

Modèles de compilation

Compilateur de Focalize

- Sortie OCaml (exécution) ;
- Sortie Coq (certification) ;
- FocDoc (documentation)

Différents modèles

- Modèle à objets (seulement pour OCaml) ;
- Modèle à enregistrements (pour OCaml et Coq) ;
- Modèle à modules (pour OCaml et Coq).

Modèle à enregistrements

- Implanté dans le compilateur actuel ;
- Spécifications mises à plat (héritage résolu) ;
- Utilisation d'enregistrements dépendants (Coq).

Setoids

species *Setoid* =

signature *element* : **Self**;

signature *equal* : **Self** → **Self** → *bool*;

let *different* (*x*, *y*) = $\sim\sim$ *equal* (*x*, *y*);

property *equal_reflexive* : **all** *x* : **Self**, *equal* (*x*, *x*); ...

theorem *same_is_not_different* : **all** *x y* : **Self**,
different (*x*, *y*) \leftrightarrow \sim *equal* (*x*, *y*)

proof = **by definition of** *different*;

theorem *different_is_irreflexive* : **all** *x* : **Self**, \sim *different* (*x*, *x*)

proof = **by property** *equal_reflexive*, *same_is_not_different*; ...

end;;

```
module Setoid =  
struct  
  
  type 'a species = {  
    element : 'a;  
    equal : 'a  $\rightarrow$  'a  $\rightarrow$  Basics.bool;  
    different : 'a  $\rightarrow$  'a  $\rightarrow$  Basics.bool; }  
  
  let different abst_equal (x : 'a) (y : 'a) = Basics.not_bool (abst_equal x y)  
  
end ;;
```

Module *Setoid*.

Record *Setoid* : **Type** :=

```
mk_record {  
  T :> Set ;  
  element : T ;  
  equal : T → T → basics.bool ;  
  different : T → T → basics.bool ;  
  equal_reflexive : forall x : T, ls_true (equal x x); ...  
  same_is_not_different : forall x y : T,  
    ls_true (different x y) ↔ ~ls_true (equal x y) ;  
  different_is_irreflexive : forall x : T, ~ls_true (different x x); ... }.
```

Definition *different* (abst_T : **Set**)

```
(equal : abst_T → abst_T → basics.bool) (x : abst_T) (y : abst_T) :  
basics.bool := (basics.not_bool (abst_equal x y)). ...
```

End *Setoid*.

Section *Proof_of_same_is_not_different*.

Variable *abst_T* : **Set**.

Variable *abst_equal* : *abst_T* \rightarrow *abst_T* \rightarrow *basics.bool*.

Let *abst_different* := *different* *abst_T* *abst_equal*.

Theorem *same_is_not_different* : **forall** *x y* : *abst_T*,
is_true (*abst_different* *x y*) \leftrightarrow \sim (*is_true* (*abst_equal* *x y*)).

Proof.

(* proof generated by Zenon *)

Save.

End *Proof_of_same_is_not_different*.

Section *Proof_of_different_is_irreflexive*.

Variable *abst_T* : **Set**.

Variable *abst_equal* : *abst_T* \rightarrow *abst_T* \rightarrow *basics.bool*.

Variable *abst_different* : *abst_T* \rightarrow *abst_T* \rightarrow *basics.bool*.

Hypothesis *abst_equal_reflexive* : **forall** *x* : *abst_T*,
 ls_true (*abst_equal* *x* *x*).

Hypothesis *abst_same_is_not_different* : **forall** *x y* : *abst_T*,
 ls_true (*abst_different* *x y*) \leftrightarrow \sim *ls_true* (*abst_equal* *x y*).

Theorem *for_zenon_different_is_irreflexive* : **forall** *x* : *abst_T*,
 \sim (*ls_true* (*abst_different* *x* *x*)).

Proof.

(** proof generated by Zenon **)

Save.

End *Proof_of_different_is_irreflexive*.

Setoids d'entiers

```
species Setoid_int =  
  inherit Setoid;  
  representation = int;  
  let element = 0;  
  let equal = ( =0x );  
  
  proof of equal_reflexive = ...; ...  
end;;
```

```
module Setoid_int =  
struct  
  
  type 'a species = {  
    element : 'a;  
    equal : 'a → 'a → Basics.bool;  
    different : 'a → 'a → Basics.bool; }  
  
  let element = 0  
  let equal = Basics.equal_0x  
  
  let collection_create () =  
    let local_element = element in  
    let local_equal = equal in  
    let local_different = Setoid.different local_equal in  
    { element = local_element;  
      equal = local_equal;  
      different = local_different; }  
  
end ;;
```


Module *Setoid_int*.

Record *Setoid* : **Type** :=

```
mk_record {  
  T := Set ;  
  element : T ;  
  equal : T → T → basics.bool;  
  different : T → T → basics.bool;  
  equal_reflexive : forall x : T, ls_true (equal x x); ...  
  same_is_not_different : forall x y : T,  
    ls_true (rf_different x y) ↔ ~ls_true (equal x y) ;  
  different_is_irreflexive : forall x : T, ~ls_true (different x x); ... }.
```

Definition *element* (*abst_T* := *basics.int*) : *abst_T* := 0.

