
- 3 tags (Statement, Decision, Condition) pour reconnaître les conditions et les décisions
- Une condition connaît la liste des expressions parentes immédiates booléennes

```
( a || (let x = b && c in not x) )
```

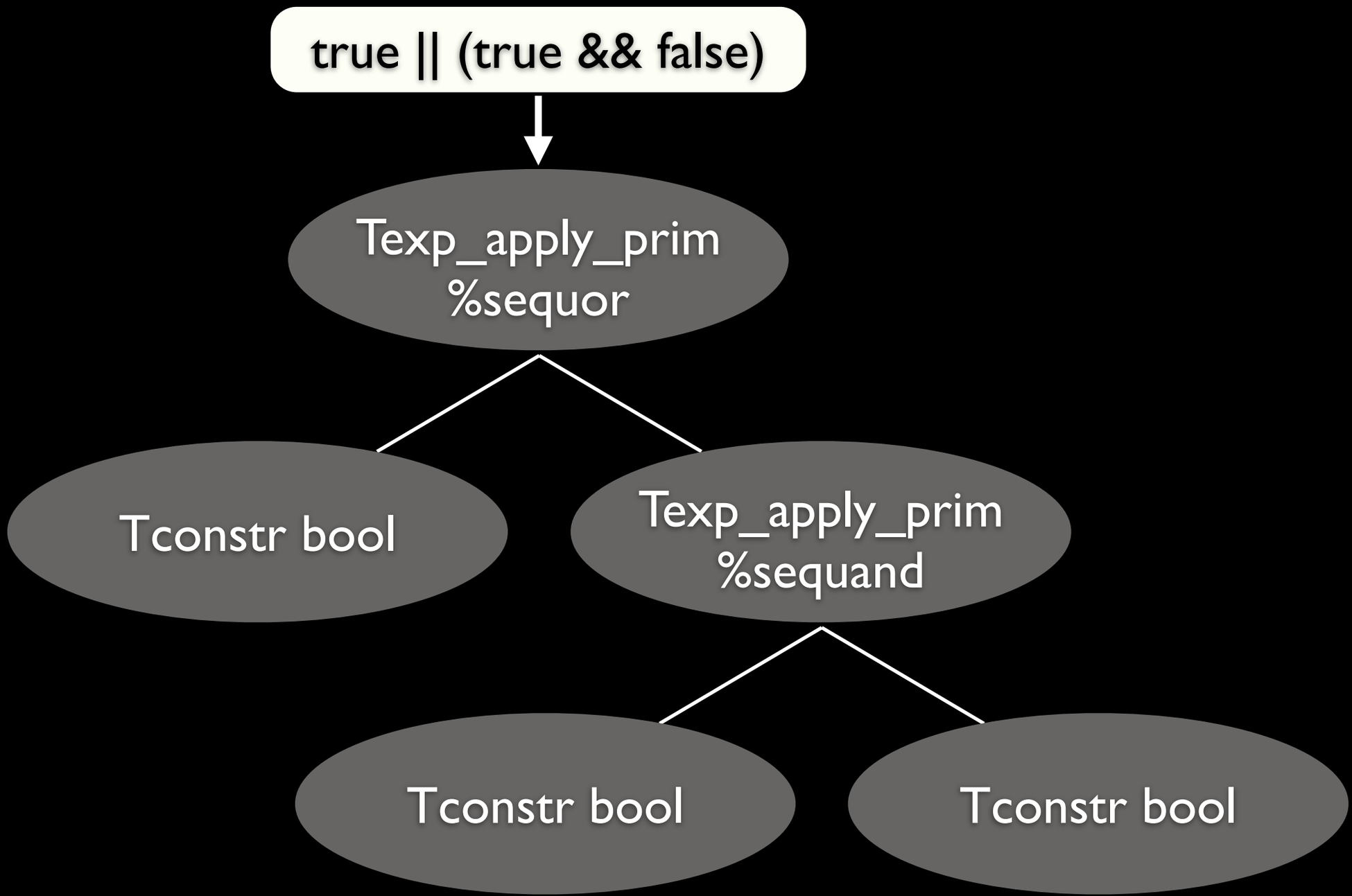
Informations de couverture

$$
\begin{array}{rcl}
F(constant) & = & \$\$constant \\
F(identifier) & = & \$\$identifier \\
F(construct) & = & \$\$construct \\
F(e_0\ e_1) & = & (F(e_0))\ (F(e_1))\ \$\$ \\
F(\texttt{let}\ [\texttt{rec}]\ p\ =\ e_0\ \texttt{in}\ e_1) & = & \texttt{let}\ [\texttt{rec}]\ p\ =\ F(e_0)\ \texttt{in}\ F(e_1) \\
F(\texttt{fun}\ p_0\ ...\ p_n\ \rightarrow\ e) & = & \texttt{fun}\ p_0\ ...\ p_n\ \rightarrow\ F(e) \\
F(\texttt{function}\ P) & = & \texttt{function}\ G(P) \\
