Design/CPN
Overview of CPN ML Syntax
Version 3.0
Design/CPN
Overview of CPN ML Syntax
Version 3.0

Table of Contents

Chapter 1
Colour Sets
Relational Operations..............................................................5
Simple Colour Sets.................................................................5
Compound Colour Sets.........................................................7
Declare Clause........................................................................9

Chapter 2
Multi-sets
Operations on Multi-Sets.....................................................11
Timed Simulations................................................................12

Chapter 3
Miscellaneous
Identifiers...........................................................................13
Values..................................................................................13
Variables.............................................................................13
Reference Variables...........................................................14
Functions.............................................................................14
Chapter 1
Colour Sets

Colour sets can be declared in many different ways, but they are all in some way constructed from basic SML-types. This means that the colour sets automatically inherit a number standard functions and operations.

Relational Operations

The equality operators = and <> are defined for all colour sets, while <, >, <= and >= only are defined for integers, reals and strings. To test the order of the elements in other colour sets, use the lt function (see section on declare clause).

Simple Colour Sets

Unit colour sets

color name = unit [with new_unit];
Order: trivial

Boolean colour sets

color name = bool [with (new_false, new_true)];
Order: false before true
Operations:
not b  negation of the boolean value b
b1 andalso b2  boolean conjunction, and
b1 orelse b2  boolean disjunction, inclusive or
Overview of CPN ML Syntax

**Integer colour sets**

\[ \text{color name} = \text{int} \ [\text{with } \text{int-exp}_1..\text{int-exp}_j]; \]

Order: usual ordering of numbers

Operations:

- \(~i\) negation of the integer value \(i\)
- \(i_1 + i_2\) addition
- \(i_1 - i_2\) subtraction
- \(i_1 * i_2\) multiplication
- \(i_1 \div i_2\) division, quotient
- \(i_1 \mod i_2\) modulus, remainder
- \(\text{abs } i\) absolute value of \(i\)
- \(\text{min } (i_1, i_2)\) minimum of \(i_1\) and \(i_2\)
- \(\text{max } (i_1, i_2)\) maximum of \(i_1\) and \(i_2\)

**Real colour sets**

\[ \text{color name} = \text{real} \ [\text{with } \text{real-exp}_1..\text{real-exp}_j]; \]

Order: usual ordering of numbers

Operations:

- \(~r\) negation of the real value \(r\)
- \(r_1 + r_2\) addition
- \(r_1 - r_2\) subtraction
- \(r_1 * r_2\) multiplication
- \(r_1 / r_2\) division
- \(\text{sqrt } r\) square root
- \(\text{abs } r\) absolute value
- \(\text{min } (r_1, r_2)\) minimum of \(r_1\) and \(r_2\)
- \(\text{max } (r_1, r_2)\) maximum of \(r_1\) and \(r_2\)
- \(\text{floor } r\) convert real to integer
- \(\ln r\) natural logarithm
- \(\exp r\) exponential
- \(\sin r\) sine
- \(\cos r\) cosine
- \(\tan r\) tangent
- \(\text{arctan } r\) arc tangent
- \(\text{real } i\) convert integer \(i\) to real value
### Colour Sets

#### String Colour Sets

```ml
color name = string [with string-exp_i..string-exp_j 
[and int-exp_min..int-exp_max]];
```

**Order:** lexicographic (with the ascii ordering)

**Operations:**

- `s_1 ^ s_2`: concatenate the strings `s_1` and `s_2`
- `size s`: number of characters in `s`
- `substring (s,i,l)`: extract a substring of length `l` starting at position `i` in `s`, first position is 0
- `expode s`: convert string `s` to list of one character strings
- `implode l`: convert list `l` of strings to a string
- `ord s`: ordinal value of first character of `s`
- `ordof (s,i)`: ordinal value of the `i`th character first position is 0
- `chr i`: single-character string from ordinal value `i`

#### Enumerated Colour Sets

```ml
color name = with id_1 | id_2 | ... | id_n;
```

**Order:** as in the declaration

#### Index Colour Sets

```ml
color name = index id with int-exp_min..int-exp_max;
```

**Order:** usual ordering on the indexes

#### Compound Colour Sets

#### Product Colour Sets

```ml
color name = product name_1 * name_2 * ... * name_n;
```

**Order:** lexicographic (with respect to ordering of base colour sets)

**Values:** `(v_1, v_2, ..., v_n)`

**Operations:**

- `#i`: extract the `i`th element of tuple (does not work for the Edinburgh ML compiler)
- `_`: omit component in tuple (not allowed in CPN inscriptions)
Overview of CPN ML Syntax

