Technical Description of CADLIVE SIMULATOR

A. PARSEDAE 6
1 Abstract 6
1.1 Input data 6
1.2 Mathematical formulas 7
2 Input data 7
2.1 Structure of input data 7
2.2 Species naming rules 9
2.3 Complex 9
2.4 Species Attribute 10
2.5 Definition of regulator-reaction equations 10
3 Conversion to mathematical models 13
3.1 TT (ordinary Transcription and Translation) 13
3.1.1 Transcription without regulators (gene0) 14
3.1.2 Transcription with regulators (gene1) 14
3.1.3 Translation 14
3.2 CMA (Conventional Mass Action) 14
3.2.1 Binding reactions 14
3.2.2 Conversion reaction 15
3.2.3 General enzyme reaction 15
3.2.4 Transport 16
3.2.5 Spontaneous decomposition 16
3.3 GMA (General Mass Action) 17
3.4 MM (simplified Michaelis-Menten equations) 18
3.5 DAEs 18
3.5.1 Steady-state approximation I 18
3.5.2 Steady-state approximation II 19
3.5.3 Rapid equilibrium approximation 19
3.5.4 Complexes in TPP 20
3.6 S-system 20
3.7 Connection between layers 21
4. Selection of conversion methods 22
4.1 Gene-protein layer 22
4.2 Metabolic layer 22
4.3 Connection between the layers 22
B. CHECKDAE 23

1 Objectives 23
2. Function 23
2.1 Grammar check for mathematical models 23
2.2 Expansion to mathematical equation trees 24
2.3 Function for checking parameters and variables 24
2.4 Function for expanding temporary expressions 24
2.5 Function for indexing variables and parameters 24
2.6 Function for replacing the indexes 24
2.7 Symbolic partial differentiation 24
2.8 Function for S-system conversion 24
2.9 Parameter survey 24
3 Specification of input and output 24
3.1 Specification of input 24
3.2 Specification of output 24
3.2.1 User function file 24
3.2.2 Parameter file 25
3.2.3 Temporary mathematical expression file 27
3.2.4 S-system coefficient file 27
4 Data structure 28
4.1 Main structures 28
4.2 Node_T structure 29
5 Procedure 30
5.1 Input procedure (Read_data) 33
5.1.1 Reading checkdae files: data split and data number acquisition (split_dae_file) 33
5.1.2 Index check and memory allocation (allocate_area). 33
5.1.3 Reading parameter labels (read_param_base) 33
5.1.4 Reading parameters (read_param) 33
5.1.5 Reading variables (read_var) 33
5.1.6 Reading temporary mathematical expressions (read_mid) 33
5.1.7 Reading differential equations and algebraic equations (read_equation) 33
5.2 Expansion of mathematical equations in a tree structure (set_expression、Make_node) 34
5.2.1 Reading mathematical expression (get_token) 34
5.2.2 Checking mathematical expression (check_token) 34
5.2.3 Adding a blank child node (add_emptyChild) 34
5.2.4 Inserting a child node (insert_childNode) 34
5.2.5 Inserting a parent node (insert_parentNode) 34
5.2.6 Deleting nodes (del_node) 34
5.2.7 Searching equivalent or less prior nodes (seek_same) 35

5.2.8 Reconstructing mathematical expressions (Set_expr) 35
5.3 Checking variables (check_node、check_detail) 35
5.4 Expanding temporary mathematical expressions (expand_mid) 35
5.5 Assigning an index to a variable (set_variable_index) 35
5.6 Converting variables (set_numerical_index, check_node, replace_variable) 35
5.7 Symbolic partial differentiation (Make_diff) 35
5.8 Output process 36
5.8.1 Output of user functions for simulators (Write_userfunc) 36
5.8.2 Output of parameter files (Write_paramFile) 36
5.8.3 Output of temporary mathematical expression obtaining file (Write_flux) 36
5.9 S-system conversion（Set_Ssystem） 36
3.2.5 S-system user function file 37
C. SOLVER 39
1. Introduction 39
2. Analysis types 39
3. Input for control data 39
D. MERGEPARAM 41
1. Objectives 41
2. Function 41
3. Input/output specification 41
3.1 Y_START 41
3.2 PARAM 41
4. Command 42
5. Error message 42
E. SENSITIVITY AND STABILITY ANALYSIS BY S-SYSTEM 43
1. Objectives 43
2. Introduction of S-system 43
2.1 S-system conversion 43
2.2 Sensitivity analysis 44
2.2.1. Logarithmic gain of a metabolite 44
2.2.2. Logarithmic gain of a flux 45
2.2.3. Sensitivity with respect to a rate constant 45
2.2.4. Sensitivity of dependent variables with respect to a kinetic order 45
2.2.5. Sensitivity of a flux to a rate constant 46
2.2.6. Sensitivity of a flux with respect to a kinetic order 46
2.3 Stability analysis 46
3 Process flow 47
4 Function 49

4.1 Creation of S-system parameter file 49
4.2 Sensitivity analysis 49
4.3 Stability analysis 49
5 Input/Output Specification 49
5.1 Input specification 49
5.2 Output specification 50
F. OPTIMIZER (GA) 52
1. Introduction 52
2. Function specification and application problems 52
2.1 Function Specification 52
2.2 Application problem 53
3. Execution of programs 53
4. Input/output specification 54
4.1 Parameter-range setting file 54
4.2 GA-parameter setting file 55
4.2.1 GA-parameter setting file 55
4.2.2 Examples for GA parameter setting file 57
4.2.3 GA-population setting file 58
4.2.4 Examples for GA-population setting file 58
4.3 Standard output 59
4.4 Output file 61
5. Detailed specification of functions 61
5.1 Encode method 61
5.1.1 Bit-string GA 61
5.1.2 Real GA 62
5.2 GA type 62
5.2.1 Distributed GA 62
5.2.2 Distributed and integrated GA (DIGA) 62
5.3 Generation alternation 62
5.3.1 Ordinary generation alternation method 63
5.3.2 Minimal Generation Gap (MGG) 63
5.4 Crossover method 63
5.4.1 BLX-α [RGA] 63
5.4.2 UNDX / UNDX-m [RGA] 64
5.4.3 SPX [RGA] 65
5.4.4 N point crossover [BGA] 66
5.5 Mutation method 66
5.5.1 Uniform mutation within the region [RGA] 66

5.5.2 Uniform mutation with fixed width [RGA] 66
5.5.3 Normal mutation with fixed width [RGA] 66
5.5.4 Uniform mutation with variable width [RGA] 67
5.5.5 Normal mutation with variable width [RGA] 67
5.5.6 Bit reverse mutation [BGA] 67
6. Flow chart 67
6.1 Single without MPI 67
6.2 MPI 68
6.2.1 Master-slave model 68
6.2.2 A flow char in MPI 69
G. CLIENT-SERVER MODEL 72

A. PARSEDAE

1 Abstract

In order to convert the regulator-reaction equations into mathematical models, CADLIVE classifies biochemical ractions into three layers: gene, protein, and metabolic layers, and divides the conversion process into two stages, the first conversion (ordinary transcription and translation equations = TT, conventional mass action = CMA, general mass action = GMA, simplified Michaelis-Menten equations = MM), and the second conversion (differential and algebraic equations = DAEs, S-system). From the standpoint of mathematical conversions, the applied mathematical conversion strongly depends on the layer that the regulator-reaction equations belong to. At the first stage, both gene and protein networks employ TT and CMA, whereas the metabolic network uses GMA or MM. In the gene layer, since various molecules such as proteins, amino acids, nucleic acids, and RNAs act in concert for transcription and translation, it is difficult to mathematically describe such reactions based on their concrete molecular mechanism. The use of TT is a rational choice for taking in gene regulations within a cell. In the conversion of metabolic networks into GMA or MM, the concentrations of enzyme-metabolite complexes are cancelled compared with those of metabolites, because the former are far less than the latter. By contrast, in the protein layer, many proteins function in a complex or modified form, thus it is not practical to cancel the concentrations of the active complexes or modified ones. Thus, the use of CMA and TT describes the protein signal transduction pathways. Problems for CMA are that its differential equations are stiff due to the huge differences in values of kinetic parameters and molecular concentrations, and that many kinetic parameters are required.

At the second stage, in order to overcome such problems, we applied the Two-Phase Partition (TPP) method to the conversion of CMA with TT to differential and algebraic equations (DAEs), thereby reducing not only the stiffness, but also decreasing the number of kinetic parameters. The TPP method divides the kinetics of molecular interactions into two phases, the molecular binding phase and the reaction phase, assuming that association/dissociation rates between proteins to be quite fast compared with the rates of synthesis/degradation of mRNAs or proteins. This is a commonly used assumption in biological systems. Consequently, TPP substitutes algebraic equations for stiff differential equations. In addition to DAEs, CADLIVE is able to convert the ordinary differential equations of TT, CMA, GMA, and MM into S-system at the steady state, which is useful for analyzing the sensitivity and stability in symbolic form.

The parsedae module converts the regulator-reaction equations (sanac file Fig. A1), which are constructed by the CADLIVE editors, into the mathematical equations (checkdae file) including TT, CMA, GMA, MM, DAEs, and S-system. The checkdae file employs the indexes of species' names so that one can understand the model or edit it manually. One is allowed to edit the checkdae file directly according to the instruction. The checkdae file is written in the text format.

1.1 Input data

The parsedae module inputs the "sanac" file (Fig. A1) that mainly consists of two kinds of data. One is the species (molecules) with their associated attributes. The other is the regulator-reaction equation that determines the interaction among species.

1.2 Mathematical formulas

The following mathematical formulas have been employed.

Protein layer

CMA (Conventional Mass Action) is applied to the protein layer, focusing on the formation of complexes and modified proteins.

Gene layer

TT (ordinary transcription-translation equations) is applied to the gene layer.

Metabolic layer

GMA (General Mass Action) is applied to the metabolic layer, neglecting detailed mechanisms including the formation of enzyme-metabolite complexes.

MM (simplified Michaelis-Menten equations) is applied to the metabolic layer, canceling the concentrations of enzyme-metabolite complexes. MM is obtained by simplifying ordinary Michaelis-Menten equations.

All the layers

DAEs (differential and algebraic equations) are applied to the gene-protein layer using the two-phase partition (TPP) method.

S-system can be applied to all the layers.

2 Input data

2.1 Structure of input data

Systems Biology Markup Language (SBML) is one of the most advanced markup language that describes biochemical networks at a concrete molecular interaction level. We extend SBML level 2 to establish the sanac format that includes all the information necessary for mathematical conversion and dynamic simulation. The original sanac file (Fig. 1A) contains various data including the coordinates of the species, which are used for drawing a biochemical network map by the CADLIVE GUI editors. Here we omit the data that are not necessary for mathematical conversion and dynamic simulation.

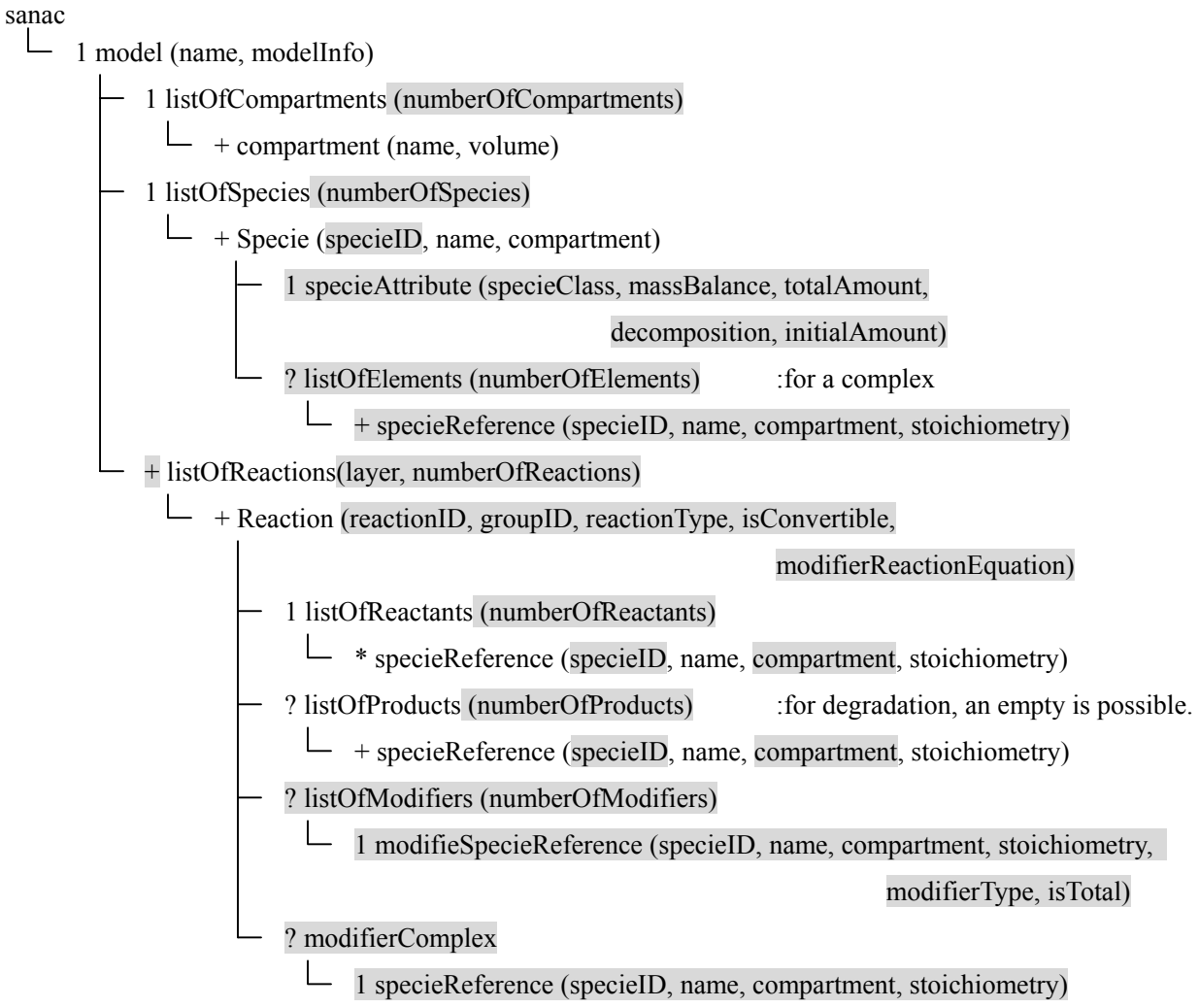


Fig. A1 Conversion rules implemented in the "sanac" representation. They are necessary and sufficient for computationally converting biochemical networks into mathematical models. The regulator is exactly the same as the SBML-defined modifier. The net regions are the extension from SBML level 2. Legend: 1: an element that appears once. *: an element that repeats more than zero. +: an element that repeats more than one. ?: an element that appears once or zero. t: an element that has text in content. (): provided as attributes. The net regions are the extension from SBML level 2.

