Towards Formalising the Guard Checker of Rocq

TYPES 2025, Glasgow

Yee-Jian Tan, Yannick Forster

June 13, 2025

Institut Polytechnique de Paris, Inria Paris

Eliminators and Fixpoints

- · Rocq is based on the Calculus of *Inductive* Constructions (CIC) [PP89].
- To construct: constructors

To eliminate: eliminators (aka recursors, destructors), or <u>fixpoints + match</u>.

```
Fixpoint add (m \ n : nat) {struct m} := match m with 0 => n \mid S \mid m' => add \mid m' \mid (S \mid n) \mid end.
```

· Advantage: extracted code to e.g. OCaml is more idiomatic

Eliminators and Fixpoints

Unrestricted fixpoints can be non-terminating...

```
#[bypass_check(guard)]
Fixpoint boom (n : nat) : False := boom n.
```

Eliminators and Fixpoints

Unrestricted fixpoints can be non-terminating...

```
#[bypass_check(guard)]
Fixpoint boom (n : nat) : False := boom n.
```

and break consistency!

```
Check (boom 0). (* False *)
```

How does Rocq avoid non-termination?

The guard checker! It checks for **structural recursion**.

```
Fixpoint add (m n : nat)
  {struct m} : nat :=
  match m with
  | 0 => n
  | S m' => add m' (S n)
  end.
```

How does Rocq avoid non-termination?

The guard checker! It checks for structural recursion.

```
Fixpoint add (m n : nat)
  {struct m} : nat :=
  match m with
  | 0 => n
  | S m' => add m' (S n)
  end.
```

How does Rocq avoid non-termination?

The guard checker! It checks for structural recursion.

```
Fixpoint add (m n : nat)
  {struct m} : nat :=
  match m with
  | 0 => n
  | S m' => add m' (S n)
  end.
```

Simple!

Another example:

```
Fixpoint minus (a b : nat) {struct a} :=
  match a, b with
  | 0 , _ => 0
  | a , 0 => a
  | S a', S b' => minus a' b'
  end.
```

```
Fixpoint div (m n : nat) {struct m} :=
  match m with
  | 0 => 0
  | S k => S (div (minus k n) n)
  end.
```

```
Fixpoint minus (a b : nat) {struct a} :=
  match a, b with
  | 0 , _ => 0
  | a , 0 => a
  | S a', S b' => minus a' b'
  end.

div is not guarded! Why?
```

```
Fixpoint div (m n : nat) {struct m} :=
  match m with
  | 0 => 0
  | S k => S (div (minus k n) n)
  end.
```

Because 0 is not a subterm of m!

```
Fixpoint div (m n : nat) {struct m} :=
  match m with
  | 0 => 0
  | S k => S (div (minus k n) n)
  end.
```

```
Fixpoint minus (a b : nat) {struct a} :=

match a, b with
| 0 , _ => a
| a , 0 => a
| S a', S b' => minus a' b'
end.
```

This is structural!
Things are not as simple as they seem.

```
Fixpoint div (m n : nat) {struct m} :=
  match m with
  | 0 => 0
  | S k => S (div (minus k n) n)
  end.
```

