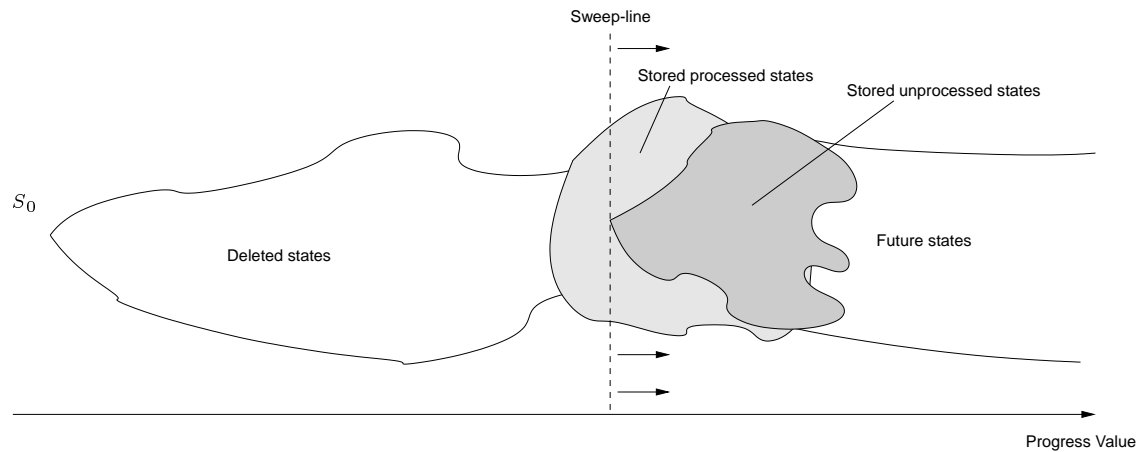


Sweep-Line State Space Exploration for Coloured Petri Nets



Thomas Mailund,

Department of Computer Science, University of Aarhus

Guy Gallasch, Lars Kristensen,

School of Electrical and Information Engineering

University of South Australia

— Sweep/CPN and the Sweep-Line Method —

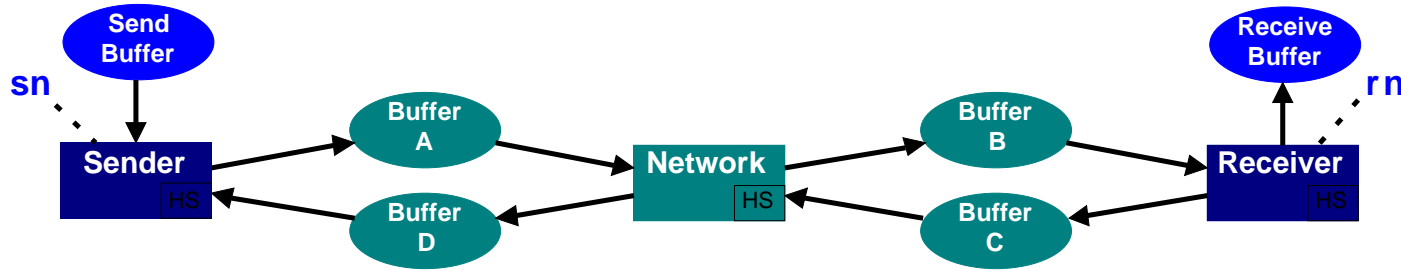
- Sweep/CPN is a library implementing the *sweep-line method* in Design/CPN.
- The sweep-line method is a state space exploration method that
 - exploits progress to
 - reclaim memorythus alleviating the state explosion problem.

Outline

- Basic Idea and Intuition: Communication Protocol.
- Progress Measures.
- Sweep-Line State Space Generation Algorithm.
- Sweep/CPN Library
- Conclusions and Future Work.

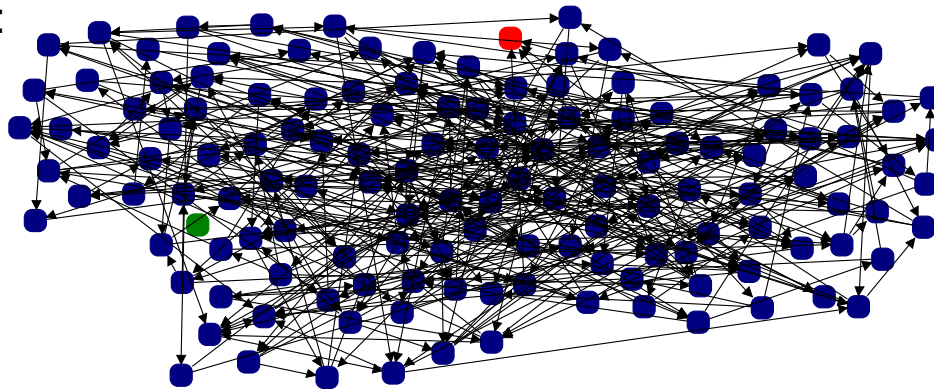
Example: Communication Protocol

- Packet transmission across an unreliable network.