The major extensions are as follows:

- The attribute of "specieID" refers from the lists of reaction equations or species.
- The element of <listOfElements> shows the components of the complexes, which are defined as "specieReference".
- The attribute of "layer" is added to the element of <listOfReactions>, whereby a biochemical network can be divided into three layers, metabolic (metabolic), protein signal transduction (protein), and gene regulatory networks (gene).
- The attribute of "modifierReactionEquation" is added to the element of <reaction>.

- The elements of <listOfModifiers> and <modifierComplex> are added to the element of <Reaction>, which are necessary for mathematical conversion. The modifierComplex corresponds to substrate-enzyme complex in ordinary enzyme reactions.
- The attribute of "reactionType" is added to distinguish regulator-reaction equations, which are indispensable for mathematical conversion.
- The attribute of "modelInfo" is added to comment the model.

2.2 Species naming rules

The following characters are allowed to name species

Initial character

- upper and lower alphabets [a-z] | [A-Z]
- number [0-9]
- left parentheses (

Second characters

- upper and lower alphabets [a-z] | [A-Z]
- underbar _
- number [0-9]
- left and right parentheses ( )
- colon :
- hyphen _
- period .

The period may be used for indicating a decimal point.

2.3 Complex

A complex is defined by listing its components as specieReference in <listOfElements>. The use of "stoichiometry" of "specieReference" is able to determine the ratio of components. Not only a monomer/modified but also a complex is allowed to list as the components of the complex, thus users are able to define a higher complex by using complex, modified, or monomer. It is required to keep the naming rules. The employed species are required not to show reference circulation and to be defined in the model.

The element of <listOfElements> is used:

1: to assign the species decomposition to the respective differential equations,

2: to generate the mass balance equations and the differential equations for the total amounts of the species whose class is protein or modified in the TPP conversion,

3: to calculate the total concentrations of promoters and enhancers in the "gene1" reaction.

2.4 Species Attribute

The species is defined by three indispensable attributes:

- specieID :The integer unique to the combination of species' name with its compartment
- name :Users can name the species arbitrarily.
- compartment :Location of the species

The other attributes are provided as shown in Table A2.

Table A2 Attributes of species (molecules)

Attribute	Content	Value
specieID	the integer unique to the combination of species' name with its compartment	
Name	name of species	named by user
Compartment	location of species	see text
SpecieClass	kinds of species	DNA_gene
		DNA_promoter
		DNA_enhancer
		DNA_others
		RNA
		protein
		metabolite
		environmental_factor
		ion_signal
		complex
		modifier_complex
		modified
		small_molecule
		text_option
		others
massBalance	total mass balance equation is required.	on
		off
totalAmount	total concentration	constant
		variable
decomposition	spontaneous degradation in vivo	on
		off
initialAmount	initial concentration of species	real value

2.5 Definition of regulator-reaction equations

A model has multiple elements of <listOfReactions>, which has two indispensable attributes of "layer". The attribute of "layer" has three values, "metabolic", "protein", and "gene", which indicate where reactions occur. The metabolic layer indicates metabolic networks, the protein layer signal transduction pathways, and the gene layer transcription and translation. The mathematical formulas are determined by the layer that reaction belongs to.

- layer :determining the layer that reactions occur, which consists of "metabolic", "protein", and "gene".
- numberOfReactions :the number of reactions.

The element of <reaction> consists of the elements of <listOfReactants>, <listOfProducts>, <listOfModifiers>, and <modifierComplex>

- listOfReactants :defines reactant(s)
- listOfProducts :defines product(s), For "reactionType = degradation", an empty is possible.
- listOfModifiers :define modifier.
- modifierComplex :define the complex containing a modifier.

The " reactionType" attribute of <reaction> is shown in Table A3.

Table A3 Values of the attribute of reactionType

Value
binding
binding_with_stoichiometric_changes
homo_association_or_modification
homo_association_or_modification_with_stoichiometric_changes
elimination
elimination_with_stoichiometric_changes
reversible_conversion
irreversible_conversion
reversible_conversion_regarding_multicomponent
irreversible_conversion_regarding_multicomponent
transport
option_transport
transcription
translation
degradation

The elements of <listOfReactants>, <listOfProducts>, <modifierComplex>, and <listOfElements> have to be referred from <specieReferences>, which consists of four attributes, "specieID", "name", "compartment", and "stoichiometry".

- specieID :
- name :
- compartment :cytoplasm, nucleoplasm, membrane, environment, …..
- stoichiometry :defined by real values. The default value is one.

The element of <listOfModifiers> must be referred from <modifierSpecieReference>, which is the element of <specieReference> with the attributes of "modifierType" and "isTotal".

- modifierType :enzyme, activator, inhibitor
- isTotal :false, true

Here we explain the detailed rules by using the nitrogen assimilation system. The attribute of "isConvertible" in the element of <reaction> determines if the regulator-reaction equations can be converted into mathematical equations. The parsedae module of CADLIVE converts all the reactions with "isConvertible = on", whereas the reactions with "isConvertible = off" are excluded from the mathematical model.

The parsedae module recognizes the variables prior to conversion of the reactions with "isConvertible = on". The attribute of "totalAmount" of the element <specieAttribute> is introduced to determine whether the total concentration for the species is constant (independent variable) or variable (time-dependent variable). When "totalAmount" of protein is "variable", the protein is synthesized and decomposed according to TT, i.e., the total concentration varies with time. When the total concentration of the protein with "totalAmount = constant" is independent of time, protein synthesis and degradation are assumed not to occur. In addition, the parsedae module distinguishes the species by the attribute "compartment" in the element of <specieAttribute> that indicates which compartment the species is located.

At the first stage that generates ordinary differential equations, such as TT, CMA, GMA, and MM, the combined attributes of "layer" in the elements of <listOfReactions>, "reactionType" and "groupID" of the elements of <Reactions>, and "modifierType" in the element of <listOfModifier> produce the appropriate differential equations. The attribute of "layer", which has three values, "metabolic", "protein", and "gene", determines the basic mathematical formula. The metabolic layer employs GMA or MM, the protein layer, which indicates signal transduction pathways through protein interactions, uses CMA, and the gene layer uses TT. The attribute of "reactionType", which consists of the various kinds of regulator-reaction equations, determines which type of differential equations is employed in each type of reaction. The element of <listOfModifier> presents concrete regulator information, where the attribute of "modifierType" indicates the type of modifiers, "enzyme", "activator", or "inhibitor". The value of "enzyme" indicates general enzyme reactions, whereas "activator" or "inhibitor" is used for "reactionType = transcription". In transcription, an activator binds to an enhancer to enhance the transcription of a gene, while an inhibitor binds to a promoter to repress the transcription of a gene. The attribute of "groupID" is required to group a series of reactions, where the multiple regulators with the same "groupID" act on the identical reaction. They are usually used for the "gene1" transcription. For example, when multiple transcription regulation factors (TRFs) act on the transcription of a gene, these regulator-reaction equations with the same "groupID" generate one transcription regulation equation with multiple TRFs.

In order to express the spontaneous degradation of proteins and mRNAs *in vivo*, the attribute of "decomposition" is employed in the element of <specieAttribute>. Generally, proteins and RNAs constitutively degrade *in vivo* due to factors including proteases ("decomposition =on "), whereas DNAs do not degrade ("decomposition =off "). When the species consists of multiple components, the attribute of

"decomposition" has to be searched recursively, while checking whether "decomposition" of each component is on or off.

In the second process, the attributes of "specieClass" "massBalance", and "totalAmount" in the element of <specieAttribute> play intrinsic roles in converting the ordinary differential equations into DAEs. The attribute of "specieClass" indicates the kinds of species, such as DNAs, RNAs, proteins, complexes, and metabolites. This attribute of "specieClass" determines which components are needed to make the mass balance equations for each species, e.g., "monomer", "modified", "enhancer", or "promoter". Binding multiple molecules without any stoichiometric change generates "complex", whose components are listed in <listOfElements>. The parsedae module automatically names the complex by joining its associated elements with a colon (:), e.g., PI:PII. The value of "modified" indicates the molecules that are chemically modified through modification processes such as phosphorylation and acetylation, which are accompanied with stoichiometric changes. The value of "modified" is automatically named by joining the modifying molecule to the modified one using a hyphen (-), e.g., phosphorylated NRI is expressed as NRI-P. For example, the molecules of PI:PII, NRI-P:Enhancer, and NRI-P:NRII are "complex", whereas NRI-P and NRII-P are "modified". The element of "modifierComplex" means an active complex consisting of an enzyme and substrates, which is automatically generated when an enzyme reaction is defined. The attribute of "massBalance" determines whether total mass balance equations are made for the species with "massBalance = on", e.g., monomer, modified, enhancer, or promoter. TPP applies the quasi-steady state or rapid equilibrium approximation to the differential equations for the species with "specieClass = complex". For the species with "massBalance = on", the mass balance equations are generated that sums all the complexes containing the species as components. The total concentration of the species is named by adding "T" to the head of species' name, whereas species' name without "T" indicates a free molecule.

In order to combine the gene and protein layers with the metabolic layer, the attribute of "isTotal" is presented to determine if a modifier acts as a free species or as the total amount of the species. It is a critical attribute when the concentration of the enzyme is defined as a time-dependent variable in the metabolic layer. With "isTotal = true" for an enzyme, the total concentration of the enzyme is employed for GMA or MM type. Conversely, with "isTotal = false" for an enzyme, the free concentration of the enzyme is used for the metabolic layers. This rule enables one to combine the three layers, i.e., metabolic and gene regulatory networks.

3 Conversion to mathematical models

3.1 TT (ordinary Transcription and Translation)

In the gene layer, transcription and translation occur in complicated manners, which involve a large number of nucleic acids, amino acids, RNAs and proteins. It is not practical to describe all possible reactions, and it is difficult to convert the transcription or translation (gene layer) into CMA based on their detailed molecular mechanism. Thus, we apply a general type of differential equations suitable for transcription and translation equations. For transcription, we conveniently divide two types: gene0 and gene1. For gene0, no transcription regulation factor involves transcription; for gene1, activators or inhibitors regulate the transcription.

3.1.1 Transcription without regulators (gene0)

Assuming that no gene regulation involves transcription, the transcription occurs at a constant rate, which is provided by:

Regulator-reaction equations: $gene(A) \rightarrow mRNA(A)$ (A1)

Differential equations: $\frac{d[mRNA(A)]}{dt} = k_m[gene(A)]$ (A2)

where k_m is the transcription rate constant.

3.1.2 Transcription with regulators (gene1)

For gene1, since activators or inhibitors regulate the transcription, regulator-reaction equations are provided by:

$$Activator_i : Enhancer_i \quad ->> Gene \longrightarrow mRNA . \tag{A3}$$

$$Suppresor_i : Promoter_i \quad -\| Gene \longrightarrow mRNA \tag{A4}$$

The synthesis rates are converted into:

$$\frac{d[mRNA]}{dt} = k_m \cdot \{\sum_{i=1}^{n} or \prod_{i=1}^{n} \frac{[Activator_i : Enhancer_i]}{[TEnhancer_i]} (+or\times) \sum_{j=1}^{m} or \prod_{j=1}^{m} (1 - \frac{[Suppressor_j : Promoter_j]}{[TPromoter_j]})\} \cdot [Gene] \tag{A5}$$

$$\frac{d[Protein]}{dt} = k_p \cdot [mRNA] \tag{A6}$$

where k_m is the transcription rate constant, and k_p is the translation rate constant. Complicated transcriptions can be described by using the GUI editors.

3.1.3 Translation

For example, the protein A is synthesized from *mRNA(A),* which is given by:

$$\frac{d[A]}{dt} = k_p[mRNA(A)], \tag{A7}$$

where k_p is the translation rate constant.

3.2 CMA (Conventional Mass Action)

Regulator-reaction equations can be divided into binding reactions and conversion reactions.

3.2.1 Binding reactions

For example, a binding reaction is provided by:

$$\alpha_1\ A_1 + \alpha_2 A_2 + \ldots + \alpha_n\ A_n \underset{ks}{\overset{ka}{\rightleftarrows}} \alpha_1\ A_1 : \alpha_2 A_2 : \ldots : \alpha_n\ A_n \quad , \tag{A8}$$

where α_i is the stoichiometry.

$$\frac{d[A_i]}{dt} = \alpha_i \left\{ -ka \prod_{j=1}^{n} [A_j]^{\alpha_j} + kd[\alpha_1\ A_1 : \alpha_2 A_2 : \ldots : \alpha_n\ A_n] \right\} \qquad (\mathrm{i} = 1 \sim \mathrm{n}) \tag{A9}$$

$$\frac{d[\alpha_1\ A_1 : \alpha_2 A_2 : \ldots : \alpha_n\ A_n]}{dt} = ka \prod_{j=1}^{n} [A_j]^{\alpha_j} - kd[\alpha_1\ A_1 : \alpha_2 A_2 : \ldots : \alpha_n\ A_n], \tag{A10}$$

where ka is the binding association rate constant, and *kd* is the dissociation rate constant. The amount of the corresponding species on the left and right sides has to be the same. The parsedae checks the stoichiometry of reactions. In addition, the parsedae module checks the components of the complex. For the binding reaction equations:

$$A_1 + (A_2 : A_3) \underset{ks}{\overset{ka}{\rightleftarrows}} A_1 : (A_2 : A_3) \quad \text{and} \quad A_1 + (A_2 : A_3) \underset{ks}{\overset{ka}{\rightleftarrows}} A_1 : A_2 : A_3 \tag{A11}$$

are allowed, but the equation:

$$A_1 + A_2 + A_3 \underset{ks}{\overset{ka}{\rightleftarrows}} A_1 : (A_2 : A_3) \tag{A12}$$

is not allowed, when the parsedae outputs the error.

3.2.2 Conversion reaction

The reaction:

$$\alpha_1\ A_1 : \alpha_2 A_2 : \ldots : \alpha_n\ A_n \xrightarrow{kx} \beta_1 B_1 + \beta_2 B_2 + \ldots + \beta_m B_m \tag{A13}$$

is converted into CMA:

$$\frac{d[B_i]}{dt} = \beta_i\ kx[\alpha_1\ A_1 : \alpha_2 A_2 : \ldots : \alpha_n\ A_n] \quad (\mathrm{i} = 1 \sim \mathrm{n}) \tag{A14}$$

$$\frac{d[\alpha_1\ A_1 : \alpha_2 A_2 : \ldots : \alpha_n\ A_n]}{dt} = -kx[\alpha_1\ A_1 : \alpha_2 A_2 : \ldots : \alpha_n\ A_n], \tag{A15}$$

where kx is the conversion rate constant.