The Guard Checker of Rocq

	rel_min = remu.rel_min*n;	eatch kind c' with	let () - assert (List.is_empty large) in	(match mubt with	alt :: stack' →			check_rec_call_state renv needreduce_fix non_absorbed_stack rx (fun	match stack with
(* Check if t is a subterm of Rel n, and gives its specification, assuming lat already gives index of	genv = iterate (fun ge -> lazy Not_motors::ge) n renv.genv }	Prod(na,a,b) → let ienv' = ienv_push_var ienv (na,a,ek_norec) in	let recarg = build_recarge law (List.hd trees) b in let law' = law_push_war law (re_b,ek_norec) in	Subterm (_, _s, wf) → (" We take the subterm specs of the constructor of the record ")	(** YJ: run below until stack is empty. First whn [ar] into [t], *) (let t = whd_all Tevers env ar in	* Check if [def] is a guarded fixpoint body with decreasing arg. given [recpos], the decreasing arguments of each mutually defined	() →	(* we try hard to reduce the fix away by looking for a	elt :: stack -> let rs = check_inert_subters_rec_call renv rs a in
subteres with corresponding specifications of recursive arguments ")	let push_fix_renv renv (_,v as recdef) = let n = Arrav.lensth v in	ienv_decompose_prod Tevars ienv' (n-1) b → sexert false	recargs_constr_rec ienv' (List.tl trees) (recarg::lrec) d	Let wf_args = (dest_mubterms wf).(0) in (* We extract the tree of the projected argument *)	eatch kind t with (** Y2: There are only 2 cases for the type of [o] here: Prod or anything i	fispoint. *)		constructor in [decrâng] (we unfeld definitions too) *) if List.length stack * decrâng then None else s	let renv', stack', body' = pop_argument NoNeedReduce renv el tack x a body in
(* A powerful notion of subterm *)	{ env = push_rec_types recdef renv.env;	(* This recover slobal parameters of the inductive types in 1c (for	List.rev lrec	let n = Projection.arg p in	else.	let of: - Array.length recpos in		match I (at ath attack decedes with	check_mested_fix_body illformed nerv' (decr-1) stack' rs bo
(* To each inductive definition corresponds an array describing the	genv = iterate (fun ge -> lary Not_motorn::ge) n renv.genv }	nested inductive types only) ") let dummy_univ = Level.(make (Uilobal.make (DirPath.make [Id.of_string	recargs_constr_rec ienv trees [] c	Dead_code -> Dead_code	into Skrge. *)	(* Checks if [t] only make valid recursive calls		Skrg _ → (* A metch on the way *) None Schoure (.,_,e,reckrg) →	let renv' = push_var_renv renv (redex_level rx) (x,a) in
structure of recursive arguments for each constructor, we call it the recursive spec of the type (it has type recargs vect). For	type fix_check_result =	"implicit"]) == (1)	in (* starting with ra_env = [] seems safe because any unbounded Rel will be	Not_subters -> Not_subters Internally_bound_subters s -> Internally_bound_subters s)	We only care if type of [p] is a Frod. Otherwise we map the stack into Sirgs. ") Frod (m,a,cB) → ("" Y2: open the binder [ctx : n] into ctx ")	[stack] is the list of constructor's argument specification and arguments that will be applied after reduction.		let c = whd_all Tevars renv.env (lift n reckrg) in let hd, _ = decompose_app_list c in	<pre>check_nested_fix_body illformed renv' (decr-1) [] rs body end</pre>
checking the guard, we start from the decreasing argument (Rel n) with its recursive spec. During checking the guardness condition,	Needleduce of erv * fix guard error Nedleedleduce	let dummy_implicit_mort = mkType (Universe_make dummy_univ) let lambds_implicit n s =	satigned Norec *) build_recarge_mested (env,[]) tree (ind, args)	Const c →	let d = localtenum (n,a) in let cts, a = whd_decompose_prod_decls ?evers env a in	example u in t where we have (match with + t and) u; [rs] is the stack of redeses traversed w/o having been triggered *)		match kind hd with Construct → Some (contract_fix fix, stack)	_ → illformed ()
we collect patterns variables corresponding to subterms of n, each of them with its recursive spec. They are organised in a list lat	(* Definition and manipulation of the stack *)	let amon = Context.make_amont Anonymous Sorts.Salevant in let lambds_implicit a = mklambds (amon, dammy_implicit_mort, a) in	(* [restrict_spec env spec p] restricts the size information in spec to what is	besin try	let env = push_rel_context ctx env in	let rec check_rec_call_stack_renv_stack_rs_t = match_kind_t_with		Cofix Ind Lambda Prod Latin Sort Int Float Array -> assert false	and check_rec_call_state renv meedreduce_of_head stack rs expand_head = (" Text if either the head or the stack of a state
of type (int * recergs) list which is sorted with respect to the	tion stark element a	iterate lambda_implicit n a	allowed to flow through a match with predicate p in environment env. ") (*" TODO YJ: what ix this predicate thing about? check the usage to find out.	with	let by args - decompose_app_list (shd_all levers env a) in let alt = metch kind to with	App (f,args) →		Rel War Coret App Case Fix Prof Cast Meta Evar + None	needs the state to be reduced before continuing checking *)
	(* arguments in the evaluation stack *) (* [constr] is typed in [guard_env] and [int] is the masher of	let abstract_mind_lc ntyps mpars mind lc =		NotfusbableConst (InFrisitive (_u,op)) when list.length 1 >= CFrisitives.arity op ->		begin let re, stack -			match needreduce_of_head needreduce_of_stack stack with Nobleedleduce -> rs
		let lc = Array.map (fun (cts, c) → Term.it_mkProd_or_LetIn c cts) lc in	(** YJ: if spec is [Not_mobters] etc then no restriction to it. Otherwise,	primitive_specif Newara new op 1 NotfivelumbleCoret Not subterm	(** VI: if the binder type [a] is some inductive applied with [args], then it could be a subterm. ")	Array fold right (fun a (re, stack) ->	1.0	inst (ks, u ss cu) -> check per rell state peru lidicalischen stark re (for () de	(* Franci if morthly otherwise last charm promonts and
(* Environment annotated with marks on recursive arguments *)	Silosure of fix_chek_result "guard_env " int " constr (" arguments applied to a "match": only their spec traverse the match ") Sing of multers_spec Laxy.t	<pre>let Lc = Array.map (fun (ctx, c) → Term.it_mkProd_or_LetIn c ctx) lc in let rec replace_ind k c = let hd, argx = decompose_app_list c in match kind be with</pre>	of [p] is an inductive, then return the intersection of spec with spec of	and	(** 12: If the spec of [elt] is subterm, intersect and return subterm, otherwise return spec. *)	<pre>let stack = push_stack_closure renv needreduce a stack in (rs.stack)) args (rs.stack)</pre>		if evaluable_constant kn renv.env then Some (constant_value_in	(* Expand if possible, otherwise, last chance, propagate need for expansion, in the hope to be eventually erased *) match expand_head () with
(* tells whether it is a strict or loose subterm *) toom size * Large Strict	let (III) x v = mtch x with	match kind bd with Ind ((mind',i),_) when MutInd.CamDrd.equal mind mind' -> let rec drop manage n = function	(** NOTE II: this is to fix propest after beta-iots cuts. Lennard's code has more info about the cuts and some comments about the looks, but it is	Ver _ Sort _ Cost _ Prod _ Letin _ App _ Ind _ Construct Cofix Int Float Array -> Not subterm	Ind ind -> let mec - stack element mecif Towars elt in	in check rec call stack reny stack rs f		else None)	None -> e :: List.tl rs Some (c. stack') -> check rec call stack rerw (stack'@stack) rs c
		let rec drep_merams = *function :: args when n > 0 -> drop_params (n-1) args args -> lambda_implicit n (Nerm.mpplint (mkNml (ntype=n-k-i),	more info about the cuts and some comments about the logic, but it is basically translating code to english. The top of his document says he does not understand why the restriction either. Maxims Denne who implemented the	Construct Cofix Let Float Army → Not_subters	let many - lary (match Lazy.force spec with	check_rec_call_stack renv stack rs f	1.0	ambda (x,a,b) →	
(* merging information *) Let mine glb m1 m2 =	NewStatuce → x NoticedInduce → y	args → lambds_implicit n (form.applist (mkHml (ntyps:n:k-i), List_Smart.map (replace_ind (n:k)) args))	not understand why the restriction either. Maxims Denex who implemented the first version of this could perhaps give an explanation.")	(* Other terms are not subterms *)	Not_subters Dead_code Internally_bound_subters _ as spec ->	Rel p →		begin let meedreduce, rs = check rec call reny rs a in	and check_inert_subters_rec_call renv rs c = (* Oweck rec calls of a term which does not interact with its
match s1,s2 with	let rec needreduce_of_stack = function	in	let restrict year Zouers any year n s	and have a three and the same and the	Subtere(1, s,path) -> ("" U: Question: in the final intersection, the contribution	begin		match stack with	
Strict, Strict → Strict _ → targe	[] → NoNeedReduce Skrg _:: 1 → needreduce_of_wtack 1 Stinuses (needreduce_of_wtack 1	drop_marams opers args _ → map_with_binders succ replace_ind k c	match spec with Not_subterm Internally_bound_subterm _ → mpec	lary (subters_specif Yevers renv stack t)	of [p] boils down to [aros] only. *)	if remv.rel_min <- p && p < remv.rel_min*nfi then	stack x a b	elt :: stack -> let renv, stack, b = pop_argument Tevars meedreduce renv elt	level of the redex stack *) lat reed_reduce, rs = check_rec_call renv rs c in
(* possible specifications for a term:		in Array.map (replace_ind 0) lc	let abects, ar - whd_decompose_lambda_decls ?evers env p in	and stack_element_specif Tevers - function	(** X): Find oud how get_recarge_approx uses args (which does not really make sense). *)	(" the position of the invoked fispoint: ") let glob = renv.rel_min=nfi-1-p in	stack x a b	check_rec_call_stack renv stack rs b	CHEACHE CHARLES CONTROL (CONTROL CONTROL CONTR
- Not subterm: when the miss of a term is not related to the recursive argument of the fixpoint	let redex_level rx = List_length rx	let is primitive positive container env c =	(* Optimization: if the predicate is not dependent, no restriction is needed and we spould buildion the records from *)	Silonure (_, h_renv, _, h) -> lazy_subters_specif Nevers h_renv [] h Sing x -> x	let recargs = get_recargs_approx Tevars env path ind args in let path = inter_wf_paths path recargs in	(* the decreasing arg of the rec call: *) let up = respon.(glob) in		[] → check_rec_call_stack (push_war_renv renv (redex_level rx)	and check_rec_call new rs c = (* a) ther fulls if a non-neared call arrows or talls if there is
 Internally_board_subters: when the recursive call is in a subters of a redex and the recursive arousent is bound to a variable 	let push_stack_closure renv medreduce c stack = (%losure (mesdreduce, renv. 0, c)) :: stack	match env. retroknowledge. Retroknowledge. retro_array with	if noccur_with_mets 1 (Context.Rel.length abactx) ar then spec	and extract stack Tevers - function	Subters(1,x,path))	<pre>if List.length stack <- op then set need reduce too renv.env (NotEnoughFroumentsforFixCall)</pre>	(x,a)) [] :	x b	(* either fails if a non guarded call occurs or tells if there is rec call on a variable bound at the top of [c] and update the need for reduction in the redex stack with rec calls on
