

Documentation of APTE v0.3.1beta : Algorithm for Proving Trace Equivalence

Vincent Cheval

School of Computer Science, University of Birmingham
LSV, ENS Cachan & CNRS & INRIA Saclay Île-de-France

March 26, 2013

Contents

1	Standard Library	5
1.1	Module Term : Operations on terms	5
1.1.1	Symbol	5
1.1.2	Messages	7
1.1.3	Mapping table	11
1.1.4	Substitution	11
1.1.5	Rewrite rules	12
1.1.6	Unification	12
1.1.7	Formula	13
1.2	Module Recipe : Operations on recipes	15
1.2.1	Recipe	15
1.2.2	Variable Mapping	17
1.2.3	Substitution and unify	18
1.2.4	Path	18
1.2.5	Recipe context	19
1.2.6	Formula on contexts of recipes	20
1.3	Module Constraint : Frame and deducibility constraints	21
1.3.1	Support set	21
1.3.2	Frame	24
1.3.3	Deducibility constraint	27
1.4	Module Constraint_system : Operations on (matrices of) constraint systems	29
1.4.1	Constraint system	29
1.4.2	Functionnalities of Phase 1	32
1.4.3	Functionnalities of Phase 2	33
1.4.4	Row matrix of constraint system	35
1.4.5	Matrix of constraint systems	36
1.4.6	Rule applications	39
1.5	Module Process : Process	41
1.5.1	Symbolic process	41
2	Trace equivalence	43
2.1	Module Rules : Definitions of the rules	43
2.1.1	Rule CONS	43
2.1.2	Rule AXIOM	43
2.1.3	Rule DEST	44
2.1.4	Rule EQ-LEFT-LEFT	44
2.1.5	Rule EQ-LEFT-RIGHT	44
2.1.6	Rule EQ-RIGHT-RIGHT	44
2.1.7	Rule DED-ST	45
2.2	Module Strategy	45
2.3	Module Algorithm	45

Chapter 1

Standard Library

1.1 Module Term : Operations on terms

This module regroupes all the functions that manipulate terms. In [Che12], the terms are splitted into first (resp. second) order terms called messages (resp. recipe). In this module, we focus on the messages. The recipe are handled in a different module.

1.1.1 Symbol

A symbol can be a destructor or a constructor.

`type symbol`

The type `symbol` represents the type of function symbol.

Built-in signature

The algorithm described in [Che12] considers a fix set of cryptographic primitives whose behaviour is defined by rewrite rules plus any number of constructors. Thus, we directly defined here this set of cryptographics primitives.

Built-in constructors

`val senc : symbol`

`senc` is the symbol for symmetric encryption (arity 2).

`val aenc : symbol`

`aenc` is the symbol for asymmetric encryption (arity 2).

`val pk : symbol`

`pk` is the symbol for asymmetric public key (arity 1).

`val vk : symbol`

`vk` is the symbol for public verification key used in signature (arity 1).

`val sign : symbol`

`sign` is the symbol for asymmetric public key (arity 1).

`val hash : symbol`

`hash` is the symbol for hash function (arity 1).

Built-in destructors

val sdec : symbol

sdec is the symbol for symmetric decryption (arity 2).

val adec : symbol

adec is the symbol for asymmetric decryption (arity 2).

val checksign : symbol

checksign is the symbol for signature verification (arity 2).

Although the algorithm described in [Che12] only have a pair function of arity 2 with its associated projection, it can be extended to tuple of any arity. Thus, a user we be allowed to use such tuple.

val nth_projection : symbol -> int -> symbol

`nth_projection f i` returns the projection function symbol of the i^{th} element of tuple function symbol `f`. Note that for a tuple of arity `n`, the range of `i` is $1 \dots n$.

Raises

- `Internal_error` if `f` is not a tuple.
- `Not_found` if `f` was not previously introduced by `get_tuple`.

val get_projections : symbol -> symbol list

`get_projections f` returns the list `[f_1; ...; f_n]` with `f_i` is the projection function symbol of the i^{th} element of the tuple function symbol `f`. It returns the same result as `[nth_projection f 1; ...; nth_projection f n]`.

Raises

- `Internal_error` if `f` is not a tuple.
- `Not_found` if `f` was not previously introduced by `get_tuple`.

val all_tuple : symbol list Pervasives.ref

The list contains all tuples introduced by the algorithm.

val all_constructors : symbol list Pervasives.ref

The list of all constructors (included the tuple function symbol) used in the algorithm.

val number_of_constructors : int Pervasives.ref

The number of constructors used in the algorithm.

Addition

val new_constructor : int -> string -> symbol

`new_symbol ar s` creates a constructor function symbol with the name `s` and the arity `ar`. The resulting symbol is automatically added into `all_constructors`. Moreover, `number_of_constructors` is increased by 1. Note that if the constructor is in fact a tuple, it is better to use `get_tuple`.

val get_tuple : int -> symbol

`get_tuple ar` get the function symbol for tuple of arity `ar`. If such function symbol was not created yet, it creates it and the resulting symbol is automatically added into `all_constructors`. Moreover, `number_of_constructors` is increased by 1. At last, the associated projection function symbol are automatically added into `all_projection`.

Symbol testing

```
val is_equal_symbol : symbol -> symbol -> bool
    is_equal_symbol f1 f2 returns true iff f1 and f2 are the same function symbol.

val is_tuple : symbol -> bool
    is_tuple f returns true iff f is a tuple.

val is_constructor : symbol -> bool
    is_constructor f returns true iff f is a constructor or a tuple. Note that all tuples are
    constructors.

val is_destructor : symbol -> bool
    is_destructor f returns true iff f is a destructor.
```

Symbol Access

```
val get_arity : symbol -> int
    get_arity f returns the arity of the function symbol f.
```

Symbol Display

```
val display_symbol_without_arity : symbol -> string
val display_symbol_with_arity : symbol -> string
```

1.1.2 Messages

```
type quantifier =
  | Free
  | Existential
  | Universal
    The type quantifier is associated to a variable to quantify it.
```

```
type variable
    A variable is always quantified. It corresponds to the set  $\mathcal{X}^1$  in [Che12].
```

```
type name_status =
  | Public
  | Private
    A name is can be either public or private.
```

```
type name
    The type name corresponds to the set  $\mathcal{N}$  in [Che12].
```

```
type term
    The type term corresponds to the set  $\mathcal{T}(\mathcal{F}, \mathcal{N} \cup \mathcal{X}^1)$  in [Che12].
```

Variable generation

The variables created by the functions below are structurally and physically different

```
val fresh_variable : quantifier -> variable
    fresh_variable q creates a fresh variable quantified by q.

val fresh_variable_from_id : quantifier -> string -> variable
    fresh_variable_from_id q s creates a fresh variable quantified as q with display identifier s.

val fresh_variable_from_var : variable -> variable
    fresh_variable_from_var v creates a fresh variable with the same display identifier and
    quantifier as the variable v.

val fresh_variable_list : quantifier -> int -> variable list
    fresh_variable_list q n creates a list of n fresh variables all quantified as q.

val fresh_variable_list2 : quantifier -> int -> term list
    fresh_variable_list2 q n creates a list of n fresh variables all quantified as q and considered
    as terms.
```

Name generation

```
val fresh_name : name_status -> name
    fresh_name ns creates a fresh name with the status ns.

val fresh_name_from_id : name_status -> string -> name
    fresh_name_from_id ns s creates a fresh name with status ns and with display identifier s.

val fresh_name_from_name : name -> name
    fresh_name_from_name n creates a fresh name with the same display identifier and same status
    as n.
```

Generation of terms

```
val term_of_variable : variable -> term
    term_of_variable v returns the variable v considered as a term.

val term_of_name : name -> term
    term_of_name n returns the name n considered as a term.

val variable_of_term : term -> variable
    variable_of_term t returns the term t as a variable.
    Raises Internal_error if t is not a variable.

val name_of_term : term -> name
    name_of_term t returns the term t as a name.
    Raises Internal_error if t is not a name.

val apply_function : symbol -> term list -> term
    apply_function f args applies the the function symbol f to the arguments args. If args is the
    list [t1;...;tn] then the term obtained is f(t1,...,tn).
    [Low debugging]Raise an internal error if the number of arguments in args does not coincide
    with the arity of f.
```



```

val rename :
  (variable * variable) list ->
  (name * name) list -> term -> term
  rename v_list n_list t creates a new term from t where each v_i is replaced by v'_i and
  each n_i is replaced by n'_i where v_list is the list (v_1,v'_1),...,(v_p,v'_p) and n_list
  is the list (n_1,n'_1),...,(n_q,n'_q).

```

Access functions

```

val top : term -> symbol
  top t returns the symbol at the root position of t.
  Raises Internal_error if t is not a function symbol application.

val nth_args : term -> int -> term
  nth_args t i returns the  $i^{th}$  argument of the constructed term t. Note that the index i start
  with 1 and not 0. For example, if t is the term  $f(t_1, \dots, t_n)$  then nth_args t i returns the term
   $t_i$ .
  Raises Internal_error if t is not a function symbol application.

val get_args : term -> term list
  get_args t returns the list of argument of the constructed term t. For example, if t is the term
   $f(t_1, \dots, t_n)$  then get_args t returns the list  $[t_1; \dots; t_n]$ .
  Raises Internal_error if t is not a function symbol application.

val get_quantifier : variable -> quantifier
  get_quantifier v returns the quantifier of v.

```

Scanning

```

val var_occurs : variable -> term -> bool
  occurs v t returns true iff the variable v occurs in the term t.

val var_occurs_list : variable list -> term -> bool
  occurs_list v_list t returns true iff one of the variable in v_list occurs in the term t.

val exists_var : quantifier -> term -> bool
  exists_var q t returns true iff there exists a variable quantified as q in the term t.

val for_all_var : quantifier -> term -> bool
  for_all_var q t returns true iff all variables in the term t are quantified as q.

val exists_name_with_status : name_status -> term -> bool
  exists_name s t returns true iff there exists a name in t with status s.

val exists_name : term -> bool
  exists_name t returns true iff there exists a name in t.

val is_equal_term : term -> term -> bool
  is_equal_term t1 t2 returns true iff the terms t1 and t2 are equal.

val is_equal_and_closed_term : term -> term -> bool
  is_equal_term t1 t2 returns true iff the terms t1 and t2 are equal.

```

```

val is_equal_name : name -> name -> bool
    is_equal_name n1 n2 returns true iff the name n1 and n2 are equal.

val is_variable : term -> bool
    is_variable t returns true iff the term t is a variable.

val is_name : term -> bool
    is_name t returns true iff the term t is a name.

val is_name_status : name_status -> term -> bool
    is_name_status s t returns true iff the term t is a name with status s.

val is_function : term -> bool
    is_function t returns true iff the term t is a function symbol application.

val is_constructor_term : term -> bool
    is_constructor_term t returns true iff  $t \in \mathcal{T}(\mathcal{F}_c, \mathcal{X}^1 \cup \mathcal{N})$ .

