FMECAENGINE

a seamless tool to chain migration simulations and perform related sensitivity analysis

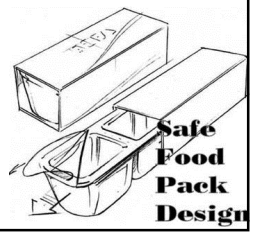
Version 0.45

(Windows & Linux i386/x64)

olivier.vitrac@agroparistech.fr

SUPPORTED BY ANR

August, 2011



LICENSE

FMECAengine is developed by Olivier Vitrac, PhD (Institut National de la Recherche Agronomique) and distributed under licence CeCILL-B (compatible GPL) as detailed below.

CeCILL-B FREE SOFTWARE LICENSE AGREEMENT
http://www.cecill.info/licences/Licence_CeCILL-B_V1-en.html

Notice

This Agreement is a Free Software license agreement that is the result of discussions between its authors in order to ensure compliance with the two main principles guiding its drafting:

- firstly, compliance with the principles governing the distribution of Free Software: access to source code, broad rights granted to users,
- secondly, the election of a governing law, French law, with which it is conformant, both as regards the law of torts and intellectual property law, and the protection that it offers to both authors and holders of the economic rights over software.

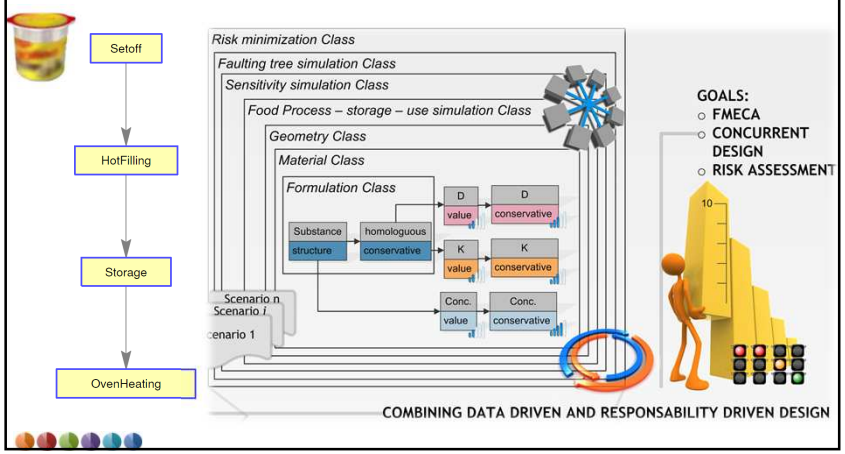
GOALS

- **FMECAengine implements FMECA concepts** (http://en.wikipedia.org/wiki/FMECA) while being compatible with the underlying physics of mass transfer between food packaging and food.
- **a formalism as an expert system** that enables the simulation of various migration scenarios and the coupling with heterogeneous databases, rulebases and knwoledge bases from industry, final users, research laboratory…
- **Possible scopes**: identification of critical steps within a food packaging flow chart (setoff, hot filling, long term storage, household heating…); seeking critical migrants for various designs (e.g. from adhesives, inks); concurrent design of complex food packaging systems (multicomponents, multilayers).

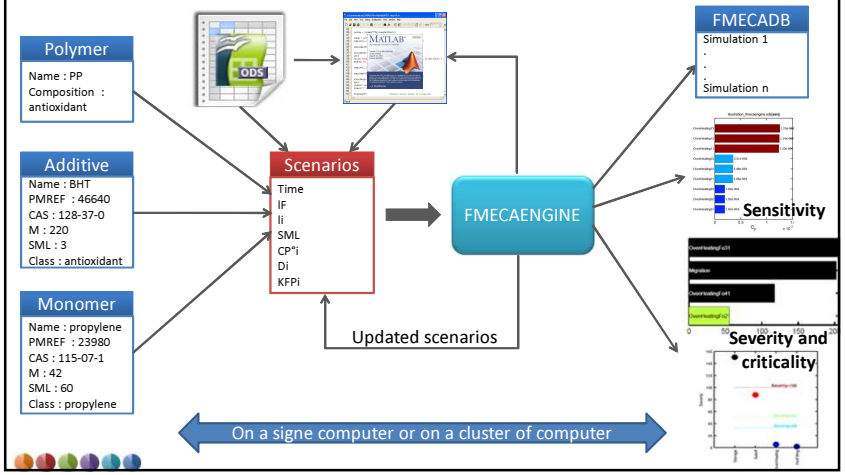
VERSION HISTORY

- Generating migration scenarios (version 0.1)
- Simulating chained scenarios (version 0.2)
- Sensitivity analysis (version 0.3)
- Severity and criticality analysis (version 0.38)
- Connecting with physico-chemical databases (version 0.41)
- Multiple keys enabled in versions 0.43 and later

Which kind of problem can be solved ?



FMECAENGINE architecture



WHAT IS CHAINED SIMULATION ?

Principles : inherited parameters, parental bond

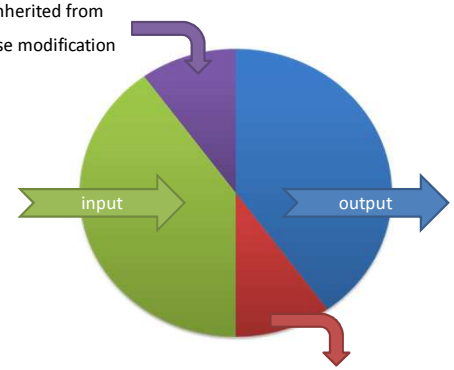
Toy example

What is migration simulation?

Parameter vector inherited from step k $[\underline{p}]_k$ + sparse modification $[\underline{\delta p}]_n$

Result from a previous simulation

(e.g. time x space concentration field denoted $[\underline{c}]_{n-1}$)



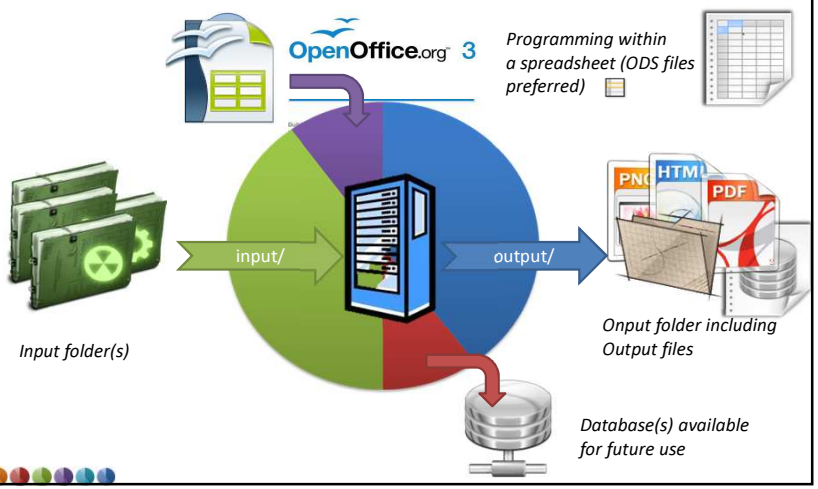
Simulated result

(denoted $[\underline{c}]_n$)

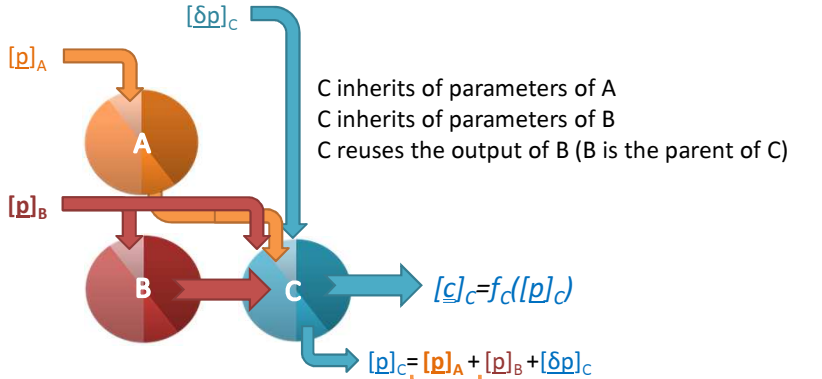
$[\underline{p}]_n = [\underline{p}]_k + [\underline{\delta p}]_n$

to propagate inheritance

Object/parallel implementation

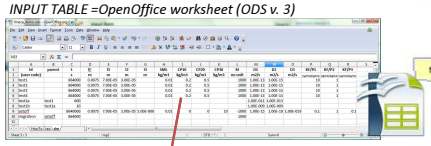


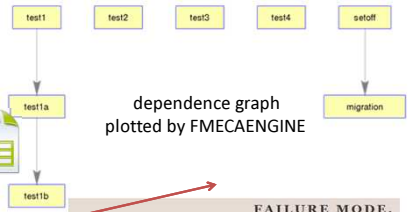
GOAL: solve complex simulation scenarios with simple inheritance operators and sparse parameters



TOY EXAMPLE: input table

INPUT TABLE =OpenOffice worksheet (ODS v. 3)





dependence graph plotted by FMECAENGINE

SAFEFOODPACK DESIGN

FAILURE MODE, EFFECTS, AND CRITICALITY ANALYSIS (FMECA)

FMECA_DEMO.ODS | 18-APR-2011 15:46:19 | 12.3 KBYTES | /HOME/OLIVIER/MAINGUYEN/ANR_SFPD/WP2_FMECA/DEMO/INPUTS

id	parent	t	l_f	l_1	l_2	l_3	SML	CP_1^0	CP_2^0	CP_3^0	Bi	D_1	D_2	D_3	KFP_1	KFP_2	KFP_3
usercode		s	m	m	m	m	kg/m³	kg/m³	kg/m³	kg/m³	no unit	m²/s	m²/s	m²/s	(kg/m³)/(kg/m³)	(kg/m³)/(kg/m³)	(kg/m³)/(kg/m³)
test1		864000	0.0075	7e-05	3e-05		0.006	0.2	0.5		1000	1e-13	1e-15		10	1	
test2		864000	0.0075	7e-05	3e-05		0.006	0.2	0.5		1000	1e-13	1e-15		10	1	
test3		864000	0.0075	7e-05	3e-05		0.006	0.2	0.5		1000	1e-13	1e-15		10	1	
test4		864000	0.0075	7e-05	3e-05		0.006	0.2	0.5		1000	1e-13	1e-15		10	1	
test1a	test1	600										1e-11	1e-13				
test1b	test1a	10										1e-09	1e-09				
setoff		8.64e+06	0.0075	7e-05	3e-05	5e-06	0.006	0	0	10	-1000	1e-15	1e-18	1e-19	0.1	1	
migration	setoff	864000									1000						
INPUT TABLE	8 simulation definitions						date	18-May-2011 16:06:27									

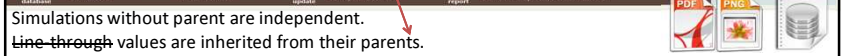
ORIGINAL TABLE repainted by FMECAENGINE

TOY EXAMPLE: output table

input » dependence graph (pdf) » result graph (pdf) » Pareto chart (pdf) » Kinetics (pdf) » Severity chart (pdf)

Chaining defined via ancestors

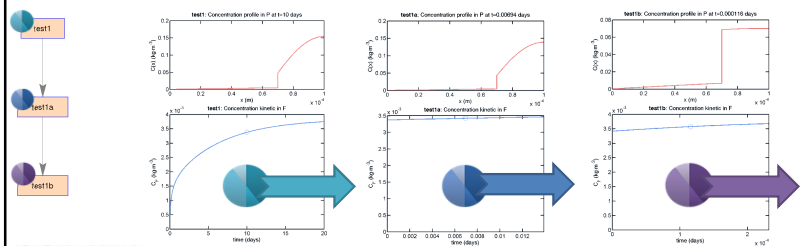
Results available as links



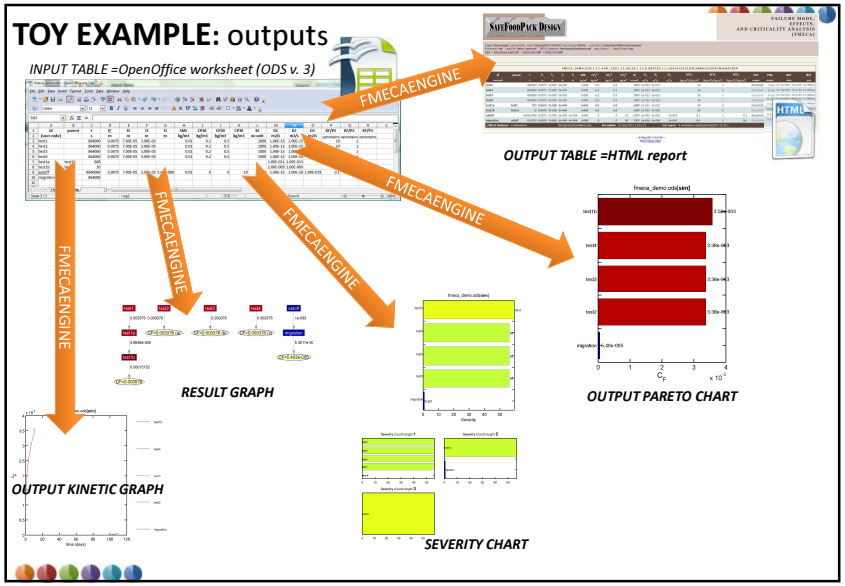
Simulations without parent are independent.

~~Line-through~~ values are inherited from their parents.

Links give direct access to plots.



