add

`Fixpoint add (m n : nat):`
`match m with`
`| O => n`
`| S m' => add m' (S n)`
`end.`
	call stack : []
	stack : []
	redex stack : []
	guard env : []
	loc env : []

`fun (n : nat) =>`
`match m with`
`| O => n`
`| S m' => add m' (S n)`
`end.`
	call stack : [tLambda]
	stack : []
	redex stack : [NoNeed]
	guard env : [Large]
	loc env : $[m, \text{add}]$

`match m with`
`| O => n`
`| S m' => add m' (S n)`
`end.`
	call stack : [tCase, tLambda]
	stack : []
	redex stack : [NoNeed]
	guard env : [Bound{1}, Large]
	loc env : $[n, m, \text{add}]$

`m (* discriminant *)`	call stack : [tRel, tCase, tLambda] stack : [] redex stack : [NoNeed, NoNeed] guard env : [Bound{1}, Large] loc env : $[n, m, \text{add}]$
`n (* 0-th branch *)`	call stack : [tRel, tCase, tLambda] stack : [] redex stack : [NoNeed, NoNeed] guard env : [Bound{1}, Large] loc env : $[n, m, \text{add}]$
`fun m' : nat => add m' (S n)` `(* 1-st branch *)`	call stack : [tLambda, tCase, tLambda] stack : [] redex stack : [NoNeed, NoNeed] guard env : [Bound{1}, Large] loc env : $[n, m, \text{add}]$
`add m' (S n)`	call stack : [tApp, tCase, tLambda] stack : [] redex stack : [NoNeed, NoNeed] guard env : [Strict, Bound{1}, Large] loc env : $[m', n, m, \text{add}]$

`S n`	call stack : [tApp, tApp, tCase, tLambda] stack : [] redex stack : [NoNeed, NoNeed, NoNeed] guard env : [Strict, Bound{1}, Large] loc env : [m', n, m, add]
`n`	call stack : [tRel, tApp, tApp, tCase, tLambda] stack : [] redex stack : [NoNeed, NoNeed, NoNeed, NoNeed] guard env : [Strict, Bound{1}, Large] loc env : [m', n, m, add]
`S`	call stack : [tConstruct, tApp, tApp, tCase, tLambda] stack : [SClosure n] redex stack : [NoNeed, NoNeed, NoNeed] guard env : [Strict, Bound{1}, Large] loc env : [m', n, m, add]
`m'`	call stack : [tRel, tApp, tCase, tLambda] stack : [] redex stack : [NoNeed, NoNeed, NoNeed] guard env : [Strict, Bound{1}, Large] loc env : [m', n, m, add]
`add`	call stack : [tRel, tApp, tCase, tLambda] stack : [SClosure m', SClosure (S n)] redex stack : [NoNeed, NoNeed] guard env : [Strict, Bound{1}, Large] loc env : [m', n, m, add]

`(* internal *)`
`check_is_subterm`
`  (subterm_specif m')`
`  (wf_paths nat)`
`== NeedReduceSubterm {}`
	call stack : [tRel, tApp, tCase, tLambda]
	stack : [SClosure m', SClosure (S n)]
	redex stack : [NoNeed, NoNeed]
	guard env : [Strict, Bound{1}, Large]
	loc env : [m', n, m, add]

`(* internal *)`
`reduce_if`
`  (needreduce discriminant ||`
`   needreduce branches)`
	call stack : [tRel, tApp, tCase, tLambda]
	stack : []
	redex stack : [NoNeed]
	guard env : [Bound{1}, Large]
	loc env : [n, m, add]

add_typo

`Fixpoint add_typo (m n : nat) :=`
`match m with`
`| O => n`
`| S unused => add_typo m (S n)`
`end.`
	call stack : []
	stack : []
	redex stack : []
	guard env : []
	loc env : []