F(\texttt{match}\ e\ \texttt{with}\ P) & = & \texttt{match}\ F(e)\ \texttt{with}\ G(P) \\
F(\texttt{try}\ e\ \texttt{with}\ P) & = & \texttt{try}\ F(e)\ \texttt{with}\ G(P) \\
F(e_0;\ e_1) & = & (F(e_0);\ F(e_1)) \\
F(\texttt{if}\ e_0\ \texttt{then}\ e_1\ [\texttt{else}\ e_2]) & = & \texttt{if}\ F(e_0)\ \texttt{then}\ F(e_1)\ [\texttt{else}\ F(e_2)] \\
F(\texttt{while}\ e_0\ \texttt{do}\ e_1\ \texttt{done}) & = & \texttt{while}\ F(e_0)\ \texttt{do}\ F(e_1)\ \texttt{done} \\
F(\texttt{for}\ i\ =\ e_0\ \texttt{to}\ e_1\ \texttt{do}\ e_2\ \texttt{done}) & = & \texttt{for}\ i\ =\ F(e_0)\ \texttt{to}\ F(e_1)\ \texttt{do}\ F(e_2)\ \texttt{done} \\
F(e_0\#m) & = & F(e_0)\#m\ \$\$ \\
\text{where} & m & \text{represents the name of a method} \\
G(P) & = & p_i\ [\texttt{when}\ F(c_i)\ ]\ \rightarrow\ F(e_i)
\end{array}
$$