```

Iterators

```

val fold_left_args : ('a -> term -> 'a) -> 'a -> term -> 'a
    fold_left_args f acc t is f (... (f (f acc t1) t2) ...) tn if t is the term  $g(t_1, \dots, t_n)$ 
    for some function symbol  $g$  .

    Raises Internal_error if t is not a function application.

val fold_right_args : (term -> 'a -> 'a) -> term -> 'a -> 'a
    fold_right_args f t acc is f t1 (f t2 (... (f tn acc) ...)) if t is the term  $g(t_1, \dots, t_n)$  for
    some function symbol  $g$  .

    Raises Internal_error if t is not a function application.

val map_args : (term -> 'a) -> term -> 'a list
    map_args f t is the list [f t1; ...; f tn] if t is the term  $g(t_1, \dots, t_n)$  for some function
    symbol  $g$  .

    Raises Internal_error if t is not a function application.

val fold_left_args2 : ('a -> term -> 'b -> 'a) -> 'a -> term -> 'b list -> 'a
    fold_left_args2 f acc t l is f (... (f (f acc t1 e1) t2 e2) ...) tn en if t is the term
     $g(t_1, \dots, t_n)$  for some function symbol  $g$  and l is the list [e1; ...; en].

    Raises Internal_error if t is not a function application.

```

Display

```

val display_term : term -> string
val display_name : name -> string
val display_variable : variable -> string

```

1.1.3 Mapping table

```
module VariableMap :
sig
  type 'a map
    'a map is the type that represents the mapping of variable to element of type 'a.

  val empty : 'a map
    empty is the empty mapping function.

  val is_empty : 'a map -> bool
    is_empty map returns true iff map is empty.

  val add : Term.variable -> 'a -> 'a map -> 'a map
    add v elt map returns a map containing the same bindings as map, plus a binding of v to
    elt. If v was already bound in map, its previous binding disappears.

  val find : Term.variable -> 'a map -> 'a
    find v map returns the current binding of v in map.
    Raises Not_found if no binding exists.

  val mem : Term.variable -> 'a map -> bool
    mem v map returns true iff map contains a binding for v.

  val display : ('a -> string) -> 'a map -> unit
end
```

1.1.4 Substitution

```
type substitution
val identity : substitution
  identity corresponds to the identity substitution.

val is_identity : substitution -> bool
  is_identity s returns true iff s is the identity substitution.

val create_substitution : variable -> term -> substitution
  create_substitution v t creates the substitution  $v \rightarrow t$ .

val compose : substitution -> substitution -> substitution
  compose  $\sigma_1 \sigma_2$  returns the substitution  $\sigma_1 \sigma_2$ .
  [Low debugging] Raise an internal error if the domain of two substitutions are not disjoint.

val filter_domain : (variable -> bool) -> substitution -> substitution
  filter_domain f s returns the substitution s restricted to variables that satisfy f.

val apply_substitution :
  substitution -> 'a -> ('a -> (term -> term) -> 'a) -> 'a
```

`apply_substitution subst elt map_elt` applies the substitution `subst` on the element `elt`. The function `map_elt` should map the terms contained in the element `elt` on which `subst` should be applied.

For example, applying a substitution `subst` on a list of terms `term_list` could be done by applying `apply_substitution subst term_list (fun l f -> List.map f l)`.

Another example: applying a substitution `subst` on the second element of a couple of terms could be done by applying `apply_substitution subst term_c (fun (t1,t2) f -> (t1, f t2))`.

```
val apply_substitution_change_detected :
  substitution ->
  'a -> ('a -> (term -> bool * term) -> 'a) -> 'a
  apply_substitution_change_detected subst elt map_elt is similar to apply_substitution
  except that the function map_elt, which should map the term to be substituted, will consider a
  function that returns if a term was modify or not.
  apply_substitution_change_detected subst elt map_elt is faster but has the same result as
  apply_substitution subst elt (fun a f ->
    map_elt a (fun t ->
      let t' = f t in not (is_equal_term t t'),t'))

val equations_of_substitution : substitution -> (term * term) list
  equations_of_substitution s returns the list [(x1,t1);...;(xn,tn)] where s is the
  substitution  $x_1 \rightarrow t_1, \dots, x_n \rightarrow t_n$ .
```

1.1.5 Rewrite rules

```
val fresh_rewrite_rule : symbol -> term list * term
  fresh_rewrite_rule f returns the couple ([t1,...,tn],t) where  $f(t_1, \dots, t_n) \rightarrow t$  is a fresh
  rewrite rule of f.

val link_destruc_construc : symbol -> symbol -> bool
  link_destruc_construc s_d s_c returns true iff s_d is the destructor symbol of the
  constructor symbol s_c.
  Raises Internal_error if s_c is not a constructor or if it is a tuple symbol.
  Example : link_destruc_construc sdec senc returns true.

val constructor_to_destructor : symbol -> symbol
  constructor_to_destructor s_c returns the destructor symbol of the constructor symbol s_c.
  Raises Internal_error if s_c is not a constructor or if it is a tuple symbol.
```

1.1.6 Unification

```
exception Not_unifiable
val unify : (term * term) list -> substitution
  unify l unifies the pairs of term in l and returns the substitution that unifies them
  Raises Not_unifiable if no unification is possible.

val is_unifiable : (term * term) list -> bool
  is_unifiable l returns true iff the pairs of term in l are unifiable.

val unify_and_apply :
  (term * term) list ->
  'a -> ('a -> (term -> term) -> 'a) -> 'a
```

`unify_and_apply 1 elt map_elt` unifies the pairs of term in `1` and apply the substitution that unifies them on the terms in `elt` according to the function `map_elt`.

Raises `Not_unifiable` if no unification is possible.

It is faster but returns the same as `apply_substitution (unify 1) elt map_elt`.

`val unify_and_apply_change_detected :`

`(term * term) list ->`

`'a -> ('a -> (term -> bool * term) -> 'a) -> 'a`

`unify_and_apply_change_detected 1 elt map_elt` unifies the pairs of term in `1` and apply the substitution that unifies them on the terms in `elt` according to the function `map_elt`.

Raises `Not_unifiable` if no unification is possible.

It is faster but returns the same as `apply_substitution_change_detected (unify 1) elt map_elt`.

`val unify_modulo_rewrite_rules : (term * term) list -> substitution`

`unify_modulo_rewrite_rules 1` unifies the pairs of term in `1` modulo the rewriting systems. All variables introduced by the unification are quantified existentially.

Raises `Not_unifiable` if no unification is possible or if a destructor cannot be reduced

`val unify_modulo_rewrite_rules_and_apply :`

`(term * term) list ->`

`'a -> ('a -> (term -> term) -> 'a) -> 'a`

`unify_modulo_rewrite_rules_and_apply 1 elt map_elt` unifies the pairs of term in `1` modulo the rewriting systems and apply the substitution that unifies them on the terms in `elt` according to the function `map_elt`. All variables introduced by the unification are quantified existentially.

Raises `Not_unifiable` if no unification is possible.

It is faster but returns the same as `apply_substitution (unify 1) elt map_elt`.

1.1.7 Formula

`type formula`

The type `formula` represents a disjunction of inequation of the form $\forall \tilde{x}. \bigvee_{i=1}^n u_i \stackrel{?}{\neq} v_i$ for some terms u_i, v_i that may contain destructor symbol. Note that the semantics of $u \stackrel{?}{\neq} v$ is given in [Che12].

`val top_formula : formula`

`top_formula` is the always true formula.

`val bottom_formula : formula`

`bottom_formula` is the always false formula.

`val create_inequation : term -> term -> formula`

`create_inequation t_1 t_2` creates the formula $t_1 \stackrel{?}{\neq} t_2$. Note that the quantifier of the variables are not modified. For instance, the existential variables in `t_1` and `t_2` do not become universal variables. Note that `create_inequation` is not commutative, i.e. `create_inequation t_1 t_2` is different from `create_inequation t_2 t_1`.

`val create_disjunction_inequation : (term * term) list -> formula`

`create_disjunction_inequation [(u_1, v_1); ...; (u_n, v_n)]` creates the formula $\bigvee_{i=1}^n u_i \stackrel{?}{\neq} v_i$. Note that the quantifier of the variables are not modified.

Iterators

```
val iter_inequation_formula : (term -> term -> unit) -> formula -> unit
    iter_inequation_formula f phi is f u1 v1; f u2 v2; ...; f un vn where phi is the
    formula  $\forall \tilde{x}. \bigvee_{i=1}^n u_i \neq v_i$  .

val map_term_formula : formula -> (term -> term) -> formula
    map_term_formula phi f is the formula create_disjunction_inequation [(f u1, f
    v1); ...; (f un, f vn)] where phi is the formula  $\forall \tilde{x}. \bigvee_{i=1}^n u_i \neq v_i$  .

val map_term_formula_change_detected :
    formula -> (term -> bool * term) -> bool * formula
    Similar to map_term_formula except that it returns the couple (b, phi') where phi' is the
    formula phi on which we applied f and b is true iff one of the application of f returned true.
```

Formula scanning

```
val find_and_apply_formula :
    (term -> term -> bool) ->
    (term -> term -> 'a) -> (unit -> 'a) -> formula -> 'a
    find_and_apply_formula f_test f_apply f_no formula searches in formula an inequation
    satisfying f_test. If such inequation exists then it applies f_apply on it else it apply the
    function f_no.

    Note that since an inequation  $u \neq v$  is semantically the same as  $v \neq u$  , it is recommended that
    f_test u v and f_test v u are equal. Same for f_apply.

val is_bottom : formula -> bool
    is_bottom formula returns true iff formula is the always false formula.

val is_top : formula -> bool
    is_true formula returns true iff formula is the always true formula.

val is_in_formula : term -> term -> formula -> bool
    is_in_formula t1 t2 formula returns true iff formula is of the form  $\forall \tilde{x}. t_1 \neq t_2 \vee F$  where
    F is a disjunction of inequation. Note that this function is commutative, i.e. is_in_formula t1
    t2 phi is the same as is_in_formula t2 t1 phi.
```

Simplification

Following [Che12], a substitution σ of constructor terms models a formula $u \neq v$, denoted $\sigma \models_c u \neq v$, if $u\sigma \downarrow \neq v\sigma \downarrow$ or $\text{Message}(u)$ or $\text{Message}(v)$. \models_c is naturally extended to formula $\forall \tilde{x}. \bigvee_i u_i \neq v_i$. The simplification functions in this section preserve the models of the formulas.

```
val simplify_formula : formula -> formula
    This function transforms a formula of the form  $\forall \tilde{x} \bigvee_i u_i \neq v_i$  into a formula of the form
     $\forall \tilde{y} \bigvee_j x_j \neq t_j$  where  $u_i, v_i, t_i$  are all constructors terms and all  $x_j$  are distinct.

val simplify_formula_phase_2 : formula -> formula
    This function simplifies a formula containing only constructor term by the simplification rules
    defined in [Che12, Figure 7.3].

val simplify_formula_modulo_rewrite_rules : formula -> formula
    This function simplifies a formula that may contain destructor symbols into a formula that
    contains only constructor terms.
```

Display

```
val display_formula : formula -> string
```

1.2 Module Recipe : Operations on recipes

This module regroups all the functions that manipulate recipes. In [Che12], the terms are splitted into first (resp. second) order terms called messages (resp. recipe). In this module, we focus on the recipes. The message are handled in a different module. In theory a recipe and a message are both terms hence one could consider this module almost as a copy of the module `Term`. However, in the algorithm presented in [Che12], the usage of message and recipe are really different.