Record colour sets

\[
\text{color name} = \text{record } \text{id}_1: \text{name}_1 * \text{id}_2: \text{name}_2 * \ldots * \text{id}_n: \text{name}_n;
\]

Order: lexicographic (with respect to ordering of base colour sets)

Values: \{id_1 = v_1, id_2 = v_2, \ldots, id_n = v_n\}

Operations:
- #\text{id}_i: extract the id_i-element from the record
- \ldots: omit field in record (not allowed in CPN inscriptions)

List colour sets

\[
\text{color name} = \text{list } \text{name}_0 [\text{with int-exp}_{\text{min}}..\text{int-exp}_{\text{max}}];
\]

Order: lexicographic (with respect to ordering of base colour set)

Values: \[v_1, v_2, \ldots, v_n\]

Operations:
- nil: empty list (same as [])
- e::l: prepend element e in head of list l
- l_1 ^ ^ l_2: concatenate the two lists l_1 and l_2
- hd l: head, the first element of the list
- tl l: tail, list with exception of first element
- length l: length of list
- nth (l, n): nth element in list
- nthtail (l, n): remove first n elements of list
- rev l: reverse list
- exists p l: true if p is true for some element in list
- null l: true if list is empty
- map f l: use function f on each value of list and returns a list with all the results
- app f l: use function f on each value of list and returns ()
- fold f l z: returns \(f(l_1, f(l_2, \ldots f(l_n, z) \ldots))\)
  where \(l = [l_1, l_2, \ldots, l_n]\)

Union colour sets

\[
\text{color name} = \text{union } \text{id}_1[: \text{name}_1] + \text{id}_2[: \text{name}_2] + \ldots + \text{id}_n[: \text{name}_n];
\]

Order: first after selectors, then after ordering of each base colour set

Values: id_i(v)
Colour Sets

Subset colour sets

color name = subset name_0 by subset-function;
Order: ordering of base colour set

color name = subset name_0 with subset-list;
Order: ordering of base colour set

Alias colour sets

color name = name_0
Order: ordering of the base colour set

Declare Clause

The declare clause is appended to the end of the colour set declaration. It makes predefined system constants, operations and functions available.

color name = .... declare id_1, id_2, ..., id_n

All colour sets:

all declare all functions available for colour set
ran'cs() returns a random value
lt'cs(v_1, v_2) less than in the colour set ordering
mkst_col'cs(v) make string representation of a colour
mkst_ms'cs(ms) make string representation of a multi-set

Small colour sets

ms multi-set with one of each element
size'cs number of elements in the colour set
first'cs first element in the colour set
last'cs last element in the colour set

Enumerated and indexed colour sets

ord'cs(i) convert value to number representing its position
col'cs(v) convert from number representing its position
dist'cs(v_1, v_2) distance between two values
rot'cs(i, v) value obtained by rotating i indexes from v
Overview of CPN ML Syntax

Indexed colour sets

\[ \text{index'cs(v(i))} \quad \text{convert identifier-value to index number} \]
\[ \text{clr'cs(i)} \quad \text{convert index number to identifier-value} \]

Product and record colour sets

\[ \text{mult'cs(ms}_1\ldots\text{ms}_n) \quad \text{product of multi-sets} \]

Subset colour sets

This includes int, real and string using the with clause.

\[ \text{in'cs(v)} \quad \text{test whether value is member of colour set} \]

Alias colour sets

\[ \text{same} \quad \text{declare same functions as for base colour set} \]

Union colour sets

\[ \text{of_id}_i'cs(cs) \quad \text{test whether value belongs to the component id}_{i} \]
Chapter 2
Multi-sets

Multi-sets are declared over colour sets. The back-quote (´) operator is the multi-set constructor (as an example 3´7 is the multi-set with three appearances of the colour 7).