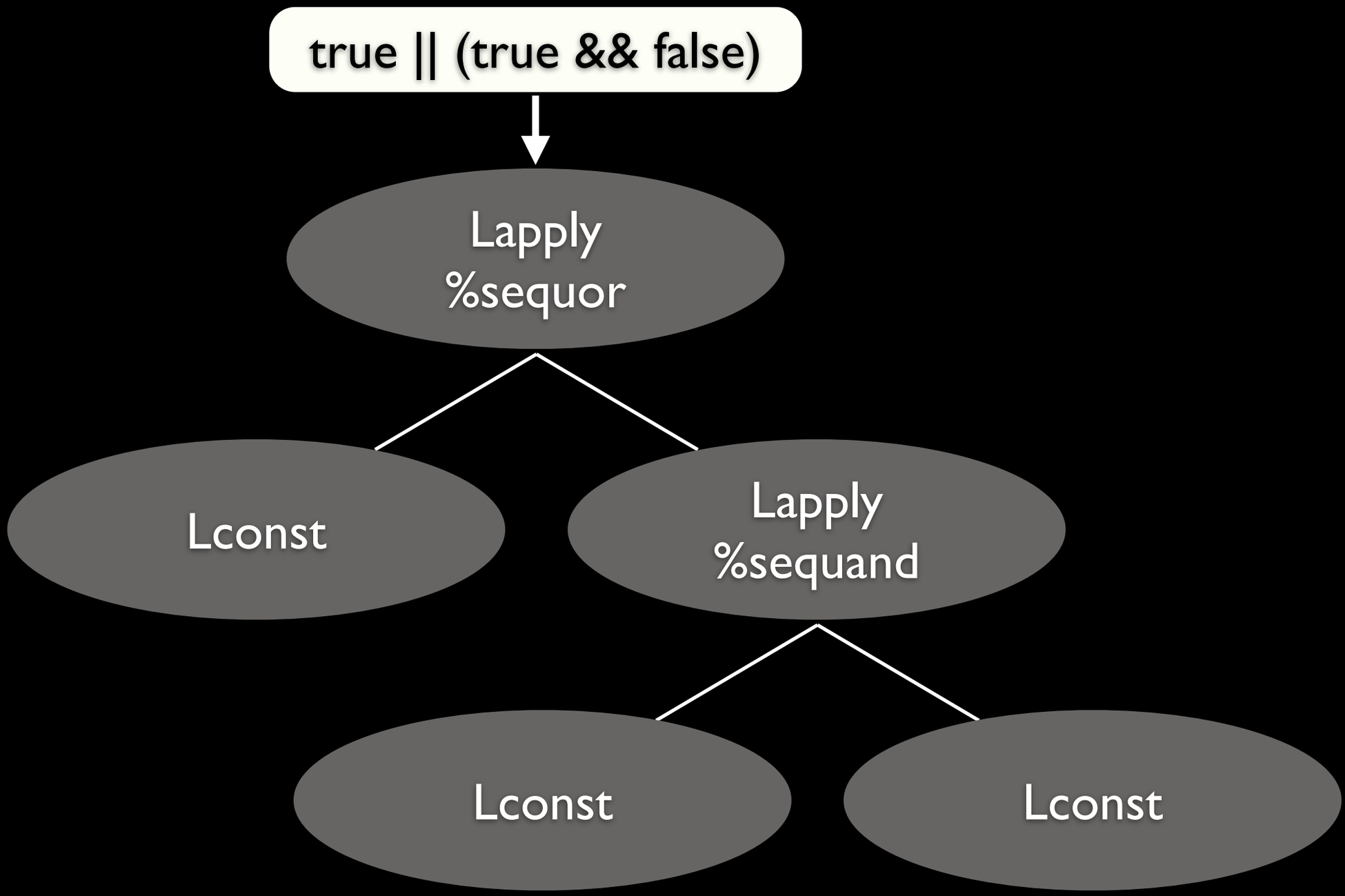
Etapes d'ajout des informations de couverture