1.2.1 Recipe

`type variable`

The type `variable` corresponds to the set \mathcal{X}^2 in [Che12]. Since the recipe variable are always introduced in the algorithm with a deducibility constraint, a recipe variable is always associated to an integer called the support in [Che12]. For example, if $X, i \vdash^? u$ is a deducibility constraint, the support of X is i . Hence a variable is always associated to a support in our module

`type axiom`

The type `axiom` corresponds to the set \mathcal{AX} in [Che12]. Similarly to the variable, a axiom is always associated to a support. In [Che12], for an axiom ax_i , i is the support.

`type recipe`

The type `recipe` corresponds to the set $\mathcal{T}(\mathcal{F}, \mathcal{AX} \cup \mathcal{X}^2)$ in [Che12]. Note that the recipes does not have names. It corresponds to the recipes used in Chapter 7 and 8 of [Che12].

Fresh function

```
val fresh_variable : int -> variable
```

`fresh_variable n` creates a fresh variable with support n .

```
val fresh_variable_from_id : string -> int -> variable
```

`fresh_variable_from_id s n` creates a fresh variable with support n and display identifier s .

```
val fresh_variable_list : int -> int -> variable list
```

`fresh_variable_list nb n` creates a list of nb fresh variables all with support n .

```
val fresh_variable_list2 : int -> int -> recipe list
```

`fresh_variable_list2 nb n` creates a list of nb fresh variables considered as recipes and all with support n .

```
val fresh_free_variable : int -> variable
```

`fresh_free_variable n` creates a fresh free variable with support n .

```
val fresh_free_variable_from_id : string -> int -> variable
```

`fresh_free_variable_from_id s n` creates a fresh free variable with support n and display identifier s .

```
val fresh_free_variable_list : int -> int -> variable list
```

`fresh_free_variable_list nb n` creates a list of nb fresh free variables all with support n .

```
val axiom : int -> axiom
```

`axiom n` creates an axiom with support n .

Generation of recipe

```
val recipe_of_variable : variable -> recipe
    recipe_of_variable v returns the variable v considered as a recipe.

val recipe_of_axiom : axiom -> recipe
    recipe_of_axiom ax returns the axiom ax considered as a recipe.

val variable_of_recipe : recipe -> variable
    variable_of_recipe r returns the recipe r as a variable.
    Raises Internal_error if r is not a variable.

val axiom_of_recipe : recipe -> axiom
    axiom_of_recipe r returns the recipe r as an axiom.
    Raises Internal_error if r is not an axiom.

val apply_function : Term.symbol -> recipe list -> recipe
    apply_function f args applies the the function symbol f to the arguments args. If args is the
    list [r1;...;rn] then the recipe obtained is f(r1,...,rn).
    [Low debugging]Raise an internal error if the number of arguments in args does not coincide
    with the arity of f.
```

Access

```
val top : recipe -> Term.symbol
    top r returns the symbol at the root position of r.
    Raises Internal_error if r is not a function symbol application.

val get_support : variable -> int
    get_support v returns the support of the variable v.
```

Testing

```
val is_equal_variable : variable -> variable -> bool
    is_equal_variable v1 v2 returns true iff v1 and v2 are the same variable.

val is_equal_axiom : axiom -> axiom -> bool
    is_equal_axiom ax1 ax2 returns true iff ax1 and ax2 are the same axioms.

val is_equal_recipe : recipe -> recipe -> bool
    is_equal_recipe r1 r2 returns true iff r1 and r2 are the same recipes.

val occurs : variable -> recipe -> bool
    occurs v r return true iff the variable v is in the recipe r

val is_free_variable : variable -> bool
    is_free_variable v returns true iff v is free.

val is_free_variable2 : recipe -> bool
    is_free_variable2 r returns true iff r is a free variable.

val is_variable : recipe -> bool
    is_variable r returns true iff r is a variable.
```



```

val is_axiom : recipe -> bool
    is_axiom r returns true iff r is an axiom.

val is_function : recipe -> bool
    is_function r returns true iff r is a function symbol application.

```

Iterators

```

val iter_args : (recipe -> unit) -> recipe -> unit
    iter_args f r is f r1; ...; f rn if r is the recipe  $g(r_1, \dots, r_n)$  for some function symbol  $g$ .
    Raises Internal_error if r is not a function application.

val map_args : (recipe -> 'a) -> recipe -> 'a list
    map_args f r is the list [f r1; ...; f rn] if r is the recipe  $g(r_1, \dots, r_n)$  for some function
    symbol  $g$ .
    Raises Internal_error if r is not a function application.

```

Display

```

val display_variable : variable -> string
val display_axiom : axiom -> string
val display_recipe : recipe -> string
val display_recipe2 :
    (recipe * 'a) list -> ('a -> string) -> recipe -> string
    display_recipe assoc f_display r display the recipe r but each variable and axiom r' in r is
    displayed as f_display b if (r',b) is in assoc else is normally displayed.

```

1.2.2 Variable Mapping

```

module VariableMap :
sig
    type 'a map
        'a map is the type that represents the mapping of variable to element of type 'a.

    val empty : 'a map
        empty is the empty mapping function.

    val is_empty : 'a map -> bool
        is_empty map returns true iff map is empty.

    val add : Recipe.variable ->
        'a -> 'a map -> 'a map
        add v elt map returns a map containing the same bindings as map, plus a binding of v to
        elt. If v was already bound in map, its previous binding disappears.

    val find : Recipe.variable -> 'a map -> 'a
        find v map returns the current binding of v in map.
        Raises Not_found if no binding exists.

    val mem : Recipe.variable -> 'a map -> bool
        mem v map returns true iff map contains a binding for v.

end

```

1.2.3 Substitution and unify

type substitution

substitution corresponds to a mapping from \mathcal{X}^2 to $\mathcal{T}(\mathcal{F}, \mathcal{AX} \cup \mathcal{X}^2)$.

val is_identity : substitution -> bool

is_identity s returns true iff s is the identity substitution.

val unify : (recipe * recipe) list -> substitution

unify l returns the most general unifier of the pairs of recipes in l.

val create_substitution : variable -> recipe -> substitution

create_substitution v r returns the substitution $\{v \mapsto r\}$.

val create_substitution2 : recipe -> recipe -> substitution

create_substitution2 v r returns the substitution $\{v \mapsto r\}$.

Raises Internal_error if v is not a variable.

val apply_substitution :

substitution ->

'a -> ('a -> (recipe -> recipe) -> 'a) -> 'a

apply_substitution subst elt map_elt applies the substitution subst on the element elt.

The function map_elt should map the recipes contained in the element elt on which subst should be applied. See Term.apply_substitution for more explanation.

val equations_from_substitution : substitution -> (recipe * recipe) list

equations_from_substitution subst returns [(v1,r1);...;(vn,rn)] if subst is the substitution $\{v_1 \mapsto r_1, \dots, v_n \mapsto r_n\}$.

val filter_domain : (variable -> bool) -> substitution -> substitution

filter_domain f s returns the substitution s restricted to variables that satisfy f.

1.2.4 Path

type path

The path corresponds to the path of a recipe defined in [Che12, Definition 7.4]. It corresponds to the set $\mathcal{F}_d^* \cdot \mathcal{AX}$ in [Che12].

val path_of_recipe : recipe -> path

path_of_recipe xi returns the path of a recipe. It corresponds to path(ξ) in [Che12] where ξ is a recipe.

Raises Internal_error if the path of xi is not closed or if the path is not defined.

val apply_function_to_path : Term.symbol -> path -> path

apply_function_to_path f p returns the path $f \cdot p$.

val axiom_path : axiom -> path

axiom_path ax returns the path ax.

Testing path

```
val is_equal_path : path -> path -> bool
  is_equal_path p1 p2 returns true iff p1 and p2 are the same path.

val is_recipe_same_path : recipe -> recipe -> bool
  is_recipe_same_path r1 r2 returns true iff the paths of r1 and of r2 are the same. Note that
  two recipes having the same path does not imply that the recipes are equal.

val is_path_of_recipe : recipe -> path -> bool
  is_path_of_recipe r p returns true iff the path of r is p.
```

Display

```
val display_path : path -> string
```

1.2.5 Recipe context

type context

The type `context` corresponds to the set $\mathcal{T}(\mathcal{F}_c, \mathcal{F}_d^* \cdot \mathcal{AX} \cup \mathcal{X}^2)$ in [Che12]. The context of a recipe, defined in [Che12, Definition 7.6], is used in the algorithm for dealing with the inequations.

```
val context_of_recipe : recipe -> context
  context_of_recipe r returns the context of the recipe r following [Che12, Definition 7.6]. Note
  that in this definition, a frame is needed as parameter. But since we consider context with only
  constructor function symbol as application function, such frame is not necessary.

val recipe_of_context : context -> recipe
  recipe_of_context c transforms the context c as a recipe if c is included in  $\mathcal{T}(\mathcal{F}, \mathcal{X}^2)$ . c cannot
  contain a path since one cannot reconstruct a recipe from a path.
  Raises Internal_error if c is not included in  $\mathcal{T}(\mathcal{F}, \mathcal{X}^2)$ .

val path_of_context : context -> path
val top_context : context -> Term.symbol
val apply_substitution_on_context :
  substitution ->
  'a -> ('a -> (context -> context) -> 'a) -> 'a
  apply_substitution_on_context theta elt map_elt first transforms the substitution
   $\theta = \{X_i \mapsto \xi_i\}_i$  into a substitution  $\theta' = \{X_i \mapsto \gamma_i\}_i$  where gamma_i is the result of
  context_of_recipe xi_i. Then it applies the substitution theta' on the elt. The function
  map_elt should map the contexts contained in the element elt on which theta' should be
  applied.
```

Testing

```
val is_variable_context : context -> bool
  is_variable_context c returns true iff c is a variable, i.e. is in  $\mathcal{X}^2$ .

val is_path_context : context -> bool
  is_path_context c returns true iff c is a path, i.e. is in  $\mathcal{F}^* \cdot \mathcal{AX}$ .

val is_closed_context : context -> bool
  is_closed_context c returns true iff c is closed, i.e. is in  $\mathcal{T}(\mathcal{F}, \mathcal{F}^* \cdot \mathcal{AX})$ .

val exists_path_in_context : context -> bool
  is_closed_context c returns true iff there exists a path subterm of c.
```

Access

`val get_max_param_context : context -> int`

`get_max_param_context c` returns the maximal parameter of the recipe context `c`, defined in [Che12, Section 7.4.2.2] and denoted $\text{param}_{\max}^{\mathcal{C}}(c)$ where \mathcal{C} is a constraint system. Note that our function does not have a constraint system as argument. Indeed, the purpose of the constraint system is to allow the association support/variable in [Che12] which is coded directly in the variables in this module.

Display

`val display_context : context -> string`

1.2.6 Formula on contexts of recipes

`type formula`

The type `formula` correspond to a disjunction of inequation between context of recipe. It corresponds to the formulas contains in the association table in [Che12, Section 7.4.2.2].

