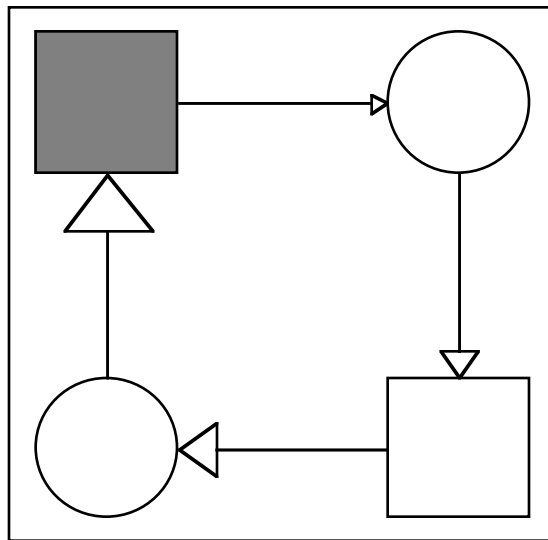


Design/CPN

Computer Tool for Coloured Petri Nets



Computer Science Department
University of Aarhus, Denmark

Ny Munkegade, Building 540
DK-8000 Aarhus C, Denmark

Phone: +45 89 42 31 88

Telefax: +45 89 42 32 55

E-mail: designCPN-support@daimi.aau.dk

URL: <http://www.daimi.aau.dk/designCPN/>

A copy of these slides can be downloaded from:
<http://www.daimi.aau.dk/designCPN/slides/>

Functionality of Design/CPN

Design/CPN supports:

- *Construction and editing of large, hierarchical CP-nets.*
- *Syntax check.*
- *Interactive simulation.*
- *Automatic simulation.*
- *Verification by means of state spaces.*

Design/CPN is available on *two different platforms*:

- *Sun Sparc with Solaris.*
- *Macintosh with Mac OS.*

History of Design/CPN

Originally developed by *Meta Software*, Cambridge MA, USA – in close cooperation with the *CPN group at University of Aarhus*, Denmark.

- *First version* was finished in 1989.
- More than *40 man-years* have been used for the design and implementation.
- *Price tag* used to be \$ 24,000.

From 1996 the distribution, maintenance and further development is done by the *CPN group at University of Aarhus*, Denmark.

- Now the tool is distribution *free of charge to all kinds of users* (including commercial companies).
- After less than two months version 3.0 is used by *100 organisations from 30 countries* all over the world.

Design/CPN WWW pages

We maintain an *elaborated set of WWW pages* offering a lot of information about *Design/CPN* and *Coloured Petri Nets*:

- General Information
 - What is Design/CPN?
 - History of Design/CPN
 - What is a Coloured Petri Net?
 - Standard ML
- More Detailed Information
 - Overview of Design/CPN, Overhead Slides
 - Hardware/Software Requirements
 - How to Become a Design/CPN User
 - What is New in Version 3.0?
- Technical Information
 - On-line Tutorial and Manuals
 - Tips'n'Tricks, FAQ, Common Problems
 - Known Bugs in Design/CPN and the Manuals
 - Future Plans for Design/CPN
 - Proposals from Users.
- Design/CPN E-mail List
 - What is the Design/CPN E-mail List?
 - How to use the Design/CPN E-mail List
 - View List of Recent Postings
- Examples and Libraries
 - Examples of Design/CPN Models
 - Design/CPN Libraries
 - Industrial use of Design/CPN.

Automatic simulation

In an automatic simulation the user does *not* intend to follow the ongoing simulation:

- The simulation is *fast* – several hundred steps per second.
- The simulator *chooses* between conflicting transitions and bindings (by means of a *random number generator*).
- The user specifies some *stop criteria*, which determine the duration of the simulation.
- When the simulation stops the graphics of the CP-net is *updated*.
- Then the user can inspect all details of the graphics, e.g., the *enabling* and the *marking*.
- Automatic simulations can be *mixed* with interactive simulations – and there are many *intermediate forms*.

To find out what happens during an *automatic simulation* the user has a number of choices.