which will be instantiated by reductor the reduct the inteners		Some c' when (Constant.equal env c c' → true → false	let any a reach real constant absents any to	→ Lazy, from wal Not subterm.	in Skry sarg	set_need_reduce_top renv.env (NotEnoughArgumentsForFixCall plob) rs		end	variables bound at higher levels of the redex stack ")
refer to the number of redeces stacked, with 1 counting for the variables bound at head in the body of the fix (as e.g. [x] in	let push_stack_closures renv 1 stack = List_fold_right (push_stack_closure renv NoNeedNeduce) 1 stack	(* [get_recargs_approx env tree ind args] builds an approximation of the	let arcts, x = whd_decompose_prod_decis Yevers env ar in let env = push_rel_context arcts env in	elt :: 1 -> stack_element_specif Tevers elt, 1	> SArg (set_iots_specif or (lary Not_subters)) in	else (" Retrieve the expected tree for the argument ")		Yed $(x, a, u) \rightarrow$ (* Note: we cannot ensure that the stack is empty because	List.sep_first (check_rec_call_stack renv [] (NoNeedReduce::rs) c)
	let out stack area 1 stack -	recargs tree for ind, knowing args. The argument tree is used to know when candidate	let i,args = decompose_app_list (whd_all Tevers env s) in eath kind i with	and printitive_specif Tevers renv op args = let open CPrintitives in	elt :: filter_stack (push_rel d env) of stack* (** Y): if the stack alament is not an induction, it must not be a	(* Check the decreasing arg is smaller *) let x = List.ath stack up in		non-accessible branches of 'match' expressions can have	in let need reduce, rx = check rec call reny [] def in
occase (metch) suctains may new concine several results; Subtains: when the term is a subtain of the recursive argument the wf_maths argument specifies which subtains are recursive;	List.fold_right (fun spec stack -> SArg spec :: stack) 1 stack	tree for inc, knowling args. He argument tree is used to know when cancidate mented types should be traversed, pruning the tree otherwise. This code is very close to check_positive in indtypes.al, but does no positivity check and does	Ind i →	match on with	subters. *)	match check_is_subterm (stack_element_specif Tevers z) tree		let rs - check_inert_subters_rec_call renv rs a in	int need_reduce, rs = Creck_red_call renv [] der in ammert (List.is_empty rs); match need_reduce with
the wf_maths argument specifies which subterms are recursive; the [int list] is used in the [match] case where one branch of the [match] might be a subterm but (an arbitrary number of)	let lift_stack k =	not.	Dead code → spec	Arrayget Arraydefault -> (* t.[i] and default t can be seen as strict subterms of t, with a	Not subtere() :: 1) stack []	glob with NeedReduceSubters 1 → set_meed_reduce renv.env 1			
others are calls to bound variables	List.map (function Silosure (needreduce.s.n.c) → Silosure (needreduce.s.n.k.c)	compute the number of recursive arguments. *) Let get receips approx 3 years env tree ind args -	Subtarm(1,st,tree) -> let recards - set recards approx Severs env tree i ards in	potentially mested rectree. *) let are = List.oth ares 1 in (* the result is a strict subters of the	filter stack env ar stack	illegal_rec_call renv glob z) rs Invalidiablers → raise (fixius-direc (renv.env.	DE N	$\label{lem:check_rec_call_stack} \mbox{(push_var_renv renv (redex_level rs) } (x,a)) \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ $	
- Dead_code: when the term has been built by elimination over an	x + x)	let get_recarge_approx Pavara env tree ind args = let rec build_recarge (env, ra_env as ienv) tree c =	let recarge = get_recarge_approx Swars env tree 1 args in let recarge = inter_wf_maths tree recarge in Sabtere(1,st_recarge)	second argument *) let subt = subterm_specif Tevars renv [] arg in	let find uniform parameters recinds marge bodies -	illegal_rec_call renv glob z))			ot inductive_of_mutfix ?evers env ((nvect,bodynum),(names,types,bodies as
empty type *)	let lift1_stack = lift_stack 1	let x,largx = decompose_app_list (whd_all Tewarx env c) in match kind x with	Subters(1,st,recergs)		let abodies - Array length bodies in	check_rec_call_state renv NoNeedReduce stack rs (fun () →		let re a Array fold left (rherk tourt mitters rec rall rang) re	ocdef)) = let obfix = Array.length bodies in
type subters_spec =		Prod (ma,b,d) -> masert (List.is_empty large);	end	Subterm (., .m. wf) -> let wf_args - (dest_subterms wf).(0) in	let min_indx = Array.fold_left min range recindx in (" We work only on the fight holy but are in the context of a bodies ")	match lookup_rel p renv.env with	typarray in	let nemy' - push_fix_nemy recodef in	if Int.equal nbfix 0 not (Int.equal (Array.length nvect) nbfix)
Subtern of (Int.Set.t " mize " of paths)	(* (6 Computing the recursive subterms of a term (propagation of size information through Cases).) *)	build_recarge (ienv_push_var ienv (ns, b, sk_norec)) tree d	(* [subters_specif renv t] computes the recursive structure of [t] and	spec_of_tree (List.nth wf_args 8) (* first and only parameter of 'array	let rec mux i k numiformparams c = let f, l = decompase app_list c in	LocalAssum _ → None LocalDef (_,c,_) → Some (lift p c, []))		Array.fold_left (fun rs body -> let meedreduce', rs = check_rec_call rens' rs body in	not (Int.equal (Array.length types) nbfix) not (Int.equal (Array.length names) nbfix)
Not, publish Not, publish Internal to brand without of fat Sat +		(* free variables are allowed and assigned Norec *)	compare its size with the size of the initial recursive argument of	Dead_code -> Dead_code	match kind f with			chard our call state new needselves' stark or (fin us line))	II bedama (8
	<pre>let lookup_minteres env ind = let (_,mip) = lookup_mind_specif env ind in</pre>	(try and (List.nth ra_env (k-1)) with failure Invalid_argument → ek_morec)	the fispoint we are checking. [renv] collects such information about variables.	Not_subters → Not_subters Internally_bound_subters n → Internally_bound_subters n	(* A recursive reference to the i-th body *) c	Case (ci, u, pes, ret, iv, c_0, br) -> (" iv ignored: it's just a cache ")		rs bodies	bodynum >= mbfix then anomaly (Pp.str "Ill-formed fix term.");
let eq_wf_paths = %tree.equal Declareops.eq_recarg	mip.mind_recargs	Ind sad kn -> (* When the inferred tree allows it, we consider that we have a		end - Not subterm	if Int.equal n (nbodies * k = i) then List.fold left i (fun i nuniformarams s →	let (ci, (p,_), _iv, c_0, brs) - expand_case renv.env (ci, u, pms ret. iv, c 0, br) in			
let inter_recorp r1 r2 = if eq_recorp r1 r2 then Some r1 else None	let match_inductive ind re -	potential	let rec subtera_specif Yevers new stack t =	let set_iota_specif or spec =	match kind a with Sal m when Int.equal m (k - j) →	let needreduce_c.0, rs = check_rec_call renv rs c.0 in let rs = check_inert_subterm_rec_call renv rs p in		rej (p, _, c) →	let vdef; = judgment_ef_fispoint recdef in let raim_er env i err = error_ill_formed_rec_body env (Type_errors.fisBuseffror err) names i
let inter_wf_paths = Rtree.inter Declareops.eq_recarg inter_recarg Norec	Mrsc (RecArgled i) → Ind.CanDrd.equal ind i	begin match dest_recarg tree with	let f,l = decompose_app_list (whd_all Tevars renv.env t) in	lary (match Lary, force spec with	(" a reference to the j-th parameter ")				
let incl of maths - Rtree.incl Declareous.em recarm inter recarm Norec	Norec Mrec (RecArgPrim _) -> false	New (RecArgind ind') when (find.equal env (fst ind_kn) ind' → build_recarge_mested ienv tree (ind_kn, large) Norec New _ → sk_morec		Not_subters -> if or >= 1 then Internally_bound_subters (Let.Set_simpleton or) also Not subters	nuniforeparaes	let rs' = NoNeedReduce::rs in let nr = redex level rs' in		<pre>iet meedreduce', rs = check_rec_call renv rs c in check_rec_call_state renv meedreduce' stack rs (fun () →</pre>	(* Check the i-th definition with recarg k *) let find ind i k def *
let spec_of_tree t =	(* In {match c as z in ci y_x return P with C_i x_x ⇒ t end) [branches_specif renv c_spec ci] returns an array of x_x specs knowing	Norse Nee -> sk_norse	Case (ci, u, pms, p, iv, c, lbr) → (* iv ignored: it's just a cache *) let (ci, (p,), iv, c, lbr) = expand_case renv.env (ci, u, pms, p, iv,	spec → spec)	(* not a parameter: this puts a bound on the size of an estrucible prefix of uniform arguments *)	<pre>let case_spec = branches_specif renv (set_iots_specif nr (lary_subters_specif ?</pre>		(" we try hard to reduce the proj sway by looking for a constructor in c (we unfold definitions too) ")	(* check fi does not appear in the k*1 first abstractions, gives the type of the k*1-eme abstraction (must be an inductive) *)
if eq.wf.paths t sk.norec	c 1010. *)	Const (c,_) when is_primitive_positive_container env c ->		(**************************************		wars reny [] c_0)) ci in		let c a shd all favore rank any c in	let per check prove any noted a
then Not_subterm else Subterm (Int.Set.empty, Strict, t)	let branches specif renv c_spec ci = let car = ("" YJ: an array of ints, arities of constructors of C ")	begin match dest_recarg tree with Nrec (RecArgPrim c') when (Constant.equal env c c' → build_recargz_nested_primitive ienv tree (c, large)	c, usr) in let stack' = push_stack_closures renv 1 stack in let cases_spac =	exception fixfuardirror of env * fix_guard_error	else muniformparams	<pre>nvars renv [] c_0)) ct in let stack' = filter_stack_domain ?evers renv.env nr p stack in let rs' =</pre>		let hd, args = decomposm_app c in let hd, args = match kind hd with Cefix cefix →	match kind (whd_all Tevers env def) with Lambda (x,a,b) →
let merce internal subterms 11 12 -	(* We fetch the regular tree associated to the inductive of the match. This is just to set the number of constructors (and constructor	build_recargs_mested_primitive iero tree (c, largs)	branches_specif renv (lary_subterm_specif Tevers renv [] c) ci in	let illegal_rec_call remy fx = function	> fold_constr_with_binders succ (aux i) k muniformparams c in	Array.fold_left_i (fun k rs' br' -> let stack_br = push_stack_args case_spec.(k) stack' in			if recour_with_mets n mbfix a them let env' = push_rel (localArmum (x,a)) env in
Int.Set.union 11 12	arities) that fit the match branches without forcing c_spec. Note that c_spec might be more precise than [v] below, because of	erd	let stl = Array.mapi (fun i br' →	There are and it	Array.fold_left_i (fun i → mux i 8) min_indx bodies	check_rec_call_stack_renv_stack_br_rs' br') rs' brs in let needreduce_br, rs = List_mep_first_rs' in	cofix, args	()))	if Int.equal n (k = 1) then
let inter_spec s1 s2 =		sk_norsc	let stack_br = push_stack_args (cases_spec.(i)) stack' in	let le_lt_wars = lary (let (_,le_wars,lt_wars) =	(** Given a fispoint [fix f x y z n := phi(f x y u t,, f x y u' t')]	check_rec_call_state renv (needreduce_br needreduce_c_8) stac		match kind hd with	(* get the inductive type of the fixpoint *) let (mind, _) =