3.2.3 General enzyme reaction

General enzyme reactions except for transcription and translation can be provided by:

$$\alpha_1 A_1 + \alpha_2 A_2 + \ \ldots\ + \alpha_n A_n \underset{kd}{\overset{ka}{\rightleftarrows}} \alpha_1 A_1 : \alpha_2 A_2 : \ldots : \alpha_n \xrightarrow[kx]{} \beta_1 B_1 + \beta_2 B_2 + \ldots + \beta_m B_m \,, \tag{A16}$$

where A_i is the substrates, B_i the products, α_i and β_i are the stoichiometric coefficients, k_a is the association rate constant and k_d is the dissociation rate constant, and k_x is the conversion rate constant. This reaction is converted into CMA as follows:

$$\frac{d[A_i]}{dt} = \alpha_i \left\{ -ka \cdot \prod_{j=1}^{n} [A_j]^{\alpha_j} + kd[\alpha_1 A_1 : \alpha_2 A_2 : \ldots : \alpha_n \ A_n] \right\} \quad (i = 1, ..., n). \qquad \text{(A17)}$$

$$\frac{d[\alpha_1 A_1 : \alpha_2 A_2 : \ldots : \alpha_n A_n]}{dt} = ka \prod_{j=1}^{n} [A_j]^{\alpha_j} - kd[\alpha_1 A_1 : \alpha_2 A_2 : \ldots : \alpha_{\beta_i} A_n] - \beta_i kx[\alpha_1 \ A_1 : \alpha_2 A_2 : \ldots : \alpha_n \ A_n] \qquad \text{(A18)}$$

$$\frac{d[B_i]}{dt} = \beta_i \cdot kx[\alpha_1 A_1 : \alpha_2 A_2 : \ldots : \alpha_n A_n] \quad (i = 1, ..., m) \qquad \text{(A19)}$$

3.2.4 Transport

The species A is transported from one compartment to another compartment (A')

$$\frac{d[A]}{dt} = -ktr[A], \qquad \text{(A20)}$$

$$\frac{d[A']}{dt} = +ktr[A] \qquad \text{(A21)}$$

where *ktr* is the transport rate constant. Active transport with an enzyme is processed in the same manner as a conversion with an enzyme.

3.2.5 Spontaneous decomposition

Proteins and RNAs are generally degraded *in vivo*. Thus, we omit the regulator-reaction equations for spontaneous decomposition. Instead, we prepare the attribute of decomposition for species. With "decomposition = on" for a species, the parsedae module adds the spontaneous degradation term to the differential equation for the species, as follows:

$$\frac{d[A]}{dt} = -kpd[A], \qquad \text{(A22)}$$

where *kpd* is the degradation rate constant. For mRNA, *kmd* is defined as the degradation rate constant for mRNAs. Notice that the reaction type of "degradation" is different from the spontaneous decomposition.

When the components have complexes, the parsedae checks the attributes of decomposition recursively, while checking whether "decomposition" is "on" or "off". For example, when the decomposition of A, B, C, A:B, B:2D, (A:B):(2(B:2D)):(3(C:D)) is on, that of D, C:D is off, the differential equations for the respective components include the following terms :

i) d[A]/dt = - *kpd*[1] * [A]

ii) d[B]/dt = - *kpd*[2] * [B]

iii) d[C]/dt = - *kpd*[3] * [C]

iv) d[D]/dt = + 2**kpd*[5] * [B:2D] + 4**kpd*[6] * [(A:B):(2(B:2D)):(3(C:D))]

v) d[A:B]/dt = - *kpd*[4] * [A:B]

vi) d[B:2D]/dt = - *kpd*[5] * [B:2D]

vii) d[C:D]/dt = + 3 * *kpd*[6] * [(A:B):(2(B:2D)):(3(C:D))]

viii) d[(A:B):(2(B:2D)):(3C:D)]/dt = - *kpd*[6] * [(A:B):(2(B:2D)):(3(C:D))]

Since the decomposition for the species of (A:B):(2(B:2D)):(3(C:D)) is on, the decomposition terms are added to the respective differential equation (vii). Next, the decomposition of its components of A:B, B:2D, and C:D is checked. Since the decomposition of A:B, A, and B is on, no source term is added to the equations (i,ii,v). The decomposition of B:2D is on, the decomposition term is not added to the respective differential equation (iv). However, since the decomposition of D is off, and the decomposition of B:2D releases 2*D, the source term for D is added to the equation (iv). Since C:D is not decomposed despite the decomposition of (A:B):(2(B:2D)):(3(C:D)), the source term for C:D is added to the equation (vii).

3.3 GMA (General Mass Action)

Generally, metabolic reactions can be described using General Mass Action (GMA) or the Michaelis-Menten equations (MM). GMA assigns each reaction to a mathematical term having a kinetic rate constant and power coefficients regarding the concentrations of metabolites and enzymes, as described by:

$$\frac{d[X_i]}{dt} = \sum_{k=1}^{m} k_{ik} \prod_{j=1}^{n} [X_j]^{f_{ijk}} \qquad \text{(A23)}$$

where X_1~X_n are the dependent variables, whose values vary with time, X_{n+1}~X_{n+m} are the independent variables, whose values are fixed as constants, and k_{ik} is the rate constant. We show an example for mathematical conversion for metabolic reactions provided by:

$$E \text{ –}o\ S_1 + S_2 > P_1 + P_2 . \qquad \text{(A24)}$$

The GMA conversion cancels the concentration of the enzyme-metabolite complex of $E:S_1:S_2$, thus the differential equation for $E:S_1:S_2$ is not produced, as follows.

$$\frac{dS_1}{dt} = - \ kxg_1 E^{f_1} S_1^{f_2} S_2^{f_3} \qquad \text{(A25)},$$

$$\frac{dS_2}{dt} = - \ kxg_1 E^{f_1} S_1^{f_2} S_2^{f_3} \qquad \text{(A27)},$$

$$\frac{dP_1}{dt} = + \ kxg_1 E^{f_1} S_1^{f_2} S_2^{f_3} \qquad \text{(A28)}$$

$$\frac{dP_2}{dt} = + \ kxg_1 E^{f_1} S_1^{f_2} S_2^{f_3} \qquad \text{(A29)},$$

where kxg_1 is the kinetic rate constant, f_i are the power coefficients. If necessary, GMA produces the term for spontaneous decomposition as follows:

$$\frac{dS}{dt} = - kxgS^{f} . \qquad \text{(A30)}$$

3.4 MM (simplified Michaelis-Menten equations)

Generally, the Michaelis-Menten equations are able to describe complicated reactions with activators or inhibitors based on their mechanism, where the concentrations of the enzyme-metabolite complexes are cancelled. The CADLIVE Simulator further simplifies the Michaelis-Menten equations. This simplification enables one to convert complicated metabolisms into simple forms efficiently. For example, the metabolic equations can be converted into the simplified Michaelis-Menten type differential equations. The regulator-reaction equation:

$$E - o\ S_1 + S_2 > P_1 + P_2 \qquad \text{(A31)}$$

is converted to :

$$\frac{dS_1}{dt} = - \ Q_1 E \frac{S_1}{Kmich_1 + S_1} \frac{S_2}{Kmich_2 + S_2} \qquad \text{(A32)},$$

$$\frac{dS_2}{dt} = - \ Q_1 E \frac{S_1}{Kmich_1 + S_1} \frac{S_2}{Kmich_2 + S_2} \qquad \text{(A33)},$$

$$\frac{dP_1}{dt} = + \ Q_1 E \frac{S_1}{Kmich_1 + S_1} \frac{S_2}{Kmich_2 + S_2} \qquad \text{(A34)},$$

$$\frac{dP_2}{dt} = + \ Q_1 E \frac{S_1}{Kmich_1 + S_1} \frac{S_2}{Kmich_2 + S_2} \qquad \text{(A35)},$$

where K_{mich1} and K_{mich2} are the Michaelis-Menten constants. If necessary, this conversion also produces the term for spontaneous decomposition as follows,

$$\frac{dS}{dt} = - Q \frac{S}{Kmich + S} \qquad \text{(A36)}.$$

3.5 DAEs

In order to solve the stiffness in CMA expressing a gene regulatory network, the two-phase partition method (TPP) has been proposed that divide biochemical reactions into two phases, binding and reaction phases, by applying rapid equilibrium approximation or quasi-steady state approximation to complex formation, which greatly reduces the number of kinetic parameters and remarkably accelerates the calculation speed. TPP is a powerful method that is indispensable for mathematical simulation of gene regulatory networks.

3.5.1 Steady-state approximation I

The two-phase partition method begins with CMA and TT.

(1) The differential equation for the complexes that have been generated through a binding reaction is set as zero, resulting in an algebraic equation. The generated equations are simplified. For example, when a + b + c + d = 0 and b – d = 0 is generated, the former is replaced by a + c = 0.

(2) For the species that do not involve binding reactions and those whose attribute of "massBalance" is off, their differential equations do not change.

(3) For the monomers that involve binding reactions and whose attribute of "massBalance" are on, the concentrations of the monomer and its derivative complexes are all added, producing the differential equations for their total concentration. For the species whose attribute of "massBalance" is off, they are not. For example, for the species regarding the monomer A, the equations are made as follows:

dy[TA]/dt = differential equations for monomer A
+ Σ (differential equations for the complex that contains A as a component x the number of A that the complex contains)

(4) The mass balance equations for the species whose "specieClass" is "monomer" is provided by:

[TA] = [A] + Σ ([complex that contains A as a component] x the number of A that the complex contains)

(5) After simplifying the DAEs, the differential equations whose right hand sides become zero are omitted, and the species are registered as constant values.

The species are classified as follows:

A) Constant

The species whose attribute of "totalAmount" is constant, and whose "massBalance" is off.

The species whose differential equation is zero (dy/dt=0).

B) Variables of algebraic equations

The species whose class is "complex".

The species whose class is "monomer" and whose "massBalance" is on.

C) Variables of differential equations

The total concentration of the species with the attribute of monomer and modified. (excludes the species whose differential equation is zero (dy/dt=0).)

3.5.2 Steady-state approximation II

The algebraic equations at the steady state are further simplified by setting *kpd* = 0.

3.5.3 Rapid equilibrium approximation

The algebraic equations at the steady state are further simplified by setting *kpd*, *kx* << *ka, kd*. In addition, the association constant of *Kb* is substituted for *ka/kd*. For example, the regulator-reaction equation:

mA + nB <-> mA:nB (A37)

is converted to the algebraic equation:

[mA:nB] = *Kb*^(m*n)*[A]^m*[B]^n. (A38)

For example, the general enzyme reaction (A16) is converted into DAEs using the rapid equilibrium approximation, as follows,

$$0 = -K \cdot \prod_{j=1}^{n} [A_j]^{\alpha_j} + [\alpha_1 A_1 : \alpha_2 A_2 : \ldots : \alpha_n A_n] \quad \text{(A39)},$$

$$[TA_i] = [A_i] + \alpha_i [\alpha_1 A_1 : \alpha_2 A_2 : \ldots : \alpha_n A_n] \quad (i = 1, ..., n) \quad \text{(A40)}$$

$$\frac{d[TA_i]}{dt} = -\beta_i \cdot kx \cdot [\alpha_1 A_1 : \alpha_2 A_2 : \ldots : \alpha_n A_n] \quad (i = 1, ..., m) \quad \text{(A41)}$$

$$\frac{d[B_i]}{dt} = \beta_i \cdot kx \cdot [\alpha_1 A_1 : \alpha_2 A_2 : \ldots : \alpha_n A_n] \quad (i = 1, ..., m) \quad \text{(A42)}$$

where K is the association constant that is provided by (k_a / k_d).

3.5.4 Complexes in TPP

We illustrate in detail how TPP derives the differential equations for the total amount of specific species and their mass balance equations. The regulator-reaction equation:

A + 2B <-> A:2B (A43)

is converted into the CMA:

d[A]/dt = - *ka*[A][B]^2 + *kd*[A:2B] (A44)

d[B]/dt = - 2*ka*[A][B]^2 + 2*kd*[A:2B] (A45)

d[A:2B]/dt = + *ka*[A][B]^2 – *kd*[A:2B] (A46).

We apply TPP to the CMA by setting the equation (A46) as zero, producing the differential equations for the total concentrations (TA, TB) for the components of A and B. In order to make the differential equation for TB, we recursively search all the components of the complexes to find the species that contain B as a component, and add them to the differential equation. They are multiplied by the number of B contained in the complexes

d[TB]/dt = the right hand side of (A45) + the right hand side of (A46) x 2

= Σ (the right hand sides of the CMA for the species that contains B as a component x the number of B that the species contains)

In the same manner, the mass balance equation for TB is produced as follows:

[TB] = [B] + [A:2B] x 2

= Σ (the concentrations of the species that contain B as a component x the number of B that the species contains)

3.6 S-system

Sensitivity and stability analysis shows how a biochemical system responds to perturbations or uncertainties. S-system is able to analyze the various sensitivities of the system at the steady state in symbolic form. In the simulator, ordinary differential equations such as TT, CMA, GMA, and MM can be converted into S-system to analyze the sensitivities and stability at the steady state.

The ordinary differential equations are divided into the positive terms and negative terms:

$$\frac{dX_i}{dt} = V_i^+ - V_i^- , \qquad (i = 1,....,n,....n+m) \qquad (A47),$$

where $\mathbf{V_i^+}$ is the sum of positive terms, and $\mathbf{V_i^-}$ is the sum of negative terms. Generally, S-system is given by:

$$\frac{dX_i}{dt} = \alpha_i \prod_{j=1}^{n+m} X_j^{g_{ij}} - \beta_i \prod_{j=1}^{n+m} X_j^{h_{ij}} \qquad (A48),$$

where X_1 - X_n are the dependent variables, whose values vary with time, and X_{n+1} - X_{n+m} are the independent variables, whose values are fixed as constants. When the concentrations of the dependent variables are given at the steady state, the coefficients ($g_{ij}, h_{ij}, \alpha_i, \beta_i$) of S-system are solved from Eq. (A48) in symbolic form:

$$g_{ij} = \frac{\partial V_i^+}{\partial X_j} \frac{X_j}{V_i^+} \qquad (A49),$$

$$h_{ij} = \frac{\partial V_i^-}{\partial X_j} \frac{X_j}{V_i^-} \qquad (A50),$$

$$\alpha_i = \frac{V_i^+}{\prod_{j=1}^{n+m} X_j^{g_{ij}}} \qquad (A51),$$

$$\beta_i = \frac{V_i^-}{\prod_{j=1}^{n+m} X_j^{h_{ij}}} \qquad (A52),$$

where g_{ij} and h_{ij} are the kinetic order and α_i and β_i are the rate constants.

3.7 Connection between layers

When DAEs express the gene-protein layers, and GMA or MM expresses the metabolic layer, dependent variables for the regulators that appear as modifiers in the metabolic layer correspond to dependent variables for the gene-protein layers. Table A4 shows how to connect the corresponding variables between the gene-protein layer and the metabolic layer. The combination of the values of the attributes of "isToal", "massBalance" and the method for conversion determines the variables that appear as a modifier in the metabolic layer.

Table A4 Connection between layers

Conversion method for Gene-Protein layer	massBalance	isTotal	The variable of E that appears as a modifier in the metabolic layer
CMA	on	true	Total concentration of enzyme: T_E.cyt
		false	y[E.cyt]
	off	true	Total concentration of enzyme T_E.cyt
		false	y[E.cyt]
DAEs (TPP)	on	true	y[TE.cyt]
		false	x[E.cyt]
	off	true	Total concentration of enzyme T_E.cyt
		false	y[E.cyt]

The total concentration of the enzyme E is the total amount of E and the complexes that contain E as a component. For example, when TE consist of E, E:A:B, and E:C, the total concentration for the enzyme E is given by:

$$T\_E.cyt = y[E.cyt] + y[E:A:B.cyt] + y[E:C.cyt] \quad (A53)$$

4. Selection of conversion methods

Selection of conversion methods is carried out for the two systems: gene-protein layer and metabolic layer.

