Coloured Petri Nets

© Kurt Jensen
Department of Computer Science
University of Aarhus, Denmark

kjensen@daimi.au.dk
www.daimi.au.dk/~kjensen/

This slide set can be downloaded from:
http://www.daimi.au.dk/CPnets/slides/

TOOLS
• editing
• simulation
• verification

THEORY
• models
• basic concepts
• analysis methods

PRACTICAL USE
• specification
• validation
• verification
• implementation
What is a Coloured Petri Net?

- **Modelling language** for systems where **synchronisation**, **communication**, and **resource sharing** are important.

- Combination of **Petri Nets** and **Programming Language**.
  - **Control structures**, **synchronisation**, **communication**, and **resource sharing** are described by **Petri Nets**.
  - **Data** and **data manipulations** are described by **functional programming language**.

- CPN models are validated by means of **simulation** and verified by means of **state spaces** and **place invariants**.

- Coloured Petri Nets is developed at **University of Aarhus, Denmark** over the last 25 years.
Why do we make models?

- **We make models** to:
  - *Learn new things* about a system.
  - To check that the system design has certain *expected properties*.

- **CPN models** are *dynamic*:
  - They can be *executed* on a *computer*.
  - This allows us to play and *investigate* different *scenarios*.
Overview of talk

Modelling

- Basic language
  - syntax
  - semantics
- Extensions
  - modules
  - time
- Tool support
  - editing
  - simulation

Analysis

- State spaces
  - full
  - symmetries
  - equivalence classes
  - sweep-line
- Place invariants
  - check of invariants
  - use of invariants
Simple protocol

Coloured Petri Nets
24/01/2005
Simple protocol

Transitions

Sender

Network

Receiver

1. (1,"Modelling")+
2. (2,"g and An")+
3. (3,"alysis b")+
4. (4,"y Means ")+
5. (5,"of Colou")+
6. (6,"red Petr")+
7. (7,"i Nets##")+
8. (6,"####")

INTxDATA

Send

(n,p)

if Ok(s,r)
then 1°(n,p)
else empty

INTxDATA

A

(n,p)

B

(n,p)

if n=k
and also
p<>stop
then str^p
else str

str

INT

NextSend

k

n

INT

n

Int_0..10

RA

D

n

INT

n

INT

C

k

if n=k
then k+1
else k

if n=k
then k+1
else k

Coloured Petri Nets
24/01/2005
Simple protocol

Sender

Receive Acknow.

Type (colour set)

Place

(n,p)

Send Packet

n

NextSend

k

Receive Packet

Network

Transmit Acknow.

if Ok(s,r) then 1^1\text{.n}\text{.p} INT\text{xDATA}
else empty

Transmit Packet

s

Receive Packet

if n=k and also p<>stop then str^p
else str

Transmit Packet

B

(n,p)

str

Transmit Packet

D

(n,p)

INT\text{xDATA}

Transmit Acknow.

n

Transmit Acknow.

if Ok(s,r) then 1^1\text{n}
else empty

Receive Acknow.

C

n

Transmit Acknow.

Receive Acknow.

n

Transmit Acknow.

Receive Acknow.

n

Transmit Acknow.

Receive Acknow.

n

Transmit Acknow.

Receive Acknow.

n

Transmit Acknow.

Receive Acknow.

n

Transmit Acknow.

Receive Acknow.

n

Transmit Acknow.

Receive Acknow.
Marking of Send

INTxDATA

Send

Number of tokens

Multi-set of token colours

1 ` (1,"Modellin") +
1 ` (2,"g and An") +
1 ` (3,"alysis b") +
1 ` (4,"y Means ") +
1 ` (5,"of Colou") +
1 ` (6,"red Petr") +
1 ` (7,"i Nets##") +
1 ` (8,"########")
Simple protocol

Arc Inscriptions

Sender

Network

Receiver

24/01/2005

Coloured Petri Nets
Simple protocol

Sender

Receive Acknow.

NextSend

Send Packet

(n,p)

INTxDATA

Transmit Packet

(n,p)

INTxDATA

Receive Packet

Received

DATA

if Ok(s,r) then 1*(n,p) else empty

if n=k and also p<>stop then str else str

if n=k then k+1 else k

if n=k then k+1 else k

if n=k then k+1 else k

if n=k then k+1 else k
Simple protocol

Buffer places Interface

Sender

Receive Acknow.

Send Packet

NextSend

Network

Transmit Acknow.

Receive Packet

Receiver

if Ok(s,r) then 1*(n,p) else empty

if n=k and also p<>stop then str^; else str

if n=k then k+1 else k

if n=k then k+1 else k

Received DATA

s

str

k

Rec

if n=k then k+1 else k

INTxDATA

(n,p)

INTxDATA

(n,p)

INTxDATA

(n,p)

1"Modellin"

1"g and An"

1"alysis b"

1"y Means"

1"of Colou"

1"red Petr"

1"i Nets#

1"#

Sender

Receive Acknow.

Transmit Packet

Buffer places Interface

Network

Receive Packet

Receiver

if Ok(s,r) then 1*n else empty

INT

n

INT

n

INT

n

INT

(n,p)

(n,p)

(n,p)
Simple protocol

Packets to be sent

Sender
- Send Packet
  - INTxDATA
  - (n,p)
  - Send
  - (n,p)
- NextSend
  - INT
  - k
- Receive Acknow.
  - INT
  - n
- Transmit Acknow.
  - INT
  - n
  - if Ok(s,r) then 1*(n,p)
  - INTxDATA
  - (n,p)

Network
- Transmit
  - INT
  - n
  - if Ok(s,r) then 1*n
  - else empty

Receiver
- Receive Packet
  - INT
  - n
  - if n=k then k+1
  - else k
  - if n=k and also p<>stop then str^p
  - else str

Coloured Petri Nets
24/01/2005
Simple protocol

Coloured Petri Nets
24/01/2005
Simple protocol

Sender

Receive Acknow.