TOY EXAMPLE: outputs



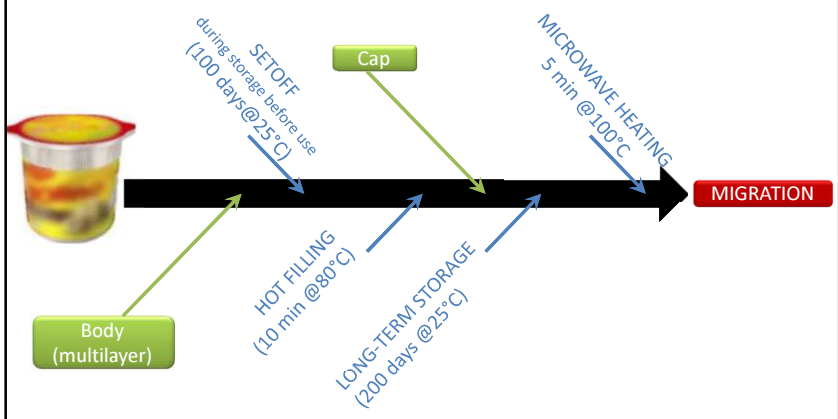
SENSIBILITY ANALYSIS

Principles : automatic creation of combinations (Foscale, Kscale)

Illustrative examples

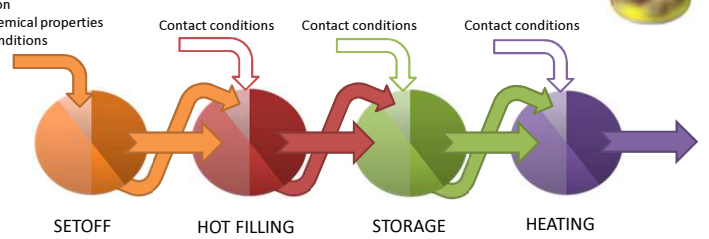
CASE STUDY

simulation chaining to assess cause trees



CORRESPONDING CODING

Geometry
Composition
Physico-chemical properties
Contact conditions



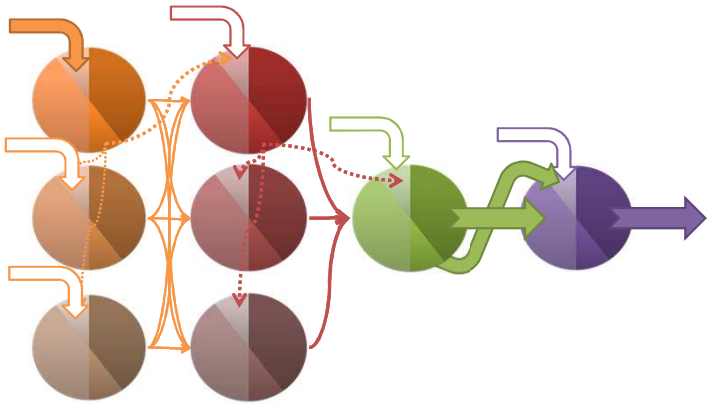
As most of values are inherited, the table is highly sparse.
Note that negative Bi values code for setoff (periodic boundary conditions).

ILLUSTRATION_FMECAENGINE.ODS | 28-AVR.-2011 22:06:43 | 23.4 KBYTES | C:\DATA\OLIVIER\INRA\ETUDIANTS & VISITEURS\AUDREY GOUJON\ILLUSTRATION_FMECA\INPUTS

row	id	parent	inherit	t	l_F	l_1	l_2	SML	CP_1^0	CP_2^0	Bi	D_1	D_2	KFP_1	KFP_2
row	usercode		inherit	s	m	m	m	kg/m²	kg/m²	kg/m²	no unit	m²/s	m²/s	(kg/m²)/(kg/m²)	(kg/m²)/(kg/m²)
1	Setoff1			8.64e+006	0.0115385	0.001	5e-005	0.0006	0	0.5	-1000	3.3e-014	1.2e-016	10	1
2	HotFilling11	Setoff1		600				0			1000	1.77e-011	2e-013		
3	Storage11	HotFilling11		1.728e+007				0				3.3e-014	1.2e-016		
4	OvenHeating11	Storage11		300				0				1.1e-010	3e-012		

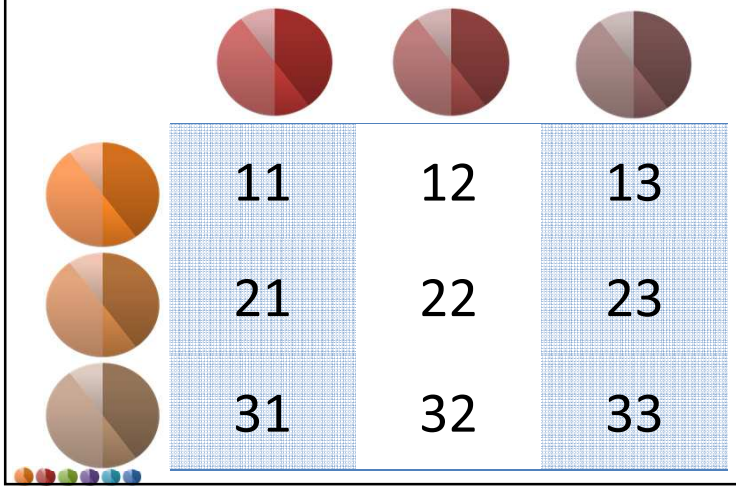
SENSITIVITY OF ANALYSIS

An example: 3 levels for each first two steps



PRINCIPLES OF SENSITIVITY OF ANALYSIS

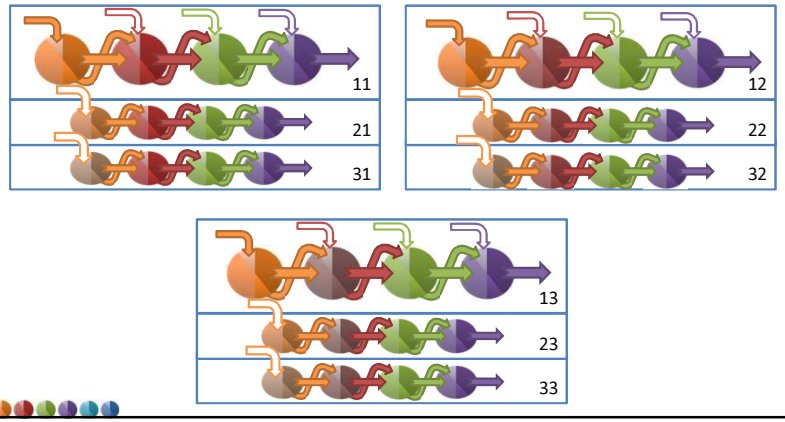
9 levels combinations



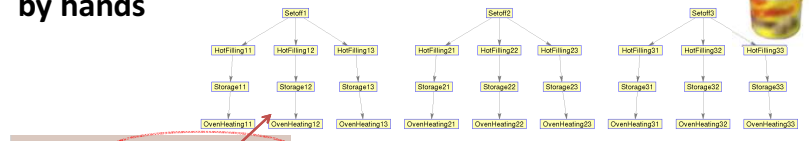
PRINCIPLES OF SENSITIVITY OF ANALYSIS

APPLICATION TO SIMULATION CHAINING

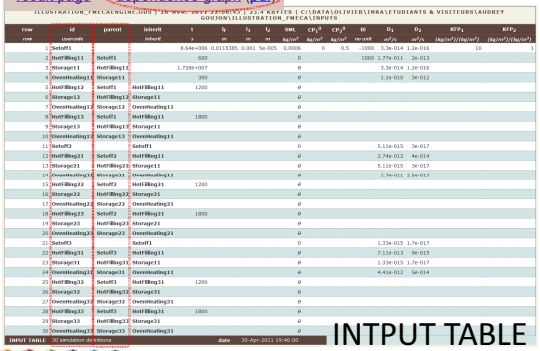
Mandatory rule = each step must have a single parent



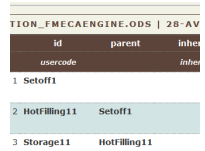
A FIRST INTUITIVE IMPLEMENTATION: by hands



result page » dependence graph (pdf)



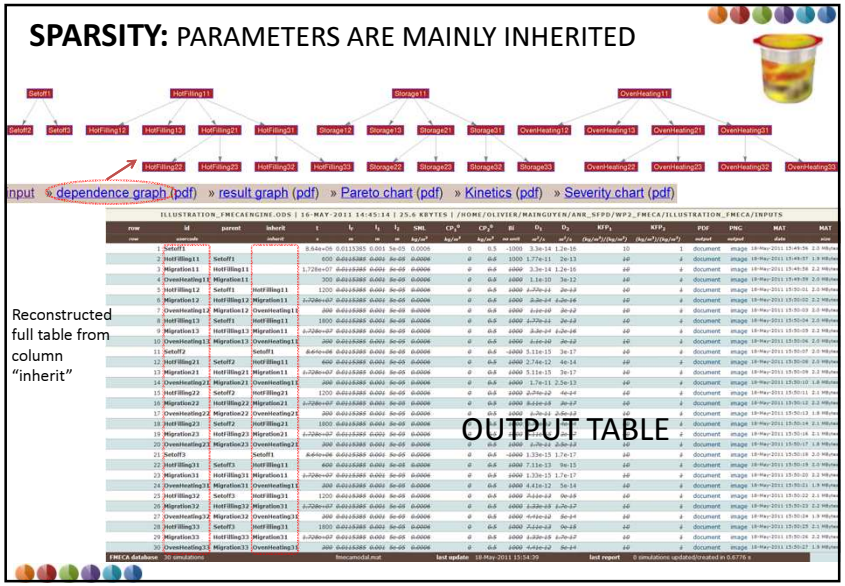
INTPUT TABLE



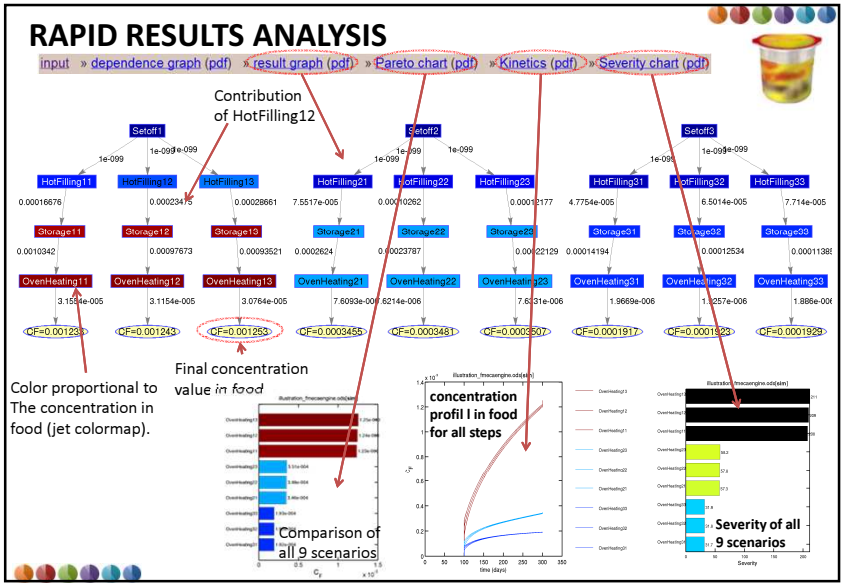
Relationship defined with “id” and “parent”

Note the sparsity of inputs

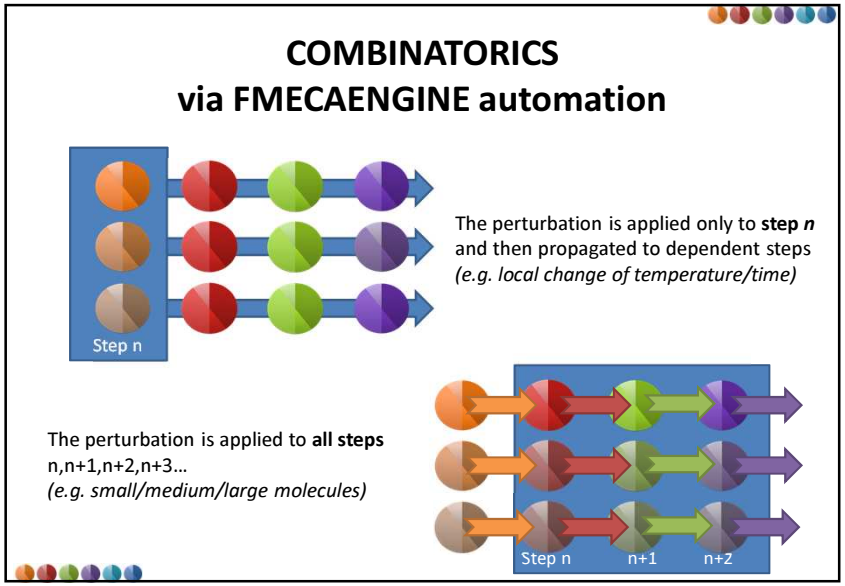
SPARSITY: PARAMETERS ARE MAINLY INHERITED



RAPID RESULTS ANALYSIS



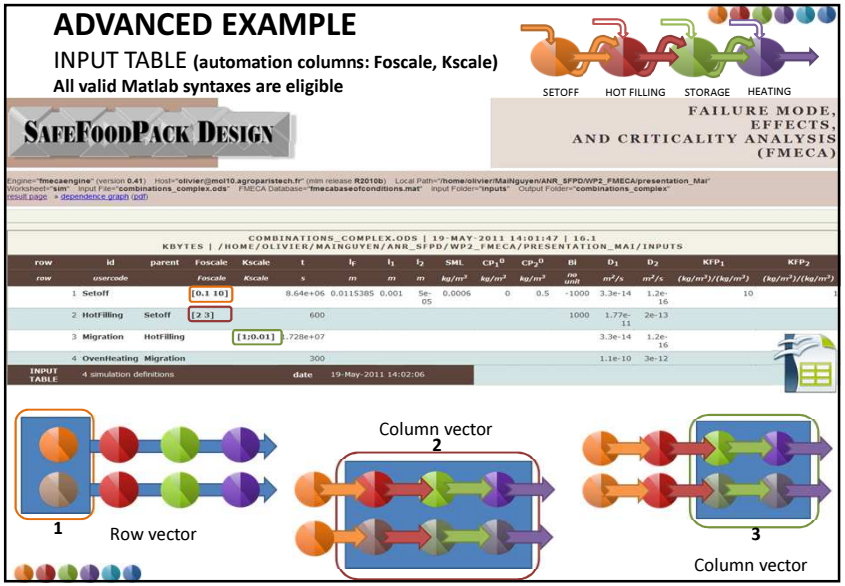
COMBINATORICS via FMECAENGINE automation