`fun (n : nat) =>`
`match m with`
`| O => n`
`| S unused => add m (S n)`
`end.`
	call stack : [tLambda]
	stack : []
	redex stack : [NoNeed]
	guard env : [Large]
	loc env : [m, add]

`match m with`
`| O => n`
`| S unused => add m (S n)`
`end.`
	call stack : [tCase, tLambda]
	stack : []
	redex stack : [NoNeed]
	guard env : [Bound{1}, Large]
	loc env : [n, m, add]

`m (* discriminant *)`	call stack : [tRel, tCase, tLambda] stack : [] redex stack : [NoNeed, NoNeed] guard env : [Bound{1}, Large] loc env : [n, m, add]
`n (* 0-th branch *)`	call stack : [tRel, tCase, tLambda] stack : [] redex stack : [NoNeed, NoNeed] guard env : [Bound{1}, Large] loc env : [n, m, add]
`fun unused : nat => add m (S n)` `(* 1-st branch *)`	call stack : [tLambda, tCase, tLambda] stack : [] redex stack : [NoNeed, NoNeed] guard env : [Bound{1}, Large] loc env : [n, m, add]
`add m (S n)`	call stack : [tApp, tCase, tLambda] stack : [] redex stack : [NoNeed, NoNeed] guard env : [Strict, Bound{1}, Large] loc env : [unused, n, m, add]

`S n`	call stack : [tApp, tApp, tCase, tLambda] stack : [] redex stack : [NoNeed, NoNeed, NoNeed] guard env : [Strict, Bound{1}, Large] loc env : [unused, n, m, add]
`m`	call stack : [tRel, tApp, tCase, tLambda] stack : [] redex stack : [NoNeed, NoNeed, NoNeed] guard env : [Strict, Bound{1}, Large] loc env : [unused, n, m, add]
`add`	call stack : [tRel, tApp, tCase, tLambda] stack : [SClosure m, SClosure (S n)] redex stack : [NoNeed, NoNeed] guard env : [Strict, Bound{1}, Large] loc env : [unused, n, m, add]

`(* internal *)`
`check_is_subterm`
`  (subterm_specif m)`
`  (wf_paths nat)`
`== NotSubterm`
	call stack : [tRel, tApp, tCase, tLambda]
	stack : [SClosure m, SClosure (S n)]
	redex stack : [NoNeed, NoNeed]
	guard env : [Strict, Bound{1}, Large]
	loc env : [unused, n, m, add]

f

`Fixpoint f (x : bool) :=` `  let _ := f x in true.`	call stack : [] stack : [] redex stack : [] guard env : [] loc env : []
`Fixpoint f (x : bool) :=` `  let b := f x in true.`	call stack : [] stack : [] redex stack : [] guard env : [] loc env : []
`let b := f x in true.`	call stack : [tLetIn] stack : [] redex stack : [NoNeed] guard env : [Large] loc env : $[x, f]$
`f x (* bound term *)`	call stack : [tApp, tLetIn] stack : [] redex stack : [NoNeed, NoNeed] guard env : [Large] loc env : $[x, f]$
`x`	call stack : [tRel, tApp, tLetIn] stack : [] redex stack : [NoNeed, NoNeed, NoNeed] guard env : [Large] loc env : $[x, f]$
`f`	call stack : [tRel, tApp, tLetIn] stack : [SClosure x] redex stack : [NoNeed, NoNeed, NoNeed] guard env : [Large] loc env : $[x, f]$
`f`	call stack : [tRel, tApp, tLetIn] stack : [SClosure x] redex stack : [NoNeed, NoNeed, NoNeed] guard env : [Large] loc env : $[x, f]$

`(* internal *)`
`check_is_subterm`
`  (subterm_specif x)`
`  (wf_paths bool)`
`== NotSubterm`
	call stack : [tRel, tApp, tLetIn]
	stack : [SClosure x]
	redex stack : [NoNeed, NoNeed, NoNeed]
	guard env : [Large]
	loc env : [x, f]