3.2.1 Construction of NP_C^{sim}:

Replace each command of C by a PN program:

$x := x+1 \rightsquigarrow x := x+1;\ \bar{x} := \bar{x} - 1;$

$x := x-1 \rightsquigarrow x := x-1;\ \bar{x} := \bar{x} - 1;$

l: goto $l' \rightsquigarrow l$: goto $l';$

Translation of

l: if $x=0$ then goto l_{zero}
else goto $l_{nonzero}$;

Construct a PN program

$Test_n(x, l_{zero}, l_{nonzero})$

as follows.

Specification of $Test_n$:

- If $x=0$ then some execution of the program leads to l_{zero} and no execution leads to $l_{nonzero}$.
- If $1 \leq x \leq 2^n$ then some execution of the program leads to $l_{nonzero}$ and no execution leads to l_{zero}.
- The program $Test_n$ has no side-effects
  after any execution leading to l_{zero} or $l_{nonzero}$, no variable has changed its value.

Program $Test_n(x, l_{zero}, l_{nonzero})$:

$Test_n'(x, l_{cont}, l_{nonzero});$

l_{cont}: $Test_n'(\bar{x}, l_{zero}, l_{nonzero});$

Program $Test_n'$:

- Easier to design
- But has a side-effect

  after an execution leading to true
  the values of x and $\bar{x}$ are swapped.
- Swapping twice ensures we have no side-effect for $Test_n$
- Remaining specification coincides with $Test_n$.

Construction of $Test_n'$:

Idea:
- Since $x \leq 2^{2^n}$, testing $x = 0$ can be replaced by
  - decrementing x ($x := x - 1$) and if we succeed $\Rightarrow x > 0$
  - decreasing $\bar{x}$ by 2^{2^n} and if we succeed $\Rightarrow \bar{x} = 2^{2^n}$
    $\Rightarrow x = 0$
    (Invariant)
- If we guess incorrectly (decrement although $x = 0$, decrease although $\bar{x} < 2^{2^n}$), PN program blocks.

Problem:
- Decrease $\bar{x}$ by 2^{2^n}
- Done by function Dec_n below.

Specification of $Dec_n(s_n)$:
- Makes use of an auxiliary variable s_n (depends on n)
- If the initial value of s_n is $< 2^{2^n}$, Dec_n fails.
- If the value of s_n is $= 2^{2^n}$, then all executions of Dec_n that terminate with a return command have the effect

  $s_n := s_n - 2^{2^n}$ and $\bar{s}_n := \bar{s}_n + 2^{2^n}$.
- There are no side-effects
- All other executions fail.

Program Test$_n$(x, ℓ_{zero}, $\ell_{nonzero}$):

goto ℓ_{pos} or goto ℓ_{loop}; // non-deterministic choice

ℓ_{pos} : $x := x - 1$; $x := x + 1$; goto $\ell_{nonzero}$ // x is not zero

ℓ_{loop} : $\bar{x} := \bar{x} - 1$; $x := x + 1$; $s_n := s_n + 1$; $\bar{s}_n := \bar{s}_n - 1$; // move $\bar{x}$ to s_n, maintain the invariant

goto ℓ_{exit} or goto ℓ_{loop}; // continue moving $\bar{x}$ or decide for having moved everything

ℓ_{exit} : gosub Dec$_n$; goto ℓ_{zero};

// Once everything has been transferred, call Dec$_n$.

// If successful, continue at ℓ_{zero}.

Construction of Dec$_n$:

Proceed by induction on n:

Base case: $n = 0$

Dec$_0$ decreases by $2^{2^0} = 2^1 = 2$.

Subroutine Dec$_0$(s_0):

$s_0 := s_0 - 1$; $\bar{s}_0 := \bar{s}_0 + 1$;

$s_0 := s_0 - 1$; $\bar{s}_0 := \bar{s}_0 + 1$;

return;

Induction hypothesis: Assume Dec$_i$ is already known, and hence we can use Test$_i$.

Induction step:

- To construct Dec$_{i+1}$, note the following trick:

$$2^{2^{i+1}} = 2^{2 \cdot 2^i} = 2^{2^i + 2^i} = 2^{2^i} \cdot 2^{2^i}.$$

- Decrease by 2^{2^i} for 2^{2^i} many times.

