CPN
A concrete language for high-level Petri nets

prof.dr.ir. Wil van der Aalst
Last week ..
5 philosophers
CPN (Colored Petri nets)

- CPN is the language developed by Kurt Jensen et al.
- CPN supports the extensions with time, color and hierarchy.
- CPN is based on standard ML.
- CPN is supported by Design/CPN and CPN Tools.
- In 2010, the support and further development of CPN Tools moved from Aarhus University (Denmark) to TU/e.
- Version 3 was the first version released by TU/e.
- For more information: http://cpntools.org
Values and types

• Syntax is needed to type places and give values (colors) to tokens.
• Adopted from Standard ML

Outline:
• Basic types: int, string, bool, (real), and unit.
• Type constructors: with, product, record, list.
• Defining constants.
Basic types

- Integers (int), e.g., 5, 34234, ~32423.
- Reals (real), e.g., 34.34, ~23.0, 7e3, 4e~2.
- Strings (string), e.g., "Hallo", "28-02-2003".
- Booleans (bool): true and false.
- unit: type with just one value ()

- ~32423 means -32423
- ~23.0 means –23, 7e3 means 7000.0, and 4e~2 means 0.04
- unit is used to represent black (i.e., uncolored) tokens
- Reals are supported in ML but cannot be used as a color set because equality is undefined and hence bindings cannot be calculated
Basic operators

- ~ for the unary minus
- + and – for reals and integers
- * (multiplication) for reals and integers
- / (division) for reals
- div and mod for integers (28 div 10 = 2, 28 mod 10 = 8)
- =, >, <, >=, <=, <> for comparing things (note the notation for >= (greater than), <= (smaller than), and <> (not equal)).
- ^ for strings (concatenation "AA"^"BB" = "AABB")
Logical operators

- **not** (for negation)
- **andalso** (for logical AND)
- **orelse** (for logical OR)
- **if then else** (choice based on Boolean argument, the then and else part should be of the same type)

- \( \text{not}(1=1) \) results in false
- \( (1=1) \) andalso \( \text{not}(0>1 \text{ orelse } 2>3) \) results in true
- if "X"="X" then 3 else 4 results in 3
Exercise: Give type and value of each result

a) if (4>=4) then ("Hello" ^ " " ^ "World") else "X"

b) if true then 20 div 8 else 20 mod 8

c) not(1=1 or else 1=2)

d) not(1=1 andalso 1=2)

e) if ("Hello" ^ " " ^ "World" = "X") then 20 else 3
Color set declarations

- A color set is a type that is defined using a color set declaration \texttt{color ... = ...},\textsuperscript{1} e.g.,
  - \texttt{color I = int;}
  - \texttt{color S = string;}
  - \texttt{color B = bool;}
  - \texttt{color U = unit;}
- Once declared, it may be used to type places.
- Newly defined types like I,S,B,U may be used in other color set declarations.

\textsuperscript{1} "color" is shown as "colset" in CPN Tools, but one can type "color"
Creating subtypes using the "with" clause

• color Age = int with 0..130;
• color Temp = int with ~30..40;
• color Alphabet = string with "a".."z";
• color YN = bool with (no,yes);
• color BlackToken = unit with null;
Creating new types using the "with" clause

• color Human = with man | woman | child;
• color ThreeColors = with Green | Red | Yellow;
Creating new types using product, record, and list constructors

- color Coordinates = product I * I * I;
- color HumanAge = product Human * Age;
- color CoordinatesR = record x:I * y:I * z:I;
- color CD = record artists:S * title:S * noftracks:I;
- color Names = list S;
- color ListOfColors = list ThreeColors;
Possible values (colors)

- Coordinates: (1,2,3), (~4,66,0), ...
- HumanAge: (man,50), (child,3), ...
- CoordinatesR: {x=1, y=2, z=3}, {x=~4, y=66, z=0}, {y=2, x=1, z=3}, ...
- CD: {artists="Havenzangers", title="La La", noftracks=10}, ...
- Names: ["John", "Liza", "Paul"], [ ], ...
- ListOfColors = [Green], [Red, Yellow], ...

Note the difference between products and records.
Example

- color Driver = string;
- color Lap = int with 1..80;
- color TimeMS = int with 0..10000000;
- color LapTime = product Lap * TimeMS;
- color LapTimes = list LapTime;
- color DriverResults = record d:Driver * r:LapTimes;
- color Race = list DriverResults;
Example (2)

A possible color of type Race is:

```json
[{d="Jos Verstappen",
  r=[(1,31000),(2,33400),(3,32800)]},
{d="Michael Schumacher",
  r=[(1,32200),(2,31600),(3,30200),(4,29600)]},
{d="Rubens Barrichello",
  r=[(1,34500),(2,32600),(3,37200),(4,42600)]}]```
Operations on lists and records

- `[ ]` denotes the empty list
- `^^` concatenates two lists, e.g., `[1,2,3]^^[4,5]` evaluates to `[1,2,3,4,5]`
- `::` adds an element in front of a list, e.g., `'a'::['b','c']` evaluates to `['a','b','c']`
- `#` extracts a field of a record `#x{x=1,y=2}` evaluates to `1`
It is possible to define constants, e.g.,
• val jv = "Jos Verstappen" : Driver;
• val lap1 = 1 : Lap;
• val start = 0 : Time;
• val seven = 7 : int;
Example

- Determine the value of constant Monaco:
  - val jv = "Jos Verstappen" : Driver;
  - val r1jos = (1,31000) : LapTime;
  - val r2jos = (2,33400) : LapTime;
  - val r3jos = (3,32800) : LapTime;
  - val r123jos = (((1,31000)::[(2,33400)])^^[(3,32800)]) : LapTimes;
  - val jos = {d=jv, r=r123jos} : DriverResults;
  - val michael = {d="Michael Schumacher", r=[(1,32200), (2,31600),
      (3,30200), (4,29600)]} : DriverResults;
  - val rubens = {d="Rubens Barrichello", r=[(1,34500), (2,32600), (3,37200), (4,42600)]} : DriverResults;
  - val Monaco = jos :: ([michael]^^[rubens]) : Race;
Exercise

• Determine the value of the following constants:
  • val e1 = r1jos::[ ];
  • val e2 = #d(michael);
  • val e3 = (#r(jos))^(#r(rubens));
So what?
We can now type and initialize places!

declarations

- color Driver = string;
- color Lap = int with 1..80;
- color Time = real with 0.0..1000.0;
- color LapTime = product Lap * Time;
- color LapTimes = list LapTime;
- color DriverResults = record d:Driver * r:LapTimes;
- color Race = List DriverResults;

val Monaco = [{d="Jos Verstappen", r=[(1,31000), (2,33400), (3,32800)]},
              {d="Michael Schumacher", r=[(1,32200), (2,31600), (3,30200), (4,29600)]},
              {d="Rubens Barrichello", r=[(1,34500), (2,32600), (3,37200), (4,42600)]}];
To initialize places with multiple tokens but also for various other purposes we need **multi-sets** also referred to as **bags**.

In CPN multi-sets are denoted using the following notation: $x_1^{v_1} ++ x_2^{v_2} ++ \ldots ++ x_n^{v_n}$ where $v_1$ is a value and $x_1$ the number of times this element appears in the multi-set, etc.

E.g., $4^{``Red``} ++ 2^{``Green``} ++ 1^{``Blue``}$ is a multi-set containing 7 elements
Initialization expressions

- **no tokens**
  - p1 INT
  - p2 STR

- **one token**
  - p1 INT
  - p2 STR

- **six tokens**
  - p1 INT
  - p2 STR

```
1 2 ++ 5 4

"John"

1"John" ++ 5"Sara"
```
• Multi-sets are implemented as lists, i.e., 4"Red" ++ 2"Green" ++ 1"Blue" can also be written as e.g. ["Red","Red","Red","Red","Green","Green","Blue"].
• This is useful when using list functions.
Arc inscriptions

- Arc inscriptions are used to define input-output behavior.
- Arc inscriptions may use variables.
- Variables are typed and need to be declared

```plaintext
| color STR = string; |
| var s:STR; |
| _________________ |
```

```
| "Hello World" |
| _____________ |
```

```
\[ \text{p1} \quad \text{STR} \quad \text{s} \quad \text{t1} \quad \text{STR} \quad \text{s} \quad \text{p2} \]
```
Example

- Give final marking.