Syntaxe abstraite typée
~> lambda-termes

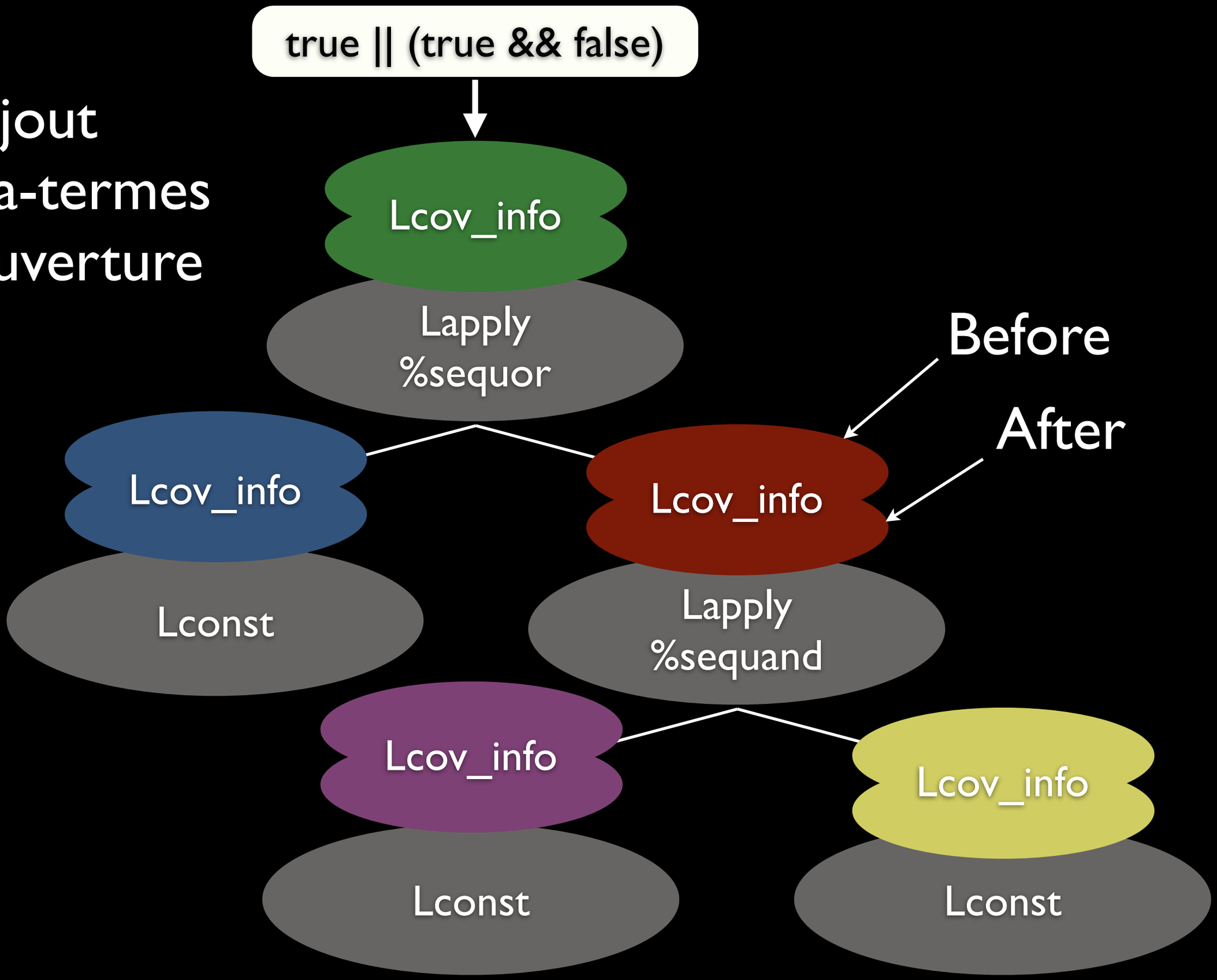
translcore.ml

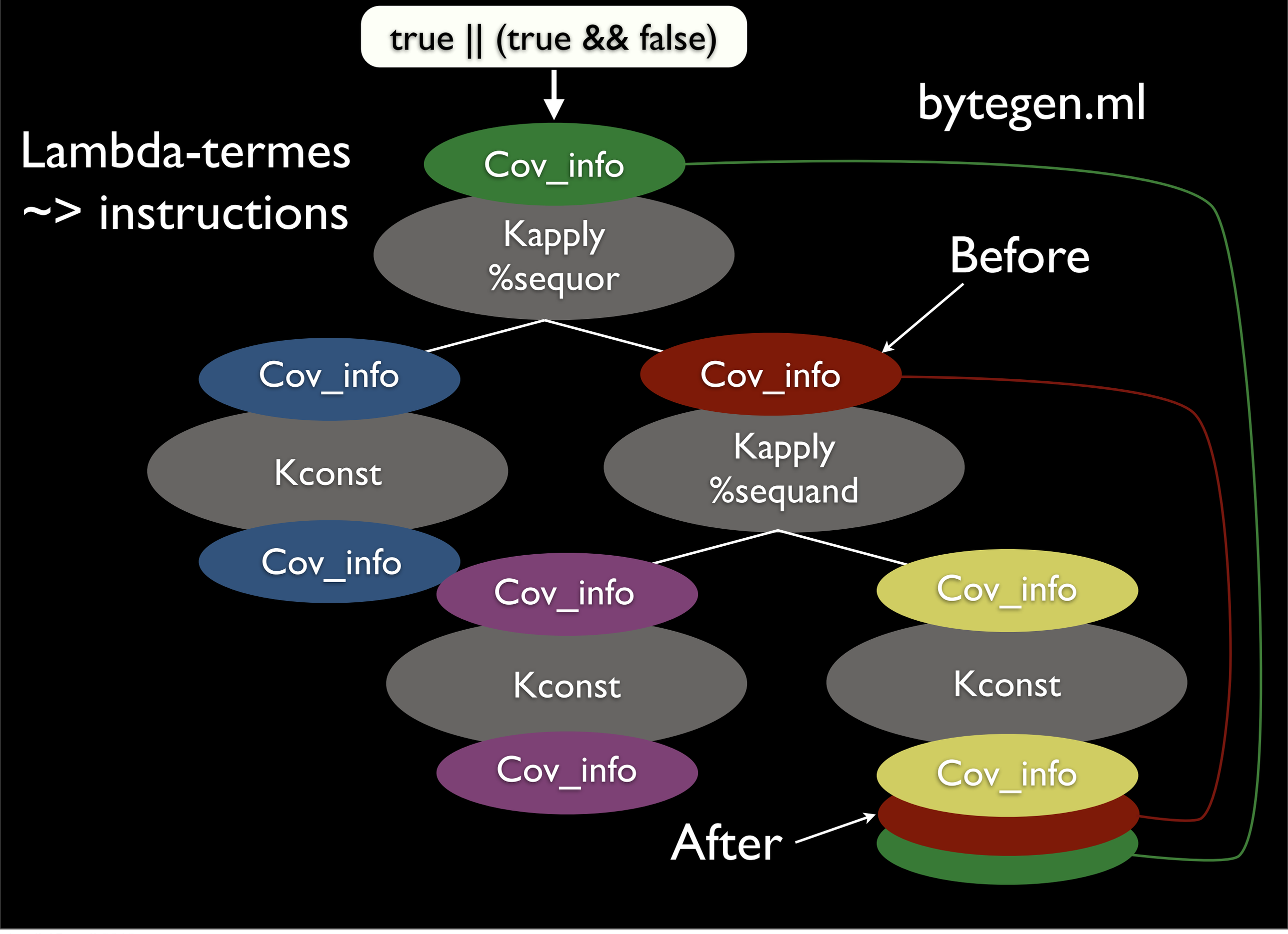


Arbre des lambda-termes

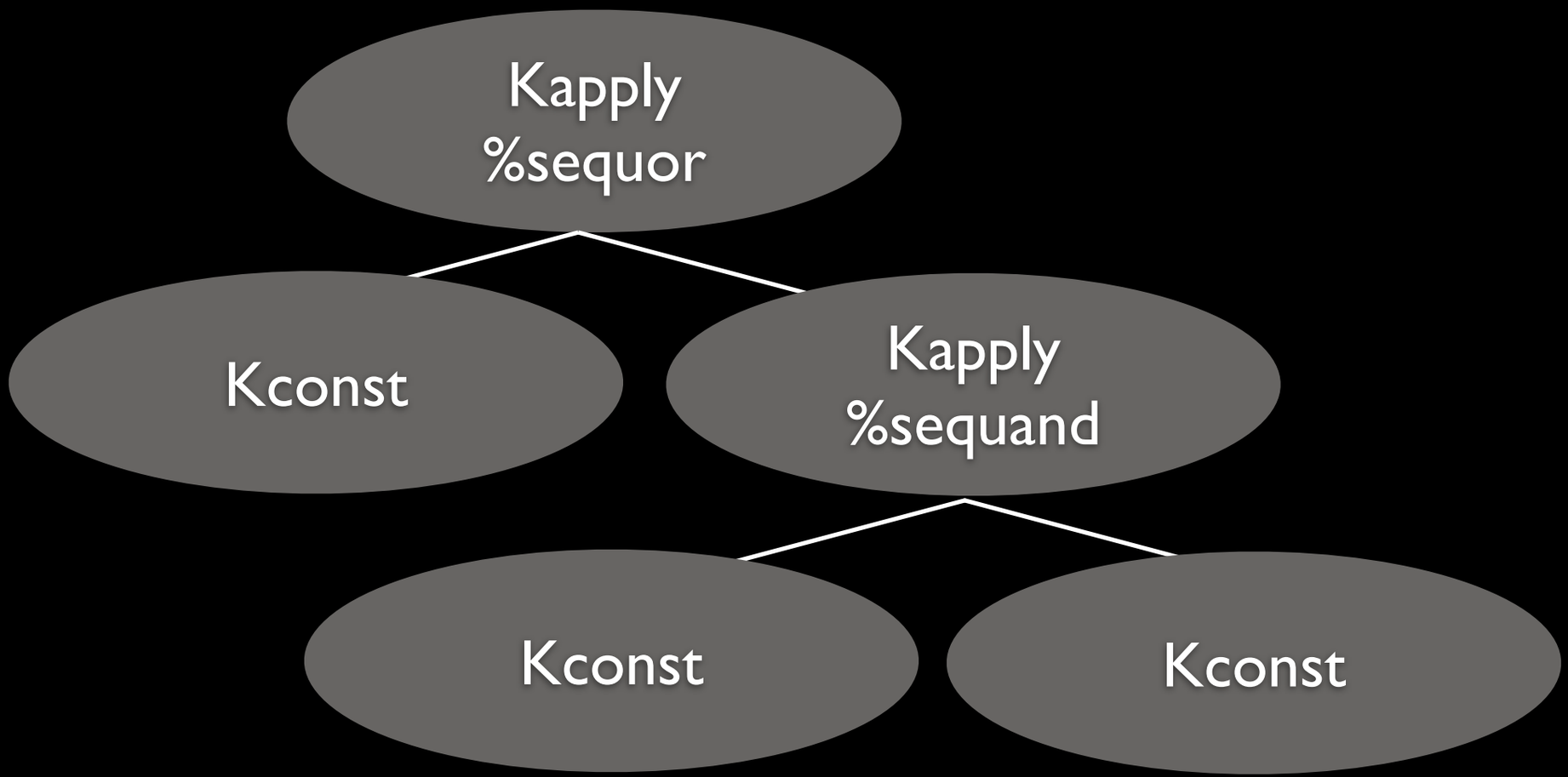


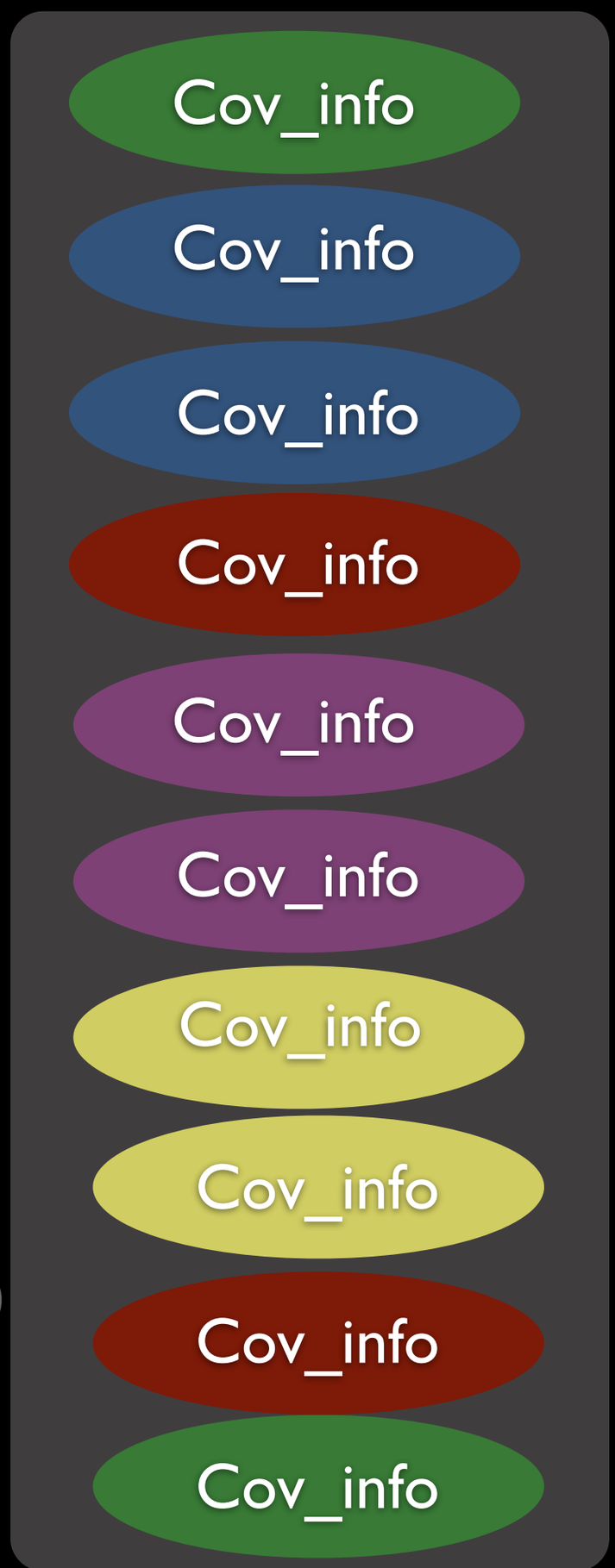
Ajout
lambda-termes
de couverture




true || (true && false)
bytegen.ml
Lambda-termes
~> instructions
Cov_info
Kapply
%sequor
Before
Cov_info
Cov_info
Kconst
Kapply
%sequand
Cov_info
Cov_info
Cov_info
Kconst
Kconst
Cov_info
Cov_info
After

Les informations de couvertures sont sérialisées dans un nouveau fichier