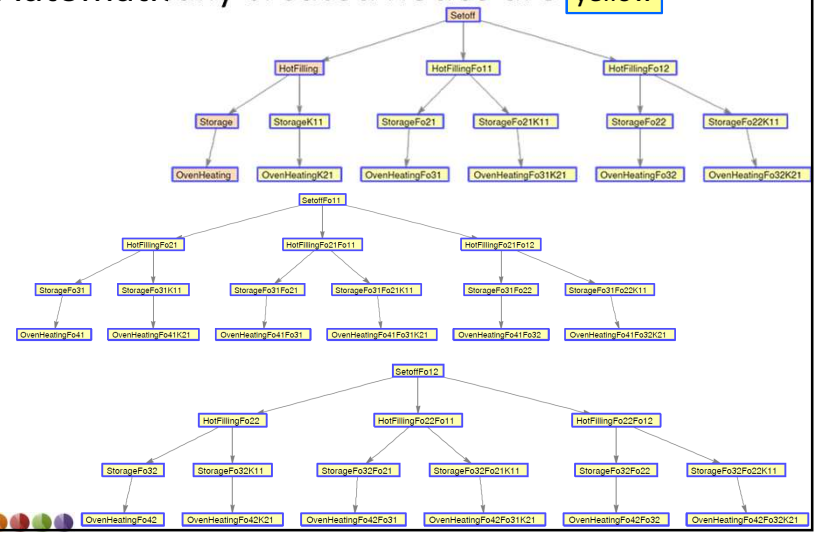
ADVANCED EXAMPLE

INPUT TABLE **(automation columns: Foscale, Kscale)**

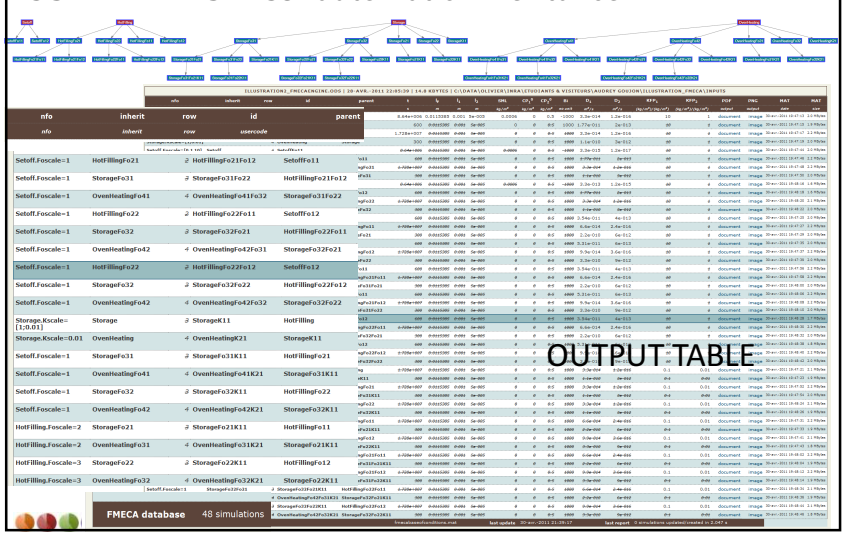
All valid Matlab syntaxes are eligible



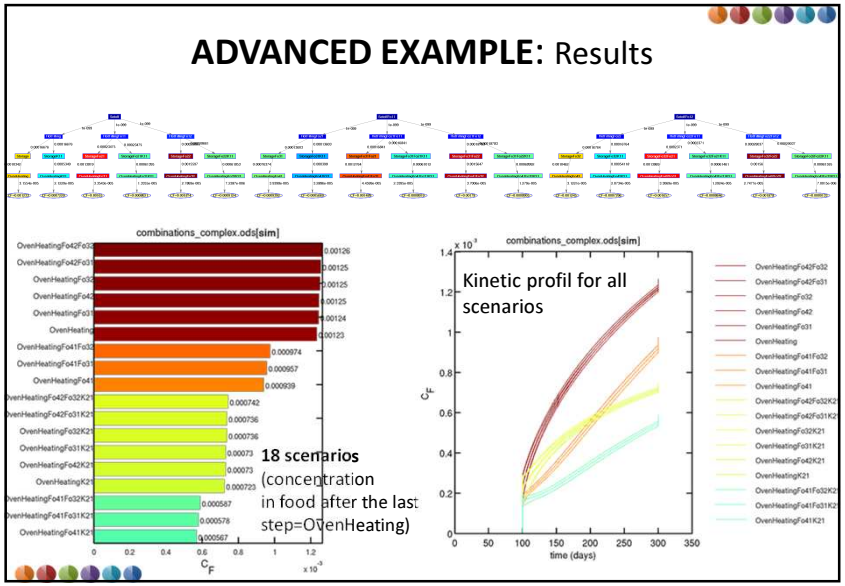
Automatically created nodes are yellow



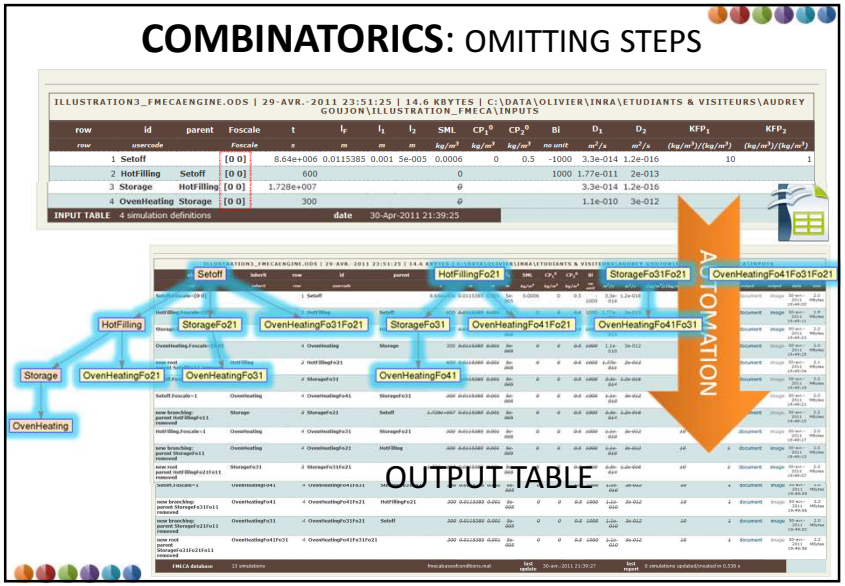
COMBINATORICS: automatic inheritance



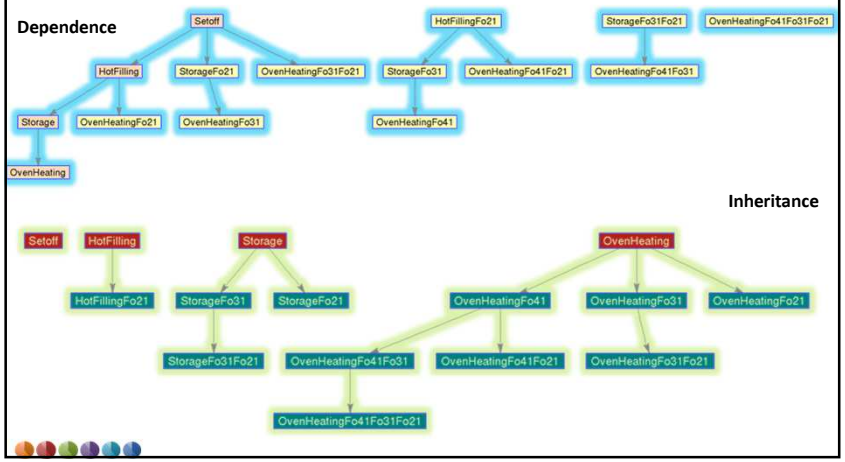
ADVANCED EXAMPLE: Results



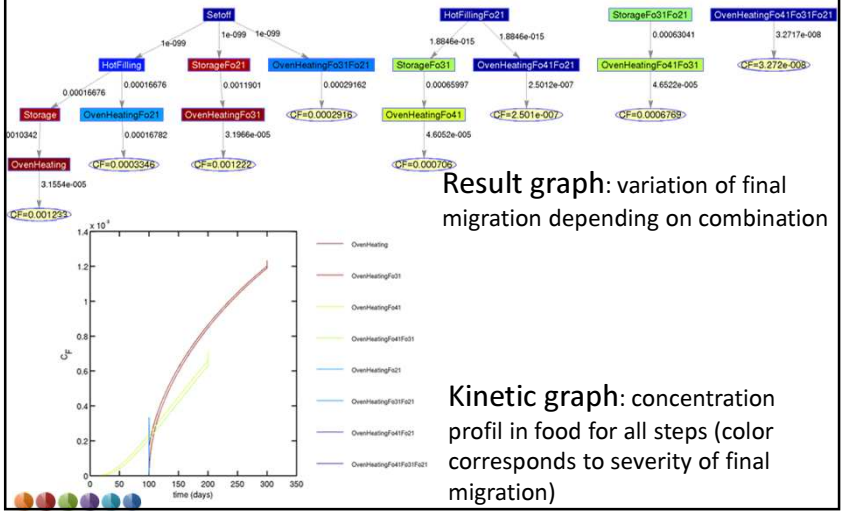
COMBINATORICS: OMITTING STEPS



OMITTING STEPS: FMECAENGINE builds all trees required to assess the effects of all combinations



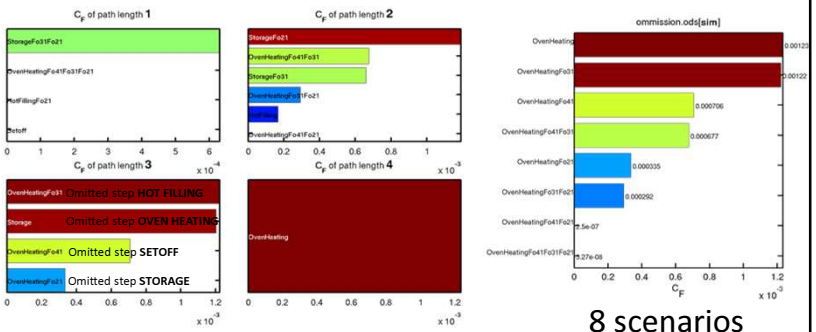
OMITTING STEPS: Results



Result graph: variation of final migration depending on combination

Kinetic graph: concentration profil in food for all steps (color corresponds to severity of final migration)

OMITTING STEPS: Results



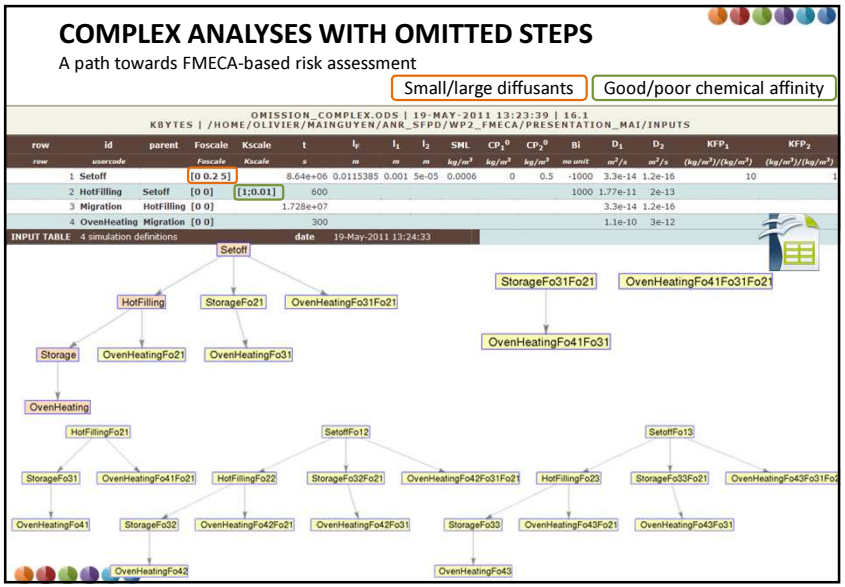
Final concentration in food for each scenario length

8 scenarios (concentration in food after the last step=OvenHeating)