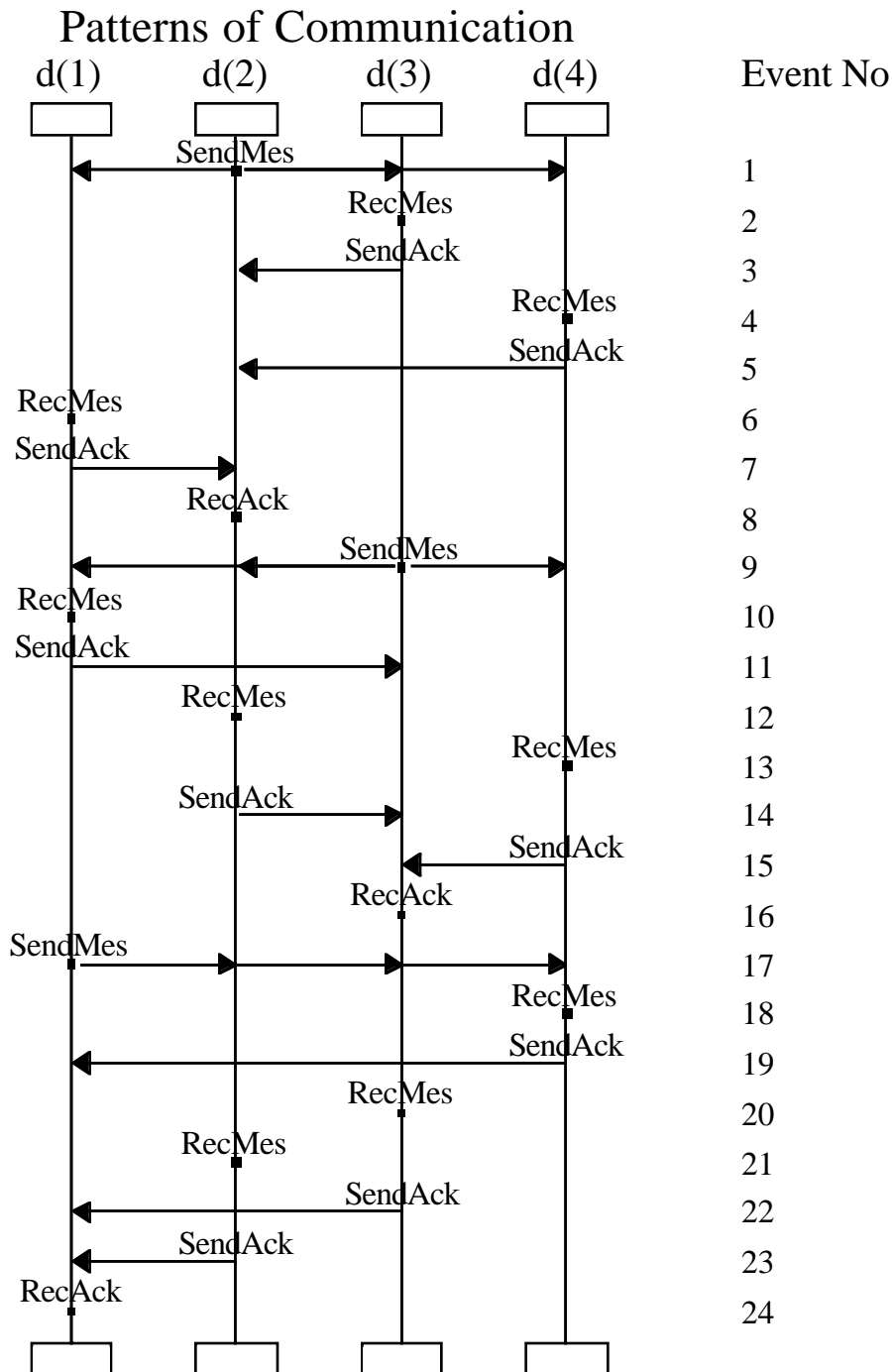
Simulation report

A *simulation report* shows the *transitions* which have occurred. The user determines whether he also wants to see the *bindings*.