Definition *equal* (*abst_T* := *basics.int*) :

```
abst_T → abst_T → basics.bool := basics.equal_0x. ...
```

```
Definition collection_create :=  
  let local_rep := basics.int in  
  let local_element := element in  
  let local_equal := equal in  
  let local_different := Setoid.different local_rep local_equal in  
  let local_equal_reflexive := equal_reflexive local_rep local_equal in ...  
  let local_same_is_not_different := Setoid.same_is_not_different local_rep  
    local_equal in  
  let local_different_is_irreflexive := Setoid.different_is_irreflexive  
    local_rep local_equal local_different local_equal_reflexive  
    local_same_is_not_different in ...  
  mk_record local_rep local_element local_equal local_different  
  local_equal_reflexive ... local_same_is_not_different  
  local_different_is_irreflexive ... .
```

End *Setoid_int*.

Setoids d'entiers

```
collection Int = implement Setoid_int; end;;
```

```
module Int =  
struct  
  type 'a species = {  
    element : 'a;  
    equal : 'a → 'a → Basics.bool;  
    different : 'a → 'a → Basics.bool; }  
  
  let effective_collection = Setoid_int.collection_create ()  
  
end ;;
```

Module *Int*.

Record *Setoid* : **Type** :=

mk_record {

T :> **Set** ;

element : *T* ;

equal : *T* → *T* → *basics.bool*;

different : *T* → *T* → *basics.bool*;

equal_reflexive : **forall** *x* : *T*, *ls_true* (*equal* *x* *x*); ...

same_is_not_different : **forall** *x y* : *T*,

ls_true (*rf_different* *x y*) ↔ ~*ls_true* (*equal* *x y*) ;

different_is_irreflexive : **forall** *x* : *T*, ~*ls_true* (*different* *x* *x*); ... }.

Let *effective_collection* := *Setoid_int.collection_create*.

End *Int*.

Piles

```
species Stack (Typ is Setoid) =  
  
  signature empty : Self ;  
  signature push : Typ → Self → Self ;  
  signature pop : Self → Self ;  
  signature last : Self → Typ ;  
  signature is_empty : Self → bool ;  
  
  property ie_push : all e : Typ, all s : Self, ~(is_empty (push (e, s)));  
  
  property lt_push : all e : Typ, all s : Self,  
    Typ!equal (last (push (e, s)), e); ...  
  
end;;
```

```
module Stack =  
struct  
  type ('typ, 'a) species = {  
    empty : 'a ;  
    is_empty : 'a → Basics.bool ;  
    last : 'a → 'typ ;  
    pop : 'a → 'a ;  
    push : 'typ → 'a → 'a ; ... }  
  
end ;;
```

Module *Stack*.

```
Record Stack (Typ_T : Set) (Typ_equal : Typ_T → Typ_T → basics.bool) : Type :=  
  mk_record {  
    T :> Set;  
    empty : T;  
    is_empty : T → basics.bool;  
    last : T → Typ_T;  
    pop : T → T;  
    push : Typ_T → T → T;  
    ie_push : forall e : Typ_T, forall s : T,  
      ~is_true (is_empty (push e s));  
    lt_push : forall e : Typ_T, forall s : T,  
      is_true (Typ_equal (last (push e s)) e) ... }.
```

End *Stack*.

Piles

```
species Finite_stack (Typ is Setoid, max in Int) =  
  inherit Stack (Typ);  
  
  signature size : Self  $\rightarrow$  Int;  
  
  let is_full (s) = Int!equal (size (s), max); ...  
  
end;;
```

```
module Finite_stack =  
struct  
  type ('typ, 'max, 'a) species = {  
    empty : 'a;  
    is_empty : 'a → Basics.bool;  
    last : 'a → 'typ;  
    pop : 'a → 'a;  
    push : 'typ → 'a → 'a;  
    size : 'a → 'max;  
    is_full : 'a → Basics.bool; ... }  
  
  let is_full max abst_size s =  
    Int.effective_collection.Int.equal (abst_size s) max  
  
end;;
```

Module *Finite_stack*.

Record *Finite_stack* (*Typ_T* : **Set**) (*max* : *Int.effective_collection*.(*Int.T*))
 (*Typ_equal* : *Typ_T* → *Typ_T* → *basics.bool*) : **Type** :=
 mk_record {
 T :> **Set**;
 empty : *T*;
 is_empty : *T* → *basics.bool*;
 last : *T* → *Typ_T*;
 pop : *T* → *T*;
 push : *Typ_T* → *T* → *T*;
 size : *T* → *Int.effective_collection*.(*Int.T*);
 is_full : *T* → *basics.bool*;
 ie_push : **forall** *e* : *Typ_T*, **forall** *s* : *T*,
 ~*is_true* (*is_empty* (*push e s*));
 lt_push : **forall** *e* : *Typ_T*, **forall** *s* : *T*,
 is_true (*Typ_equal* (*last* (*push e s*)) *e*) ... }.

Definition *is_full* (*max* : *Int.effective_collection*.(*Int.T*))
 (*abst_T* : **Set**)
 (*abst_size* : *abst_T* → *Int.effective_collection*.(*Int.T*))
 (*s* : *abst_T*) : *basics.bool* :=
 (*Int.effective_collection*.(*Int.equal*) (*abst_size s*) *max*).

End *Finite_stack*.