Implementation:

Two nested loops

```
while {
    while {
        s_{i+1} := s_{i+1} - 1
    }
}
```

Execute 2^{2^i} times

Execute 2^{2^i} times.

↳ Loop variables counted from 2^{2^i} to 0

↳ Termination by testing loop variable for being 0

⇒ Test!

Subroutine Decision (s):

// initially $y_i = 2^{2^i} = z_i = \overline{s_i}$, $\overline{y_i} = 0 = \overline{z_i} = s_i$

initialization by NP_C^{init}.

```
lout:   y_i := y_i - 1; ȳ_i := ȳ_i + 1;
  lin:  z_i := z_i - 1; z̄_i := z̄_i + 1;
        s_{i+1} := s_{i+1} - 1; s̄_{i+1} := s̄_{i+1} + 1;
        Test'(z_i, linDone, lin);
linDone: Test'(y_i, loutDone, lout);
loutDone: return;
```

inner loop

outer loop

Note that Test' undoes the swapping of z_i and $\overline{z_i}$ when linDone is taken.

Similar for y_i and $\overline{y_i}$ when loutDone is taken in the second Test'.

3.2.2 Construction of NP_C^{init}

Variables that have to be initialized by NP_C^{init}:

- $x_1, \dots, x_\ell$ of counter program C with initial value 0
  $\overline{x_1}, \dots, \overline{x_\ell}$ with initial value 2^{2^n}
- $s_0, \dots, s_n$ have initial value 0
  $\overline{s_i}$ has initial value 2^{2^i} for all $0 \leq i \leq n$.
- y_i, z_i have initial value 2^{2^i} for all $0 \leq i \leq n-1$
  complement variables $\overline{y_0}, \overline{z_0}, \dots, \overline{y_{n-1}}, \overline{z_{n-1}}$ have initial value 0.

Program NP_C^{init}:

$inc_0(\overline{s_0}, y_0, z_0);$
$inc_1(\overline{s_1}, y_1, z_1);$
$\vdots$
$inc_{n-1}(\overline{s_{n-1}}, y_{n-1}, z_{n-1});$
$inc_n(\overline{s_n}, \overline{x_1}, \dots, \overline{x_\ell});$

Here, we invoke programs $inc_i(v_1, \dots, v_m)$ that yield

$v_1 := v_1 + 2^{2^i}$
$\vdots$
$v_m := v_m + 2^{2^i}$.

No side-effects.

Definition of programs inc_i:

Again by induction, similar to decrement

Program $inc_0(v_1, \dots, v_m)$:

$v_1 := v_1 + 1;\ v_1 := v_1 + 1;$
$\vdots$
$v_m := v_m + 1;\ v_m := v_m + 1;$

Program $\mathrm{Inc}_{i+1}(v_1, \ldots, v_m)$:

// Initially. $y_i := 2^{2^i} = z_i = \overline{s_i}$

$\overline{y_i} = 0 = \overline{z_i} = s_i$

// This assumption uses the inductive hypothesis. Note that NP_C^{init} calls Inc_0 to Inc_n in the required order.

outer: $y_i := y_i - 1$; $\overline{y_i} := \overline{y_i} + 1$;

inner: $z_i := z_i - 1$; $\overline{z_i} := \overline{z_i} + 1$;

$v_1 := v_1 + 1$;

$\vdots$

$v_m := v_m + 1$;

$\mathrm{Test}_i'(z_i, \mathrm{innerdone}, \mathrm{inner})$;

innerdone: $\mathrm{Test}_i'(y_i, \mathrm{outerdone}, \mathrm{outer})$;

outerdone: return;

3.3 Complexity

- Instance of Test_n for each conditional jump } constant length
- One instance of $\mathrm{Dec}_n, \ldots, \mathrm{Dec}_0$
- One instance of $\mathrm{Inc}_n, \ldots, \mathrm{Inc}_0$
  - ↳ Instances $\mathrm{Inc}_0, \ldots, \mathrm{Inc}_{n-1}$ have constant size
  - ↳ Instance Inc_n has size $O(n)$.

Together: $O(n)$

Petri net: $O(n)$.

□