Contents

1. Parameter container: complete example

1.1 Model

Description

This model is based on plug-flow reactor (PFR) example (Levenspiel, 1999; page 106). PFR can be viualised as a tube whith input steam flowing into one side and output streams coming from another side. Inside the tube some conversion occurs that alters the composition of the input stream. The typical characteristic of a (ideal) PFR are:

- Conversion is gradual and
- There is no back-mixing of the flowing stream.

These characteristics make it easy to write mass-balance of PFR find the extent of conversion of the input stream. Conversion is defined as one minus the ratio of what is coming out over what came in, or more formally, conversion of the species A is

$$X_A = \left(1 - \frac{F_{A,out}}{F_{A,in}}\right) \times 100\% \tag{1.1}$$

where F_A is molar (or mass) flow rate (in, for example, mol/h). For bravity, we will not be deriing any mass balance equations for PFR, but only use a couple of results from Levenspiel, 1999.

PFR is proposed for the decomposition of phosphine

$$4PH_{3g} \rightarrow P_{4(g)} + 6H_2$$

Denote $A = PH_3$, $B = P_4$ and $S = H_2$. Reaction rate (i.e. the rate at which A disappears) is defined using equation:

$$-r_A = kC_A \tag{1.2}$$

with rate constant k being dependent on temperature according to *Arrhenius law*:

$$k = k_0 e^{-E/R(1/T-1/T_0)} \tag{1.3}$$

The volume of the reactor required to achieve the conversion X_A is defined by the equation (Levenspiel, 1999; equation 21):

$$V = \frac{F_{A0}}{kC_{A0}} \left[(1+\varepsilon_A)\ln\frac{1}{1-X_A} - \varepsilon_A X_A\right] \tag{1.4}$$

where F_{A0} is input flow rate of A and C_{A0} is initial concentration of A. Constant $\varepsilon_A = (7-4)/4 = 0.75$ is defined via stoichiometry (fractional change in volume).

Concentration can be determined from ideal gas law:

$$C_{A0} = \frac{p_{A0}}{RT} \tag{1.5}$$

where p_{A0} is initial partial pressure of A, R is universal gas constant and T is operating temperature.

This model can be used to estimate the volume of the reactor given economical constraints. A default parameters are provided as a guidence but are the subject to change (except for constant R).

Parameter	Default value	Units
k_0	10	1/h
E	160	kJ/mol
T_0	600	C
T	650	C
F_{A0}	40	mol/h
p	4	atm
R	8.314	J/ mol K

Our goal is to develop computer representation of this model.

Reaction rate representation

First, we need to represent the temperature dependence of the rate constant

```
(defconstant +universal-gas-constant+ 8.314d0
  "Universal gas constant R=8.314 J/mol K")

(defclass arrhenius-law ()
  ((reference-rate-constant
    :initarg :reference-rate-constant
    :accessor reference-rate-constant
    :documentation "Reference rate constant (k0)")
   (activation-energy
    :initarg :activation-energy
    :accessor activation-energy
    :documentation "Activation energy (Ea) in J/mol")
   (reference-temperature
    :initarg :reference-temperature
    :accessor reference-temperature
    :documentation "Reference temperature (T0) in K"))
  (:documentation
   "Arrhenius law of the reaction rate constant:
    k(T) = k0 * exp[- E/R * (1/T - 1/T0)]"))

(defmethod print-object ((object arrhenius-law) out)
  (with-slots ((k0 reference-rate-constant)
               (e activation-energy)
               (t0 reference-temperature))
      object
    (print-unreadable-object (object out :type t)
      (format out "~@<~:_k0 = ~A ~:_Ea = ~A J/mol ~:_T0 = ~A K~:>"
              k0 e t0))))

(defmethod rate-constant ((model arrhenius-law) temperature)
  (with-accessors ((k0 reference-rate-constant)
                   (e activation-energy)
                   (t0 reference-temperature))
      model
    (* k0 (exp (- (* (/ e +universal-gas-constant+) (- (/ temperature) (/ t0))))))))
```