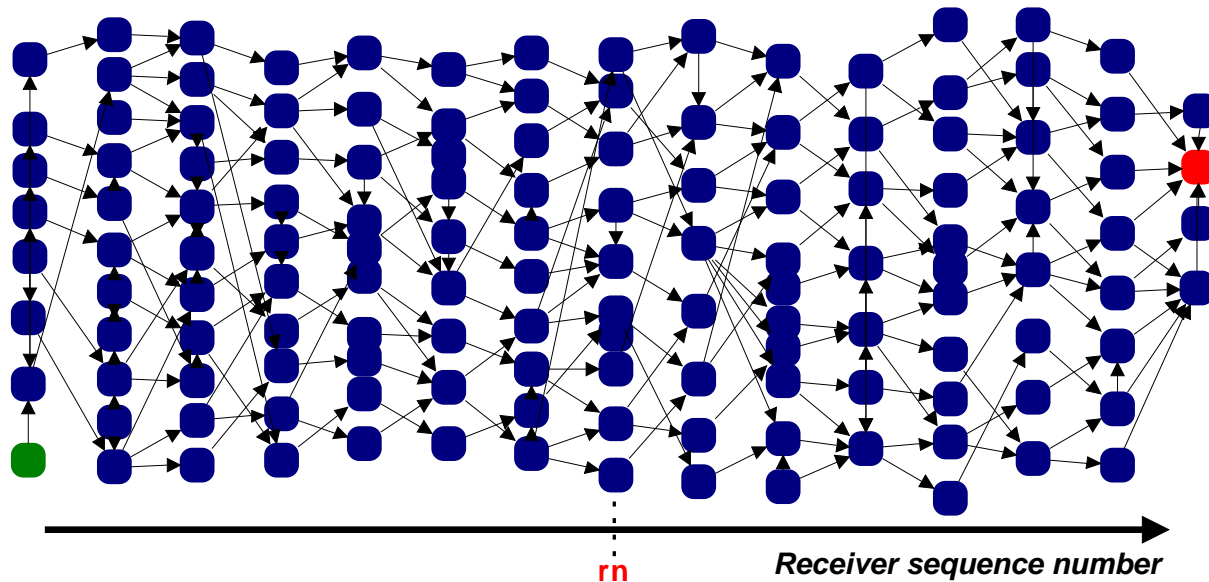
- A **stop-and-wait retransmission strategy** combined with **sequence numbers** can be used to recover from loss of packets and overtaking on the network:
 - Sender keeps sending a packet with a given sequence number sn until a matching **acknowledgement** is received.
 - Receiver increments its sequence number rn when the packet expected next arrives.

- Full state space:



Progress

- The communication protocol exhibits progress due to increasing sequence numbers in the receiver.
- Let $rn(s)$ denote the value of the receivers internal sequence number in a state s , and let s and s' be two states.
- If $rn(s') > rn(s)$, then s' represents a state of the protocol where the receiver has received more packets than in s .
- ☞ s' represent a state where the protocol has progressed further than in s .
- ☞ States can be ordered according to the receivers internal sequence number:



Conventional State Space Exploration Algorithm

The state space of a system can be explored based on a variant of the standard algorithm for traversal of a directed graph.