NextSend

INT

k

n

Receive Packet

Counter

Transmit Packet

Transmit Acknow.

if Ok(s,r) then 1\'(n,p) else empty

INT\times DATA

INT\times DATA

(n,p)

A

(n,p)

B

(n,p)

if n=k and also p<>stop then str^p else str

if n=k then k+1 else k

NextRec

INT

RA

8

Int_0..10

8

Int_0..10

INT

n

if Ok(s,r) then 1\'n else empty

INT

s

s

INT

Sender

Network

Receiver

if n=k then k+1
Simple protocol

Send Packet
(intxDATA
(n,p)
Send

NextSend
(int
(k,n)

Receive Acknow.

Transmit Acknow.

Data received

Receive Packet

Sender

Network

Receiver

if Ok(s,r)
then 1*(n,p) intxDATA

if n=k
and also
p<>stop
then str^f
else str

if n=k
then k+1
else k

if n=k
then k+1
else k

Coloured Petri Nets
24/01/2005
Simple protocol
Send packet

- The binding $<n=1,p="Modellin">$ is enabled.

- When the binding occurs it adds a token to place A.

- This represents that the packet (1,"Modellin") is sent to the network.

- The packet is not removed from place Send and the NextSend counter is not changed.
Simple protocol

Coloured Petri Nets
24/01/2005
All *enabled bindings* are on the form:

- `<n=1,p= "Modellin",s=8,r=...>`
- where $r \in 1..10$
Loss of packets

- The function $Ok(s, r)$ checks whether $r \leq s$.
  - For $r \in 1..8$, $Ok(s, r) = true$. The token is moved from A to B. This means that the packet is successfully transmitted over the network.
  - For $r \in 9..10$, $Ok(s, r) = false$. No token is added to B. This means that the packet is lost.
- The CPN simulator makes random choices between bindings: 80% chance for successful transfer.
Receive packet

- The number of the incoming packet $n$ and the number of the expected packet $k$ are compared.

```
if n=k and also p<>stop then str^p else str
```

```
Receive Packet

INTxDATA

B

1\(1\text{``Modellin''}\)

if n=k then k+1 else k

Received

DATA

if n=k then k+1 else k

NextRec

INT

C

INT

Receive Packet

1\(1\)

Received

if n=k then k+1 else k

```

Coloured Petri Nets
24/01/2005
Correct packet number

- The data in the packet is *concatenated* to the data already received.
- The *NextRec* counter is *increased by one*.
- An *acknowledgement* is sent. It contains the number of the *next packet* the receiver wants to get.
Wrong packet number

- The data in the packet is \textit{ignored}.
- The NextRec counter is \textit{unchanged}.
- An \textit{acknowledgement} is sent. It contains the number of the \textit{next packet} the receiver wants to get.

Received

1`"Modelling and An"

if n=k and also p<>stop then str^p else str

Receive Packet

1`3

if n=k then k+1 else k

NextRec

1`3

if n=k then k+1 else k

Packet

1`3

(n,p)

1`2("g and An")

B

1

1

C

acknowledgement is sent. It contains the number of the next packet the receiver wants to get.
Simple protocol
Transmit acknowledgement

This transition works in a similar way as Transmit Packet.

The marking of RA determines the success rate.
Simple protocol
Receive acknowledgement

- When an acknowledgement arrives to the **Sender** it is used to update the **NextSend** counter.
  - In this case the counter value becomes 2, and hence the **Sender** will begin to send **packet number 2**.
Intermediate state

- **Receiver** expects packet no. 6.
- **Sender** is still sending packet no. 5.
- **Acknowledgement** requesting packet no. 6 is arriving.
- Then **NextSend** is updated and **Sender** will start sending packet no. 6.
CP-nets has a formal definition

- The existence of a *formal definition* is important:
  - Basis for *simulation*, i.e., execution of the CP-net.
  - Basis for the *formal verification* methods (e.g., state spaces and place invariants).
  - Without the formal definition, it would have been impossible to obtain a *sound* net class.

- It is *not necessary* for a *user* to know the formal definition of CP-nets:
  - Correct *syntax* is checked by the CPN editor.
  - Correct *semantics* is guaranteed by the CPN simulator and the CPN verification tools.
High-level Petri nets

The relationship between CP-nets and ordinary Petri nets (PT-nets) is analogous to the relationship between high-level programming languages and assembly code.

- In theory, the two levels have exactly the same computational power.
- In practice, high-level languages have much more modelling power – because they have better structuring facilities, e.g., types and modules.

Several other kinds of high-level Petri Nets exist. However, Coloured Petri Nets is the most widely used – in particular for practical work.
Overview of talk

Modelling
- Basic language
  - syntax
  - semantics
- Extensions
  - modules
  - time
- Tool support
  - editing
  - simulation

Analysis
- State spaces
  - full
  - symmetries
  - equivalence classes
  - sweep-line
- Place invariants
  - check of invariants
  - use of invariants
CP-nets are used for large systems

- A CPN model consists of a number of *modules*.
  - Also called *subnets* or *pages*.
  - Well-defined *interfaces* and clear *semantics*.

- A typical *industrial application* of CP-nets has:
  - 10-200 modules.
  - 50-1000 places and transitions.
  - 10-200 types.

- Industrial applications of this size would be *totally impossible* without:
  - Data types and token values.
  - Modules.
  - Tool support.
Three different modules

Sender

Receiver

Network

Port places are used to exchange tokens between modules.
Abstract view

Protocol

- **Substitution transitions** refer to *modules*.
- **Socket places** are related to *port places*.
Modules can be reused
Protocol with multiple receivers

Sender

Sender

Network

Receiver

Receive

Acknow.

Transmit

Packet

Receive

Packet

Transmit

Acknow.

Transmit

Acknow.

Received

DATA

NexRec

if n=k
then k+1
else k

if n=k
and also
p<>stop
then str^p
else str

if Ok(s,r)
then 1*(n,1)
else empty

if Ok(s,r)
then 1*(n,2)
else empty

if Ok(s,r1)
thenn*(n,p)
elsolempty

if Ok(s,r2)
thenn*(n,p)
elsolempty

if n=1
then 1*(n,1)
else empty

if n=1
then 1*(n,2)
else empty

if n=1
then 1*(n,p)
else empty

INTxDATA

INTxDATA

INTxDATA

INTxDATA
Transmit packets

- Packets are *broadcasted* to the two receivers.
  - *Some* of the packets may *be lost.*
Transmit acknowledgments
Receive acknowledgments

- The sender follows the *slowest* receiver.
Hierarchical descriptions

- We use *modules* to *structure large* and *complex descriptions*.

