ccaPL LANGUAGE QUICK REFERENCE GUIDE

ccaPL Program structure

The structure of a ccaPL program is

```
BEGIN_DECLS
 Declaration1
 ...
 Declarationn
END_DECLS
 P
```

where P is a process. A declaration can be

```
 display code                 display the transitions/reductions
 display congruence           display the congruences
 mode random                  reductions chosen randomly
 mode interleaving            reductions chosen interleaved
 def name(x0,..,xn) = { K }   function declaration
 length = xx                  execution length is xx transitions
```

Processes

Process P is denoted by

```
 0                                    inactivity
 M.P                                  prefix
 n[P]                                 ambient
 new n P                              restriction
 P1|P2                                parallel composition
 !P                                   replication
 <K>M.P                               context-guarded prefix
 proc x(y0,..,yn)  P                  process abstraction x
 {P}                                  brackets
 if <K1>M1.P1 .. <Kn>Mn.Pn fi         selection (if)
 if <K1>M1.P1 .. <Kn>Mn.Pn else P fi  selection (if-else)
 find x0,..,xn : K for P              search prefix
 let x0 = E0, ..,xn = En in P         let statement
```

Capabilities

Capabilities M are denoted

```
 in n               move into the ambient n
 out                move out
 del n              delete ambient n
 skip               a single transition
 L x(x0,..,xn)      call to the process abstraction x
 L recv(y0,..,yn)   receive list of messages y0,..,yn from L
 L send(z0,..,zn)   send list of messages z0,..,zn to L
```

Locations

Locations L are denoted by

```
@          any parent
n@         parent n
#          any child
n#         child n
::         any sibling
n::        sibling n
           locally
```

Context Expressions

Context expressions K are denoted

```
0               empty context
true            true
n = m           name match
n < m           less than
n <= m          less than or equal
n > m           greater than
n >= m          greater than or equal
this            hole
not K           negation
K1 | K2         parallel composition
K1 or K2        disjunction
K1 and K2       conjunction
n[K]            location
next K          spatial next modality
somewhere K     somewhere modality
```

Arithmetic Expressions

Arithmetic expressions E can be used in a `let` statement

```
let x0 = E0, ..,xn = En in P
```

Where arithmetic expression E is denoted by

```
x           a name: string or integer
- E         unary minus
E1 + E2     addition or string concatenation
E1 - E2     minus
E1 * E2     multiplication
E1 ** E2    power
E1 % E2     modulo
E1 / E2     division
```