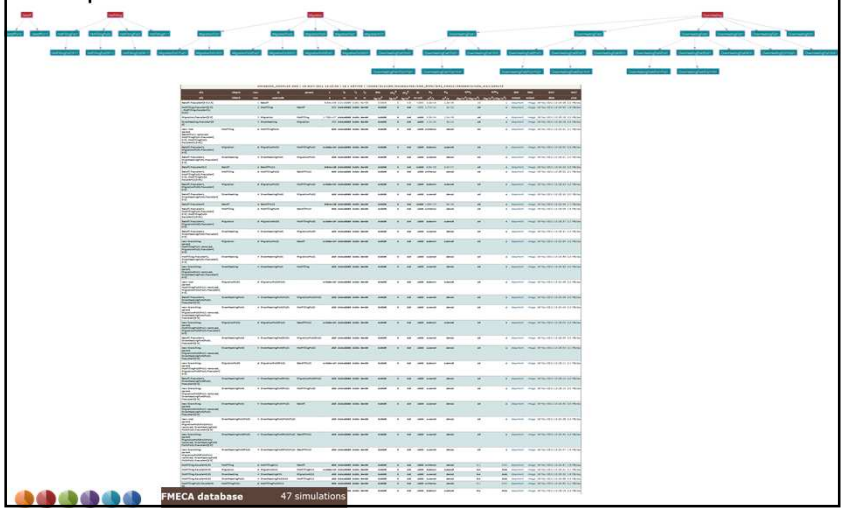
COMPLEX ANALYSES WITH OMITTED STEPS

A path towards FMECA-based risk assessment

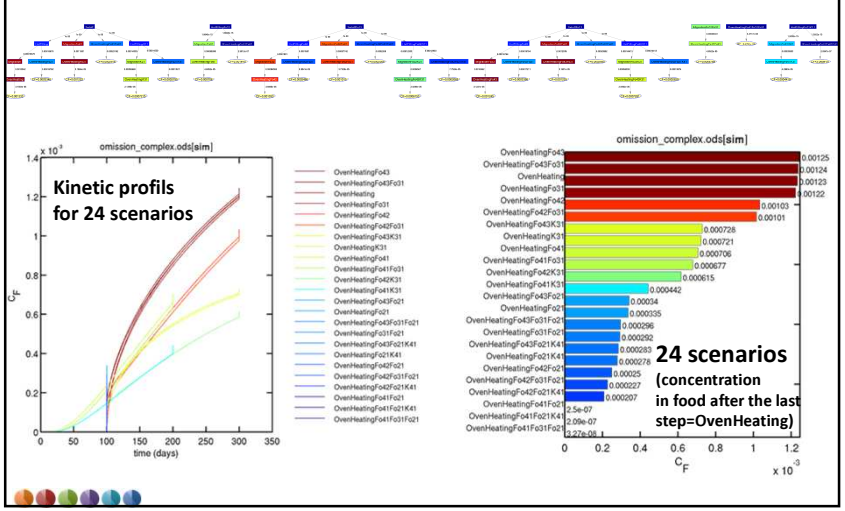
Small/large diffusants | Good/poor chemical affinity



COMPLEX ANALYSES WITH OMITTED STEPS
A path towards FMECA



COMPLEX ANALYSES WITH OMITTED STEPS: Results

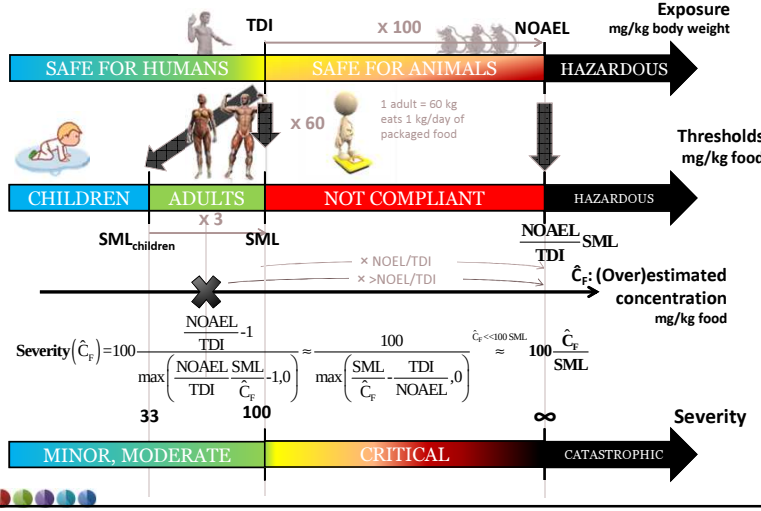


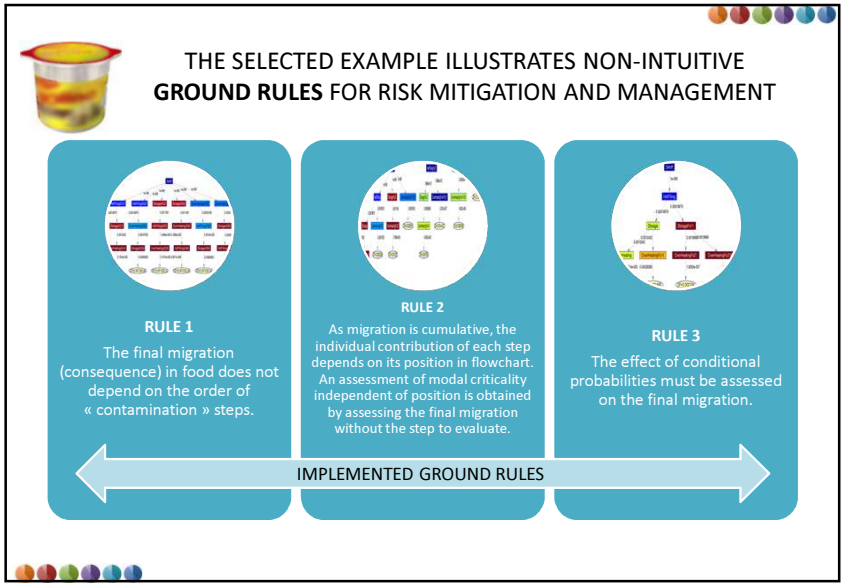
SEVERITY AND CRITICALITY ANALYSIS

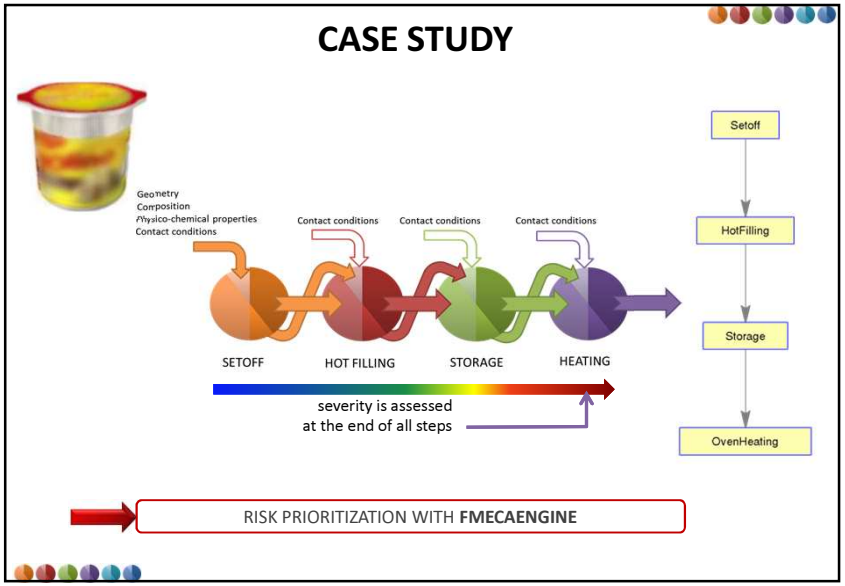
Definition of severity

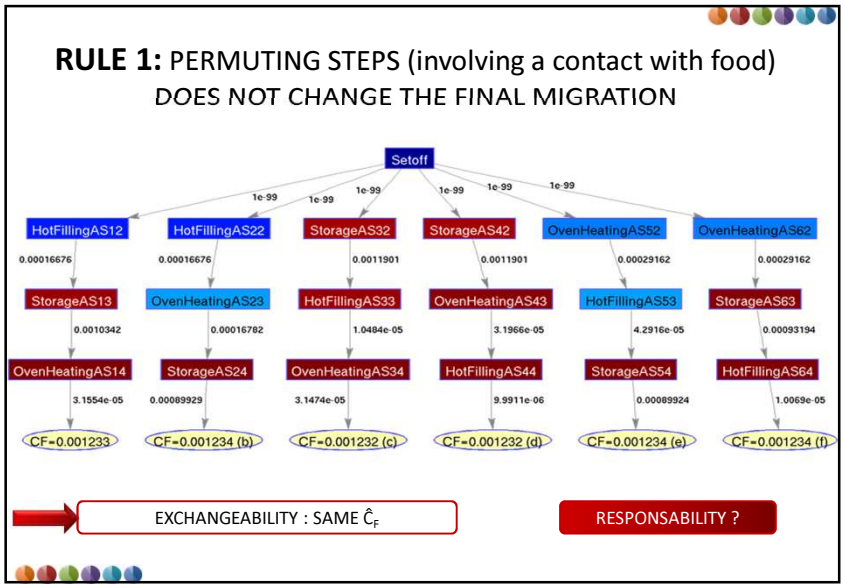
3 rules for risk migration and management → illustrative example

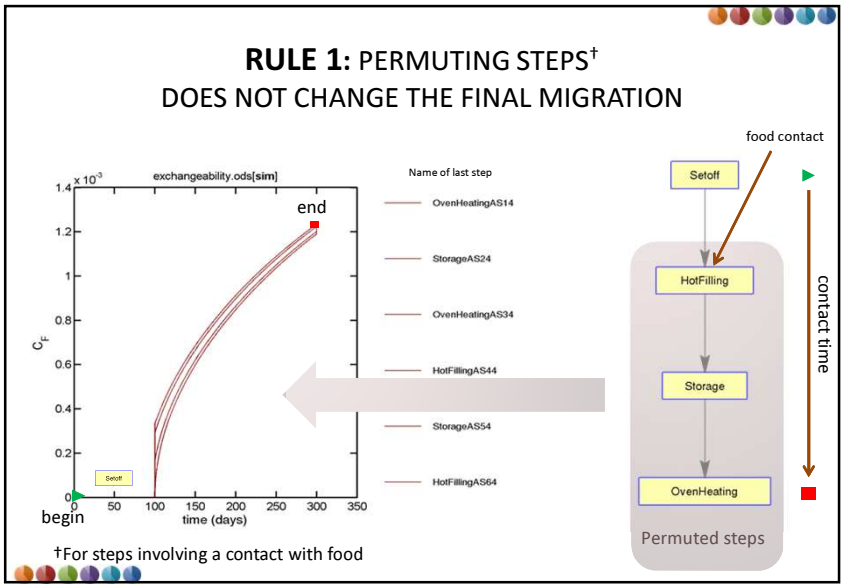
DEFINITION OF SEVERITY



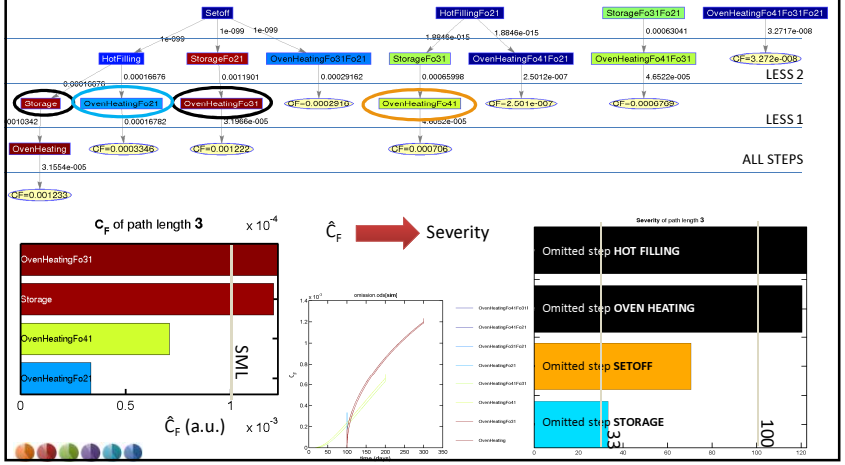

THE SELECTED EXAMPLE ILLUSTRATES NON-INTUITIVE
GROUND RULES FOR RISK MITIGATION AND MANAGEMENT
RULE 1
The final migration (consequence) in food does not depend on the order of « contamination » steps.
RULE 2
As migration is cumulative, the individual contribution of each step depends on its position in flowchart. An assessment of modal criticality independent of position is obtained by assessing the final migration without the step to evaluate.
RULE 3
The effect of conditional probabilities must be assessed on the final migration.
IMPLEMENTED GROUND RULES


CASE STUDY
Geometry
Composition
Physico-chemical properties
Contact conditions
Contact conditions
Contact conditions
Contact conditions
SETOFF
HOT FILLING
STORAGE
HEATING
severity is assessed
at the end of all steps
Setoff
HotFilling
Storage
OvenHeating
RISK PRIORITIZATION WITH FMECAENGINE

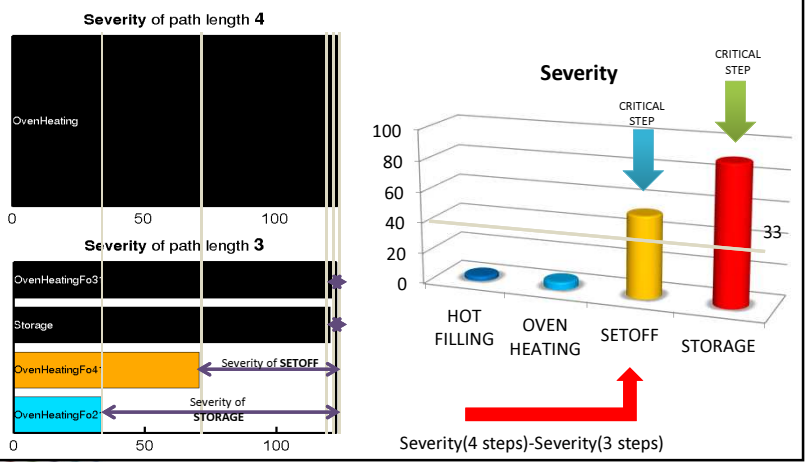

RULE 1: PERMUTING STEPS (involving a contact with food)
DOES NOT CHANGE THE FINAL MIGRATION
Setoff
HotFillingAS12
HotFillingAS22
StorageAS32
StorageAS42
OvenHeatingAS52
OvenHeatingAS62
StorageAS13
OvenHeatingAS23
HotFillingAS33
OvenHeatingAS43
HotFillingAS53
StorageAS63
OvenHeatingAS14
StorageAS24
OvenHeatingAS34
HotFillingAS44
StorageAS54
HotFillingAS64
CF=0.001233
CF=0.001234 (b)
CF=0.001232 (c)
CF=0.001232 (d)
CF=0.001234 (e)
CF=0.001234 (f)
EXCHANGEABILITY : SAME $\hat{C}_F$
RESPONSABILITY ?