=> **non-intrusives**





MC/DC

- `zamcov-run`
  - Génération des vecteurs MC/DC
  - Remplissage des vecteur lors de l'interprétation
- `zamcov-cover`
  - analyse des vecteurs MC/DC
  - génération d'un rapport de couverture MC/DC

couverture de code

Zamcov

exemples

de rapport pour un code OCaml
utilisant les informations de couverture
(slide suivant)

exemple de rapport avec Zamcov

ZamCov: Expression Coverage (queens.ml)

```
(* Benchmark on list allocation and manipulation *)

let rec append l1 l2 = match l1 with
  [] -> l2
| a::q -> a::(append q l2);;

let rec map f l = match l with [] -> [] | h::t -> (f h)::(map f t);;

let rec iter f l = match l with [] -> () | h::t -> f h ; iter f t ;;

let rec interval n m =
    if n > m then [] else (n :: interval (succ n) m);;

let rec concmap f = function
    [] -> []
  | x :: l -> append (f x) (concmap f l ) (*f x @ concmap f l*);;

let rec list_length = function
    [] -> 0
  | _::l -> 1 + list_length l;;

let rec safe d x = function
    [] -> true
  | q::l ->
      (not (x = q)) && ((not (x = q+d)) && ((not (x = q-d)) && safe (d+1) x l));;

let ok = function [] -> true | x::l -> safe 1 x l;;

let rec filter p = function
   [] -> []
 | x::l -> if p x then x::filter p l else filter p l;;
let range = interval 1;;

let queens n =
 let qs = range n in
 let testcol = function b -> filter ok (map (fun q -> q::b) qs) in
 let rec gen = function
    0 -> [[]]
  | n -> print_string "\n** : ";  print_int n;print_newline();
           let r = concmap testcol (gen (n - 1)) in
           (*print_bll r;*) r  in
 let r =  (gen n) in
   print_string "nb sols ";  print_int (list_length r); print_newline();;

queens 5 ;;
```