`exception Removal_transformation`

This exception will be triggerred when a formula will satisfy the removal transformation described in [Che12, Section 7.4.2.5].

`val create_formula : variable -> recipe -> formula`

`create_formula x xi` creates the formula $X \stackrel{?}{\neq} C[\xi]$.

Scanning

`val for_all_formula : (context * context -> bool) -> formula -> bool`

`for_all_formula f phi` returns true iff `f xi_i beta_i` returns true for all `i` where `phi` is the formula $\bigvee_i \xi_i \stackrel{?}{\neq} \beta_i$.

`val exists_formula : (context * context -> bool) -> formula -> bool`

`exists_formula f phi` returns true iff there exists `i` s.t. `f xi_i beta_i` returns true where `phi` is the formula $\bigvee_i \xi_i \stackrel{?}{\neq} \beta_i$.

`val find_and_apply_formula :`

`(context -> context -> bool) ->`

`(context -> context -> 'a) ->`

`(unit -> 'a) -> formula -> 'a`

`find_and_apply_formula f_test f_apply f_no formula` searches in `formula` an inequation satisfying `f_test`. If such inequation exists then it applies `f_apply` on it else it apply the function `f_no`.

Note that since an inequation $\xi \stackrel{?}{\neq} \beta$ is semantically the same as $\beta \stackrel{?}{\neq} \xi$, it is recommanded that `f_test xi beta` and `f_test beta xi` are equal. Same for `f_apply`.

Modification

val apply_substitution_on_formulas :

substitution ->

'a -> ('a -> (formula -> formula) -> 'a) -> 'a

apply_substitution_on_formulas theta elt map_elt first transforms a substitution $\theta = \{X \mapsto \xi\}$ into a substitution $\theta' = \{X \mapsto \gamma\}$ where gamma is the result of context_of_recipe xi. Then it applies the substitution theta' on the formulas of elt. The function map_elt should map the formulas contained in the element elt on which theta' should be applied.

Raises Internal_error if the domain of theta is different from a singleton.

val simplify_formula : formula -> formula

simplify_formula phi returns the formula phi simplified as detailed in [Che12, Section 7.4.2.2].

Raises

- **Internal_error** if phi can be simplified into a formula $f(\beta_1, \dots, \beta_n) \stackrel{?}{\neq} g(\beta'_1, \dots, \beta'_m) \vee \Phi'$ for some $f \neq g$.
- **Removal_transformation** if phi can be simplified into a formula of the form $\bigvee_i \xi_i \stackrel{?}{\neq} \beta_i$ where for all i , $\xi_i \in \mathcal{F}_d^* \cdot \mathcal{AX}$ or $\beta_i \in \mathcal{F}_d^* \cdot \mathcal{AX}$.

val apply_simplify_substitution_on_formulas :

substitution ->

'a -> ('a -> (formula -> formula) -> 'a) -> 'a

apply_simplify_substitution_on_formulas theta elt map_elt returns the same as simplify_formula (apply_substitution_on_formulas theta elt map_elt) but computes it more quickly.

val display_formula : formula -> string

1.3 Module Constraint : Frame and deducibility constraints

This module regroups all the functions that manipulate the deducibility constraints and the frame. Hence it corresponds to the elements of the form $X, i \stackrel{?}{\vdash} u$ and $\xi, j \triangleright v$ in [Che12, Chapter 7,8].

1.3.1 Support set

type 'a support_set

Both frame and deducibility constraints are theoretically a sets of elements of the form $X, i \stackrel{?}{\vdash} u$ and $\xi, j \triangleright v$ in [Che12]. However, both of these element depend of a support, i.e. an integer. Hence to improve the efficiency of our algorithm, the type support_set is an optimised set of element parametrised by an integer.

type position

The type position corresponds to the specific position of an element in a support_set. It is used to speed-up the access to element of a support_set.

val empty_set : 'a support_set

empty_set is an empty support set.

Modification

```
val add : ('a -> int) -> 'a -> 'a support_set -> 'a support_set
  add f elt set add the element elt with support f elt in the set set. f should correspond to
  the function that return the support of elt.

val add_new_support : (int -> 'a) -> 'a support_set -> 'a support_set
  add_new_support f set add the element f s in set where s-1 is the support maximal of the
  element in set.

val add_list : ('a -> int) ->
  'a list -> 'a support_set -> 'a support_set
  add_list f elt_list set add the elements in elt_list in the set set. f should correspond to
  the function that return the support of the elements of elt_list.

  Raises Internal_error if the application of f on the elements of elt_list does not return the
  same value. [Low debugging]

val replace :
  position ->
  ('a -> 'a list) ->
  'a support_set -> 'a * 'a support_set
  replace p f set replace the element in set at the position p by the elements f elt if elt is the
  element in set at the position p.

  Raises Internal_error if the position p does not correspond to any element in set.

val replace2 :
  position ->
  ('a -> 'a list * 'a list) ->
  'a support_set ->
  'a * 'a support_set * 'a support_set
  replace2 p f set returns two sets set1, set2 where set1 (resp. set2) is the set set where the
  element elt at the position p in set is replaced by elt_11 (resp. elt_12) with elt_11, elt_12
  being the result of f elt.

  Raises Internal_error if the position p does not correspond to any element in set.
```

Scanning

```
type support_range =
  | SUnique of int
    SUnique s Consider only the elements of support s.
  | SAll
    Consider all the elements in the set.
  | SUntil of int
    SUntil s considers only the elements of support inferior or equal to s.
  | SFrom of int
    SFrom s considers only the elements of support superior or equal to s.
  | SBetween of int * int
    SBetween s1 s2 considers only the elements of support superior or equal to s1, and
    inferior or equal to s2.

support_range is a parameter for scanning function. It allows more efficient and precise search
on the sets.
```

```

val search :
  support_range ->
  ('a -> bool) -> 'a support_set -> 'a * position
  search s_range test set returns elt,pos where elt is an element in set whose support
  satisfies s_range and such that test elt returns true. pos is the position of elt in set.
  Raises Not_found if no element of set satisfies the function test.

val search_and_replace :
  support_range ->
  ('a -> bool) ->
  ('a -> 'a list) ->
  'a support_set ->
  'a * position * 'a support_set
  search_and_replace s_range test f set is an optimisation of { let (elt,pos) = search
  s_range test set in elt,pos, replace pos f set}

val search_and_replace2 :
  support_range ->
  ('a -> bool) ->
  ('a -> 'a list * 'a list) ->
  'a support_set ->
  'a * position * 'a support_set *
  'a support_set
  search_and_replace2 s_range test f set is an optimisation of { let (elt,pos) = search
  s_range test set in let set1,set2 = replace2 pos f set in elt,pos,set1,set2}

val for_all : support_range -> ('a -> bool) -> 'a support_set -> bool
  for_all s_range test set returns true iff for all elements elt in set whose support satisfies
  s_range, test elt returns true.

val exists : support_range -> ('a -> bool) -> 'a support_set -> bool
  exists s_range test set returns true iff there exists an element elt in set whose support
  satisfies s_range and such that test elt returns true.

```

Access

```

val get : position -> 'a support_set -> 'a
  get pos set returns the element of set at the position pos.
  Raises Internal_error if pos is not a position in set.

```

Iterators

```

val iter : support_range -> ('a -> unit) -> 'a support_set -> unit
  iter s_range f set is f e1; ...; f en where apply the function f to all elements of set
  whose support satisfies s_range. The order on the element on which f is applied is by increasing
  support first and then in the order in which they were added in the set.

  Note that the function replace modifies the order in which elements are added: For example,
  consider a set set of elements with same support such that elt1, elt2, elt3 was added in this
  particular order by call the function add. Consider pos2 the position of elt2 in set and the
  function g = fun e -> [e;e]. We have that iter SAll f (replace pos2 g set) is f elt1;
  f elt2; f elt2; f elt3.

```

```

val map : support_range ->
  ('a -> 'a) -> 'a support_set -> 'a support_set

```

`map s_range f set` returns the set `set` where the function `f` was applied on all the elements of `set` satisfying `s_range`.

```
val fold_left :
  support_range ->
  ('a -> 'b -> 'a) -> 'a -> 'b support_set -> 'a

val iter2 :
  support_range ->
  ('a -> 'a -> unit) ->
  'a support_set -> 'a support_set -> unit
  iter2 s_range f set1 set2 is f e1 d1; f e2 d2; ...; fen dn where set1 (resp. set2) is a
  set whose elements satisfying s_range have the application order e1;...; en (resp. d1;...;dn).
  See Constraint.iter for more details on the application order.

Raises Internal_error if set1 and set2 do not have the same number of elements of equal
  support.
```

Display

```
val display_horizontally : ('a -> string) -> 'a support_set -> string
val display_vertically : ('a -> string) -> string -> 'a support_set -> string
```

1.3.2 Frame

In [Che12], a frame is a set $\{\xi_1, i_1 \triangleright u_1; \dots; \xi_n, i_n \triangleright u_n\}$ where $\xi_j \in \mathcal{T}(\mathcal{F}, \mathcal{AX} \cup \mathcal{X}^2)$, $\text{path}(\xi_j)$ exists and $u_j \in T(\mathcal{F}_c, \mathcal{N} \cup \mathcal{X}^1)$ for all $j \in \{1, \dots, n\}$. Note that compare to [Che12], a frame in this implementation is extended by the addition of some flags which will represents different notions used later on in the constraint systems.

```
module Frame :
```

```
  sig
```

```
    type elt
```

A frame constraint represents in [Che12] an element of the form $\xi, i \triangleright_F u$ with ξ a recipe, i a integer, u a constructor term and F a set of flags.

```
    val create : Recipe.recipe -> int -> Term.term -> elt
```

`create frame_constraint xi s m` returns the frame constraint $\xi, s \triangleright_{\emptyset} m$.

Raises Internal_error if `m` is not a constructor term. **[High debugging]**

Access

```
    val get_recipe : elt -> Recipe.recipe
```

`get_recipe fc` returns the recipe of `fc`.

```
    val get_support : elt -> int
```

`get_suport fc` returns the support of `fc`.

```
    val get_message : elt -> Term.term
```

`get_message fc` returns the message of `fc`.

Modification

```

val replace_recipe : elt ->
  (Recipe.recipe -> Recipe.recipe) -> elt

  replace_recipe fc rep returns the frame constraint fc with the recipe rep r where r was
  the recipe of fc.

val replace_message : elt -> (Term.term -> Term.term) -> elt

  replace_message fc rep returns the frame constraint fc with the message rep m where m
  was the message of fc.

```

Flags

In [Che12], the notion of flag does not exist. However they correspond to other elements or properties of constraint systems. Hence, we will give the semantics of each flag when introducing their adding function. For this, we will consider a constraint system \mathcal{C} , its associated frame Φ and let fc be a frame element $(\xi, i \triangleright_F u) \in \Phi$.

```

val add_noDedSubterm : elt -> Term.symbol -> int -> elt

  add_noDedSubterm fc f s adds a flag NODEDSUBTERM( $f, s$ )  $\in \mathcal{F}$  with  $f \in \mathcal{F}_c$ . It
  corresponds to the non-deducibility constraint  $f(x_1, \dots, x_n) \stackrel{?}{\neq} u \vee s \stackrel{?}{\not\vdash} x_1 \vee \dots \vee s \stackrel{?}{\not\vdash} x_n$  where
   $x_1, \dots, x_n$  are fresh variables.

