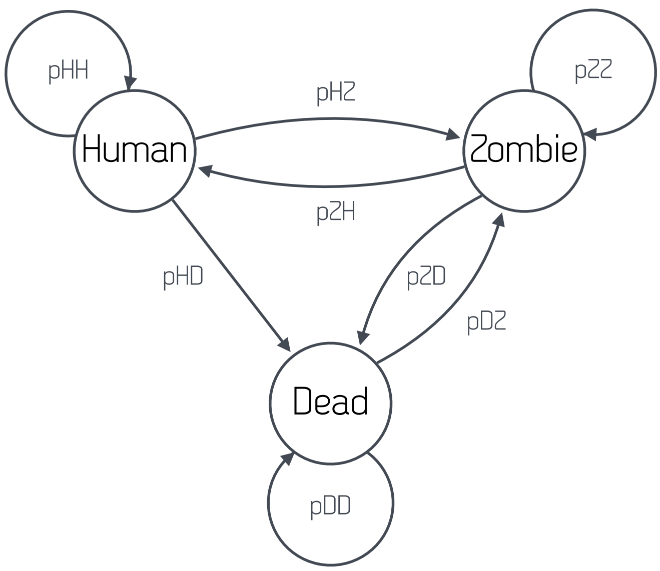
Internal Release

Extensions in PharmML 0.8 & 0.8.1

Authors:
Maciej J Swat
Florent Yvon
Sarala Wimalaratne

with contributions from:
Roberto Bizzotto
Emmanuelle Comets
Gareth Smith



March 16, 2016

Contents

Chapter 1

Overview

This document describes extensions and changes in PharmML between versions 0.7.3 & 0.8 and 0.8 & 0.8.1. The evolution of PharmML since the project start are visualised in Figure 1.1.

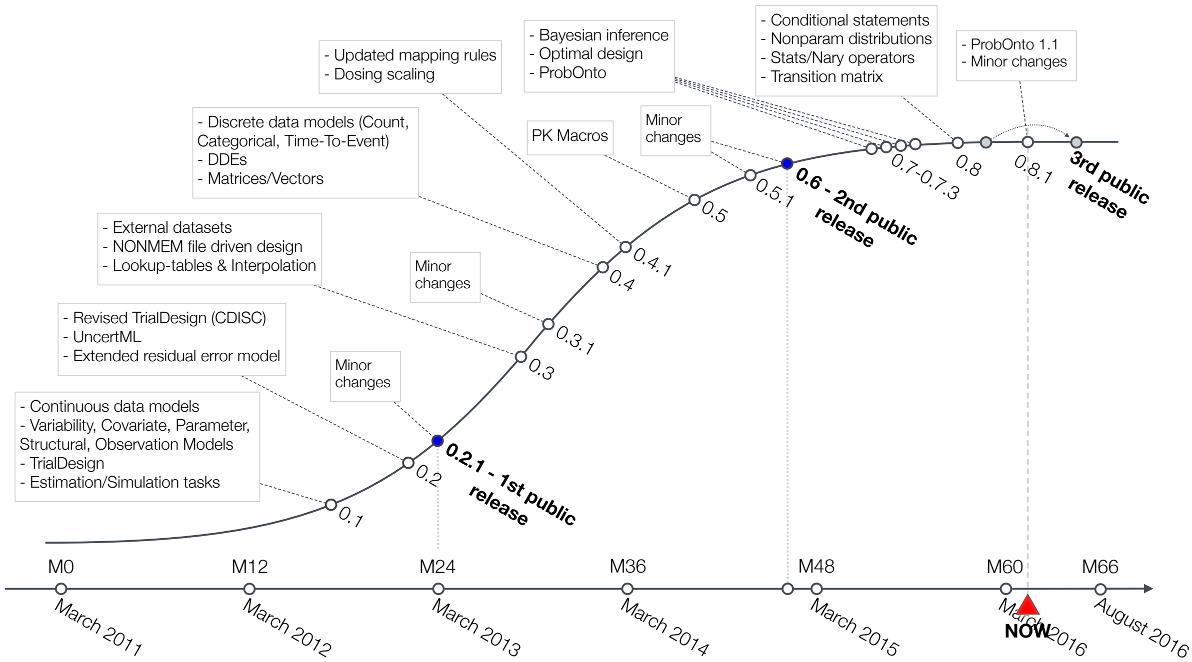


Figure 1.1: PharmML release history with major extensions shown of each version.

1.1 Summary of changes/extensions in 0.8

The following table summarises the major changes in version 0.8 compared to 0.7.3 described in detail in following chapters.

PharmML element or modelling aspect	version ≤ 0.7.3	version 0.8
Assignment statements Section 2.2	not supported	NEW `<AssignStatement>` element available in covariate, parameter, structural and observation models
Independent variable Section 2.2.2	only one allowed	multiple `<IndependentVariable>` elements allowed
Conditional statements Section 2.3	not supported	NEW `<ConditionalStatement>` element with child elements `<If>`, `<ElseIf>` and `<Else>`

Parameter, (random) variables	`symbId` only	additionally also `symbIdRef` available when referencing required
Parameters, Section 2.13	`<SimpleParameter>` not available in v0.7-0.7.3	restored and available as `<Parameter>`
Nested piecewise Section 2.4	Simple piecewise	supported
Probability functions Section 2.5	not supported	NEW `<PDF>`, `<CDF>`, `<HF>`, `<SF>` elements for every continuous distribution available in ProbOnto
Time-to-event data models Section 2.5.3	explicit hazard/survival fcts formulas required	encodable using `<HF>`/`<SF>` (see above)
Empirical distribution Section 2.6	not supported	NEW `RandomSample`, `SystematicSample` `UnknownSample` and `weight` parameter accessible via ProbOnto
Random realisations Section 2.7	not supported	NEW `<Realisation>` for all distributions featured in ProbOnto
Markov models Section 2.8	only pair-wise transition probabilities	NEW `<TransitionMatrix>` element with `type` attribute: *leftStochastic*, *rightStochastic* and *doublyStochastic*
N-ary operators Section 2.9	not supported	NEW `plus`, `times`, `min`, `max`, `gcd`, `lcm`
Statistical operators Section 2.10	not supported	NEW `centredMoment`, `coefficientOfVariation`, `correlation`, `decile`, `geometricMean`, `kurtosis`, `mean`, `median`, `mode`, `moment`, `percentile`, `quantile`, `quartile`, `range`, `skewness`, `standardDeviation`, `variance`
Covariate model Section 2.11	covariate type not annotated	NEW attribute `type` with values *occasionDependent*, *timeDependent*, *constant*
Categorical covariate model Section 2.12	defining distribution only with `<Probability>` element	– full distribution support via ProbOnto – covariates declaration/assignment – conditional distributions allowed
Dataset Section 2.14.2	single ignore character declaration only	conditional ignoring statements

Table 1.1: Overview of major differences between versions 0.8 and 0.7.3

1.2 Summary of changes/extensions in 0.8.1

PharmML element or modelling aspect	version $\leq$ 0.8	version 0.8.1
Setting and output files 4.1	no support	NEW `<SoftwareSettings>` added NEW `<OutputFile>` added
Dataset definition section 4.3.2	missing or redundant values	`columnType` with new values *varLevel*, *variance*, *stdev*, *mode*, *median* NEW – removed *varParameter_corr*, *varParameter_cov*, *varParameter_stdev*, *varParameter_var* – multiple values allowed
– for use in SO only see [Terranova et al., 2016]	not supported	NEW `level` attribute
Optimal Design Section 4.2	missing stage support	NEW `<StageDefinition>` element with `<Stage>` child element

	missing params to estimate support in OD	NEW `<ParametersToEstimate>`
	FIM type missing	– NEW `type` with values {B, I, P} – `<Matrix>` support removed
Variables section 4.5	attribute mandatory	`symbolType` attribute is optional
Integer declaration section 4.5	types used: `int`, `integer` or `positiveInteger`	`int` only
Piecewise declaration section 4.5	occurring in many namespaces	declared in `math` namespace only
Conditional statements Section 4.6	ODEs not supported	ODEs allowed
Functions Section 4.7	not supported	NEW `gamma`

Table 1.2: Overview of major differences between versions 0.8.1 and 0.8

Chapter 2

Changes in 0.8

2.1 Backwards compatibility with 0.7.3

The 0.8 version is backwards compatible with 0.7.3 as tested on

- more then 80 examples provided with each release and
- new 14 IOG use cases[1] publicly released with the Product 4.1.

This means that no elements/attributes have been removed, renamed or otherwise modified in a way which would make 0.7.3 models invalid with respect to the new schema and specification.

2.2 Assignment statements

So far assignments were performed together with declarations. The new `<AssignStatement>` element allows to loosen up this structure and do it more flexibly separately. For example the following snippet shows how to define a parameter, k, and subsequently how to assign an expression, CL/V, to it.

option supported so far	new option in version 0.8
<!-- declaration & assignment --> <IndividualParameter symbId="k"> <ct:Assign> <math:Binop op="divide"> <ct:SymbRef symbIdRef="CL"/> <ct:SymbRef symbIdRef="V"/> </math:Binop> </ct:Assign> </IndividualParameter>	<!-- declaration --> <IndividualParameter symbId="k"/> <!-- assignment, k=CL/V --> <ct:AssignStatement op="eq"> <!-- LHS --> <ct:SymbRef symbIdRef="k"/> <!-- RHS --> <math:Binop op="divide"> <ct:SymbRef symbIdRef="CL"/> <ct:SymbRef symbIdRef="V"/> </math:Binop> </ct:AssignStatement>

Table 2.1: Assignment statements are new in v0.8.

The assignment element is available in the covariate, parameter, structural and the observation models. It can be applied generally but will turn out to be especially useful in connection with the new conditional assignment statements, see section 2.3.

2.2.1 Assignment rules

- One assignment per `<AssignStatement>` is allowed.
- Left and right hand sides of the assignment can be any expressions which evaluate to scalar values.

[1] `http://sourceforge.net/projects/ddmore/files/install/SEE/Demonstrator-1.2.0/resources/converter-systemtest-1.3.0-results.zip/download`

- Parameters and variables used within the assignment statements must the declared before, see for an example Table 2.1 (right).
- Only one assignment of each parameter or variable is allowed within a model, unless used in a conditional statement, see section 2.3.3.

2.2.2 New opportunities

The introduction of assignment statements opens new opportunities to encode models relevant e.g. for studying of drug-drug interactions. While the majority of such models was implementable in previous versions, the so-called *Greco-model*, [Greco et al., 1990], shown below is treatable in PharmML only thanks to this extension

$$\frac{d_1}{\mu_1\left[\frac{E}{E_c-E}\right]^{1/m_1}} + \frac{d_2}{\mu_2\left[\frac{E}{E_c-E}\right]^{1/m_2}} + \frac{\alpha\, d_1 d_2}{\mu_1\mu_2\left[\frac{E}{E_c-E}\right]^{(1/2m_1+1/2m_2)}} = 1$$

This is a pharmacodynamic model for a combined effect, E, a function of two drug doses, d_1 and d_2, which does not have a *closed* form. In other words the effect cannot be expressed directly in terms of d_1 and d_2 as a function $E = f(d_1, d_2, \dots)$. The full PharmML code for this model is provided with this release in the example folder *others*.

It is an interesting model for another reason as well. It is one which requires the declaration of two independent variables, d_1 and d_2. Thus the limitation to only one `<IndependentVariable>` has been removed.

2.3 Conditional statements

Conditional assignments are new in this version and provide additional flexibility to the language. So far only piecewise statements were supported which have the drawback that only single variables are assigned per statement.

Conditional assignments don't replace the existing piecewise statements, they are an entirely independent structure.

2.3.1 Tool coverage – capabilities and restrictions

The extend to which various target tools cover conditional assignments differs and posses a problem for the interoperability. Below we summarise the support and scope of these assignments in the major target tools.

- `MLXTRAN`, supports nested conditional `if-elseif-else-end` statements, e.g.

```
if (TIME <=10 && ID == 1)
        G = a1
elseif (TIME > 10 && ID == 1)
        G = a2
elseif
         ...
end
```

  with following features as listed in the language manual, [Lixoft Team, 2014a],

  - Several `elseif` keywords can be chained, and the conditions are exclusive in sequence.
  - A default value can be provided using the keyword `else`, but also as a simple definition preceding the conditional structure
  - Only intermediate variables can be defined within conditional statements, the structure of the model cannot depend on such conditions.
  - The derivatives of an ODE, or the PK elements of a prediction sub-model, cannot be defined within conditionals, e.g. the following is a valid MLXTRAN code

```
if t > T_end
        coeff = 0
else
        coeff = 1
```

```
end
eta_cond = coeff*eta
eps_cond = coeff*epsilon
ddt_TC = s - d*TC - beta*(1-eta_cond)*TC*VL
ddt_IC = beta*(1-eta_cond)*TC*VL - delta*IC
ddt_VL = (1-eps_cond)*p*IC - c*VL
```

- `NMTRAN`, [Beal et al., 2009], allows nested conditional statements with the only restriction known to us
  - NMTRAN doesn't accept nested conditional statements including random variables - described in 'NONMEM Users Guide - Part V', page 79, NONMEM 7.2.
- `winBUGS`, [Lunn et al., 2009]
  - supports only a step function of the form

    ```
    step(e) 1 if e >= 0; 0 otherwise
    ```

    where 'e' any expression
  - Note however, that according to the winBUGS translation team, in principle, thanks to the Pascal language interface, virtually any nested conditional statement can be implemented for the execution in winBUGS.
- Common Converter, provided by Gareth Smith (Cyprotex), is able to handle (nested) conditional statements.
- `STAN`, [STAN Development Team, 2015], the potential future target tool for the DDMoRe interoperability platform offers unrestricted support for the conditional statements.

2.3.2 Definition

The general form of the (nested) conditional statement looks as follows.

if (condition1) **then**
  statement1
else if (condition2) **then**
  statement2
else if (conditionN-1) **then**
  statementN-1
else
  statementN
end if

The following definition of the conditional statement has been adapted from the STAN specification v2.9.0, [STAN Development Team, 2015]:
There must be a single leading `if` clause, which may be followed by any number of `else if` clauses, all of which may be optionally followed by an `else` clause. Each condition must be a TRUE or FALSE value. Nested `if-then-else` as part of each statement are allowed.

The entire sequence of `if-then-else` clauses forms a single conditional statement for evaluation. The conditions are evaluated in order until one of the conditions evaluates to a TRUE value, at which point its corresponding statement is executed and the conditional statement finishes execution. If none of the conditions evaluates to a TRUE value and there is a final `else` clause, its statement is executed.

2.3.3 Rules for the use of conditional statements

There are few rules to follow when using conditional constructs in PharmML coded models

1. The conditionals are available in the covariate, parameter, structural and the observation models, similarly to the *assignment statements*, described in section 2.2.
2. Allowed statement elements are

- individual and population parameters
- variables and random variables
- assignment statements
- nested conditional statements

3. Parameters, variables assigned and/or referred to in a conditional statement must be declared outside the conditional in the first level of the according model block.

4. Parameters or variables assigned within the conditional statement have to be referred to using the attribute **symbIdRef** (new in this version, not available for parameters or variables so far).
   In the following example the attribute **symbId** is used for the declaration of a population parameter

```
<!-- 1. DECLARE 'k' using symbId-->
<PopulationParameter symbId="k"/>
```

   but a subsequent assignment requires the **symbIdRef** attribute

```
<!-- 2. ASSIGN 'k' using symbIdRef -->
<ConditionalStatement>
    <math:If>
        <math:Condition>
            <math:LogicBinop op="leq">
                <ct:SymbRef symbIdRef="t"/>
                <ct:Real>10</ct:Real>
            </math:LogicBinop>
        </math:Condition>
        <PopulationParameter symbIdRef="k">
            <ct:Assign>
                <ct:Real>10</ct:Real>
            </ct:Assign>
        </PopulationParameter>
    </math:If>
    <math:Else>
        <PopulationParameter symbIdRef="k">
            <ct:Assign>
                <ct:Real>5</ct:Real>
            </ct:Assign>
        </PopulationParameter>
    </math:Else>
</ConditionalStatement>
```

5. Parameters or variables assigned within the conditional statement cannot be assigned elsewhere unless such assignment is coupled with its declaration, e.g.

```
<PopulationParameter symbId="k">
    <ct:Assign>
        <ct:Real>10</ct:Real>
    </ct:Assign>
</PopulationParameter>
```

   The assignment of a parameter or variable within a conditional statement has precedence over an assignment coupled with a declaration.