RULE 1: PERMUTING STEPS†
DOES NOT CHANGE THE FINAL MIGRATION
food contact
exchangeability.ods[sim]
end
begin
time (days)
Name of last step
OvenHeatingAS14
StorageAS24
OvenHeatingAS34
HotFillingAS44
StorageAS54
HotFillingAS64
Setoff
HotFilling
Storage
OvenHeating
contact time
Permuted steps
†For steps involving a contact with food

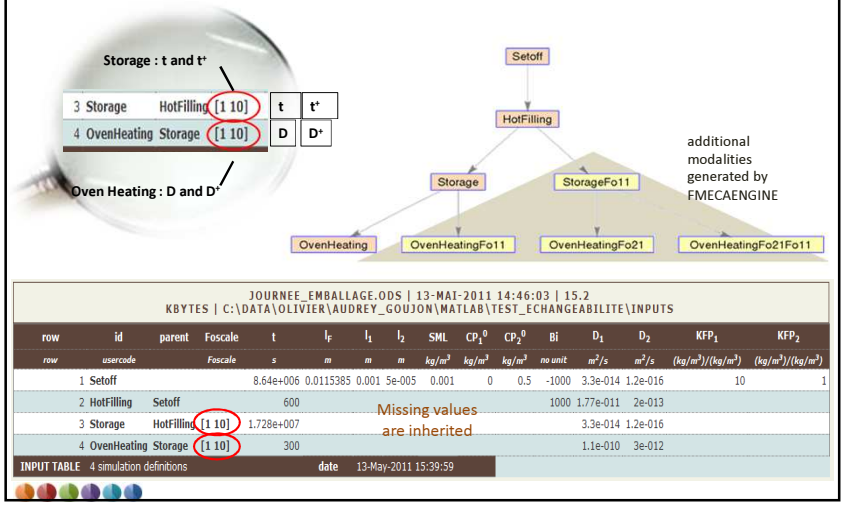
RULE 2: THE MODAL CRITICALITY ASSOCIATED TO ONE OR A COMBINED STEP(S) IS OBTAINED BY OMITTING THE STEP TO ASSESS



RULE 2: CALCULATION PROCEDURE TO EXTRACT THE SEVERITY OF A SINGLE STEP

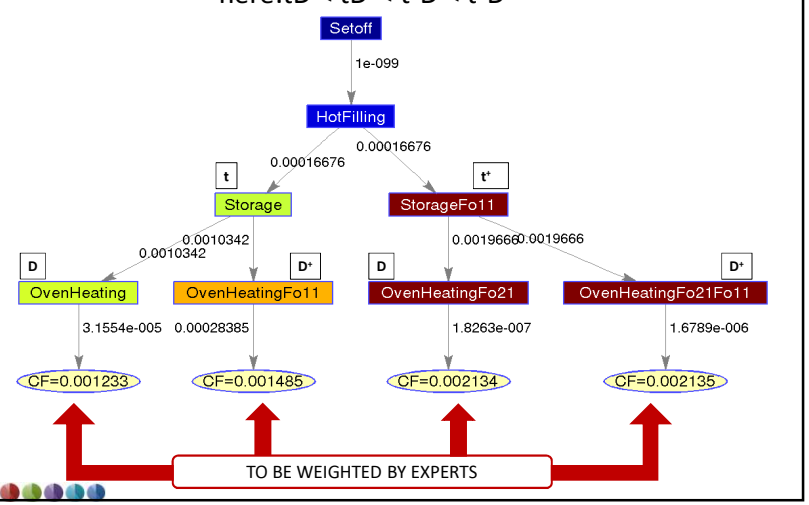


RULE 3: RATING COMBINED MODALITIES REQUIRES TO ASSESS THE CUMULATIVE EFFECT OF ALL COMBINATIONS (e.g. excessive storage and heating)



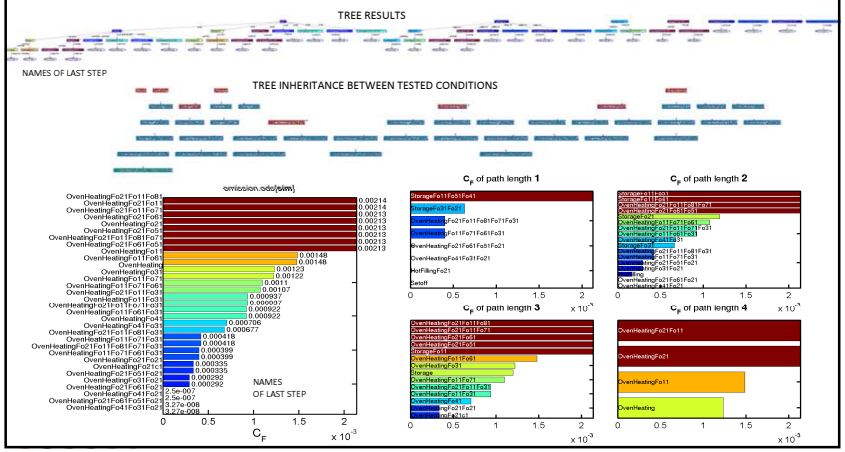
row	id	parent	Foscale	t	l_F	l_1	l_2	SML	CP_1^0	CP_2^0	Bi	D_1	D_2	KFP_1	KFP_2
row	usercode		Foscale	s	m	m	m	kg/m³	kg/m³	kg/m³	no unit	m²/s	m²/s	(kg/m³)/(kg/m³)	(kg/m³)/(kg/m³)
1	Setoff			8.64e+006	0.0115385	0.001	5e-005	0.001	0	0.5	-1000	3.3e-014	1.2e-016	10	1
2	HotFilling	Setoff		600							1000	1.77e-011	2e-013		
3	Storage	HotFilling	[1 10]	1.728e+007								3.3e-014	1.2e-016		
4	OvenHeating	Storage	[1 10]	300								1.1e-010	3e-012		

RULE 3: RATING COMBINED EFFECTS here:tD < tD⁺< t⁺D < t⁺D⁺



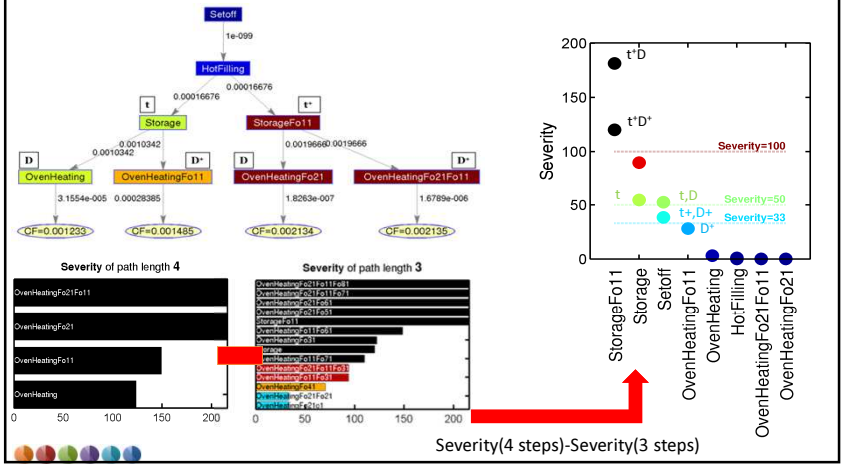
RULE 3: MODAL CRITICALITY ASSOCIATED TO [t,D] × [t⁺,D⁺]

➔ all sequences with omitted steps (1..3) are tested (**43 simulations !**).



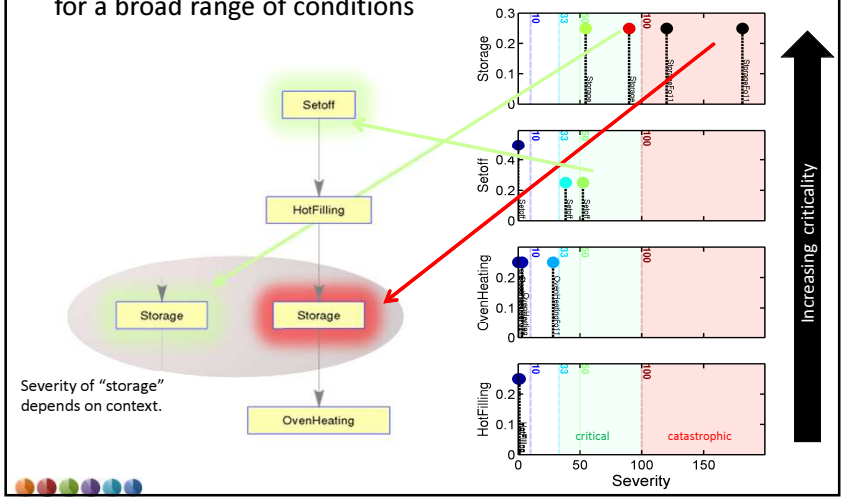
RULE 3: MODAL CRITICALITY ASSOCIATED TO [t,D] × [t⁺,D⁺]

➔ all sequences with omitted steps (1..3) are tested (43 simulations).

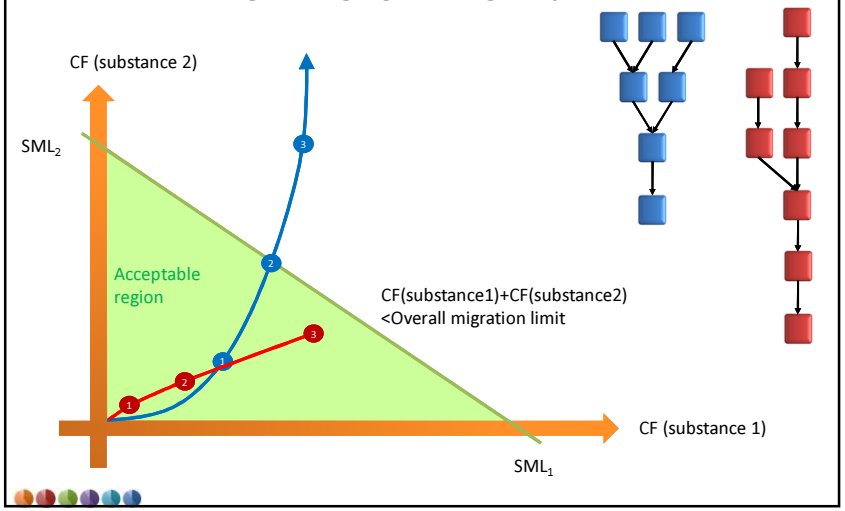


RATING SEVERITY

for a broad range of conditions

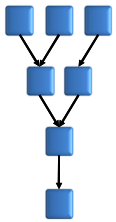


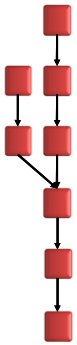
HOW DOES IT WORK?



APPLICATION: CONCURRENT DESIGN/SCREENING

- DESIGN A
- DESIGN B



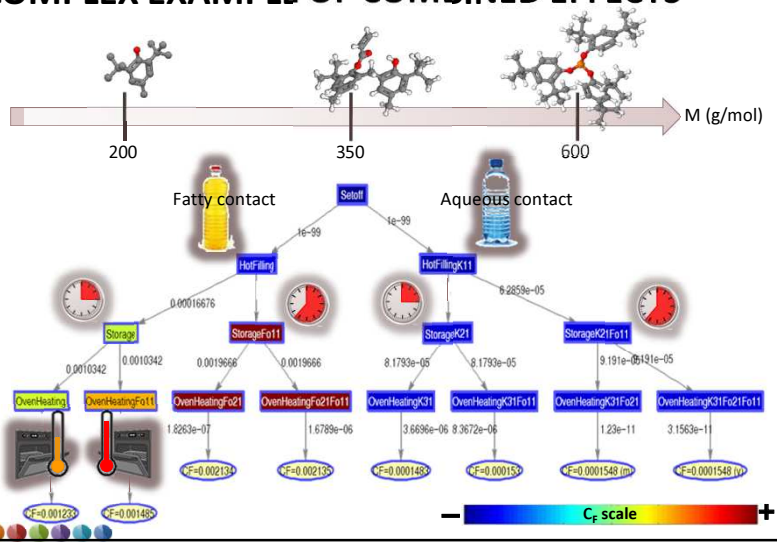


Identification of
- critical components
- critical steps
- uncorrectable design areas

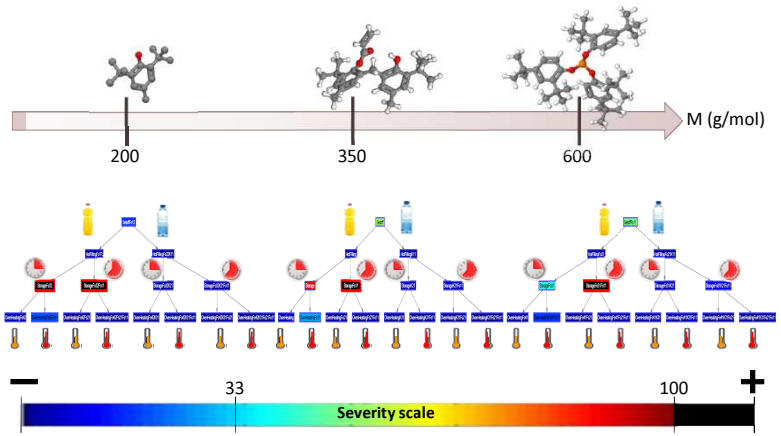
Feed results into
- Design
- Quality management procedures
- Suppliers audit