- Modules allow us to *hide details* that we do not want to consider at a certain *level of abstraction*.

- Modules have *well-defined interfaces*, consisting of *socket* and *port places*, through which the modules *exchange tokens* with each other.

- Modules can be *reused*. 
Another solution

- **Multiple receivers** may also be modelled by adding a *new component* to the *token colours*.

- Similar changes for *Transmit Packet* and *Transmit Acknowledgment*.
Protocol for ISDN network

- Most abstract view of the system.
Overview of user site

Coloured Petri Nets
24/01/2005
This module describes the actions that can happen when the user site is in state U8.

The node shapes have a meaning in SDL.
Typical transition

Status Enquiry message received in state U8.

- **Guard** checks:
  - Message is a *Status Enquiry* message.
  - *Call Reference* is correct (i.e., matches the one in the *User State* token at place U8).

- A *Status message* is sent to the *network site*. It tells that the user site is in state U8.

Coloured Petri Nets
24/01/2005
Some modules are used many times

- **43 modules** with more than **100 instances**.
- **Entire model** was made in only **3 man-weeks**.
Time analysis

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

  - **Logical correctness.** Desired functionality, absence of deadlocks, etc.
  - **Performance.** How fast is the system and how many resources are used.
How to add time

- *Time* has been added to *Petri net models* in many different ways – typically by specifying *delays on places or transitions*.

- *Time stamp* determines *when* the token can be used, i.e., *consumed by a transition*.
  - *Delays* can be *fixed*.
  - Determined by an *arbitrary distribution*. 
A timed CP-net for protocol

Retransmission delay

Fixed delay

Variable delay
Application areas

Protocols and Networks
- Intelligent Networks at Deutsche Telekom
- IEEE 802.6 Configuration Control at Telstra Research Labs
- Allocation Policies in the Fieldbus Protocol in Japan
- ISDN Services at Telstra Research Laboratories
- Protocol for an Audio/Video System at Bang & Olufsen
- TCP Protocols at Hewlett-Packard
- Local Area Network at University of Las Palmas
- UPC Algorithms in ATM Networks at University of Aarhus
- BRI Protocol in ISDN Networks
- Network Management System at RC International A/S
- Interprocess Communication in Pool IDA at King’s College

Software
- Mobile Phones at Nokia
- Bank Transactions & Interconnect Fabric at Hewlett-Packard
- Mutual Exclusion Algorithm at University of Aarhus
- Distributed Program Execution at University of Aarhus
- Internet Cache at the Hungarian Academy of Science
- Electronic Funds Transfer in the US
- Document Storage System at Bull AG
- ADA Program at Draper Laboratories
Control of Systems
- Security and Access Control Systems at Dalcotech A/S
- Mechatronic Systems in Cars at Peugeot-Citroën in France
- European Train Control System in Germany
- Flowmeter System at Danfoss
- Traffic Signals in Brazil
- Chemical Production in Germany
- Model Train System at University of Kiel

Hardware
- Superscalar Processor Architectures at Univ. of Newcastle
- VLSI Chip in the US
- Arbiter Cascade at Meta Software Corp.

Military Systems
- Military Communications Gateway in Australia
- Influence Nets for the US Air Force
- Missile Simulator in Australia
- Naval Command and Control System in Canada

Other Systems
- Bank Courier Network at Shawmut National Coop.
- Nuclear Waste Management Programme in the US
Overview of talk

Modelling
- Basic language
  - syntax
  - semantics
- Extensions
  - modules
  - time
- Tool support
  - editing
  - simulation

Analysis
- State spaces
  - full
  - symmetries
  - equivalence classes
  - sweep-line
- Place invariants
  - check of invariants
  - use of invariants
Computer tools

- **Design/CPN** was developed in the late 80'ies and early 90'ies.
  - Until recently, it was the *most widely used* Petri net package.
  - Used by *1000 different organisations* in more than *60 countries* – including *200 commercial companies*.
- **CPN Tools** is the *next generation* of tool support for Coloured Petri Nets.
  - It has now *replaced Design/CPN* with *1250 users* in more than *75 countries*.
  - Development *started in 1999* and a total of *25 man-years* have been used.
  - Development *continues* with an *expected effort of 5 man-years* per year.
The functionality of the two tools is the same:

- **Editing** and **syntax check** of CP-nets.
- **Interactive** and **automatic simulation**.
- **Construction** and **analysis** of **state spaces**.
- **Communication** with other tools.
- Simulation based **performance analysis**.
- **Graphical animation** of simulation results.
What is new in CPN Tools?

- *Windows XP*. Later versions will also support *Linux*.
- *On-the-fly, incremental syntax check*.
- *Much more efficient simulation engine* in particular for:
  - Models with *many tokens*.
  - *Timed* models.
- *New user interface* with a number of state-of-the-art interaction mechanisms:
  - Possible to have a *mouse in each hand*.
  - Tool glasses, floating palettes and circular marking menus.
Types, arc expressions and guards are specified in *Standard ML*, which is a strongly typed, functional programming language developed by Robin Milner.

**Data types** can be:
- *Atomic* (integers, strings, booleans and enumerations).
- *Structured* (products, records, unions, lists, and subsets).

Arbitrary complex *functions* and *operations* can be defined (e.g., using polymorphism).

Standard ML is well-known, well-tested and very general. Several *text books* are available.
We want to move the selected part to a new module.

This is done by a single operation.
Abstract view

Sockets (interface)

Database

Substitution transition

Name of new module

Update and Send Messages

Received all Acknowledgments

Acknowledged

Sent

Database

Database

New#2

Coloured Petri Nets
24/01/2005
Interfaces and detailed relationship between the two modules are automatically determined by the CPN editor.
Simulation of CP-nets

- When a syntactical correct CPN diagram has been constructed, the CPN tool generates the necessary code to perform simulations.
  - Calculates whether the individual transitions and bindings are enabled.
  - Calculates the effect of occurring transitions and bindings.
- The syntax check and code generation are incremental. Hence it is fast to make small changes to the CPN diagram.
- We distinguish between two kinds of simulations:
  - In an interactive simulation the user is in control, but most of the work is done by the system.
  - In an automatic simulation the system does all the work.
Simulation results are shown directly on the CP-net.