6. Neither PharmML nor libPharmML do process the conditional statements in any way. The order of statements encoded in PharmML will be preserved when translating to a target tool.

7. libPharmML doesn't have the capability to evaluate conditional statements – it is the responsibility of the user to define them in a meaningful and unambiguous way (the condition domains defined should be mutually exclusive).

The application of these rules is shown in the following example section.

2.3.4 Examples

Example 1

The first example underlines how important is to analyse the conditions and to make sure that they are mutually exclusive. Consider the following example

if $CLCR > 0$ **then**
    RF = CLCR/6
else if $AGE \leq 20$ **then**
    RF = 1
else if $AGE \geq 80$ **then**
    RF = 0.3
else
    RF = -99
end if

It follows from the above formulation, also visualised in Figure 2.1, that the condition domains are overlapping (are not mutually exclusive). The sequence of the first two conditions determines the value assigned to variable RF (consider for example CLCR=3 and AGE=10).

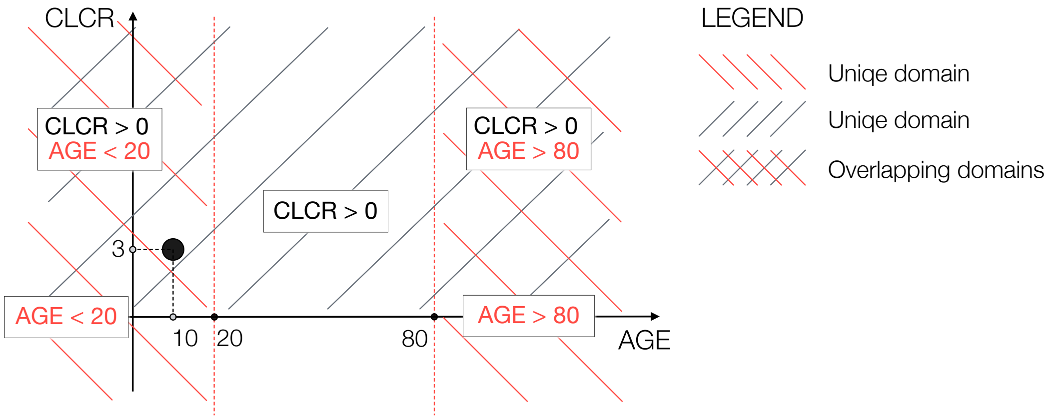


Figure 2.1: Domains for the conditions should be disjoint to avoid ambiguous assignments or the assignments have to be identical if domains overlap. For example for CLCR=3 and AGE=10 the assignment of RF depends on the sequence of the first two conditions. Because any domain declaration is allowed in PharmML and no automatic validity check is performed, such declarations are ultimately the responsibility of the modeller.

Example 2

The following conditional assignment

if $(Group = 1)$ **then**
    $ka = \theta 1 * \exp(\eta_1)$
    $ALAG1 = \theta_3$
else if $(Group = 2)$ **then**
    $ka = \theta_3 * \exp(\eta_1)$
    $ALAG1 = 0$
end if

assumes that ka and $ALAG1$ are declared first, i.e.

```xml
<!-- declaration -->
<IndividualParameter symbId="ka"/>
<IndividualParameter symbId="ALAG1"/>
```

and then the conditional assignments can be specified

```xml
<!-- conditional assignments -->
<ConditionalStatement>
    <math:If>
        <math:Condition>
            <math:LogicBinop op="eq">
                <ct:SymbRef symbIdRef="Group"/>
                <ct:Int>1</ct:Int>
            </math:LogicBinop>
        </math:Condition>
```

```
                <ct:AssignStatement op="eq">
                    <ct:SymbRef symbIdRef="ka"/>
                    <math:Binop op="times">
                        <ct:SymbRef symbIdRef="theta1"/>
                        <math:Uniop op="exp">
                            <ct:SymbRef symbIdRef="eta1"/>
                        </math:Uniop>
                    </math:Binop>
                </ct:AssignStatement>
                <ct:AssignStatement op="eq">
                    <ct:SymbRef symbIdRef="ALAG1"/>
                    <ct:SymbRef symbIdRef="theta3"/>
                </ct:AssignStatement>
            </math:If>
            <math:ElseIf>
                <math:Condition>
                    <math:LogicBinop op="eq">
                        <ct:SymbRef symbIdRef="Group"/>
                        <ct:Int>2</ct:Int>
                    </math:LogicBinop>
                </math:Condition>
                <ct:AssignStatement op="eq">
                <!-- omitted assignments for ka and ALAG1 -->
                </ct:AssignStatement>
            </math:ElseIf>
        </ConditionalStatement>
```

Example 3

This example, from Fisher/Shafer NONMEM course, [Fisher and Shafer, 2007], shows parameter assignment conditional on a categorical covariate.

```
IF (SEX.EQ.0) THEN
        V= THETA(1) * EXP(ETA(1)) ; volume in men
ELSE
        V= THETA(2) * EXP(ETA(1)) ; volume in women
ENDIF
```

Declare first the covariate, Sex

```
<CovariateModel blkId="cm1fdsfe">
    <Covariate symbId="Sex">
        <Categorical>
            <Category catId="1">
                <ct:Name>Female</ct:Name>
            </Category>
            <Category catId="0">
                <ct:Name>Male</ct:Name>
            </Category>
        </Categorical>
    </Covariate>
</CovariateModel>
```

and condition on it the assignment of the individual parameter, V. Note that parameter V needs to be declared first.

```
<ParameterModel blkId="pm1">
    <IndividualParameter symbId="V"/>

    <PopulationParameter symbId="theta1"/>
    <PopulationParameter symbId="theta2"/>
```

followed by the conditional assignments

```
    <RandomVariable symbId="eta1">
        <ct:VariabilityReference>
            <ct:SymbRef symbIdRef="indiv"/>
        </ct:VariabilityReference>
        <!-- Distribution omitted -->
    </RandomVariable>
    <ConditionalStatement>
        <math:If>
```

```
                <math:Condition>
                    <math:LogicBinop op="eq">
                        <ct:SymbRef symbIdRef="Sex"/>
                        <ct:CatRef catIdRef="0"/>
                    </math:LogicBinop>
                </math:Condition>
                <IndividualParameter symbIdRef="V">
                    <StructuredModel>
                        <PopulationValue>
                            <ct:Assign>
                                <ct:SymbRef symbIdRef="theta1"/>
                            </ct:Assign>
                        </PopulationValue>
                        <RandomEffects>
                            <ct:SymbRef symbIdRef="eta1"/>
                        </RandomEffects>
                    </StructuredModel>
                </IndividualParameter>
            </math:If>
            <math:Else>
                <IndividualParameter symbIdRef="V">
                    <StructuredModel>
                        <PopulationValue>
                            <ct:Assign>
                                <ct:SymbRef symbIdRef="theta2"/>
                            </ct:Assign>
                        </PopulationValue>
                        <RandomEffects>
                            <ct:SymbRef symbIdRef="eta1"/>
                        </RandomEffects>
                    </StructuredModel>
                </IndividualParameter>
            </math:Else>
        </ConditionalStatement>
        ...
    </ParameterModel>
```

Note that the attribute `symbIdRef`, used to reference the individual parameters in each statement, was not available for parameters or variables previously.

Example 4 – 'Three Compartment Infusion, Coefficient and Exponents'

Source: Fisher/Shafer NONMEM course, [Fisher and Shafer, 2007]. The NMTRAN code for the structural and observation model reads

```
; C1, C2, C3, L1, L2, L3, RATE, DUR defined elsewhere
IF (TIME.LE.DUR) THEN
        TY1 = RATE*C1/L1*(1-EXP(-L1*TIME))
        TY2 = RATE*C2/L2*(1-EXP(-L2*TIME))
        TY3 = RATE*C3/L3*(1-EXP(-L3*TIME))
ELSE
        TY1 = RATE*C1/L1*(1-EXP(-L1*DUR))*EXP(-L1*(TIME-DUR))
        TY2 = RATE*C2/L2*(1-EXP(-L2*DUR))*EXP(-L2*(TIME-DUR))
        TY3 = RATE*C3/L3*(1-EXP(-L3*DUR))*EXP(-L3*(TIME-DUR))
ENDIF

Y=(TY1+TY2+TY3)*(1+EPS(1)) ; Constant CV model
```

and in PharmML

```
<!-- STRUCTURAL MODEL -->
<StructuralModel blkId="sm1">

    <ct:Variable symbolType="real" symbId="TY1"/>
    <ct:Variable symbolType="real" symbId="TY2"/>
    <ct:Variable symbolType="real" symbId="TY3"/>

    <ConditionalStatement>
        <math:If>
```

```
                <math:Condition>
                    <math:LogicBinop op="leq">
                        <ct:SymbRef symbIdRef="t"/>
                        <ct:SymbRef symbIdRef="DUR"/>
                    </math:LogicBinop>
                </math:Condition>
                <!-- TY1 = RATE*C1/L1*(1 - EXP(-L1*TIME)) -->
                <ct:AssignStatement op="eq">
                    <ct:SymbRef symbIdRef="TY1"/>
                    <!-- omitted RHS expression -->
                </ct:AssignStatement>
                <!-- TY2 = RATE*C2/L2*(1-EXP(-L2*TIME)) -->
                <ct:AssignStatement op="eq">
                    <ct:SymbRef symbIdRef="TY2"/>
                    <!-- omitted RHS expression -->
                </ct:AssignStatement>
                <ct:AssignStatement op="eq">
                    <ct:SymbRef symbIdRef="TY3"/>
                    <!-- omitted RHS expression -->
                </ct:AssignStatement>
            </math:If>
            <math:Else>
                <!-- TY1 = RATE*C1/L1*(1-EXP(-L1*DUR))*EXP(-L1*(TIME-DUR)) -->
                <ct:AssignStatement op="eq">
                    <ct:SymbRef symbIdRef="TY1"/>
                    <!-- omitted RHS expression -->
                </ct:AssignStatement>
                <!-- TY2 = RATE*C2/L2*(1-EXP(-L2*DUR))*EXP(-L2*(TIME-DUR)) -->
                <ct:AssignStatement op="eq">
                    <ct:SymbRef symbIdRef="TY2"/>
                    <!-- omitted RHS expression -->
                </ct:AssignStatement>
                <!-- TY3 = RATE*C3/L3*(1-EXP(-L3*DUR))*EXP(-L3*(TIME-DUR)) -->
                <ct:AssignStatement op="eq">
                    <ct:SymbRef symbIdRef="TY3"/>
                    <!-- omitted RHS expression -->
                </ct:AssignStatement>

            </math:Else>
        </ConditionalStatement>
```

The observation model is skipped here.

2.4 Nested piecewise

Nesting of piecewise statements was not supported so far, here a typical one

$$f(x) = \begin{cases} \begin{cases} 1 & \text{for} \quad x < 1 \\ 2 & \text{else} \end{cases} & \text{for} \quad x > 0 \\ \quad 3 & \text{else} \end{cases}$$

implemented in PharmML as the following snippet shows

```
        <ct:Variable symbolType="real" symbId="f">
            <ct:Assign>
                <ct:Piecewise>
                    <math:Piece>
                        <!-- nested piecewise -->
                        <math:Piecewise>
                            <math:Piece>
                                <ct:Real>1</ct:Real>
                                <math:Condition>
                                    <math:LogicBinop op="lt">
                                        <ct:SymbRef symbIdRef="x"/>
                                        <ct:Real>1</ct:Real>
                                    </math:LogicBinop>
                                </math:Condition>
                            </math:Piece>
                            <math:Piece>
```

```
                        <ct:Real>2</ct:Real>
                        <math:Condition>
                            <math:Otherwise/>
                        </math:Condition>
                    </math:Piece>
                </math:Piecewise>
                <math:Condition>
                    <math:LogicBinop op="gt">
                        <ct:SymbRef symbIdRef="x"/>
                        <ct:Real>0</ct:Real>
                    </math:LogicBinop>
                </math:Condition>
            </math:Piece>
            <math:Piece>
                <ct:Real>3</ct:Real>
                <math:Condition>
                    <math:Otherwise/>
                </math:Condition>
            </math:Piece>
        </ct:Piecewise>
    </ct:Assign>
</ct:Variable>
```

2.5 Probability functions support

Following probability functions are available

- CDF(x) – cumulative distribution function, `<CDF>`
- PDF(x) – probability density function, `<PDF>`
- HF(x) – hazard function, `<HF>`
- SF(x) – survival function, `<SF>`

The implementation of these probability functions is straightforward, despite their often very complex expressions which are only referred to, and requires the use of the

1. according XML element, e.g. `<CDF>`
2. specification of the distribution of interest and
3. (optional) function argument, x, which can be any expression.

The introduction of these generic elements allows to use any of the ProbOnto univariate continuos distributions, [Swat and Grenon, 2015]. Consider for example the cumulative distribution function, *PHI*, of the standard normal distribution with mean 0 and standard deviation 1 which the two basic function read

$$\text{PDF}: \quad f(x) = \frac{e^{-\frac{1}{2}x^2}}{\sqrt{2\pi}}$$

$$\text{CDF}: \quad F(x) = \frac{1}{2}\left[1 + \text{erf}\left(\frac{x}{\sqrt{2}}\right)\right] \quad \text{with} \quad \text{erf}(z) = \frac{2}{\pi}\int_0^z e^{-t^2} dt$$

where *erf* is the error function[2].

The abbreviated XML code explains its encoding reduced to the specification of the distribution function, the code name, *StandardNormal1*, and the optional argument

```
<ct:Variable symbolType="real" symbId="PHI">
    <ct:Assign>
        <math:CDF>
            <math:Distribution>
                <po:ProbOnto name="StandardNormal1"/>
            </math:Distribution>
```

[2]mathworld.wolfram.com/Erf.html

```
                <ct:Assign>
                <!-- x, argument -->
                </ct:Assign>
            </math:CDF>
        </ct:Assign>
    </ct:Variable>
```

Note, that CDF of the standard normal is straightforward as it doesn't requiere specification of parameters, which are by default $mean = 0$ and $stdev = 1$. The use of probability functions is explained in the following examples.

2.5.1 Example 1 – basic example

Source: Part VII, Help Guide of the [Beal et al., 2009]. The cumulative distribution function, *PHI*, of the standard normal distribution with mean 0 and standard deviation 1, may be used as in this is basic example

```
A=THETA(1)*EXP(ETA(1))
B=PHI(A)
```

$A = \theta_1 \exp(\eta_1)$	<IndividualParameter symbId="A"> <ct:Assign> <math:Binop op="times"> <ct:SymbRef symbIdRef="THETA1"/> <math:Uniop op="exp"> <ct:SymbRef symbIdRef="ETA1"/> </math:Uniop> </math:Binop> </ct:Assign> </IndividualParameter>
$B = \mathrm{CDF}(A)$	<IndividualParameter symbId="B"> <ct:Assign> <math:CDF> <math:Distribution> <po:ProbOnto name="StandardNormal1"/> </math:Distribution> <ct:Assign> <ct:SymbRef symbIdRef="A"/> </ct:Assign> </math:CDF> </ct:Assign> </IndividualParameter>

Table 2.2: Encoding of the basic example using cumulative distribution function, CDF.

2.5.2 Example 2 – M3-method for handling of BLQ data

According to [Beal, 2001], the likelihood for a censored observation at time t, is given by

$$l(t) = \Phi((QL - f(t))/\sqrt{g(t)}), \quad \text{with } \Phi \text{ the CDF of the } \mathcal{N}(0,1)$$

which is the probability that the observation is BQL, (i.e., is between $-\infty$ and quantification limit, QL. An example how to implement the so-called M3-method for handling of such data provides the tutorial by [Mould and Upton, 2013] with the following NMTRAN from the supplementary material of the paper.

```
$ERROR ; Beal Method 3
;residual error is coded here using THETA's rather than EPS's
ADD = THETA(6)
PROP = F*THETA(7)
SD = SQRT(ADD*ADD+PROP*PROP) ; combined error model
LLOQ=0.05 ; lower limit of quantification
IF (BLQ.EQ.0) THEN
        F_FLAG=0 ;regular likelihood for measured concentrations
        Y = F + SD*EPS(1)
ENDIF
IF (BLQ.EQ.1) THEN
        F_FLAG=1 ;probability that F is less than LLOQ for missing concentrations
```

```
        Y=PHI((LLOQ−F)/SD)
    ENDIF
```

While the first couple of lines are standard and will be omitted here, the encoding of the essential line with PHI function, Y=PHI((LLOQ-F)/SD), is shown in the following snippet

```
<ObservationModel blkId="om4">
    <ContinuousData>
        <!-- Y=PHI((LLOQ-F)/SD) -->
        <General symbId="Y">
            <ct:Assign>
                <math:CDF>
                    <math:Distribution>
                        <po:ProbOnto name="StandardNormal1"/>
                    </math:Distribution>
                    <ct:Assign>
                        <math:Binop op="divide">
                            <math:Binop op="minus">
                                <ct:SymbRef symbIdRef="LLOQ"/>
                                <ct:SymbRef blkIdRef="sm1" symbIdRef="F"/>
                            </math:Binop>
                            <ct:SymbRef symbIdRef="SD"/>
                        </math:Binop>
                    </ct:Assign>
                </math:CDF>
            </ct:Assign>
        </General>
    </ContinuousData>
</ObservationModel>
```

As noted before, the encoding of the CDF of the standard normal doesn't require the specification of default, $mean = 0$ and $stdev = 1$, parameters.