```plaintext
| color INT = int; |
| var x:INT; |
```

Diagram:

- p1
- 1`2 ++ 2`3
- INT
- x
- t1
- x+2
- p2
- INT
Binding

- Given a transition \( t \) with variables \( x_1, x_2, \ldots, x_n \) on its input and output arcs, a binding of \( t \) allocates a concrete value to each of these variables. These values should be of the corresponding type.
- A binding is enabled if there are tokens matching the values of the arc inscriptions.
- If a binding is enabled, it can occur, i.e., the transition fires while consuming and producing the corresponding tokens.
- The pair \((t_1, \langle x_1=v_1, x_2=v_2, \ldots x_n=v_n \rangle)\) is called a binding element.
Example

- Two binding elements: \((t1, <x=2>)\) and \((t1, <x=3>)\)

```haskell
| color INT = int;
| var x:INT;
```

```
INT
```

```
t1
```

```
p2
```

```
p1
```

```
1\`2 ++ 2\`3
```

```
x + 2
```

```
x
```

```
x
```

```
x
```

```
x
```

```
x
```

```
x
```

```
x
```
Example

- Binding element \((t_1, \langle x=0 \rangle)\). After it occurred \((t_1, \langle x=1 \rangle)\), etc.

```plaintext
color INT = int;
var x:INT;
```

![Diagram](image)
Example

color INT = int;
var x:INT;
var y:INT;

(t2, <x=1, y=2>)

No binding possible!

(t2, <x=1, y=3>)
Exercise

Give all possible binding elements and final markings

```
color INT = int;
var x:INT;
var y:INT;
```
Exercise

- Give all possible binding elements and final markings

```plaintext
color INT = int;
var x:INT;
var y:INT;
```

Diagram:
- p4
- INT
- 2\(^{5}++3\^{7}\)
- t2
- INT
- x
- if x <=6 then 1\^x else empty
- if x >6 then 1\^x else empty
- p5
- INT
- x
- p6
- INT
- x
Exercise

- Give all possible binding elements and final markings

```plaintext
color INT = int;
var x:INT;
var y:INT;
```

![Diagram](image)
Exercise

• Give all possible binding elements and a final marking

```
color STR = string;
var x:STR;
var y:STR;
color INT = int;
var z:INT;
color S = list STR;
var s:S;
color R = record a:STR * b:S;
var r:R;
```
Exercise

• Give all possible binding elements and a final marking

```plaintext
| color STR = string; |
| var x:STR; |
| var y:STR; |
| color INT = int; |
| var z:INT; |
| color S = list STR; |
| var s:S; |
| color R = record a:STR * b:S; |
| var r:R; |
```

```
p5

3"Hi"++2"Ho"

STR

p2

x

t2

x::s

z

z+1

s

p6

0

p7

[ ]

INT

S
```
Example: Voting

color Party = with CDA | PVDA | VVD;
var x:Party;
color Count = int with 0..200000000;
var y:Count;
color PC = product Party * Count;
```
2 `CDA++
3 `PVDA++
1 `VWD

vote
Party

(x,y)
(x,y+1)

votes
PC

1 `(CDA,0)++
1 `(PVDA,0)++
1 `(VWD,0)

2 `CDA ++ 3 `PVDA ++ 1 `VWD
```

```
var x:Party;
var y:Count;
var PC = product Party * Count;
```
Exercise: Bank

• Consider a simple banking system. There are 1000 accounts numbered from 1 to 1000. People can deposit or withdraw money. Only amounts between 1 EURO and 5000 EURO can be deposited or withdrawn. The account may have a negative balance.

• Model this in terms of a CPN model.
Exercise: Article database

• Consider a database system where authors can submit articles. The articles are stored in such a way that it is possible to get a sequential list of articles per author. The list is ordered in such a way that the oldest articles appear first.

• Note that the system should support two actions: submit articles (with name of author and article) and get articles of a given author.

• We assume that each article has a single author and that only authors already registered in the database can submit articles.

• Model this in terms of a CPN model.
Exercise (2)

- Extend the CPN model such that each article can have multiple authors, i.e., the article is stored once for each author, and that there is an explicit action to add authors to the database.
Guard

- A **guard** is a Boolean expression attached to a transition. Only binding elements which evaluate to true are enabled.
- Denoted by square brackets.

```plaintext
<table>
<thead>
<tr>
<th>color</th>
<th>INT = int;</th>
</tr>
</thead>
<tbody>
<tr>
<td>var x</td>
<td>x:INT;</td>
</tr>
<tr>
<td>var y</td>
<td>y:INT;</td>
</tr>
<tr>
<td>var z</td>
<td>z:INT;</td>
</tr>
</tbody>
</table>
```

Guard evaluates to false for binding `(t1,<x=1, y=2>)`
Example

- Give all enabled binding elements and the final marking

```
color INT = int;
var x:INT;
var y:INT;
var z:INT;
```
Exercise

- Give all enabled binding elements and the final marking

```plaintext
color Person= str;
color Text = str;
color Mail = record p:Person * t:Text;
var x:Mail;
```
Exercise

• Give all enabled binding elements and all possible final marking

```
color INT = int;
var x:INT;
```

```
INT
p3

p2
INT
```

```
INT
p1

[<x>5]

INT
```

```
INT
p2

[<x<10]

INT
```

```
INT
p3
```

```
[12++17++115]
```

```
INT
p1

t1

x
```

```
INT
```

```
INT
```

```
INT
```

```
INT
```

```
INT
```

```
INT
```

```
INT
```

```
INT
```

```
INT
```

```
INT
```

```
INT
```
The CPN model assumes that an account could have a negative balance. Change the model such that the balance cannot become negative, i.e., do not accept transactions which lead to a negative balance.
Function declarations

color INT = int;
fun fac(x:INT) = if x>1 then x*fac(x-1) else 1;
fun fib(x:INT) = if x<2 then 1 else fib(x-1) + fib(x-2);
color L = list INT;
fun sum(x:L) = if x=[] then 0 else hd(x)+sum(tl(x));
fun odd(x:L) = if x=[] then [] else hd(x)::(if tl(x)=[] then []
else odd(tl(tl(x))));

• Calculate fac(fib(3)) and modify odd such that the odd lines are returned in reversed order.
Where to find standard functions?

• These sheets.

• cpntools.org, see for example
  http://cpntools.org/documentation/concepts/colors/declarations/colorsets/list_colour_sets and
  http://cpntools.org/documentation/concepts/colors/declarations/colorsets/colour_set_functions

• www.standardml.org, see for example
  http://www.standardml.org/Basis/list.html#LIST:SIG:SPEC for list functions,
  http://www.standardml.org/Basis/integer.html#INTEGER:SIG:SPEC for integer functions,
  http://www.standardml.org/Basis/string.html#STRING:SIG:SPEC for string functions, etc.
Example

color INT: int;
color Author = string;
color Article = string;
color AL = list Article;
color AAL = product Author * AL;
var x: Author;
var y: AL;
fun count(z: AL) = if z = [] then 0 else 1 + tl(z)
Questions

• Is it possible to have multiple arcs connecting a place and a transition?
• Is it possible to have multi-sets as arc inscriptions on input arcs?
• Is it possible to use constants or other expressions without variables as arc inscriptions?
• Is it possible to use records, lists, etc. with variables (e.g., \{a=x,b=y\} and x::y) in arc inscriptions?
Example: Multiple arcs

- \text{color } I = \text{int};
- \text{color } U = \text{unit};
- \text{color } L = \text{list } I;
- \text{color } R = \text{record } a: I * b: I;
- \text{var } x: I;
- \text{var } y: I;
- \text{var } z: I;
- \text{var } s: L;

\begin{itemize}
\item multiple arcs are allowed
\end{itemize}
Example: Multi-sets and constants

- color I = int;
- color U = unit;
- color L = list I;
- color R = record a:I * b:I;
- var x:I;
- var y:I;
- var z:I;
- var s:L;

Multisets on input arcs are allowed

Constants are allowed
Example: Records