Transitions are chosen by the user or the simulator.

User can observe all details and set breakpoints.
Automatic simulation

- The user does not intend to follow the simulation:
  - Simulation can be very fast - several thousand steps per second.
  - 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.

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

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"\}
Message sequence chart

Sender

Network

Receiver

Sender

SendPack: (1,"Modellin")

SendPack: (1,"Modellin")

SendPack: (1,"Modellin")

Network

TranPack: (1,"Modellin")

TranPack: (1,"Modellin")

TranPack: (1,"Modellin")

Receiver

RecPack: (1,"Modellin")

SendAck: 2

Ack Lost: 2
Business charts

Packets Received

Packets

- pack1
- pack2
- pack3
- pack4
- pack5
- pack6

Successes
Failures
Lost
Enroute

Packets

Step No.

Packet No

0 1 2 3 4 5 6 7 8 9 10

20 40 60 80 100 120 140 160 180 200

0 1 2 3 4 5 6 7 8 9 10

5 5 7 5 4 7 8 9 10

5 5 7 5 4 7 8 9 10

012345678910
Automatic code generation

- CPN models are often used to *specify* and *validate* new software.

- It is also possible to *implement* the software by *automatic code generation.*
  
  - This method has been applied to develop a system for *access control* to buildings.
  
  - The source code for the final implementation was generated *automatically* from the CPN specification - by extracting parts of the *Standard ML code* used by the CPN simulator.

  - The approach is only adequate for systems that are *not time critical* and systems that are produced in *small numbers.*
Overview of talk

Modelling
- Basic language
  - syntax
  - semantics
- Extensions
  - modules
  - time
- Tool support
  - editing
  - simulation

Analysis
- State spaces
  - full
  - symmetries
  - equivalence classes
  - sweep-line
- Place invariants
  - check of invariants
  - use of invariants
State spaces

- A state space is a directed graph with:
  - A node for each reachable marking (i.e., state).
  - An arc for each occurring binding element.
State space tool

- State spaces are often very large.
- The **CPN state space tool** allows the user to:
  - **Generate** state spaces.
  - **Analyse** state spaces to obtain information about the *behaviour* of the modelled system.
- **Generation** is totally *automatic* while **analysis** is *automatic* or *semi-automatic* (based on queries from the user).
State space report

- Generation of the *state space report* takes often only a *few seconds*.
  - The report contains a lot of useful information about the *behaviour* of the CP-net.
  - The report is excellent for *locating errors* or to *increase our confidence* in the *correctness* of the system.
To obtain a finite state space, we:
- Only have 4 packets.
- Limit the number of tokens on A, B, C, and D.
- Binary choice between success and failure.
State space report for protocol

<table>
<thead>
<tr>
<th>Occurrence Graph Statistics</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Nodes</td>
<td>428</td>
</tr>
<tr>
<td>Arcs</td>
<td>1130</td>
</tr>
<tr>
<td>Secs</td>
<td>0</td>
</tr>
<tr>
<td>Status</td>
<td>Full</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Scc Graph Statistics</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Nodes</td>
<td>182</td>
</tr>
<tr>
<td>Arcs</td>
<td>673</td>
</tr>
<tr>
<td>Secs</td>
<td>0</td>
</tr>
</tbody>
</table>
**Integer bounds**

- A, B, C, D, Limit: 0-2
- NextSend, NextRec, Received: 1
- Send: 4

*Integer bounds* tell the *maximal* and *minimal number of tokens* on the individual places.
Integer bounds

Coloured Petri Nets
24/01/2005
Upper multi-set bounds

A, B: 2\(1, "Modellin"\) + 2\(2, "g and An"\) + 2\(3, "alysis##"\) + 2\(4, "########"\)

C, D: 2\(2\) + 2\(3\) + 2\(4\) + 2\(5\)

Limit: 2\(e\)

NextSend, NextRec: 1\(1\) + 1\(2\) + 1\(3\) + 1\(4\) + 1\(5\)

Received: 1\"" + 1\"" Modellin" + 1\""Modelling and An" + 1\""Modelling and Analysis##"

Send: 1\(1, "Modellin"\) + 1\(2, "g and An"\) + 1\(3, "alysis##"\) + 1\(4, "########"\)
Home and liveness properties

<table>
<thead>
<tr>
<th>Home Properties</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Home Markings:</td>
<td>[235]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Liveness Properties</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Dead Markings:</td>
<td>[235]</td>
</tr>
<tr>
<td>Live Transitions:</td>
<td>None</td>
</tr>
</tbody>
</table>

Marking no. 235 is the desired final marking where all packets have been received in correct order.

NextSend = 5  
NextRec = 5  
Received = "Modelling and Analysis##"
Investigation of dead marking

- Marking 235 is the *only dead marking*.
  - This implies that the protocol is *partially correct* (if execution stops it stops in the desired final marking).

- Marking 235 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).
Fairness properties

- **Send Packet**: Impartial
- **Transmit Packet**: Impartial
- **Receive Packet**: No Fairness
- **Transmit Acknow**: No Fairness
- **Receive Acknow**: No Fairness

*Fairness properties* tell *how often* the individual transitions *occur.*
Investigation of shortest path

- We want to find one of the shortest paths from the initial marking to the dead marking.

\[
\text{val path = NodesInPath}(1,235); \\
\text{Length(path)};
\]

\[
> \text{val path = [1,2,3,4,6,8,10,15,20,27,50,64,80,102,133,164,179,192,201,215,235] : Node list} \\
> 20 : \text{int}
\]
We want to investigate the beginning of the calculated shortest path.