1	SendPack@(1:Top#1)	{n = 1, p = "Modellin"}
2	TranPack@(1:Top#1)	{n = 1, p = "Modellin", r = 6, s = 8}
3	SendPack@(1:Top#1)	{n = 1, p = "Modellin"}
4	TranPack@(1:Top#1)	{n = 1, p = "Modellin", r = 3, s = 8}
5	RecPack@(1:Top#1)	{k = 1, n = 1, p = "Modellin", str = ""}
6	SendPack@(1:Top#1)	{n = 1, p = "Modellin"}
7	SendPack@(1:Top#1)	{n = 1, p = "Modellin"}
8	TranAck@(1:Top#1)	{n = 2, r = 2, s = 8}
9	TranPack@(1:Top#1)	{n = 1, p = "Modellin", r = 7, s = 8}
10	RecPack@(1:Top#1)	{k = 2, n = 1, p = "Modellin", str = "Modellin"}
11	RecAck@(1:Top#1)	{k = 1, n = 2}
12	RecPack@(1:Top#1)	{k = 2, n = 1, p = "Modellin", str = "Modellin"}
13	TranAck@(1:Top#1)	{n = 2, r = 7, s = 8}
14	TranPack@(1:Top#1)	{n = 1, p = "Modellin", r = 6, s = 8}
15	RecAck@(1:Top#1)	{k = 2, n = 2}
16	SendPack@(1:Top#1)	{n = 2, p = "g and An"}
17	TranAck@(1:Top#1)	{n = 2, r = 6, s = 8}
18	RecPack@(1:Top#1)	{k = 2, n = 1, p = "Modellin", str = "Modellin"}
19	RecAck@(1:Top#1)	{k = 2, n = 2}
20	SendPack@(1:Top#1)	{n = 2, p = "g and An"}

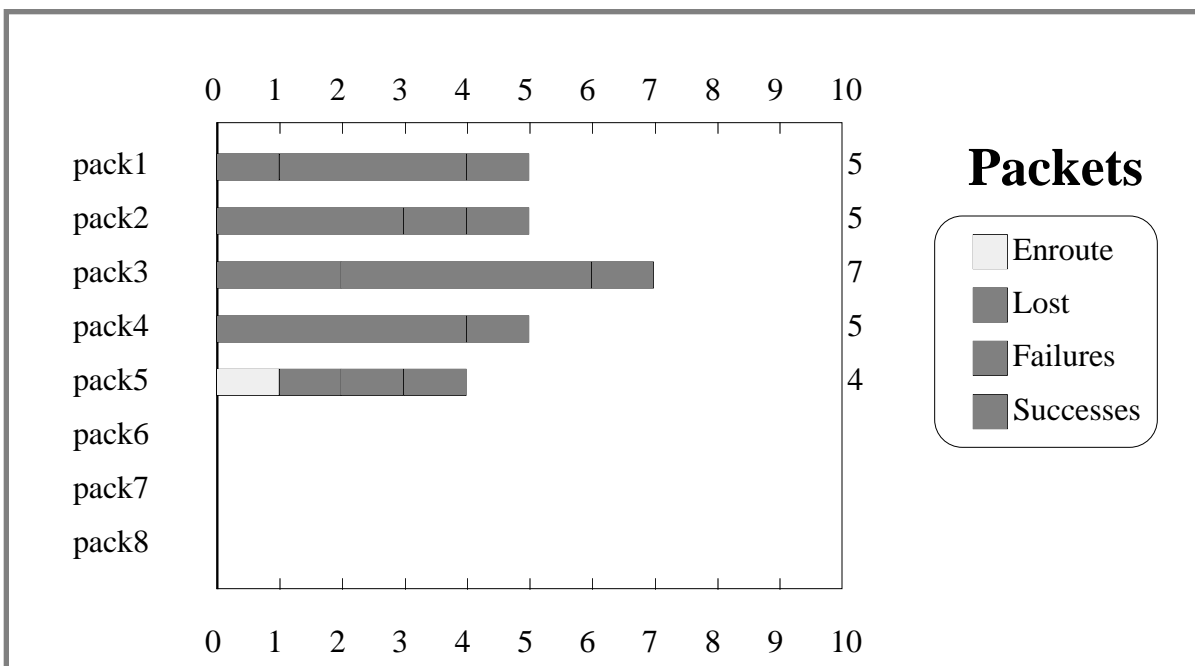
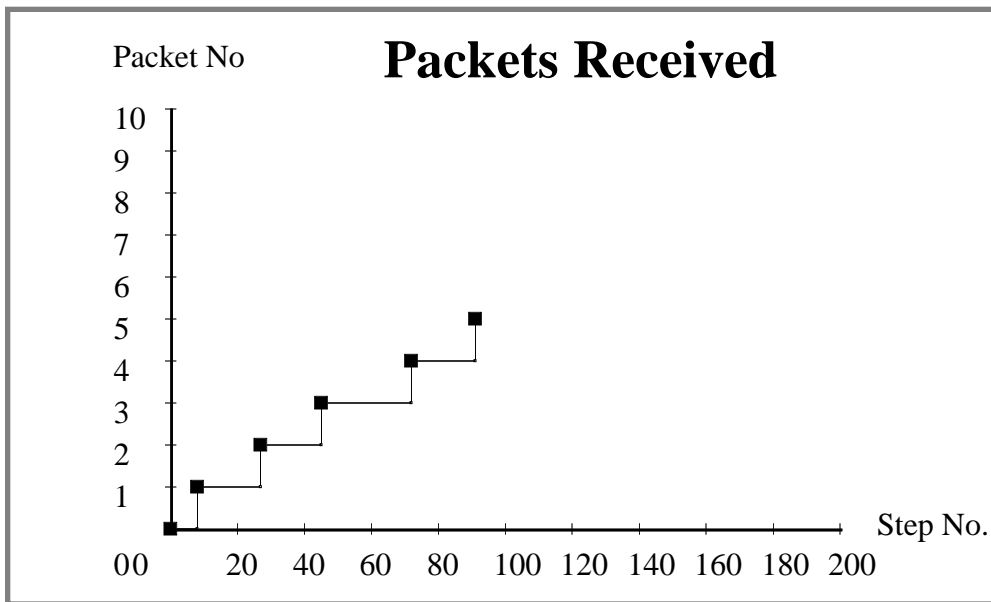
Event traces