Considering related decisions

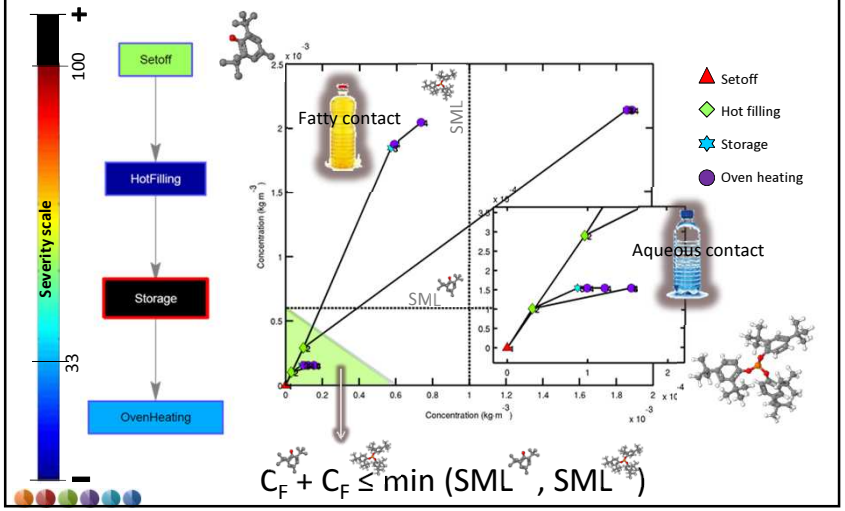
COMPLEX EXAMPLE OF COMBINED EFFECTS



RATING SEVERITY for several migrants



RISK BALANCING between migrants



$C_F + C_F \leq \min(SML^{\cdot}, SML^{\cdot})$

CONNECTING WITH PHYSICO-CHEMICAL DATABASES

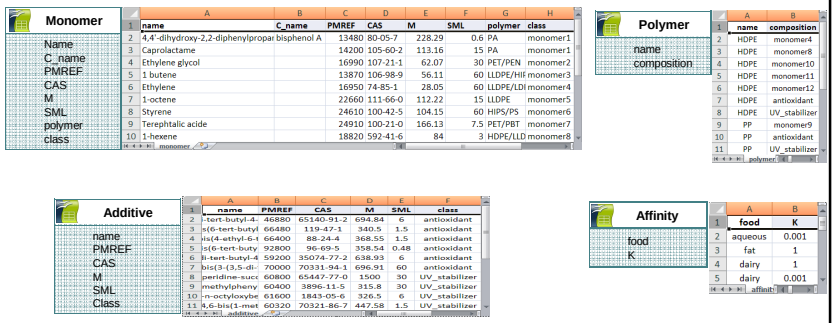
Relational databases

Key examples (coding with fmecaengine V 0.41 and later)

RELATIONAL DATABASES

Database = collection of tables including primary, super key and foreign keys

Examples



IMPLEMENTED RELATIONSHIPS

Implemented relationships include:

- one-to-one relationships
- one-to-many relationships
- many-to-many relationships
- cascading between keys

Operations above are coded using a special syntax (more flexible and concise than SQL) that mimics UML (http://en.wikipedia.org/wiki/Unified_Modeling_Language) :

PP:polymer::name->classadditives
PP:polymer::name->classadditives:substance::class->name
PP:polymer::name->classadditives:substance::class->M

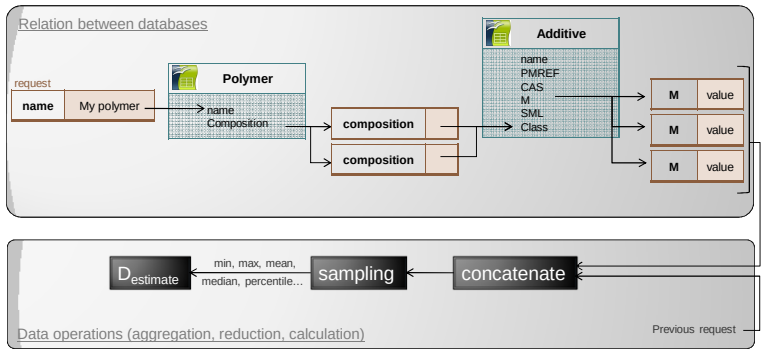
By default, a UNIQUE operator operator (data reduction) is apply to all relationships involving strings or cell array of strings. The request order is respected only for single requests.

It is worth to notice that final results are also included within a large database that can be resused in future simulations or in subsequent analysis (e.g. for calculating severity).

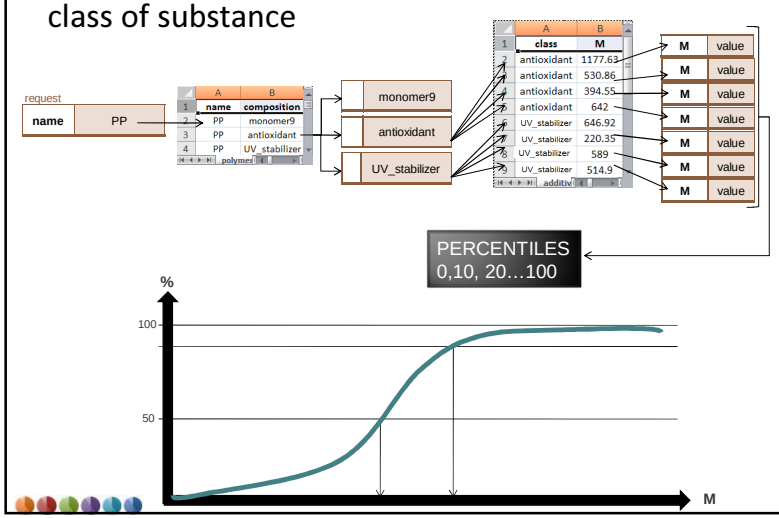
Background:
http://en.wikipedia.org/wiki/Many-to-many_(data_model)
http://en.wikipedia.org/wiki/Junction_table
http://en.wikipedia.org/wiki/Foreign_key
http://en.wikipedia.org/wiki/Superkey
http://en.wikipedia.org/wiki/Associative_Entities

EXAMPLE OF **ONE TO MANY** RELATIONSHIPS

The example depicts how databases could be used to derive the formulation (substance content) of a common polymer and finally to sample the distribution of molecular mass (M) of possible migrants. These values are finally used for the calculation of some estimates of diffusion coefficients (D).



TIP: the name of a substance can be replaced by a class of substance



RELATIONSHIPS are called KEY or KEY-to-KEY

KEY2KEY syntax

Basic coding

- keyA:tableA::columnA1->columnA2
- keyA:tableA::columnA1->columnA2:tableB::columnB1->columnB2
- keyA:tableA::columnA1->columnA2:tableB::columnB1->columnB2:tableC::columnC1->columnC2

Extended coding with mathematical operators

- max(Dpiringer(PP,PP:polymer::name->classadditives:substance::class->M,40))
- Dpiringer(PP,min(PP:polymer::name->classadditives:substance::class->M),40)

Extended coding with regular expressions

- \antiUV{3|2}\ :substance::name->M
- \.#PE.#\:polymer::name->classadditives:substance::class->name:substance::name->M

Extended coding with lists

- LDPE|undefined|PP:polymer::name->classadditives:substance::class->name:substance::name->M

EXAMPLES

Collection of D

Dpiringer(PP,PP:polymer::name→composition:additive::class→M,25) ➔ D_{min}, D_{med}, D_{max}
Dpiringer(PP,min(PP:polymer::name→composition:additive::class→M,25))
ou
max(Dpiringer(PP,PP:polymer::name→composition:additive::class→M,25)) } D_{max}

Collection of K

dairy:affinity::food→K ➔ K_{min}, K_{med}, K_{max}

Collection of SML

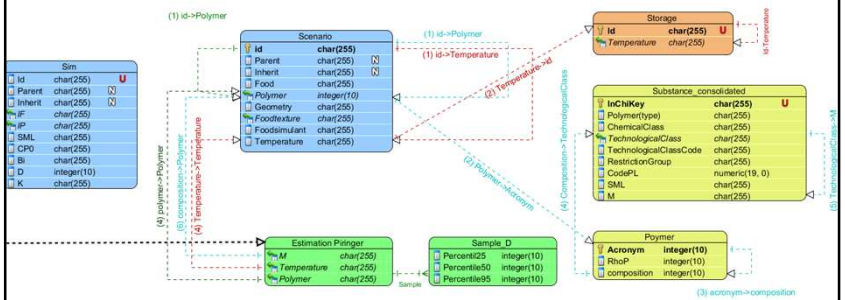
mean(PP:polymer::name→composition:additive::class→SML) ➔ SML_{mean}

➔Any valid Matlab expressions can be accepted

EXAMPLE with multiple keys

note that all keys must be enclosed within ()

Dpiringer((LDPEYogurtBottle:scenario::id->polymer),
(LDPEYogurtBottle:scenario::id->polymer:polymer::acronym->composition:substance::technologicalclass->M),
(LDPEYogurtBottle:scenario::id->temperature:storage::id->temperature))



HOW TO BUILD USER DATABASES, RULEBASES, KNOWLEDGEBASES

Database, rulebase, knowledgebase = collection of tables including primary, super key and foreign keys

Examples: Tables of physico-chemical properties

Additive	
InChIKey:*string (U)*	NLZUEZXRPGMBCV-UHFFFAOYSA-N
Name:*string*	2,6-Di-t-butyl-4-hydroxytoluene;2,6-di-tert-butyl-4-methyl phenol;4-Methyl-2,6-bis(2-methyl-2-propanyl)phenol;4-Methyl-2,6-di-t-butyl-phenol ;4-Methyl-2,6-ditertbutylphenol;BHT;Butylated hydroxytoluene;phenol, 2,6-bis(1,1-dimethylethyl)-4-methyl;2,6-Di-t-butyl-p-cresol
CAS:*list of strings*	128-37-0;1948-33-0;25377-21-3;50-78-2;62698-56-0;720-39-8;950-56-1;97123-41-6
PMREF:*integer*	46640
M:*real*	220.35
SML:*real*	3
TDI:*real*	**0.05**
logP: *real*	5.2
logPmin: *real*	5.32
logPmax: *real*	5.03
Class:***string***	Antioxidant

Monomer	
InChIKey:*string (U)*	QQONPFPTGQHPMA-UHFFFAOYSA-N
Name:*string*	name: propene;1-propene;propylene;methylethene;1-propylene;prop-1-ene
CAS:*list of strings*	115-07-1;33004-01-2;676-63-1
PMREF:*integer*	23980
M:*real*	42.08
SML:*real*	60
TDI:*real*	NaN
logP:*list of reals*	1.7
Class:*string*	propylene
Polymer:*string*	PP

Polymer	
Acronym:*string*	PP
Name:*string*	polypropylene
Class:*string*	PO
Density:*real*	0.95;0.92
Composition: *list of strings*	antioxidant, UV_stabilizer, monomer

ALL DATA TYPES ARE RECOGNIZED IMPLICITELY ACCORDING TO THE FOLLOWING RULES	
Empty cell	= Not A Number
Scalar	Any numeric value
List	String containing ";" or "|"
Unique (U)	is enforced for all keys returning an array of strings
Primary/Foreign key	Any key with string/char content (possibly including some lists)

EXAMPLES: RULE BASES AND KNOWLEDGE BASES

scenario	
Primary key	**Id:** gobeletPP
Foreign key	**Food :** hotdrink
List/Foreign key	**Foodsimulant:** fatty, ethanol50
Foreign key	**Material :** PP
Foreign key	**Geometry :** gobelet
Foreign key	**Step** : consumption
Foreign key	**Temperature :** hot

Geometry	
Primary key	**Id:** gobelet
List/ Foreign key	**A:** 170
List/ Foreign key	**Aunit:** cm2
List/ Foreign key	**mF:** 200
List/ Foreign key	**mFunit:** g
List/ Foreign key	**IP :** 150
	IPunit : µm

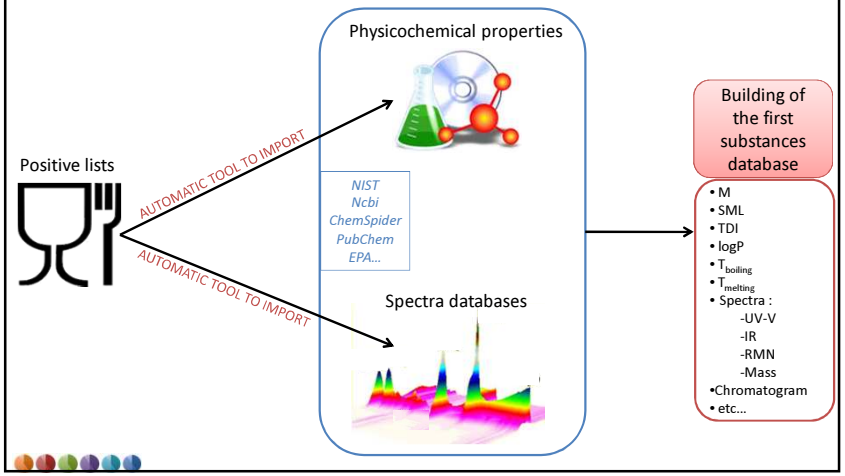
Food	
Primary key	**Id:** hotdrink
List/ Foreign key	**Shelflife**: 2
List/ Foreign key	**Shelflifeunit :** h
List/ Foreign key	**rhoF**: 1000

Foodsimulant	
Primary key	**Id:** ethanol50
List	**K:** 0.1

Consumption	
Primary key	**Id** : hot
List/ Foreign key	**Temperature:** 70
List/ Foreign key	**Time:** 2
List/ Foreign key	**Unit:** h

