Path Expressions

- a declarative specification of the synchronization desired among...
- a set of procedures that may be executed concurrently where...
- automatic enforcement of the synchronization is provided by automatically generated code that uses (an extended form of) semaphores.

With declarative approaches there is usually a tradeoff between the power of the expressions (i.e., the class of problems to which a solution can be expressed) and the feasibility of the expressions (i.e., the extent to which the expression can be translated into an (efficient) implementation).

General Scheme

**General Form**

The general form of a path expression is:

```plaintext
path <exp>, <exp>, ..., <exp> end
```

where `<exp>` is an expression formed from the following operators:

- sequencer: `x ; y`
  synchronizes the beginning of `y`
  with the completion of `x`
- restrictor: `n:( x )`
  limits to `n` the number of concurrent invocations of `x`
- derestrictor: `[ x ]`
  allows an unlimited number of concurrent invocations of `x`
- grouping: `( ... )`
  to express precedence or nesting

**Examples**

- sequencing
  ```plaintext
  path put; get end
  ```
  The get procedure cannot begin its ith invocation until the put procedure has completed its ith invocation.
  No synchronization is implied about concurrent executions of put or concurrent executions of get.
- restriction: mutual exclusion
  ```plaintext
  path 1:( write ) end
  ```
  only 1 procedure at a time can execute the procedure write.
- restriction: mutual exclusive choice
  ```plaintext
  path 1:( write, read ) end
  ```
  the procedures write and read cannot both be executing concurrently
- restriction: limited concurrency
  ```plaintext
  path 10:( read ) end
  ```
  up to 10 invocations of the read procedure can be in progress concurrently
- simple readers-writers
  ```plaintext
  path 1: ( write , [ read ] ) end
  ```
  Either exactly one write procedure is executed or an unbounded number of concurrent executions of read

**Examples**

- The producer-consumer problem for a buffer of size `n` has three constraints that can be stated in a single path expression:

  ```plaintext
  path 1:( put, get ), n: ( put ; get ) end
  ```

- buffer overflow
- buffer underflow
Translating Path Expressions

\[ \text{path } l: \text{ ( put , get ) } , n: \text{ ( put ; get ) end } \]

\[ \text{P(S1) put , get V(S1) } \]
\[ \text{P(S2) put ; get V(S2) } \]

\[ \text{P(S1) put V(S1) } \]
\[ \text{P(S1) get V(S1) } \]

\[ \text{P(S2) put V(S3) V(S1) } \]
\[ \text{P(S3) put V(S2) V(S1) } \]

\[ \text{semaphore S1 initially 1; } \]
\[ \text{semaphore S2 initially n; } \]
\[ \text{semaphore S3 initially 0; } \]