2.5.3 Example 3 – TTE model

The probability functions turn out to be very useful also in the encoding of time-to-event data. While so far encoding of hazard or survival functions formulas was the only option to defined such models, the function elements offer a quick and safe way to encode often complex function of interest as the following example demonstrates for a Weibull model

```
<HazardFunction symbId="h2">
    <ct:Assign>
        <math:HF>
            <math:Distribution>
                <ProbOnto xmlns="http://www.pharmml.org/probonto/ProbOnto" name="Weibull1">
                    <Parameter name="scale">
                        <ct:Assign>
                            <ct:SymbRef blkIdRef="pm1" symbIdRef="beta"/>
                        </ct:Assign>
                    </Parameter>
                    <Parameter name="shape">
                        <ct:Assign>
                            <ct:SymbRef blkIdRef="pm1" symbIdRef="lambda"/>
                        </ct:Assign>
                    </Parameter>
                </ProbOnto>
            </math:Distribution>
            <!-- function argument is optional -->
            <ct:Assign>
                <ct:SymbRef symbIdRef="t"/>
            </ct:Assign>
        </math:HF>
    </ct:Assign>
</HazardFunction>
```

2.6 Samples – empirical distributions support

When formulating models it is often not possible to specify a probability distribution using a parametric one because its defining function is not known. Instead, a set of values can be considered as constituting a

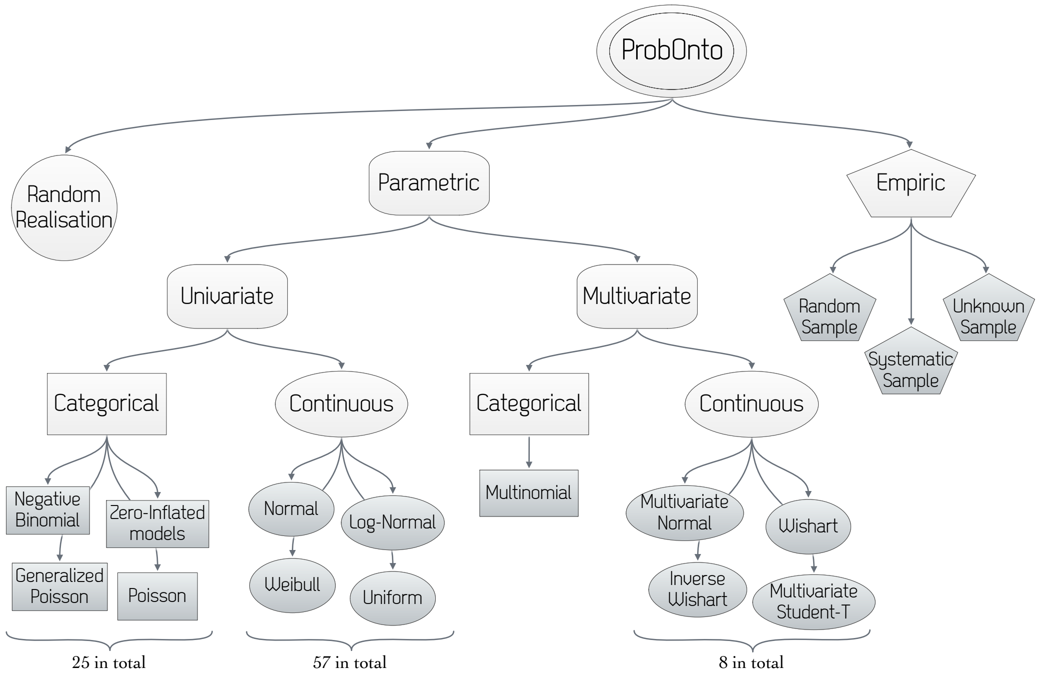


Figure 2.2: ProbOnto tree – showing the current structure. For brevity only few distributions per category are listed.

distribution model. According extensions were required in ProbOnto and PharmML.

Samples are sets of realisations obtained from a known or unknown distribution. Following UncertML [UncertML Team, 2014] we distinguish between

- Random sample, `<RandomSample>`
- Systematic sample, `<SystematicSample>`
- Unknown sample, `<UnknownSample>`

The only (optional) parameter is

- `weight` – to encode weighting given to each realisation.

The main type which should be used for our purposes is the first one defined as

Random Sample – a set of independent realisations, x_i, drawn from a probability distribution $p(x)$ (or alternatively a population where every member has an equal chance of being drawn, but is randomly selected). The sample will typically be obtained using some form of simulation algorithm for distributions using a random number generator.

See also ProbOnto specification for more details, [Swat and Grenon, 2015].

2.6.1 Two modes of operation

There are two ways samples can be encoded in PharmML, Table 2.3:

- Option 1: Data stored together with the corresponding parameter
- Option 2: Data stored in `<TrialDesign>`

Both encoding result in exact the same model. In both cases the mapping between data columns and parameters is essential to assure complete model definition (if only one column is given, mapping is optional, see Table 2.3 alternative **Option 1** version).

Option 1: Data stored together with parameter

```
<ParameterModel blkId="PM1">
    <PopulationParameter symbId="POP_K">
        <Distribution>
            <po:ProbOnto name="RandomSample">
                <po:ColumnMapping>
                    <ds:ColumnRef columnIdRef="POP_K_sample"/>
                    <ct:SymbRef symbIdRef="POP_K"/>
                </po:ColumnMapping>
                <ds:DataSet>
                    <ds:Definition>
                        <ds:Column columnId="POP_K_sample" valueType="real" columnNum="1"/>
                    </ds:Definition>
                    <ds:Table>
                        <ds:Row><ct:Real>0.10</ct:Real></ds:Row>
                        <!-- other values omitted -->
                    </ds:Table>
                </ds:DataSet>
            </po:ProbOnto>
        </Distribution>
    </PopulationParameter>

    <!-- ALTERNATIVE - short version -->
    <PopulationParameter symbId="POP_K">
        <Distribution>
            <po:ProbOnto name="RandomSample">
                <ds:DataSet>
                    <ds:Table>
                        <ds:Row><ct:Real>0.10</ct:Real></ds:Row>
                        <!-- other values omitted -->
                    </ds:Table>
                </ds:DataSet>
            </po:ProbOnto>
        </Distribution>
    </PopulationParameter>
</ParameterModel>
```

Option 2: Data stored in <TrialDesign>

```
<!-- PART 1 -->
<ParameterModel blkId="PM2">
    <PopulationParameter symbId="POP_K">
        <Distribution>
            <po:ProbOnto name="RandomSample"/>
        </Distribution>
    </PopulationParameter>
</ParameterModel>

<!-- PART 2 -->
<TrialDesign xmlns="http://www.pharmml.org/pharmml/0.8/TrialDesign">
    <ExternalDataSet oid="RdataSet">
        <ColumnMapping>
            <ds:ColumnRef columnIdRef="POP_K_sample"/>
            <ct:SymbRef blkIdRef="PM2" symbIdRef="POP_K"/>
        </ColumnMapping>
        <ds:DataSet>
            <ds:Definition>
                <ds:Column columnId="POP_K_sample" valueType="real" columnNum="1"/>
            </ds:Definition>
            <ds:Table>
                <ds:Row><ct:Real>0.10</ct:Real></ds:Row>
                <!-- other values omitted -->
            </ds:Table>
        </ds:DataSet>
    </ExternalDataSet>
</TrialDesign>
```

Table 2.3: Two ways to encode samples. **Option 1**: data stored together with parameter declaration in <ModelDefinition>. Two version are shown: (top) with full data/model mapping and dataset column declaration, (bottom) short form when only one dataset is required – mapping and column declaration are redundant. **Option 2**: data stored in the <TrialDesign> – mapping is mandatory even for single column.

2.6.2 Mapping without `weight` parameter

Option 1

(See the use case *bayesianHierarchical/example3423.xml* using this option.)

Sometimes the **weight** parameter is not defined and the assumption is that all values are equally probable with $p = 1/n$ where n is equal the row number, see Table 2.4.

POP_V
5.050013
5.064166
5.078318
...

Table 2.4: A dataset represented using equiprobable samples. Because no **weight** parameter is defined, the assumption is that all values are equiprobable with $p = 1/n$ where n is equal the row number.

```
<PopulationParameter symbId="POP_K">
    <Distribution>
        <po:ProbOnto name="RandomSample">
            <ds:DataSet>
                <ds:Table>
                    <ds:Row><ct:Real>0.10</ct:Real></ds:Row>
                    <ds:Row><ct:Real>0.23</ct:Real></ds:Row>
                    <ds:Row><ct:Real>0.3</ct:Real></ds:Row>
                </ds:Table>
                <!-- <ds:ExternalFile oid="sxtData">
                    <ds:path>POP_K_sample.csv</ds:path>
                </ds:ExternalFile>-->
            </ds:DataSet>
        </po:ProbOnto>
    </Distribution>
</PopulationParameter>
```

Here the short version was used which is applicable when only one data column in required. Providing mapping and column definition is in such cases redundant, unless the data is stored in **<TrialDesign>**, see below.

Option 2

Figure 2.3 shows how to store and map dataset with samples declared in **<TrialDesign>** to a parameter defined in **<ParameterModel>**, here *POP_V*. In this case column mapping and column declaration are required.

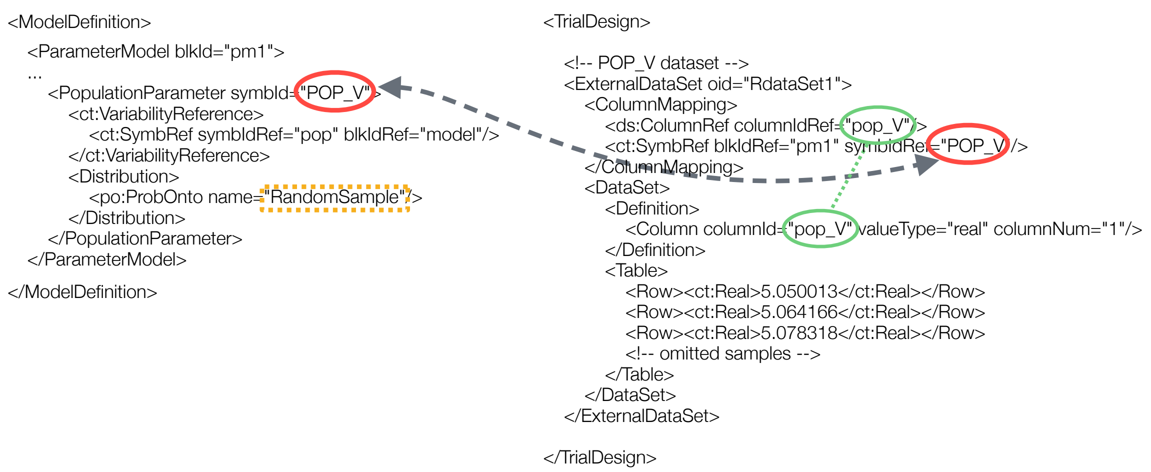


Figure 2.3: Mapping of samples without weights – explained graphically.

2.6.3 Mapping with `weight` parameter

Option 1

(See the use case *bayesianHierarchical/example3423.xml* using this option.)

In the case when single variables are defined with `weight` parameter, p.

P	POP_K
0.25	0.10
0.25	0.23
0.5	0.3

Table 2.5: This data set represents sample use case with `weight` parameter, P.

Here the according samples are encoded directly with the parameter of interest. The mapping is self-explanatory. The samples can be encoded either inline, using `<Table>`, or in external datasets referenced within the `<ExternalFile>` element.

```
<PopulationParameter symbId="p"/>
<PopulationParameter symbId="POP_K">
    <Distribution>
        <po:ProbOnto name="RandomSample">
            <po:Parameter name="weight">
                <ct:Assign>
                    <ct:SymbRef symbIdRef="p"/>
                </ct:Assign>
            </po:Parameter>
            <po:ColumnMapping xmlns="http://www.pharmml.org/pharmml/0.8/TrialDesign">
                <ds:ColumnRef columnIdRef="P"/>
                <ct:SymbRef symbIdRef="p"/>
            </po:ColumnMapping>
            <po:ColumnMapping>
                <ds:ColumnRef columnIdRef="POP_K_sample"/>
                <ct:SymbRef symbIdRef="POP_K"/>
            </po:ColumnMapping>
            <ds:DataSet>
                <ds:Definition>
                    <ds:Column columnId="P" valueType="real" columnNum="1"/>
                    <ds:Column columnId="POP_K_sample" valueType="real" columnNum="2"/>
                </ds:Definition>
                <ds:Table>
                    <ds:Row><ct:Real>0.25</ct:Real><ct:Real>0.10</ct:Real></ds:Row>
                    <ds:Row><ct:Real>0.25</ct:Real><ct:Real>0.23</ct:Real></ds:Row>
                    <ds:Row><ct:Real>0.5</ct:Real><ct:Real>0.3</ct:Real></ds:Row>
                </ds:Table>
                <!-- <ds:ExternalFile oid="sxtData">
                        <ds:path>POP_K_sample.csv</ds:path>
                    </ds:ExternalFile>-->
            </ds:DataSet>
        </po:ProbOnto>
    </Distribution>
</PopulationParameter>
```

Option 2

This is an example for a model with two parameters POP_K and POP_K drawn from a random sample with probability p, see Table 2.6 and Figure 2.4 for a graphical explanation,

- `RandomSample` is used with
- `weight` parameter p declared as `<PopulationParameter>` applies to each row.

POP_V	POP_K	p
8	0.10	0.25
14	0.23	0.25
32	0.3	0.5

Table 2.6: Test data set represented using samples. Weight, p, applies to each pair of values.

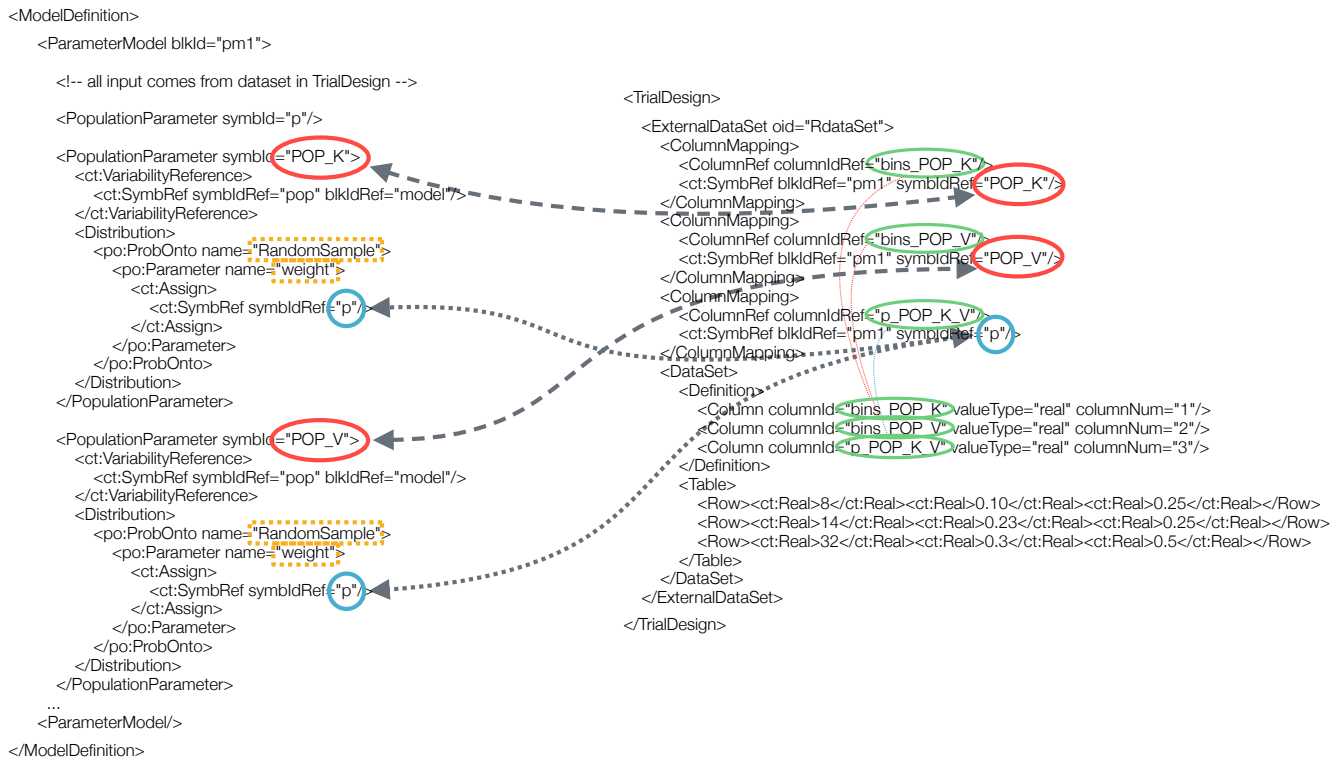


Figure 2.4: Mapping of samples with weights – explained graphically.