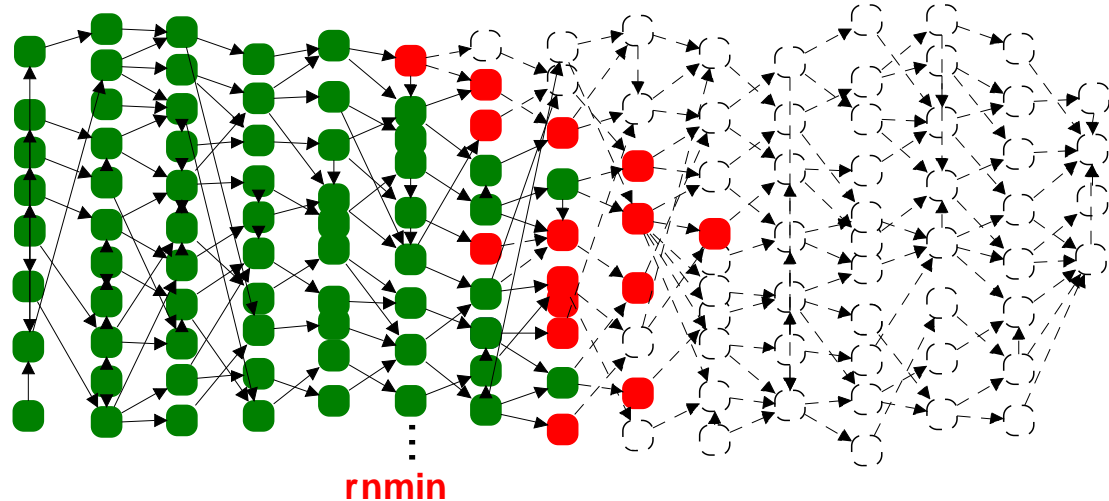
```
1: UNPROCESSED.ADD( $s_0$ )
2: NODES.ADD( $s_0$ )
3: while  $\neg$  UNPROCESSED.EMPTY() do
4:    $s \leftarrow$  UNPROCESSED.GETNEXT()
5:   for all ( $t, s'$ ) such that  $s \xrightarrow{t} s'$  do
6:     if  $\neg$  NODES.CONTAINS( $s'$ ) then
7:       NODES.ADD( $s'$ )
8:       UNPROCESSED.ADD( $s'$ )
9:     end if
10:  end for
11: end while
```

Snapshot of State Space Generation

```

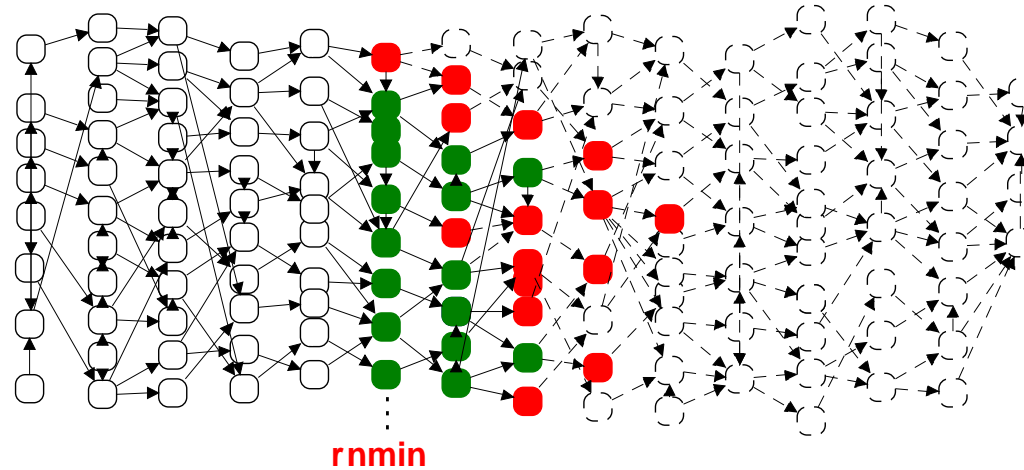
1: UNPROCESSED.ADD( $s_0$ )
2: NODES.ADD( $s_0$ )
3: while  $\neg$  UNPROCESSED.EMPTY() do
4:    $s \leftarrow$  UNPROCESSED.GETNEXT()
5:   for all ( $t, s'$ ) such that  $s \xrightarrow{t} s'$  do
6:     if  $\neg$  NODES.CONTAINS( $s'$ ) then
7:       NODES.ADD( $s'$ )
8:       UNPROCESSED.ADD( $s'$ )
9:     end if
10:  end for
11: end while