Event traces provides a *graphical overview* of the *key events* in a simulation.



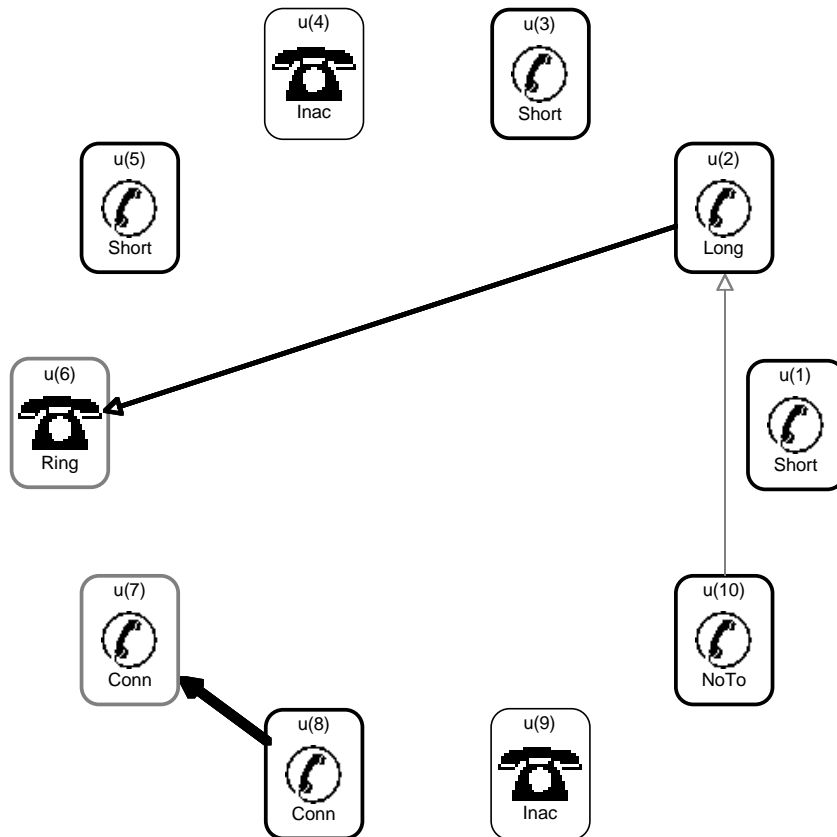
Business charts

It is possible to use different kinds of *business charts*: The *user* determines how often the charts are updated.



Other kinds of graphics

It is also possible to make more *customised* kinds of graphical feedback, e.g.:



Each transition may have a *code segment*, i.e., a piece of *sequential code* which is used, e.g., to:

- Update *charts* and other kinds of *graphics*.
- Read and write *files*.

