

Copyright ©1998, 1999 Timothy Howard Merrett  
Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and full citation in a prominent place. Copyright for components of this work owned by others than T. H. Merrett must be honoured. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, or to redistribute to lists, requires prior specific permission and/or fee. Request permission to republish from: T. H. Merrett, School of Computer Science, McGill University, fax 514 398 3883.

The author gratefully acknowledges support from the taxpayers of Québec and of Canada who have paid his salary and research grants while this work was developed at McGill University, and from his students (who built the implementations and investigated the data structures and algorithms) and their funding agencies.

T. H. Merrett

©98/11

# Concurrent Relational Programming

The Sequential (`--`) and Parallel (`||`) Combinators

```
relation CommonRel(attr)  $\leftarrow$   $\{(0)\}$ ;  
update CommonRel change attr  $\leftarrow$  attr +1 ||  
update CommonRel change attr  $\leftarrow$  attr -1;
```

This executes the two statements in any order (“parallel”), and leaves *CommonRel* unchanged because the operations cancel.

```
relation CommonRel(attr)  $\leftarrow$   $\{(0)\}$ ;  
( LocalRel1  $\leftarrow$  CommonRel --  
  update LocalRel1 change attr  $\leftarrow$  attr +1 --  
  CommonRel  $\leftarrow$  LocalRel1  
) ||  
( LocalRel2  $\leftarrow$  CommonRel --  
  update LocalRel2 change attr  $\leftarrow$  attr -1 --  
  CommonRel  $\leftarrow$  LocalRel2  
);
```

This has 20 possible interleavings. Two give the right result, 0; 9 give -1; and 9 give 1.

# Concurrent Relational Programming

## Synchronization

Synchronization is by a blocking T-selector:

if the result of an ordinary T-selector would be empty, the process blocks until the operand is changed to make the result not empty.

The **where** is replaced by **when**.

# Synchronization

This idea from *Linda*

(Carreiro & Gelernter, CACM 32 (1989) 444):

**out**("a string", 15.01, 17, "another string")

... puts a "tuple" into "tuple space".

**rd**("a string", ? f, ? i, "another string")

... reads it, assigning values to the variables;

*blocks* if no corresponding tuple;

*nondeterministically* returns one, if many tuples.

**in**("a string", ? f, ? i, "another string")

... like **rd** but *consumes* the tuple.

## Synchronization

```
relation tSpace(S1, N1, N2, S2) ←  
  {"a string", 15.01, 17, "another string"};  
synch ← when S1="a string"  
  and S1="another string" in tSpace;
```

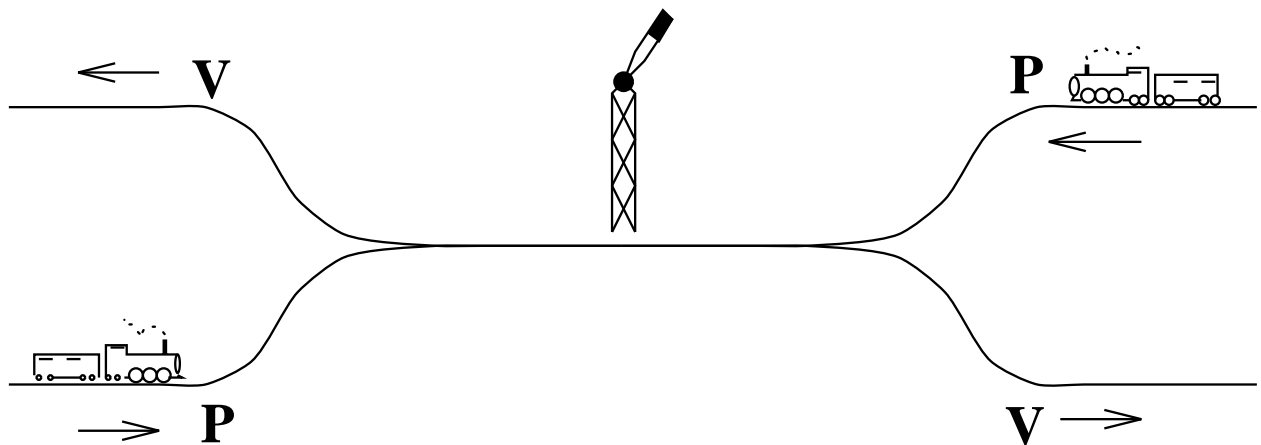
N.B. Does not consume; reads all.

(*Nondeterminism* is from another operator, another syntax, **pick**).

## Semaphores (Dijkstra, CACM 11 (1968) 341)

P(s) “proberen”      if s.cnt=0 then WAIT  
                                 dec(s.cnt)

V(s) “vrijgeven”      inc(s.cnt)  
                                 waken waiters



# Semaphores

Here is a semaphore,

```
relation SEMAPHORE(Sem_name, Sem_count);  
proc I(sema) is { SEMAPHORE <+ sema; };  
proc P(sema) is  
{ S <- when Sem_count > 0 in  
  (SEMAPHORE ijoin [Sem_name] in sema);  
  update SEMAPHORE change Sem_count <-  
    Sem_count - 1 using ([Sem_name] in sema);  
};  
proc V(sema) is  
{ update SEMAPHORE change Sem_count <-  
  Sem_count + 1 using ([Sem_name] in sema);  
};
```

## Semaphores

and here we use it as a *mutex*.

```
relation Sem1(Sem_name, Sem_count)←  
    {"sem1", 1});  
I(in Sem1);  
relation CommonRel(attr) ← {0});  
( P(in Sem1) --  
    LocalRel1 ← CommonRel --  
    update LocalRel1 change attr← attr+1 --  
    CommonRel ← LocalRel1 --  
    V(in Sem1)  
) ||  
( P(in Sem1) --  
    LocalRel2 ← CommonRel --  
    update LocalRel2 change attr← attr-1 --  
    CommonRel ← LocalRel2 --  
    V(in Sem1)  
);
```



## A Brief History of Concurrency Mechanisms

Coroutines SIMULA 67	Conway, 1963
Semaphores "THE" multiprogramming system (Conditional) Critical Region (replaced by Monitors)	Dijkstra, 1968
Monitors	Brinch Hansen, 1972 Hoare, 1972 Brinch Hansen, 1973 Hoare, 1974
Guarded Commands	Dijkstra, 1975
Path Expressions (limited, but cf. ALGOL 68)	Campbell & Haberman, 1974
CSP (Communicating Sequential Processes)	Hoare, 1978, 1985
DP (Distributed Processes)	Brinch Hansen, 1978

T. H. Merrett

©99/11