```
| color l = int;  
| color U = unit;  
| color L = list l;  
| color R = record a:l * b:l;  
| var x:l;        
| var y:l;        
| var z:l;        
| var s:L;        
```

Records with variables as arc inscriptions.
Example: Lists

color I = int;
color U = unit;
color L = list I;
color R = record a:I * b:I;
var x:I;
var y:I;
var z:I;
var s:L;

lists with
variables as arc
inscriptions
Requirement

```
color I = int;
color U = unit;
color L = list I;
color R = record a:I * b:I;
var x:I;
var y:I;
var z:I;
var s:L;
```

ERROR: z is unbound!

It should be possible to bind variables to concrete token values!!
Trick: use lists on arcs to produce/consume multi-sets of tokens
Another example

```haskell
\n\n\n\n\n```
Priority (no priority P_NORMAL = 1000)
Priority (same)
Priority (P_HIGH wins)
Priority: Guess final state
Result

```cpp
mixed-priority.cpn
Step: 50
Time: 0

Standard priorities
- val P_HIGH = 100;
- val P_NORMAL = 1000;
- val P_LOW = 10000;

Standard declarations
- colset UNIT
- colset INT
- colset BOOL
- colset STRING

Monitors
different-prio
```

![CPN Tools interface with a diagram of a Petri net model labeled 'mixed-priority.cpn' with steps for different-prio and priority declarations showing transitions between places and transitions labeled with priorities.]
Global property (t2 never fires)
Time in CPN

- Tokens are either **untimed** (are always available) or **timed** (bear a timestamp).
- Color sets can be made timed color sets by adding the keyword **timed**.
- A **delay** is modeled by `v@+d` as arc expression on an outgoing arc where `v` is the value and `d` is the delay of the produced token.
- Delays may depend on the values of tokens to be consumed (i.e., through the binding of variables).
Example

color STR = string timed;
var x:STR;
color INT = int;
var y:INT;
Exercise

- Determine a possible final state.

```plaintext
| color STR = string timed; |
| var x:STR; |
| color INT = int; |
| var y:INT; |
```

```
2"Hi"++2"Ho"
```
Time (t1 is enabled at time 2)
t1 fired at time 2; t2 is enabled at time 4
“Real” time

Please save the net, close it and reopen it for the change to take effect.
Note the types and the @++
Determine final state
Final state (time = 10000)
Overview of CPN (with color and time)

color INT = int;
color STR = string timed;
var x:INT;
var y:STR;
val n = 4:INT;
fun incr(z:INT) = z+1;

place name

initialization
equation
expression

transition name

arc inscription

constant

variable

place type

delay

guard

timed color set

declarations

color set
Coffee and tea example (1)

- We need to produce 100 cups of tea and 100 cups of coffee.
- There are two persons able to manufacture these drinks: Adam and Eve.
- Assume "random allocation".
- Production times:

<table>
<thead>
<tr>
<th></th>
<th>Eve</th>
<th>Adam</th>
</tr>
</thead>
<tbody>
<tr>
<td>tea</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>coffee</td>
<td>12</td>
<td>4</td>
</tr>
</tbody>
</table>
Coffee and tea example (2)

- Simulate the model a couple of times and record the makespan.
- Evaluate two control strategies:
  - Eve just makes tea and Adam just makes coffee.
  - Adam makes coffee and Eve can make both.
  - Eve makes tea and Adam can make both.
- Why is it difficult to model priorities/preferences?

<table>
<thead>
<tr>
<th></th>
<th>Eve</th>
<th>Adam</th>
</tr>
</thead>
<tbody>
<tr>
<td>tea</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>coffee</td>
<td>12</td>
<td>4</td>
</tr>
</tbody>
</table>
Ready @ +/- 620
Eve just makes tea and Adam just makes coffee.

Ready @ 400
Adam makes coffee and Eve can make both.
Eve makes tea and Adam can make both

Ready @ +/- 460
A smarter strategy

- Eve just makes tea and Adam just makes coffee unless ...
  - Eve can make coffee if there are no tea orders left.
  - Adam can make tea if there are no coffee orders left.
- Why is it difficult to model priorities/preferences?
- Let us look at an intermediate solution using counters rather than lists.
CPN model with counters
Almost optimal makespan ...

Adam: 87 coffee
Eve: 100 tea and 13 coffee
Makespan = 356

Optimal
Adam: 88 coffee
Eve: 100 tea and 12 coffee
Makespan = 352
intermezzo
M/G/1 queue

Declarations
- Standard declarations
- colset JobID = int timed;
- colset Timestamp = int;
- colset Job = product JobID*Timestamp timed;
- colset JobList = list Job;
- colset Res = unit;
- var j:JobID;
- var r:Res;
- var l:JobList;
- var t:Timestamp;
- fun iat() = round(exponential(0.05));
- fun pt() = round(normal(10.0,1.0));
- fun Mtime() = IntInf.toInt(time()):int;

Poisson arrival process (expected average interarrival time is $1/0.05=20$).
Expected service time is 10 (Normal distribution)
To measure results: CPN monitors
Create monitors
```
\begin{verbatim}
Var result:INT;

Monitors
- Marking_size_mm1'free_1
  - Type: Marking size
  - Nodes ordered by pages
- Marking_size_mm1'busy_1
  - Type: Marking size
  - Nodes ordered by pages
- List_length_dc_mm1'queue_1
  - Type: List length data collection
  - Nodes ordered by pages

flowtime
- Type: Data collection
- Nodes ordered by pages
- Predicate
- Observer
  fun obs (bindelem) =
    let
      fun obsBindElem (mm1'measure (1, {j,t})) = Mtime() - t
    in
      obsBindElem bindelem
    end
- Init function
- Stop
\end{verbatim}
```
Monitors

Marking_size_mm1'free_1
  Type: Marking size
  □ Logging
  ▶ Nodes ordered by pages

Marking_size_mm1'busy_1
  Type: Marking size
  □ Logging
  ▶ Nodes ordered by pages

List_length_dc_mm1'queue_1
  Type: List length data collection
  □ Logging
  ▶ Nodes ordered by pages

flowtime
  Type: Data collection
  □ Timed
  □ Logging
  ▶ Nodes ordered by pages
  ▶ Predicate
 Observer
  fun obs (bindelem) =
    let
      fun obsBindElem (mm1'measure (1, {j,t})) = Mtime() - t
    in
      obsBindElem _ = ~1
    in
      obsBindElem bindelem
  end
One run

CPN Tools Simulation Performance Report
Net: D:\courses\BIS-2010\CPN\coffee-and-tea:mm1-test.cpn

Note that these statistics have been calculated for data that is not necessarily independent or identically distributed.