Ideal gas law

To find the concentration from given pressure and temperature:

```
(defun ideal-gas-concentration (pressure temperature)
  "Calculates concentration (mol/m3) based in PRESSURE (Pa) and
temperature (K) using ideal gas law: c = p / RT"
  (/ pressure (* +universal-gas-constant+ temperature)))
```

Plug-flow reactor model for phosphine decomposition

Plug flow reactor model is composed of the following parts:

- Inlet flow rate F_{A0},
- Final (target) conversion X_A,
- Operating temperature T,
- Operating pressure p, and
- Reaction rate constant defined by Arrhenius law.

```
(defclass PFR-phosphine ()
  ((inlet-flow-rate
    :initarg :inlet-flow-rate
    :documentation "F_A0 in mol/s"
    :accessor inlet-flow-rate)
   (final-conversion
    :initarg :final-conversion
    :accessor final-conversion
    :documentation "Target conversion X_A")
   (operating-temperature
    :initarg :operating-temperature
    :accessor operating-temperature
    :documentation "PFR operating temperature (K)")
   (operating-pressure
    :initarg :operating-pressure
    :accessor operating-pressure
    :documentation "PFR operating pressure (Pa)")
   (phosphine-decomposition-rate-coefficient
    :initarg :rate-coefficient
    :accessor phosphine-decomposition-rate-coefficient
    :documentation
    "Rate constant 'k' object specializing RATE-CONSTANT generic function"))
  (:documentation
   "Plug-flow-reactor for phosphine decomposition
    4PH3 -> P4 + 6H2"))

(defmethod print-object ((object PFR-phosphine) out)
  (with-slots ((f inlet-flow-rate)
               (x final-conversion)
               (temperature operating-temperature)
               (p operating-pressure)
               (rate-expression phosphine-decomposition-rate-coefficient))
      object
    (print-unreadable-object (object out :type t)
      (format out "~@<~:_F = ~A mol/s ~:_X = ~A ~:_T = ~A K ~:_p = ~A Pa~:>"
              f x temperature p))))
```

Use equation (**??**) to calculate volume of the reactor:

```
(defmethod PFR-phosphine-volume ((model pfr-phosphine))
  "Uses equation (Levenspiel, 1999; equation 21):
     F    _                          _
      A0|               1             |
V = ----| (1 + e )ln{------}  - e X   |
    kC  |       A    1 - X       A A  |
      A0|_                A          _|

to calculate the volume of the plug-flow reactor
"
  (with-accessors ((f inlet-flow-rate)
                   (x final-conversion)
                   (temperature operating-temperature)
                   (p operating-pressure)
                   (rate phosphine-decomposition-rate-coefficient))
      model
    (let ((k (rate-constant rate temperature))
          (eps 0.75d0)
          (c (ideal-gas-concentration p temperature)))
      (* (/ f (* k c))
         (- (* (1+ eps) (log (/ (- 1 x))))
            (* eps x))))))
```

Model parameters

The model consists of a large number of parameters. To complicate the matter further, some parameters can be represented in different units of measure. Just to keep the description short, the only different units parameter we will consider is temperature. To assist in defining the temperature in different units (Celcius, Kelvin or Fahrenheit) we will use the following function:

```
(defun make-temperature (default-value default-units name id)
  "Produces PARAMETER-OPTIONS parameter for temperature, represented
in different units: Celsius, Kelvin and Fahrenheit. Arguments:

DEFAULT-VALUE : default numerical value to be used
DEFAULT-UNITS : a keyword (:C, :K or :F) denoting default units
NAME          : a string name of the resulting PARAMETER-OPTIONS
ID            : a symbolic ID of the resulting PARAMETER-OPTIONS"
  (let* ((units (reverse
                  (remove-duplicates
                   (reverse (cons default-units '(:C :K :F))))))
         (values-units (mapcar
                        (lambda (to-units)
                          (list
                           to-units
                           (convert-temperature default-units
                                                to-units
                                                default-value)))
                        units)))
    (parameter
     :name name
     :id id
     :options
     (loop for (units value) in values-units
        collect (let ((units units))
                  (parameter
                   :name (format nil "Temperature in ~A" units)
                   :id units
                   :value value
                   :units (format nil "~A" units)
                   :constructor (lambda (x) (convert-temperature units :K x))))))))
```

This function produces a value of type `PARAMETER-OPTIONS` to specify the temperature in convenient units. It makes sure that the provided `DEFAULT-UNITS` will be the default choice for units. It uses `CONVERT-TEMPERATURE` to provide the conversions between different units:

```
(defmethod convert-temperature ((from (eql :C)) (to (eql :C)) value)
  "Converts temperature VALUE from units FROM to units TO"
  value)

(defmethod convert-temperature ((from (eql :F)) (to (eql :F)) value)
  value)

(defmethod convert-temperature ((from (eql :K)) (to (eql :K)) value)
  value)

(defmethod convert-temperature ((from (eql :F)) (to (eql :C)) value)
  (* 5/9 (- value 32d0)))

(defmethod convert-temperature ((from (eql :C)) (to (eql :F)) value)
  (+ 32d0 (* 9/5 value)))

(defmethod convert-temperature ((from (eql :C)) (to (eql :K)) value)
  (+ value 273.15d0))

(defmethod convert-temperature ((from (eql :K)) (to (eql :C)) value)
  (- value 273.15d0))

(defmethod convert-temperature ((from (eql :F)) (to (eql :K)) value)
  (convert-temperature :C :K (convert-temperature :F :C value)))

(defmethod convert-temperature ((from (eql :K)) (to (eql :F)) value)
  (convert-temperature :C :F (convert-temperature :K :C value)))
```

Arrhenius law parameters are grouped together to form `ARRHENIUS-LAW` object and the rest: to form `PFR-PHOSPHINE`.

Firts, Arrhenius law parameters:

```
(defun make-phosphine-default-arrhenius-law ()
  (parameter
   :name "Arrhenius law for phosphine decomposition"
   :id :rate-coefficient
   :children
   (list
    (parameter
     :name "Reference rate, k0"
     :id :reference-rate-constant
     :value 10d0
     :units "1/h"
     :constructor (lambda (x) (/ x 3600d0)))
    (parameter
     :name "Activation energy, Ea"
     :id :activation-energy
     :value 160d0
     :units "kJ/mol"
     :constructor (lambda (x) (* x 1000d0)))
    (make-temperature 649d0 :C "Reference temperature, T0" :reference-temperature))
   :constructor (lambda (&rest args)
                  (apply #'make-instance 'arrhenius-law args))))
```

In PFR model parameters, to make things a bit more interesting and help with testing, temperature is provided as an option of temperature in either Celcius or Fahrenheit.

```
(defun make-default-pfr-phosphine ()
  (parameter
   :name "PFR for phosphine decomposition"
   :children
   (list
    (parameter
     :name "Inlet flow rate"
     :value 40d0
     :perturbation 0.2d0
     :units "mol/h"
     :constructor (lambda (x) (/ x 3600d0)))
    (parameter
     :name "Final conversion"
     :value 80d0
     :units "%"
     :constructor (lambda (x) (/ x 100d0)))
    (make-temperature 1200.2d0 :F "Operating temperature" :operating-temperature)
    (parameter
     :name "Operating pressure"
     :value 4.55d0
     :units "atm"
     :constructor (lambda (x) (* x 1.01d5)))
    (make-phosphine-default-arrhenius-law))
   :constructor (lambda (&rest args)
                  (apply #'make-instance 'PFR-phosphine args))))

(defvar *PFR-synonyms*
  '(:temperature :operating-temperature
    :ref-rate :reference-rate-constant
    :ref-temperature :reference-temperature))
```