DisplayNodePath [1,2,3,4,6,8];
Draw more complex subgraph
Non-standard queries

Can the NextSend counter be decreased?
Query in Standard ML

PredAllArcs
(fn a => ((ms_to_col(Mark.NextSend 1 (SourceNode a))) >
(ms_to_col(Mark.NextSend 1 (DestNode a)))));

>[973,951,934,921,920,895,894,845,844,843,818,817,
  753,729,663,648,587,573,567,517,499,497,429,
  428,360,310,271,233] : Arc list

Yes!
Counter example

DisplayArcs [973];  > () : unit

NextSend = 4
NextRec = 5
Received = "Modelling and Analysis##"
B = 1 `(4,"########")
D = 1 `3

NextSend = 3
NextRec = 5
Received = "Modelling and Analysis##"
B = 1 `(4,"########")

RecAck = {n=3,k=4}
Improved protocol

Sender

Send Packet

(n,p)

Send

Received

DATA

if n=k
and also
p<>stop
then str^p
else str

Network

Transmit Packet

if Ok(s,r)
then 1`(n,p)
else empty

Transmit Acknow.

Receive Packet

if n=k
then k+1
else k

Receive Acknow.

if Ok(s,r)
then 1`n
else empty

Receive

INTxDATA

(n,p)

 NextSend

INT

K

max(n,k)

Sender

Receive Acknow.

INT

n

Network

Transmit Packet

(n,p)

Transmit Acknow.

INT

s

s

1

NextRec

INT

n

k

1

NextRec

INT

n

k

Coloured Petri Nets
24/01/2005
Temporal logic

- It is also possible to make *state space queries* by means of a CTL-like *temporal logic*.
  - *States*.
  - *Transitions*.
  - *Binding elements*. 
State spaces - pro/contra

- State spaces are powerful and easy to use.
  - Construction and analysis can be automated.
  - No need to know the mathematics behind the analysis methods.

- The main drawback is the state explosion, i.e., the size of the state space.
  - The present version of our tool handles graphs with one million states.
  - For many systems this is not sufficient.
## Statistics – full state spaces

<table>
<thead>
<tr>
<th>Limit:</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Original</td>
<td>33</td>
<td>428</td>
<td>3,329</td>
<td>18,520</td>
<td>82,260</td>
</tr>
<tr>
<td></td>
<td>Max</td>
<td>33</td>
<td>293</td>
<td>1,829</td>
<td>9,025</td>
<td>37,477</td>
</tr>
<tr>
<td></td>
<td>Ratio</td>
<td>1.0</td>
<td>1.46</td>
<td>1.82</td>
<td>2.05</td>
<td>2.19</td>
</tr>
<tr>
<td>Nodes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Original</td>
<td>44</td>
<td>1,130</td>
<td>12,825</td>
<td>91,220</td>
<td>483,562</td>
</tr>
<tr>
<td></td>
<td>Max</td>
<td>44</td>
<td>764</td>
<td>6,860</td>
<td>43,124</td>
<td>213,902</td>
</tr>
<tr>
<td></td>
<td>Ratio</td>
<td>1.0</td>
<td>1.48</td>
<td>1.87</td>
<td>2.12</td>
<td>2.26</td>
</tr>
<tr>
<td>Arcs</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Original</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>41</td>
<td>560</td>
</tr>
<tr>
<td></td>
<td>Max</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>16</td>
<td>153</td>
</tr>
<tr>
<td></td>
<td>Ratio</td>
<td>-----</td>
<td>-----</td>
<td>1.5</td>
<td>2.56</td>
<td>3.66</td>
</tr>
<tr>
<td>Secs</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

Intel Pentium III, 1GHz, 1 GB RAM

Coloured Petri Nets
24/01/2005
Condensed state spaces

- Fortunately, it is sometimes possible to construct much more compact state spaces — without loosing information.

- This is done by exploiting:
  - Symmetries in the modelled system.
  - Other kinds of equivalent behaviour.
  - Progress measure.
  - Concurrency between events.
Protocol with multiple receivers
State space for three receivers

- The red nodes are equivalent (or symmetrical).
- They also have equivalent:
  - direct successors,
  - enabled binding elements.
Condensed state space for three receivers

21 nodes instead of 62 nodes
Symmetries

◆ A symmetry is a function $\phi$ that maps:
  - markings into equivalent markings,
  - binding elements into equivalent binding elements.

◆ A symmetry specification is a set of functions $\Phi \subseteq \mathcal{M} \cup \mathcal{BE} \rightarrow \mathcal{M} \cup \mathcal{BE}$ such that:
  - $\forall \phi \in \Phi: (\phi \big| M) \in \mathcal{M} \rightarrow \mathcal{M} \land (\phi \big| \mathcal{BE}) \in \mathcal{BE} \rightarrow \mathcal{BE}$.
  - $(\Phi, \circ)$ is an algebraic group.

Each element of $\Phi$ is called a symmetry.
Equivalent markings

- Two markings $M$ and $M^*$ are equivalent iff there exist a symmetry $\phi$ that maps $M^*$ into $M$:
  \[ M \approx_M M^* \iff \exists \phi \in \Phi: M = \phi(M^*). \]

- Two binding elements $b$ and $b^*$ are equivalent iff there exist a symmetry $\phi$ that maps $b^*$ into $b$:
  \[ M \approx_{BE} M^* \iff \exists \phi \in \Phi: b = \phi(b^*). \]

- $(\Phi, \circ)$ is an algebraic group. This implies that $\approx_M$ and $\approx_{BE}$ are equivalence relations.
Consistency

- We demand that *equivalent markings* must have:
  - *equivalent direct successors*,
  - *equivalent enabled binding elements*.

- A *symmetry specification* $\Phi$ is *consistent* iff the following properties are satisfied for all symmetries $\phi \in \Phi$, all reachable markings $M_1, M_2$ and all binding elements $b$:
  - $M_1 \xrightarrow{b} M_2 \iff \phi(M_1) \xrightarrow{\phi(b)} \phi(M_2)$.
  - $\phi(M_0) = M_0$. 

\[ \text{24/01/2005} \]
Protocol with multiple receivers

- **Symmetries** are defined as *consistent permutations* of *receiver-IDs*:
  - When we model each receiver by a *separate module* we permute the *markings* of these modules.
  - When we model all receivers by a single module (adding a *new component* to the token colours) we permute the *colour values* in the type:
    \[ REC = \{ \text{rec}_1, \text{rec}_2, \text{rec}_3, \ldots \}. \]
Construction of state spaces with symmetries

- State spaces with *symmetries* are *constructed* in the same way as *ordinary state spaces*, except that:
  - Before *adding a new node* we check whether the marking is *equivalent* to the marking of an existing node.
  - Before *adding a new arc* we check whether the binding element is *equivalent* to the binding element of an existing arc (from the same source node).
What can we prove from state spaces with symmetries?