Standard ML

Declarations, net inscriptions and code segments are specified in a *programming language* called *Standard ML*.

- *Strongly typed, functional* language.
- *Data types* can be:
 - *Atomic* (integers, reals, strings, booleans and enumerations).
 - *Structured* (products, records, unions, lists and subsets).
- Arbitrary complex *functions* and *operations* can be defined (polymorphism and overloading).
- Computational power of expressions are equivalent to *lambda calculus* (and hence to Turing machines).

Developed at *Edinburgh University* by Robin Milner and his group.

- Standard ML is well-known, well-tested and very general. Several *text books* are available.

Performance analysis

CP-nets can be extended with a *time concept*. This means that the *same language* can be used to investigate:

- *Logical correctness*.
Desired functionality, absence of deadlocks, etc.
- *Performance*.
Remove bottlenecks. Predict mean waiting times and average throughput. Compare different strategies.

In a timed CP-net each token carries a *colour* (data value) and a *time stamp* (telling when it can be used).

- *Time stamps* may depend upon *colour values* and are specified by *ML expressions*. Hence the user can specify, e.g., *fixed* delays, *interval* delays and *exponential* delays.

State space analysis

State spaces are also known as reachability graphs, reachability trees and occurrence graphs. They are *directed graphs* with:

- A *node* for each *reachable marking*.
- An *arc* for each *occurring transition*.

State spaces are *powerful* and *easy* to use.

- They can be used to investigate *all behavioural properties* (except concurrency).
- *Construction* of state spaces and *analysis* of state spaces can be *automated*.

The main drawback is the *state explosion*, i.e., the size of the state space.

- The *present version* handles state spaces with more than 100,000 nodes and 500,000 arcs.
- Future versions are expected to handle *much larger* state spaces.
- Moreover, it is sometimes possible to use *equivalence relations* or *symmetries* to construct much more *condensed* state spaces – *without losing information*.

State space report for protocol

Contains a lot of *useful information* about the *behaviour* of the CP-net.

- It is excellent for *locating errors* or to *increase our confidence* in the correctness of the system.

Statistics

Occurrence Graph

Nodes: 4298
 Arcs: 15887
 Secs: 53
 Status: Full

Scc Graph

Nodes: 2406
 Arcs: 11677
 Secs: 17

Boundedness Properties

Upper Integer Bounds

A: 1
 B: 2
 C: 1
 D: 2
 NextRec: 1
 NextSend: 1
 RA: 1
 RP: 1
 Received: 1
 Send: 4

Lower Integer Bounds

A: 0
 B: 0
 C: 0
 D: 0
 NextRec: 1
 NextSend: 1
 RA: 1
 RP: 1
 Received: 1
 Send: 4

State space report (continued)

Upper Multi-set Bounds

A: $1^{\setminus}(1, \text{"Coloured"}) + 1^{\setminus}(2, \text{" Petri N"}) +$
 $1^{\setminus}(3, \text{"ets#####"}) + 1^{\setminus}(4, \text{"#####"})$
 B: $2^{\setminus}(1, \text{"Coloured"}) + 2^{\setminus}(2, \text{" Petri N"}) +$
 $2^{\setminus}(3, \text{"ets#####"}) + 2^{\setminus}(4, \text{"#####"})$
 C: $1^{\setminus}2 + 1^{\setminus}3 + 1^{\setminus}4 + 1^{\setminus}5$
 D: $2^{\setminus}2 + 2^{\setminus}3 + 2^{\setminus}4 + 2^{\setminus}5$
 NextRec: $1^{\setminus}1 + 1^{\setminus}2 + 1^{\setminus}3 + 1^{\setminus}4 + 1^{\setminus}5$
 NextSend: $1^{\setminus}1 + 1^{\setminus}2 + 1^{\setminus}3 + 1^{\setminus}4 + 1^{\setminus}5$
 RA: $1^{\setminus}1$
 RP: $1^{\setminus}1$
 Received: $1^{\setminus}"" + 1^{\setminus}\text{"Coloured"} + 1^{\setminus}\text{"Coloured Petri N"} +$
 $1^{\setminus}\text{"Coloured Petri Nets#####"}$
 Send: $1^{\setminus}(1, \text{"Coloured"}) + 1^{\setminus}(2, \text{" Petri N"}) +$
 $1^{\setminus}(3, \text{"ets#####"}) + 1^{\setminus}(4, \text{"#####"})$