4.1 Gene-protein layer

For the gene-protein layers, users select four types of conversion methods: CMA, TPP_STEADYSTATE_1, TPP_STEADYSTATE_2, and TPP_RAPID, which are suitable for converting the regulator-reaction equations that generate various complexes. The gene-protein layer can be converted into GMA or MM, but it is not practically useful. Thus, such conversions are error.

4.2 Metabolic layer

For the metabolic layer, GMA and MM are generally employed. It is possible to convert the metabolic layer by using CMA or TPP, but it is not useful practically.

4.3 Connection between the layers

Since the different conversion methods are applied to the gene-protein and metabolic layers, the differential equations for the species common to both layers have to be combined to make one differential equation, which enables one to simulate the three layers together.

B. CHECKDAE

1 Objectives

The variables employed in the checkdae file are named by the index made of the combination of species' name and its localization, as exemplified by y[RNAP.cyt], so that users can understand the meaning of the variables. However, in order for computers to simulate mathematical equations (TT, CMA, GMA, MM, DAEs, S-system), it is required to replace the indexes of variable names by sequential integers, to make the user functions for mathematical equations, and to produce their Jacobian function that is the partial derivatives of the user functions with respect to each variable in symbolic form. The checkdae module generates computer-readable parameters, mathematical functions, and their Jacobian. These functions and parameter files are described in the C programming language.

The checkdae module examines if a checkdae file (mathematical equations), which the parsedae module has generated or users have edited manually, is acceptable for converting it into simulation programs (user functions and parameters). The checkdae module generates not only the user functions for mathematical equations and their associated Jacobian, but also temporary mathematical expressions such as fluxes. In addition, the checkdae module is able to convert ordinary differential equations (TT, CMA, GMA, MM) into S-system at the steady state, generating the user functions for S-system with its associated coefficient functions and their parameters. Simultaneously, it creates the parameter files for setting initial values and kinetic parameters.

In summary, from a checkdae file, the checkdae module creates the five files: a parameter file, a user function file, a temporary mathematical expression file, an S-system coefficient file, and an S-system user function file.

2. Function

Finishing checking the grammar of a mathematical model (checkdae file), the checkdae module divides it into the nodes consisting of one operator in order to expand them on a binary tree structure. The checkdae first examines if the employed parameters and variables are defined in a checkdae file. Second, it converts the index of species' names to the numerical one, and solves the partial derivatives of the mathematical equations to obtain their Jacobian in symbolic form, resulting in a user function file with a parameter file that contains initial values and kinetic constants. The use of GMA or MM conversion creates another file that contains the temporary mathematical expressions for its fluxes. When S-system is employed, it produces an S-system coefficient file and an S-system user function file.

2.1 Grammar check for mathematical models

The checkdae examines the correctness of the checkdae file (mathematical equations) and temporary mathematical expressions from the standpoint of grammar, which are restricted to five operators consisting of addition (+), subtraction (-), multiplication (*), division (/), and power (^). The symbols except numbers are all regarded as variables.

2.2 Expansion to mathematical equation trees

Mathematical equations are expanded on binary trees whose nodes consist of an operator. The subsequent operations are carried out based on the binary trees.

2.3 Function for checking parameters and variables

This function checks if the variables, parameters, and temporary mathematical expressions employed in mathematical equations have been defined in the checkdae file.

2.4 Function for expanding temporary expressions

The temporary mathematical equations are expanded to generate a user function, and then the temporary equations are removed from the user function.

2.5 Function for indexing variables and parameters

The sequential integers are provided for all the variables and parameters that users have defined.

2.6 Function for replacing the indexes

The numerical indexes are substituted for the variables named with the combination of species' name and its compartment.

2.7 Symbolic partial differentiation

The partial derivatives are solved with respect to the variables of interest in symbolic form.

2.8 Function for S-system conversion

Ordinary differential equations are converted into S-system. The detailed explanation is described elsewhere.

2.9 Parameter survey

Users are able to explore the parameter spaces by using the parameter file, where the values of parameters are varied in the arithmetic or geometric series. Details are described in the parameter file (3.2.2).

3 Specification of input and output

3.1 Specification of input

The checkdae module reads the checkdae file (mathematical equations) to make the user functions and their associated parameter files necessary for simulation. The details of the checkdae file are explained in the instruction of the parsedae module.

3.2 Specification of output

3.2.1 User function file

The user function consists of mathematical equations and their Jacobian. The Jacobian is obtained in symbolic form

by solving the partial derivatives of mathematical equations with respect to each variable. The user function contains two functions: usr_fvec and usr_fjac, as follows.

```
/*
 * TITLE:test
 * INFO:test
 * CONVERSION TYPE
 *        GENE-PROTEIN:TPP_RAPID
 *        METABOLIC:NONE
 */

#include "life.h"

void usr_fvec(double fvec[], double y[], double Gene, Parameter *p)
{
        double *constantPlayer          = p->val[1];
        double *Kb              = p->val[2];
        double *kp              = p->val[3];
        double *kpd             = p->val[4];
        double *km              = p->val[5];
        double *kmd             = p->val[6];

        fvec[    1] = y[12] - (y[1] + y[6] + y[7] + pow(y[8], 1.5));
        fvec[    2] = constantPlayer[2] - (y[2] + pow(y[6], 2) + pow(y[7], 2) + pow(y[8], 3));
        fvec[    3] = constantPlayer[3] - (y[3] + pow(y[7], 1.5) + y[10]);
                --- omission ---
        fvec[ 12] = -kpd[1]*y[1] - kpd[2]*y[6] - kpd[3]*y[7] - 1.5*kpd[4]*y[8];
        fvec[ 13] = kp[1]*y[11] - kpd[5]*y[5] - kpd[6]*y[9] - kpd[7]*y[10];
}

void usr_fjac(double fjac[][MAX_VAR_NUM], double y[], double Gene, Parameter *p)
{
        double *constantPlayer          = p->val[1];
        double *Kb              = p->val[2];
        double *kp              = p->val[3];
        double *kpd             = p->val[4];
        double *km              = p->val[5];
        double *kmd             = p->val[6];

        fjac[    1][    1] = -1;
        fjac[    1][    6] = -1;
                --- omission ---
        fjac[ 13][    9] = -kpd[6];
        fjac[ 13][ 10] = -kpd[7];
        fjac[ 13][ 11] = kp[1];
}
```

Equations are indexed in the sequential order of algebraic and differential equations. Variables are marked by "y". The fjac is described only when its value is nonzero.

3.2.2 Parameter file

The checkdae module outputs the parameter file for setting initial values and kinetic parameters. The line up of PARAM_NAME corresponds to the numerical index order of the parameters in the user function. When users gave

values to parameters and initial concentrations in the chekdae file, the values appeared at the corresponding order of PARAM_NAME.

```
##### Model ################
# TITLE:test
# INFO:test
# CONVERSION TYPE
#         GENE-PROTEIN:TPP_RAPID
#         METABOLIC:NONE
N_VAR     ;    13; # num of variables(all)
N_ALGEBR;    10; # num of variables(Algebraic Eq)

#Y_START ; index; initial_value; tag #comment
Y_START ;      1;    1.2000e+00;    A.cyt       #
Y_START ;      2;    2.1000e+00;    B.cyt       #
Y_START ;      3;    3.1000e+00;    C(pro).cyt              #
                --- omission ---
Y_START ;    11;    1.0000e-01;    mRNA(test).cyt       #
Y_START ;    12;    1.7650e+00;    TA.cyt    #
Y_START ;    13;    6.0000e-02;    Ttest.cyt   #

#PARAM;name;index;val;val_start;num_survey;D/R/S;change-val;GA_start;GA_end;tag #comment
PARAM      ;constantPlayer;      1;    1.2000e+00;;0;D;;;; G(test).cyt             #
PARAM      ;constantPlayer;      2;    3.2300e+00;;0;D;;;; TB.cyt    #
                --- omission ---
PARAM      ;kmd           ;      1;      ;;0;D;;;; decomposition_rate_constant_mRNA(test).cyt           #

#T_EVENT ; name; index; time; value; #comment


######## Don't edit the following lines. ######
PARAM_NAME ;constantPlayer    #
PARAM_NAME ;Kb                      #
PARAM_NAME ;kp                     #
PARAM_NAME ;kpd                    #
PARAM_NAME ;km                      #
PARAM_NAME ;kmd                     #
```

The checkdae module also implements the function for surveying kinetic parameters to investigate the behavior of a model over a specific range of the values of some parameters. This allows users to effectively study the dependency of the model's behavior on the kinetic parameters, and is therefore a very effective means of forecasting the effects of parameter perturbations.

Details of the #PARAM line are explained as follows,

name: the name of the parameter

index: the index of the parameter

val: the value of the parameter

val_start: the starting value of the parameters, which must be set as the same value as "val" when the parameter survey is carried out.

num_survey: the number of surveyed parameters. When the parameter survey is not performed, set one (1).
D/R/S: The option for the parameter survey. "D" indicates arithmetic series, "R" indicates geometric series, and "S "means the parameter search by GAs or RS (Details are written elsewhere).
change_val: changing value.
GA_start; The starting value of the explored variable region, which is employed for search for GAs or RS.
GA_end; The ending value of the explored variable region, which is employed for search for GAs or RS.

For example, when users change the parameter kx[3] as 1, 2, 3, and 4 in the arithmetic series, the line of #PARAM is provided by:

```
PARAM    ;kx;    3;   1;   1 ;   4;   D;   1;   ;   ;   tag  #
```

When users change the parameter kx[3] as 1, 2, 4, and 8 in the geometric series, the line of #PARAM is provided by:

```
PARAM    ;kx;    3;   1;   1 ;   4;   R;   2;   ;   ;   tag  #
```

3.2.3 Temporary mathematical expression file

When temporary mathematical equations such as a flux and total enzyme are defined in the checkdae file, the corresponding user function is generated automatically. The temporary expressions appear in the checkdae file, when GMA or MM conversion is selected. Mathematical equations for fluxes are shown as follows.

```
/*
 * TITLE:test
 * INFO:test
 * CONVERSION TYPE
 *        GENE-PROTEIN:TPP_RAPID
 *        METABOLIC:NONE
 */
#include "life.h"

void usr_flux(double total[], double flux[], double y[], double Gene, Parameter *p)
{
   double *constantPlayer          = p->val[1];
   double *Kb              = p->val[2];
   double *kp              = p->val[3];
   double *kpd             = p->val[4];
   double *km              = p->val[5];
   double *kmd             = p->val[6];
   double *Kg              = p->val[7];
   double *f = p->val[8];

   flux[    1] = Kg[1]*pow(y[1], f[1]); /* flux_A.cyt_to_B.cyt */

   /* No Total expression. */
}
```

3.2.4 S-system coefficient file

The S-system coefficient file is used for calculating the coefficients of S-system at the steady state, which are sent

to the program for the sensitivity and stability analysis of S-system.

```
/*
  * TITLE:Glycolysis.plc
  * INFO:Glycolytic-Glycogenolytic Pathway Model in Chapter 11 PLUS
  */

#include "life.h"

void get_GHdata(double **gd, double **gi, double **hd, double **hi,
   double *v_plus, double *v_minus, double *y, Parameter *p)
{
   double *constantPlayer            = p->val[1];

   v_plus [    1] = 0.0778843*pow(constantPlayer[1], 0.66)*constantPlayer[3];
   gi[    1][    1] = 0.66*pow(constantPlayer[1], -0.34)*0.0778843*constantPlayer[3];
   gi[    1][    3] = 0.0778843*pow(constantPlayer[1], 0.66);
   v_minus[    1] = 1.06271*pow(y[1], 1.53)*pow(y[2], (-0.59))*constantPlayer[4];
   hd[    1][    1] = 1.53*pow(y[1], 0.53)*1.06271*pow(y[2], (-0.59))*constantPlayer[4];
   hd[    1][    2] = (-0.59)*pow(y[2], (-0.59) - 1)*1.06271*pow(y[1], 1.53)*constantPlayer[4];
   hi[    1][    4] = 1.06271*pow(y[1], 1.53)*pow(y[2], (-0.59));
            --- omission ---
   gi[    3][    5] = 0.000793456*pow(y[2], 3.97)*pow(y[3], (-3.06));
   v_minus[    3] = 1.05881*pow(y[3], 0.3)*constantPlayer[6];
   hd[    3][    3] = 0.3*pow(y[3], -0.7)*1.05881*constantPlayer[6];
   hi[    3][    6] = 1.05881*pow(y[3], 0.3);
}

void get_GHnum(int *n_var, int *n_const)
{
   *n_var = 3;
   *n_const = 7;
}
```

4 Data structure

4.1 Main structures

```
EqSystem_T
  |
  +-int    n_var                 total number of variables = n_varX + n_varY
  +-int    n_varX                number of x variables
  +-int    n_varY                number of y variables
  +-int    n_mid                 number of temporary mathematical expressions
  +-int    n_equation            number of equations (n_varX + n_varY)
  +-int    n_param  number of parameter species (containing constantPlayer)
  |
  +-Variable_T *var(n_var)   variable arrays
  |     |
  |     +-int            index
```

```
|    +-char        *tag                 molecular name
|    +-double      val
|    +-int         f_isX                x or y flags
|    +-int         f_isUsed             flags for use or non-use
|
+-Expression_T *mid(n_mid)              array for temporary mathematical expressions
|    |
|    +-int         f_isXorT             (X/Y or) Total/Flux flag
|    +-char        *name                left hand side equations (identification name)
|    +-char        *expr                expanded equations
|    |
|    +-Node_T      *root                expanded equations in binary trees (list)
|
|
+-Expression_T *equation(n_varX + n_varY)   array for mathematical expressions
|    |
|    +-int         f_isXorT             x or y flags
|    +-char        *name                Y: it is used, X: it is not used.
|    +-char        *expr                expanded equations
|    |
|    +-Node_T      *root                expanded equations in binary trees (list)
|
+-ParamBase_T *param_base(n_param)           array for parameter species
     |
     +-char        *label                 ka, kb, etc
     +-int         max_index              maximum index
     +-int         f_isUsed               flags for use or non-use
     |
     +-Parameter_T *param(max_index)      array for each parameter species
          |
          +-char          *tag            content
          +-double        val
          +-int           f_isUsed        flags for use or non-use
```

4.2 Node_T structure

In pdiff.h, the definition of the Node_T structure, which is key to mathematical derivations, is provided as follows.

```
typedef struct _node {
   char              op;
   int               rank;
   int               f_bracket;
   int               f_chunk;
   struct _node      *parent;
   struct _node      *childL;
   char              *charL;
   int               rank_charL;
   struct _node      *childR;
   char              *charR;
   int               rank_charR;
   char              *expr;
   int               rank_expr;
   char              *diff;
   int               rank_diff;
} Node_T;
```

This structure is generated with respect to an operator (+ , -, *, / , or ^). Mathematical equations are expressed by linking the structures in a binary tree. Binominal operations are carried out, but it is possible to define '-' as a single operator. In this case, neither childL nor charL is defined. In the other cases, operands are defined in both the right and the left. For example, the left operand is defined either as charL or childL. If the operand is provided as charL, rank_charL defines the value that identifies either numerical values (R_5VAL) or strings (R_4CHAR), where childL is NULL. Conversely, if the operator is defined as childL, charL is NULL and rank_charL is provided by R_NULL. The right operand is defined in the same manner.