PharmML implementation of the parameter model with distributions defined using *RandomSample* with *weight* is as following

```
<ParameterModel blkId="pm3">
    <PopulationParameter symbId="p"/>
    <PopulationParameter symbId="POP_K">
        <ct:VariabilityReference>
            <ct:SymbRef symbIdRef="pop" blkIdRef="vm1"/>
        </ct:VariabilityReference>
        <Distribution>
            <po:ProbOnto name="RandomSample">
                <po:Parameter name="weight">
                    <ct:Assign>
                        <ct:SymbRef symbIdRef="p"/>
                    </ct:Assign>
                </po:Parameter>
            </po:ProbOnto>
        </Distribution>
    </PopulationParameter>
    <PopulationParameter symbId="POP_V_sample">
        <ct:VariabilityReference>
            <ct:SymbRef symbIdRef="pop" blkIdRef="vm1"/>
        </ct:VariabilityReference>
        <Distribution>
            <po:ProbOnto name="RandomSample">
                <po:Parameter name="weight">
                    <ct:Assign>
                        <ct:SymbRef symbIdRef="p"/>
                    </ct:Assign>
```

```
                </po:Parameter>
            </po:ProbOnto>
        </Distribution>
    </PopulationParameter>
</ParameterModel>
```

Then the dataset is defined in the `<TrialDesign>` as shown below with the proper column mapping.

```
<TrialDesign xmlns="http://www.pharmml.org/pharmml/0.8/TrialDesign">
    <ExternalDataSet oid="RdataSet">
        <ColumnMapping>
            <ds:ColumnRef columnIdRef="bins_POP_K"/>
            <ct:SymbRef blkIdRef="pm1" symbIdRef="POP_K"/>
        </ColumnMapping>
        <ColumnMapping>
            <ds:ColumnRef columnIdRef="bins_POP_V"/>
            <ct:SymbRef blkIdRef="pm1" symbIdRef="POP_V"/>
        </ColumnMapping>
        <ColumnMapping>
            <ds:ColumnRef columnIdRef="p_POP_V_K"/>
            <ct:SymbRef blkIdRef="pm1" symbIdRef="p"/>
        </ColumnMapping>
        <ds:DataSet>
            <ds:Definition>
                <ds:Column columnId="bins_POP_K" valueType="real" columnNum="1"/>
                <ds:Column columnId="bins_POP_V" valueType="real" columnNum="2"/>
                <ds:Column columnId="p_POP_V_K" valueType="real" columnNum="3"/>
            </ds:Definition>
            <ds:Table>
                <ds:Row><ct:Real>8</ct:Real><ct:Real>0.10</ct:Real><ct:Real>0.25</ct:Real></ds:Row>
                <ds:Row><ct:Real>14</ct:Real><ct:Real>0.23</ct:Real><ct:Real>0.25</ct:Real></ds:Row>
                <ds:Row><ct:Real>32</ct:Real><ct:Real>0.3</ct:Real><ct:Real>0.5</ct:Real></ds:Row>
            </ds:Table>
            <!-- <ds:ExternalFile oid="sxtData">
                    <ds:path>POP_V_K_sample.csv</ds:path>
                 </ds:ExternalFile>-->
        </ds:DataSet>
    </ExternalDataSet>
</TrialDesign>
```

2.6.4 Mapping using vectors

(See the use case *bayesianHierarchical/example3421dep_NM.xml* using this option.)

In the case the modeller wants to encode a parameter in vector form, here POP_K and POP_V as vector elements of POP_K_V (see use cases in the *bayesianHierarchical* example folder) there exist a possibility to map the vector to sample elements, see Figure 2.5 for a graphical explanation or the code snippet below.

POP_K_V is stored in dataset as $bins_POP_K$ in `<TrialDesign>` section and mapping between columns and model elements is performed using `<ColumnMapping>`

```
<ExternalDataSet oid="RdataSet">
    <ColumnMapping>
        <ColumnRef xmlns="http://www.pharmml.org/pharmml/0.8/Dataset" columnIdRef="bins_POP_K"/>
        <ct:Assign>
            <ct:VectorSelector>
                <ct:SymbRef symbIdRef="POP_K_V" blkIdRef="pm1"/>
                <ct:Cell>
                    <ct:Int>1</ct:Int>
                </ct:Cell>
            </ct:VectorSelector>
        </ct:Assign>
    </ColumnMapping>
    <ColumnMapping>
        <ColumnRef xmlns="http://www.pharmml.org/pharmml/0.8/Dataset" columnIdRef="bins_POP_V"/>
        <ct:Assign>
            <ct:VectorSelector>
                <ct:SymbRef symbIdRef="POP_K_V" blkIdRef="pm1"/>
                <ct:Cell>
                    <ct:Int>2</ct:Int>
                </ct:Cell>
```

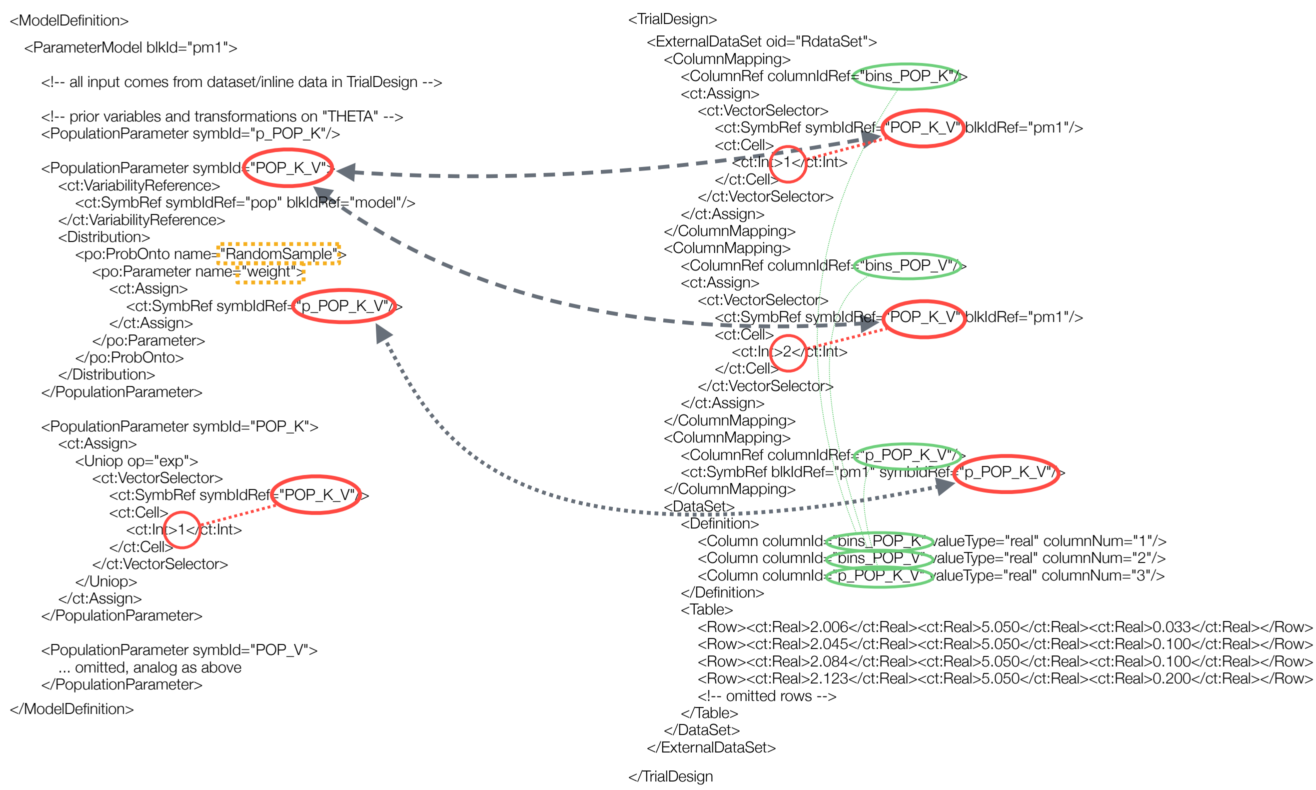


Figure 2.5: Mapping of samples with vectors and weights – explained graphically.

```
            </ct:VectorSelector>
        </ct:Assign>
    </ColumnMapping>
    <ColumnMapping>
        <ColumnRef xmlns="http://www.pharmml.org/pharmml/0.8/Dataset" columnIdRef="p_POP_K_V"/>
        <ct:SymbRef symbIdRef="p_POP_K_V" blkIdRef="pm1"/>
    </ColumnMapping>
    <DataSet xmlns="http://www.pharmml.org/pharmml/0.8/Dataset">
        <Definition>
            <Column columnId="bins_POP_K" valueType="real" columnNum="1"/>
            <Column columnId="bins_POP_V" valueType="real" columnNum="2"/>
            <Column columnId="p_POP_K_V" valueType="real" columnNum="3"/>
        </Definition>
        <Table>
            <Row><ct:Real>2.006510</ct:Real><ct:Real>5.050013</ct:Real><ct:Real>0.033</ct:Real></Row>
            <Row><ct:Real>2.045465</ct:Real><ct:Real>5.050013</ct:Real><ct:Real>0.100</ct:Real></Row>
            <Row><ct:Real>2.084421</ct:Real><ct:Real>5.050013</ct:Real><ct:Real>0.100</ct:Real></Row>
            <Row><ct:Real>2.123377</ct:Real><ct:Real>5.050013</ct:Real><ct:Real>0.200</ct:Real></Row>
            <!-- rows skipped -->
        </Table>
    </DataSet>
</ExternalDataSet>
```

What is new in this version is the ability to map a particular vector element. Here the essential part is repeated to underline how the map of column *bins_POP_V* and vector element $POP_K_V[2]$ works.

```
    <ColumnMapping>
        <ColumnRef xmlns="http://www.pharmml.org/pharmml/0.8/Dataset" columnIdRef="bins_POP_V"/>
        <ct:Assign>
            <ct:VectorSelector>
                <ct:SymbRef symbIdRef="POP_K_V" blkIdRef="pm1"/>
                <ct:Cell>
                    <ct:Int>2</ct:Int>
                </ct:Cell>
            </ct:VectorSelector>
        </ct:Assign>
    </ColumnMapping>
```

See also Figure 2.5 which shows the mappings graphically.

2.6.5 Samples in SO

The concept of encoding samples of empirical distributions, either as inline tables of external dataset, is very useful also in the SO context, [Terranova et al., 2016]. For example the posterior distribution of individual parameters estimates is captured using an empirical distribution, *RandomSample*. Here the values for three estimated parameters are encoded inline.

```
<PosteriorDistribution>
   <Distribution>
      <po:ProbOnto name="RandomSample">
         <ds:DataSet>
            <ds:Definition>
               <ds:Column columnId="K" columnType="popParameter" valueType="real" columnNum="1"/>
               <ds:Column columnId="V" columnType="popParameter" valueType="real" columnNum="2"/>
               <ds:Column columnId="CL" columnType="popParameter" valueType="real" columnNum="3"/>
            </ds:Definition>
            <ds:Table>
               <ds:Row><ct:Real>2.006510</ct:Real><ct:Real>5.050013</ct:Real><ct:Real>0.033333</ct:Real></ds:Row>
               <ds:Row><ct:Real>2.045465</ct:Real><ct:Real>5.050013</ct:Real><ct:Real>0.100000</ct:Real></ds:Row>
               <ds:Row><ct:Real>2.084421</ct:Real><ct:Real>5.050013</ct:Real><ct:Real>0.100000</ct:Real></ds:Row>
               <ds:Row><ct:Real>2.123377</ct:Real><ct:Real>5.050013</ct:Real><ct:Real>0.200000</ct:Real></ds:Row>
               <ds:Row><ct:Real>2.162333</ct:Real><ct:Real>5.064166</ct:Real><ct:Real>0.100000</ct:Real></ds:Row>
               <ds:Row><ct:Real>2.201288</ct:Real><ct:Real>5.064166</ct:Real><ct:Real>0.066667</ct:Real></ds:Row>
               <!-- other sample rows skipped -->
            </ds:Table>
            <!--
               <ds:ExternalFile oid="extDataId">
                  <ds:path>samples_KVCL.csv</ds:path>
               </ds:ExternalFile>-->
         </ds:DataSet>
      </po:ProbOnto>
   </Distribution>
</PosteriorDistribution>
```

The alternative using external datasets is commented out. Note also that no mapping is required in this case, the symbol assigned to the attribute `columnId` identifies the parameter.

2.7 Random realisations

Another new feature in version 0.8 is the possibility to specify a single realisation from any univariate distribution and is based on analogue feature in UncertML, [UncertML Team, 2014]. The definition reads:

Realisation is a single instance of a random variable and can be used to imply an observed value, or, as more widely used, a single draw, x_i, from a probability distribution, $p(x)$.

A new element `<Realisation>` has been introduced referring to ProbOnto with the specification of the code name of the distribution of interest and its parameters as the following example shows.

Example

Source [Lixoft Team, 2014b]. Categorical covariates can be defined as a discrete transformation of continuous random variables. Consider for instance the following model encoded in MLXTRAN.

```
[COVARIATE]
DEFINITION:
z = {distribution=normal, mean=0, sd=1}

EQUATION:
if z<-0.2533 ; P(z<-0.2533)=0.4
        cz = 0
else
        cz = 1
end
c=cz
```

Note, that z={distribution=normal,...} in the DEFINITION block is a sampling assignment which is encoded in the element <Realisation>.

```
    <CovariateModel blkId="cm2">
        <Covariate symbId="z">
            <Continuous>
                <Realisation>
                    <po:ProbOnto name="StandardNormal1"/>
                </Realisation>
            </Continuous>
        </Covariate>
```

The remaining definition and assignment of a new covariate is conditional on the sampled covariate value z

```
        <Covariate symbId="cz">
            <Categorical>
                <ct:Assign>
                    <ct:Piecewise>
                        <math:Piece>
                            <ct:Real>0</ct:Real>
                            <math:Condition>
                                <math:LogicBinop op="lt">
                                    <ct:SymbRef symbIdRef="z"/>
                                    <ct:Real>-0.2533</ct:Real>
                                </math:LogicBinop>
                            </math:Condition>
                        </math:Piece>
                        <math:Piece>
                            <ct:Real>1</ct:Real>
                            <math:Condition>
                                <math:Otherwise/>
                            </math:Condition>
                        </math:Piece>
                    </ct:Piecewise>
                </ct:Assign>
            </Categorical>
        </Covariate>
    </CovariateModel>
```

2.8 Markov models

New element – transition aka stochastic matrix – has been introduced to simplify encoding of Markov models.

2.8.1 Transition matrix

Stochastic (aka transition) matrix is used to describe the transitions in a Markov chain[3]. A stochastic matrix can be characterised further by an optional attribute

- Left stochastic (default), type="leftStochastic", a real square matrix, with each row summing to 1.
- Right stochastic, type="rightStochastic", a real square matrix, with each column summing to 1.
- Doubly stochastic, type="doubleStochastic", both columns and rows are summing to 1.