Lower Multi-set Bounds

A: empty
 B: empty
 C: empty
 D: empty
 NextRec: empty
 NextSend: empty
 RA: $1^{\setminus}1$
 RP: $1^{\setminus}1$
 Received: empty
 Send: $1^{\setminus}(1, \text{"Coloured"}) + 1^{\setminus}(2, \text{" Petri N"}) +$
 $1^{\setminus}(3, \text{"ets#####"}) + 1^{\setminus}(4, \text{"#####"})$

State space report (continued)

Home Properties

Home Markings: 1 [452]

Liveness Properties

Dead Markings: 1 [452]

Live Transitions: None

Fairness Properties

Send Packet: Impartial

Transmit Packet: Impartial

Receive Packet: No Fairness

Transmit Acknow: No Fairness

Receive Acknow: No Fairness

Investigation of dead marking

We ask the system to display marking number 452.

```
452
NextSend = 5
NextRec = 5
Received = "Coloured Petri Nets#####"
```

452
8:0

Marking no. 452 is the *desired final marking* (all packets has been received in the correct order)

Marking 452 is *dead*:

- This implies that the protocol is *partially correct* (if execution stops it stops in the desired final marking).

Marking 452 is a *home marking*:

- This implies that we *always have a chance to finish correctly* (it is impossible to reach a state from which we cannot reach the desired final marking).

Investigation of shortest path

We ask the system to calculate one of the *shortest paths* from the initial marking to the dead marking:

```
val path =  
NodesInPath(1, 452);
```

```
> val path =  
[1, 2, 3, 5, 8, 11, 15, 20, 27, 38, 50,  
64, 80, 102, 133, 164, 199, 243,  
301, 375, 452] : Node list
```

```
Length(path);
```

```
> 20 : int
```

The calculated path contains *20 transitions*.

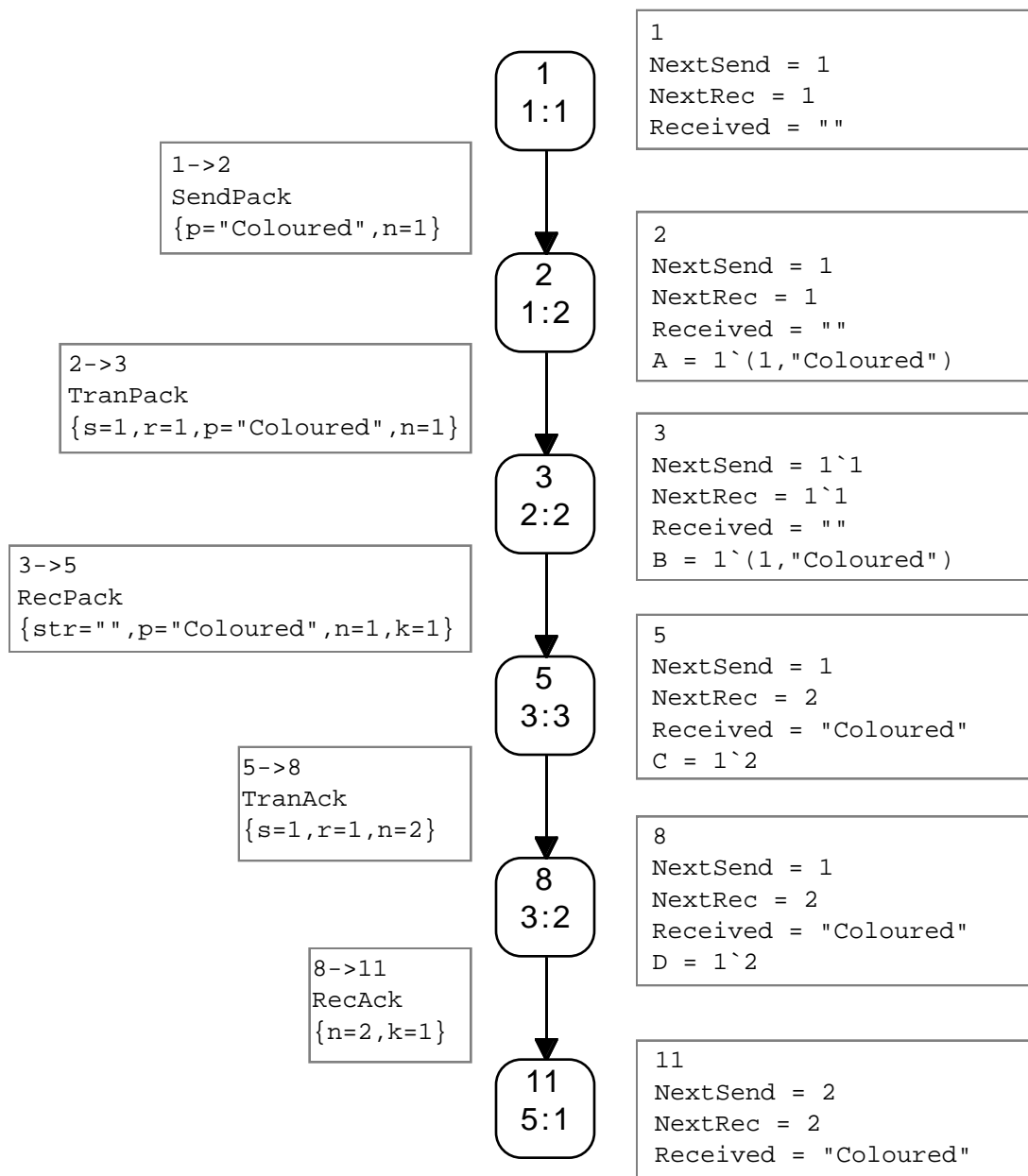
- This is as expected because there are *4 packets* which each needs *5 transitions* to occur.