```

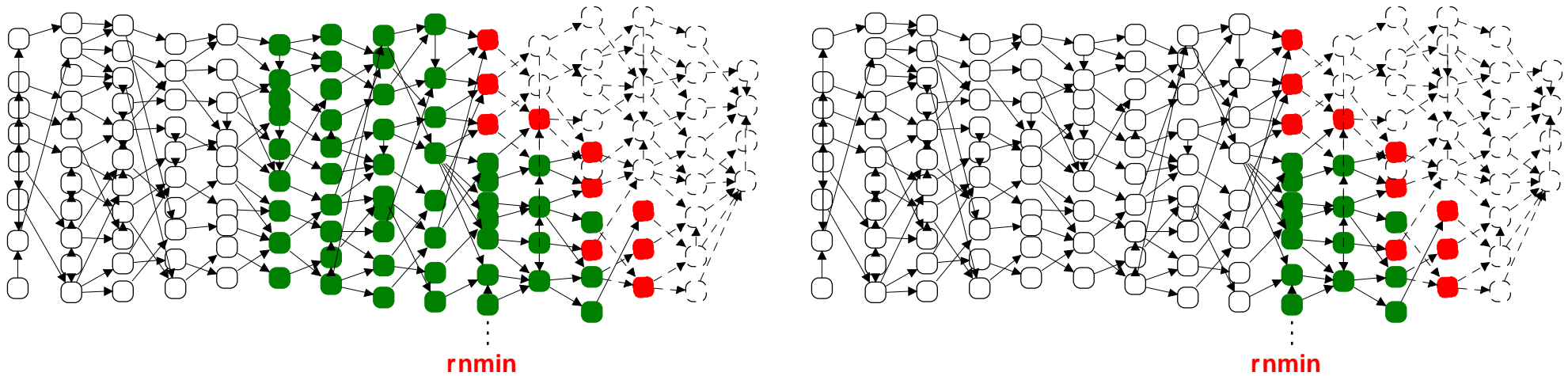


- A subset of the states stored in memory has been fully processed, i.e., their successor states have been calculated.
- A subset of the states stored in memory is unprocessed, i.e., their successor states have not yet been calculated.
- Fully processed states where the receiver sequence number is strictly less than **rmin** cannot be reached from the unprocessed states.
- ☞ These states can safely be deleted as they are not needed for comparison with newly generated states.

State Space Generation



□ Generation can now resume and deletion of states repeated later:



☞ This process can be viewed as a **sweep** through the state space, adding states in front of the **sweep-line** and deleting states behind the sweep-line.

Progress Measures

- Receiver sequence numbers is just one instance of the notion of progress underlying the sweep-line method.
- The precise notion of progress exploited by the sweep-line method is captured by the concept of **progress measures**.
- Consider a reachability graph $G = (V, E)$ where
 - $V = [M_0\rangle$
 - $E = \{(M, (t, b), M') \in V \times \text{BE} \times V \mid M[(t, b)\rangle M'\}$
- A **progress measure** on G is a tuple $\mathcal{P} = (O, \sqsubseteq, \psi)$ where:
 - O is a set of **progress values**.
 - \sqsubseteq is a **partial order** on O (reflexive, anti-symmetric, and transitive relation).
 - $\psi : V \rightarrow O$ is a **progress mapping** *monotone* in the sense:

$$\forall s, s' \in V : s[t, b\rangle s' \Rightarrow \psi(s) \sqsubseteq \psi(s')$$

Sweep-Line State Space Exploration Algorithm

- Derived from conventional state space generation algorithm by adding **garbage collection**, and a **consistency check** on the progress measure.

```
1: UNPROCESSED.ADD( $s_0$ )
2: NODES.ADD( $s_0$ )
3: while  $\neg$  UNPROCESSED.EMPTY() do
4:    $s \leftarrow$  UNPROCESSED.GETNEXT()
5:   for all ( $t, s'$ ) such that  $s \xrightarrow{t} s'$  do
6:     if  $\psi(s) \not\sqsubseteq \psi(s')$  then
7:       Stop("Progress measure rejected:", ( $s, t, s'$ ))
8:     end if
9:     if  $\neg$  NODES.CONTAINS( $s'$ ) then
10:      NODES.ADD( $s'$ )
11:      UNPROCESSED.ADD( $s'$ )
12:    end if
13:  end for
14:  GarbageCollect (Nodes)
15: end while
```

- Different schemes can be applied for implementing the garbage collection.