match s1, s2 with	(* let (_,mip) = lookup_mind_specif renv.env ci.ci_ind in *) (* let v = dest_mbterms mip.mind_recargs in *)	and build_recorps_mested (envrs_env as ienv) tree (((mind,i),u), largs) -	subterm_specif Tevers renv stack_br br') lbr in	<pre>tist.fold.left (fun (i,le,lt) sbt →</pre>	structural recursive on [n], restart [v-1, u-8(v), v-0(v v)] a term	s (fun () -> (* we try hard to reduce the match away by looking for a	p), [])	Construct> Some (args.(Projection.mpars p - Projection.arg	try find inductive Tevers env s with Not found →
Dead_code, -> x2	<pre>let v = dest_subterms (lookup_subterms renv.env ci.ci_ind) in Array.map List.length v in</pre>	(* If the inferred tree already disallows recursion, no need to go further *)	<pre>let spec = mubters_spec_glb stl in restrict_spec ?evers renv.env spec p</pre>	match Lazy, force sht with (Subtered ,Strict,) Deed code) -> (i+1, le, i::lt)	[fix f z n := $phi(f u t,, f u' t')]$, may $[pxi]$, of some type $[forall (z:C(x,y)) (n:I(x,y,z)), I(x,y,z,n)]$, so that	constructor in c_8 (we unfold definitions too) ") let c_8 = whd_all ?evers renv.env c_8 in		Cofix Ind Lambda Prod Latin Sort Int Float Array -> assert false	raise err env i (becursionNothhinductiveType a) in let mib, - lookup.mind_specif env (out_punivs mind) in
_, Not_mathers → m2 Internally bound subters 11, Internally bound subters 12 →	(** YJ: for each constructor, generates a list of size-[nca]. *) (** YJ: if discriminant most [c most] is the same inductive as *)	if eq.wf.paths tree sk_morec them tree	fix ((recindex.i).(.tempray.bodies as recdef)) →	(Subters(,Strict,) Desd_code) → (i*1, le, i::lt) (Subters(,large,)) → (i*1, i::le, lt) → (i*1, le.lt)	[finall $(z:\mathbb{C}(x,y))$ $(n:\mathbb{I}(x,y,z))$, $\mathbb{I}(x,y,z,n)$], so that [fin $x \ y \ z \Rightarrow psi \ z$] is of same type as the original term *)	let hd, args = decompose_app_list c_8 in let hd. args = match kind hd with		Rel War Coret App Case Fix Proi Cast Note Ever -> None)	if mib.mind finite + finite then raise err env i (RecursionNotEnlocativeType s):
Internally_bound_mathems	Array mani	let mib - Environ.lookup_mind mind env in	("when proving that the fispoint f(x)= is less than n, it is enough to prove that a is less than n assuming f is less than n	(1,[],[]) renv.genv in (le_vers,lt_vers)) in	let drop_uniform_parameters muniformparams bodies -	Delity polity via		end	raise_err env i (securionsommentettweighe a); (eind, (env', b)) else check_occur env' (n-1) b
	(fun i mcm → (* i*1-th catructor has arity nos *) let lyra : mosc tary, t = lary	let mumper = mib.mind_rparams_rec in let nonrecpar = mib.mind_nparams = mumper in	to prove that e is less than n assuming f is less than n furthermore when f is applied to a term which is strictly less than	(le_vers,lt_vers)) in RecursionDelllegelTere(fx,(erg_renv.env, arg),le_lt_vers)	let mbodies = Array.length bodies in let rec mux i k c = (decompose_app_list (whd_all Newers renv.env (Nerm.applist (contract_cofix cofix, args)))		br id →	else check_occur env' (n=1) b else anomaly -label: "check_one_fix" (Pp.str "Bad occurrence of
Internally_bound_mattern 11, Sabtern (12,a2,t2) -> Subtern (merom_internal_matterns_11 12.a2,t2)	let lors: spec Lary.t = lary (match lary.force c_spec with Subter (t) when match inductive ci.ci.ind (dest_recarg t)	let (lear.) = List.chop suppor large in	furthermore when f is applied to a term which is strictly less than n, one may assume that x itself is strictly less than n	Skrg> (* Twically the case of a recursive call encouplated under a	let f, 1 = decompose_app_list c in match kind f with	_ → hd, args in match kind hd with		check_rec_call_state renw NoNeedReduce stack rs (fun () -> r let_com! Contest_Named.Declaration in	cursive call.") -> raise err env i NotfrouchAbstractionInfisBody
Subtern (11,a1,t1), Subtern (12,a2,t2) -> Subtern (merge_internal_nubterns 11 12, mine_plb s1 a2, inter_sd_paths t1	let vra - Array of list (dest subteres t).(1) in	let awartyp = mib.mind.ptypes in (* Extends the environment with a variable corresponding to	if not (check_inductive_codomain Tevers renv.env typerray.(i)) then	rewriting before been applied to the parameter of a constructor *) NotinoupArgumentsforfisCall fx	Rel n →	Construct cstr → Some (apply_branch cstr args ci brs, [])		match lookup_named id renv.env with	in .
Subterm (merge_internal_mubterms 11 12, mixe_glb al a2, inter_wf_paths t1 42)	accert (Int.equal ncs (Array, length yrs));	the inductive def ") let (env',_ as ienv') = ienv_push_inductive Tevars ienv ((eind,u),lpsr) in	Not_subterm else		(" A recursive reference to the i-th body ") if Int.equal n (mbodies * k - i) then	Cofix Ind Lambda Prod Latin Sort Int Float Array + assert false		LocalAssum _ → None LocalDef (_,c_,) → Sone (c, []))	<pre>let ((ind, u), _) as res = check_occur fixenv 1 def in let _, mip = lookup_mind_specif env ind in</pre>
let withers mar nih s	Array.map spec_of_tree vrs	(* Parameters expressed in env' *) let lpar' = List.map (lift months) lpar in	let (ctxt,clfix) = whd_decompose_prod Tevers renv.env typerray.(i) in	let set_med_reduce_one enw or err rs =	let new_args = List.skipn_at_best numiforeparams 1 in Term.applist (f, new_args)	Rel Yer Const App Case Fix Proj Cast Nets Eusr + Nore		etto (e.e. t.b) us	(* recursive sprop means non record with projections -> squashed *)
<pre>let mbterm.spec_glb = Array.fold_left inter_spec Desd_code</pre>	Dead_code - Array.make ncm Dead_code Dead_code - Array.make ncm Dead_code Internally_bound_subters _ ms x -> Array.make ncm x Subters _ Not_mubters -> Array.make ncm Not_mubters) in	(* In case of mutual inductive types, we use the recarge tree which was computed statically. This is fine because nested inductive types with	let oind = let env' = push_rel_context ctxt renv.env in try Some(fat (find_inductive Texars env' clfix))	let or = List.length rs in let rs1, rs2 = List.chop (or-or) rs in let rs2 = List.chop (ir-or) rs in	else	(* Enables to traverse fiscoint definitions in a more intelligent		atln (s,c,t,b) -> lat medration_c, rs = check_rsc_call renv rs c in lat medration_t, rs = check_rsc_call renv rs t in	if Environ.is_type_in_type env (GlobRef.IndRef ind) then () else match relevance_of_ind_body mip u with
type guard_env =	List.init nos (fun j > lary (lary.force lvrs).(j)))	computes statically. This is fine occase meter inductive types with mutually recursive containers are not supported. ") let freez a		rs1 @ NeedReduce (env, err) :: rs2	- map_with_binders mucc (mux i) k c	sex, is, the rule :		begin match needreduce of stack stack needreduce c needreduce t	Serts, Irrelevent Serts, RelevanceVar as rind ->
{ env : env; (" dB of last fixpoint ")	CBF	if Int.equal accetyp 1 them [dest_subterms tree]	<pre>(match oind with None → Not_matterm (* happens if fix is polymorphic *)</pre>	let set_meed_reduce env 1 err rs =	in Array.mapi (fun i → mux i 8) bodies	if - g = fix g (y1:11)(yp:1p) (struct yp) := a & - f is guarded with respect to the set of pattern variables S	with		<pre>if not (Sorts.relevance_equal names.(i).Contest.binder_relevance ind)</pre>
rel_min : int; (" dS of variables denoting subterms ")	(** T2: checks if the codomain of predicate [p] is an inductive by wh- unwramping. *)	else Array.map (fun mip → dest_mabterms mip.mind_recargs) mib.mind packets	Some (ind, _) → let obfix = Array.length becarray in	Int.Set.fold (fun n → set_meed_reduce_one env n err) 1 rs	let filter fix stack domain or decrare stack muniformearans -	in all and B		NoticeCheduce -> (* Stack do not require to beta-reduce: let's look if the	then raise err env i fixpointfollreleventInductive Sorts Relevant > ()
genv : subterm_spec Lazy.t list;	Let check_inductive_codomain Tevars env p = let shock_inductive_codomain Tevars env p = let shocks, ar - whd_decompose_lambds_decls Tevars env p in let env = push_rel_context abouts env in	in let sk_irecarge j sip =	let recarge - lookup_subteres renv.env ind in	let set_meed_reduce_top env err rs = set_meed_reduce_com env (List.length rs) err rs	let rec sux i sund foreparent stack = match stack with	in T1 Tp & - ap is a sub-term of the formal argument of f & - f is guarded with respect to the set of pattern variables	body of the	let needs *) let appc = lary_subterm_specif Tevers renv [] c in	in .
	let env = push_rel_context about env in	(* The nested inductive type with parameters removed *)	let new' = push_fix_renv renv recdef in		10 + 0	- my is a sub-term or the rormal argument of f & - f is guarded with respect to the set of pattern variables		let stack - lift1 stack stack in	in
let make_renv env recarg tree = { env = env;	let arcts, s = whd_decompose_prod_decis Tevars env ar in let anv = much rel context arcts any in	<pre>let auxlovect = abstract_mind_lc auxntyp auxnpar mind mip.mind_nf_lc in let paths = Array.mapi</pre>	let rem' - (* Why Strict here ? To be general, it could also be	type check_mabters_result = InvalidSubters	a :: stack -> let uniform, nuniformparams - if nuniformparams - 8 then false, 8 else	S*(yp) in a then f is guarded with respect to S in (g a1 am). Eduardo 7/9/98 *)		check_rec_call_stack (push_let renv (s,c,t,spec)) stack rs b Needleduce> check_rec_call_stack renv stack rs (subst1 c b)	let ry - Array.map2 i find ind nyect bodies in
rel_min = recarg=2; (* recarg = 0 -> Rel 1 -> recarg; Rel 2 -> fix *) genv = [Lazy.from_val(Subtere(Int.Set.espty, Large,tree))])	let i,_1' = decompose_app (whd_all Towars env s) in	(fun k c -> let c' = hnf_prod_applist ?evarx env' c lpar' in	Large") assign_var_apec new'	NeedleduceSubterm of Int.Set.t (* empty - NoNeedleduce *)	true, nuniformparame -1 in let a =	fduardo 7/9/9E *) fix ((recindox,i),(_,typarray,bodies as recdef) as fix) →		end	(Array.map fut rv, Array.map and rv)
New - [restqs_variousters[tur=ser.empry, range,tree])]]	ARANA A			(* Check term c can be applied to one of the mutual fixpoints. *)	if uniform Int.equal i decrarg them a		10	$\operatorname{int} (c_{s-}, t) \rightarrow$	
			let decrArg * recindox.(i) in	let check_is_subters x tree =	else (* descrivate the status of non-uniform parameters since we	let rhodies = Array.length bodies in let rs' = Array.fold_left (check_inert_subterm_rec_call renv) NoMeedReduce::rs) typarray in		bet rs = check_inert_mabters_rec_call renv rs t in let rs = check_rec_call_stack_renv stack_rs c in	ot check fix Teware env ((nvect,_),(names_,,bodies as recdef) as fix) = let (minds, rdef) = inductive_of_matfix Tevars env fix in let flags = Environ.tspling.flags env in
let push_ver renv (x,ty,spec) = { env = push_rel (tocalAssum (x,ty)) renv.env;	(** 12: I guess [serv] means inductive environment, and this is code to exild wf_paths. *)	<pre>let (iem',c') = iem_decompose_prod ?evers iem' nonrecper c' in build_recerps_constructors iem' trees.(j).(k) c')</pre>	let theBody = bodies.(i) in						
