

# Unstable-Languages-beta: SIMPLE-THR-3-Statements \*

The P<sub>Plan</sub>CompS Project

SIMPLE-THR-3-Statements.cbs | PLAIN | PRETTY

---

Language "SIMPLE-THR"

## 3 Statements

Syntax *Block* : **block** ::= '{' *stmts*? '}'  
*Stmts* : **stmts** ::= *stmt* *stmts*?  
*Stmt* : **stmt** ::= *imp-stmt* | *vars-decl*  
*ImpStmt* : **imp-stmt** ::= *block*  
| *exp* ';'   
| 'if' '(' *exp* ')' *block* ('else' *block*)?   
| 'while' '(' *exp* ')' *block*   
| 'for' '(' *stmt* *exp* ';' *exp* ')' *block*   
| 'print' '(' *exps* ')' ';'   
| 'return' *exp*? ';'   
| 'try' *block* 'catch' '(' *id* ')' *block*   
| 'throw' *exp* ';'   
| 'join' *exp* ';'   
| 'acquire' *exp* ';'   
| 'release' *exp* ';'   
| 'rendezvous' *exp* ';'

Rule  $\llbracket$  'if' '(' *Exp* ')' *Block*  $\rrbracket$  : *stmt* =  
 $\llbracket$  'if' '(' *Exp* ')' *Block* 'else' '{' '}'  $\rrbracket$

Rule  $\llbracket$  'for' '(' *Stmt* *Exp*<sub>1</sub> ';' *Exp*<sub>2</sub> ')'   
 '{' *Stmts* '}'  $\rrbracket$  : *stmt* =  
 $\llbracket$  '{' *Stmt*   
 'while' '(' *Exp*<sub>1</sub> ')'   
 '{' '{' *Stmts* '}' *Exp*<sub>2</sub> ';' '}'   
 '}'  $\rrbracket$

---

\*Suggestions for improvement: [plancomps@gmail.com](mailto:plancomps@gmail.com).  
Reports of issues: <https://github.com/plancomps/CBS-beta/issues>.

Semantics  $\text{exec}[\_ : \text{stmts}] : \Rightarrow \text{null-type}$

Rule  $\text{exec}[\text{'{' '}}] = \text{null}$

Rule  $\text{exec}[\text{'{' Stmt '}}] = \text{exec}[\text{Stmt}]$

Rule  $\text{exec}[\text{ImpStmt Stmt}] =$   
 $\text{sequential}(\text{exec}[\text{ImpStmt}], \text{exec}[\text{Stmt}])$

Rule  $\text{exec}[\text{VarsDecl Stmt}] =$   
 $\text{scope}(\text{declare}[\text{VarsDecl}], \text{exec}[\text{Stmt}])$

Rule  $\text{exec}[\text{VarsDecl}] = \text{effect}(\text{declare}[\text{VarsDecl}])$

Rule  $\text{exec}[\text{Exp ';' }] = \text{effect}(\text{rval}[\text{Exp}])$

Rule  $\text{exec}[\text{'if' '(' Exp ')' Block<sub>1</sub> 'else' Block<sub>2</sub>}] =$   
 $\text{if-else}(\text{rval}[\text{Exp}], \text{exec}[\text{Block<sub>1</sub>}], \text{exec}[\text{Block<sub>2</sub>}] )$

Rule  $\text{exec}[\text{'while' '(' Exp ')' Block}] = \text{while}(\text{rval}[\text{Exp}], \text{exec}[\text{Block}])$

Rule  $\text{exec}[\text{'print' '(' Exps ')' ';' }] = \text{print}(\text{rvals}[\text{Exps}])$

Rule  $\text{exec}[\text{'return' Exp ';' }] = \text{return}(\text{rval}[\text{Exp}])$

Rule  $\text{exec}[\text{'return' ';' }] = \text{return}(\text{null})$

Rule  $\text{exec}[\text{'try' Block<sub>1</sub> 'catch' '(' Id ')' Block<sub>2</sub>}] =$   
 $\text{handle-thrown}(\text{exec}[\text{Block<sub>1</sub>}],$   
 $\text{scope}(\text{bind}(\text{id}[\text{Id}], \text{allocate-initialised-variable}(\text{values}, \text{given})),$   
 $\text{exec}[\text{Block<sub>2</sub>}] ) )$

Rule  $\text{exec}[\text{'throw' Exp ';' }] = \text{throw}(\text{rval}[\text{Exp}])$

SIMPLE uses natural numbers to identify threads; the use of  $\text{lookup-index}(\_)$  below converts a natural number to the associated thread-id.

Rule  $\text{exec}[\text{'join' Exp ';' }] =$   
 $\text{thread-join } \text{lookup-index}(\text{rval}[\text{Exp}])$

The use of  $\text{memo-value}(V, SY)$  below associates  $V$  with a lock. When a thread requests a lock already held by another thread, the requesting thread is suspended until the request is granted. The use of  $\text{postpone}(\_)$  below automatically releases held locks when the current thread terminates.

Rule  $\text{exec}[\text{'acquire' Exp ';' }] =$   
 $\text{give}(\text{memo-value}(\text{rval}[\text{Exp}], \text{reentrant-lock-create}),$   
 $\text{sequential}(\text{postpone}$   
 $\text{if-true-else}(\text{is-exclusive-lock-holder given},$   
 $\text{reentrant-lock-release given},$   
 $\text{null-value}),$   
 $\text{reentrant-lock-sync-else-wait given}))$

The use of  $\text{memo-value-recall}(V)$  below gives the lock associated with  $V$ .

Rule  $\text{exec}[\text{'release' Exp ';' }] =$   
 $\text{reentrant-lock-exit } \text{memo-value-recall } \text{rval}[\text{Exp}]$

The use of  $\text{memo-value}(V, SY)$  below associates  $V$  with a rendezvous. When a thread requests a rendezvous on a particular value, and there is no previous uncompleted request for a rendezvous on the same value, the requesting thread is suspended until the request is granted.

*Rule* `exec`[[ 'rendezvous' *Exp* ';' ]] =  
rendezvous-sync-else-wait(  
memo-value("rendezvous", rendezvous-create(2)),  
`rval`[[ *Exp* ]])