Stop-and-Wait Communication Protocol

□ Progress measure based on receiver sequence numbers:

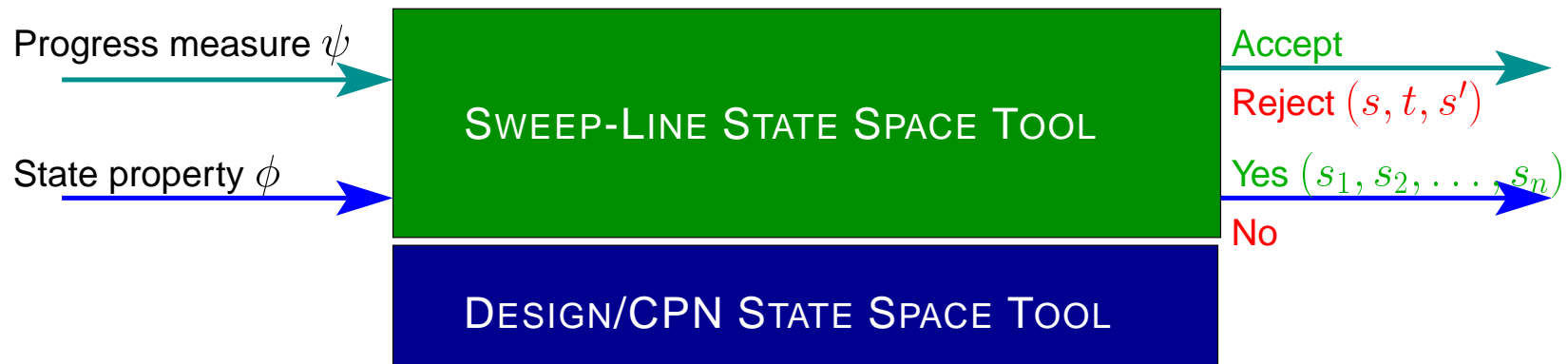
- Progress values $O = \mathbb{N}$
- Ordering on progress values $\sqsubseteq = \leq$
- Progress mapping $\psi(s) = rn(s)$

□ State space generation statistics (GC for each 2000 states):

Packets	Full State Spaces		Sweep-Line Method		Reduction	
	States	Time	States	Time	States	Time
10	2,576	0:00:01	2,001	0:00:01	22.3%	0.0%
50	13,416	0:00:05	2,280	0:00:04	83.0%	20.0%
100	26,966	0:00:14	2,280	0:00:07	91.5%	50.0%
200	54,066	0:00:42	2,280	0:00:14	95.8%	66.7%
500	135,366	0:03:11	2,287	0:00:38	98.3%	80.1%
1000	270,866	0:11:21	2,287	0:01:21	99.2%	88.1%
2000	541,866	0:47:10	2,287	0:03:05	99.6%	93.5%

Computer Tool Support

- An implementation of the sweep-line method has been implemented on top of the DESIGN/CPN STATE SPACE TOOL:



- Supports the sweep-line method for progress measures based on a total order on the progress values.
- UNPROCESSED implemented as a priority queue on progress values in order to move the sweep-line as fast as possible.
- Simple threshold based stop-and-copy garbage collection scheme: garbage collect whenever n new states have been added to the state space.

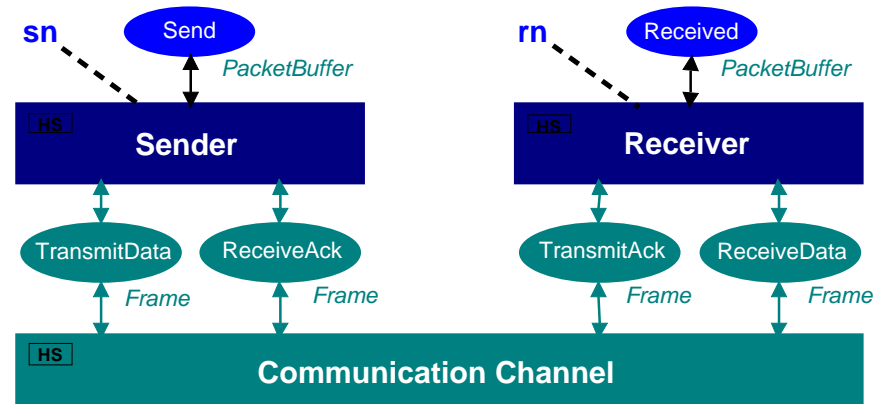
The Sweep/CPN Library

- The progress measure is provided by the user as a function

```
val progress_measure : SLMark → IntInf.int
```

- For the stop and wait protocol we can use the number of packets received using the function:

```
fun SWPM M =
  IntInf.fromInt (List.length
    (ms_to_col (SLMark.SWProtocol'Received 1 M)))
```