lat push,ver renv (x,ty,xpec) = { env = push,rel ((conlineau (x,ty)) renv.env; rel,min = renv.rel,min*; oney = none; renv.com; }	(* The following functions are almost duplicated from indtypes.el, except	let (iem',c') = iem_decemposs_prod levers iem' ronrecpar c' in build_recarge_constructors iem' trees.(j).(k) c') assicvect in	let derArg = reciter(1) in let theBody = bodies(1) in let nb878bat = derArg=1 in let nb878bat = derArg=1 in	let chack is mattern a tree = match lany, force x with Subtern (need_reduce,Strict,tree') -> if incl of onths tree tree' then NeedleduceSubtern need reduce	cannot guarantee that they are preserve in the recursive calls *)	MudleedReduce::rx) typarray in let reny' = push fix reny reny recdef in		rs .	if flacs, check quarded then
genv - spec:: renv.genv)	(* The following functions are almost duplicated from indtypes.ml, except that they carry here a poorer environment (containing less information). *) let iero und vor (em. 120 (s.a.m.) =	auxlevect in	nb0fAbst theBody in	else Invalidablers	calls *)	let nuniforsparams - find_unifors_parameters recindss (List.length	11	ort Int Float	let out tree (kn.i) =
<pre>perv = spect: renv.genv ; but push_let renv (s,c,ty,spec) = { env = push_rel ((ocaller (s,c,ty)) renv.env;</pre>	(" The following functions are almost duplicated from indippes.el, escept that they carry have a power environment (containing less information). ") let immy_punh_yer (env, lrs) (.a,p.m) " (punh_red (iocalhesse (x,a)) env, (torec,rs)::lrs)	mulcovet in ak_paths (Nrac (NacArgled (mind,)))) paths in bet irecarge = Array.mapi ak_irecarge mih.mind_packets in	let sign, strippedody " wnd_oscompose_ismode_n_ssiss revers renv.env nbDfAbst theBody in (" pushing the fix parameters ") let stack' = push_stack_closures renv l stack in	In incl. or party the tree than membershabban meet results also invalidabban Dead_code -> Needleduc&stature Int.Set.empty Not_makers Subters (_,large_,) -> Invalidabbans	calls *)	let ren: " pun_til_ren/ ren/ recer in let renidreparams = find_uniform_parameters recindss (List.lengt) stack) bodies in let bodies = drop_uniform_parameters nuniformparams bodies in	- 19		
gen = spec: renv.gen ; let outh let renv (s.c.ts.spec) =	("The following functions are almost deplicated from independal, except that they carry here a pourse environment (containing less information). "I let imm.push.you (sew, len) (x,a,rn) " (push,rn) (localizame (x,a)) war, (lorency,rn)::lray) let imm.push.induction howeve (sew, rn,a,wu) ((cind,vu),lpar) " let imm.push.induction induced (sew rise)."	mandrowct in sk_pathm (force (backeplad (mind,)))) pathm in lit invaring * Array.mgi sk_iracarga sib.mind_packets in (three Ar.vec invarys).(1)	redificant theodoy in (* pushing the fits parameters *) let task* * push, tasks, closures reev a tasks in let reev' * push, tasks, closures reev a tasks in let reev' * push, task, closures reev a tasks in let reev' * push, task, reev rees' sign in	it int. of parts tree tree the manuscripture need recipe also Invalidates Dand code > NeedBeacaCubters Int.Set.emply Not., makers Subters (.imags.) > Encellabless Internally, bound, matters 1 > NeedBeacaCubters 1	calls *) Sky (un_inter_apacif or (lawy Not_mobsers)) in s: naw (i+1) numiformperses stack s naw 0 maniformperses stack in aw 0 maniformperses stack int com arranged Newer membrades new alt stack a b *	let reme " paum, til_reme reme recome in let minformparmes ' find_miform_parmesters recindus (List.lengti tabek) bodies in let bodies = drop_miform_parmesters minformparmes bodies in let fig_tback = filter_fig_tback_domain (redex_lenw1 re) decrArg track_minformparmes in.		<pre>iort _ Int _ Float _ → ('See [Prod]: we cannot ensure that the stack is empty *) rese (w.t.def.tv) →</pre>	if TimpLones, guarante trum Ant pet true (Dr. 1) = let min * Environ.lookup mind he env in mih.mind_macket.(i).mind_recorgs in bet trees * Array.max (fam (mind.) >> out tree mind) minds in bet trees * Array.max (fam (mind.) >> out tree mind) minds in
gate = specif rems.gate ; fat mah_laft rems (s.c., th. yapes) = { am = pash yad (consider (s.c., thy)) rems.ams; rad_laft = rems.rad_laft*; gate = specif = mought*;	(* The following functions was least deplicated from intigencial, except that they carry here a passer environment (montaining lass information). *) let inter_much you (now, las) (x,a,m) = (punk_mel (incalisates (x,a)) now, (inter_n, n)::in) [punk_mel (incalisates (x,a)) now, (inter_n, n)::in) Let all inter_much intitute Norms (now, m, now) ((inter_n), lyan) = let all + four-muchology, abel side into vio. Let there a side into theme:	mulcovet in sk gaths (Frec (Sacingled (mind, j)))) paths in let franceps "Forey, major sk, frecurps sith mind packets in (blue and, rec incorps).(1) and build organization printing (mover, a men) trap (r. laves) a	" and the transfer of the tran	if the L. of gold to track the monotone control near years and Invalidation of Manufacture Int. Set. maps (Manufacture Int. Set. maps (Manufacture Int. Set. maps (Manufacture Int. Set. maps (Manufacture Internally Internally Internally Internally Internally Internally Internally Internally Internal Manufacture Internally Internal Manufacture Internally Internal Manufacture Internal Ma	calls ") Sing (sk.jok.yapeif or (lary Not.mothers)) in s :: naw (k:1) norditroperson thack s in an W norditroperson thack its long.reporate You'r maddedice new all stack x a b - s the norminal control of the con	ibit reas' pagants/servenorecers ibit nosformpares - fraginiform_paraster reciniss (List.length tack) bette nosformpares - fraginiform_paraster reciniss (List.length tack) bette - drop_uniform_parasters nuniformpares bodies in list fragitude - filter_fis_tack_domain (mode_lown) = dering tack nuniformpares - filter_fis_tack_domain (mode_lown) = dering tack nuniformpares - filter_fis_tack_domain (mode_lown) = der_lown_filer_firet_nuniformpares - der_lown_filer		ort_ [st_ Float_→ ("See [Pred]; we cannot ensure that the stack is empty ") res (_t,t_def,ty) → ("See [Float]; we remove assume that the stack is empty ")	if TimpLones, guarante trans Ast pet trans (Do.1) = let min * Environ.lookup mind he new in mih.mind_macket.(i).mind_recorgs in bet trans * Array.mam (fam (mind.) \rightarrow out trans mind) minds in bet trans * Array.mam (fam (mind.) \rightarrow out trans mind) minds in
gene = spect: renv.gene ; let push_let renv (s,c,ty,upsc) = { enc = push_rel (t,c,slbrf (s,c,ty)) renv.env; nd sin = renv.rel sinch: }	(" the full-long functions are almost deplicated from independal, security that they carry here a power entirement (containing least information).") lest inser, parts, war (see, 'ze') ('.e., 'ze') ('.e., 'ze') ('.e., 'ze') ('.e., 'ze') [end-red (Constitution (x)) one, ('.e., 'ze') ((sind, ze'), 'per') let inser, parts, intentive Swaret (see, 'r., ze, 'ze') ((sind, ze'), 'per') let sin be "swiret-long parts sind one is let sin be "swiret-long parts sind one is let stylene - sind-long parts sind one is let stylene - sind-long parts sind one is let stylene - sind-long parts sind one is	malevet in, path in,	the trajectropomoney - wan possepsus (amount, plasma remark remark and catefulnet the deep in the promotions ") Let them? - punk, take, classimas reme I stack in let remark - punk, take, reme remark sign in let remark - punk, take, reme remark sign in let remark - punk, take, reme remark - size in the contrast - contrast - contrast - contrast - size in the contrast - contrast	If the control of the	calls ") Sing (sk.jok.yapeif or (lary Not.mothers)) in s :: naw (k:1) norditroperson thack s in an W norditroperson thack its long.reporate You'r maddedice new all stack x a b - s the norminal control of the con	it rev' = pacting for the recent parameters recinds (listings) before incompresses = frequency and frequency recinds (listings) in the list of the recind of		Let $_{+}$ let $_{+}$ first $_{+}$ \Rightarrow (* See [Fred]: we cannot ensure that the stack is empty ") resp ($_{+}$ ($_{+}$ ($_{+}$ ($_{+}$ ($_{+}$ ($_{+}$)) \Rightarrow (* See [Fred]: we commod ensure that the stack is empty ") let $_{+}$ = Array, field left (check_inext_mathers_rec_call resp) as t	if Tagic, resc, garage two Let all point res (c.), and resure Let all - fortren. Lanks, paind in one in sh. and garage (c.), and revery Lat two - Array map (for (cind.)) - a gat, two sind) sinds in for i = 0 to Array, angle for (cind.) - 1 do Let (Mon, dout) - rest (c.) in Let row - a sing, row fore more (c.) tree. (c) in
pure "apaci; returbar y bar palici returbar y (mo - pub/ya) (mostalir (sc,ty)) returbar; spalici - returbar (mostalir (sc,ty)) returbar; spalici - returbar (mostalir) (mostalir) - spalici returbar (mostalir) - s	(" The following functions we about deplicated from integrand, a sough that they carry have a purpose enterposed, containing these information). ") that they carry have a purpose enterposed (containing the information). ") (party-file (contained (p.d)) may, (flower, pr)); that is in expany, integrated beautiful property (flower, pr) ((flower, p), (part) = ist with "fortives. Intelligent party (flower, pr) (flower, p), (party) = ist with "fortives. Intelligent party (flower, p), (party) = ist with "enterposed party (p), (p), (p), (p) = (p), (p), (p), (p), (p), (p), (p), (p),	mandownt majoret major	the taghtroposomery was passenges amounts. As an even even were con- dense todays as punishing the figure memoriars? Let stack "punkt, tack, claimous even i stack in let rese" — punkt, tack, claimous even i stack in let rese" — punkt, tack, claimous even i stack in let rese" — punkt, tack, claimous even i stack in let rese" — if intil length stack "eldfillast then rese" also let rese the stack tack' deckey in let an even "tack deserve seef however deckers in the stack of the stack of t	If the Life part the case the measurements many record and individual conditions that Licety and the Life part of the Life pa	calls ") and call the second of the second	in the 'special point of 'special point of the 'special point of		ort. Det. Flant> ("Den (Pred): we convex somme that the stack is empty ") To For (Sept. (Sept. def. flant): For (Sept. (Sept. def. flant): For (Sept. flant	is regularized, gardened comes sub-scale for sub-scale for sub-scale for sub-scale for sub- scale for sub-scale for sub-scale for sub-scale for sub-scale for sub-scale for sub-scale for sub-scale for for i = 0 to Kreya, Sumply Scales = 1 on scale (few. Joseph.) - scale (file) in last row = sub-grown form rounce(1) from.() in they scale for sub-scale for sub-scale for sub-scale for scale for sub-scale for sub-scale for sub-scale for sub- graphic forms to scale for sub-scale for sub-scale for sub- fer scale for sub-scale for sub-scale for sub-scale for sub- fer sub-scale for scale for sub-scale for su