**Timed statistics**

<table>
<thead>
<tr>
<th>Name</th>
<th>Count</th>
<th>Avrg</th>
<th>Min</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>List_length_dc:mm1'queue_1</td>
<td>2002</td>
<td>0.319276</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>Marking_size:mm1'busy_1</td>
<td>2002</td>
<td>0.508891</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Marking_size:mm1'free_1</td>
<td>2002</td>
<td>0.491109</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

**Untimed statistics**

<table>
<thead>
<tr>
<th>Name</th>
<th>Count</th>
<th>Sum</th>
<th>Avrg</th>
<th>Min</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>flowtime</td>
<td>1000</td>
<td>16300</td>
<td>16300000</td>
<td>7</td>
<td>71</td>
</tr>
</tbody>
</table>

Simulation steps executed: 4000
Model time: 19682
Generated: Mon Mar 15 20:51:11 2010
Multiple subruns

CPN'Replications.nreplications 10
Results with confidence intervals

<table>
<thead>
<tr>
<th>Name</th>
<th>Count</th>
<th>Avg</th>
<th>99% Half Length</th>
<th>95% Half Length</th>
<th>99% Half Length</th>
<th>Std</th>
<th>Min</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>List_length</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-</td>
<td></td>
<td>2002.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>2002</td>
<td>2002</td>
</tr>
<tr>
<td>Max_id</td>
<td></td>
<td>6.100000</td>
<td>1.294686</td>
<td>1.597697</td>
<td>2.295542</td>
<td>2.233582</td>
<td>5</td>
<td>12</td>
</tr>
<tr>
<td>Min_id</td>
<td></td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Avg_id</td>
<td></td>
<td>0.254992</td>
<td>0.020017</td>
<td>0.024702</td>
<td>0.034923</td>
<td>0.034534</td>
<td>0.202483</td>
<td>0.363056</td>
</tr>
<tr>
<td>Marking_size</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-</td>
<td></td>
<td>2002.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>2002</td>
<td>2002</td>
</tr>
<tr>
<td>Max_id</td>
<td></td>
<td>1.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Min_id</td>
<td></td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Avg_id</td>
<td></td>
<td>0.495415</td>
<td>0.006489</td>
<td>0.007958</td>
<td>0.011434</td>
<td>0.011125</td>
<td>0.472663</td>
<td>0.510056</td>
</tr>
<tr>
<td>Marking_size</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-</td>
<td></td>
<td>2002.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>2002</td>
<td>2002</td>
</tr>
<tr>
<td>Max_id</td>
<td></td>
<td>1.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Min_id</td>
<td></td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Avg_id</td>
<td></td>
<td>0.504555</td>
<td>0.006489</td>
<td>0.007958</td>
<td>0.011434</td>
<td>0.011125</td>
<td>0.489941</td>
<td>0.527337</td>
</tr>
</tbody>
</table>

Flowtime

<table>
<thead>
<tr>
<th>Name</th>
<th>Count</th>
<th>Avg</th>
<th>99% Half Length</th>
<th>95% Half Length</th>
<th>99% Half Length</th>
<th>Std</th>
<th>Min</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max_id</td>
<td></td>
<td>1000.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>1000</td>
<td>1000</td>
</tr>
<tr>
<td>Min_id</td>
<td></td>
<td>65.700000</td>
<td>12.670113</td>
<td>15.635459</td>
<td>22.467397</td>
<td>21.853833</td>
<td>48</td>
<td>119</td>
</tr>
<tr>
<td>Avg_id</td>
<td></td>
<td>6.600000</td>
<td>0.495292</td>
<td>0.500147</td>
<td>0.718602</td>
<td>0.609206</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>Num_id</td>
<td></td>
<td>15131.000000</td>
<td>373.615684</td>
<td>461.057652</td>
<td>662.489155</td>
<td>644.558936</td>
<td>14055</td>
<td>16106</td>
</tr>
<tr>
<td>Avg_id</td>
<td></td>
<td>15.131000</td>
<td>0.375616</td>
<td>0.461058</td>
<td>0.662459</td>
<td>0.644559</td>
<td>14.055000</td>
<td>16.106000</td>
</tr>
</tbody>
</table>
CPN Tools Performance Report
Net: D:\courses\BIS-2010\CPN\coffee-and-tea\mm1-test.cpn
Number of replications: 10

<table>
<thead>
<tr>
<th>Statistics</th>
<th>Name</th>
<th>Avrg</th>
<th>90% Half Length</th>
<th>95% Half Length</th>
<th>99% Half Length</th>
<th>StD</th>
<th>Min</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>List_length_dc_mm1'queue_1</td>
<td>count_iid</td>
<td>2002.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>2002</td>
<td>2002</td>
</tr>
<tr>
<td></td>
<td>max_iid</td>
<td>6.100000</td>
<td>1.294686</td>
<td>1.597697</td>
<td>2.295542</td>
<td>2.233582</td>
<td>5</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>min_iid</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>avg_iid</td>
<td>0.254992</td>
<td>0.020017</td>
<td>0.024702</td>
<td>0.035492</td>
<td>0.034534</td>
<td>0.202483</td>
<td>0.303056</td>
</tr>
<tr>
<td>Marking_size_mm1'busy_1</td>
<td>count_iid</td>
<td>2002.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>2002</td>
<td>2002</td>
</tr>
<tr>
<td></td>
<td>max_iid</td>
<td>1.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>min_iid</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>avg_iid</td>
<td>0.495445</td>
<td>0.006449</td>
<td>0.007958</td>
<td>0.011434</td>
<td>0.011125</td>
<td>0.472663</td>
<td>0.510056</td>
</tr>
<tr>
<td>Marking_size_mm1'free_1</td>
<td>count_iid</td>
<td>2002.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>2002</td>
<td>2002</td>
</tr>
<tr>
<td></td>
<td>maxiid</td>
<td>1.294686</td>
<td>1.597697</td>
<td>2.295542</td>
<td>2.233582</td>
<td>5</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>-------</td>
<td>-------------</td>
<td>-----------</td>
<td>-----------</td>
<td>-----------</td>
<td>-----------</td>
<td>---</td>
<td>----</td>
<td></td>
</tr>
<tr>
<td>miniid</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>avgiid</td>
<td>0.254992</td>
<td>0.020017</td>
<td>0.024702</td>
<td>0.035492</td>
<td>0.034534</td>
<td>0.202483</td>
<td>0.303056</td>
<td></td>
</tr>
</tbody>
</table>

**Marking_size_mml'busy_1**

<table>
<thead>
<tr>
<th></th>
<th>countiid</th>
<th>2002.000000</th>
<th>0.000000</th>
<th>0.000000</th>
<th>0.000000</th>
<th>0.000000</th>
<th>2002</th>
<th>2002</th>
</tr>
</thead>
<tbody>
<tr>
<td>maxiid</td>
<td>1.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>miniid</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>avgiid</td>
<td>0.495445</td>
<td>0.006449</td>
<td>0.007958</td>
<td>0.011434</td>
<td>0.011125</td>
<td>0.472663</td>
<td>0.510056</td>
<td></td>
</tr>
</tbody>
</table>

**Marking_size_mml'free_1**

<table>
<thead>
<tr>
<th></th>
<th>countiid</th>
<th>2002.000000</th>
<th>0.000000</th>
<th>0.000000</th>
<th>0.000000</th>
<th>0.000000</th>
<th>2002</th>
<th>2002</th>
</tr>
</thead>
<tbody>
<tr>
<td>maxiid</td>
<td>1.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>miniid</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>avgiid</td>
<td>0.504555</td>
<td>0.006449</td>
<td>0.007958</td>
<td>0.011434</td>
<td>0.011125</td>
<td>0.489944</td>
<td>0.527337</td>
<td></td>
</tr>
</tbody>
</table>

**flowtime**

<table>
<thead>
<tr>
<th></th>
<th>countiid</th>
<th>1000.000000</th>
<th>0.000000</th>
<th>0.000000</th>
<th>0.000000</th>
<th>0.000000</th>
<th>1000</th>
<th>1000</th>
</tr>
</thead>
<tbody>
<tr>
<td>maxiid</td>
<td>65.700000</td>
<td>12.670113</td>
<td>15.635459</td>
<td>22.464739</td>
<td>21.858383</td>
<td>48</td>
<td>119</td>
<td></td>
</tr>
<tr>
<td>miniid</td>
<td>6.600000</td>
<td>0.405292</td>
<td>0.500147</td>
<td>0.718602</td>
<td>0.699206</td>
<td>5</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>sumiid</td>
<td>15131.000000</td>
<td>373.615684</td>
<td>461.057652</td>
<td>662.439155</td>
<td>644.558936</td>
<td>14055</td>
<td>16106</td>
<td></td>
</tr>
<tr>
<td>avgiid</td>
<td>15.131000</td>
<td>0.373616</td>
<td>0.461058</td>
<td>0.662439</td>
<td>0.644559</td>
<td>14.055000</td>
<td>16.106000</td>
<td></td>
</tr>
</tbody>
</table>
Will be explained in detail …
Coffee and tea example (3)

• Assume a continuous flow of tea and coffee drinker, i.e., every 5 minutes there is request for tea and every 10 minutes there is a request of coffee.