- This can then be used in a *sweep-line specification*

```
val SWSweepSpec = { PM = SWPM,      (* progress measure *)
                   GC = 100,        (* GC threshold *)
                   Logfile = NONE,  (* logging *)
                   Secs = 300 }     (* stop criteria *)
```

The Sweep/CPN Library

- ❑ The state space exploration in the Sweep/CPN library is conducted using the function `SL.Explore` parameterised with 19 parameters.
- ❑ For convenience the library provides three common-case wrappers: `SL.ExploreStates`, `SL.ExploreArcs`, and `SL.ExploreDead`.
- ❑ The convenience functions are on the same form:

```
val SL.ExploreStates : { Spec    : SweepSpec,  
                        Init    : 'a,  
                        Comb    : 'a * 'b → 'a,  
                        Eval    : SLMark → 'b,  
                        Pred    : SLMark → bool,  
                        Store   : SLMark → bool } → 'a
```

- ❑ Each state satisfying `Pred` is evaluated by `Eval` and combined with previous evaluations using `Comb`, with the initial value `Init`.
- ❑ The result of the function is the result of the last combination.
- ❑ The predicate `Store` is used to prevent garbage collection of certain states. States satisfying both `Store` and `Pred` are not garbage collected.

Standard Queries : Reachability Properties

- The library provides a number of standard query functions implemented using exploration functions.
- SLReachable is used for checking reachability of predicates

```
val SLReachable : SweepSpec * (SLMark → bool) * bool → bool
```

- SweepSpec is the sweep-line specification
- the function SLMark → **bool** is the predicate we are searching for
- the third parameter determines whether matched states should be stored for later analysis.

```
fun SLReachable (spec, markpred, keep) =  
  (SL.ExploreStates  
   { Spec      = spec,  
     Pred      = markpred,  
     Store     = (fn _ ⇒ keep),  
     Eval      = (fn _ ⇒ 1),  
     Comb      = (fn (a,b) ⇒ a+b),  
     Init      = 0}) > 0
```

Standard Queries : Reachability Properties

Example: Searching for duplicated packets:

```
fun DupPackets M =  
  let val packets = (ms_to_col (SLMark.Receiver'Received 1 M))  
  in (List.length (remdupl packets))  $\neq$  (List.length packets)  
  end
```

```
SLQuery.SLReachable (SWSweepSpec, DupPackets, false)
```

Standard Queries : Integer Bounds

- SLBestUpperInteger and SLBestLowerInteger for checking integer bounds.

```
SLUpperInteger : SweepSpec → (SLMark → int) → int
```

- SweepSpec is the sweep-line specification
- the function SLMark → int derives an integer value from a marking
- the return value is the best upper bound of the integer values

```
fun SLBestUpperInteger (spec, evalfun) =  
  (SL.ExploreStates  
   { Spec      = spec,  
     Pred      = (fn _ ⇒ true),  
     Store     = (fn _ ⇒ false),  
     Eval      = evalfun,  
     Comb      = (fn (a,b) ⇒ Int.max(a,b)),  
     Init      = 0})
```

- Example: Maximal number of packets on TransmitData

```
fun TransmitDataCount M = mssize (SLMark.Sender'TransmitData 1 M)
```

```
SLQuery.SLBestUpperInteger (SWSweepSpec, TransmitDataCount)
```

Standard Queries : Terminal States

- SLListDeadMarkings can be used for finding terminal (dead) states.

```
SLListDeadMarkings : SweepSpec → Node list
```

- SweepSpec is the sweep-line specification
- the return value is a list of all terminal nodes

```
fun SLListDeadMarkings spec =  
  SL.ExploreDeadStates  
    { Spec      = spec,  
      Pred      = (fn _ ⇒ true),  
      Store     = (fn _ ⇒ true),  
      Eval      = GetNodeNo,  
      Comb      = (fn (a,b) ⇒ b::a),  
      Init      = [] }
```

- Example:

```
SLQuery.SLListDeadMarkings SWSweepSpec
```

Conclusions and Future Work

- ❑ The sweep-line method is a on-the-fly state space exploration method exploiting a state based notion of progress.
- ❑ Implemented in Design/CPN as the Sweep/CPN library.
- ❑ Application areas: transport protocols and transaction protocols, Timed Coloured Petri Nets, ...

- ❑ Still some open questions:
 - Counter example generation/debugging information
 - Behavioral properties that can be analysed (CTL checking, LTL checking, ...)
 - Combinations with other reduction techniques (symmetry/equivalence reduction, partial-order methods, ...)