que a que carrecta que y la que a que carrecta que a que a que carrecta (que que a que a que carrecta (que que a que a que carrecta (que que a que a que a que carrecta (que que a que a que a que carrecta (que que a que a que a que a que a que a que a que a que a q	(" he stilling feature are about adjusted from intigental, amough this flow grave from a promise mentioned intigental gain information,") let inter, parks of (m, p, 10) (m, p, n). Let inter, parks of (m, p, 10) (m, p, n). Let inter, parks of (m, p, n). Let inter, of (m, n). Let inte	maleses a grade (the (Gardapin (gast_A)))) paths is a grade (the (Gardapin (gast_A)))) paths is the transpar k -free passion of the transpar in k -free passion k -free partial k -free part	is the "of the programme of the programm	It this, a plant to two the assumed to the series of the s	calls ") and call the second of the second	is the "property of the property of the proper		Let $_{+}$ let $_{+}$ first $_{+}$ \Rightarrow (* See [Fred]: we cannot ensure that the stack is empty ") resp ($_{+}$ ($_{+}$ ($_{+}$ ($_{+}$ ($_{+}$ ($_{+}$)) \Rightarrow (* See [Fred]: we commod ensure that the stack is empty ") let $_{+}$ = Array, field left (check_inext_mathers_rec_call resp) as t	if right, reading parties than in the distribution parties that is the first housing parts in one in min. And, packets (1), sind, recents let true = Array, ang (miss.,) -> get_true mind) which in for 1 = 0 to Array, length budges -1 do let (Men., bady) -> can't (1) true. (1) in try chad, ang it's hower new need trues long try chad, ang it's hower new need trues long try chad, ang it's hower new need trues long to
pure "apaci; returbar y bar palici returbar y (mo - pub/ya) (mostalir (sc,ty)) returbar; spalici - returbar (mostalir (sc,ty)) returbar; spalici - returbar (mostalir) (mostalir) - spalici returbar (mostalir) - s	(* In this bidding features was about adjusted from indigental, amough that they sure yet was possessed sentendent place in the relation, ") that they sure yet was possessed sentendent, ") (and red. (insidence (xx)) we, (insert, x)) (find x,x) (insidence (xx)) we, (insert, x)) (find x,x), have been a find that the footence, those gainst sink on the find that the final results with a find the final results with a find the final results with a fincome final results with a final results with a final results wit	mandownt majoret major	the target comment of the comments and the comments and the comments of the co	It became the desirable and the association to the support to the control of the	calls "] Soly (but that question or (lawy Nor, mileses)) in Soly (but that question or (lawy Nor, mileses)) in Soly (but that question or sold sold sold sold sold sold sold sold	in the 'special point of 'special point of the 'special point of	in (*	set z [let z .] Final z . ∞ (" for [Prol]) we extend remove that the stant is unjuy") re respectively. The property of the stant is unjuy") for [Prol] we extend remove that the stant is unjuy") let $n = r$ r r r r r r r r r	is regularized, gardened comes sub-scale for sub-scale for sub-scale for sub-scale for sub- scale for sub-scale for sub-scale for sub-scale for sub-scale for sub-scale for sub-scale for sub-scale for for i = 0 to Kreya, Sumply Scales = 1 on scale (few. Joseph.) - scale (file) in last row = sub-grown form rounce(1) from.() in they scale for sub-scale for sub-scale for sub-scale for scale for sub-scale for sub-scale for sub-scale for sub- graphic forms to scale for sub-scale for sub-scale for sub- fer scale for sub-scale for sub-scale for sub-scale for sub- fer sub-scale for scale for sub-scale for su
gas vages: rescapes; if are capting (scale) (e.g., type) = (ser - page, th) (scale) (e.g., typ) resc. energ (ser - page, th) (scale) (e.g., typ) resc. energ (ser - page, tree (ser (f., page)) intensity are page resc (f., page) intensity are page resc (f., page) intensity are page resc (f., page) intensity are page (f., page) intensity are page (f., page) intensity are page (f., page) (intensity are page)	(" he shiring feature we should adjusted for a disputal, a cough that they carry two again and constant of the country of the	is already as the property of	The tright dependency was questions, among class or near retraction and the state of the state o	The control of the co	calls ") and call the second of the second	The control of the co	in (*	not _ list _ Plast_ or (" for [Prol]' we creat more that the stack is empty") re reproject_AdA_AD_yellow (" for [Prol]' we come amount that the stack is empty") but no * prop_for_AD_AD_yellow_come_not come or not but no * prop_for_AD_AD_yellow_come_not come not do to but no * prop_for_AD_AD_yellow_come_not come not do to but no * prop_for_AD_AD_yellow_come_not come not do to	is tragicallous, gardent come in the sink foreign content of the sink foreign content
gave regarder recognition of the control of the con	(* The Ministry functions was about adjustment from indigenously, amount, to the time pure types are presented (ministrate), into indigenously, amount (ministrate) (makes (ministrate (makes (ministrate (makes (ministrate) (ministrate))))) (ministrate)	is always and the second of t	with the control of t	and includations The contract of the contract	called "Section of the Conference of the Confere	In antiferent and in antiferent antiferent and in antiferent and in a first part of the first parts of the first par	in (*) () and check	and a line of lines are the state of any of a line of lines and the state of any of a line of lines are the state of any of a line of lines are the state of any of a line of a line of lines are the state of any of a line of a line of lines are the state of any of a line of lines are the state of any of a line of a line of lines are the state of any of a line of lines are the state of lines are th	is tragicallocal, gardent com- lice that 'ferrican' indexing print is now in which captured, to justify a room in which captured, to justify a room in better to be the room in the first transition of the letter to be the room in the first transition of the letter to be the room in the first transition of the letter to be the room in the first transition of the letter to be the room in the room in the letter (tense, our room count true to body with 'thin antiferrical ('tense, our ') were July Justify to the room in the room in the were July Justify to high the room ('pure arrange) in the room in the room in the room in the room in the were July Justify to high the room ('pure arrange) in the room in the ro
gas vages: rescapes; if are capting (scale) (e.g., type) = (ser - page, th) (scale) (e.g., typ) resc. energ (ser - page, th) (scale) (e.g., typ) resc. energ (ser - page, tree (ser (f., page)) intensity are page resc (f., page) intensity are page resc (f., page) intensity are page resc (f., page) intensity are page (f., page) intensity are page (f., page) intensity are page (f., page) (intensity are page)	(*) The Ministry functions are shorted adjustment from indigeneral, a rough in the content of th	in advanced in a particular of the property of	whether the control of the control o	The control of the co	called "Section of the Conference of the Confere	The control of the co	in (*) (*) (*) (*) (*) (*) (*) (*	set $_{i}$ [be $_{i}$ [rises, $_{i}$ = (i.e. Drawl) = count enters that the class is equity?) or $_{i}$ [c. [c	is tragicallocal, gardent com- lice that 'ferrican' indexing print is now in which captured, to justify a room in which captured, to justify a room in better to be the room in the first transition of the letter to be the room in the first transition of the letter to be the room in the first transition of the letter to be the room in the first transition of the letter to be the room in the room in the letter (tense, our room count true to body with 'thin antiferrical ('tense, our ') were July Justify to the room in the room in the were July Justify to high the room ('pure arrange) in the room in the room in the room in the room in the were July Justify to high the room ('pure arrange) in the room in the ro
gave - specific recording is gave - specific recording in gave - specific recording gave - speci	(* In this beginning fractions was about adjustment from indignosal, amount to the upper rate of an experimental (manadiscular the upper rate of an experimental (manadiscular the upper rate of part	is always to desire the property of the prope	with the control of t	and includations of the control of t	called " and " of the state of	In antiferent and the control of the	in (*) (and chucket check ch	and c. The c. These, are "for formally or the third in supply"). The formally are counter among that the state is supply") and formally counter among for the counter among the third is supply. The formally supply of the formally of the formally supply of the formally of th	is tragicallocal, gardent com- lice that 'ferrican' indexing print is now in which captured, to justify a room in which captured, to justify a room in better to be the room in the first transition of the letter to be the room in the first transition of the letter to be the room in the first transition of the letter to be the room in the first transition of the letter to be the room in the room in the letter (tense, our room count true to body with 'thin antiferrical ('tense, our ') were July Justify to the room in the room in the were July Justify to high the room ('pure arrange) in the room in the room in the room in the room in the were July Justify to high the room ('pure arrange) in the room in the ro
gave regarder recognition of the control of the con	(*) The Ministry functions are shorted adjustment from indigeneral, a rough in the content of th	in advanced in a particular of the property of	whether the control of the control o	The control of the co	called " and " of the state of	In and regiment. The confirmation is a confirmation of the confirm	in (*) (*) (*) (*) (*) (*) (*) (*	set $_{i}$ [be $_{i}$ [rises, $_{i}$ = (i.e. Drawl) = count enters that the class is equity?) or $_{i}$ [c. [c	is tragicallocal, gardent com- lice that 'ferrican' indexing print is now in which captured, to justify a room in which captured, to justify a room in better to be the room in the first transition of the letter to be the room in the first transition of the letter to be the room in the first transition of the letter to be the room in the first transition of the letter to be the room in the room in the letter (tense, our room count true to body with 'thin antiferrical ('tense, our ') were July Justify to the room in the room in the were July Justify to high the room ('pure arrange) in the room in the room in the room in the room in the were July Justify to high the room ('pure arrange) in the room in the ro