• There are two persons able to manufacture these drinks (Adam and Eve) and the production times are as before.

• Process the requests in FIFO (first-in-first-out) order.
Flow

arrival process  processing  departure process
FIFO

colset T=string;
colset Prod = string timed;
colset ProdT = product Prod*T;
colset PL = list ProdT;
colset Res = string timed;
colset PR = product ProdT * Res timed;

var p: Prod;
var r:Res;
var l:PL;
var i:INT;
var t:T;
var pt:ProdT;

fun delay(p,r) = 
  if p="tea" then 
    if r="Eve" then 2.0 else 6.0 
  else 
    if r="Eve" then 12.0 else 4.0;

fun interarrivaltime(p)= 
  if p = "tea" then 5.0 else 10.0;

fun Mtime() = ModelTime.time():time;
fun MtimeS() = ModelTime.toString(Mtime()):string;
fun StoR(t:string) = ModelTime.maketime(t): time;
Monitors
- Marking_size_Main_free_1
  - Type: Marking size
  - Nodes ordered by pages
- Marking_size_Main_in_production_1
  - Type: Marking size
  - Nodes ordered by pages
- List_length_dc_Main_to_produce_1
  - Type: List length data collection
  - Nodes ordered by pages
- Tea flow time
  - Type: Data collection
  - Nodes ordered by pages
  - Predicate
  - Observer
    - fun obs (bindelem) =
      let
      fun obsBindElem (Main'measure_tea (1, {p,t})) = Mtime() - StoR(t)
      in
      obsBindElem bindelem
    end
    - Init function
    - Stop
- Coffee flow time
  - Type: Data collection
  - Nodes ordered by pages
  - Predicate
  - Observer
    - fun obs (bindelem) =
      let
      fun obsBindElem (Main'measure_coffee (1, {p,t})) = Mtime() - StoR(t)
      in
      obsBindElem bindelem
    end
    - Init function
    - Stop

fun Mtime() = ModelTime.time():time;
fun MtimeS() = ModelTime.toString(Mtime()):string;
fun StoR(t:string) = ModelTime.maketime(t): time;
average queue length = 0.74 +/- 0.09

average utilization = \frac{2-(0.37 +/- 0.03)}{2}

average flow time coffee = 10.49 +/- 0.46

average flow time tea = 6.63 +/- 0.39
Coffee and tea example (4)

- Assume a continuous flow of tea and coffee drinker, but now evaluate the following alternatives:
  - LIFO (last-in-first-out) order
  - SPT (tea before coffee) order
  - FIFO with Eve preferably working on tea and Adam on coffee.
- Test also your own strategy.
average queue length = 0.65+/-0.08
average utilization = (2-(0.35+/-0.02))/2
average flow time coffee= 10.19+/-0.39
average flow time tea= 6.43+/-0.31

<table>
<thead>
<tr>
<th>Statistics</th>
<th>Name</th>
<th>Avrg</th>
<th>95% Half Length</th>
<th>99% Half Length</th>
<th>StD</th>
<th>Min</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>List_length_dc_Main'to_produce_1</strong></td>
<td>count_id</td>
<td>2002.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>2002</td>
<td>2002</td>
</tr>
<tr>
<td></td>
<td>max_id</td>
<td>4.900000</td>
<td>0.794300</td>
<td>0.980200</td>
<td>1.408333</td>
<td>1.370320</td>
<td></td>
</tr>
<tr>
<td></td>
<td>min_id</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td></td>
</tr>
<tr>
<td></td>
<td>avg_id</td>
<td>0.654748</td>
<td>0.082484</td>
<td>0.101788</td>
<td>0.146248</td>
<td>0.142300</td>
<td>0.489820</td>
</tr>
<tr>
<td><strong>Marking_size_Main't_free_1</strong></td>
<td>count_id</td>
<td>2002.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>2002</td>
<td>2002</td>
</tr>
<tr>
<td></td>
<td>max_id</td>
<td>2.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td></td>
</tr>
<tr>
<td></td>
<td>min_id</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td></td>
</tr>
<tr>
<td></td>
<td>avg_id</td>
<td>0.353071</td>
<td>0.019841</td>
<td>0.024484</td>
<td>0.035178</td>
<td>0.034229</td>
<td>0.311191</td>
</tr>
<tr>
<td><strong>Marking_size_Main'in_production_1</strong></td>
<td>count_id</td>
<td>2002.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>2002</td>
<td>2002</td>
</tr>
<tr>
<td></td>
<td>max_id</td>
<td>2.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td></td>
</tr>
<tr>
<td></td>
<td>min_id</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td></td>
</tr>
<tr>
<td></td>
<td>avg_id</td>
<td>1.646929</td>
<td>0.019841</td>
<td>0.024484</td>
<td>0.035178</td>
<td>0.034229</td>
<td>1.578443</td>
</tr>
<tr>
<td><strong>coffee_flow_time</strong></td>
<td>count_id</td>
<td>333.700000</td>
<td>0.279995</td>
<td>0.345526</td>
<td>0.496446</td>
<td>0.483046</td>
<td>333</td>
</tr>
<tr>
<td></td>
<td>avg_id</td>
<td>10.194212</td>
<td>0.391368</td>
<td>0.482895</td>
<td>0.693916</td>
<td>0.675186</td>
<td></td>
</tr>
<tr>
<td></td>
<td>max_id</td>
<td>91.500000</td>
<td>51.205482</td>
<td>63.189744</td>
<td>90.789862</td>
<td>88.339308</td>
<td></td>
</tr>
<tr>
<td></td>
<td>min_id</td>
<td>4.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td></td>
</tr>
<tr>
<td></td>
<td>sum_id</td>
<td>3401.900000</td>
<td>131.594550</td>
<td>162.93274</td>
<td>233.323670</td>
<td>227.025916</td>
<td>3099.000000</td>
</tr>
<tr>
<td><strong>tea_flow_time</strong></td>
<td>count_id</td>
<td>666.300000</td>
<td>0.279995</td>
<td>0.345526</td>
<td>0.496446</td>
<td>0.483046</td>
<td>666</td>
</tr>
<tr>
<td></td>
<td>avg_id</td>
<td>6.433586</td>
<td>0.307211</td>
<td>0.379112</td>
<td>0.544701</td>
<td>0.529998</td>
<td></td>
</tr>
<tr>
<td></td>
<td>max_id</td>
<td>114.500000</td>
<td>47.711433</td>
<td>58.877938</td>
<td>84.594739</td>
<td>82.311401</td>
<td></td>
</tr>
<tr>
<td></td>
<td>min_id</td>
<td>2.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td></td>
</tr>
<tr>
<td></td>
<td>sum_id</td>
<td>4286.700000</td>
<td>2044.66040</td>
<td>252.566602</td>
<td>362.883049</td>
<td>353.088296</td>
<td>3809.000000</td>
</tr>
</tbody>
</table>
fun delay((p,t),r) = 
    if p="tea" then 
      (if r="Eve" then 2.0 else 6.0) else 
      (if r="Eve" then 12.0 else 4.0) ;

fun interarrivaltime(p)=
    if p = "tea" then 5.0 else 10.0;

val null = ("error","error"):ProdT;

fun sel2([]) =null |
    sel2((p,t)::l) = if p = "tea" then (p,t) else sel2(l);

fun sel(l) = 
    if l=[] then null 
    else if sel2(l) = null then hd(l) else sel2(l); 

fun rem(x,[]) = [] |
    rem(x,y::z) = if x=y then z else x::rem(x,z);

fun Mtime() = ModelTime.time().time;

fun MtimeS() = ModelTime.toString(Mtime()):string;

fun StoR(t:string) = ModelTime.maketime(t): time;
The table shows statistics for various processes.