Figure 2.6 visualises a basic Markov model with the according transition matrix and the XML code shows how it is implemented

```
    <Dependance type="discreteMarkov"/>
    <TransitionMatrix type="leftStochastic">
        <ct:Matrix matrixType="Any">
            <ct:ColumnNames>
                <ct:SymbRef symbIdRef="cat1"/>
                <ct:SymbRef symbIdRef="cat2"/>
```

[3] https://en.wikipedia.org/wiki/Stochastic_matrix

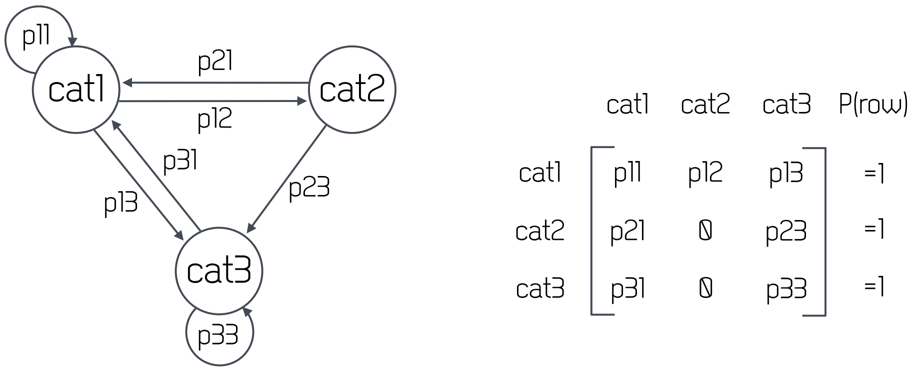


Figure 2.6: An example for a transition matrix for a Markov chain. (Left) the state transition diagram with three possible states cat1, cat2, and cat3. (Right) The left stochastic transition matrix.

```
                <ct:SymbRef symbIdRef="cat3"/>
            </ct:ColumnNames>
            <ct:MatrixRow>
                <ct:SymbRef symbIdRef="p11"/>
                <ct:SymbRef symbIdRef="p12"/>
                <ct:SymbRef symbIdRef="p13"/>
            </ct:MatrixRow>
            <ct:MatrixRow>
                <ct:SymbRef symbIdRef="p21"/>
                <ct:Real>0</ct:Real>
                <ct:SymbRef symbIdRef="p23"/>
            </ct:MatrixRow>
            <ct:MatrixRow>
                <ct:SymbRef symbIdRef="p31"/>
                <ct:Real>0</ct:Real>
                <ct:SymbRef symbIdRef="p33"/>
            </ct:MatrixRow>
        </ct:Matrix>
    </TransitionMatrix>
```

If the `type` attribute is not used, the default is the `leftStochastic` matrix.

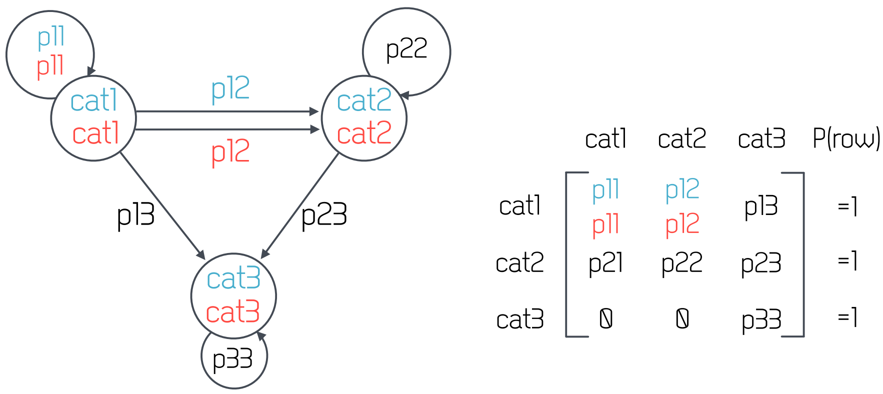


Figure 2.7: An example of a conditional Markov model with the according transition matrix. It is a left stochastic matrix with probabilities in each raw summing up to 1.

2.8.2 Conditional transition probabilities/matrix

Conditional transition probabilities can be implemented pairwise, e.g.

$$P(\text{Cat1} \rightarrow \text{Cat2}|\ \text{COLOR} == \text{BLUE}) = p12_{BLUE}$$
$$P(\text{Cat1} \rightarrow \text{Cat2}|\ \text{COLOR} == \text{RED}) = p12_{RED}$$

other probabilities omitted...

or with a conditional transition matrix using piecewise statements were required, Figure 2.7. See also section 3.2 for complete examples.

2.9 N-ary operators

N-ary operators are an extension of binary operators in cases when more then two arguments are provided. The current version supports the following operators

N-ary operator	Name	Argument 1 ... N	Return type
Addition	`plus`	`X`, ..., `X`	`Real`
Multiplication	`times`	`X`, ..., `X`	`Real`
Smallest of the arguments	`min`	`X`, ..., `X`	`Real`
Largest of the arguments	`max`	`X`, ..., `X`	`Real`
Greatest common divisor	`gcd`	`X`, ..., `X`	`Real`
Least common multiple	`lcm`	`X`, ..., `X`	`Real`

Table 2.7: N-ary operators supported in v0.8.

2.9.1 Rules for using <Naryop>

- `X` can be either a variable reference, real vector, vector/matrix (if a raw/column is required) selector or sequence.
- when more then one argument is required `<Vector>` with `<VectorElements>` has to be used.

Example

An example of the application of N-ary operators is shown below using the `max` operator acting on a vector of arguments, e.g. Amax = max(1, a2, 3, a4, fifthElement, . . .).

```
<IndividualParameter symbId="Amax">
    <ct:Assign>
        <math:Naryop op="max">
            <ct:Vector>
                <ct:VectorElements>
                    <ct:Real>1</ct:Real>
                    <ct:SymbRef symbIdRef="a2"/>
                    <ct:Real>3</ct:Real>
                    <ct:SymbRef symbIdRef="a4"/>
                    <ct:SymbRef symbIdRef="fifthElement"/>
                    <!-- ... -->
                </ct:VectorElements>
            </ct:Vector>
        </math:Naryop>
    </ct:Assign>
</IndividualParameter>
```

2.10 Statistical operators

Based on the UncertML collection, a set of basic statistics is supported in v0.8 as shown in Table 2.8. Arguments X and Y can be references to variables or explicit vectors of values, see examples below. In few cases a second argument is required, *level* or *order*. The order of the arguments, 'Argument 2' must follow 'Argument 1', is important as it determines the meaning of the argument.

Statistic	Name	1^{st}argument	2^{nd}argument	Return type
Centred moment	`centredMoment`	`X`	`order` $\in \mathbb{N}$	`Real`
Coeff. of variation	`coefficientOfVariation`	`X`	–	`Real`
Correlation	`correlation`	`X`	`Y`	`Real`
Decile	`decile`	`X`	`level` $\in \{1, ..., 9\}$	`Real`
Geometric mean	`geometricMean`	`X`	–	`Real`

Kurtosis	kurtosis	X	–	Real
Mean	mean	X	–	Real
Median	median	X	–	Real
Mode	mode	X	–	Real
Moment	moment	X	order $\in \mathbb{N}$	Real
Percentile	percentile	X	level $\in [0, 100]$	Real
Quantile	quantile	X	level $\in [0, 1]$	Real
Quartile	quartile	X	level $\in \{0.25, 0.5, 0.75, 1\}$	Real
Range	range	X	–	Real
Skewness	skewness	X	–	Real
Stand. deviation	standardDeviation	X	–	Real
Variance	variance	X	–	Real

Table 2.8: Statistics supported in v0.8. The type of the arguments X and Y can be either variable references or real vectors. Multiple values without <Vector> element are not allowed. In few cases a second argument is required, level or order. The correct sequence of the arguments, first 'Argument 1' then 'Argument 2', is important as it determines the meaning of the arguments.

2.10.1 Rules for using <Statsop>

- arguments X and Y can be either variable references or real vectors
- type of arguments level and order is specified in the Table 2.8
- only two arguments are allowed as children of <Statsop>, see Table 2.8
- for statistics where more then two arguments are expected, <Vector> with <VectorElements> has to be used.

2.10.2 Examples

Some examples of the use of statistical operators are shown below.

Basic example

This example shows how to use the *mean* operator acting on existing parameters

```
<PopulationParameter symbId="Qm">
    <ct:Assign>
        <math:Statsop op="mean">
            <ct:Vector>
                <ct:VectorElements>
                    <ct:SymbRef blkIdRef="pm1" symbIdRef="Q0"/>
                    <ct:SymbRef blkIdRef="pm1" symbIdRef="Q1"/>
                    <ct:SymbRef blkIdRef="pm1" symbIdRef="Q2"/>
                </ct:VectorElements>
            </ct:Vector>
        </math:Statsop>
    </ct:Assign>
</PopulationParameter>
```

More example

The next few examples show how to use these operators given a dummy dataset for which a histogram and a box plot has been plotted in Figure 2.8. The dataset scores = [78, 84, 83, ..., 81, 92, 89][4] can be imagined being mapped from an external file or a non-parametric distribution using the sample concept, see section 2.6, here with inline encoded data

[4]The whole set is: [78, 84, 83, 80, 94, 90, 81, 79, 79, 81, 85, 87, 86, 89, 92, 78, 74, 83, 80, 74, 80, 82, 89, 89, 73, 75, 87, 76, 79, 84, 78, 76, 75, 85, 79, 82, 87, 84, 94, 88, 80, 82, 81, 92, 89]

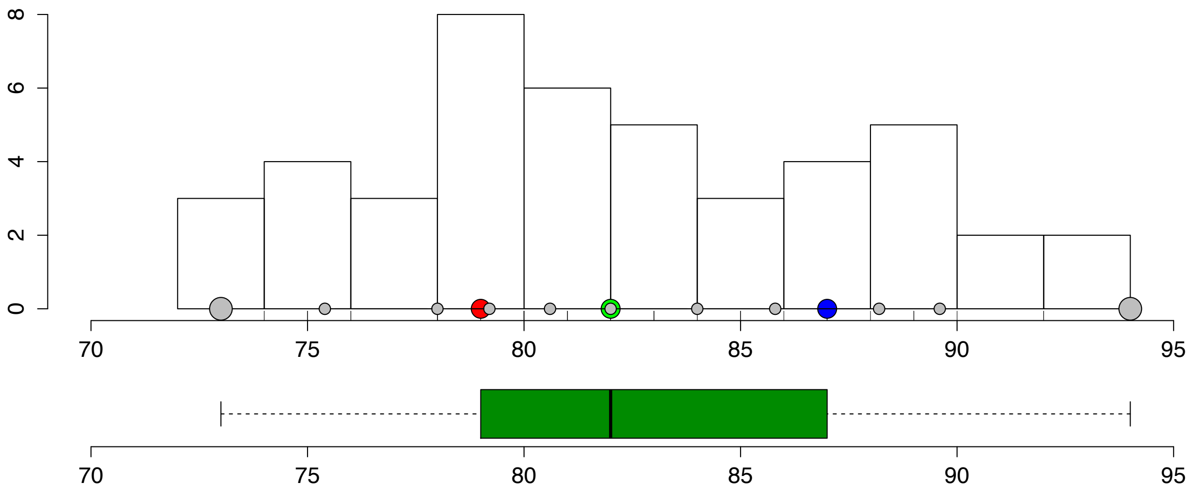


Figure 2.8: Deciles (grey), quartiles: 1^{st} (red), 2^{nd}/median (green) and 3^{rd} (blue) and min/max (large grey circles). Any of these characteristic points can be specified now in PharmML for a given dataset or random variable using either the statistical or n-ary operators.

```
<PopulationParameter symbId="scores">
    <Distribution>
        <po:ProbOnto name="RandomSample">
            <ds:DataSet>
                <ds:Table>
                    <!-- 78, 84, 83, ..., 81, 92, 89 -->
                    <ds:Row>
                        <ct:Real>78</ct:Real><ct:Real>84</ct:Real><ct:Real>83</ct:Real>
                        <!-- omitted values -->
                        <ct:Real>81</ct:Real><ct:Real>92</ct:Real><ct:Real>89</ct:Real>
                    </ds:Row>
                </ds:Table>
            </ds:DataSet>
        </po:ProbOnto>
    </Distribution>
</PopulationParameter>
```

The typical numbers characterising this sample, such as deciles or quartiles, can be defined in PharmML as the following snippets show

- Q1 – 1^{st} quartile (red dot in Figure 2.8)

```
<PopulationParameter symbId="Q1">
    <ct:Assign>
        <math:Statsop op="quartile">
            <ct:SymbRef symbIdRef="scores"/>
            <ct:Real>0.25</ct:Real>
        </math:Statsop>
    </ct:Assign>
</PopulationParameter>
```

- Median/Q2 – median / 2^{nd} quartile (green)

```
<PopulationParameter symbId="Median">
    <ct:Assign>
        <math:Statsop op="median">
            <ct:SymbRef symbIdRef="scores"/>
        </math:Statsop>
    </ct:Assign>
</PopulationParameter>

<PopulationParameter symbId="Q2">
    <ct:Assign>
        <math:Statsop op="quartile">
            <ct:SymbRef symbIdRef="scores"/>
            <ct:Real>0.5</ct:Real>
        </math:Statsop>
```

```
    </ct:Assign>
</PopulationParameter>
```

- Q3 – 3^{rd} quartile (blue)

```
<PopulationParameter symbId="Q3">
    <ct:Assign>
        <math:Statsop op="quartile">
            <ct:SymbRef symbIdRef="scores"/>
            <ct:Real>0.75</ct:Real>
        </math:Statsop>
    </ct:Assign>
</PopulationParameter>
```

- D5 – 4^{th} decile (gray)

```
<PopulationParameter symbId="D4">
    <ct:Assign>
        <math:Statsop op="decile">
            <ct:SymbRef symbIdRef="scores"/>
            <ct:Int>4</ct:Int>
        </math:Statsop>
    </ct:Assign>
</PopulationParameter>
```

2.11 Covariates versus regressors

There is no unified naming convention for covariates and regressors and we assume the following classification based on the DDMoRe discussions and target tool related literature, [Beal et al., 2009] and [Lavielle, 2014],

- constant covariates
- occasion dependent covariates
- time dependent covariates – *aka* regressors

Due to the various interpretation and nomenclature there is a need to identify those and this can be done also in the `<CovariateModel>` by assigning an optional attribute `type` with values *occasionDependent*, *timeDependent*, *constant*. The following snippet shows how this is done for a occasion dependent covariate

```
<CovariateModel blkId="cm1">
    <Covariate type="occasionDependent" symbId="CovariateX">
        <!-- ... -->
    </Covariate>
</CovariateModel>
```

For more on regressor support in PharmML, see Section "6.4 Regressor support" in v0.7 spec [Swat et al., 2015].

2.12 Extension in categorical covariate model

Until now the only option for categorical covariates was to declare them with their categories and (if required) associated probabilities. Version 0.8 comes few a number of extensions.

2.12.1 Covariate declaration/assignment

Declaration of new categorical covariates, based on existing ones, is now possible. Here a simple example of a conditional definition from one of the IOG use cases

```
DDU = if (DDUR > 2) then 1 # duration > 1 month
else 0 # duration of current episode < 1 month
```

which implementation in XML using the piecewise structure reads

```xml
<Covariate symbId="DDU">
    <Categorical>
        <ct:Assign>
            <ct:Piecewise>
                <math:Piece>
                    <ct:Real>1</ct:Real>
                    <math:Condition>
                        <math:LogicBinop op="lt">
                            <ct:SymbRef symbIdRef="DDUR"/>
                            <ct:Real>2</ct:Real>
                        </math:LogicBinop>
                    </math:Condition>
                </math:Piece>
                <math:Piece>
                    <ct:Real>0</ct:Real>
                    <math:Condition>
                        <math:Otherwise/>
                    </math:Condition>
                </math:Piece>
            </ct:Piecewise>
        </ct:Assign>
    </Categorical>
</Covariate>
```

2.12.2 Unified distribution definition

The possibility to define distribution of categorical covariates was given before but was inconsistent with the way continuous covariates are handled. While the previous option (using `<Probability>` for each category is still supported), the `<Distribution>` element referring to any ProbOnto distribution can be used now as well.

option supported in $\leq$ 0.7.3	new option in version 0.8

option supported in $\leq$ 0.7.3:

```xml
<Covariate symbId="Sex">
    <Categorical>
        <Category catId="F">
            <Probability>
                <ct:SymbRef symbIdRef="p1"/>
            </Probability>
        </Category>
        <Category catId="M">
            <Probability>
                <ct:SymbRef symbIdRef="p2"/>
            </Probability>
        </Category>
    </Categorical>
</Covariate>
```

new option in version 0.8:

```xml
<Covariate symbId="Sex">
    <Categorical>
        <Category catId="F"/>
        <Category catId="M"/>
        <Distribution>
            <po:ProbOnto name="CategoricalNonordered1">
                <po:Parameter name="categoryProb">
                    <ct:Assign>
                        <ct:Vector>
                            <ct:VectorElements>
                                <ct:SymbRef symbIdRef="p1"/>
                                <ct:SymbRef symbIdRef="p2"/>
                            </ct:VectorElements>
                        </ct:Vector>
                    </ct:Assign>
                </po:Parameter>
            </po:ProbOnto>
        </Distribution>
    </Categorical>
</Covariate>
```

Table 2.9: Declaration of categorical covariates distribution using `<Distribution>` and `<ProbOnto>` elements in now supported (right).