exemple

de rapport pour un code OCaml

MLcov — Source Code Report (file queens.ml)

http://www.algo-prog.info/mlcov/

```
(* Benchmark on list allocation and manipulation *)

let rec append l1 l2 = match l1 with
  [] -> l2
| a::q -> a::(append q l2);;

let rec map f l = match l with [] -> [] | h::t -> (f h)::(map f t);;

let rec iter f l = match l with [] -> () | h::t -> f h ; iter f t ;;

let rec interval n m =
    if n > m then [] else (n :: interval (succ n) m);;

let rec concmap f = function
    [] -> []
  | x :: l -> append (f x) (concmap f l ) (*f x @ concmap f l*);;

let rec list_length = function
    [] -> 0
  | _::l -> 1 + list_length l;;

let rec safe d x = function
    [] -> true
  | q::l -> (not (x = q)) & ((not (x = q+d)) & ((not (x = q-d)) &
            safe (d+1) x l));;

let ok = function [] -> true | x::l -> safe 1 x l;;

let rec filter p = function
   [] -> []
 | x::l -> if p x then x::filter p l else filter p l;;
let range = interval 1;;

let queens n =
 let qs = range n in
 let testcol = function b -> filter ok (map (fun q -> q::b) qs) in
 let rec gen = function
    0 -> [[]]
  | n -> print_string "\n** : ";  print_int n;print_newline();
           let r = concmap testcol (gen (n - 1)) in
           (*print_bll r;*) r  in
 let r =  (gen n) in
   print_string "nb sols ";  print_int (list_length r); print_newline();;

queens 5 ;;
```

ZamCov: Expression Coverage (test_gdt.ml)

```
let all_positive1 a b c =
   (a > 0) && (b > 0) && (c > 0);;

all_positive1 1 1 1;;
all_positive1 1 1 0;;

let all_positive2 a b c =
   (a > 0) && (b > 0) && (c > 0);;

all_positive2 1 1 1;;
all_positive2 1 0 1;;
all_positive2 1 1 0;;

let all_positive3 a b c =
   (a > 0) && (b > 0) && (c > 0);;

all_positive3 1 1 1;;
all_positive3 0 1 1;;
all_positive3 1 0 1;;
all_positive3 1 1 0;;
```

DEC line 2: (a > 0) && (b > 0) && (c > 0)	(a > 0)	(b > 0)	(c > 0)
T : 1	T	T	T
F : 1	T	T	F

DEC line 8: (a > 0) && (b > 0) && (c > 0)	(a > 0)	(b > 0)	(c > 0)
T : 1	T	T	T
F : 1	T	F	-
F : 1	T	T	F

DEC line 15: (a > 0) && (b > 0) && (c > 0)	(a > 0)	(b > 0)	(c > 0)
T : 1	T	T	T
F : 1	F	-	-
F : 1	T	F	-
F : 1	T	T	F

Distribution

Compilation du fichier .ml

```
$ zamcov-compile queens.ml -o queens
```

Interprétation et génération de la trace

```
$ zamcov-run -trace queens.trace -mcdc queens.mcdc queens
```

Analyse de couverture

```
$ zamcov-cover -trace queens.trace -mcdc queens.mcdc -exec queens
```

Conclusion