- **List_length_main_to_produce_1**
  - Average queue length: 0.23 +/- 0.02
  - Average utilization: \((2-(0.59 +/- 0.02))/2\)

- **Marking_size_main_free_1**
  - Average flow time coffee: 8.44 +/- 0.27
  - Average flow time tea: 4.63 +/- 0.09

- **Marking_size_main_in_production_1**
  - Average utilization: \((1.41 +/- 0.02)/2\)
SMART
average queue length = 0.21 +/- 0.01

average utilization = (2 - (0.58 +/- 0.02))/2

average utilization = (1.42 +/- 0.02)/2

average flow time coffee = 7.56 +/- 0.12

average flow time tea = 4.33 +/- 0.06
Compare (1/2)

- **FIFO**
  - Average queue length: $0.74 \pm 0.09$
  - Average utilization: $(1.63 \pm 0.03)/2$

- **LIFO**
  - Average queue length: $0.65 \pm 0.08$
  - Average utilization: $(1.65 \pm 0.02)/2$

- **SPT**
  - Average queue length: $0.23 \pm 0.02$
  - Average utilization: $(1.41 \pm 0.02)/2$

- **SMART**
  - Average queue length: $0.21 \pm 0.01$
  - Average utilization: $(1.42 \pm 0.02)/2$
Compare (2/2)

average flow time coffee = 10.49 +/- 0.46
average flow time coffee = 10.19 +/- 0.39
average flow time coffee = 8.44 +/- 0.27
average flow time coffee = 7.56 +/- 0.12

++

average flow time tea = 6.63 +/- 0.39
average flow time tea = 6.43 +/- 0.31
average flow time tea = 4.63 +/- 0.09
average flow time tea = 4.33 +/- 0.06

++
Example revisited: Punch card desk

```
color STR = string;
color INT = int;
color Pat = record Name:STR * Address:STR *
             DateOfBirth:STR * Gender:STR;
color Emp = record EmpNo:INT * Experience:INT;
color EP = product Pat * Emp;
var p:Pat;
var e:Emp;
val Klaas = {Name="Klaas", Address="Plein 10",
             DateOfBirth="13-Dec-1962", Gender="M"};
val Ann = {EmpNo=641112, Experience=7};
fun d(e:Emp) = if #Experience(e) > 5 then 3 else 4;
```
Improved color sets

color Name = string;
color Street = string;
color Number = int;
color Town = string;
color Address = record s:Street * n:Number * t:Town;
color Day = int with 1..31;
color Month = with Jan | Feb | Mar | Apr | May | Jun | Jul | Aug | Sep | Oct | Nov | Dec;
color Year = int with 0..2100;
color Date = record d:Day * m:Month * y:Year;
color Gender = with male | female;
color Pat = record name:Name * address:Address *
    birthdate:Date * gender:Gender timed;
color EmpNo = int with 100000..999999;
color Emp = record empno:EmpNo *
    experience:Year timed;
color EP = product Pat * Emp timed;
var p:Pat;
var e:Emp;
val Klaas = {name="Klaas",
    address={s="Plein",n=10,t="Unknown"},
    birthdate={d=13,m=Dec,y=1962},
    gender=male}:Pat;
val Ann = {empno=641112, experience=7}:Emp;
fun d(x:Emp) = if #experience(x) > 5 then 3 else 4;
Example revisited: Stock keeping system

```plaintext
color Product = string;
color Number = int;
color StockItem = record prod:Product * number:Number;
color Stock = list StockItem;
var x:StockItem;
var s:Stock;
fun incrs(x:StockItem,s:Stock) = if s=[] then [x] else (if (#prod(hd(s)))=(#prod(x))
  then {prod=(#prod(hd(s))),number=(#number(hd(s)))+(#number(x)))}::tl(s)
  else hd(s):: incrs(x,tl(s)));
fun decrs(x:StockItem,s:Stock)= incrs({prod=(#prod(x)),number=(~(#number(x)))},s);
fun check(s:Stock)= if s=[] then true else if (#number(hd(s)))<0 then false
  else check(tl(s));
val initstock = [{prod="bike", number=4},{prod="wheel", number=2},
  {prod="bell", number=3}, {prod="steering wheel", number=3},
  {prod="frame", number=2}];
```

```
in 1`{prod="bell", number=3}
  StockItem
  x
  increase
  incrs(x,s)
  stock
  1\ initstock
  s
  [check(descrs(x,s))] decrease
  x
  StockItem
  out
```
Store

• Place stock is a so-called store, i.e., it will always contain a single token.
• Only the value of the token matters (not its presence).
• Stores that aggregate elements are always of type list.
• Drawback: complex functions/inscriptions
• Advantage: easy to query the individual items as a whole, e.g., taking the sum of things ...
Function "totalstock"

fun totalstock(s:Stock) = 
  if s=[]
  then 0
  else (#number(hd(s)))+totalstock(tl(s));
Alternative model

```plaintext
| color Product = string;  
| color Number = int;     
| color StockItem = product Product*Number; 
| var p:Product;          
| var x:Number;           
| var y:Number;           
```

```
in
1`("bell",2)  
(p,x)  
(p,y)  
StockItem  

increase

(p,x+y)

stock

1`("bike",4)++  
1`("wheel",2)++  
1`("bell",3)++  
1`("steering wheel",3)++  
1`("frame",2)

[y>=x]

[diagram]

decrease

(p,y-x)

(p,x+y)

(p,y)

(p,x)

out

StockItem

Note the simplicity/elegance of the arc inscriptions.
```
Example: Signing documents

- Documents need to be signed by persons.
- Four persons: Tim, Sue, Clare and John.
- Each document requires three signatures.
- No two signatures of the same person.
- Work in progress is limited to five documents.
color Doc = string;
color Person = string;
color Signatures = list Person;
color SignedDoc = product Doc * Person;
color BlackToken = unit;
var d:Doc;
var p:Person;
var s:Signatures;
fun notin(p:Person, s:Signatures) = 
  if s=[] then true else if p=hd(s) then false else notin(p,tl(s));
fun count(s:Signatures) = if s=[] then 0 else 1+count(tl(s));
Signing documents: Network structure

- **Person**
  - **1"Tim"++
  - **1"Sue"++
  - **1"Clare"++
  - **1"John"

- **Doc**
  - **(d,[])**
  - **(d,[])**

- **SignedDoc**
  - **(d,s)**
  - **(d,s)**

- **_signed_doc**
  - **(d,s)**

- **BlackToken**
  - **5`()**

- **Free**

- **[count(s)<3 andalso notin(p,s)]**

- **[count(s)>=3]**
Exercise

- Replace place free by a place always holding one token.

```plaintext
color Doc = string;
color Person = string;
color Signatures = list Person;
color SignedDoc = product Doc * Person;
color BlackToken = unit;

var d:Doc;
var p:Person;
var s:Signatures;

fun notin(p:Person,s:Signatures) =
  if s=[] then true else if p=hd(s) then false else notin(p,tl(s));

fun count(s:Signatures) = if s=[] then 0 else 1+count(tl(s));
```

```
unsigned_doc

Doc

d

accept

(d,[])

SignedDoc

(d,s)

pile

[d,[]]

(d,s)

SignedDoc

[d,s]

signed_doc

[d,s]

release

[d,s]

Free

BlackToken

[5]

person

Person

"Tim"

"Sue"

"Clare"

"John"

sign

[d,p::s]

[count(s)<3 andalso notin(p,s)]
```
Example: Thermostat system

- At any point the room has a temperature (initially 15 degrees centigrade).
- There is a heater to warm up the house and there is a door which opens every hour such that part of the warmth escapes.
- When the door opens the temperature in the room suddenly drops by three degrees centigrade.
- The heater has a capacity of heating the room 1 degree centigrade every 15 minutes.
- When the heater would be switched on the whole time the temperature would continue to rise by 1 degree per hour. Therefore, there is a control system, i.e., the thermostat, which switches off the heater. The thermostat uses the following rules.
  - If the temperature drops below 18, the heater is switched on.
  - If the temperature rises above 22, the heater is switched off.
CPN model of thermostat system

```
color Temp = string;
color B = unit;
color BT = B timed;
var t:Temp;
var a:B;
var b:BT;
```

```
= [t>22]
= [t<18]
```

```
switch_off
switch_on
```

```
on
1`()off
```

```
B
```

```
t
```

```
t-3t+1
```

```
t+1
```

```
heater
```

```
cool_down
```

```
BT
```

```
temp
```

```
t1`15
```

```
t@+15
```

```
t@+60
```

```
BT
```
Exercise

• Describe the room temperature in time starting in the initial state shown, i.e., play a timed, colored \``token game``.