The Guard Checker of Rocq

- · About 1,000 lines of **unspecified, unexplained** OCaml code
- Iterated by different authors over 30 years
- Multiple dimensions of complexity

Contribution

Two main contributions of this project:

Implementation

A full implementation of Rocq's Guard Checker in Rocq, using the MetaRocq project.

Extending previous work by Lennard Gäher [Gäh21].

Documentation

In the report: examples (Chapter 2, Appendix) and explanations (Chapter 3).

Available on HAL.

MetaRocq project

- Formalises Rocq's type theory in Rocq (faithful) [Soz+20a]
- A verified implementation of type checker [Soz+20b]
- A verified extraction function to OCaml [FST24]

Proved:

- Subject Reduction and Canonicity,
- parameterised by a guard checker
- assumed Normalisation

Implementation of the Guard Checker

```
From MetaRocq.Guarded Require Import plugin.
(* define your fixpoint *)
Fixpoint add (m n : nat) : nat :=
  match m with
  | 0 => n
  S m' => add m' (S n)
  end.
MetaRocq Quote add_syntax := add.
Check check_fix.
Compute (check_fix add_syntax).
```

Implementation of the Guard Checker

```
From MetaRocq.Guarded Require Import plugin.

(* define your fixpoint *)
Fixpoint add (m n : nat) : nat :=
  match m with
  | 0 => n
  | S m' => add m' (S n)
  end.
```

```
MetaRocq Quote add_syntax := add.
Check check_fix.
Compute (check_fix add_syntax).
```

Implementation of the Guard Checker

```
From MetaRocq.Guarded Require Import plugin.
(* define your fixpoint *)
Fixpoint add (m n : nat) : nat :=
  match m with
  | 0 => n
  S m' \Rightarrow add m' (S n)
  end.
MetaRocq Quote add_syntax := add.
Check check fix.
Compute (check_fix add_syntax).
```

```
add syntax : Ast.term :=
  (Ast.tFix [{|
    dname := {| binder_name := nNamed "add" |};
    dtype := Ast.tProd { | binder_name := nNamed "m" | }
      (Ast.tInd { | inductive mind := "nat" | } [])
     (\ldots):
    dbody :=
      Ast.tLambda
       {| binder name := nNamed "m" |}
        (Ast.tInd {| inductive_mind := "nat"; inductive_ind := 0 |} [])
        (Ast.tLambda
          {| binder_name := nNamed "n" |}
          (Ast.tInd {...} [])
          (Ast.tCase
              {| ci ind := {| inductive mind := "nat" |}: |}
              {| Ast.pcontext := [{| binder_name := nNamed "m"; |}];
                 Ast.preturn := Ast.tInd {| inductive_mind := "nat" |} []
              1}
              (Ast.tRel 1)
              [ { | Ast.bcontext := []; Ast.bbody := Ast.tRel 0 | };
                {| Ast.bcontext := [{| binder_name := nNamed "m'"; |}];
                   Ast.bbody :=
                     Ast.tApp (Ast.tRel 3)
                       [Ast.tRel 0;
                        Ast.tApp
                          (Ast.tConstruct { | inductive_mind := "nat"; inductive_ind := 0 |})
                          [Ast.tRel 1]]
                 |}]));
          rarq := 0
      [{]
```

Implementation of the Guard Checker

```
From MetaRocq.Guarded Require Import plugin.
(* define your fixpoint *)
Fixpoint add (m n : nat) : nat :=
  match m with
  | 0 => n
  | S m' => add m' (S n)
  end.
MetaRocq Quote add_syntax := add.
Check check_fix.
Compute (check_fix add_syntax).
```

check_fix : Ast.term -> bool

Implementation of the Guard Checker

```
From MetaRocq.Guarded Require Import plugin.
(* define your fixpoint *)
Fixpoint add (m n : nat) : nat :=
  match m with
  | 0 => n
  | S m' => add m' (S n)
  end.
MetaRocq Quote add_syntax := add.
Check check_fix.
Compute (check_fix add_syntax).
```

= true : bool

History of the Guard Checker

Phase 1: Beginnings

- Inductive + CoC = CIC [Pfenning and Paulin-Mohring (1989)]
- Pattern Matching with Dependent Types [Coquand (1992)]
- The first Guard Checker in Rocq v5.10.2 by Paulin-Mohring [Cornes et al. (1996)]

Phase 2: Specifications

- Inductive + CoC = CIC [Pfenning and Paulin-Mohring (1989)]
- Pattern Matching with Dependent Types [Coquand (1992)]
- The first Guard Checker in Rocq v5.10.2 by Paulin-Mohring [Cornes et al. (1996)]
- · Codifying Recursive Definition with Recursive Schemes [Giménez (1994)]
- Inductive Definitions for Type Theory [Paulin-Mohring (1996)]
- Un Calcul de Constructions Infinies et son application à la vérification de systèmes communicants [Giménez (1996)]

Phase 3: Big Changes

- Inductive + CoC = CIC [Pfenning and Paulin-Mohring (1989)]
- Pattern Matching with Dependent Types [Coquand (1992)]
- The first Guard Checker in Rocq v5.10.2 by Paulin-Mohring [Cornes et al. (1996)]
- Codifying Recursive Definition with Recursive Schemes [Giménez (1994)]
- Inductive Definitions for Type Theory [Paulin-Mohring (1996)]
- Un Calcul de Constructions Infinies et son application à la vérification de systèmes communicants [Giménez (1996)]
- β - ι commutative cuts subterm rule (Pierre Boutillier, 2010) [Bou12]

```
match v2 in with
| nil => (fun _ => nil C)
| cons h2 t2 => (fun t1' => cons (f h1 h2) (map2 f t1' t2))
end t1
```

Two Weeks before Christmas, 2013

```
From: Daniel Schepler <dschepler AT qmail.com>
To: Coq Club <coq-club AT inria.fr>
Subject: [Coq-Club] bijective function implies equal types is provably inconsistent with functional extensionality in
Coq
Date: Thu, 12 Dec 2013 11:02:00 -0800
Section bijective_impl_eq.
Hypothesis functional_extensionality :
  forall (A B:Type) (f q:A->B),
  (forall x:A, f x = q x) -> f = q.
. . .
Definition not_bijective_impl_eq : False := func_unit_discr unit_eq_False_False_funs.
End bijective_impl_eq.
Daniel Schepler
```

Phase 3: Big Changes

- Inductive + CoC = CIC [Pfenning and Paulin-Mohring (1989)]
- Pattern Matching with Dependent Types [Coquand (1992)]
- The first Guard Checker in Rocq v5.10.2 by Paulin-Mohring [Cornes et al. (1996)]
- Codifying Recursive Definition with Recursive Schemes [Giménez (1994)]
- Inductive Definitions for Type Theory [Paulin-Mohring (1996)]
- Un Calcul de Constructions Infinies et son application à la vérification de systèmes communicants [Giménez (1996)]
- β - ι commutative cuts subterm rule [Boutillier (2012)]
- · Restore compatibility with Propositional Extensionality [Dénès (2014)]

Phase 3: Big Changes

- Inductive + CoC = CIC [Pfenning and Paulin-Mohring (1989)]
- Pattern Matching with Dependent Types [Coquand (1992)]
- The first Guard Checker in Rocq v5.10.2 by Paulin-Mohring [Cornes et al. (1996)]
- Codifying Recursive Definition with Recursive Schemes [Giménez (1994)]
- Inductive Definitions for Type Theory [Paulin-Mohring (1996)]
- Un Calcul de Constructions Infinies et son application à la vérification de systèmes communicants [Giménez (1996)]
- β - ι commutative cuts subterm rule [Boutillier (2012)]
- · Restore compatibility with Propositional Extensionality [Dénès (2014)]
- Restore strong normalisation [Herbelin (2022)]
- Extrude uniform parameters [Herbelin (2024)]

A Taste of the Guard Checker

```
Fixpoint add (m n : nat)
  {struct m} : nat :=
  match m with
  | 0 => n
  | S m' => add m' (S n)
  end.
```

Goal: check that add is guarded.

Guarded: All recursive calls have a **strict subterm** as the **recursive argument**.

```
Fixpoint add (m n : nat)
  {struct m} : nat :=
  match m with
  | 0 => n
  | S m' => add m' (S n)
  end.
```

Subterm Specification

With respect to the **recursive parameter** m, terms can be a

- Large Subterm (e.g. m)
- •
- •

```
Fixpoint add (m n : nat)
  {struct m} : nat :=
  match m with
  | 0 => n
  | S m' => add m' (S n)
  end.
```

Subterm Specification

With respect to the **recursive parameter** m, terms can be a

- Large Subterm (e.g. m)
- Strict Subterm (e.g. m')

•

```
Fixpoint add (m n : nat)
  {struct m} : nat :=
  match m with
  | 0 => n
  | S m' => add m' (S n)
  end.
```

Subterm Specification

With respect to the **recursive parameter** m, terms can be a

- Large Subterm (e.g. m)
- Strict Subterm (e.g. m')
- Not Subterm (e.g. n)

```
Fixpoint add (m n : nat)
  {struct m} : nat :=
  match m with
  | 0 => n
  | S m' => add m' (S n)
  end.
```

Guard Env : [n:Bound{1}|m:Large|add]

Guard Environment

Subterm specifications of terms in the local context are stored.

```
Fixpoint add (m n : nat)
  {struct m} : nat :=
  match m with
  | 0 => n
  | S m' => add m' (S n)
  end.
```

```
Guard Env : [...]
Stack : [Closure m'|Closure(S n)]
```

Stack of subterm specifications

The subterm information of arguments are stored on a stack when checking the head of an application.

```
Fixpoint add (m : nat) :=
fun (n : nat) =>
match m return nat with
| 0 => n
| S m' => add m' (S n)
end.
```

```
Guard env: [m:Large|add]
Stack: []
```

Initial state. Parameters after the recursive parameter are turned into lambdas.

```
Guard env: [m:Large|add]
Stack: []
```

For a lambda to be guarded, its

- · binder type must be guarded, and
- body must be guarded.

