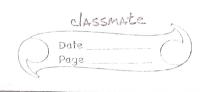
CONTINUATION PASSING INTERPRETERS e ::= n b (c e) (if e e e) (un (x)e) (assume ((xe)) e) x. samantic clamains Value Continuations: ast ans	10 A A A A A A A A A A A A A A A A A A A	Date
e::= n b (c e) Cif e e e) ((un (x)e) (assume ((xe))e) x. semantic clomains volve continuations: ast ans, env evol-ast k answer: ret (val) + exestr) Simple vs unbounded time function. Ct x s) (f x s) (k (+ x s)) (f' x s k) (define eval-ast /k (x (a env k) (cases (ast a [num (n) (k n)]		Page
(assume ((xe)) e) 1 x. semantic domains Value Continuation(: ast ans env evol-ast k ans env evol-ast conserver: ret (val) + extestr) Simple vs unbounded -time bunction: Ct x s) (f x s) (k (+ x s)) (f' x s k) (define evol-ast/k (x (a env k) (cases fasta [num (n) (k n)]		CONTINUATION PASSING INTERPRETERS
Value Continuations: ast ans, env avol-ast k Consever: ret (val) + evilstr) Simple vs unbounded time function: C+ n s) (f x s) (k (+ x s)) (f' x s k) Cdefine eval-ast/k (2 (a env k) Cases (asta Enum (n) (k n)]		e ::= n b (e e) (if e e e) (on (n) e) (assume ((xe)) e) n.
ast oval-ast ans oval-ast k consever: vet (val) + evilstr) Simple vs unbounded time bunction: (+ x s) (+ x s) (k (+ x s)) (- x s k) (define eval-ast/k (x (a env k) (cases (asta [num (n) (k n)]		semantic domains
answer: ret (val) + extestr) Simple vs unbounded-time bunction: C+ x s) (+ x s) (k (+ x s)) (+ x s k) (define eval-ast/k (x (a env k) (cases (asta [num (n) (k n)]		value continuations:
Simple vs unbounded-time function: Ct x s) (f x 5) (k (+ x 8)) (f' x 5 k) Cdefine eval-ast/k (x (a env k) Cases (asta [num (n) (k n)]		env eval-ast
Ct x s) (f x 5) (k (+ x 8)) (f' x 5 k) (define eval-ast/k (x (a env u) (cases fasta [num (n) (k n)]		answer! ret (val) + evilstr)
(k (+ x 5)) (f' x 5 k) (define eval-ast/k (x (a env k) (cases (asta [num (n) (k n)]		sumple us unbounded-time function:
(cases (asta [num (n) (k n)]		1
(cases (asta [num (n) (k n)]		
Cases (asta Enum (n) (k n)]		
[num (n) (k n)]		
id (a) (bokup-env/k env a k)		[num (n) (k n)]
id (a) (lookupenv/k env n k)		
		id (x) (lookupenv/k env x k)



Ceval-ast/k test env)
(coal-ast/k test env)

lif (boolean? v)

Coval-ask/K Cif v then else Jenu K)
RHOP-K* (format 4-- ~~4 v)))

Cassume Chinas body)
Coval-ast-binas/k binds env

(2 (rib)

than (formols body)

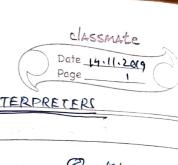
(k (make-dosure formals body env))

[app (rator rands)

rator

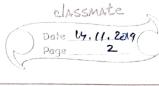
ranges.

e: = ... (abort e) | (break) | (resomev)



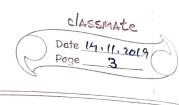
CONTINUATION PASSING INTERPRETERS

Dabort & try Roth @ letce @ break / resume @ throw > (+3 (abort 5)) > (+3 (break 5) > Gresumen) > (* resume * 7) to (define eval-ast/k (2 (ma env x) Labor (a) Ceral-aut/k (a) (en) *top-k*)] t break Ca) & (eval-a8t/k a) env Cset1 * resumer (morten-lambda (resume) -> L'() (kv)] (resone 4) [(list w) (kw)] (*top-k* v)))]]



e !:= | (try e v e) try block la (4 hrow e) (define eval-ast/k (2 (a env K) ex-K) [throw a) (eval-ast/k a env ex-k ex-k) [try Cabody exil handler) Ceval-ast/k body env K > Ca (exn) Clet (Enew-env Cextended-env (list exn-id) (list exn)) Ceval-ast/k handlen new-env K en-k)] Ceval-ast a *top-enux Fop-ex- *top-ex- kx Celefine * top-k* (2 (v) v)) Cdefine atop-en-k & (2 (v) "unrought exception "))

(oroutinal, thread) -> continuation program letcc.



(letre (K e) C+ 4 (letce 1 (K 5)) (44 Cloter (x 2 (R 5)) (define eval-ast Hs I leta Coym body) Cextended env - (let (thew env (list sym) (list (continuation-proc k)) Ceval-ask/k body new-enr K ex-k) Cachine apply-proc/K Eprimitive .. E closure -[continuation-proc Cap Cappiorol (14 14 angs)

Capply KI args) "

	nov 10 maker paire
	nov 10 maker paire
	aluminium x
	cander portel. apjandul madam.
	Caracter Caracter
ar d'acce	
(art) was	
ar (maior)	
Table 10	
Topode	
1/115	