2.12.3 Conditional distributions

Also new is the possibility to define a conditional covariate distribution. E.g. the previous example in Table 2.9 is extended in that the SEX distribution varies between studies, with probabilities given by the two following vectors $\boldsymbol{p_{S1}} = \{p1_{S1}, p2_{S1}\}$ or $\boldsymbol{p_{S2}} = \{p1_{S2}, p2_{S2}\}$. Such model reads then

if $STUDY == S1$ **then**
    SEX $\sim$ Categorical($\boldsymbol{p_{S1}}$)
else if $STUDY == S2$ **then**
    SEX $\sim$ Categorical($\boldsymbol{p_{S2}}$)

end if

First the STUDY covariate is declared with category identifiers S1 and S2

```
<Covariate symbId="STUDY">
    <Categorical>
        <Category catId="S1"/>
        <Category catId="S2"/>
    </Categorical>
</Covariate>
```

then the distribution of SEX can be defined using here the *CategoricalNonordered1* distribution defined in ProbOnto

```
<Covariate symbId="SEX">
    <Categorical>
        <Category catId="F"/>
        <Category catId="M"/>
        <Distribution>
            <Piecewise>
                <math:Piece>
                    <po:ProbOnto name="CategoricalNonordered1">
                        <po:Parameter name="categoryProb">
                            <ct:Assign>
                                <ct:Vector>
                                    <ct:VectorElements>
                                        <ct:SymbRef symbIdRef="p1_S1"/>
                                        <ct:SymbRef symbIdRef="p2_S1"/>
                                    </ct:VectorElements>
                                </ct:Vector>
                            </ct:Assign>
                        </po:Parameter>
                    </po:ProbOnto>
                    <math:Condition>
                        <math:LogicBinop op="eq">
                            <ct:SymbRef symbIdRef="STUDY"/>
                            <ct:CatRef catIdRef="S1"/>
                        </math:LogicBinop>
                    </math:Condition>
                </math:Piece>
                <math:Piece>
                    <po:ProbOnto name="CategoricalNonordered1">
                        <po:Parameter name="categoryProb">
                            <ct:Assign>
                                <ct:Vector>
                                    <ct:VectorElements>
                                        <ct:SymbRef symbIdRef="p1_S2"/>
                                        <ct:SymbRef symbIdRef="p2_S2"/>
                                    </ct:VectorElements>
                                </ct:Vector>
                            </ct:Assign>
                        </po:Parameter>
                    </po:ProbOnto>
                    <math:Condition>
                        <math:LogicBinop op="eq">
                            <ct:SymbRef symbIdRef="STUDY"/>
                            <ct:CatRef catIdRef="S2"/>
                        </math:LogicBinop>
                    </math:Condition>
                </math:Piece>
            </Piecewise>
        </Distribution>
    </Categorical>
</Covariate>
```

2.13 The return of the basic parameter

`<SimpleParameter>` available from the very beginning until version 0.6, replaced in 0.7.3 version by the `<PopulationParameter>`, has been now restored under the name `<Parameter>`. It turned out that it is

useful in QSP and SB modelling where distinction between individual or population parameters is not possible or required.

Because version 0.7.3 was not in use so far neither in the IOG use case nor tool converters, the restoration of the simple `<Parameter>` shouldn't cause any confusions apart from the required renaming. Below we provide a short overview of parameters as used in last three major versions

- version 0.6.1
  - `<SimpleParameter>` – assignment
  - `<IndividualParameter>` – assignment, structured models with linear and nonlinear covariate models, fixed and random effects and variability level reference
- version 0.7.3
  - `<IndividualParameter>` – assignment, structured models with linear and nonlinear covariate models, fixed and random effects; distribution model type and variability level reference
  - `<PopulationParameter>` – assignment or distribution model type and variability level reference
- version 0.8
  - `<Parameter>` – assignment
  - `<IndividualParameter>` – assignment, structured models with linear and nonlinear covariate models, fixed and random effects; distribution model type and variability level reference
  - `<PopulationParameter>` – assignment or distribution model type and variability level reference

In the majority of the use cases the `<SimpleParameter>` has been replaced by the `<PopulationParameter>` except models where `<Parameter>` is better suited. See for application examples such as models on drug-drug interaction, *../others/GrecoTung_drugDrugInteraction.xml*, and a PBPK model, *../others/Bradshaw-Pierce_2007.xml*. The last one is an update of a v0.4 model converted by the Cyprotex Model Writer.

2.14 Minor extensions

2.14.1 Datasets declaration

The declaration of the external dataset provides more flexibility. While in previous versions the allowed values for the mandatory `toolName` attribute were predefined with only options {Monolix, NONMEM, BUGS}, e.g.

```
<ExternalDataSet toolName="BUGS" oid="RdataSet">
    <ColumnMapping>
    <!-- ... -->
```

in version 0.8 any name can be used or the attribute can be skipped entirely so that the following

```
<ExternalDataSet oid="RdataSet">
    <ColumnMapping>
    <!-- ... -->
```

is sufficient. This change is essential to allow the use of any datasets and formats.

2.14.2 Conditional 'ignorance'

The conditional ignoring of data records is now supported, for example

```
IGNORE=(MDV0.EQ.1)
IGNORE=(STAGE.GT.5)
IGNORE=(STAGE.EQ.3)
```

would be implemented as

```
<ExternalDataSet oid="NMoid">
    <ds:DataSet>
        <ds:Definition>
            <ds:Column columnId="ID" columnType="id" valueType="string" columnNum="1"/>
            <ds:Column columnId="TIME" columnType="time" valueType="real" columnNum="2"/>
            <ds:Column columnId="Y" columnType="dv" valueType="real" columnNum="3"/>
```

```
                <ds:Column columnId="MDV0" columnType="covariate" valueType="real" columnNum="4"/>
                <ds:Column columnId="STAGE" columnType="covariate" valueType="real" columnNum="5"/>
                <ds:IgnoreLine>
                    <math:Condition>
                        <math:LogicBinop op="eq">
                            <ds:ColumnRef columnIdRef="MDV0"/>
                            <ct:Real>1</ct:Real>
                        </math:LogicBinop>
                    </math:Condition>
                </ds:IgnoreLine>
                <ds:IgnoreLine>
                    <math:Condition>
                        <math:LogicBinop op="gt">
                            <ds:ColumnRef columnIdRef="STAGE"/>
                            <ct:Real>5</ct:Real>
                        </math:LogicBinop>
                    </math:Condition>
                </ds:IgnoreLine>
                <ds:IgnoreLine>
                    <math:Condition>
                        <math:LogicBinop op="eq">
                            <ds:ColumnRef columnIdRef="STAGE"/>
                            <ct:Real>3</ct:Real>
                        </math:LogicBinop>
                    </math:Condition>
                </ds:IgnoreLine>
            </ds:Definition>
            <ds:ExternalFile oid="dataOid">
                <ds:path>example4.csv</ds:path>
                <ds:format>CSV</ds:format>
                <ds:delimiter>COMMA</ds:delimiter>
            </ds:ExternalFile>
        </ds:DataSet>
    </ExternalDataSet>
```

Note, that the default option, works without conditioning and is available unchanged, e.g.

```
<ds:IgnoreLine symbol="@"/>
```

See for more details the 0.7.3 specification [Swat, 2015].

Chapter 3

New applications using 0.8

3.1 Baseline models

Hansson model [Hansson et al., 2013] is a complex model and particularly interesting from the interoperability perspective. It contains number of component, both continuous and discrete, such as

- Biomarker model – ODEs and algebraic equations
- Model for tumor growth inhibition – ODEs and algebraic equations including a baseline model, based on [Dansirikul et al., 2008]
- Dropout model – logistic regression model (simulation only)
- Survival model – time-to-event data model

Here we discuss the baseline model because of its structure not discussed so far in any of the use cases. It requires the implementation of mapping of the dependent variable at time t=0. The NMTRAN code for the baseline model elements reads

```
IF(TIME.EQ.0.AND.FLAG.EQ.4)THEN
  OBASE = DV   ; observed tumor size at baseline (T= 0)
ENDIF

W1 = THETA(4)*OBASE
IBASE = OBASE+ETA(5)*W1  ; observed tumor size at baseline acknowledging residual error
...
A_0(4) = IBASE ; TUMOR
```

PharmML implementation of the baseline is done in two steps, first by declaring the variable (could also be declared as covariate), i.e.

```
<ct:Variable symbolType="real" symbId="OBASE"/>
```

which then can be mapped to the dependent variable, DV, column in the dataset conditional on TIME==0 as the following snippet shows

```
<ColumnMapping>
    <ColumnRef xmlns="http://www.pharmml.org/pharmml/0.7/Dataset" columnIdRef="DV"/>
    <Piecewise xmlns="http://www.pharmml.org/pharmml/0.7/Dataset">
        <math:Piece>
            <ct:SymbRef blkIdRef="sm1" symbIdRef="OBASE"/>
            <math:Condition>
                <math:LogicBinop op="eq">
                    <ColumnRef columnIdRef="TIME"/>
                    <ct:Real>0</ct:Real>
                </math:LogicBinop>
            </math:Condition>
        </math:Piece>
    </Piecewise>
</ColumnMapping>
```

Once this is done, the variable OBASE can be used to define the initial condition for the tumor growth variable defined by an ODE, $A4$,

$$\frac{A4}{dt} = \mathrm{KGA4} - [\mathrm{AUC1} + (-\mathrm{SKIT}) + (-\mathrm{VEG3})]\exp(-(\mathrm{LAMBDA} \times \mathrm{T}))\mathrm{A4}$$
$$A4(t = 0) = \mathrm{IBASE} = \mathrm{OBASE} + \mathrm{ETA}(5) * \mathrm{W1}$$

```
<ct:Variable symbolType="real" symbId="W1">
    <ct:Assign>
        <math:Binop op="times">
            <ct:SymbRef blkIdRef="pm1" symbIdRef="theta4"/>
            <ct:SymbRef symbIdRef="OBASE"/>
        </math:Binop>
    </ct:Assign>
</ct:Variable>

<!-- initial condition value, IBASE -->
<!-- IBASE = A4(t=0) = OBASE+ETA(5)*W1 -->
<ct:Variable symbolType="real" symbId="IBASE">
    <ct:Assign>
        <math:Binop op="plus">
            <ct:SymbRef blkIdRef="sm1" symbIdRef="OBASE"/>
            <math:Binop op="times">
                <ct:SymbRef symbIdRef="W1"/>
                <ct:SymbRef blkIdRef="pm1" symbIdRef="eta5"/>
            </math:Binop>
        </math:Binop>
    </ct:Assign>
</ct:Variable>

<!-- dA4/dt -->
<ct:DerivativeVariable symbolType="real" symbId="A4">
    <ct:Assign>
        <!-- skipped RHS expression: -->
        <!-- KG*A(4)-[AUC1+(-SKIT)+(-VEG3)]*EXP(-(LAMBDA*T))*A(4) -->
    </ct:Assign>
    <ct:InitialCondition>
        <ct:InitialValue>
            <ct:Assign>
                <ct:SymbRef blkIdRef="pm1" symbIdRef="IBASE"/>
            </ct:Assign>
        </ct:InitialValue>
        <ct:InitialTime>
            <ct:Assign><ct:Real>0</ct:Real></ct:Assign>
        </ct:InitialTime>
    </ct:InitialCondition>
</ct:DerivativeVariable>
```

3.2 Markov models

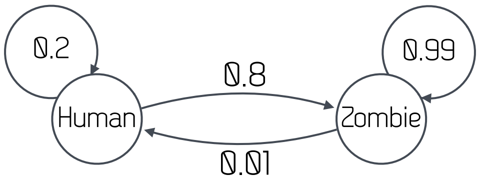
Stimulated by the recent discussion on the DDMoRe forum around Markov models[1] two examples are discussed with a detailed description and implementation.

It turns out that Markov models, beyond for example disease modelling demonstrated in the 2^{nd} example, have other quite unexpected yet relevant application such as zombie attack modelling. These attacks have been documented in well-known movies such as *Shaun of the Dead* although there many who see this as pure science fiction. Nevertheless it is a very educational use case described in the first example.

3.2.1 Example 1 – Zombie attack

The following basic zombie attack model is based on an example in lecture notes on Markov Models[2]. Its extension to that shown on the front page of this specification is straightforward. Other more advanced models are available as well, e.g. [Munz et al., 2009], [Witkowski and Blais, 2013], [Woolley et al., 2014].

[1] http://www.ddmore.eu/forum/pharmml-and-sbml
[2] http://www.poritz.net/jonathan/matvec/markov.html



	Human	Zombie
Human	0.2	0.8
Zombie	0.01	0.99

Figure 3.1: Basic Markov model for zombie attack.

Model definition

Observation model

- Type of observed variable – discrete / categorical
- Category variable: Y
- Initial state variable: Y_{init}
- Set of categories: $\{\text{Human}, \text{Zombie}\}$
- Transition probabilities
  - as pairwise conditional transition probabilities

$$
\begin{aligned}
P(\text{Human} \to \text{Zombie}) &= 0.8 \\
P(\text{Zombie} \to \text{Human}) &= 0.01 \\
P(\text{Human} \to \text{Human}) &= 0.2 \\
P(\text{Zombie} \to \text{Zombie}) &= 0.99
\end{aligned}
$$

  - or as transition (aka stochastic) matrix

$$
\begin{array}{cc}
 & \begin{array}{cc} H & Z \end{array} \\
\begin{array}{c} H \\ Z \end{array} & \begin{bmatrix} 0.1 & 0.8 \\ 0.01 & 0.99 \end{bmatrix}
\end{array}
$$

Trial Design

- Observations: Y at t=1,...,12 (months).