- State spaces with *symmetries* can be used to *investigate* the same kinds of *behavioural properties* as ordinary state spaces, but only *modulo equivalence*.

- As an example, this means that:
  - We *cannot* investigate whether a certain *marking* is reachable *itself*.
  - Instead we *can* investigate whether there is an *equivalent marking* which is reachable.
### Statistics – symmetries

<table>
<thead>
<tr>
<th>Limit = Receivers</th>
<th>2</th>
<th>3</th>
<th>4 (3 packets)</th>
<th>5 (2 packets)</th>
<th>6 (2 packets)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Nodes</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Full</td>
<td>921</td>
<td>22,371</td>
<td>172,581</td>
<td>486,767</td>
<td>5,917,145</td>
</tr>
<tr>
<td>Sym</td>
<td>477</td>
<td>4,195</td>
<td>9,888</td>
<td>8,387</td>
<td>24,122</td>
</tr>
<tr>
<td>Ratio</td>
<td>1.93</td>
<td>5.33</td>
<td>17.45</td>
<td>58.04</td>
<td>245.30</td>
</tr>
<tr>
<td><strong>Arcs</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Full</td>
<td>1,832</td>
<td>64,684</td>
<td>671,948</td>
<td>2,392,458</td>
<td>35,068,448</td>
</tr>
<tr>
<td>Sym</td>
<td>924</td>
<td>11,280</td>
<td>32,963</td>
<td>31,110</td>
<td>101,240</td>
</tr>
<tr>
<td>Ratio</td>
<td>1.98</td>
<td>5.73</td>
<td>20.38</td>
<td>76.90</td>
<td>346.39</td>
</tr>
<tr>
<td><strong>Time</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Full</td>
<td>2 secs</td>
<td>4 mins</td>
<td>191 mins</td>
<td>-----</td>
<td>-----</td>
</tr>
<tr>
<td>Sym</td>
<td>3 secs</td>
<td>2 mins</td>
<td>8 mins</td>
<td>8 mins</td>
<td>1 hour</td>
</tr>
<tr>
<td>Ratio</td>
<td>0.7</td>
<td>2.0</td>
<td>23.9</td>
<td>-----</td>
<td>-----</td>
</tr>
<tr>
<td><strong>Perms</strong></td>
<td>n!</td>
<td>2</td>
<td>6</td>
<td>24</td>
<td>120</td>
</tr>
</tbody>
</table>

Prototype implementation in 1998.
We can be more general

- We have defined the *equivalence relations* for markings and bindings elements from a *set of symmetry functions*.

- Instead we may define the *equivalence relations* *directly* (i.e. from scratch).

- An *equivalence specification* is a pair \((\approx_M, \approx_{BE})\) where:
  - \(\approx_M\) is an *equivalence relation* on the set of *all markings*.
  - \(\approx_{BE}\) is an *equivalence relation* on the set of *all binding elements*. 
Consistency

- As before, we demand that *equivalent markings* must have:
  - *equivalent direct successors*,
  - *equivalent enabled binding elements*.

- An *equivalence specification* \((\approx_m, \approx_{BE})\) is *consistent* iff for all reachable markings \(M_1, M_2, M\) and all binding elements \(b\):
  \[
  M_1 \approx_m M_2 \land M_1 \xrightarrow{b} M \implies \exists M^* \approx_m M \exists b^* \approx_{BE} b : M_2 \xrightarrow{b^*} M^*.
  \]
State spaces with equivalence classes

- State spaces with *equivalence classes* are constructed in the same way as state spaces with *symmetries*.
- They can be used to *investigate* the same kinds of *behavioural properties*.
- State spaces with *symmetries* is a *special case* of state spaces with *equivalence classes*.
Intermediate state of protocol

- **Receiver** expects packet no. 6.
- **Sender** is still sending packet no. 5.
- This packet will be ignored. It is *old*.
- This acknowledgment will also be ignored. It is *old*.
Equivalence relation

- A marking \( M(p) \) where \( p \) is one of the network places A, B, C, D is *split* into two parts:
  \[
  M(p) = M(p)_{\text{OLD}} + M(p)_{\text{NEW}}
  \]
  Old packets/acks

- Two markings \( M_1 \) and \( M_2 \) are *equivalent* iff:
  \[
  \begin{align*}
  M_1(p) &= M_2(p) \quad \text{for } p \notin \{A, B, C, D\} \\
  |M_1(p)_{\text{OLD}}| &= |M_2(p)_{\text{OLD}}| \\
  M_1(p)_{\text{NEW}} &= M_2(p)_{\text{NEW}}
  \end{align*}
  \]
  All remaining packets/acks
Two equivalent states
## Statistics – equivalence classes