Drawing of shortest path

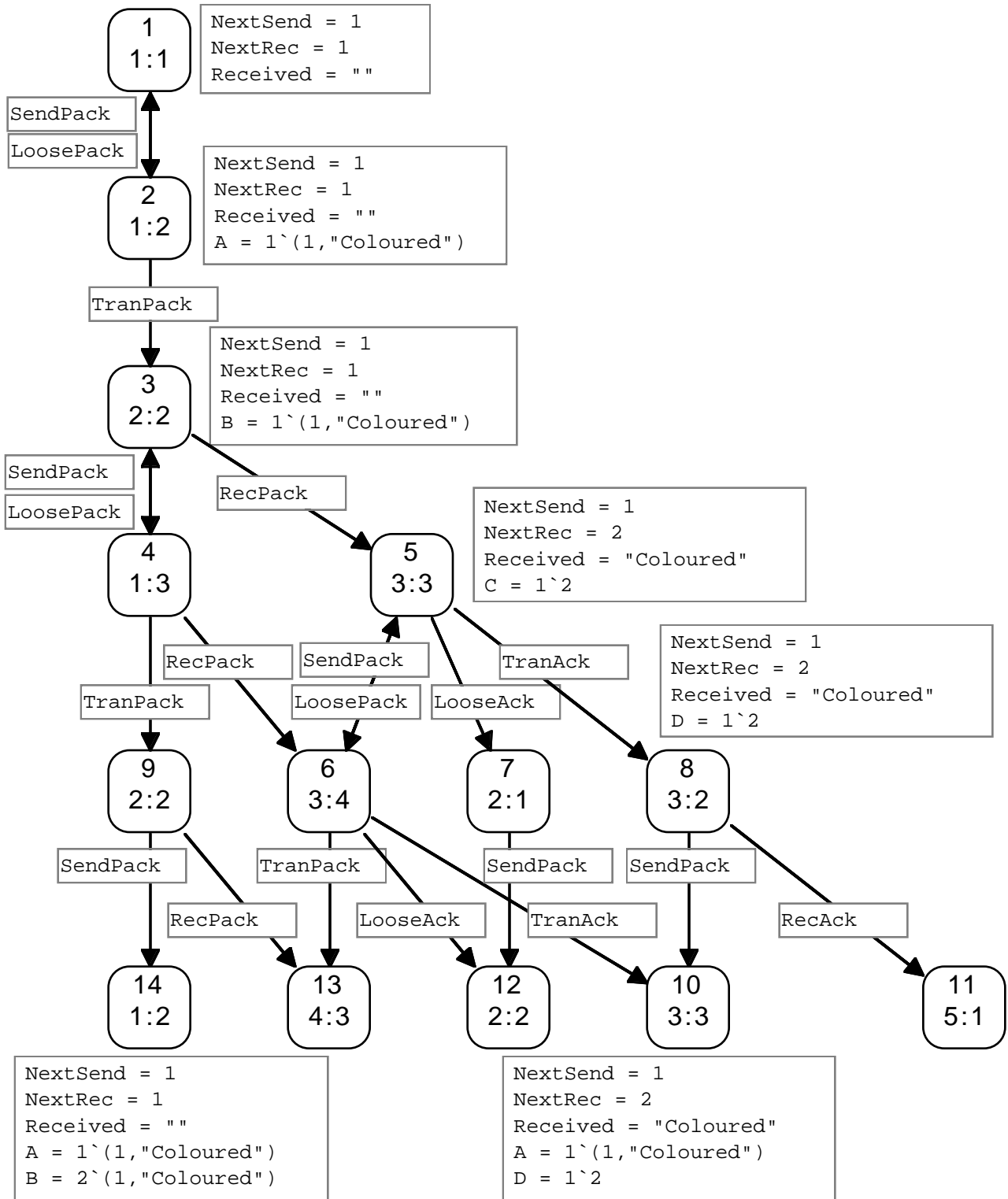
We ask the system to draw the *first six nodes* in the calculated shortest path:

```
DisplayNodePath; [1,2,3,5,8,11];
```

```
> () : unit
```