Modelling steps

- Initial states

$$
Y_{init} = \begin{pmatrix} 100 \\ 0 \end{pmatrix}
$$

PharmML implementation

```
<ModelDefinition xmlns="http://www.pharmml.org/pharmml/0.8/ModelDefinition">
    <!-- OBSERVATIONS -->
    <ObservationModel blkId="om1">
        <Discrete>
            <CategoricalData>

                <ListOfCategories>
                    <Category symbId="Human"/>
                    <Category symbId="Zombie"/>
                </ListOfCategories>

                <CategoryVariable symbId="Y"/>
                <InitialStateVariable symbId="Yinit"/>
                <PreviousStateVariable symbId="Yp"/>
```

```xml
                    <Dependance type="discreteMarkov"/>

                    <TransitionMatrix type="leftStochastic">
                        <ct:Matrix matrixType="Any">
                            <ct:RowNames>
                                <ct:SymbRef symbIdRef="Human"/><ct:SymbRef symbIdRef="Zombie"/>
                            </ct:RowNames>
                            <ct:MatrixRow>
                                <ct:Real>0.2</ct:Real><ct:Real>0.8</ct:Real>
                            </ct:MatrixRow>
                            <ct:MatrixRow>
                                <ct:Real>0.01</ct:Real><ct:Real>0.99</ct:Real>
                            </ct:MatrixRow>
                        </ct:Matrix>
                    </TransitionMatrix>

                    <!-- ALTERNATIVELY usign Pairwise probabilities -->
                    <!-- P(Y=Zombie|Yp=Human)=0.8 -->
                    <ProbabilityAssignment>
                        <Probability symbId="p1">
                            <CurrentState>
                                <math:LogicBinop op="eq">
                                    <ct:SymbRef symbIdRef="Y"/>
                                    <ct:SymbRef symbIdRef="Zombie"/>
                                </math:LogicBinop>
                            </CurrentState>
                            <PreviousState>
                                <math:LogicBinop op="eq">
                                    <ct:SymbRef symbIdRef="Yp"/>
                                    <ct:SymbRef symbIdRef="Human"/>
                                </math:LogicBinop>
                            </PreviousState>
                        </Probability>
                        <ct:Assign>
                            <ct:Real>0.8</ct:Real>
                        </ct:Assign>
                    </ProbabilityAssignment>
                    <!-- other probabilities analog - skipped here -->

                </CategoricalData>
            </Discrete>
        </ObservationModel>
    </ModelDefinition>
```

Trial design: output of variable Y at t=1,...,12 (months).

```xml
    <!-- OBSERVATION DEFINITION: Number of humans/zombies for months 1-12 -->
    <TrialDesign xmlns="http://www.pharmml.org/pharmml/0.8/TrialDesign">
        <Observations>
            <Observation oid="obsOid">
                <ObservationTimes>
                    <ct:Assign>
                        <ct:Sequence>
                            <ct:Begin>
                                <ct:Real>1</ct:Real>
                            </ct:Begin>
                            <ct:StepSize>
                                <ct:Real>1</ct:Real>
                            </ct:StepSize>
                            <ct:End>
                                <ct:Real>12</ct:Real>
                            </ct:End>
                        </ct:Sequence>
                    </ct:Assign>
                </ObservationTimes>
                <Discrete>
                    <ct:SymbRef blkIdRef="om1" symbIdRef="Y"/>
                </Discrete>
            </Observation>
        </Observations>
    </TrialDesign>
```

Modelling step definition and initial assignments, $Y_{init} = (100, 0)$

```xml
<mstep:ModellingSteps>
    <mstep:SimulationStep oid="simOid">

        <mstep:ObservationsReference>
            <ct:OidRef oidRef="obsOid"/>
        </mstep:ObservationsReference>

        <ct:VariableAssignment>
            <ct:SymbRef blkIdRef="om1" symbIdRef="Yinit"/>
            <ct:Assign>
                <ct:Vector>
                    <ct:VectorElements>
                        <ct:Real>100</ct:Real>
                        <ct:Real>0</ct:Real>
                    </ct:VectorElements>
                </ct:Vector>
            </ct:Assign>
        </ct:VariableAssignment>

        <mstep:Operation order="1" opType="Number of humans/zombies for months 1-12"/>
    </mstep:SimulationStep>
</mstep:ModellingSteps>
```

3.2.1.1 Zombies and epidemiology

Zombies attack modelling is more then just fun application of mathematics and has close parallels to epidemiological modelling – the zombies are walking representations of a contagion. See for more on: `http://www.livescience.com/38527-surviving-a-zombie-apocalypse-math.html`

3.2.2 Example 2 – HIV model

The second example is based on the published model [Lee et al., 2014] and describes Markov chain modelling analysis of HIV/AIDS progression. Figure 3.2 shows the four states

- S1 – vulnerable
- S2 – HIV infective
- S3 – clinical AIDS persons
- S4 – death

model and the according transition matrix which is *Race* dependent.

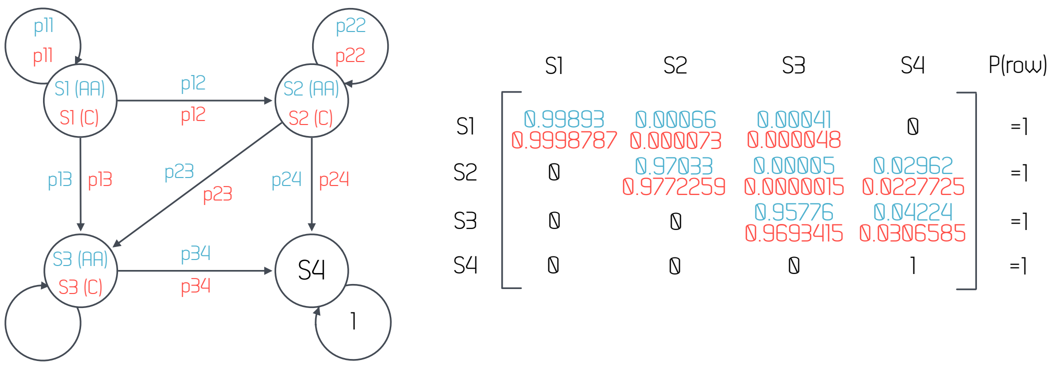


Figure 3.2: Markov model of HIV/AIDS progression.

Model definition

Covariate model

- Race = {African Americans (AA), Caucasians (C)} – categorical covariate

Observation model

- Type of observed variable – discrete / categorical
- Category variable: Y
- Initial state variable: Y_{init}
- Set of categories: $\{\mathrm{S1}, \mathrm{S2}, \mathrm{S3}, \mathrm{S4}\}$
- Transition probabilities
  - as pairwise conditional transition probabilities

$$
\begin{aligned}
&P(\mathrm{S1} \rightarrow \mathrm{S1}|\ Race == AA) = 0.99893 \\
&P(\mathrm{S1} \rightarrow \mathrm{S1}|\ Race == C) = 0.9998787 \\
&P(\mathrm{S1} \rightarrow \mathrm{S2}|\ Race == AA) = 0.00066 \\
&P(\mathrm{S1} \rightarrow \mathrm{S2}|\ Race == C) = 0.000073 \\
&P(\mathrm{S1} \rightarrow \mathrm{S3}|\ Race == AA) = 0.00041 \\
&P(\mathrm{S1} \rightarrow \mathrm{S3}|\ Race == C) = 0.000048 \\
&P(\mathrm{S1} \rightarrow \mathrm{S4}) = 0 \\
&\text{other probabilities follow from the property 'left stochastic matrix'}
\end{aligned}
$$

  - or as transition (aka stochastic) matrix – see Figure 3.2 (right)

Trial Design

- Observations: Y at t=1,...,10 (years).
- Covariates: Race=AA, C, AA, C, . . . , AA, C.

Modelling steps

- Initial states

$$
Y_{init} = \begin{pmatrix} 100 \\ 0 \\ 0 \\ 0 \end{pmatrix}
$$

PharmML implementation

- Covariate Model

```
<ModelDefinition xmlns="http://www.pharmml.org/pharmml/0.8/ModelDefinition">

    <!-- Covariate Model -->
    <CovariateModel blkId="cm1">
        <Covariate symbId="RACE">
            <Categorical>
                <Category catId="AA"/>
                <Category catId="C"/>
            </Categorical>
        </Covariate>
    </CovariateModel>
```

- Observation Model with transition matrix – we use the fact that a matrix element can contain an arbitrary expression, also a piecewise function used here for the transition probability from the *S1* to *S1*, *S2*, . . . , *S4* states conditioned on the covariate *Race*.

```xml
<!-- Observation Model -->
<ObservationModel blkId="om1">
    <Discrete>
        <CategoricalData>

            <ListOfCategories>
                <Category symbId="S1"/>
                <Category symbId="S2"/>
                <Category symbId="S3"/>
                <Category symbId="S4"/>
            </ListOfCategories>

            <CategoryVariable symbId="Y"/>

            <InitialStateVariable symbId="Yinit"/>
            <PreviousStateVariable symbId="Yp"/>

            <Dependance type="discreteMarkov"/>

            <TransitionMatrix type="leftStochastic">
                <ct:Matrix matrixType="Any">
                    <ct:RowNames>
                        <ct:SymbRef symbIdRef="S1"/>
                        <ct:SymbRef symbIdRef="S2"/>
                        <ct:SymbRef symbIdRef="S3"/>
                        <ct:SymbRef symbIdRef="S4"/>
                    </ct:RowNames>
                    <ct:MatrixRow>
                        <ct:Assign>
                            <ct:Piecewise>
                                <math:Piece>
                                    <ct:Real>0.99893</ct:Real>
                                    <math:Condition>
                                        <math:LogicBinop op="eq">
                                            <ct:SymbRef symbIdRef="RACE"/>
                                            <ct:CatRef catIdRef="AA"/>
                                        </math:LogicBinop>
                                    </math:Condition>
                                </math:Piece>
                                <math:Piece>
                                    <ct:Real>0.9998787</ct:Real>
                                    <math:Condition>
                                        <math:LogicBinop op="eq">
                                            <ct:SymbRef symbIdRef="RACE"/>
                                            <ct:CatRef catIdRef="C"/>
                                        </math:LogicBinop>
                                    </math:Condition>
                                </math:Piece>
                            </ct:Piecewise>
                        </ct:Assign>
                        <ct:Assign>
                            <ct:Piecewise>
                                <math:Piece>
                                    <ct:Real>0.00066</ct:Real>
                                    <math:Condition>
                                        <math:LogicBinop op="eq">
                                            <ct:SymbRef symbIdRef="RACE"/>
                                            <ct:CatRef catIdRef="AA"/>
                                        </math:LogicBinop>
                                    </math:Condition>
                                </math:Piece>
                                <math:Piece>
                                    <ct:Real>0.000073</ct:Real>
                                    <math:Condition>
                                        <math:LogicBinop op="eq">
                                            <ct:SymbRef symbIdRef="RACE"/>
                                            <ct:CatRef catIdRef="C"/>
                                        </math:LogicBinop>
                                    </math:Condition>
                                </math:Piece>
                            </ct:Piecewise>
                        </ct:Assign>
```

```
            <ct:Assign>
                <ct:Piecewise>
                    <math:Piece>
                        <ct:Real>0.00041</ct:Real>
                        <math:Condition>
                            <math:LogicBinop op="eq">
                                <ct:SymbRef symbIdRef="RACE"/>
                                <ct:CatRef catIdRef="AA"/>
                            </math:LogicBinop>
                        </math:Condition>
                    </math:Piece>
                    <math:Piece>
                        <ct:Real>0.000048</ct:Real>
                        <math:Condition>
                            <math:LogicBinop op="eq">
                                <ct:SymbRef symbIdRef="RACE"/>
                                <ct:CatRef catIdRef="C"/>
                            </math:LogicBinop>
                        </math:Condition>
                    </math:Piece>
                </ct:Piecewise>
            </ct:Assign>
            <ct:Real>0</ct:Real>
        </ct:MatrixRow>
        <ct:MatrixRow>
            <!-- 2nd row skipped -->
        </ct:MatrixRow>
        <ct:MatrixRow>
            <!-- 3rd row skipped -->
        </ct:MatrixRow>
        <ct:MatrixRow>
            <ct:Real>0</ct:Real><ct:Real>0</ct:Real><ct:Real>0</ct:Real><ct:Real>1</ct:Real>
        </ct:MatrixRow>
    </ct:Matrix>
</TransitionMatrix>
```

- Trial design
  - Output of variable Y at t=1,...,10 (years).
  - Covariates: Race=AA, C, AA, C, . . . , AA, C.

```
<TrialDesign xmlns="http://www.pharmml.org/pharmml/0.8/TrialDesign">

    <!-- <Observations> skipped as identical to that in previous example -->

    <Covariates>
        <IndividualCovariates>
            <ColumnMapping>
                <ds:ColumnRef columnIdRef="race"/>
                <ct:SymbRef blkIdRef="cm1" symbIdRef="RACE"/>
            </ColumnMapping>
            <ds:DataSet>
                <ds:Definition>
                    <ds:Column columnId="ID" valueType="string" columnNum="1"/>
                    <ds:Column columnId="race" valueType="string" columnNum="2"/>
                </ds:Definition>
                <ds:Table>
                    <ds:Row><ct:String>1</ct:String><ct:String>AA</ct:String></ds:Row>
                    <ds:Row><ct:String>2</ct:String><ct:String>C</ct:String></ds:Row>
                    <ds:Row><ct:String>3</ct:String><ct:String>AA</ct:String></ds:Row>
                    <ds:Row><ct:String>4</ct:String><ct:String>C</ct:String></ds:Row>
                    <ds:Row><ct:String>5</ct:String><ct:String>AA</ct:String></ds:Row>
                    <!-- subject omitted -->
                    <ds:Row><ct:String>99</ct:String><ct:String>AA</ct:String></ds:Row>
                    <ds:Row><ct:String>100</ct:String><ct:String>C</ct:String></ds:Row>
                </ds:Table>
            </ds:DataSet>
        </IndividualCovariates>
    </Covariates>
</TrialDesign>
```

- Modelling step definition is omitted as analog to those in the previous example.

3.2.3 Out of scope

Compared to the discussion on the DDMoRe forum website around the Markov models not all requested features, especially around the task execution, are covered by this PharmML version. From the start we have focused on a declarative model description and don't cover many structures featured in a typical programming language.

Nevertheless, additional features could be build in into the format both on model definition and task description side. It will require an additional discussion and well defined requirements to cover them if needed.

Chapter 4

Changes in 0.8.1

4.1 Support for setting and output files

4.1.1 Software specific settings

The element <SoftwareSettings> has been added to common task types (estimation and simulation) description allowing for optional storage of software specific settings – in agreement with recent MDL task proposal. Note that optimal design tasks can use this option already since version 0.7.2, [Swat et al., 2015].

4.1.2 Output files

This new element <OutputFile> offers the specification of the

- target SO file
- or a target-specific results files

as the example code below shows

```
<ModellingSteps xmlns="http://www.pharmml.org/pharmml/0.8/ModellingSteps">

    <TargetTool oid="targetTool">
        <TargetToolName>PFIM</TargetToolName>
    </TargetTool>

    <!-- Standard Output file -->
    <OutputFile oid="resultsSO">
            <ds:path>task_XYZ_SO.xml</ds:path>
    </OutputFile>

    <!-- or alternatively any number of output files -->
    <OutputFile oid="res1">
            <ds:path>results1.csv</ds:path>
    </OutputFile>

    <OutputFile oid="res2">
            <ds:path>results2.csv</ds:path>
    </OutputFile>
    <!-- ... -->
```

Note, that only the <path> is mandatory, <format> and <delimiter> are optional.

4.2 Optimal design extensions

Optimal design (OD) and related task has been extended with few missing components.

4.2.1 Parameter settings

The element <ParametersToEstimate> has been added to the optimal design task description – an element featured in estimation task already. Its reuse here allows for specification of parameters initial values, lower/upper bound values assignment or indication of which parameters are to be kept fixed.

4.2.2 *Stage* definition

Although described in the optimal design specification, [Comets et al., 2015b], we have managed to ignore it when implementing the OD support in PharmML v0.7.2, [Swat et al., 2015], which are now fully supported.

The following examples are borrowed from the OD example collection [Comets et al., 2015a] and visualise the options available for the new `<StageDefinition>` element

- Single stage specification

  ```
  DS{name=t1, element=doseTime, range=[0,6], stage = 1 }
  ```

  with PharmML implementation

  ```
  <StageDefinition>
      <math:LogicBinop op="eq">
          <math:Stage/>
          <ct:Real>1</ct:Real>
      </math:LogicBinop>
  </StageDefinition>
  ```

- Stage set specification

  ```
  DS{name=t1, element=doseTime, range=[0,8], stage = {1,2,3} }
  ```

  with PharmML implementation

  ```
  <StageDefinition>
      <math:LogicBinop op="eq">
          <design:Stage/>
          <ct:Vector>
              <ct:VectorElements>
                  <ct:Real>1</ct:Real>
                  <ct:Real>2</ct:Real>
                  <ct:Real>3</ct:Real>
              </ct:VectorElements>
          </ct:Vector>
      </math:LogicBinop>
  </StageDefinition>
  ```

- Interval stage specification

  ```
  DS{name=t1, element=doseTime, range=[0,8], stage > 1 & stage < 4 }
  ```

  with PharmML implementation

  ```
  <StageDefinition>
      <math:LogicBinop op="or">
          <math:LogicBinop op="gt">
              <Stage/>
              <ct:Int>1</ct:Int>
          </math:LogicBinop>
          <math:LogicBinop op="lt">
              <Stage/>
              <ct:Int>4</ct:Int>
          </math:LogicBinop>
      </math:LogicBinop>
  </StageDefinition>
  ```

4.2.3 FIM encoding

New attribute `type` has been introduced in the `<FIM>` element. It defines the type of the FIM-atrix by assigning one of the allowed values $\{B, I, P\}$, e.g.

```
<FIM type="P"/>
```

The matrix encoding support available previously has been removed as redundant.

4.3 Dataset definition

4.3.1 New attribute `level`

Allows to indicate levels of variability in SO files, see for detailed discussion the SO v0.3.1 specification document, [Terranova et al., 2016].

4.3.2 Extensions in `columnType`

Following changes in the dataset declaration were required because of demands both in PharmML and SO[1] such as

- new values of the `columnType` attribute have been introduced such as
  - *varLevel*
  - (only relevant for SO) *variance*, *stdev*, *mode*, *median*
- few values has been removed such as *varParameter_corr*, *varParameter_cov*, *varParameter_stdev*, *varParameter_var*
- allowing using multiple values in combination e.g.

```
columnType="covariate varLevel"
```

This relatively minor extension has a number of applications discussed in section 4.4 and the newest SO specification, [Terranova et al., 2016].