Diffusion	
Primary key	**InChIKey**: NLZUEZXRPGMBCV-UHFFFAOYSA-N
	M: 220.35
	polymer: PP
	rhoP: 0.95
	temperature: 40
List	**D**: 4.6E-15;2.7E-16;5E-15

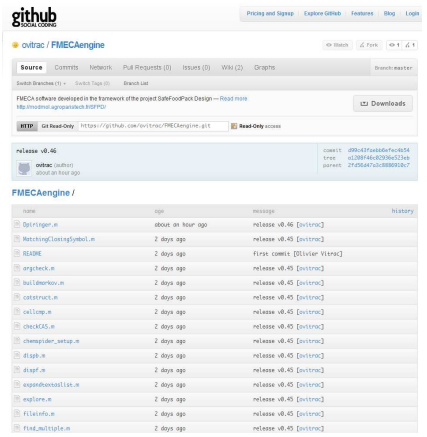
POSSIBLE EXTENSIONS



All functions can be downloaded at:

https://github.com/ovitrac/FMECAengine

DETAILED MANUAL



fmecaengine

fmecaengine: advanced script/function (object-oriented) to launch complex FMECA calculations from an OpenOffice Spreadsheet

```
   USAGE AS A SCRIPT (depreciated)
   Once the section "path and file definitions" properly defined, you need only to press F5 (i.e. run)
   To override default values, define user variables in the workspace

   USAGE AS A FUNCTION (recommended)
     Syntax: fmecaenfine('variable1','value1','variable2','value2','variable3','value3',...)
             fmecaenfine(definition,...) whith definition a structure with fields definition.variable1 = 'value1'

   LIST OF USER VARIABLES
               local  root directory of all files and folders (default = pwd)
           inputpath  folder that contains all inputs (relative to local, default = 'inputs')
          outputpath  folder that contains all outputs (relative to local, default = 'outputs')
       fmecamainfile  filename of ODSfile (located in inputpath, default = 'fmeca_demo.ods')
                      a structure (matching the output data0) can be used instead to link several calls of fmecaengine (use inputpath
                      setup the pattern for output filenames, it must include an )
          fmecadbfile  fmecadbfile of the meta database (default = 'fmecabaseofconditions.mat')
      fmecasheetname  name of spreadheet that (default = 'sim')
                      If you use several sheets, do not use same id between sheets as it will be break the consistency of simulation
             database  a database to feed all numeric input values (with statistics if needed) including Foscale, Kscale
                      a structure such database.table.column than can be indexed as valueA:table::columnA->columnB
                      a valid ODS file that code for a similar structure (table=sheetname, column names=first row)
                      By default, fmecaengine assumes that fmecamainfile is a such database
             severity vectorized anonymous function to define severity (default = @(CF,SML) 100*99./max(100*SML./CF-1,0))
                      Alternative, define first NOELfact = 100;
                      @(CF,SML) NOELfact*(NOELfact-1)./max(NOELfact*SML./CF-1,0)
               sample anonymous function to setup how a one->many or a many->many relationship is sampled
                      default = @(x) prctile(x,[5 50 95]) (only 5th, 50th and 95th percentiles are consideredà
                      versions older than 0.43 used @(x) [min(x) median(x) max(x)]
                      Note that Foscale and Kscale derived from one->many or a many->many relationships are normalized from the medi
                      the sample: setdiff(sample,median(sample))/median(sample)
   ADDITIONAL PROPERTIES FOR ADVANCED USERS (change solver performances used in senspatankar and variants, enable linked instances o
             nograph flag value (default=true) to prevent graphs from being plotted
             options ODESET structure
                      default value = odeset('RelTol',1e-4,'AbsTol',1e-4,'Initialstep',1e-8,'Maxstep',1e3,'Maxorder',2); % note based
               nmesh number of finite volumes for a layer with a dimensionless thickness of 1 (accurate for most of purposes)
                      default value = 200
       fmecamainfile  matching the output structure data0 can be used instead of a filename to link several calls of fmecaengine
           inputpath (when fmecamainfile is a structure) setups the pattern for output filenames (it must include the extension .ods

   OTHER PROPERTIES (to change the behavior of fmecaengine, mainly for developers, see code for additional details)
             headers number of header lines (default=2) in fmecamainfile:fmecasheetname
     databaseheaders number of header lines (default=1) in database file
           regular_l, regular_D, regular_K, regular_C:
                 regular expressions to identify column titles matching l,D,K,C
           print_id, print_parent, print_inherit, print_l, print_D, print_K, print_C, print_t, print_L, print_Bi:
                 expressions (matching fprintf syntax) to name layers id,parent,inherit,l,D,K,C,t,L,Bi

   OUTPUTS
   fmecadb = fmecaenfine(...) returns the global database listing all conditions
   [fmecadb,data0,data,options] = fmecaenfine(...) returns the input parameters as a structure
       data0: initial content of fmecamainfile as a structure
        data: state of data0 after execution of fmecaengine
     options: options used in the current instance of fmecaengine

   OUTPUT FILES (located in outputpath)
       raw MAT files for post-processing
       HTML report with linked PNG and PDF files (individual simulations, dependence/inheritance/result graphs, Pareto charts)

 DETAILED DESCRIPTION
 > The current script reads the spreadsheet 'sim' from an ODS file (OpenOffice Spreadsheet) to setup complex migration simulations.
 > Possible simulations (possibly combined) include: multilayer, chained scenarios, setoff effect, hot-filling, sterilization...
 > Titles in the spreadsheet are used to recognize variables/inputs.
 > Please, check closely the structure of the demonstration ODS file (fmeca_demo.ods) before using this script.
 > Each row in the spreadsheet codes for a single simulation. The spreadsheet can contains an arbitrary number of rows.

 > A column "id" is used to identify each simulation inputs and outputs.
 > Complex chaining of simulations is enabled by using the column "parent" that gives the id of the previous result
   to be used as input for the next simulation.
   The script does not post-process the simulations. It stores all intermediate simulations results as a MAT file (id.mat).
   For routine control, a corresponding id.png and id.pdf, including the concentration profile and concentration cinetic
   with twice the requested time, are also saved together.
   A metadatabase 'fmecadb' (data about simulations), whose filename is defined by the variable fmecadbfile, stores all
   simulation conditions including solver parameters.

 > The script manages the prioritization and history of all simulations regardless their order in the spreadsheet and
   their chaining complexity (only limitation: a single parent).

 > For dependent simulations, default input inheritence is also accepted so that only modified parameters need to be specified
   and all other variables/inputs can be left empty.
 > The script detects any change in unputs and restart all dependent simulations in cascade.
 > Removing any id.MAT file or the entire output directory forces corresponding simulations to be restarted.
 > Removing an entry id in 'fmecadb' (with the command rmfield) is the same as removing the corresponding id.MAT
   (the corresponding MAT file will be rewritten).

 > Inheritance of the sole inputs (while discarding simulations results) is possible via an optional column 'inherit'.
   All empty or NaN cells are replaced by their equivalent contents from the row with id corresponding to 'inherit'.
   Complex propagation are allowed for inheritance (e.g. cascading, tree) whatever the dependence enforced by parent.
   In case of conflicts between inputs inheritance between 'parent' and 'inherit', the later have a higher precedence.
   As a rule of thumb, use 'parent' to chain simulations and prefer 'inherit' for sensitivity analysis
   (e.g. to calculate Jacobians).
```

```
> Combinatory simulations (Full or pseudo Monte-Carlo [preferred]) are setup via special columns that contain vectors of scaling va
  A valid Matlab syntax embedded with a string can be used to define vectors (1:3, [.1 10], logspace(-3,3,10) ...)
  Note that the scaling value 1 is removed from the list as the original row with defined values is considered to be
  the reference simulation with a unitary scaling (i.e. no scaling).
  Tree bifurcations/ramifications induced by multiple values are automatically generated. New id are based on original
  id by combining the name of scaling variable (e.g. Fo or K) the following tree level (1..m) and the scaling value (1..n).
  Row vectors (e.g. 1:3) do not propagate scaling values to children (to simulate different contact times at a step).
  Column vectors (e.g. (1:3)') propagate scaling values to children (to simulate different classes of molecules).
  Currently, the following scaling parameters are recognized:
      Foscale scale to simulate indifferently variable diffusion time and diffusion coefficients
      Kscale to simulate different chemical affinity.

> To assess the effect of a single setup, use 0 as scaling parameter (non-propagable value). The corresponding step
  will be removed. When required, new roots are added to the simulation tree.

  fmecaengine can be linked with internal and external databases with keyword 'database' and relational syntaxes (see KEY2KEY).
  EXAMPLES for fmeca_demo3.ods:
   100*(test_key:sim::id->CP20) to reuse the concentration value of the simulation with the id test_key
   1000*min(PP:polymer::name->classadditives:substance::class->SML) replacing a numerical SML value
   min(Dpiringer(PP,PP:polymer::name->classadditives:substance::class->M,40)) replacing a numerical D value
   Dpiringer(PP,max(PP:polymer::name->classadditives:substance::class->M),40) replacing a numerical D value
   Dpiringer(PP,PP:polymer::name->classadditives:substance::class->M,40) replacing a numerical D value and setting up Foscale value
 ADVANCED SYNTAXES (see key2key examples for further details)
  e.g. request based on alternatives
      HDPE|PP:polymer::name->classadditives:substance::class->M
  e.g. request based on superclass defined in "polymerfamily": polyolefin = 'LLDPE;LDPE;MDPE;HDPE;PP'
  polyolefin:polymerfamily::name->polymer:polymer::name->classadditives:substance::class->name:substance::name->M
  e.g. request based on regular expressions (simple regular expressions are accepted)
  \antiUV{3|2}\ :substance::name->M

> fmecaengine is designed to run in paralell (trough several instances) on the computing cluster of JRU 1145
  on either Windows or Linux (preferred for efficiency) nodes.
> Please write, an additional script or function for complex post-treatment. It was not the intend of this script. Please,
  note that for accuracy while preserving memory a linear time scale is used for kinetics and time scale propertional to the
  square root of time is used for concentration profiles.

ADDITIONAL COMMENTS
  Note 1: To add more layers, update the spreadsheet consistently (no need to modify this script/function)
  Note 2: It is expected that simulation chaining is applied to consistent structures (i.e. with a same number of layers)
  Note 3: Any negative Bi value is interpreted as "setoff" simulation (to be placed as a starting node)
  Note 4: Any change in solver paremeters is also detected and forces the whole database to be recaculated
          (use different output folders to store the results from different strategies if required)
  Note 5: For debugging purposes, the structure array s stores all simulation parameters in same order as in the original
          spreadsheet. Use r = senspatankar(senspatankar_wrapper(s(i))) to restart the ith simulation
  Note 6: Creation of PDF files on the fly may interact with antivirus software on windows systems. If you get randomly
          messages such as "print at XXX %temp%\tempfile.ps: Cannot open file: permission denied.", restart the script/function
          and it should solve the issue.

SEE ALSO: FMECASINGLE FMECAROOT BUILDMARKOV KEY2KEY LOADFMECAENGINEDB

DEPENDENCY INFORMATION TO IMPLEMENT THIS SCRIPT/FUNCTION IN OTHER PROJECTS
> Dependencies to other functions (written by INRA\Olivier Vitrac)


DEMO (Fmecaengine v0.25)
        fmecaengine(... without auto expansion
            'local','C:\Data\Olivier\INRA\Etudiants & visiteurs\Audrey Goujon\illustration_fmeca\',...
            'fmecamainfile','illustration_fmecaengine.ods',...
            'inputpath','inputs', ...
            'outputpath','tmp' ...
            );
        fmecaengine(... automatic expansion
            'local','C:\Data\Olivier\INRA\Etudiants & visiteurs\Audrey Goujon\illustration_fmeca\',...
            'fmecamainfile','illustration2_fmecaengine.ods',...
            'inputpath','inputs', ...
            'outputpath','tmp2' ...
            );

DEMO (Fmecaengine v0.27)
        fmecaengine(...
            'local','C:\Data\Olivier\INRA\Etudiants & visiteurs\Audrey Goujon\illustration_fmeca\',...
            'fmecamainfile','illustration3_fmecaengine.ods',...
            'inputpath','inputs', ...
            'outputpath','tmp3' ...
            );

        fmecaengine(...
            'local','C:\Data\Olivier\INRA\Etudiants & visiteurs\Audrey Goujon\illustration_fmeca\',...
            'fmecamainfile','illustration4_fmecaengine.ods',...
            'inputpath','inputs', ...
            'outputpath','tmp4' ...
            );

CONTACT
Any question to this script/function must be addressed to: olivier.vitrac@agroparistech.fr
The script/function was designed to run on the cluster of JRU 1145 Food Process Engineering (admin: Olivier Vitrac)

Migration 2.1 (Fmecaengine v0.45) - 10/04/2011 - INRA\Olivier Vitrac - Audrey Goujon - rev. 26/08/2011
```