Binder type is guarded.

```
Guard env: [n:Bound{1}|m:Large|add]
Stack: []
```

The body is checked with a updated guard environment.

Guard env: [n:Bound{1}|m:Large|add]

Stack: []

For a match to be guarded, its

- · discriminant,
- return type, and
- every branch

must be guarded.

```
Fixpoint add (m : nat) :=
  fun (n : nat) =>
  match m return nat with
  | 0 => n
  | S m' => add m' (S n)
  end.
```

Guard env: [n:Bound{1}|m:Large|add]

Stack: []

Discriminant (m) and the return type (nat) are guarded.

```
Guard env: [n:Bound{1}|m:Large|add]
Stack: []
```

To check a branch:

- expand into a lambda
- specify parameters
- · check the lambda

```
Fixpoint add (m : nat) :=
  fun (n : nat) =>
  match m return nat with
  | 0 => n
  | S m' => add m' (S n)
  end.
```

```
Guard env: [n:Bound{1}|m:Large|add]
```

Stack: []

0-th branch has no parameter.

- · expand into a lambda
- specify parameters
- · check the "lambda": guarded.

```
Guard env: [n:Bound{1}|m:Large|add]
Stack: [m':Strict]
```

1-st branch:

- · expand into a lambda
- specify parameters: strict!

•

```
Guard env: [n:Bound{1}|m:Large|add]
```

Stack: [m':Strict]

1-st branch:

- expand into a lambda
- specify parameters: strict!
- · check the lambda

1-st branch:

- · expand into a lambda
- specify parameters: strict!
- · check the lambda

Application with the recursive call is guarded if

- arguments are all guarded, and
- key case: the recursive argument is a strict subterm (on the stack)

Stack: [Closure m'|Closure(S n)]

Arguments are checked from right to left: both guarded.

Stack is populated with closures.

Fixpoint add (m : nat) :=

Since the recursive parameter of add is at position 0, specify the 0-th element of the stack.

Since the recursive parameter of add is at position 0, specify the 0-th element of the stack.

m' is a **strict** subterm according to the Guard Environment.

Since the recursive parameter of add is at position 0, specify the 0-th element of the stack.

m' is a **strict** subterm according to the Guard Environment.

Done!

What if...

Delayed? Answer: Stack handles this well.

```
Fixpoint add (m n : nat) {struct m} : nat :=
  (fun k => match k with
  | 0 => n
  | S m' => add m' (S n)
  end) m.
```

What if...

Obfuscated? Answer: weak-head reduction **only** when checking subterm specification.

```
Fixpoint add (m n : nat) {struct m} : nat :=
  (fun k => match (id k) with
  | 0 => n
  | S m' => add (pred (S m')) (S n)
  end) m.
```

```
What if...
```

Not guarded in erasable subterms?

Answer: strong normalisation (reduction only when needed).

```
Fail Fixpoint add (m n : nat) {struct m} : nat :=
let _ := add m (add m m) in
  (fun k => match (id k) with
  | 0 => n
  | S m' => add (pred (S m')) (S n)
  end) m.
```

What if...

Not guarded in erasable subterms?

Answer: strong normalisation (reduction only when needed).

```
Fail Fixpoint add (m n : nat) {struct m} : nat :=
let _ := add m (add m m) in
  (fun k => match (id k) with
  | 0 => n
  | S m' => add (pred (S m')) (S n)
  end) m.
```

Not covered in example: β - ι cuts, redex stack, nested fix, ...

The (at least) 4 Dimensions of Complexity

Dimensions of Complexity

- 1. The stack of subterm specifications for β - ι commutative cuts
- 2. Strong normalisation:
 - a redex stack
 - only reduce terms to weak-head normal form when needed
- 3. Support for mutual and nested fixpoints
 - regular trees
- 4. OCaml lazy for efficiency

Resulting in 1,000 lines of OCaml code.

Full Implementation in Rocq

- · Complete, available as a MetaRocq (TemplateRocq) plugin.
- Feature parity with the kernel
- Test parity* with the kernel
- Intentionally kept as close as possible to Rocq's guard checker
- Available at: https://github.com/inria-cambium/m1-tan/tree/v1.0.0

Conclusion and Future Work

Summary

Conclusion

- implemented the Guard Checker in Rocq (code on <u>Github</u>)
- documented its features
- gave examples of its behaviour
- full report on HAL: https://inria.hal.science/hal-04983786

Future Work

- verify that the guard checker itself is a terminating program
- specification of an abstract guard condition of the checker
- verify that the guard checker implements the guard condition
- relative consistency proofs for its soundness

Well-Founded Recursion

- An alternative to structural recursion
- · Rocq: structural by default; well-founded using Program Fixpoint or Equations

Lean: structural by default; well-founded attempted otherwise (termination_by)

Agda: structural by default; well-founded using Induction. WellFounded

Agda: Semantic Termination Checking

	Syntactic	Semantic
Example	Rocq	Agda
Reduction	Minimal	Full
Mechanism	Guard	(Possible) Sized Types
Advantage	Fast	Accurate

- Chan, Li, and Bowman [CLB23] attempted Sized Types in Rocq in 2019, compilation time increased as much as 5-15x on the Rocq Standard Library.
- New algorithm in Agda by Nisht and Abel [NA24] is linear on input, but not yet proven complete.

Lean: Native Eliminators

- Lean is the opposite of Rocq: **eliminators** are native in the kernel, recursive functions only exist in the surface syntax
- Type Checking:
 - 1. Eliminators are generated for Inductive Types
 - 2. A strong (aka course-of-values) induction principle is defined using the said eliminators
 - 3. Recursive functions are translated into an encoding by the strong induction principle
- Extraction (Code Generation/Compilation) to C: the syntax gets extracted as-is
- Advantage: eliminators are simpler for the theory
 Disadvantage: hard to prove extraction correct, possible suprising behaviour

[PP89]	Semantics, 5th International Conference, Tulane University, New Orleans, Louisiana, USA, March 29 - April 1, 1989, Proceedings, M. G. Main, A. Melton, M. W. Mislove, and D. A. Schmidt, Eds., in Lecture Notes in Computer
	Science, vol. 442. Springer, 1989, pp. 209–228. doi: <u>10.1007/BFB0040259</u> .
[Gäh21]	L. Gäher, "Guard Checker in MetaCoq." GitHub, 2021.
[Soz+20]	M. Sozeau <i>et al.</i> , "The MetaCoq Project," <i>J. Autom. Reason.</i> , vol. 64, no. 5, pp. 947–999, 2020a, doi: <u>10.1007/S10817-019-09540-0</u> .
[Soz+20]	M. Sozeau, S. Boulier, Y. Forster, N. Tabareau, and T. Winterhalter, "Coq Coq correct! verification of type checking and erasure for Coq, in Coq," <i>Proc. ACM Program. Lang.</i> , vol. 4, no. POPL, pp. 1–28, 2020b, doi: 10.1145/3371076.

F. Pfenning and C. Paulin-Mohring, "Inductively Defined Types in the

Calculus of Constructions," in Mathematical Foundations of Programming

[FST24]	Y. Forster, M. Sozeau, and N. Tabareau, "Verified Extraction from Coq to OCaml," <i>Proc. ACM Program. Lang.</i> , vol. 8, no. PLDI, Jun. 2024, doi: 10.1145/3656379.
[6]	T. Coquand, "Pattern matching with dependent types," in <i>Informal proceedings of Logical Frameworks</i> , 1992, pp. 66–79.
[7]	C. Cornes et al., "The Coq Proof Assistant-Reference Manual," INRIA Rocquencourt and ENS Lyon, version, vol. 5, 1996.
[8]	E. Giménez, "Codifying Guarded Definitions with Recursive Schemes," in Types for Proofs and Programs, International Workshop TYPES'94, Båstad, Sweden, June 6-10, 1994, Selected Papers, P. Dybjer, B. Nordström, and J. M. Smith, Eds., in Lecture Notes in Computer Science, vol. 996. Springer, 1994, pp. 39–59. doi: 10.1007/3-540-60579-7_3.

[9]	C. Paulin-Mohring, Définitions Inductives en Théorie des Types. (Inductive Definitions in Type Theory). 1996. [Online]. Available: https://tel.archives-ouvertes.fr/tel-00431817
[10]	E. Giménez, "Un Calcul de Constructions Infinies et son application a la vérification de systemes communicants," 1996.
[Bou12]	P. Boutillier, "A relaxation of Coq's guard condition," in <i>JFLA - Journées Francophones des langages applicatifs - 2012</i> , Carnac, France, Feb. 2012, pp. 1–14. [Online]. Available: https://hal.science/hal-00651780
[12]	M. Dénès, "Tentative fix for the commutative cut subterm rule." [Online]. Available: https://github.com/coq/coq/commit/9b272a861bc3263c69b699cd2ac40ab2606543fa
[13]	H. Herbelin, "Check guardedness of fixpoints also in erasable subterms." [Online]. Available: https://github.com/coq/coq/pull/15434

[14]	H. Herbelin, "Extrude uniform parameters of inner fixpoints in guard condition check." [Online]. Available: https://github.com/coq/coq/pull/17986
[CLB23]	J. Chan, Y. Li, and W. J. Bowman, "Is sized typing for Coq practical?," <i>J. Funct. Program.</i> , vol. 33, p. e1, 2023, doi: <u>10.1017/S0956796822000120</u> .
[NA24]	K. Nisht and A. Abel, "Type-Based Termination Checking in Agda," 30th International Conference on Types for Proofs, Programs, TYPES 2024, pp. 32–33, 2024. [Online]. Available: https://types2024.itu.dk/abstracts.pdf