```

Raises

- **Internal_error** if a flag NOUSE is already in \mathcal{F} . **[Low debugging]**
- **Internal_error** if a flag YESDEDSUBTERM(g, s') was already in F for any g, s' except when $g = f$ and $s < s'$.
- **Internal_error** if f is not a constructor function symbol or if it is a tuple. **[Low debugging]**
- **Internal_error** if $u \in \mathcal{X}^1$ **[Low debugging]**.

```

val add_yesDedSubterm : elt -> Term.symbol -> int -> elt

  add_YesDedSubterm fc f s adds a flag YESDEDSUBTERM( $f, s$ )  $\in \mathcal{F}$ . It corresponds to
  there exists  $X_1, \dots, X_n \in vars^2(\mathcal{C})$  such that for all  $i \in \{1, \dots, n\}$ ,  $param_{\max}^{\mathcal{C}}(X_i \theta) \leq s$  and

```

$$C[f(X_1, \dots, X_n)\theta]_{\Phi} acc^1(\mathcal{C}) = v$$

Intuitively, it indicates that u can be constructed in \mathcal{C} by applying f with support inferior or equal to s .

Raises

- **Internal_error** if a flag NODEDSUBTERM(f, s') or NOUSE was already in F with $s \leq s'$. **[Low debugging]**
- **Internal_error** if f is not a constructor function symbol or if it is a tuple. **[Low debugging]**
- **Internal_error** if $u \in \mathcal{X}^1$ **[Low debugging]**.

```

val add_noDest : elt -> Term.symbol -> int -> elt

  add_noDest fc f s adds a flag NODEST( $f, s$ )  $\in \mathcal{F}$  with  $f \in \mathcal{F}_d$ . It corresponds to the
  non-deducibility constraint  $\forall \tilde{x}. u \stackrel{?}{\neq} v_1 \vee s \stackrel{?}{\not\vdash} v_2 \vee \dots \vee s \stackrel{?}{\not\vdash} v_n$  where  $f(v_1, \dots, v_n) \rightarrow w$  is a
  fresh rewrite rule with  $\tilde{x} = vars(v_1)$ .

```

Raises

- **Internal_error** if a flag YESDEST or NOUSE was already in F

- `Internal_error` if `f` is not a destructor function symbol. **[Low debugging]**
- `Internal_error` if `f` is a projection function symbol. **[High debugging]**
- `Internal_error` if $u \in \mathcal{X}^1$ **[Low debugging]**.

`val add_yesDest : elt -> elt`

`add_yesDest fc` adds a flag `YESDEST` $\in F$. It corresponds to the fact that there exists $(\zeta, k \triangleright v) \in \Phi$ such that $\text{path}(\zeta) = g \cdot \text{path}(\xi)$ and `Term.link_destruc_construc g f` returns `true` where $g = \text{root}(u)$.

Raises

- `Internal_error` if a flag `NOUSE` is already in \mathcal{F} .
- `Internal_error` if $u \in \mathcal{X}^1$ **[Low debugging]**.

`val add_noUse : elt -> elt`

`add_noUse fc` adds a flag `NOUSE` $\in F$. It corresponds to the fact that $(\xi, i \triangleright u) \in \text{NoUse}(\mathcal{C})$.

`val is_noDedSubterm : elt -> int -> bool`

`is_noDedSubterm fc s` returns `true` iff there is a flag `NODEDSUBTERM`(f, s') $\in F$ with $s \leq s'$ where $f = \text{root}(u)$.

`val is_yesDedSubterm : elt -> int -> bool`

`is_yesDedSubterm fc s` returns `true` iff there is a flag `YESDEDSUBTERM`(f, s') $\in F$ with $s \geq s'$ where $f = \text{root}(u)$.

`val is_noDest : elt -> int -> bool`

`is_noDest fc s` returns `true` iff there is a flag `NODEST`(f, s') $\in F$ with $s \leq s'$ where f is the corresponding destructor of $\text{root}(u)$.

`val is_yesDest : elt -> bool`

`is_yesDest fc` returns `true` iff there is a flag `YESDEST` $\in F$ where $f = \text{root}(u)$.

`val is_noUse : elt -> bool`

`is_noUse fc` returns `true` iff there is a flag `NOUSE`.

Testing on frame

`val is_same_structure :`

`elt Constraint.support_set ->`

`elt Constraint.support_set -> bool`

`is_same_structure frame1 frame2` checks that every couple of frame constraints in `frame1` and `frame2` of same application order have:

- the same recipe
- the same support
- the same set of flags

Display

`val display : elt -> string`

`display fc` display the frame constraint without considering the flags.

`end`

1.3.3 Deducibility constraint

In [Che12], the deducibility constraints are element of the form $X, i \vdash^? u$ where $X \in \mathcal{X}^2$, $i \in \mathbb{N}$ and $u \in \mathcal{T}(\mathcal{F}_c, \mathcal{N} \cup \mathcal{X}^1)$. Note that compare to [Che12], a deducibility constraints in this implementation is extended by the addition of some flags which will represents different notions used later on in the constraint systems.

module Deducibility :

sig

type elt

val create : Recipe.variable -> int -> Term.term -> elt

create v s t creates a deducibility constraint with the variable v, the support s and the term t.

Raises

- Internal_error if t is not a constructor term. **[High debugging]**
- Internal_error if s is different from the support of v.

Access

val get_recipe_variable : elt -> Recipe.variable

get_recipe_variable dc returns the recipe variable of dc.

val get_support : elt -> int

get_support dc returns the support of dc.

val get_message : elt -> Term.term

get_message dc returns the message of dc.

Modification

val replace_message : elt ->

(Term.term -> Term.term) -> elt

replace_message cc rep returns the deducibility constraint dc with the message rep m where m was the message of dc.

Flags

Similarly to the module [Frame], the flags in deducibility constraint correspond to other elements or properties of constraint systems. Hence, we will give the semantics of each flag when introducing their adding function. For this, we will consider a constraint system \mathcal{C} , its associated deducibility constraint set D and let dc be a deducibility constraint $(X, i \vdash_F^? u) \in D$.

val add_noCons : elt -> Term.symbol -> elt

add_noCons dc f adds the flag NOCONS(f) $\in F$. It corresponds to the inequation

$\text{root}((\)X) \neq^? f$.

Raises Internal_error if the flag was already added. **[Low debugging]**

val add_noAxiom : elt ->

Constraint.position -> elt

`add_noAxiom dc p` add the flag $\text{NOAXIOM}(p) \in \mathcal{F}$. It corresponds to the inequation $X \stackrel{?}{\neq} \xi$ where $(\xi, j \triangleright v)$ is the frame constraint in $\Phi(\mathcal{C})$ at the position `p`. Note: the flags `NOAXIOM` must be added by the rule `AXIOM` for all cases except when a `NOUSE` is detected.
Raises `Internal_error` if the flag was already added. **[Low debugging]**

```
val compare_noCons : elt ->
  elt -> Term.symbol list * Term.symbol list

  compare_noCons dc1 dc2 compare the flags NOCONS in dc1 and dc2. It returns a pair of
  set of function symbols  $(S_1, S_2)$  where:
    • for all  $f \in S_1$ ,  $\text{NOCONS}(f) \in \mathcal{F}_2$  but  $\text{NOCONS}(f) \notin \mathcal{F}_1$ .
    • for all  $f \in S_2$ ,  $\text{NOCONS}(f) \in \mathcal{F}_1$  but  $\text{NOCONS}(f) \notin \mathcal{F}_2$ .

val compare_noAxiom :
  elt ->
  elt ->
  int -> Constraint.position list * Constraint.position list

  compare_noAxiom c1 c2 s compare the flags NOAXIOM in dc1 and dc2. It returns a pair
  of set of position  $(P_1, P_2)$  where:
    • for all  $p \in P_1$  of support  $s$ ,  $\text{NOAXIOM}(p) \in \mathcal{F}_2$  but  $\text{NOAXIOM}(p) \notin \mathcal{F}_1$ .
    • for all  $p \in P_2$  of support  $s$ ,  $\text{NOAXIOM}(p) \in \mathcal{F}_1$  but  $\text{NOAXIOM}(p) \notin \mathcal{F}_2$ .

val fold_left_frame_free_of_noAxiom :
  elt ->
  ('a -> Constraint.Frame.elt -> 'a) ->
  'a -> Constraint.Frame.elt Constraint.support_set -> 'a

  fold_left_frame_free_of_noAxiom dc f acc frame is similar to fold_left (SUntil s)
  f acc frame but f is only applied to the element of position pos of frame such that the flag
  NoAxiom pos is not in dc. Moreover, s is the support of dc
```

Scanning

```
val is_all_noCons : elt -> bool

  is_all_noCons dc returns true iff the flags NoCons f is in dc for all constructors f.

val is_same_structure :
  elt Constraint.support_set ->
  elt Constraint.support_set -> bool

  is_same_structure dc_set1 dc_set2 checks that every couple of deducibility constraints
  in dc_set1 and dc_set2 of same application order have:
    • the same variable
    • the same support
    • the same set of flags

val is_noCons : elt -> Term.symbol -> bool

val is_unsatisfiable :
  Constraint.Frame.elt Constraint.support_set ->
  elt -> bool
```

Display

```
val display : elt -> string

end
```

1.4 Module `Constraint_system` : Operations on (matrices of) constraint systems

This module regroups all the functions that manipulate the constraint systems and the matrices of constraint system. In [Che12], there are several definitions of constraint systems but we are only interested in the constraint system of [Che12, Chapter 7].

1.4.1 Constraint system

`type constraint_system`

`constraint_system` corresponds to [Che12, Definition 7.6] . Moreover, it will contain additional information used in the algorithm such as association table (see [Che12, Section 7.4.2.2]).

`val empty : constraint_system`

`empty` is the constraint system that accept any solution. It does not contain any deducibility constraint, nor frame constraint, nor equation, nor inequation.

`val bottom : constraint_system`

`bottom` is the constraint system with no solution. It corresponds to \perp in [Che12].

Iterators

`val map_message_inequation :`

`(Term.formula -> Term.formula) ->`
`constraint_system -> constraint_system`

Modification functions

`val add_message_equation :`

`constraint_system ->`
`Term.term -> Term.term -> constraint_system`

`add_message_equation csys t1 t2` returns the constraint system `csys` with the added equation $t_1 \stackrel{?}{=} t_2$.

`val add_message_formula :`

`constraint_system ->`
`Term.formula -> constraint_system`

`add_message_formula csys phi` returns the constraint system `csys` with the added message formula `phi`

`val add_new_deducibility_constraint :`

`constraint_system ->`
`Recipe.variable -> Term.term -> constraint_system`

`add_new_deducibility_constraint csys X t` returns the constraint system `csys` with the added deducibility constraint $X, i \stackrel{?}{\vdash} t$ where i is the maximal support of `csys`.