4.4 Mapping of variability levels

Assigning multiple values to the `columnType` attribute is often required. The new value *varLevel* introduced above is especially useful in combination with value *covariate*, i.e. when a dataset column is a covariate and simultaneously provides a reason to consider additional variability level and must therefore be linked to the variability model.

4.4.1 IOV mapping

It is worth noting that previously the only `columnType` attribute value related to higher variability levels was *occasion*. This situation is illustrated with the following example of a dataset definition

```
<ds:Definition>
    <ds:Column columnId="ID" columnType="id" valueType="string" columnNum="1"/>
    <ds:Column columnId="TIME" columnType="time" valueType="real" columnNum="2"/>
    <ds:Column columnId="Y" columnType="dv" valueType="real" columnNum="3"/>
    <ds:Column columnId="AMT" columnType="dose" valueType="real" columnNum="4"/>
    <ds:Column columnId="OCC" columnType="covariate occasion" valueType="int" columnNum="5"/>
    <!-- ... -->
</ds:Definition>
```

In this case a double mapping of the OCC column is required, i.e.

- covariate mapping and
- variability model mapping

as shown in the following snippet

```
<ColumnMapping>
    <ds:ColumnRef columnIdRef="OCC"/>
    <ct:SymbRef blkIdRef="cm1" symbIdRef="Occasion"/>
    <ds:CategoryMapping>
        <ds:Map modelSymbol="occ1" dataSymbol="1"/>
        <ds:Map modelSymbol="occ2" dataSymbol="2"/>
    </ds:CategoryMapping>
```

[1] The schema of the Standard Output (SO) reuses certain PharmML constructs, such as data declaration support, but it is otherwise independent from PharmML.

```
        </ColumnMapping>
        <ColumnMapping>
            <ds:ColumnRef columnIdRef="OCC"/>
            <ct:SymbRef blkIdRef="vm1" symbIdRef="iov1"/>
        </ColumnMapping>
```

This solution works but only for inter-occasion variability, higher levels could not be accordingly annotated.

The usefulness of *varLevel* value of the `columnType` attribute is that it provides a hint about the column (additional) use and the need to map it to the variability model. It is more importantly generic, i.e. it can be applied for any variability level. The above dataset definition can therefore be extended and reads (only the relevant column is shown)

```
    <ds:Definition>
        <!-- ... -->
        <ds:Column columnId="OCC" columnType="covariate varLevel" valueType="int" columnNum="5"/>
        <!-- ... -->
    </ds:Definition>
```

This 'annotation' of the dataset column can also be used for validation purposes whether the mapping to the variability and covariate model has been declared or not.

4.4.2 Higher levels of variability

The usage of the *varLevel* extends easily to cases when dealing with multiple higher levels of variability. We consider here a design (in its explicit form) with an additional inter-country variability level located above the subject level as shown in Figure 4.1

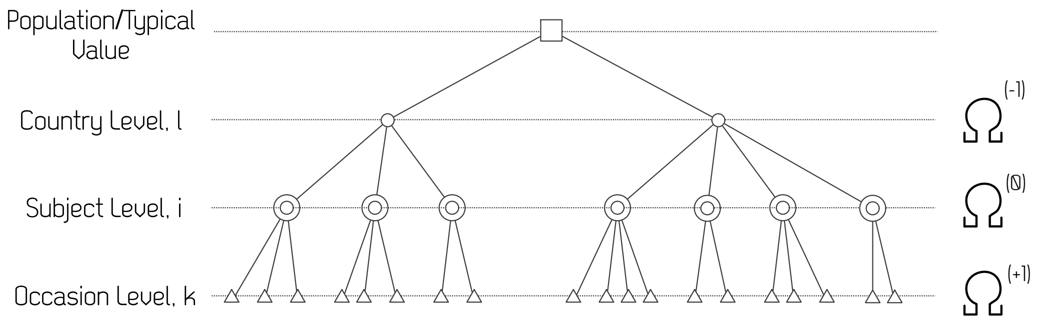


Figure 4.1: Higher variability levels.

The following code snippet illustrates the implementation in PharmML of this complex variability model

```
    <VariabilityModel blkId="vm1" type="parameterVariability">
        <Level symbId="country"/>
        <Level referenceLevel="true" symbId="indiv">
            <ParentLevel>
                <ct:SymbRef symbIdRef="country"/>
            </ParentLevel>
        </Level>
        <Level symbId="iov">
            <ParentLevel>
                <ct:SymbRef symbIdRef="indiv"/>
            </ParentLevel>
        </Level>
    </VariabilityModel>
```

'Country' covariate is stored, when using the explicit design, in `<IndividualCovariates>` tag. The dataset definition and mappings read

```
    <Covariates>
        <IndividualCovariates>
            <ColumnMapping>
                <ds:ColumnRef columnIdRef="COUNTRY"/>
                <ct:SymbRef blkIdRef="cm1" symbIdRef="Country"/>
            </ColumnMapping>
            <ColumnMapping>
```

```
            <ds:ColumnRef columnIdRef="COUNTRY"/>
            <ct:SymbRef blkIdRef="vm1" symbIdRef="country"/>
        </ColumnMapping>
        <!-- ... -->
        <ds:DataSet>
            <ds:Definition>
                <ds:Column columnId="ID" columnType="id" valueType="string" columnNum="1"/>
                <ds:Column columnId="ARM" columnType="arm" valueType="string" columnNum="2"/>
                <!-- ... -->
                <ds:Column columnId="COUNTRY" columnType="covariate varLevel" valueType="string" columnNum="5"/>
            </ds:Definition>
            <ds:Table>
```

As before the *varLevel* value allows to indicate the double use of column COUNTRY and the need to map it accordingly to a level of the variability model.

4.5 Changed in the schema

4.5.1 Attribute symbolType for variables is optional

Now, instead of providing the mandatory attribute e.g.

```
<ct:Variable symbId="CONC" symbolType="real">
```

it can be skipped and simply encoded as

```
<ct:Variable symbId="CONC">
```

The same applies to <DerivativeVariable>.

4.5.2 Integer declaration unification

The following data types[2]

- integer – an integer value
- positiveInteger – an integer containing only positive values (1,2,..)

are replaced by

- int – signed 32-bit integer

The positivity validation on certain elements where this is desired, and were previously encoded using positiveInteger, will be performed in the libPharmML API.

4.5.3 Piecewise namespace correction

It was pointed out that the <Piecewise> element occurs in more than one namespace. This was due to using the related type, for example

```
<xs:element name="Piecewise" type="math:PiecewiseType"/>
```

instead of a reference

```
<xs:element ref="math:Piecewise"/>
```

This redundancy has been now removed and the piecewise statement is consistently declared across all PharmML schemas.

For some of use cases it means that those using this element have to be corrected. For example in one of the Product 4.1 use examples instead of

```
<design:ColumnMapping>
        <ds:ColumnRef columnIdRef="AMT"/>
        <ds:Piecewise>
                <math:Piece>
                        <ct:SymbRef blkIdRef="sm" symbIdRef="GUT"/>
                        <math:Condition>
```

[2] All definition are provided on http://www.w3schools.com/xml/schema_dtypes_numeric.asp

```
                        <math:LogicBinop op="gt">
                            <ds:ColumnRef columnIdRef="AMT"/>
                            <ct:Int>0</ct:Int>
                        </math:LogicBinop>
                    </math:Condition>
                </math:Piece>
            </ds:Piecewise>
        </design:ColumnMapping>
```

the namespace prefix has to be changed from `ds` to `math` i.e.

```
        <design:ColumnMapping>
            <ds:ColumnRef columnIdRef="AMT"/>
            <math:Piecewise>
                <math:Piece>
                    <ct:SymbRef blkIdRef="sm" symbIdRef="GUT"/>
                    <math:Condition>
                        <math:LogicBinop op="gt">
                            <ds:ColumnRef columnIdRef="AMT"/>
                            <ct:Int>0</ct:Int>
                        </math:LogicBinop>
                    </math:Condition>
                </math:Piece>
            </math:Piecewise>
        </design:ColumnMapping>
```

In general, this change should not impact any of the converters using the libPharmML.

4.6 ODEs in conditionals

Ordinary differential equations are from now on allowed in `<ConditionalStatement>` which means an extension of the rules defined in section 2.3.3. As with variables and parameters, the `<DerivativeVariable>` has to be first declared at the root level of `<StructuralModel>` using the attribute `symbId` and then it can be assigned in the conditional statement if required using `symbIdRef` as the following example shows

```
<ct:DerivativeVariable symbId="Q1"/>
<ct:DerivativeVariable symbId="Q2"/>

<ConditionalStatement>
    <math:If>
        <math:Condition>
            <math:LogicBinop op="lt">
                <ct:SymbRef symbIdRef="t"/>
                <ct:SymbRef symbIdRef="tmax1"/>
            </math:LogicBinop>
        </math:Condition>
        <ct:DerivativeVariable symbIdRef="Q1">
            <ct:Assign>
                <!-- omitted RHS -->
            </ct:Assign>
        </ct:DerivativeVariable>
        <ct:DerivativeVariable symbIdRef="Q2">
            <ct:Assign>
                <!-- omitted RHS -->
            </ct:Assign>
        </ct:DerivativeVariable>
    </math:If>
    <math:ElseIf>
        <math:Condition>
            <math:LogicBinop op="geq">
                <ct:SymbRef symbIdRef="t"/>
                <ct:SymbRef symbIdRef="tmax1"/>
            </math:LogicBinop>
        </math:Condition>
        <ct:DerivativeVariable symbIdRef="Q1">
            <ct:Assign>
                <!-- omitted RHS -->
            </ct:Assign>
        </ct:DerivativeVariable>
        <ct:DerivativeVariable symbIdRef="Q2">
            <ct:Assign>
```

```
                <!-- omitted RHS -->
            </ct:Assign>
        </ct:DerivativeVariable>
    </math:ElseIf>
    <math:Else>
        <!-- something else -->
    </math:Else>
</ConditionalStatement>
```

4.7 Minor changes

- `gamma` function has been introduced
- Encoding of standard normal distribution – it is a suggested change in the encoding practise. Its implementation became very easy with ProbOnto. Instead of the encoding $N(0,1)$ via the Normal1 parameterisation with the lengthily code

```
<RandomVariable symbId="epsilon_Css">
    <ct:VariabilityReference>
        <ct:SymbRef blkIdRef="vm2" symbIdRef="resErr"/>
    </ct:VariabilityReference>
    <Distribution>
        <po:ProbOnto name="Normal1">
            <po:Parameter name="mean">
                <ct:Assign>
                    <ct:Real>0</ct:Real>
                </ct:Assign>
            </po:Parameter>
            <po:Parameter name="stdev">
                <ct:Assign>
                    <ct:Real>1</ct:Real>
                </ct:Assign>
            </po:Parameter>
        </po:ProbOnto>
    </Distribution>
</RandomVariable>
```

  it is simply enough to declare the available standard normal distribution with code name `StandardNormal1` as following snippet shows

```
<RandomVariable symbId="epsilon_Css">
    <ct:VariabilityReference>
        <ct:SymbRef blkIdRef="vm2" symbIdRef="resErr"/>
    </ct:VariabilityReference>
    <Distribution>
        <po:ProbOnto name="StandardNormal1"/>
    </Distribution>
</RandomVariable>
```

Bibliography

[Beal, 2001] Beal, S. L. (2001). Ways to fit a pk model with some data below the quantification limit. *Journal of Pharmacokinetics and Pharmacodynamics*, 28(5):481–504.

[Beal et al., 2009] Beal, S. L., Sheiner, L. B., Boeckmann, A. J., and Bauer, R. J. (2009). NONMEM User's Guides. (1989-2009). Technical report, Icon Development Solutions, Ellicott City, MD, USA.

[Comets et al., 2015a] Comets, E., Chenel, M., and Hooker, A. (2015a). Modelling Description Language, Design elements - Examples, Draft 1 version 2. Technical report, INSERM, UPD; Servier; Uppsala University.

[Comets et al., 2015b] Comets, E., Chenel, M., and Hooker, A. (2015b). Modelling Description Language, Design elements, Draft 1 version 2. Technical report, INSERM, UPD; Servier; Uppsala University.

[Dansirikul et al., 2008] Dansirikul, C., Silber, H. E., and Karlsson, M. O. (2008). Approaches to handling pharmacodynamic baseline responses. *Journal of pharmacokinetics and pharmacodynamics*, 35(3):269–283.

[Fisher and Shafer, 2007] Fisher, D. and Shafer, S. (2007). Fisher/Shafer NONMEM Workshop Pharmacokinetic and Pharmacodynamic Analysis with NONMEM - Basic Concepts. *Het Pand, Ghent, Belgium.*

[Greco et al., 1990] Greco, W. R., Park, H. S., and Rustum, Y. M. (1990). Application of a new approach for the quantitation of drug synergism to the combination of cis-diamminedichloroplatinum and 1-beta-d-arabinofuranosylcytosine. *Cancer Res*, 50(17):5318–27.

[Hansson et al., 2013] Hansson, E. K., Amantea, M. A., Westwood, P., Milligan, P. A., Houk, B. E., French, J., Karlsson, M. O., and Friberg, L. E. (2013). Pkpd modeling of vegf, svegfr-2, svegfr-3, and skit as predictors of tumor dynamics and overall survival following sunitinib treatment in gist. *CPT Pharmacometrics Syst Pharmacol*, 2:e84.

[Lavielle, 2014] Lavielle, M. (2014). *Mixed Effects Models for the Population Approach: Models, Tasks, Methods and Tools*. Chapman & Hall/CRC Biostatistics Series.

[Lee et al., 2014] Lee, S., Ko, J., Tan, X., Patel, I., Balkrishnan, R., and Chang, J. (2014). Markov chain modelling analysis of hiv/aids progression: A race-based forecast in the united states. *Indian J Pharm Sci*, 76(2):107–15.

[Lixoft Team, 2014a] Lixoft Team (2014a). MLXTRAN, The model coding language for Monolix. Technical report, INRIA Saclay & Lixoft.

[Lixoft Team, 2014b] Lixoft Team (2014b). MLXTRAN, The model coding language for Simulator. Technical report, INRIA Saclay & Lixoft.

[Lunn et al., 2009] Lunn, D., Spiegelhalter, D., Thomas, A., and Best, N. (2009). The BUGS project: Evolution, critique and future directions. *Stat Med*, 28(25):3049–67.

[Mould and Upton, 2013] Mould, D. R. and Upton, R. N. (2013). Basic Concepts in Population Modeling, Simulation, and Model-Based Drug Development – Part 2: Introduction to Pharmacokinetic Modeling Methods. *CPT Pharmacometrics Syst Pharmacol*, 2(e38).

[Munz et al., 2009] Munz, P., Hudea, I., Imad, J., and Smith, R. J. (2009). When zombies attack!: mathematical modelling of an outbreak of zombie infection. *Infectious Disease Modelling Research Progress*, 4:133–150.

[STAN Development Team, 2015] STAN Development Team (2015). Stan Modeling Language Users Guide and Reference Manual, Version 2.9.0.

[Swat, 2015] Swat, M. J. (2015). Extentions in PharmML 0.7.3. Technical report, EMBL-EBI.

[Swat and Grenon, 2015] Swat, M. J. and Grenon, P. (2015). ProbOnto 0.3. Technical report, EMBL-EBI, Hinxton, UK; UCL, UK.

[Swat et al., 2015] Swat, M. J., Grenon, P., Yvon, F., Wimalaratne, S., and Kristensen, N. R. (2015). Extentions in PharmML 0.7-0.7.2. Technical report, EMBL-EBI.

[Terranova et al., 2016] Terranova, N., Lavielle, M., Smith, M. K., Comets, E., Nordgren, R., Edwards, D., Sarr, C., Harling, K., Hooker, A. C., Mentré, F., Yvon, F., and Swat, M. J. (2016). Standard Output (SO): Format Specification for Version 0.3.1. Technical report, DDMoRe SO group.

[UncertML Team, 2014] UncertML Team (2014). Uncertainty Markup Language (UncertML) Version 3.0. URL: http://www.uncertml.org.

[Witkowski and Blais, 2013] Witkowski, C. and Blais, B. (2013). Bayesian analysis of epidemics-zombies, influenza, and other diseases. *arXiv preprint arXiv:1311.6376*.

[Woolley et al., 2014] Woolley, T. E., Baker, R. E., Gaffney, E. A., and Maini, P. K. (2014). How long can we survive?