• Extend the model such that there is a day program and a night program. From midnight to 8am, the thermostat tries to keep the temperature between 14 and 18 degrees centigrade. (If the temperature drops below 14 the heater is switched on. If the temperature rises above 18 the heater is switched off.) From 8am to midnight, the temperature is kept between 18 and 22 degrees, like before.
WARNING

It is not sufficient to understand the (process) models. You have to be able to design them yourself!
Exercise: Train system

- 7 sectors (tracks)
- 2 trains: A and B
- When moving to a new sector both this sector and the next one should be empty.
- Trains drive in one direction.

- Model as a classical Petri net.
- Model in terms of CPN without folding the tracks.
- Model as a CPN with folding the tracks (i.e., only two places).
Partially folded
Trains and tracks folded
Exercise: Philosophers

- 5 philosophers
- 5 chopsticks
- Each philosopher is either thinking or eating.
- For eating two chopsticks are needed.
- Chopsticks need to be shared among neighbors.
- Both chopsticks are taken and released at the same time.

- Model as a classical Petri net.
- Model in terms of CPN using only three places and two transitions.
Classical Petri net
Exercise: Philosophers (2)

- 5 philosophers
- 5 chopsticks
- Each philosopher is either thinking or eating.
- For eating two chopsticks are needed.
- Chopsticks need to be shared among neighbors.
- *First the right chopstick is taken. Then the second one is taken.*
- *The two chopstick are released in reversed order.*

- Model in terms of CPN.
- Are deadlocks possible?
Initial state
4 is eating, 3 took his right chopstick
Deadlock
From state space report

- **State Space**
  - Nodes: 392
  - Arcs: 1415
- One home marking
- One dead marking
Adding philosophers (n=8)

Nodes: 14158
Arcs: 81848
Exercise: Philosophers (3)

- 5 philosophers
- 5 chopsticks
- Each philosopher is either thinking or eating.
- For eating two chopsticks are needed.
- Chopsticks need to be shared among neighbors.
- *First the one chopstick (either left or right) is taken. Then the other one is taken.*
- *Also released in arbitrary order.*

- Model in terms of CPN.
- Are deadlocks possible?
State space analysis

- 1473 states
- 6270 transitions
- two dead markings
More information in tokens
- color sets, functions, etc.
- behavior may be hidden in “code”
- extreme case: all behavior folded into one place and one transition

More information in network
- possibly spaghetti networks to encode simple things
- behavior may be incomprehensible
- cannot be parameterized
- extreme case: (infinite) classical Petri net
More on functions: Recursion

• “fun fac(x:INT) = if x>1 then x*fac(x-1) else 1” is a recursive function since the function is expressed in terms of itself.
• Two cases:
  • fac(x) = x*fac(x-1)
  • fac(1) = 1
• fac(10)=10*fac(9)=10*9*fac(8)=10*9*8*fac(7)= … = 10*9*8*7*6*5*4*3*2*1 = 3628800
Recursion (1)

color Product = string;
color Number = int;
color StockItem = record prod:Product * number:Number;
color Stock = list StockItem;
fun totalstock(s:Stock) =
  if s = [ ]
  then 0
  else (#number(hd(s)))+totalstock(tl(s));
Recursion (2)

fun maxstock(s:Stock) =
    if s = [ ]
    then 0
    else if (#number(hd(s))) >= maxstock(tl(s)) then #number(hd(s))
    else maxstock(tl(s));

Instead of sum the maximum is taken

Prod:Product  Number:number
"apple"       301
"orange"      504
"pear"        423
“banana”      134
...           ...

504
fun maxstockname(s:Stock) =
  if s = []
  then "no product found"
  else if (#number(hd(s)))=maxstock(tl(s)) then #prod(hd(s))
  else maxstockname(tl(s));

<table>
<thead>
<tr>
<th>Prod:Product</th>
<th>Number:number</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;apple&quot;</td>
<td>301</td>
</tr>
<tr>
<td>&quot;orange&quot;</td>
<td>504</td>
</tr>
<tr>
<td>&quot;pear&quot;</td>
<td>423</td>
</tr>
<tr>
<td>&quot;banana&quot;</td>
<td>134</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
fun enoughstock(s:Stock,n:Number) =
  if s = []
  then []
  else if (#number(hd(s)))>= n then hd(s)::enoughstock(tl(s),n)
  else enoughstock(tl(s),n);

<table>
<thead>
<tr>
<th>Prod:Product</th>
<th>Number:number</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;apple&quot;</td>
<td>301</td>
</tr>
<tr>
<td>&quot;orange&quot;</td>
<td>504</td>
</tr>
<tr>
<td>&quot;pear&quot;</td>
<td>423</td>
</tr>
<tr>
<td>&quot;banana&quot;</td>
<td>134</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

n=400

<table>
<thead>
<tr>
<th>Prod:Product</th>
<th>Number:number</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;orange&quot;</td>
<td>504</td>
</tr>
<tr>
<td>&quot;pear&quot;</td>
<td>423</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
fun enoughstockn(s:Stock,n:Number) =
  if s = [ ]
  then 0
  else if (#number(hd(s)))>= n then 1+enoughstockn(tl(s),n)
  else enoughstockn(tl(s),n);
More on functions: Pattern matching

fun lenlist1(s: Stock) =
  if s = []
  then 0
  else 1 + lenlist(tl(s));

fun lenlist2([],) = 0 |
  lenlist2(si::s) = 1 + lenlist2(s);;

No explicit typing!!!
Pattern matching (1)

fun totalstock(s:Stock) =
if s = [ ]
then 0
else (#number(hd(s)))+totalstock(tl(s));

fun totalstock([ ] : Stock) = 0 |
  totalstock(si::s) = (#number(si))+totalstock(s);
fun maxstock(s:Stock) =
    if s=[ ]
    then 0
    else if (#number(hd(s))) >= maxstock(tl(s)) then #number(hd(s))
    else maxstock(tl(s));

fun maxstock([ ]:Stock) = 0 |
maxstock(si::s) = if (#number(si)) > maxstock(s) then #number(si)
    else maxstock(s);
fun incrs(x:StockItem,[ ]:Stock) = [x] |
  incrs (x,(si::s)) =
    if (#prod(si))=(#prod(x))
    then {prod=(#prod(si)),
          number=((#number(si))+(#number(x)))}
        ::incrs(x,s)
    else (si::incrs(x,s));

Prod:Product | Number:number
-------------|----------------
"apple"      | 301
"orange"     | 504
"pear"       | 423
"banana"     | 134
...           | ...