The others are as follows.

char op;
define an operator of the node (+, -, *, / , or ^)

int rank;
define the priority value of operators (R_1PLUS、R_MULT、R_3POW) to the above operator.

int f_bracket;
frag necessary for handling "(" on the way to expansion of equations.

int f_chunk;
flag showing that the operation of the node employs brackets.

struct _node *parent;
pointer to the parent

char *expr;
registering mathematical expressions

int rank_expr;
showing the rank of the above mathematical expressions, which defines R_**.

char *diff;
registering mathematical expressions that are partially differentiated.

int rank_diff;
showing the rank of the above mathematical expressions, which defines R_**.

In order to identify the priority of an operator and the order of operations, rank** is provided as follows.

R_NULL	:not defined
R_1PLUS	:addition or subtraction, or mathematical expression containing them
R_2MULT	:multiplication or division, or mathematical expression containing them but excluding addition and subtraction
R_3POW	:power, or mathematical expression consists of powers.
R_4CHAR	:variables (in mathematical expression)
R_5VAL	:values (in mathematical expression)

5 Procedure

The relationships of functions are described in a tree structure. The functions followed by '…' indicates that they have been already called. The functions preceded by '…' are recursive functions.

```
main
  +-my_init                    setting file names
  +-Read_data                  read data
  |    +-split_dae_file                division of a checkdae file, read the numbers of data
  |    +-allocate_area                 memory allocation for data
  |    +-read_param_base               read parameter labels.
  |    |    +-divide_line                      Strings are sectioned by '=' or ';'
  |    +-read_param              read parameters
  |    |    +-read_param_index         obtain the maximum number of parameters
  |    |    |    +-divide_line ...
  |    |    |    +-Divide_variable             divide variable and parameter strings by '[' and ']'
  |    |    |    +-FindParamBaseAddr           examine if parameter labels are registered
  |    |    +-read_param_data          read parameter data
  |    |         +-divide_line ...
  |    |         +-Divide_variable ...
  |    |         +-FindParamBaseAddr ...
  |    +-read_var                      read variables
  |    |    +-divide_line ...
  |    |    +-Divide_variable ...
  |    +-read_mid_expression  read temporary mathematical expressions
  |         +-fix_equation_line        rearrangement of equations
  |         |    +-Trim_tail                        trim line ends
  |    |    +-divide_line    ...
  |    +-read_equation                 read differential and algebraic equations
  |         +-fix_equation_line    ...
  |         +-divide_line    ...
  |         +-Divide_variable    ...
  +-set_expression   expand mathematical equations in a binary tree.
  |    +-Make_node             expand mathematical equations in a tree structure
  |    |    +-create_node                  create nodes
  |    |    +-get_token                    read mathematical equations
  |    |    +-check_token                  check the grammar of mathematical equations
  |    |    +-add_emptyChild               create blank child nodes
  |    |    |    +-create_node ...
  |    |    +-seek_bracket                 search a node whose flag is '('
  |    |    |    +-... seek_bracket ...
  |    |    +-del_node                     delete nodes
  |    |    +-set_oprator                  set operator information
  |    |    +-insert_childNode        insert child nodes
  |    |    |    +-create_node ...
  |    |    +-seek_same                    search equivalent or less prior nodes
  |    |    |    +-... seek_same ...
  |    |    +-insert_parentNode       insert a parent node
  |    |    |    +-create_node ...
  |    |    +-set_operand                  a set of operands
  |    +-Set_expr                      reconstruct mathematical equations based on a tree structure
  |    |    +-... Set_expr ...
  |    |    +-del_bracket                  delete brackets
  |    |    +-add_bracket                  add brackets
  |    +-check_node            check nodes recursively
  |         +-... check_node ...
  |         +-replace_variable        replace with numerical indexes
  |         |    +-Divide_variable ...
  |         |    +-getVarIndex          obtain numerical indexes for variable names
  |         +-check_detail                 check nodes
  |              +-Divide_variable ...
```

```
|               +-findMidAddr           examine if temporary mathematical expressions are registered
|               +-expand_mid                    expand temporary mathematical expressions
|               |    +-Make_node ...
|               |    +-Seek_root                         obtain the address of a root node
|               |    |    +-... Seek_root ...
|               |    +-Set_expr ...
|               +-findVarAddr                   examine if variables are registered
|               +-FindParamBaseAddr ...
|               +-setUsedParam                           Add flags to employed parameters
+-set_param_flag  examine if parameter labels are used or not
+-set_variable_index        assign numerical indexes to variables
|    +-getDiffEqAddr                obtain the address for registering differential equations
+-set_numerical_index       replace with numerical indexes
|    +-check_node ...
|    +-Set_expr ...
+-Write_usrfunc             create user functions for simulator
|    +-write_header                 write header information
|    +-write_param_label            write information that relates parameters
|    +-Make_diff                    symbolic partial differentiation
|         +-... Make_diff ...
|         +-diff_add                          partial differentiation for addition
|         |    +-del_bracket ...
|         +-diff_subtract                     partial differentiation for subtraction
|         |    +-del_bracket ...
|         +-diff_multiply                     partial differentiation for multiplication
|         |    +-multiply                          multiplication
|         |         +-del_bracket ...
|         +-diff_divide                       partial differentiation for division
|         |    +-multiply ...
|         +-diff_power                        partial differentiation for power
|         |    +-multiply ...
|         |    +-del_bracket ...
|         +-add_bracket ...
+-Write_paramFile            create parameter files
|    +-write_header ...
+-Write_flux                create temporary mathematical expression files
|    +-write_header ...
|    +-write_param_label ...
|
+-Set_Ssystem               write S-system user function files
       +-write_GEhead               write the header part of the file that defines g and h
       |    +-Write_header ...
       |    +-Write_param_label ...
       +-make_Sdata                 write the core part of S-system user functions
       |    +-Set_expr ...
       |    +-separate_terms        divide differential equation into V+ and V-
       |    +-get_preGH                    writhe g, h, V+, and V-
       |    |    +-Del_tree                       delete unnecessary expansion trees
       |    |    |    +-... Del_tree ...
       |    |    +-Make_node ...
       |    |    +-Set_expr ...
       |    |    +-Make_diff ...
       |    +-write_Sfvec                  write fvec in a temporary file
       |    +-write_Sfjac                  write fjac in a temporary file
       +-write_GEtail               write the end part of the file that defines g and h
       +-write_S_usrfunc            write S-system user function file
            +-Write_header ...
```

+-write_S_funcs write S-system differential equations

Main procedures are explained as follows.

5.1 Input procedure (Read_data)

5.1.1 Reading checkdae files: data split and data number acquisition (split_dae_file)

A checkdae file is read as input data, and divided according to the identification lines. The divided data are sent to a temporary file. The numbers of various data are obtained by counting semicolons. Each valid data has one semicolon.

5.1.2 Index check and memory allocation (allocate_area).

After checking the numbers the have been obtained by split_dae_file, the memory necessary for structure arrays for the employed parameters are allocated.

5.1.3 Reading parameter labels (read_param_base)

After reading the parameter labels from the temporary file generated by split_dae_file, the parameter labels and constantPlayer are registered in ParamBase_T.

5.1.4 Reading parameters (read_param)

The maximum number of each parameter species is read from the temporary file generated by split_dae_file, and the memory necessary for the parameter arrays is allocated (by read_param_index). Then the data is read again and registered in Parameter_T (by read_param_data).

5.1.5 Reading variables (read_var)

The variables are read from the temporary file generated by split_dae_file, and registered in Variable_T. The correctness for the assigned indexes is checked.

5.1.6 Reading temporary mathematical expressions (read_mid)

The temporary mathematical expressions are read from the temporary file generated by split_dae_file, and remade by fix_equation_line into another temporary file. The remade data are read, and registered in Expression_T.

5.1.7 Reading differential equations and algebraic equations (read_equation)

The files for differential equations and algebraic equations that have been generated by split_dae_file are rewritten by fix_equation_line in other temporary files. From which, equation data are read and registered in Expression_T. In processing algebraic equations, the right-hand equations are moved to the left. The resultant left-had side is registered as expr. In processing differential equations, the right-hand side is registered in expr. The index of the left-hand side equation is registered as name.

5.2 Expansion of mathematical equations in a tree structure (set_expression、Make_node)

The temporary mathematical expressions, differential equations, and algebraic equations, which are registered as expr in Expression_T, are expanded in a tree structure by Make_node, where Node_T is employed as nodes. In Make_node, the root node is made as a current node. The strings of mathematical equations are sectioned by an operator, an operand, or brackets (by the get_token), and their grammar is checked (by check_token). According to the data, a new node is linked to the current node as a parent or a child. This process is iterated until the whole equations are read. Consequently, expr registers the expanded mathematical equations in a node tree. After expansion, mathematical expressions are reconstructed based on the generated tree (by Set_expr function). Important functions in Make_node are follows.

5.2.1 Reading mathematical expression (get_token)

Strings of mathematical expressions are read by a character, grouped by an operator, an operand, or brackets, and registered in the structure of Token_T. Token_T is as follows:

```
typedef struct _token {
    char    op;     operator or parentheses: '+ ', '(', variable: ’a’, value: ‘v’
    int     rank;   operator:R_PLUS、R_MULT or R_POW. others: R_NULL。
    char    *expr;  pointers to strings of variables and values
    int     bra;    counter for '(' (check_token)
    int     cket;   counter for ')' (check_token)
} Token_T;
```

The index within [] is recognized as characters. All the blanks are eliminated during pursing.

5.2.2 Checking mathematical expression (check_token)

The correctness of the data read by get_token and the balance of brackets are checked.

5.2.3 Adding a blank child node (add_emptyChild)

If the operator of the current node is set, a blank child node is added to the right. Otherwise, it is added to the left.

5.2.4 Inserting a child node (insert_childNode)

A new child node is inserted as the right hand of the current node. The new node receives the right-hand information of the parent node as the left-hand information.

5.2.5 Inserting a parent node (insert_parentNode)

A new node is inserted as the parent node of the current node. The current node is the right-hand child node of the new node.

5.2.6 Deleting nodes (del_node)

It the extra number of brackets are found, the unnecessary nodes are eliminated.

5.2.7 Searching equivalent or less prior nodes (seek_same)

When the operator that has been read shows a high priority to the current node, it is defined as a child node of the current node. However, when the operator shows a less priority or equivalence to the current, the nodes, whose priority is equivalent or less than the operator, whose flag is '(', or whose operator is not defined, are searched in the direction of the root, placing the operator node in the correct position. If the node is not found, return the root.

5.2.8 Reconstructing mathematical expressions (Set_expr)

Mathematical expressions are reconstructed recursively from the expanded binary tree by Make_node. Numerical values are calculated when they appear.

5.3 Checking variables (check_node、check_detail)

All the variables, parameters, and temporary mathematical expressions that are defined in cahrL and charR of each node of a binary tree are recursively checked (by check_node). If there are non-defined values, the program will stop and display "error". In the case of variables and parameters, the flag of f_isUsed is put up. In the case of temporary mathematical expressions, they are expanded in a tree by expand_mid.

5.4 Expanding temporary mathematical expressions (expand_mid)

When a temporary mathematical expression is contained in a mathematical equation, the expression is expanded in a binary tree by Make_node. The generated tree is connected to the child node of the mathematical equation. Finally, the bound expressions are reconstructed by Set_expr.

5.5 Assigning an index to a variable (set_variable_index)

Integer consecutive numbers are assigned to the employed variables.

5.6 Converting variables (set_numerical_index, check_node, replace_variable)

After searching the nodes that set variables (by check_node), the name-indexed variables are replaced by integer indexes (by replace_variable). After replacement, the resultant expressions are reconstructed by Set_expr.

5.7 Symbolic partial differentiation (Make_diff)

Each term for the expanded mathematical equations is partially differentiated from the root node. The process is carried out recursively. The partial differentiation is applicable to the five operators:

addition	:diff_add
subtraction	:diff_subtract
multiplication	:diff_multiply
division	:diff_divide
power	:diff_power

The Diff_T structure is employed to send and receive data.

```
typedef struct _diff {
   int         rank_exprL;
   int         rank_exprR;
   int         rank_diffL;
   int         rank_diffR;
   char        exprL[BUFFSIZE];
   char        exprR[BUFFSIZE];
   char        diffL[BUFFSIZE];
   char        diffR[BUFFSIZE];
} Diff_T;
```

Make_diff is called to make Jacobian, which is defined as a user function.

5.8 Output process

5.8.1 Output of user functions for simulators (Write_userfunc)

There are two types of user functions: usr_fvec and user_fjac. The usr_fvec consists of algebraic and differential equations that have been expanded and replaced with numerical indexes. The user_fjac is the Jacobian, where the variable is partially differentiated with respect to each variable. Note that the fjac whose value is zero is excluded. First, the header part that has been obtained by split_dae_file is described as comments in the C language format, and then usr_fvec and usr_fjac are output. In the user functions, the employed parameters are linked to the parameter structure (write_param_label). In user_fvec, expr in the Expression_T structure that stores mathematical expressions is output sequentially. In usr_fjac, Make_diff is applied to a differential equation with respect to each variable, and the resultant Jacobian is output in order.

5.8.2 Output of parameter files (Write_paramFile)

First, the header part that has been obtained by split_dae_file is described as comments. Next, the initial value setting part, parameter-setting part, and the data that relate them to the parameter labels are described. When the initial values and parameter values are not set, blanks are output.

5.8.3 Output of temporary mathematical expression obtaining file　(Write_flux)

Instead of usr_fvec, usr_flux is output that is the function to defines temporary mathematical expression The total concentration of enzymes is defined as the array "total", that of fluxes as array of "flux". The original temporary mathematical expressions are output as comments on the right-hand side of the equations.

5.9 S-system conversion（Set_Ssystem）

The S-system conversion is carried out with respect to each differential equation by make_Sdata, as follows:

1. Differntial equations are redescribed by Set_expr.
2. All the terms of the right-hand side are divided into V+ and V-, respectively, by separate_terms.
3. V+ and V- are output by get_preGH. Made_diff generates and writes $\frac{\partial V_+}{\partial X}$ and. $\frac{\partial V_-}{\partial X}$ The partial differential formula are described separately with respect to the dependent or independent variables.
4. write_Sfvec writes the differential equations of S-system with respsect to the dependent variables.
5. write_Sfjac generate and write the Jacobian.
6. The resultant expression are output as the S-system coefficient file and S-system user function file.