Operations on Multi-Sets

- \( m_s_1 == m_s_2 \): multi-set equality.
- \( m_s_1 <> m_s_2 \): multi-sets inequality.
- \( m_s_1 > > m_s_2 \): multi-set greater than.
- \( m_s_1 > > = m_s_2 \): multi-set greater than or equal to.
- \( m_s_1 << m_s_2 \): multi-set less than.
- \( m_s_1 << = m_s_2 \): multi-set less than or equal to.
- \( m_s_1 + m_s_2 \): multi-set addition.
- \( m_s_1 - m_s_2 \): multi-set subtraction (\( m_s_2 \) must be less than or equal to \( m_s_1 \)).
- \( c * m_s \): scalar multiplication.
- \( \text{size } m_s \): size of Multi-set.
- \( \text{random } m_s \): returns a pseudo random colour from \( m_s \).
- \( \text{cf} (c, m_s) \): returns the number of appearances of colour \( c \) in \( m_s \).
- \( \text{filter } p \ m_s \): takes a predicate \( p \) and a multi-set \( m_s \) and produces the multi-set of all the appearances in \( m_s \) satisfying the predicate.
- \( \text{ext}_\text{col} f \ m_s \): takes a function \( f \) and a multi-set \( c_1 \ s_1 + c_2 \ s_2 + \ldots + c_n \ s_n \) and produces the multi-set \( c_1 \ f(s_1) + c_2 \ f(s_2) + \ldots + c_n \ f(s_n) \).
- \( \text{ext}_\text{ms} f \ m_s \): takes a function \( f \) and a multi-set \( c_1 \ s_1 + c_2 \ s_2 + \ldots + c_n \ s_n \) and produces the multi-set \( c_1 \ * f(s_1) + c_2 \ * f(s_2) + \ldots + c_n \ * f(s_n) \).
Timed Simulations

To simulate with time choose *With Time* in **Simulation Code Options** and specify whether time should be measured in integer or reals.

A colour set is timed by appending the keyword timed to the end of its declaration:

```plaintext
    color name = ........ timed;
```

All simple colour sets are by default untimed while compound colour sets are timed if and only if at least one of the base colour sets are timed. To make a timed colour set untimed, append the keyword untimed to the end of its declaration.

The time stamp is added to the multi-set value by adding the at-sign (@) and an integer list specifying the time stamps, e.g., 3`e@[2,4,6]. In the output arc inscriptions the time delay expression consists of the keyword @+ followed by an integer or real expression, e.g. 1`e@+5. Input arc inscriptions are not allowed to specify time delays. The keyword @ignore in an input arc inscription causes the simulator to ignore that the colour set is timed, e.g. 1`e@ignore.

The current model-time and step number can be inspected by means of the function `time()` and `step()` respectively. The function `with_time()` tests whether or not the simulation is with time.
Chapter 3
Miscellaneous

Identifiers
An identifiers is a sequence of letters, digits, primes, and underscores – starting with a letter.

Values
A value declaration binds a value to an identifier (which then works as a constant)

   \[ \text{val id = exp;} \]

Variables
A variable is bound to a value, the scope of a variable is local to the transition. If the variable is from a colour set with less than 100 elements, the simulator is always able to bind a value to it.

   \[ \text{var id_1, id_2, \ldots, id_n: cs\_name;} \]
Overview of CPN ML Syntax

Reference Variables

A reference variable is similar to a pointer in C. The references may only be used in code segments. Do not use references in a way that effects transition enabling.

globref id = exp;  a global reference variable can be declared in global and temporary declaration nodes, the scope is the entire CP-net.
pageref id = exp;  a page reference variable can be declared in a local declaration nodes, the scope is all instances of the page.
instref id = exp;  an instance reference variable can be declared in a local declaration nodes, the scope is a single instance of the page.

Operations:

<table>
<thead>
<tr>
<th>!r</th>
<th>contents of the reference r</th>
</tr>
</thead>
<tbody>
<tr>
<td>r := v</td>
<td>assignment of the value v to the reference r</td>
</tr>
<tr>
<td>ref v</td>
<td>reference constructor</td>
</tr>
<tr>
<td>inc r</td>
<td>increment contents of integer reference r</td>
</tr>
<tr>
<td>dec r</td>
<td>decrement contents of integer reference r</td>
</tr>
</tbody>
</table>

Functions

fun id pat1 = exp1
  | id pat2 = exp2
  | ........
  | id patn = expn;

The _ (underscore) can be used to omit fields in the pattern. As an example look at the following function. It is a function with two parameters a constant and a list, and it multiplies each entry in the list with the constant, returning the result.

fun list_mult (c, x::xs) = (c * x)::lmult(c, xs)
  | list_mult (_., nil) = nil

To turn a function f (with two parameters) into an infixed operator write:

infix f;
**Local Declarations**

```ml
let
  val pat_1 = exp_1;
  val pat_2 = exp_2;
  ........
  val pat_n = exp_n
in
  exp
end;
```

**Control Structures**

```ml
if bool-exp then exp_1 else exp_2;

case exp of
  pat_1 => exp_1
  | pat_2 => exp_2
  ........
  | pat_n => exp_n;
```