<table>
<thead>
<tr>
<th>Limit:</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Nodes</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Full</td>
<td>33</td>
<td>293</td>
<td>1,829</td>
<td>9,025</td>
<td>37,477</td>
<td>136,107</td>
</tr>
<tr>
<td>Equiv</td>
<td>33</td>
<td>155</td>
<td>492</td>
<td>1,260</td>
<td>2,803</td>
<td>5,635</td>
</tr>
<tr>
<td>Ratio</td>
<td>1.0</td>
<td>1.89</td>
<td>3.72</td>
<td>7.16</td>
<td>13.37</td>
<td>24.15</td>
</tr>
<tr>
<td><strong>Arrows</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Full</td>
<td>44</td>
<td>764</td>
<td>6,860</td>
<td>43,124</td>
<td>213,902</td>
<td>891,830</td>
</tr>
<tr>
<td>Equiv</td>
<td>44</td>
<td>383</td>
<td>1,632</td>
<td>5,019</td>
<td>12,685</td>
<td>28,044</td>
</tr>
<tr>
<td>Ratio</td>
<td>1.0</td>
<td>1.99</td>
<td>4.20</td>
<td>8.59</td>
<td>16.86</td>
<td>31.80</td>
</tr>
<tr>
<td><strong>Seconds</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Full</td>
<td>1</td>
<td>1</td>
<td>6</td>
<td>56</td>
<td>642</td>
<td>7,507</td>
</tr>
<tr>
<td>Equiv</td>
<td>1</td>
<td>1</td>
<td>7</td>
<td>36</td>
<td>157</td>
<td>553</td>
</tr>
<tr>
<td>Ratio</td>
<td>1.0</td>
<td>1.0</td>
<td>0.9</td>
<td>1.56</td>
<td>4.09</td>
<td>13.58</td>
</tr>
</tbody>
</table>

Timed protocol

Creation time 787 → 0

Coloured Petri Nets
24/01/2005
Timed protocol
Simple protocol

The two counters are **monotonously increased.**

They can be used as a **progress measure.**
Progress measure

- **PM**: STATES → A

- **Monotone (non-decreasing):**
  \[ PM(X) \leq PM(Y) \]

- **Protocol:** (NextSend, NextRec)
  lexicographical ordering.
States sorted by progress measure

Initial state
Construction of state space

- All nodes to be processed are in front of the sweep-line.
- All arcs go left-to-right or vertical.
- All new nodes are added in front of the sweep-line.
- We do not need the nodes behind the sweep-line. They can be deleted from memory.
We continue the construction

- The *sweep-line* moves from left to right.
  - In front of it, we *add new nodes*.
  - Behind it, we *remove nodes*. 
## Statistics – sweep-line

<table>
<thead>
<tr>
<th>Limit:</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Nodes</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Full</td>
<td>33</td>
<td>293</td>
<td>1,829</td>
<td>9,025</td>
<td>37,477</td>
<td>136,107</td>
</tr>
<tr>
<td>Sweep</td>
<td>33</td>
<td>134</td>
<td>758</td>
<td>4,449</td>
<td>20,826</td>
<td>82,586</td>
</tr>
<tr>
<td>Ratio</td>
<td>1.0</td>
<td>2.19</td>
<td>2.41</td>
<td>2.03</td>
<td>1.80</td>
<td>1.65</td>
</tr>
<tr>
<td><strong>Arcs</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Full</td>
<td>44</td>
<td>764</td>
<td>6,860</td>
<td>43,124</td>
<td>213,902</td>
<td>891,830</td>
</tr>
<tr>
<td>Sweep</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
</tr>
<tr>
<td>Ratio</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
</tr>
<tr>
<td><strong>Secs</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Full</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>16</td>
<td>153</td>
<td>1,634</td>
</tr>
<tr>
<td>Sweep</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>9</td>
<td>93</td>
<td>1,083</td>
</tr>
<tr>
<td>Ratio</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>1.78</td>
<td>1.65</td>
<td>1.51</td>
</tr>
</tbody>
</table>

- Intel Pentium III, 1GHz, 1 GB RAM
### Statistics – sweep-line

<table>
<thead>
<tr>
<th>Packets:</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Nodes</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Full</td>
<td>9,025</td>
<td>20,016</td>
<td>38,885</td>
<td>68,720</td>
<td>113,121</td>
</tr>
<tr>
<td>Sweep</td>
<td>4,449</td>
<td>8,521</td>
<td>14,545</td>
<td>22,905</td>
<td>33,985</td>
</tr>
<tr>
<td>Ratio</td>
<td>2.03</td>
<td>2.35</td>
<td>2.67</td>
<td>3.00</td>
<td>3.33</td>
</tr>
<tr>
<td><strong>Arks</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Full</td>
<td>43,124</td>
<td>99,355</td>
<td>198,150</td>
<td>356,965</td>
<td>596,264</td>
</tr>
<tr>
<td>Sweep</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
</tr>
<tr>
<td>Ratio</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
</tr>
<tr>
<td><strong>Secs</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Full</td>
<td>12</td>
<td>41</td>
<td>125</td>
<td>345</td>
<td>864</td>
</tr>
<tr>
<td>Sweep</td>
<td>7</td>
<td>21</td>
<td>57</td>
<td>152</td>
<td>359</td>
</tr>
<tr>
<td>Ratio</td>
<td>1.71</td>
<td>1.95</td>
<td>2.19</td>
<td>2.27</td>
<td>2.41</td>
</tr>
</tbody>
</table>

- AMD Athlon 1.33GHz, 512 MB RAM

Coloured Petri Nets
24/01/2005
Sweep-line method – pro/contra

- We can construct *larger state spaces*, since we do not need to have all states in *memory at the same time*.
- In a *timed CP-net* we can use the *global clock* as a *progress measure* – *time does not go backwards*.
- “Problems”:
  - *Analysis* must be done *on the-fly*.
  - To deal with *reactive systems* we need to be able to use *non-monotonous* progress measures.
  - *Counter examples* are *more difficult* to construct, since part of the state space has been *deleted from memory*. 
Overview of talk

**Modelling**
- Basic language
  - syntax
  - semantics
- Extensions
  - modules
  - time
- Tool support
  - editing
  - simulation

**Analysis**
- State spaces
  - full
  - symmetries
  - equivalence classes
  - sweep-line
- Place invariants
  - check of invariants
  - use of invariants
Place invariants

