Spanning Tree

Root:

```
loop
  for m := 1 to δ do
    write r_mi := r_im := ⟨0, 0⟩
  end for
end loop
```

Others:

```
loop
  for m := 1 to δ do
    lr_mi := read(r_mi)
    FirstFound := false
    dist := 1 + min{lr_mi.dis | 1 ≤ m ≤ δ}

    for m := 1 to δ do
      if FirstFound and lr_mi.dis = dist - 1 then
        write r_im := ⟨1, dist⟩
        FirstFound := true
      else
        write r_im := ⟨0, dist⟩
      end if
    end for

  end for
end loop
```

Dijkstras mutual exclusion

P_1:

```
loop
  if x_1 = x_n then
    x_1 := (x_1 + 1) mod (n + 1)
  end if
end loop
```

P_i $(i \neq 1)$:

```
loop
  if x_i ≠ x_i-1 then
    x_i := x_i-1
  end if
end loop
```

Mutual exclusion for tree structure

Root:

loop
  read $lr_{1,i} := \text{read}(r_{1,i})$
  if $lr_{\delta,i} = r_{i,1}$ **then**
    write $r_{i,2} := (lr_{1,i} + 1) \bmod (4n + 5)$
  end if

  for $m := 2$ **to** δ **do**
    $lr_{m,i} := \text{read}(r_{m,i})$
    write $r_{i,m+1} := lr_{m,i}$
  end for
end loop

Others:

loop
  read $lr_{1,i} := \text{read}(r_{1,i})$
  if $lr_{1,i} \neq r_{i,2}$ **then**
    write $r_{i,2} := lr_{1,i}$
  end if

  for $m := 2$ **to** δ **do**
    $lr_{m,i} := \text{read}(r_{m,i})$
    write $r_{i,m+1} := lr_{m,i}$
  end for
end loop

Maximal matching

loop
  if $\text{pointer}_i = \text{null}$ **and** $(\exists\ P_j \in N(i) \mid \text{pointer}_j = i)$ **then**
    $\text{pointer}_i := j$
  end if

  if $\text{pointer}_i = \text{null}$ **and** $(\forall\ P_j \in N(i) \mid \text{pointer}_j \neq i)$ **and** $(\exists\ P_j \in N(i) \mid \text{pointer}_j = \text{null})$ **then**
    $\text{pointer}_i := j$
  end if

  if $\text{pointer}_i = j$ **and** $\text{pointer}_j = k$ **and** $k \neq i$ **then**
    $\text{pointer}_i := \text{null}$
  end if
end loop

Leader election in general graph

loop
  $\langle$candidate, distance$\rangle$:= $\langle$ID(i, 0)$\rangle$
  for all $P_j \in N(i)$ **do**
    $\langle leader_i[j], dist_i[j]\rangle$:= read($\langle leader_i, dist_i\rangle$)

    if (($dist_i[j] < N$) **and** ($leader_i[j] <$ candidate)) **or** (($leader_i[j] =$ candidate) **and** ($dist_i[j] <$ distance)) **then**
      $\langle$candidate, distance$\rangle$:= read($\langle leader_i, dist_i\rangle$)
    end if

  end for
  write $\langle leader_i, dist_i\rangle$:= $\langle$candidate, distance$\rangle$
end loop

Self-stabilizing counting

Root:

loop
  sum := 0
  for all $P_j \in$ children(i) **do**
    $lr_{j,i}$:= read($r_{j,i}$)
    sum := sum + $lr_{j,i}$.count
  end for
  $count_i$:= sum + 1
end loop

Others:

loop
  sum := 0
  for all $P_j \in$ children(i) **do**
    $lr_{j,i}$:= read($r_{j,i}$)
    sum := sum + $lr_{j,i}$.count
  end for
  $count_i$:= sum + 1
  write $r_{i,\ parent}$.count := $count_i$
end loop

Self-stabilizing naming

Root:

loop
  $\mathrm{ID_i} := 1$
  $\mathrm{sum} := 0$
  for all $\mathrm{P_j} \in \mathrm{children}(i)$ **do**
    $lr_{\mathrm{j,i}} := \mathrm{read}(\mathrm{r_{j,i}})$
    $lr_{\mathrm{j,i}} := \mathrm{ID_i} + \mathrm{sum} + 1$
    $\mathrm{sum} := \mathrm{sum}\ lr_{\mathrm{j,i}}.\mathrm{count}$
  end for
end loop

Others:

loop
  $\mathrm{sum} := 0$
  $lr_{\mathrm{parent,i}} := \mathrm{read}(\mathrm{r_{parent,i}})$
  $\mathrm{ID_i} := lr_{\mathrm{parent,i}}.\mathrm{identifier}$
  for all $\mathrm{P_j} \in \mathrm{children}(i)$ **do**
    $lr_{\mathrm{j,i}} := \mathrm{read}(\mathrm{r_{j,i}})$
    $lr_{\mathrm{j,i}} := \mathrm{ID_i} + \mathrm{sum} + 1$
    $\mathrm{sum} := \mathrm{sum}\ lr_{\mathrm{j,i}}.\mathrm{count}$
  end for
end loop

Digital clock synchronization (bounded)

Upon a pulse:

for all $\mathrm{P_j} \in \mathrm{N}(i)$ **do**
  $\mathrm{send}(j, \mathrm{clock_i})$
end for
$\mathrm{max} := \mathrm{clock_i}$
for all $\mathrm{P_j} \in \mathrm{N}(i)$ **do**
  $\mathrm{receive}(\mathrm{clock_j})$
  if $\mathrm{max} < \mathrm{clock_j}$ **then**
    $\mathrm{max} := \mathrm{clock_j}$
  end if
end for$\mathrm{clock_i} := (\mathrm{max} + 1) \bmod ((n+1)d+1)$

Self-stabilizing counting in non-rooted tree

```
loop
  for all P_j ∈ N(i) do
    lr_{j,i} := read(r_{j,i})
    sum_i := 0

    for all P_j ∈ N(i) do
      sum_j := 0

      for all P_k ∈ N(i) do
        if P_j ≠ P_k then
          sum_j := sum_j + lr_{ki}.count
        end if
      end for

    end for
    count_i[j] := sum_j + 1
    sum_i := sum_i + sum_j
    write r_{ij}.count := count_i[j]
  end for
  count_i := sum_i + 1
end loop
```

Update algorithm for P_i

```
loop
  Readset := ∅
  for all P_j ∈ N(i) do
    Readset_i := Readset_i ∪ read(Processors_j)
  end for
  Readset_i := Readset_i \\ ⟨i, ∗⟩
  Readset_i := Readset_i ++ ⟨∗, 1⟩
  Readset_i := Readset_i ∪ ⟨i, 0⟩
  for all P_j ∈ Processors(Readset_i) do
    Readset_i := Readset_i \\ NotMinDis(P_j, Readset_i)
  end for
  write Processors_i := ConPrefix(Readset_i)
end loop
```

Superstabilizing coloring

```
loop
  AColors := ∅
  GColors := ∅
  for m := 1 to δ do
    lr_m := read(r_m)
    AColors = Acolors ∪ lr_m.color
    if ID(m) > i then
      GColors = Gcolors ∪ lr_m.color
    end if
  end for
  if colors_i = ⊥ or color_i ∈ GColor then
    color_i = choose(\\ AColor)
  end if
  write color_i := color
end loop
```

Interrupt section

```
if receiver_ij and j > i then
  colors_i = ⊥
  write colors_i = ⊥
end if
```