3.2.5 S-system user function file

The checkdae module converts ordinary differential equations into S-system, generating the user function of S-system, as follows.

```
/*
  * TITLE:Glycolysis.plc
  * INFO:Glycolytic-Glycogenolytic Pathway Model in Chapter 11 PLUS
  */
/*

#include "life.h"

void usr_fvec(double fvec[], double y[], double Gene, Parameter *p)
{
    double *constantPlayer            = p->val[1];
    double *kplus          = p->val[2];
    double *kminus         = p->val[3];
    double *g = p->val[4];
    double *h = p->val[5];

    fvec[    1] = kplus[1]*pow(constantPlayer[1],g[1])*pow(constantPlayer[3],g[2])
                    - kminus[1]*pow(y[1],h[1])*pow(y[2],h[2])*pow(constantPlayer[4],h[3]);
               --- omission ---
    fvec[    3] = kplus[3]*pow(y[2],g[8])*pow(y[3],g[9])*pow(constantPlayer[5],g[10])
                   - kminus[3]*pow(y[3],h[7])*pow(constantPlayer[6],h[8]);
}

void usr_fjac(double fjac[][MAX_VAR_NUM], double y[], double Gene, Parameter *p)
{
    double *constantPlayer            = p->val[1];
    double *kplus          = p->val[2];
    double *kminus         = p->val[3];
    double *g = p->val[4];
    double *h = p->val[5];

    fjac[    1][    1] =    - kminus[1]*h[1]*pow(y[1],h[1]-1)*pow(y[2],h[2])*pow(constantPlayer[4],h[3]);
    fjac[    1][    2] =    - kminus[1]*pow(y[1],h[1])*h[2]*pow(y[2],h[2]-1)*pow(constantPlayer[4],h[3]);
               --- omission ---
    fjac[    2][    3] =    - kminus[2]*pow(y[2],h[4])*h[5]*pow(y[3],h[5]-1)*pow(constantPlayer[5],h[6]);
    fjac[    3][    2] = kplus[3]*g[8]*pow(y[2],g[8]-1)*pow(y[3],g[9])*pow(constantPlayer[5],g[10]);
```

```
    fjac[    3][    3] = kplus[3]*pow(y[2],g[8])*g[9]*pow(y[3],g[9]-1)*pow(constantPlayer[5],g[10])
                - kminus[3]*h[7]*pow(y[3],h[7]-1)*pow(constantPlayer[6],h[8]);
}
```

C. SOLVER

1. Introduction

Two distinct types of simulation can be carried out: (i) time course simulation, where the values of variables are determined as a time series; (ii) steady state analysis, where the values of variables are determined for a state in which metabolite concentration does not change. For differential equations, the CADLIVE Simulator implements the Runge-Kutta method, the step-adaptive Runge-Kutta method, and the NDF that can be applied to highly stiff differential equations. The NDF has been programmed based on ode15s of MATLAB. The Runge-Kutta or the step-adaptive Runge-Kutta method is employed for non-stiff differential equations. The Newton-Raphson algorithm is employed to solve algebraic equations. The combination of the Newton-Raphson algorithm with the Runge-Kutta or with the step-adaptive Runge-Kutta method, or the NDF solves DAEs. The solver not only simulates dynamic behaviors of biochemical networks, but also solves the steady state solutions by applying the quasi-steady state approximation to all the differential equations, resulting in generating algebraic equations. The Newton-Raphson method solves such algebraic equations to obtain the steady state concentrations. In order to enhance the calculation rate, the CADLIVE Simulator employs the Message-Passing Interface (MPI) for parallel calculation.

2. Analysis types

As "Analsys type", users can choose either "Dynamic" or "Steady-state". "Dynamic" simulates the time evolution of the concentrations by calculating DAEs, and "Steady-state" calculates the concentrations at steady state by solving algebraic equations. The checkbox of "Parameter survey" determines if the simulator surveys the parameter space. Checking the checkbox of "Parallel calculation" carries out parallel calculation that employs the Message Passing Interface (MPI). Notice that the checkbox of "Parallel calculation" cannot be selected prior to checking the parameter survey.

3. Input for control data

The control parameters for the solvers are described as Table C1-C3. How to provide the value is shown in the examples (CADLIVE web page).

Table C1 Control parameters for differential equations

Solver Type	
Runge-Kutta	Constructed based on "Numerical Recipes in C" (Cambridge University Press)
Runge-Kutta (Adaptive step-size)	
NDF	Constructed based on ode15s of MATLAB
Set time span and time step-size.	
Start time	The start time of simulation
End time	The end time of simulation
(Initial)time step-size	Initial time step size
Monitoring interval	The simulated time course is picked up by the monitoring interval value.

Table C2 Control parameters for algebraic equations

Solver type	
Newton-Raphson Method	Constructed based on "Numerical Recipes in C" (Cambridge University Press)
Maximum trial times	The number of iterated calculations
Tolerance for convergence of functions	Convergence condition
Tolerance for convergence of variables	Convergence condition
Ratio of changing parameters	When the value for kinetic parameter is changed, the value should be varied by the ratio of changing parameters. Because the great change in the kinetic parameter sometimes causes the calculation to fail.
change width calc. sensitivity (STD).	The ratio of perturbation that is given to parameters is set as STD. This is used for analyzing the sensitivity of models.

Table C3 Other parameters

Other	
G-value	Always 1.0
Y default value	default values for y (molecular concentrations)

D. MERGEPARAM

1. Objectives

It is laborious and time-consuming to set many parameters manually. The mergeParams module helps users setting parameters by copying or merging the existing data, which greatly reduces such laborious parameter setting. The mergeParams module updates the current parameter file by a new parameter file.

2. Function

The mergeParams module updates the values for the variables, constants, and parameters of the current parameter file by a new one according to their tags (#comment). The tags are added to all the variables, constants and parameters in order to distinguish them. Comparing the new parameter values with the current parameter values, the current parameters with the same tag as the new ones are updated, if necessary. When some values of the current parameters have been already set, users can chose two types:

1) Only empty parameters are updated.

2) The values for all the current parameters are updated.

3. Input/output specification

The mergeParams module reads a new parameter file and the current file, and updates the current parameters by the new ones. Both parameter files are text files, the lines to be merged are:

- Y_START : the line starting from "Y_START". the initial values of variables.
- PARAM : the line starting from "PARAM". the values of kinetic parameters.

This module processes only the lines to be merged, the other lines never be changed.

3.1 Y_START

Y_START ; index; initial_value; tag #comment

Each field is distinguished with semicolon";". The mergeParams skips the region preceded by "#". The tag "#comment" distinguishes the parameters to be merged, and "initial_value" is merged.

3.2 PARAM

PARAM;name;index;val;val_start;num_survey;D/R;change-val;GA_start;GA_end;tag #comment

Each field is distinguished with semicolon";". The mergeParams skips the region preceded by "#". The tag

"#comment" distinguishes the parameters to be merged, and "val" is merged.

4. Command

Users use the mergeParams program on command lines as follows.

mergeParams –IMathParam.txt –Rinput.txt [–B]

MathParam.txt is the current parameter file. The space between the file and the option is not allowed.

- MathParam.txt input/output : the current parameter file
- input.txt input : the parameter file that updates the current one.

The flag –B indicates that only empty parameters are updated.

No flag indicates that all the current parameters are updated.

5. Error message

"Too many parameters."

"Invalid line:

LINE_TEXT"

"Cannot open FILENAME for reading."

"Cannot read parameters. (file = FILENAME)"

"Cannot open FILENAME for writing."

"Invalid argument !!"

"Too few arguments !!"

"Cannot import files !!"

"Cannot merge parameters !!"

"Cannot export parameters !!"

E. SENSITIVITY AND STABILITY ANALYSIS BY S-SYSTEM

1. Objectives

Sensitivity analysis shows how a biochemical system responds to perturbations or uncertainties. Among many methods for system analysis, the sensitivity and stability analysis is useful for characterizing the robustness of mathematical models at a steady-state level, and allows one to determine which parameters have the most effect on the dynamics, or which factors cause to the system to oscillate. S-system is employed to analyze the various sensitivities and stability of the system at the steady state, because the use of S-system solves them in symbolic form. In this module, ordinary differential equations such as TT, CMA, GMA, and MM are converted into S-system to analyze the sensitivities and stability.

2. Introduction of S-system

We show how S-system is derived from ordinary differential equations, and how the stability and sensitivity are defined at the steady state.

2.1 S-system conversion

Ordinary differential equations are divided into the positive terms and negative terms:

$$\frac{dX_i}{dt} = V_i^+ - V_i^- , \qquad (i = 1,....,n,....n+m) \qquad (E1),$$

where $\mathbf{V_i^+}$ is the sum of positive terms, and $\mathbf{V_i^-}$ is the sum of negative terms. Generally, S-system is given by:

$$\frac{dX_i}{dt} = \alpha_i \prod_{j=1}^{n+m} X_j^{g_{ij}} - \beta_i \prod_{j=1}^{n+m} X_j^{h_{ij}} \qquad (E2),$$

where $X_1,\ldots, X_n$ are the dependent variables, whose values vary with time, and $X_{n+1},\ldots,X_{n+m}$ are the independent variables, whose values are fixed as constants. When the concentrations of the dependent variables are given at the steady state, the coefficients (g_{ij}, h_{ij}, α_i, β_i) of S-system are solved from Eq. (E2) in symbolic form:

$$g_{ij} = \frac{\partial V_i^+}{\partial X_j}\frac{X_j}{V_i^+} \qquad (E3)$$

$$h_{ij} = \frac{\partial V_i^-}{\partial X_j}\frac{X_j}{V_i^-} \qquad (E4)$$

$$\alpha_i = \frac{V_i^+}{\prod_{j=1}^{n+m} X_j^{g_{ij}}} \qquad (E5)$$

$$\beta_i = \frac{V_i^-}{\prod_{j=1}^{n+m} X_j^{h_{ij}}} \qquad \text{(E6)},$$

where g_{ij} and h_{ij} are the kinetic order and α_i and β_i are the rate constants. At the steady state:

$$\frac{dX_i}{dt} = 0,$$

we can take logarithms on both sides of Eq. (E2), the equation reduces to a linear equation:

$$\ln \alpha_i + \sum_{j=1}^{n+m} g_{ij} \ln X_j = \ln \beta_i + \sum_{j=1}^{n+m} h_{ij} \ln X_j \quad ,$$

or

$$\sum_{j=1}^{n} (g_{ij} - h_{ij}) \ln X_j = \ln \frac{\beta_i}{\alpha_i} - \sum_{j=n+1}^{n+m} (g_{ij} - h_{ij}) \ln X_j \qquad \text{(E7)}.$$

Substituting $\ln X_j = y_j$, $\ln \frac{\beta_i}{\alpha_i} = b_i$. The equations (E7) is changed to the vector-matrix expression:

$$(\mathbf{G_D} - \mathbf{H_D})\vec{\mathbf{y}}_\mathbf{D} = \vec{\mathbf{b}} - (\mathbf{G_I} - \mathbf{H_I})\vec{\mathbf{y}}_\mathbf{I} \qquad \text{(E8)}.$$

We substitute the matrixes as follows,

$$\mathbf{G_D} - \mathbf{H_D} = \mathbf{A_D}\ , \mathbf{G_I} - \mathbf{H_I} = \mathbf{A_I} \qquad \text{(E9)}$$

to obtain the equation:

$$A_D \vec{y}_D = \vec{b} - A_I \vec{y}_I \qquad \text{(E10)},$$

where $\mathbf{A_D}$ is the coefficient matrix of the kinetic order regarding dependent variables, $\mathbf{A_I}$ is the coefficient matrix of the kinetic order regarding independent variables, $\vec{\mathbf{y}}_\mathbf{D}$ is the solution vector of dependent variables, $\vec{\mathbf{y}}_\mathbf{I}$ is the vector of independent variables, and $\vec{\mathbf{b}}$ is the vector regarding the rate constants. Consequently, the vector of the dependent variables is symbolically solved as:

$$\vec{y}_D = A_D^{-1}\vec{b} - A_D^{-1} A_I \vec{y}_I \qquad \text{(E11)}.$$

2.2 Sensitivity analysis

We define five types of gains and sensitivities. Here we explain the mathematical definition of them.

2.2.1. Logarithmic gain of a metabolite

In order to answer the question of how a relative change in an independent variable affect the steady-state concentration of a metabolite, the logarithmic gain of a metabolite $\mathbf{L}(\vec{\mathbf{X}}_\mathbf{D}, \vec{\mathbf{X}}_\mathbf{I})$ is defined in terms of the coefficient matrixes $\mathbf{A}_\mathbf{D}^{-1}$ and $\mathbf{A_I}$ as:

$$\mathbf{L}(\vec{\mathbf{X}}_\mathbf{D}, \vec{\mathbf{X}}_\mathbf{I}) = -\mathbf{A}_\mathbf{D}^{-1}\mathbf{A_I} \qquad \text{(E12)}.$$

2.2.2. Logarithmic gain of a flux

In order to answer the question of how a relative change in a flux affect the steady-state concentration of a metabolite, the logarithmic gain of a flux $\mathbf{L}(\vec{\mathbf{V}}_\mathbf{D}, \vec{\mathbf{X}}_\mathbf{I})$ is defined in terms of the coefficient matrices $\mathbf{A}_\mathbf{D}^{-1}$ and $\mathbf{A}_\mathbf{I}$ as:

$$L(\vec{V}^+, \vec{X}_I) = G_I + G_D L(\vec{X}_D, \vec{X}_I) \qquad \text{(E13).}$$

2.2.3. Sensitivity with respect to a rate constant

In order to answer the question of how a relative change in a rate constant affect the steady state concentration of a metabolite, we solve the sensitivity with respect to rate constants, which is defined as:

$$\mathbf{S}(\mathbf{X_i}, \alpha_\mathbf{j}) = -\frac{\partial \ln \mathbf{X_i}}{\partial \ln \alpha_\mathbf{j}} \qquad \text{(E14),}$$

and

$$\mathbf{S}(\mathbf{X_i}, \beta_\mathbf{j}) = -\frac{\partial \ln \mathbf{X_i}}{\partial \ln \beta_\mathbf{j}} \qquad \text{(E15).}$$

These sensitivities are expressed using vectors and matrixes as:

$$S(\vec{X}_D, \vec{\alpha}) = -A_D^{-1} \qquad \text{(E16),}$$

$$S(\vec{X}_D, \vec{\beta}) = A_D^{-1} \qquad \text{(E17).}$$

2.2.4. Sensitivity of dependent variables with respect to a kinetic order

In order to answer the question of how a relative change in a kinetic order affect the steady-state concentration of a metabolite, we define the sensitivity with respect to the kinetic order as follows,

$$\mathbf{S}(\mathbf{X_i}, \mathbf{g_{ij}}) = -\frac{\partial \ln \mathbf{X_i}}{\partial \ln \mathbf{g_{ij}}} \qquad \text{(E18),}$$

$$\mathbf{S}(\mathbf{X_i}, \mathbf{h_{ij}}) = -\frac{\partial \ln \mathbf{X_i}}{\partial \ln \mathbf{h_{ij}}} \qquad \text{(E19).}$$

The partial derivatives of Eq. (E8) regarding g_{ij} are provided by:

$$\frac{\partial A_D}{\partial \mathrm{g_{ij}}} \vec{y}_D + A_D \frac{\partial \vec{y}_D}{\partial \mathrm{g_{ij}}} = 0, \qquad \text{(E20)}$$

or

$$\frac{\partial \vec{y}_D}{\partial \mathrm{g_{ij}}} = -A_D^{-1} \frac{\partial A_D}{\partial \mathrm{g_{ij}}} \vec{y}_D \quad . \qquad \text{(E21).}$$