Prod:Product | Number:number
-------------|----------------
"apple"      | 321
"orange"     | 504
"pear"       | 423
"banana"     | 134
...           | ...
fun incrs(x:StockItem, [ ]:Stock) = [x] |
  incrs (x, (si::s)) =
  if (#prod(si))=(#prod(x))
    then {prod=(#prod(si)),
    number=((#number(si))+(#number(x)))}
    ::incrs(x,s)
  else (si::incrs(x,s));
fun reverse([ ]) = [ ] | reverse(x::y) = reverse(y)^^[x];

fun elt([ ], a) = false | elt((x::xs), a) = a=x orelse elt(xs, a);

fun del(a,[ ]) = [ ] | del(a,(x::xs)) = if a=x then xs else x::(del(a,xs));

fun intersect([ ], ys) = [ ] |
intersect(xs, [ ]) = [ ] |
intersect ((x::xs), ys) = if elt(ys,x)
    then x::(intersect(xs,(del(x,ys))))
    else intersect(xs, ys);
Example: Sudoku

colset Index = int with 0..8;
colset Cel = int with ~1..9;
colset Cels = list Cel;
colset Pos = product Index * Index;
colset Val = product Pos * Cel;
colset Sudoku = list Val;

Write an ML function to solve a Sudoku assuming that in each step there is a "deterministic candidate", i.e., no backtracking needed.
val v4 = [
[6,0,0, 0,8,0, 0,0,9],
[0,7,0, 4,0,6, 0,8,0],
[0,0,0, 5,0,1, 0,0,0],
[0,1,7, 2,0,9, 8,5,0],
[2,0,0, 0,0,0, 0,0,1],
[0,8,4, 1,0,3, 6,7,0],
[0,0,0, 3,0,8, 0,0,0],
[0,4,0, 9,0,5, 0,1,0],
[8,0,0, 0,7,0, 0,0,5]
];

fun readcell(x,i,j) = if x=[ ] then [ ] else if
hd(x) = 0 then readcell(tl(x),i,j+1) else
((i,j),hd(x))::readcell(tl(x),i,j+1);

fun readrow(x,i) = if x=[ ] then [ ] else
readcell(hd(x),i,0)^^readrow(tl(x),i+1);

fun read(x) = readrow(x,0):Sudoku;

to map “string” (list of lists)
representation to list of ((i,j),c)
values

0 values are not inserted

0 values are empty
fun dom([ ]) = [ ] | dom((x,y)::l) = x::dom(l);
fun elt([ ], a) = false | elt((x::xs), a) = a=x orelse elt(xs, a);
fun fmap([ ],z) = 0 | fmap((x,y)::l,z) = if x=z then y else fmap(l,z);
fun sdiff([ ],z) = [ ] | sdiff(x::y,z) = if elt(z,x) then sdiff(y,z) else x::sdiff(y,z);
infix sdiff;

difference of two sets
fun row([],k) = [] | row(((i,j),c)::s,k) = if i=k then c::row(s,k) else row(s,k) : Cels;

fun column([],k) = [] | column(((i,j),c)::s,k) = if j=k then c::column(s,k) else column(s,k) : Cels;

fun de(i,j) = (i div 3) = (j div 3);

fun block([],i,j) = [] | block(((i1,j1),c)::s,i,j) = if de(i,i1) andalso de(j,j1) then c::block(s,i,j) else block(s,i,j) : Cels;

val uni = [1,2,3,4,5,6,7,8,9] : Cels;

fun free(s,i,j) = ((uni sdiff row(s,i)) sdiff column(s,j)) sdiff block(s,i,j) : Cels;

val all cell values in row k

val all cell values in column k

val same block

val values in the block containing (i,j)

val all cell values in row k

val all values

val remaining options
fun allpos() = Pos.all();
fun undef(s) = allpos() sdiff dom(s);
fun analyze1(s,[]) = [] | analyze1(s,(i,j)::l) = ((i,j),free(s,i,j))::analyze1(s,l);
fun analyze(s) = analyze1(s,undef(s));
fun new([]) = [] | new(((i,j),[])::s) = ((i,j),~1)::new(s) | new(((i,j),[c])::s) = ((i,j),c)::new(s) | new(((i,j),c::cs)::s) = new(s);

fun solve(s) =
if new(analyze(s)) = []
then s
else solve(new(analyze(s))^^s);

add error entry (no options left)
add entry with just one possible move c
skip if multiple moves possible
repeatedly call solve until no entries can be added (done or non-deterministic choice needed)
fun sord(((x1,y1),z1),((x2,y2),z2)) = (x1 < x2) orelse
(x1=x2 andalso y1 < y2);
fun solver(s) =  sort sord (solve(s));

sort function is built in
“sort lt_fun l” sorts list l using the function lt_fun to
determine when one element in the list is less than
another.
See http://cpntools.org/documentation/concepts/colors/declarations/colorsets/list_colour_sets
fun result1(s,i) = if i>= 81 then "---------------------\n" else 
  (Int.toString(fmap(s,(i div 9, i mod 9))) ^ (if i mod 9 = 8 then "\n" else " " )^ result1(s,i+1));

fun result(s) = 
  "\n---------------------\n"^result1(solver(s),0);
```
val v5

*** replace v1 by v2, v3, v4 or your own sudoku ***

result(read(v5))

[1,2,3,4,5,6,7,8,9] sdiff [2,3]
```

```
fun dom([]) = [] | dom([(x,y):l]) = x::dom(l);
fun elt([], a) = false | elt([(x,y):l], a) = a=x | else elt(l, a);
fun fmap([], z) = 0 | fmap([(x,y):l], z) = if x=z then y else fmap(l, z);
fun sdiff([], z) = [] | sdiff([(x,y):l], z) = if elt(l, z) then sdiff(l, z) else x::sdiff(y, z);

fun readcell(x, y, j) = if x=j then [] else if hd(x) = 0 then readcell(tl(x), j+1) else ((i, j), hd(x))::readcell(tl(x), i+1);

fun readrow(x, i) = if x=[i] then [] else readcell(hd(x), i, 0)~~readrow(tl(x), i+1);

fun readcol(y, j) = readrow(0, y, j);

fun readrow([], y, k) = if y=k then c::row(s, k) else row(s, k) : Cels;

fun column([], k) = [] | column([(i, j):l], k) = if j=k then c::column(s, k) else column(s, k) : Cels;

fun da(i, j) = (i div 3) = (j div 3);

fun block([], i, j) = [] | block([(i, j), c:[]], i, j) = if da(i, j) andalso da(i, j) then c::block(s, i, j) else block(s, i, j) : Cels;

fun free(s, i, j) = (undi row(s)) sdiff column(s) sdiff block(s, i, j) : Cels;

fun alnpos() = Pos.all();

fun undef(s) = alnpos() sdiff dom(s);

fun analyze1([], i) = [] | analyze1([(i, j):l]) = (i, j, free(s, i, j))::analyze1(s, l);

fun analyze(s) = analyze1(s, undef(s));

fun new1([], i) = [] | new1([(i, j):l]) = (i, j, new(s))::new1(s);

fun new([], i) = new1([], i) sdiff [];

fun solve() = if new1(analyze(s)) = [] then s else solve(new1(analyze(s))~~s);

fun sord1([(x1, y1), z1], [(x2, y2), z2]) = (x1 < x2) orelse (x1=x2 andalso y1 < y2);

fun solver(s) = sord sord1(solve(s));

fun result1(s) = if s > 81 then "-------------------------\n" else (Int.toString(fmap(s, i div 9, i mod 9))) ^ (if i mod 9 = 8 then "n" else " ") ^ result1(s, i+1);

fun result(s) = "\n-------------------------\n"\nresult1(solver(s), 0);
```
val v5 = [
[0,2,0,5,0,0,0,3,0],
[0,0,5,2,0,9,0,6,1],
[0,9,3,0,0,7,5,0,0],
[7,0,0,9,4,0,3,0,0],
[3,4,0,0,0,0,0,9,6],
[0,0,2,0,1,8,0,0,5],
[0,0,7,8,0,0,6,5,0],
[6,1,0,4,0,2,7,0,0],
[0,8,0,0,0,3,0,1,0]
];
More information

• About Standard ML:
  • http://www.standardml.org/Basis/ (for functions)

• About CPN: