Contents

1. Parameter container: complete example

1.1 Model

Description

This model is based on plug-flow reactor example (Levenspiel, 1999; page 106). Plug flow reactor 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 is defined using Arrhenius law

$$-r_A = kC_A \tag{1.1}$$

with

$$k = k_0 e^{-E/R(1/T - 1/T_0)} \tag{1.2}$$

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

$$V = \frac{F_{A0}}{kC_{A0}} \left[(1+\varepsilon_A) \ln \frac{1}{1-X_A} - \varepsilon_A X_A \right] \tag{1.3}$$

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:

$$C_{A0} = \frac{p_{A0}}{RT} \tag{1.4}$$

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

Reaction rate representation

First 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.

```
(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-constant
    :initarg :rate-constant
    :accessor phosphine-decomposition-rate-constant
    :documentation
    "Rate constant 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-constant))
      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))))
```

Calculate volume of the reactor:

```
(defmethod PFR-phosphine-volume ((model pfr-phosphine))
  (with-accessors ((f inlet-flow-rate)
                   (x final-conversion)
                   (temperature operating-temperature)
                   (p operating-pressure)
                   (rate phosphine-decomposition-rate-constant))
      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. 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-constant
   :children
   (list
    (parameter
     :name "Reference rate, k0"
     :id :reference-rate-constant
     :value 10d0
     :units "1/h"
     :value-transformer (lambda (x) (/ x 3600d0)))
    (parameter
     :name "Activation energy, Ea"
     :id :activation-energy
     :value 160d0
     :units "kJ/mol"
```

```
     :value-transformer (lambda (x) (* x 1000d0)))
    (parameter
     :name "Reference temperature, T0"
     :id :reference-temperature
     :value 650d0
     :units "C"
     :value-transformer (lambda (x) (+ x 273.15d0))))
   :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
     :units "mol/h"
     :value-transformer (lambda (x) (/ x 3600d0)))
    (parameter
     :name "Final conversion"
     :value 80d0
     :units "%"
     :value-transformer (lambda (x) (/ x 100d0)))
    (parameter
     :name "Operating temperature"
     :options
     (list
      (parameter
       :name "Operating temperature (C)"
       :value 650d0
       :units "C"
       :value-transformer (lambda (x) (+ x 273.15d0)))
      (parameter
       :name "Operating temperature (F)"
       :value 1202d0
       :units "F"
       :value-transformer (lambda (x) (+ 273.15d0 (* 5/9 (- x 32d0)))))))
    (parameter
     :name "Operating pressure"
     :value 4.54d0
```

```
     :units "atm"
     :value-transformer (lambda (x) (* x 1.01d5)))
    (make-phosphine-default-arrhenius-law))
   :constructor (lambda (&rest args)
                  (apply #'make-instance 'PFR-phosphine args))))
```

1.2 Tests

Use FiveAM. Put all the tests into their own suite.

```
(def-suite pfr-phosphine-suite)

(in-suite pfr-phosphine-suite)
```

Make sure `PARAMETER-VALUE` on `PARAMETER-CONTAINER` returns itself.

```
(test test-parameter-container-value
  (let ((arrhenius-law (make-phosphine-default-arrhenius-law)))
    (is (eq (parameter-value arrhenius-law)
            arrhenius-law))))
```

Instantiated object is of the right type

```
(test test-parameter-container-instantiate-object
  (let ((arrhenius-law (make-phosphine-default-arrhenius-law)))
    (let ((result (instantiate-object arrhenius-law)))
      (is (typep result 'arrhenius-law)))))
```

Container must instantiate objects recursively and the parameters must be properly adjusted

```
(test test-composite-parameter-container-instantiate-object
  (let ((pfr (make-default-pfr-phosphine)))
    ;; Checks if it propogates and converts
    (setf (parameter-value (parameter-ref pfr :inlet-flow-rate))
          50d0)
    (let ((result (instantiate-object pfr)))
      (is (typep result 'PFR-phosphine))
      (is (typep (phosphine-decomposition-rate-constant
                  result)
                 'arrhenius-law))
      (is (approx= (inlet-flow-rate result) (/ 50d0 3600d0))))))
```

Finally, this test combines the test on parameters and the model: compare the result with the result from Levenspiel, 1999; Example 5.5:

```
(test test-pfr-volume "Compare result with Levenspiel's example"
  (let* ((pfr-parameters (make-default-pfr-phosphine))
         (pfr-model (instantiate-object pfr-parameters))
```

```
        (volume (pfr-phosphine-volume pfr-model))
        (levenspiel-result 0.148d0))
    (is (approx= levenspiel-result volume :abstol 1d-3 :reltol 1d-3))))
```

To run the tests:

(let ((**debug-on-error** t)) (run! 'pfr-phosphine-suite))

1.3 To run the interface

To run the interface part, execute the following:

(parameters-interface:show-model (make-default-pfr-phosphine))