Raises

- `Internal_error` if `csys` is the bottom constraint system.
- `Internal_error` if `t` is not a constructor term. **[High debugging]**
- `Internal_error` if the support associated to `X` is not equal to the maximal support of `csys`.

```

val add_deducibility_constraint :
  constraint_system ->
  Constraint.Deducibility.elts list -> constraint_system
val add_new_axiom : constraint_system ->
  Term.term -> constraint_system
  add_new_axiom csys t returns the constraint system csys with the frame  $\Phi \cup \{ax_i, i \triangleright t\}$  where
   $\Phi$  is the frame of csys and  $i$  is the maximal support of  $\Phi$  .
  Raises Internal_error if csys is the bottom constraint system.

val add_frame_constraint :
  constraint_system ->
  Constraint.Frame.elts list -> constraint_system
val frame_replace :
  constraint_system ->
  Constraint.position ->
  (Constraint.Frame.elts -> Constraint.Frame.elts list) ->
  Constraint.Frame.elts * constraint_system
  frame_replace c p f replace the element in the frame of c at the position p by the elements f
  elt if elt is the element in the frame of c at the position p.
  Raises Internal_error if the position p does not correspond to any element in the frame of c.

val frame_replace2 :
  constraint_system ->
  Constraint.position ->
  (Constraint.Frame.elts ->
    Constraint.Frame.elts list * Constraint.Frame.elts list) ->
  Constraint.Frame.elts * constraint_system *
  constraint_system
  frame_replace2 c p f returns two constraint systems c1,c2 where c1 (resp. c2) is the
  constraint system c where the element elt at the position p in the frame of c is replaced by
  elt_l1 (resp. elt_l2) with elt_l1,elt_l2 being the result of f elt.
  Raises Internal_error if the position p does not correspond to any element in the frame of c.

val frame_search_and_replace :
  constraint_system ->
  Constraint.support_range ->
  (Constraint.Frame.elts -> bool) ->
  (Constraint.Frame.elts -> Constraint.Frame.elts list) ->
  Constraint.Frame.elts * Constraint.position *
  constraint_system
  frame_search_and_replace c s_range test f is an optimisation of { let (elt,pos) =
  Constraint.search s_range test (get_frame c) in elt,pos, frame_replace c pos f}

val frame_search_and_replace2 :
  constraint_system ->
  Constraint.support_range ->
  (Constraint.Frame.elts -> bool) ->
  (Constraint.Frame.elts ->
    Constraint.Frame.elts list * Constraint.Frame.elts list) ->
  Constraint.Frame.elts * Constraint.position *
  constraint_system * constraint_system
  frame_search_and_replace2 c s_range test f is an optimisation of { let (elt,pos) =
  Constraint.search s_range test (get_frame c) in let set1,set2 = frame_replace2 c
  pos f in elt,pos,set1,set2}

```

Access functions

```
val get_deducibility_constraint_set :  
  constraint_system ->  
  Constraint.Deducibility.elts Constraint.support_set  
  get_deducibility_constraint_set csys returns the set of deducibility constraints of csys.  
  Raises Internal_error if csys is the bottom constraint system.  
  
val get_frame :  
  constraint_system ->  
  Constraint.Frame.elts Constraint.support_set  
  get_frame csys returns the frame of csys.  
  Raises Internal_error if csys is the bottom constraint system.  
  
val get_message_equations : constraint_system -> (Term.term * Term.term) list  
  get_message_equations csys returns the list [(u_1,v_1);...;(u_n,v_n)] where  $\bigwedge_{i=1}^n u_i \stackrel{?}{=} v_i$   
  is the conjunction of equations between constructor terms in csys.  
  Raises Internal_error if csys is the bottom constraint system.  
  
val get_recipe_equations :  
  constraint_system -> (Recipe.recipe * Recipe.recipe) list  
  get_recipe_equations csys returns the list [(xi_1,zeta_1);...;(xi_n,zeta_n)] where  
   $\bigwedge_{i=1}^n \xi_i \stackrel{?}{=} \zeta_i$  is the conjunction of equations between recipes in csys.  
  Raises Internal_error if csys is the bottom constraint system.  
  
val get_maximal_support : constraint_system -> int  
  get_maximal_support csys returns maximal support of the frame of csys.  
  Raises Internal_error if csys is the bottom constraint system.
```

Testing functions

```
val is_semi_solved_form : constraint_system -> bool  
val set_semi_solved_form : constraint_system -> constraint_system  
val unset_semi_solved_form : constraint_system -> constraint_system  
val is_no_universal_variable : constraint_system -> bool  
val set_no_universal_variable : constraint_system -> constraint_system  
val unset_no_universal_variable : constraint_system -> constraint_system  
val is_bottom : constraint_system -> bool  
  is_bottom c returns true iff c is the constraint system  $\perp$ .  
  
val check_same_structure : constraint_system ->  
  constraint_system -> unit  
  check_same_structure c1 c2 does nothing if c1 and c2 have same structure else it raises the  
  exception Internal_error. The definition of structure is given in [Che12, Section 7.1.2].  
  
val check_same_shape : constraint_system ->  
  constraint_system -> unit  
  check_same_shape c1 c2 does nothing if c1 and c2 have same shape else it raises the exception  
  Internal_error. The definition of shape is given in [Che12, Definition 7.11].  
  
val display : constraint_system -> string  
val is_unsatisfiable : constraint_system -> bool
```

1.4.2 Fonctionnalités of Phase 1

In the strategy on the rules described in [Che12, Section 7.4], there are two different phases of rule application. Hence this section describes the optimised functions used in Phase 1 of the strategy. Due to the lack of invariant during this phase, these functions are quite general.

```
module Phase_1 :
```

```
sig
```

```
val activate_phase :  
  Constraint_system.constraint_system -> Constraint_system.constraint_system  
  
  activate_phase csys returns the constraint system csys optimised for Phase 1 of the  
  strategy.
```

Modifications

```
val deducibility_replace :  
  Constraint_system.constraint_system ->  
  Constraint.position ->  
  (Constraint.Deducibility.elt -> Constraint.Deducibility.elt list) ->  
  Constraint.Deducibility.elt * Constraint_system.constraint_system  
  
  deducibility_replace c p f replace the deducibility constraint of c at the position p by  
  the deducibility constraints f dc if dc is the deducibility constraint of c at the position p.  
  
  Raises Internal_error if the position p does not correspond to any deducibility constraint  
  in c.
```

```
val deducibility_replace2 :  
  Constraint_system.constraint_system ->  
  Constraint.position ->  
  (Constraint.Deducibility.elt ->  
   Constraint.Deducibility.elt list * Constraint.Deducibility.elt list) ->  
  Constraint.Deducibility.elt * Constraint_system.constraint_system *  
  Constraint_system.constraint_system  
  
  deducibility_replace2 c p f returns two constraint systems c1,c2 where c1 (resp. c2)  
  is the constraint system c where the deducibility constraint dc at the position p in c is  
  replaced by dc_l1 (resp. dc_l2) with dc_l1,dc_l2 being the result of f dc.  
  
  Raises Internal_error if the position p does not correspond to any deducibility constraint  
  in c.
```

```
val deducibility_search_and_replace :  
  Constraint_system.constraint_system ->  
  Constraint.support_range ->  
  (Constraint.Deducibility.elt -> bool) ->  
  (Constraint.Deducibility.elt -> Constraint.Deducibility.elt list) ->  
  Constraint.Deducibility.elt * Constraint.position *  
  Constraint_system.constraint_system  
  
  deducibility_search_and_replace c s_range test f is an optimisation of { let  
    (elt,pos) = Constraint.search s_range test (get_deducibility_constraint_set  
    c) in elt,pos, deducibility_replace c pos f}
```

```
val deducibility_search_and_replace2 :  
  Constraint_system.constraint_system ->  
  Constraint.support_range ->  
  (Constraint.Deducibility.elt -> bool) ->  
  (Constraint.Deducibility.elt ->
```



```

    Constraint.Deducibility.elt list * Constraint.Deducibility.elt list) ->
    Constraint.Deducibility.elt * Constraint.position *
    Constraint_system.constraint_system * Constraint_system.constraint_system

    deducibility_search_and_replace2 c s_range test f is an optimisation of { let
    (elt,pos) = Constraint.search s_range test (get_deducibility_constraint_set
    c) in let set1,set2 = deducibility_replace2 c pos f in elt,pos,set1,set2}

```

Substitution

```

val unify_and_apply_message_equations :
    Constraint_system.constraint_system ->
    (Term.term * Term.term) list -> Constraint_system.constraint_system

    unify_and_apply_message_equations csys eq_1 returns the normalised constraint
    system csys on which the most general unifier of eq_1 was applied.
    Raises Term.Not_unifiable if eq_1 is no unifiable.

val apply_message_substitution :
    Constraint_system.constraint_system ->
    Term.substitution -> Constraint_system.constraint_system

    apply_message_equations csys subst returns the normalised constraint system csys on
    which subst was applied.

val apply_recipe_substitution :
    Constraint_system.constraint_system ->
    Recipe.substitution -> Constraint_system.constraint_system

    apply_recipe_substitution csys subst returns the normalised constraint system csys
    on which subst was applied.
    Raises Internal_error if the domain of subst intersects with the left hand side variables
    of csys. [High debugging]

val normalise :
    Constraint_system.constraint_system -> Constraint_system.constraint_system

    normalise csys returns the constraint system csys normalised. It may contain destructors
    function symbol in inequations and equations. This normalisation corresponds to the
    transformation induced by [Che12, Lemma 6.10].

end

```

1.4.3 Fonctionnalités of Phase 2

As mention in the previous section, there are two different phases of rule application described in the strategy (see [Che12, Section 7.4]). This section describes the optimised functions used in Phase 2 of the strategy. These functions will benefit from the fact that the right hand term of constraint system are variables. On the other hand, they consider the association tables in the constraint system.

```

module Phase_2 :
sig
    val activate_phase :
        Constraint_system.constraint_system -> Constraint_system.constraint_system

        activate_phase csys returns the constraint system csys optimised for Phase 2 of the
        strategy.

```

```

val add_message_inequation :
  Constraint_system.constraint_system ->
  Term.variable ->
  Term.term ->
  Recipe.variable -> Recipe.recipe -> Constraint_system.constraint_system

```

Modifications

```

val deducibility_replace :
  Constraint_system.constraint_system ->
  Constraint.position ->
  (Constraint.Deducibility.elts -> Constraint.Deducibility.elts list) ->
  Constraint.Deducibility.elts * Constraint_system.constraint_system

  See Phase_1.deducibility_replace.

```

```

val deducibility_replace2 :
  Constraint_system.constraint_system ->
  Constraint.position ->
  (Constraint.Deducibility.elts ->
   Constraint.Deducibility.elts list * Constraint.Deducibility.elts list) ->
  Constraint.Deducibility.elts * Constraint_system.constraint_system *
  Constraint_system.constraint_system

  See Phase_1.deducibility_replace2.

```

```

val deducibility_search_and_replace :
  Constraint_system.constraint_system ->
  Constraint.support_range ->
  (Constraint.Deducibility.elts -> bool) ->
  (Constraint.Deducibility.elts -> Constraint.Deducibility.elts list) ->
  Constraint.Deducibility.elts * Constraint.position *
  Constraint_system.constraint_system

  See Phase_1.deducibility_search_and_replace.