Therefore,

$$S(\vec{X}_D, \mathrm{g_{ij}}) = \mathrm{g_{ij}} \frac{\partial \vec{y}_D}{\partial \mathrm{g_{ij}}} = -\mathrm{g_{ij}} A_D^{-1} \frac{\partial A_D}{\partial \mathrm{g_{ij}}} \vec{y}_D \quad \text{(E22)}$$

$$S(\vec{X}_D, \mathrm{h_{ij}}) = \mathrm{h_{ij}} \frac{\partial \vec{y}_D}{\partial \mathrm{h_{ij}}} = -\mathrm{h_{ij}} A_D^{-1} \frac{\partial A_D}{\partial \mathrm{h_{ij}}} \vec{y}_D \quad \text{(E23).}$$

2.2.5. Sensitivity of a flux to a rate constant

In order to answer the question of how a relative change in a rate constant affect the steady-state concentration of a flux, we define the sensitivity with respect to a rate constant. We take logarithms in the first term of the right hand side of S-system:

$$\ln V_l = \ln \alpha_l + \sum_{k=1}^{n+m} g_{lk} \ln X_k \quad \text{(E24)}$$

Here, we differentiate to both side regarding $\ln \alpha_{\mathrm{i}}$ to obtain:

$$S(V_l, \alpha_{\mathrm{i}}) = \delta_{\mathrm{i}l} + \sum_{k=1}^{n} g_{lk} S(X_k, \alpha_{\mathrm{i}}) \quad \text{(E25)}$$

In analogy, partial derivatives of Eq.(E25) with respect to $\ln \beta_{\mathrm{i}}$ give:

$$S(V_l, \beta_{\mathrm{i}}) = \sum_{k=1}^{n} g_{lk} S(X_k, \beta_{\mathrm{i}}) \quad \text{(E26).}$$

2.2.6. Sensitivity of a flux with respect to a kinetic order

In order to answer the question of how a relative change in a kinetic order affect the steady-state concentration of a flux, we define the sensitivity with respect to a kinetic order. The partial derivatives of Eq. (E24) with respect of $\ln \mathrm{g_{ij}}$ are provided as:

$$S(V_l, \mathrm{g_{ij}}) = \delta_{il} g_{ij} \ln X_j + \sum_{k=1}^{n} g_{lk} S(X_k, g_{ij}) \quad \text{(E27).}$$

In analogy, the partial derivatives of Eq. (E24) with respect of $\ln \mathrm{h_{ij}}$ are provided as:

$$S(V_l, \mathrm{h_{ij}}) = \delta_{il} h_{ij} \ln X_j + \sum_{k=1}^{n} h_{lk} S(X_k, h_{ij}) \quad \text{(E28).}$$

2.3 Stability analysis

Defining the equation:

$\delta \mathrm{X_k} = \mathrm{X_k} - \mathrm{X_{kS}}$ ($\mathrm{X_{kS}}$ steady state),

we apply the first order approximation to Eq. (E2) to obtain:

$$\delta \dot{X}_i = \sum_{k=1}^{n} \frac{g_{ik} \alpha_i \prod_{j=1}^{n+m} X_j^{g_{ij}}}{X_k} \Big|_{X_k = X_{kS}} \delta X_k - \sum_{k=1}^{n} \frac{h_{ik} \beta_i \prod_{j=1}^{n+m} X_j^{h_{ij}}}{X_k} \Big|_{X_k = X_{kS}} \delta X_k \quad \text{(E29),}$$

or

$$\delta \dot{X}_i = \sum_{k=1}^{n} \frac{g_{ik} V_{iS}^{+}}{X_{kS}} \delta X_k - \sum_{k=1}^{n} \frac{h_{ik} V_{iS}^{-}}{X_{kS}} \delta X_k \quad \text{(E30),}$$

As $V_{iS}^{+} = V_{iS}^{-}$, we define:

$$f_{ik} = \frac{V_{iS}^{+}}{X_{kS}} = \frac{V_{iS}^{-}}{X_{kS}} \quad \text{(E31)}$$

to deduce to:

$$\delta \dot{X}_i = \sum_{k=1}^{n} (g_{ik} - h_{ik}) f_{ik} \delta X_k \quad \text{(E32).}$$

Stability at the steady state is investigated by solving the eigenvalues of the coefficient matrix consisting of:

$$(g_{ik} - h_{ik}) f_{ik} \quad \text{(E33).}$$

3 Process flow

Figure D1 shows the whole processes of the simulator, where the process flow of S-system mainly consists of three parts:

1) creation of the S-system coefficient file, which consists of the kinetic orders and rate constants that are solved in symbolic form at the steady state,

2) creation of the S-system parameter file (the coefficients and steady-state concentrations) necessary for sensitivity/stability analysis, which is made using the S-system coefficient function and the steady state concentrations.

3) simulation of S-system.

The checkdae module carries out the process of 1), following that the user function for ordinary differential equations (TT, CMA, GMA, MM) are generated. However, the checkdae module does not convert the differential and algebraic equations (DAEs), because they cannot be converted into S-system directly. The checkdae module carries out the analysis of the sensitivities and stability of S-system by linking the S-system coefficient function to the steady-state concentrations. The concentrations at the steady state must be solved before this analysis.

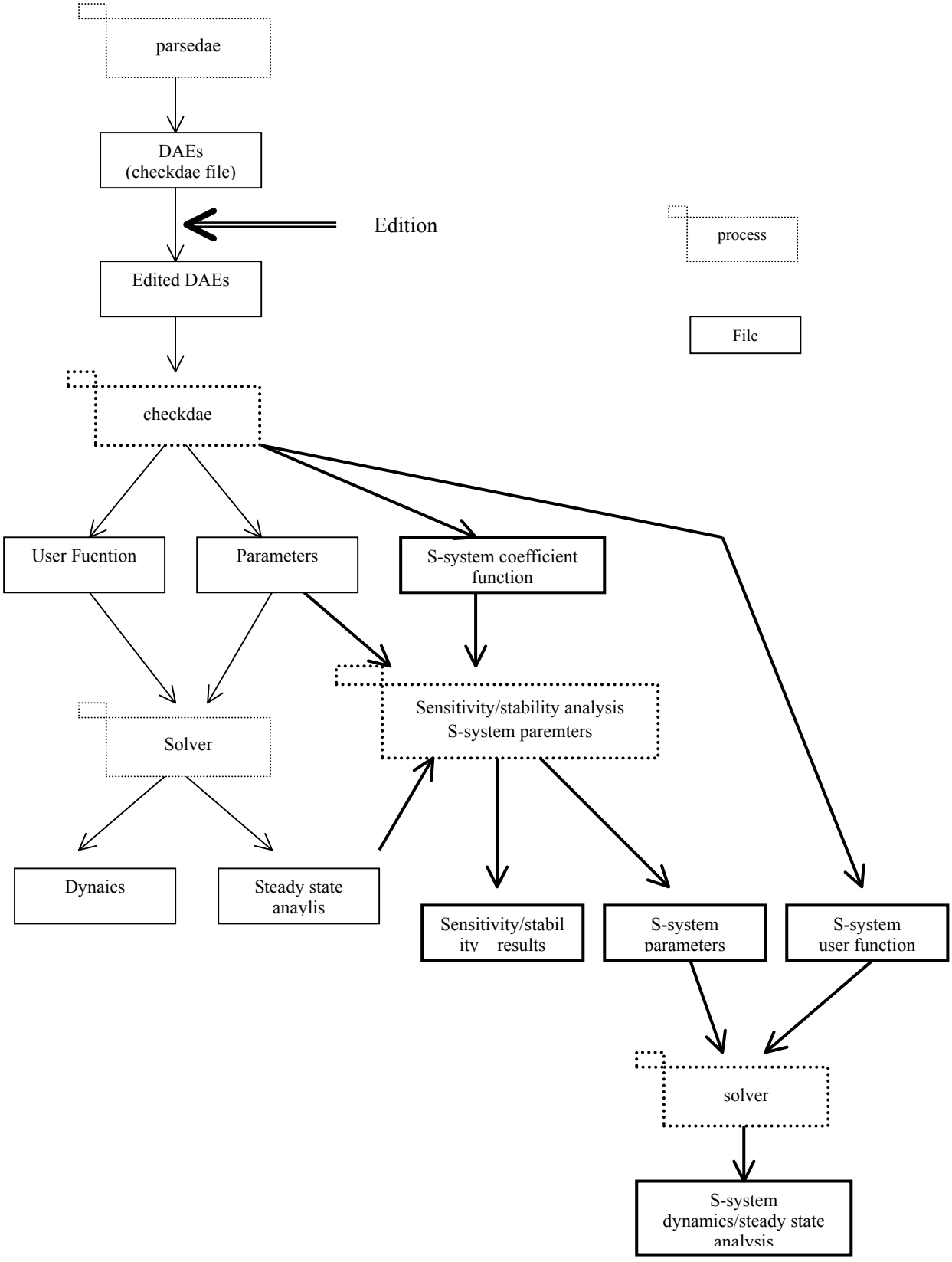


Fig. E1 Process flow for S-system analysis

4 Function

4.1 Creation of S-system parameter file

Following calculating the coefficients of S-system, the S-system parameter file is created. The S-system parameters provide the values of the coefficients at the steady state for the S-system user function.

4.2 Sensitivity analysis

The coefficient matrix of S-system is created to solve the various kinds of the sensitivities.

4.3 Stability analysis

After the coefficient matrix of the system is created, the eigenvalues of the matrix are calculated by the Hessenberg QR method.

5 Input/Output Specification

5.1 Input specification

The input files necessary for calculating the sensitivity and stability are the S-system user function that has been obtained at the steady state, and the S-system parameter file. The S-system parameter file is as follows,

```
##### Model ################
N_VAR     ;      3; # num of variables(all)
N_ALGEBR;      0; # num of variables(all)

#Y_START ; index; initial_value; tag #comment
Y_START ;      1;    6.7000e-02; X1              #
Y_START ;      2;    4.6501e-01; X2              #
Y_START ;      3;    1.5000e-01; X3              #

#PARAM;name;index;val;val_start;num_survey;D/R/S;change-val;GA_min;GA_max;tag #comment
PARAM     ;constantPlayer;      1;    1.0000e+01;;0;D;;;; X4              #
PARAM     ;constantPlayer;      2;    5.0000e+00;;0;D;;;; X5              #
PARAM     ;constantPlayer;      3;    3.0000e+00;;0;D;;;; X6              #
PARAM     ;constantPlayer;      4;    4.0000e+01;;0;D;;;; X7              #
PARAM     ;constantPlayer;      5;    1.3600e+02;;0;D;;;; X8              #
PARAM     ;constantPlayer;      6;    2.8600e+00;;0;D;;;; X9              #
PARAM     ;constantPlayer;      7;    4.0000e+00;;0;D;;;; X10             #
PARAM     ;kplus                ;      1;    7.7884e-02;;0;D;;;; alpha_1     #
PARAM     ;kplus                ;      2;    5.8501e-01;;0;D;;;; alpha_2     #
PARAM     ;kplus                ;      3;    7.9346e-04;;0;D;;;; alpha_3     #
PARAM     ;kminus               ;      1;    1.0627e+00;;0;D;;;; beta_1      #
PARAM     ;kminus               ;      2;    7.9346e-04;;0;D;;;; beta_2      #
PARAM     ;kminus               ;      3;    1.0588e+00;;0;D;;;; beta_3      #
PARAM     ;g                    ;      1;    6.6000e-01;;0;D;;;; gi_1_1      #
PARAM     ;g                    ;      2;    1.0000e+00;;0;D;;;; gi_1_3      #
PARAM     ;g                    ;      3;    9.5000e-01;;0;D;;;; gd_2_1      #
                          ---omission---
PARAM     ;h                    ;      5; -3.0600e+00;;0;D;;;; hd_2_3      #
PARAM     ;h                    ;      6;    1.0000e+00;;0;D;;;; hi_2_5      #
PARAM     ;h                    ;      7;    3.0000e-01;;0;D;;;; hd_3_3      #
PARAM     ;h                    ;      8;    1.0000e+00;;0;D;;;; hi_3_6      #

#T_EVENT ; name; index; time; value; #comment

######## Don't edit the following lines. ######
PARAM_NAME ;constantPlayer    #
PARAM_NAME ;kplus                  #
PARAM_NAME ;kminus                  #
```

```
PARAM_NAME ;g                    #
PARAM_NAME ;h                    #
```

5.2 Output specification

Following the analytical solution, the various sensitivities and stability are provided as follows.

```
##### Fri Mar 28 16:14:40 2003 #####

# Logarithmic Gains and Sensitivities.

# The following data is the things when calculating STD.
##### Fri Mar 28 16:14:39 2003 #####

Solver No.              :            11
Param Survey            :             0
NR_TRIAL                :            20
NR_TOL_F                : 1.000E-15
NR_TOL_X                : 1.000E-15
---omission---
h[ 8]      =    1.0000e+00          #hi_3_6

# Below this line is as a result of sensitivity analysis etc.

$ Fluxes $
index,       y value ,          Flux
      1,     6.700e-02,     1.068e+00, #X1
      2,     4.650e-01,     1.714e+00, #X2
      3,     1.500e-01,     1.714e+00, #X3

$ Eigenvalues $
No.   ,           Real ,     Imaginary
      1,   -2.508e+01,     0.000e+00
      2,   -1.037e+00,     0.000e+00
      3,   -5.281e+01,     0.000e+00

$ Logarythmic Gains of Metabolites $
param/var              ,        y[   1],        y[   2],        y[   3]
constantPlayer[   1],    8.283e-01,    1.029e+00,    1.216e+00
constantPlayer[   2],    3.100e-01,    8.038e-01,    9.497e-01
constantPlayer[   3],    1.255e+00,    1.560e+00,    1.843e+00
constantPlayer[   4], -6.545e-01, -2.298e-03, -2.716e-03
constantPlayer[   5], -8.648e-02, -2.243e-01,    3.264e-02
constantPlayer[   6], -8.821e-01, -2.288e+00, -3.000e+00
constantPlayer[   7],    3.681e-01,    9.545e-01,    1.128e+00

$ Sensitivities of Metabolites with respect to rate constants $
param/var              ,        y[   1],        y[   2],        y[   3]
            alpha[   1],    1.255e+00,    1.560e+00,    1.843e+00
            alpha[   2],    9.686e-01,    2.512e+00,    2.968e+00
            alpha[   3],    8.821e-01,    2.288e+00,    3.000e+00

$ Sensitivities of Metabolites with respect to kinetic orders $
param/var              ,        y[   1],        y[   2],        y[   3]
                  g[   1],    1.907e+00,    2.370e+00,    2.800e+00
                  g[   2],    1.379e+00,    1.713e+00,    2.024e+00
- - - - - omission - - - - -
                  h[   7],    5.020e-01,    1.302e+00,    1.708e+00
                  h[   8], -9.269e-01, -2.404e+00, -3.153e+00

$ Logarythmic Gains of Fluxes $
param/flux             ,        v[   1],        v[   2],        v[   3]
```

```
constantPlayer[    1],    6.600e-01,    3.649e-01,    3.649e-01
constantPlayer[    2],    0.000e+00,    2.849e-01,    2.849e-01
constantPlayer[    3],    1.000e+00,    5.528e-01,    5.528e-01
constantPlayer[    4],    0.000e+00, -8.147e-04, -8.147e-04
constantPlayer[    5],    0.000e+00,    9.791e-03,    9.791e-03
constantPlayer[    6],    0.000e+00,    9.987e-02,    9.987e-02
constantPlayer[    7],    0.000e+00,    3.383e-01,    3.383e-01

$ Sensitivities of Fluxes with respect to rate constants $
param/var              ,        v[    1],        v[    2],        v[    3]
            alpha[    1],    1.000e+00,    5.528e-01,    5.528e-01
            alpha[    2],    0.000e+00,    8.903e-01,    8.903e-01
            alpha[    3],    0.000e+00, -9.987e-02,    9.001e-01

$ Sensitivities of Fluxes with respect to kinetic orders $
param/var                ,        v[    1],        v[    2],        v[    3]
                  g[    1],    1.520e+00,    8.401e-01,    8.401e-01
                  g[    2],    1.099e+00,    6.073e-01,    6.073e-01
                  g[    3],    0.000e+00, -2.286e+00, -2.286e+00
                  g[    4],    0.000e+00,    2.795e-01,    2.795e-01
- - - - - omission - - - - -
                  h[    6], -3.170e-16,    5.387e-01, -4.374e+00
                  h[    7], -1.545e-17, -5.684e-02, -5.684e-02
                  h[    8],    2.851e-17,    1.049e-01,    1.049e-01
```