See also

argcheck(*), buildmarkov(*), cellcmp(*), dispb, dispf, Dpiringer(*), fileinfo, find_multiple(*), formatfig, formatax, key2key(*), localname, loadods(*), LOADFMECENGINEDB (*), MatchingClosingSymbol, matcmp(*), nearestpoint(*), print_pdf, print_png, rgb, senspatankar(*), senspatankarC(*), setoffpatankar(*), structcmp(*), subplots, wraptext (*) indispensable functions (shared between toolboxes migration and MS) > Commercial tooboxes required: Bioinformatics Toolbox (only for Graph plotting)

key2key

```
key2key returns the values related to keyA:tableA::columnA->tableB::columnB:tableC::columnC->tableD::columnD...
      SCOPE key2key is a multivalued function (in a Mathematical sense) that implements cardinality operations also known as relatio
           between tables via primary key, super key and foreign keys. Implemented relationships include:
           - one-to-one relationships
           - one-to-many relationships
           - many-to-many relationships
           - cascading between keys
          Operations above are applied via an intuitive syntax, different of SQL for concision.
          By default, a UNIQUE operator operator is apply to all relationships involving strings or cell array of strings.
          The request order is respected only for single requests.

   Examples of relationships in connection with FMECAENGINE
       PP:polymer::name->classadditives
       PP:polymer::name->classadditives:substance::class->name
       PP:polymer::name->classadditives:substance::class->M

       Background:
           http://en.wikipedia.org/wiki/Many-to-many_(data_model)
           http://en.wikipedia.org/wiki/Junction_table
           http://en.wikipedia.org/wiki/Foreign_key
           http://en.wikipedia.org/wiki/Superkey
           http://en.wikipedia.org/wiki/Associative_Entities

   Basic syntax: val = key2key(db,key)

      INPUTS
             db: structure such as db.tableX.columnY = numerical value or string
                 alternatively a structure coding for an ODS file such as: db=struct('filename',...,'headers',...)
            key: string or a lxn/nx1 string cell array coding for
                   keyA:tableA::columnA1->columnA2
                   keyA:tableA::columnA1->columnA2:tableB::columnB1->columnB2
                   keyA:tableA::columnA1->columnA2:tableB::columnB1->columnB2:tableC::columnC1->columnC2
           key can contain be combined with numbers or can contain Matlab expressions (see examples for details)
       NOTE 1: about lists embedded in strings
           db.tableX.columnY and keys can include list of strings.
           Values are separated by symbols: ';' (semi-colon) or  '|' (vertical bar)
           As a rule of thumb, '|' must be preferred in keys as they mimic the behavior or operator OR.
       NOTE 2: about extra spaces
           Spaces are tolerated within key values but note that they are trimmed around ';' and '|'.
           Do not use spaces without checking the reliability of the database content.
       NOTE 3: about regular expressions
           Regular expressions are partly implemented (see the rules below).
           A regular expression must start with symbol '\' and end with '\'
           Symbol '*' must be replaced by '#' (* is reserved for multiplications).
           Symbols '(' and ')' must be replaced by '{' and '}' respectively
           Quantifiers {} must be replaced by {{}}.
           Quantifier ? is available
           Operators | . \d \D \s \W are implemented
           Operators [] ^ $ \< \> (?< (?= (?<! and quantifiers + are not implemented !!!
       NOTE 4: about multiple keys
           When multiple keys are combined in expressions, they must be enclosed within ()
       NOTE 5: about mathematical expressions
          Almost any valid Matlab expressions can be combined with keys.
          Diringer() has been implemented within a syntax simpler than with Matlab conventional one
          Dpiringer(polymercode,key:table1::columnA1->columnB1:table2::columnA2->columnB2 ...->M)
          Dpiringer(polymercode,key:table1::columnA1->columnB1:table2::columnA2->columnB2 ...->M,temperature)

      OUTPUTS
             val: array of values as stored in db when key is a string
                  a nx3 array as [min(values median(values) maximum(values)]

   Advanced syntax: [val,dbout] = key2key(...)
                    dbout: database (loaded if needed)

   Internal syntax: val = key2key(db,key,interpreterflag,recursionflag)
                   interpreterflag (default = true) executes keys as possible Matlab expressions
                   recursionflag (default=false) forces key2key to drop the recursion

   Syntax with specific statistical analysis (works only with cell array of keys)
                    val = key2key(db,keyarray,[],[],percentiles)
                    default percentiles = 5th, 50th, 100th

  TOY EXAMPLES, execute lines between {} with F9 (evaluate)
  %{
   %  EXAMPLE 1 (full toy example)
        key = 'PP:polymer::name->classadditives:substance::class->name';
       db.polymer = ...
           struct('name', {{'LLDPE' 'LLDPE' 'LDPE' 'LDPE' 'MDPE' 'MDPE' 'HDPE' 'HDPE' 'PP' 'PP'}'},...
         'classadditives', {{'antoxidants' 'antiUV' 'antoxidants' 'antiUV' 'antoxidants' 'antiUV' 'antoxidants' 'antiUV' 'antoxidants
                  );
       db.substance = ...
           struct('class', {{'antoxidants' 'antoxidants' 'antoxidants' 'antiUV' 'antiUV' 'antiUV'}'},...
                  'name', {{'antiox1' 'antiox2' 'antiox3' 'antiUV1' 'antiUV2' 'antiUV3'}'},...
                  'M',     [  101       102      103       201       202       203 ]'...
                  );
        key2key(db,key)
  %}
  %{
  %  EXAMPLE 2: Variation based on example 1 (not very useful but illustrate the jump between multiple tables)
        key2 = 'PP:polymer::name->classadditives:substance::class->name:substance::name->M';
        key2key(db,key2)
  %}
  %{
```

```
%  EXAMPLE 3: Variation based on example 2 (use of superclass as key)
    key3 = 'polyolefin:polymerfamily::name->polymer:polymer::name->classadditives:substance::class->name:substance::name->M';
    db.polymerfamily = struct(...
            'name',{{'polyolefin' 'polystyrene'}'},...
            'polymer',{{'LLDPE;LDPE|MDPE;HDPE;PP' 'HIPS;PS'}'} ....
            );
    key2key(db,key3)
%}
%{
%  EXAMPLE 4: Variation based on example 3 (behavior in presence of missing values)
    key4 = 'LDPE|undefined|PP:polymer::name->classadditives:substance::class->name:substance::name->M';
    key2key(db,key4)
%}
%{
%  EXAMPLE 5: Variation based on example 4 (alternative values, note that extra spaces can be included arround operator '|')
    key5 = 'antiUV2 | antiox1 | antiUV3 :substance::name->M';
    key2key(db,key5)
%}
%{
%  EXAMPLES 6: Variation based on example 5 (search all anitUVi with i=0..9)
    key6a = '\antiUV\d\ :substance::name->M';
    key6b = '\antiUV\d{{1}}\ :substance::name->M';
    key6c = '\antiUV\d{{1,1}}\ :substance::name->M';
    key6d = '\antiUV\d?\ :substance::name->M';
    key2key(db,key6a)
    key2key(db,key6b)
    key2key(db,key6c)
    key2key(db,key6d)
%}
%{
%  EXAMPLES 7: Variation based on example 6 (to illustrate the difference beween regular expressions)
    key7a = '\antiUV{3|2}\ :substance::name->M';
    key7b = '\antiUV3|2\ :substance::name->M';
    key7c = '\.#PE.#\:polymer::name->classadditives:substance::class->name:substance::name->M'
    key7d = '\PE\:polymer::name->classadditives:substance::class->name:substance::name->M'
    key7e = 'PE:polymer::name->classadditives:substance::class->name:substance::name->M'
    key2key(db,key7a) % interpreted as containing antiUV3 or containing antiUV2
    key2key(db,key7b) % interpreted as containing antiUV3 or containing 2
    key2key(db,key7c)
    key2key(db,key7d)
    key2key(db,key7e)
%}
%{
%  EXAMPLES 8: with mathematical formula
    key8a = 'Dpiringer(PP,PP:polymer::name->classadditives:substance::class->M)'
    key8b = 'min(Dpiringer(PP,PP:polymer::name->classadditives:substance::class->M,40))';
    key8c = 'Dpiringer(PP,max(PP:polymer::name->classadditives:substance::class->M))'
    key8d = 'min(Dpiringer(PP,PP:polymer::name->classadditives:substance::class->M,100))';
    key8e = 'Dpiringer(LDPE,\antiUV{3|2}\:substance::name->M)';
    key8f = 'Dpiringer(LDPE,[(\antiUV{3|2}\:substance::name->M);(\antiUV1\:substance::name->M)])';
    key2key(db,key8a)
    key2key(db,key8b)
    key2key(db,key8c)
    key2key(db,key8d)
    key2key(db,key8e)
    key2key(db,key8f)
%}
%{
%  EXAMPLE 9: example 8a with double and triple keys
    db.contact = struct('condition',{{'cond1' 'cond2' 'cond3'}},'temperature',[25 40 110]);
    key9a = 'Dpiringer(PP,(PP:polymer::name->classadditives:substance::class->M),(cond3:contact::condition->temperature))';
    key2key(db,key9a)
    db.scenario = struct('id',{{'scenarioA' 'scenarioB' 'scenarioC'}},'polymer',{{'PP' 'LDPE' 'PS'}});
    key9b = 'Dpiringer((scenarioA:scenario::id->polymer),(PP:polymer::name->classadditives:substance::class->M),(cond3:contact::con
    key2key(db,key9b)
    key9c = 'Dpiringer((scenarioA:scenario::id->polymer),(scenarioA:scenario::id->polymer:polymer::name->classadditives:substance::
    key2key(db,key9c)
%}
```

See also

fmecaengine fmecasingle ismemberlist expandtextaslist loadfmecaenginedb

fmecasingle

```
 fmecasingle calculate the migration contribution of single steps from FMECA simulations including omitted steps (e.g. Foscale = [0 0])
    SYNTAX  res = fmecasingle(db [,ref,proba])
            [res,resunique] = fmecasingle(...)
    INPUTS
           db: database generated by FMECAENGINE with Foscale=[0 0] forced to all steps
               All considered simulations (possibly independent) must have originally a similar number of steps.
               Simulations can be derived from a previous simulation with modalities (several Foscale or Kscale values possibly combined)
          ref: names or indices of reference steps (by default they are derived from inherit information)
               Reference simulations = simulations before omission/deletion with Foscale = [0 0]
               default = []
        proba: probability of terminal nodes (their sum must be equal to 1)
               It can be an array in the same order as terminal nodes in db
               or a structure such as proba.node = value
               The structure assignment must be preferred when the order is unknown.
               default = [] assigns 1/n where n is the number of terminal nodes

    OUTPUTS
          res: structure with fields coding for all steps used as input (identified from
               res.(step) = [dCF values]
   resunique: multiple structure that match only steps without modalities
               resunique.value.(step0)=[dCF values]
               resunique.id.(step0)={corresponding names}
               resunique.proba.(step0)=[corresponding probabilities]

    Example (used for developement):
        load('C:\Data\Olivier\Communications\LNE2011\drafts\fmecabaseofconditions.mat')
        res = fmecasingle(fmecadb);
        SML=1e-3; severity=@(CF) 100*99./max(100*SML./CF-1,0);
        hfig=figure; formatfig(hfig,'figname','omission_dseverity','paperposition',[ 5.0467    7.2245   10.8907   15.2284]);
        hs = subplots([.2 1],[.6 .4],[],[],'alive',3);
        subplot(hs), hold on, i = 0;
        for step=fieldnames(res)'
            i=i+1;
            res.(step{1}) = struct('dCF',res.(step{1}),'severity',severity(res.(step{1})),'col',interp1(linspace(0,1,64),jet(64),severity(res.(step{1}))'/100 ) );
            res.(step{1}).severity(res.(step{1}).severity==0)=NaN;
            res.(step{1}).col(isnan(res.(step{1}).col))=0;
            plotpub(i*ones(size(res.(step{1}).severity)),res.(step{1}).severity,'markerfacecolor',res.(step{1}).col,'markeredgecolor',res.(step{1}).col,'linestyle','none','markersize',12,'marker','o')
            text(i,0,[step{1} '   '],'fontsize',16,'rotation',90,'HorizontalAlignment','right','VerticalAlignment','middle');
        end
        hr = refline(0,100); set(hr,'color','r','linewidth',2,'linestyle',':');
        hr = refline(0,33); set(hr,'color','c','linewidth',2,'linestyle',':');
        formatax(hs,'fontsize',14,'xticklabel',''), ylabel('Severity','fontsize',16);
        print_png(300,get(gcf,'filename'),'C:\Data\Olivier\Communications\LNE2011\drafts');
```

SEE ALSO

fmecaengine fmecaroot key2key buildmarkov loadfmecaenginedb

fmecaroot

```
fmecaroot returns the root (according to prop) of each node (to be created with FMECAENGINE)
  syntax: root = fmecaroot(fmecadb [,prop])
       fmecadb: output object of FMECAengine
          prop: 'parent' (default) or 'inherit'
```

See also

fmecaengine, fmecasingle, buildmarkov, loadfmecaenginedb, key2key

buildmarkov

```
buildmarkov builds a Markov chain based on a list keywords or values and a list of parent
   [treename,treeind] = buildmarkov(list,parent)
       treename: nx1 cell array giving all possible n paths as names (taken from list)
                 structure generated by fmecaengine
        treeind: nx1 cell array giving all possible n paths as indices
   [treename,treeind,map,succ] = buildmarkov(list,parent,'property',value)
            Implemented properties/values
               sort = 'none' (default), 'ascend' or 'descend'
             map: mxn array (m=max path depth) matching treeind but coded as find_multiple does with format=1
            succ: lxn array (l=max number of children) coding for the list of children (coded as find_multiple does with format=1)

 example
   list =   {'A1' 'B1' 'C1' 'A2' 'A3' 'A4' 'C3' 'C2' 'AA1' 'AA2' 'AB1' 'AB2'}
   parent = {''   ''   ''   'A1' 'A2' 'A3' 'C2' 'C1' 'A1'  'AA1' 'AA1' 'AB1'}
   [pathsnames,pathsind]=buildmarkov(list,parent)
     pathsnames{:}
     ans =
         'A1'    'A2'    'A3'    'A4'
     ans =
         'A1'    'AA1'    'AA2'
     ans =
         'A1'    'AA1'    'AB1'    'AB2'
     ans =
         'B1'
     ans =
         'C1'    'C2'    'C3'
```

SEE ALSO

fmecaengine, fmecasingle, loadfmecaenginedb, key2key, ismemberlist