```

```

val deducibility_search_and_replace2 :
  Constraint_system.constraint_system ->
  Constraint.support_range ->
  (Constraint.Deducibility.elts -> bool) ->
  (Constraint.Deducibility.elts ->
   Constraint.Deducibility.elts list * Constraint.Deducibility.elts list) ->
  Constraint.Deducibility.elts * Constraint.position *
  Constraint_system.constraint_system * Constraint_system.constraint_system

  See Phase_1.deducibility_search_and_replace2.

```

Substitution

```

val unify_and_apply_message_equations :
  Constraint_system.constraint_system ->
  (Term.term * Term.term) list -> Constraint_system.constraint_system

  See Phase_1.unify_and_apply_message_equations.

```

```

val apply_message_substitution :
  Constraint_system.constraint_system ->
  Term.substitution -> Constraint_system.constraint_system

```

See `Phase_1.apply_message_substitution`.

```
val apply_recipe_substitution :
  Constraint_system.constraint_system ->
  Recipe.substitution -> Constraint_system.constraint_system
```

See `Phase_1.apply_recipe_substitution`.

Access functions

```
val term_of_recipe :
  Constraint_system.constraint_system -> Recipe.recipe -> Term.term

  term_of_recipe c xi returns the term  $\xi_{\text{acc}^1(\mathcal{C})}$ .
```

Raises

- `Internal_error` if $\xi \notin \mathcal{T}(\mathcal{F}_c, \mathcal{X}^2)$
- `Not_found` if $\text{vars}^2(\xi) \setminus \text{vars}^2(D(\mathcal{C})) \neq \emptyset$.

```
val recipe_of_term :
  Constraint_system.constraint_system -> Term.term -> Recipe.recipe

  recipe_of_term c t returns the recipe  $\xi$  such that  $\xi_{\text{acc}^1(\mathcal{C})} = t$ .
```

Raises

- `Internal_error` if $t \notin \mathcal{T}(\mathcal{F}_c, \mathcal{X}^1)$
- `Not_found` if $\text{vars}^1(\xi) \setminus \text{vars}^1(D(\mathcal{C})) \neq \emptyset$.

```
val get_max_param_context :
  Constraint_system.constraint_system -> Recipe.recipe -> int

  get_max_param_context c xi returns the integer  $\text{param}_{\max}^{\mathcal{C}}(\mathcal{C}[\xi]_c)$ .
```

```
val get_max_param_context_from_term :
  Constraint_system.constraint_system -> Term.term -> int

  get_max_param_context_from_term c t returns the same result as
  get_max_param_context c (recipe_of_term c t) but is more efficient.
```

Formula inequation functions

```
val map_message_inequations :
  (Term.formula ->
   Recipe.formula option -> Term.formula * Recipe.formula option) ->
  Constraint_system.constraint_system -> Constraint_system.constraint_system

val fold_left_message_inequation :
  ('a -> Term.formula -> Recipe.formula option -> 'a) ->
  'a -> Constraint_system.constraint_system -> 'a
```

end

1.4.4 Row matrix of constraint system

The types `vector` and `matrix` corresponds to the vectors and matrices of constraint systems used in [Che12, Chapter 7-8].

```
type row_matrix
```

```
module Row :
```

```
  sig
```

```

exception All_bottom

val create :
  int ->
  Constraint_system.constraint_system list -> Constraint_system.row_matrix

  Row.create_row_matrix s csys_l creates a row matrix of constraint system of size s
  where the element are the constraint systems in csys_l.

  Raises
    • Internal_error if the constraint systems in csys_l do not have the same structure.
      [High debugging]
    • Internal_error if s is different from the number of element in csys_l
    • Internal_error if the elements of csys_l do not have the same maximal support.

val get :
  Constraint_system.row_matrix -> int -> Constraint_system.constraint_system
val get_number_column : Constraint_system.row_matrix -> int
  Row.get_number_column rm returns the number of column of rm.

val get_maximal_support : Constraint_system.row_matrix -> int
  get_maximal_support rm returns the maximal support of the constraint systems in rm.

val iter :
  (Constraint_system.constraint_system -> unit) ->
  Constraint_system.row_matrix -> unit
val map :
  (Constraint_system.constraint_system -> Constraint_system.constraint_system) ->
  Constraint_system.row_matrix -> Constraint_system.row_matrix
val map2 :
  ('a ->
   Constraint_system.constraint_system -> Constraint_system.constraint_system) ->
  'a list -> Constraint_system.row_matrix -> Constraint_system.row_matrix
val fold_right :
  (Constraint_system.constraint_system -> 'a -> 'a) ->
  Constraint_system.row_matrix -> 'a -> 'a
val fold_left :
  ('a -> Constraint_system.constraint_system -> 'a) ->
  'a -> Constraint_system.row_matrix -> 'a
val check_structure : Constraint_system.row_matrix -> unit
  check_structure rm does nothing if rm have is well structured else it raises the exception
  Internal_error. The definition of well structured row matrix is given in [Che12, Section
  7.3.2.1].

end

```

1.4.5 Matrix of constraint systems

```

type matrix
module Matrix :
  sig
    val empty : Constraint_system.matrix
    val matrix_of_row_matrix :
      Constraint_system.row_matrix -> Constraint_system.matrix

```

`matrix_of_row_matrix rm` returns the row matrix `rm` considered as a matrix with one line.

```
val add_row :  
  Constraint_system.matrix ->  
  Constraint_system.row_matrix -> Constraint_system.matrix
```

Access

```
val get_number_column : Constraint_system.matrix -> int  
  get_number_column m returns the number of column of m.
```

```
val get_number_line : Constraint_system.matrix -> int  
  get_number_line m returns the number of line of m.
```

```
val get_maximal_support : Constraint_system.matrix -> int  
  get_maximal_support m returns the maximal support of the constraint systems in m.
```

Iterators

```
val replace_row :  
  (Constraint_system.row_matrix -> Constraint_system.row_matrix list) ->  
  Constraint_system.matrix -> Constraint_system.matrix  
  
  If m is the matrix  $[V_1; \dots; V_n]$  where the  $V_i$  are row matrices, then replace_row m f returns  
  the matrix  $[V_1^1; \dots; V_1^{k_1}; V_2^1; \dots; V_n^{k_n}]$  where for all  $i \in 1, \dots, n$ , the application of f on  $V_i$  is  
  the list of row matrices  $V_i^1, \dots, V_i^{k_i}$ .  
  
  Raises Internal_error if the maximal support of the number of column of the row  
  matrices produced by f do not match.
```

```
val fold_left_on_column :  
  int ->  
  ('a -> Constraint_system.constraint_system -> 'a) ->  
  'a -> Constraint_system.matrix -> 'a  
  
  fold_left_column j f acc m is f (... f (f acc c1) c2 ...) cn where  $[c1; \dots; cn]$   
  is the vector of constraint systems corresponding to the jth column of m.
```

```
val fold_left_on_row :  
  int ->  
  ('a -> Constraint_system.constraint_system -> 'a) ->  
  'a -> Constraint_system.matrix -> 'a  
  
  fold_left_row j f acc m is f (... f (f acc c1) c2 ...) cn where  $[c1; \dots; cn]$  is  
  the vector of constraint systems corresponding to the jth line of m.
```

```
val fold_left_row :  
  ('a -> Constraint_system.row_matrix -> 'a) ->  
  'a -> Constraint_system.matrix -> 'a
```

```
val fold_right_row :  
  (Constraint_system.row_matrix -> 'a -> 'a) ->  
  Constraint_system.matrix -> 'a -> 'a
```

```
val iter :  
  (Constraint_system.constraint_system -> unit) ->  
  Constraint_system.matrix -> unit
```

iter f matrix is f c_{1_1}; f c_{1_2}; ...; f c_{1_m}; f c_{2_1}; ...; f c_{n_m} where matrix is the matrix

$$\begin{bmatrix} c_{1,1} & \cdots & c_{1,m} \\ \vdots & \ddots & \vdots \\ c_{n,1} & \cdots & c_{n,m} \end{bmatrix}$$

```
val iter_row :
  (Constraint_system.row_matrix -> unit) -> Constraint_system.matrix -> unit
val map :
  (Constraint_system.constraint_system -> Constraint_system.constraint_system) ->
  Constraint_system.matrix -> Constraint_system.matrix
val map_on_column :
  int ->
  (Constraint_system.constraint_system -> Constraint_system.constraint_system) ->
  Constraint_system.matrix -> Constraint_system.matrix
```

Matrix searching

```
val find_in_row :
  int ->
  (Constraint_system.constraint_system -> bool) ->
  Constraint_system.matrix -> Constraint_system.constraint_system * int
  find_in_row i f_test matrix searches the first constraint system in the line i of matrix
  that satisfies f_test.
  Raises Not_found if no such constraint system exists.

val find_in_col :
  int ->
  (Constraint_system.constraint_system -> bool) ->
  Constraint_system.matrix -> Constraint_system.constraint_system * int
  find_in_col j f_test matrix searches the first constraint system in the column j of
  matrix that satisfies f_test.
  Raises Not_found if no such constraint system exists.

val find_in_row_between_col_index :
  int ->
  int ->
  int ->
  (Constraint_system.constraint_system -> bool) ->
  Constraint_system.matrix -> Constraint_system.constraint_system * int
  find_in_row_between_col_index i j j' f_test matrix searches the first constraint
  system in line i of matrix that satisfies f_test and whose column index is between j and j'.
  Raises
    • Not_found if no such constraint system exists.
    • Internal_error if the column indexes are not correct. [Low debugging]

val find_in_col_between_row_index :
  int ->
  int ->
  int ->
  (Constraint_system.constraint_system -> bool) ->
  Constraint_system.matrix -> Constraint_system.constraint_system * int
  find_in_col_between_row_index j i i' f_test matrix searches the first constraint
  system in column j of matrix that satisfies f_test and whose line index is between i and i'.
  Raises Not_found if no such constraint system exists.
```

Matrix scanning

```
val exists_in_row :
  int ->
  (Constraint_system.constraint_system -> bool) ->
  Constraint_system.matrix -> bool

  exists_in_row i f_test matrix retrurns true iff there exists a constraint system in the
  line i of matrix that satisfies f_test.

val exists_in_row_between_col_index :
  int ->
  int ->
  int ->
  (Constraint_system.constraint_system -> bool) ->
  Constraint_system.matrix -> bool

  exists_in_row i j j' f_test matrix retrurns true iff there exists a constraint system in
  the line i of matrix that satisfies f_test and whose column index is between j and j'.

val exists_in_col :
  int ->
  (Constraint_system.constraint_system -> bool) ->
  Constraint_system.matrix -> bool

  exists_in_col j f_test matrix retrurns true iff there exists a constraint system in the
  column j of matrix that satisfies f_test.

val exists_in_col_between_row_index :
  int ->
  int ->
  int ->
  (Constraint_system.constraint_system -> bool) ->
  Constraint_system.matrix -> bool

  exists_in_col j i i' f_test matrix retrurns true iff there exists a constraint system in
  the column j of matrix that satisfies f_test and whose line index is between i and i'.

val is_empty : Constraint_system.matrix -> bool

  is_empty m returns true iff and only m is the empty matrix.

val check_structure : Constraint_system.matrix -> unit

  check_structure m does nothing if m have is well structured else it raises the exception
  Internal_error. The definition of well structured matrix is given in [Che12, Section
  7.3.2.1].

val display : Constraint_system.matrix -> string
val normalise : Constraint_system.matrix -> Constraint_system.matrix
end
```

1.4.6 Rule applications

exception Not_applicable

The exception `Not_applicable` is launched when a rule cannot be applied on a row matrix usually due to a condition of the structure of the constraint systems in the row.