Drawing of subgraph



Non-standard queries

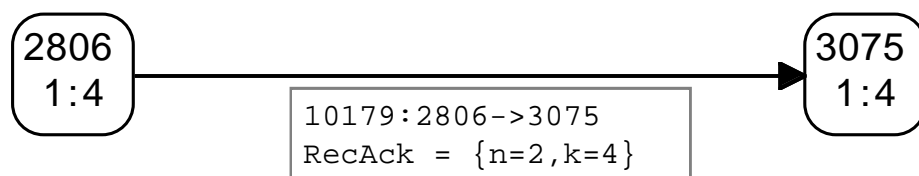
We ask the system to search *all arcs* in the *entire graph* and return the *first 10 arcs* where *NextSend* has a *larger* value in the *source marking* than it has in the *destination marking*.

```
PredArcs
  (EntireGraph,
   fn a => ((ms_to_col(Mark.NextSend 1
                      (SourceNode a))) >
            (ms_to_col(Mark.NextSend 1
                      (DestNode a))))),
   10)
end;
```

```
>[10179,10167,10165,10159,10055,10052,10035,
10031,10019,10007] : Arc list
```

```
NextSend = 4
NextRec = 5
Received = "Coloured Petri
Nets#####"
A = 1^(4,"#####")
B = 2^(4,"#####")
C = 1^5
D = 1^2+ 1^5
```

```
NextSend = 2
NextRec = 5
Received = "Coloured
Petri Nets#####"
A = 1^(4,"#####")
B = 2^(4,"#####")
C = 1^5
D = 1^5
```



Design/CPN libraries

A number of *libraries* extends the functionality of the Design/CPN tool:

- *Animation* by Mimic/CPN.
- *Event traces* diagrams.
- *Temporal logic* for state spaces.
- State spaces with *equivalence classes*.

Design/CPN examples

We also offer a large number of CPN examples with *detailed explanations*.

- Can be studied *without* prior knowledge of CP-nets and *without* using the tool.

For more information on *libraries* and *CP-net examples*, see the Design/CPN WWW pages:

<http://www.daimi.aau.dk/designCPN/>

Future plans for Design/CPN

We are *currently* working on the following *improvements*:

- *New simulation engine* (1000 times faster).
- *Syntax directed interface* for declarations (much easier to use and with better browsing facilities).
- *More incremental* syntax check and code generation (to decrease the turn-around time).
- *Textual interchange format* (to share models with other tools).

Later we will also *extend Design/CPN* to support:

- *Place invariant analysis*.
- *Different extensions* of Coloured Petri Nets (capacities, inhibitor arcs, FIFO places, communication channels, etc.).

For more information on *future plans*, see the Design/CPN WWW pages:

<http://www.daimi.aau.dk/designCPN/>