- The **basic idea** is similar to the use of invariants in **program verification**.
- An **invariant** describes a **property** which is **fulfilled for all reachable states**.
  - We first **construct** a set of place invariants.
  - Then we **check** whether they are fulfilled.
  - Finally, we **use** the place invariants to **prove behavioural properties** of the CP-net.
Logo of Petri net community
Distributed data base
Data base managers

\[ DBM = \{d(1), d(2), d(3)\} \]

\[ 1 \cdot d(1) + 1 \cdot d(2) + 1 \cdot d(3) \]
Message buffers

\[ MES = \{(s,r) \in \text{DBM} \times \text{DBM} \mid s \neq r\} \]

\[
1\cdot(d(1),d(2)) + \\
1\cdot(d(1),d(3)) + \\
1\cdot(d(2),d(1)) + \\
1\cdot(d(2),d(3)) + \\
1\cdot(d(3),d(1)) + \\
1\cdot(d(3),d(2))
\]

\[
\text{Mes}(d(2)) = 1\cdot(d(2),d(1)) + 1\cdot(d(2),d(3))
\]
Mutual exclusion

\[ E = \{ e \} \]
Distributed data base

\[ s = d(3) \]
Distributed data base

\[ s = d(3) \]
\[ r = d(1) \]
Distributed data base

Coloured Petri Nets

24/01/2005
Distributed data base

Update and Send Messages

Send an Acknowledgment

Receive a Message

Active

Inactive

Waiting

Send an Acknowledgment

(3,1)

(1,2)

(2,3)

(1,3)

(2,1)

(3,2)

(3,1)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)

(MES)
Distributed data base

\[ s = d(3) \]
Distributed data base

Initial marking

Coloured Petri Nets
24/01/2005
Data base managers

M(Waiting) + M (Inactive) + M(Performing) = DBM
Message buffers

\[ M(\text{Unused}) + M(\text{Sent}) + M(\text{Received}) + M(\text{Acknowl}) = MES \]
Mutual exclusion

\[ M(\text{Active}) + M(\text{Passive}) = E \]
Received messages

Weight function

**MES → DBM**

**Rec(s,r) = r**

Rec( M(Received)) = M(Performing)

Coloured Petri Nets
24/01/2005
Used messages

\[ \text{Mes(Waiting)} = \text{M(Sent)} + \text{M(Received)} + \text{M(Acknowledged)} \]
Active and waiting

\[ \text{Ign}(M(\text{Waiting})) = M(\text{Active}) \]

\[ \text{Ign}(x) = e \]
Place invariants

- Waiting + Inactive + Performing = DBM
- Unused + Sent + Receive + Acknowledged = MES
- Active + Passive = E
- Rec(Received) = Performing
- Mes(Waiting) = Sent + Received + Acknowledge
- Ign(Waiting) = Active

More invariants can be obtained by linear combinations:
- Ign(Waiting) + Passive = E
Construction of invariants

- **Construction** of invariants can be *manual*. This is often straightforward:
  - *System specification*.
  - *Knowledge of system*.

- **Automatic calculation** of all place invariants is possible, but:
  - *Rather complex*.
  - *Results* are *difficult* to represent in a form which is *useful for analysis*.

- **Interactive calculation** is much more suitable:
  - The *user* proposes *some* of the weights.
  - The *tool* calculates the *remaining* weights or reports an *inconsistency*. 
How to use invariants

◆ Ordinary *programming languages*:
  - No one would construct a *large program* and then expect *afterwards* to be able to *calculate invariants*.
  - Instead *invariants* are constructed *together* with the program.

◆ For *CP-nets* we should do the same:
  - During the system specification and modelling the designer gets a lot of *knowledge* about the system.
  - Some of this knowledge can easily be *formulated* as place *invariants*.
  - The *invariants* can be *checked* and in this way *errors* can be found.
  - The *errors* can easily be *located*.
We use invariants to prove behavioural properties of the system

- As an example, let us prove that the data base system cannot deadlock.
  - Let a reachable marking be given.
  - We will then prove that at least one transition is enabled.

All invariants are fulfilled
M(Waiting) + M(Inactive) + M(Performing) = DBM

All data base managers must be:

Let us assume that at least one manager is Waiting
Rec( M(Received)) = M(Performing)

There is a message buffer at Received with d(i) as receiver
Next let us assume that at least one manager is Waiting
Coloured Petri Nets
24/01/2005

Ign(Waiting) + Passive = E

Exactly one token on Waiting
Ign(Waiting) = Active

Exactly one token on Active

Coloured Petri Nets
24/01/2005
M(Waiting) + M(Inactive) + M(Performing) = DBM

The other data base managers must be Inactive
\[
\text{Mes(Waiting)} = \text{M(Sent)} + \text{M(Received)} + \text{M(Acknowledged)}
\]

The message buffers sent by \( d(i) \) must be:

\[
\text{Rec( M(Received)) = M(Performing)}
\]
M(Flowing) + M (Inactive) + M(Performing) = DBM

All data base managers must be Inactive

No tokens on Waiting
Mes(Waiting) = M(Sent) + M(Received) + M(Acknowledged)

No tokens on Sent, Received, and Acknowledged
M(Unused) + M(Sent) + M(Received) + M(Acknowl) = MES

All message buffers are Unused
Ig(Waiting) = Active

No tokens on Active

OK

Active + Passive = E

One e-token on Passive

Initial marking

Received

Update and Send Messages

Receive all Acknowledgements

Send an Acknowledgement

Send a Message

DBM

Initial marking

E

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)

Mes(s)
We have now investigated all possible reachable markings

- For each of them we have used the invariants to prove that at least one transition is enabled.

- Hence, we conclude that the data base system cannot deadlock.
Invariants - pro/contra

- Invariants can be used to verify *large systems*.
  - *No complexity* problems.
  - It is possible to *combine* invariants from *individual modules*.

- Invariants can be used to verify a system *without fixing* the *system parameters* such as the number of sites in the data base system.

- The main drawback is that the *user* needs some *ingenuity* to:
  - *Construct invariants*. This can be supported by *computer tools* – *interactive process*.
  - *Use invariants*. This can also be supported by *computer tools* – *interactive process*. 
Conclusion

One of the reasons for the success of CP-nets is the fact that we simultaneously have worked in all three areas.
More information on CP-nets

- The following *web-pages* contain a lot of information about CP-nets and their tools:
  http://www.daimi.au.dk/CPnets/

- *Introduction to CP-nets*, including a number of detailed examples.

- Manual for *CPN Tools*.
  - The tool is *free of charge* also for commercial companies.

- A list of more than 50 published papers describing different *industrial applications* of CP-nets and the CPN tools.

- Details of a 3-volume *CPN text book*. 

Coloured Petri Nets
24/01/2005