The following functions describe the mechanism for applying a rule on matrices of constraint system. Each of these functions have as arguments at least the two following functions:

- `search` : `constraint_system -> 'a * constraint_system * constraint_system`
- `apply` : `'a -> constraint_system -> constraint_system * constraint_system`

Typically, applying a rule on a constraint system depend on parameter that can depend themselves on elements of the frame, deducibility constraints, equations, ... The function `search` searches for the correspondances between the paramaters of the rule and the constraint system, then it applies the rule on the constraint system hence producing two new constraint systems. However, since a rule will always be applied on row matrices that contains constraint systems of same structure, `search` also returns enough informations for the function `apply` to apply the rules on a constraint system without having to search again the correspondance between parameter and the constrain system.

```
val apply_rule_on_row_matrix :
  (constraint_system ->
    'a * constraint_system *
    constraint_system) ->
  ('a ->
    constraint_system ->
    constraint_system * constraint_system) ->
  row_matrix ->
  row_matrix option * row_matrix option
  apply_rule_on_row_matrix search apply r apply the rule on the row matrix r. It returns a
  pair of row matrix option (r_left,r_right) where r_left (resp. r_right) is None if the
  application of the rule produces an unsatisfiable left (resp. right) row matrix, i.e. a row matrice
  with only  $\perp$  as constraint systems. See [Che12, Definition 7.10] for more detail on the application
  of a rule on a row matrix.
```

Raises `Internal_error` if the constraint systems produced by `search` or `apply` do not have the same maximal supports as those in `r`.

```
val apply_external_rule :
  (constraint_system ->
    'a * constraint_system *
    constraint_system) ->
  ('a ->
    constraint_system ->
    constraint_system * constraint_system) ->
  matrix ->
  matrix * matrix
  apply_external_rule search apply m apply an external rule on the matrix m. See [Che12,
  Section 7.3.2.2] for more detail on the application of an external rule on a matrix.
```

Raises `Internal_error` if the constraint systems produced by `search` or `apply` do not have the same maximal supports as those in `m`.

```
val apply_internal_rule :
  (constraint_system ->
    'a * constraint_system *
    constraint_system) ->
  ('a ->
    constraint_system ->
    constraint_system * constraint_system) ->
  int -> matrix -> matrix
  apply_internal_rule search apply i m apply an internal rule on the ith line of matrix m.
  See [Che12, Section 7.3.2.2] for more detail on the application of an internal rule on a matrix.
```

Raises

- `Internal_error` if the constraint systems produced by `search` or `apply` do not have the same maximal supports as those in `m`.

- `Internal_error` if `i` is not the index of a line of `m`.

```
val apply_internal_rule_full_column :
  (constraint_system ->
    'a * constraint_system *
    constraint_system) ->
  ('a ->
    constraint_system ->
    constraint_system * constraint_system) ->
  matrix -> matrix
```

`apply_internal_rule_full_column search apply m` apply an internal rule on each line line of matrix `m` hence returning a matrix with twice the number of line as `m` (when counting the line with only bottom constraint system). It will be used to apply rule `DEST` and `EQ-LEFT-RIGHT`. See [Che12, Section 7.4.1.1] for more detail on the application of these rules.

Raises `Internal_error` if the constraint systems produced by `search` or `apply` do not have the same maximal supports as those in `m`.

1.5 Module Process : Process

```
type label
val fresh_label : unit -> label
type formula =
  | Eq of Term.term * Term.term
  | Neq of Term.term * Term.term
  | And of formula * formula
  | Or of formula * formula
type pattern =
  | Var of Term.variable
  | Tuple of Term.symbol * pattern list
type process =
  | Nil
  | Choice of process * process
  | Par of process * process
  | New of Term.name * process * label
  | In of Term.term * Term.variable * process * label
  | Out of Term.term * Term.term * process * label
  | Let of pattern * Term.term * process * label
  | IfThenElse of formula * process * process * label
val refresh_label : process -> process
val rename : process -> process
val iter_term_process : process -> (Term.term -> Term.term) -> process
val get_free_names : process -> Term.name list
val display_process : process -> string
```

1.5.1 Symbolic process

```
type symbolic_process
val create_symbolic :
  (Recipe.recipe * Term.term) list ->
  process ->
```

```

    Constraint_system.constraint_system -> symbolic_process
val display_trace : symbolic_process -> string
val display_trace_no_unif : symbolic_process -> string

```

Testing

```

val is_bottom : symbolic_process -> bool

```

Access and modification

```

val get_constraint_system :
  symbolic_process -> Constraint_system.constraint_system
val replace_constraint_system :
  Constraint_system.constraint_system ->
  symbolic_process -> symbolic_process
val simplify : symbolic_process -> symbolic_process
val size_trace : symbolic_process -> int
val instanciate_trace : symbolic_process -> symbolic_process

```

Transition application

```

val apply_internal_transition :
  bool ->
  (symbolic_process -> unit) -> symbolic_process -> unit
val apply_input :
  (symbolic_process -> unit) ->
  Recipe.variable -> Recipe.variable -> symbolic_process -> unit
val apply_output :
  (symbolic_process -> unit) ->
  Recipe.variable -> symbolic_process -> unit

```

Optimisation

```

val is_same_input_output : symbolic_process -> symbolic_process -> bool

```

Chapter 2

Trace equivalence

2.1 Module Rules : Definitions of the rules

This module regroupes all the functions that describes the application of rules on constraint systems.

2.1.1 Rule CONS

```
val apply_cons_row_matrix :  
  Standard_library.Recipe.variable ->  
  Standard_library.Term.symbol ->  
  Standard_library.Constraint_system.row_matrix ->  
  Standard_library.Constraint_system.row_matrix option *  
  Standard_library.Constraint_system.row_matrix option  
  
val apply_external_cons_phase_1 :  
  Standard_library.Recipe.variable ->  
  Standard_library.Term.symbol ->  
  Standard_library.Constraint_system.matrix ->  
  Standard_library.Constraint_system.matrix *  
  Standard_library.Constraint_system.matrix  
  
val apply_external_cons_phase_2 :  
  Standard_library.Recipe.variable ->  
  Standard_library.Term.symbol ->  
  Standard_library.Constraint_system.matrix ->  
  Standard_library.Constraint_system.matrix *  
  Standard_library.Constraint_system.matrix
```

2.1.2 Rule AXIOM

```
val apply_axiom_row_matrix :  
  int ->  
  Standard_library.Recipe.variable ->  
  Standard_library.Recipe.path ->  
  Standard_library.Constraint_system.row_matrix ->  
  Standard_library.Constraint_system.row_matrix option *  
  Standard_library.Constraint_system.row_matrix option  
  
val apply_external_axiom_phase_1 :  
  int ->  
  Standard_library.Recipe.variable ->  
  Standard_library.Recipe.path ->  
  Standard_library.Constraint_system.matrix ->
```

```

Standard_library.Constraint_system.matrix *
Standard_library.Constraint_system.matrix
val apply_external_axiom_phase_2 :
  int ->
  Standard_library.Recipe.variable ->
  Standard_library.Recipe.path ->
  Standard_library.Constraint_system.matrix ->
  Standard_library.Constraint_system.matrix *
  Standard_library.Constraint_system.matrix

```

2.1.3 Rule DEST

```

val apply_full_column_dest :
  int ->
  Standard_library.Recipe.path ->
  int ->
  Standard_library.Term.symbol ->
  Standard_library.Constraint_system.matrix ->
  Standard_library.Constraint_system.matrix *
  (Standard_library.Recipe.path * Standard_library.Recipe.variable) list
val apply_full_column_dest_tuple :
  int ->
  Standard_library.Recipe.path ->
  Standard_library.Term.symbol ->
  Standard_library.Constraint_system.matrix ->
  Standard_library.Constraint_system.matrix *
  (Standard_library.Recipe.path * Standard_library.Recipe.variable) list

```

2.1.4 Rule EQ-LEFT-LEFT

```

val apply_eqll :
  int ->
  Standard_library.Constraint_system.matrix ->
  Standard_library.Constraint_system.matrix

```

2.1.5 Rule EQ-LEFT-RIGHT

```

val apply_full_column_eqlr :
  int ->
  Standard_library.Recipe.path ->
  Standard_library.Recipe.variable ->
  Standard_library.Constraint_system.matrix ->
  Standard_library.Constraint_system.matrix
val apply_full_column_eqlr_frame :
  int ->
  Standard_library.Recipe.path ->
  int ->
  Standard_library.Recipe.path ->
  Standard_library.Constraint_system.matrix ->
  Standard_library.Constraint_system.matrix

```

2.1.6 Rule EQ-RIGHT-RIGHT

```

val apply_eqrr_row_matrix :
  Standard_library.Recipe.variable ->

```

```

Standard_library.Recipe.variable ->
Standard_library.Constraint_system.row_matrix ->
Standard_library.Constraint_system.row_matrix option *
Standard_library.Constraint_system.row_matrix option
val apply_external_eqrr_phase_1 :
  Standard_library.Recipe.variable ->
  Standard_library.Recipe.variable ->
  Standard_library.Constraint_system.matrix ->
  Standard_library.Constraint_system.matrix *
  Standard_library.Constraint_system.matrix
val apply_external_eqrr_phase_2 :
  Standard_library.Recipe.variable ->
  Standard_library.Recipe.recipe ->
  Standard_library.Constraint_system.matrix ->
  Standard_library.Constraint_system.matrix *
  Standard_library.Constraint_system.matrix

```

2.1.7 Rule DED-ST

```

val apply_dedsubterm_row_matrix :
  Standard_library.Recipe.path ->
  int ->
  Standard_library.Term.symbol ->
  int ->
  Standard_library.Constraint_system.row_matrix ->
  Standard_library.Constraint_system.row_matrix option *
  Standard_library.Constraint_system.row_matrix option

```

2.2 Module Strategy

```

val apply_strategy_input :
  (Standard_library.Constraint_system.matrix -> unit) ->
  Standard_library.Constraint_system.matrix -> unit
val apply_strategy_output :
  (Standard_library.Constraint_system.matrix -> unit) ->
  Standard_library.Constraint_system.matrix -> unit

```

2.3 Module Algorithm

```

val decide_trace_equivalence :
  Standard_library.Process.process -> Standard_library.Process.process -> bool
val internal_communication : bool Pervasives.ref

```


Bibliography

- [Che12] Vincent Cheval. *Automatic verification of cryptographic protocols: privacy-type properties*. Thèse de doctorat, Laboratoire Spécification et Vérification, ENS Cachan, France, December 2012.