F. OPTIMIZER (GA)

1. Introduction

The Genetic Algorithms (GAs) are known as one of the algorithms that can seek out the global minimum. GA is based on the heuristic assumptions that the best solution will be found in the regions of the parameter space containing a relatively high proportion of good solutions and that these regions can be explored by the genetic operators of selection, crossover, and mutation. GA offers a number of advantages, e.g., GA searches form a set of designs, and can explore the parameter space without trapping in local optima. However, this method has the disadvantage that the computational cost of many runs for the design code is considerably large, and there have been many parameters regarding the operations to determine with experiences and intuitions. In order to execute GAs efficiently and systematically, we have developed the search modules The algorithms suitable for optimizing biochemical networks are presented so that users can make a efficient strategy for the parameter search. This search can be readily carried out on parallel computers.

2. Function specification and application problems

2.1 Function Specification

Algorithm

- Encode method
  - -Real-coded GA (RGA)
  - -Bit-string GA (BGA)
    - -Binary coding
    - -Gray coding
- GA type [RGA, BGA]
  - -Distributed GA
  - -Distributed and Integrated GA
- -Generation alternation [RGA, BGA]
  - -Elite conservation strategy (Normal)
  - -Minimal Generation Gap (MGG)
    - -Selection method: Roulette, Tournament, Random.
- Crossover
  - -BLX (Blend Crossover)[RGA]
  - -UNDX (Unimodal Normal Distribution crossover) [RGA]
  - -UNDXm (Multi-parental Unimodal Normal Distribution crossover) [RGA]
  - -SPX (Symplex crossover)[RGA]
  - -n-point crossover [BGA]
- Mutation
  - -Uniform mutation within region [RGA]

-Uniform mutation with fixed width [RGA]
-Normal mutation with fixed width [RGA]
-Uniform mutation with variable width [RGA]
-Normal mutation with variable width [RGA]
-Bit reverse mutation [BGA]

Input files and data for GAs

- GA-parameter setting file
- GA-population setting file
- parameter file (created by the checkdae module)
- parameter-range setting file
- Fitness function

Output file and data

Parameters for executing GAs (standard output, output file)
Information regarding the best individual of each generation (standard output)
Information regarding the whole individuals every generation (output file)

2.2 Application problem

This search engine module can be applied to a maximization problem that defines a fitness function by using continuous and independent variables with the limited ranges.

Exceptionally, this search engine can be applied to the optimization of discrete problems such as a knapsack problem. In the bit string GA for which the range of variables is set [0, 1] and the quantization number is 1, the variables are determined either 0 or 1.

Note that the range is closed.

3. Execution of programs

Execution on a command line

% load module <arguments>
Arguments:
-r [parameter-range setting file] or –I [parameter file] (required)
-p [GA-parameter setting file] (required)
-s [GA-population setting file] (option)
-f [Output file]
-h Show help.

The parameter file is exactly the same as the parameter file that the CADLIVE Simulator creates. If users employ the parameter file to determine the regions of the parameters, the parameter-range setting file is not required.

Examples:

(1)

%load_module_name –r domain.dat -p param.dat -f output.dat -s init.dat

(2)

%load_module_name -h

For MPI,

(3)

%mpirun –np <number of process> <load module name> <arguments>

%mpirun –np 3 <load module name> –r domain.dat -p param.dat -f output.dat

For MPICH-G,

(4)

%mpirun –globusrsl <rsl file>

The "rsl" file is described in the manual of MPICH-G.

One process for one CPU is recommended.

How to connect the GA module to the CADLIVE Simulator

Users are able to connect the GA module to the CADLIVE Simulator, where the ranges of the parameters are defined in the parameter file (parameter setting file). In order to use the parameter file as the input file for GA modules, users have to edit the parameter file whose format is suitable for the GA module. The program is executed as follows:

%load module name - I[parameter file] -p[GA parameter setting file] - f[GA result file].

The detailed instruction is described elsewhere.

4. Input/output specification

4.1 Parameter-range setting file

The closed ranges of the parameters are defined in the parameter-range setting file, and the number of the records is the number of the parameters to search, as shown in the following box.

Format for a parameter range-setting file To maximum of 20 records.
lower value of range, upper value of range # comments

Format: Fields are sectioned by using comma (,). The preceded parts by # are comments.

For example, the ranges of three parameters are [-5.12,5.12], [-5.12, 0.0], [0.0, 3.14] as follows.

```
# Example: parameter range-setting file

-5.12, 5.12          # x1
-5.12, 0.0           # x2
0.0, 3.14            # x3
```

4.2 GA-parameter setting file

4.2.1 GA-parameter setting file

The control parameters for GAs are classified by five key words: encode method, GA type, digenesis (generation alternation), crossover, and mutation. The GA-parameter setting file is provided as follows.

```
# Format of GA-parameter setting file
Key word; Alternatives; parameter_1; ****; parameter_n;        # comments
```

The preceded parts by # are comments. The key words and alternatives are selected from the strings shown in Table F1.

Table F1 Control parameters for setting GAs. Key words and their alternatives are selected. Upper and lower cases are distinguished for defining the parameters. The program runs until the fitness becomes more than the value for terminating a search. Both RGA and BGA can select the transparent alternatives. The thin gray alternatives can be selected only by RGA, and the dark gray alternatives can be selected only by BGA.

Key word	Alternatives	Meaning
ENCODE (Encode method)	REAL	Real GA
	BINARY	Binary coding bit string type GA
	GRAY	Gray coding bit string type GA
GATYPE (Island model)	DGA	Distributed GA (island model)
	DIGA	Distributed and Integrated GA
DIGENESIS (Generation alternation)	NORMAL	Normal generation
	MGG	MGG
CROSSOVER (Crossover method)	BLX	Blend crossover
	UNDX	Unimodal Normal Distribution crossover
	UNDXm	Multi-parental Unimodal Normal Distribution crossover
	SPX	Symplex crossover
	NPOINTS	n-point crossover (BGA)
	NONE	No crossover
MUTATION (Mutation method)	RegionUni	Uniform mutation within region
	FixedUni	Uniform mutation with fixed width
	FixedNormal	Normal mutation with fixed width
	VariableUni	Uniform mutation with variable width

	VariableNormal	Normal mutation with variable width
	BitReverse	Bit reverse mutation
	NONE	None

Users can set the parameters with respect to each alternative in the following box.

```
# Parameter description with respect to key words and alternatives.
ENCODE; REAL;
ENCODE; BINARY; quantization number;
ENCODE; GRAY; quantization number;
GATYPE; DGA; maximum generation number; value for terminating a search; number of islands; population number within islands; immigration method; immigration interval; immigration rate;
(The final two parameters are required for immigration operation. Actually the above parameters must be on one line.)
GATYPE; DIGA; maximum generation number; value for terminating a search;  number of islands; population number within islands; generation number for integration;
DIGENESIS; NORMAL; number of elites; rule for selection; tournament size (if tournament is employed);
DIGENESIS; MGG; number of children generated;
CROSSOVER; BLX; alpha;
CROSSOVER; UNDX; alpha; beta;
CROSSOVER; UNDXm; alpha; beta; m;
CROSSOVER; SPX; epsilon;
CROSSOCER; NPOINTS; n;
CROSSOVER; NONE;
MUTATION; FixedUni; mutation rate; width of parameter range;
MUTATION; FixedNormal; mutation rate; standard deviation;
MUTATION; VariableUni; mutation rate;
MUTATION; VariableNormal; mutation rate;
MUTATION; RegionUni; mutation rate;
MUTATION; BitReverse; mutation rate;
MUTATION; NONE;
```

In the bit-string GA, the real values are converted into the bit sequences by dividing the range of the variables with 2^n. The number of n that determines the resolution is named as the quantization number. A part of the parameters for GA type, generation alternation, crossover, and mutation depends on the parameters of the encode methods. The parameters for GA type, generation alternation, crossover, and mutation are independent mutually.

Table F2 Values that users are allowed to set with respect to each parameter. Note that the GA module can distinguish the upper or lower cases of strings.

Parameter	Values	Default value
quantization number	Integer ≥ 1	1
maximum generation number	Integer ≥ 1	100
value for terminating a search	real value (double type)	-1.00E-10
number of islands	Integer ≥ 1	3
population number within islands	variable number +2 ≤ Integer ≤ Maximum	30
immigration switch	ON, OFF	ON
immigration interval	Integer ≥ 1	3
immigration rate	Real value [0,1]	1
generation for integration	Integer ≥1	5
number of elites	Integer ≥ 0	population number
selection rule	Roulette, Tournament, Random	Tournament
size of tournament	1 ≤ Integer ≤(population number of islands – number of parents +1)	2
number of children generated by MGG	1 ≤ Integer ≤ Maximum	population number of islands
alpha (BLX)	Real value > 0	0.5
alpha (UNDX)	Real value > 0	1
alpha (UNDXm)	Real value > 0	1
beta (UNDX)	Real value > 0	0.35
beta (UNDXm)	Real value > 0	0.35
M	1 ≤ Integer ≤ number of variables	max(1, the number of variables -2)
epsilon	Real value > 0	1/sqrt (number of variables +2)
N	Integer≥ 1	1
mutation rate	Real value [0,1]	1/ number of variables
width	Real value > 0	20
standard deviation	Real value > 0	2

4.2.2 Examples for GA parameter setting file

Example 1:

Real GA

Distributed GA, To the maximum of 100, Value for terminating a search = 0, number of islands = 5, population number within islands = 100, Immigration is ON, Immigration internal = 10, Immigration rate = 0.25, MGG, children = 10, UNDX, parameter defaults, No mutation is applied.

```
# example 1
ENCODE; REAL;
GATYPE; DGA; 100; 0; 5; 100; ON; 10; 0.25;
DIGENESIS; MGG; 10;
```

```
CROSSOVER; UNDX; ; ;
MUTATION; NONE;
```

Example 2

Binary coding bit-string GA, Quantization number = 10, Distributed and integrated GA, maximum generation 200, value for completing execution = 0, number of island = 3, population number within island = 30, generation number for integration = 20, Normal generation alternation, number of elites = 5, Roulette selection, Two-point crossover, Bit-reverse mutation, mutation rate 0.3

```
# example 2
ENCODE; BINARY; 10;
GATYPE; DIGA; 200; 0; 3; 30; 20;
DIGENESIS; Normal; 5; Roulette;
CROSSOVER; NPOINTS; 2;
MUTATION; BitReverse; 0.3;
```

4.2.3 GA-population setting file

The GA module is able to read the result file from the random search. In the n-dimensional parameter space to be searched, the GA-population setting file is provided by:

```
# Format for GA-population setting file
SearchID, fitness, parameter1, ....., , parameter_n
.....
**
SearchID, fitness, parameter1, ....., , parameter_n
```

The symbol of "#" indicates that the line is comments. The section between islands is expressed by "**". Users neither have to put "**" on the initial island nor under the final island. If such descriptions are employed, an empty island is regarded to exist.

4.2.4 Examples for GA-population setting file

Three islands have five individuals, respectively

```
# island No1
No0-67,-0.304018,0.915873,3.28519
No0-66,-9.88925,2.14165,1.47417
No0-43,-21.9843,2.8856,1.10932
No0-51,-25.6788,2.53911,1.00846
```

```
No0-5,-28.4887,0.792621,2.64166
**    #island No2
No0-53,-3.04234,0.987835,2.93188
No0-41,-17.3451,1.64468,1.63063
No0-71,-23.0538,1.26285,1.99734
No0-40,-25.7664,1.81623,1.37769
No0-42,-29.2198,2.79655,1.15365
**    #island No3
No0-10,-4.99208,0.972839,3.17752
No0-88,-17.9748,1.64134,1.6242
No0-44,-24.7664,1.06062,2.22619
No0-16,-27.1525,1.76528,1.33739
No0-6,-30.4514,1.33549,1.56515
```

4.3 Standard output

Before execution, the values of the parameters are displayed on the screen as follows.

```
===GA PARAMETERS======================
Encode Type                 : Real-coded GA

GA type                     : DGA
Max Generation     : 1000
Fin value                   : -1.000000e-10
Total population            : 1000
Number of islands  : 10
Island population  : 100

Digenesis type              : NORMAL
Number of elites   : 1
Selection type              : Tournament
Tournament Size             : 2

Crossover type              : SPX
SPX(epsilon)                : 2.645751

Mutation type               : RegionUni
Mutation rate               : 0.200000
```

```
Immigration type              : Random ring
Immigration interval          : 5
Immigration rate              : 0.500000

===Searching space information==========
Number of Variable     : 5
Domain variable[0]     : [-5.120000, 5.120000]
Domain variable[1]     : [-5.120000, 5.120000]
Domain variable[2]     : [-5.120000, 5.120000]
Domain variable[3]     : [-5.120000, 5.120000]
Domain variable[4]     : [-5.120000, 5.120000]
========================================
```

A summary for the GA-population setting file is displayed, which includes the file name, the island numbers, the population number within each island, the number of short islands, and the number of redundant islands. In the case of the short islands, the program generates a new island with a random population automatically. When the number of islands is larger than the appointed one, the number of the redundant islands is displayed.

```
===Initial Generation File==============
File name:rs_out_list
Island, Population, # of read, Random, Redundant
Island[0], 15, 20, 0, 5
Island[1], 15, 19, 0, 4
Island[2], 15, 5, 10, 0
Island[3], 15, 31, 0, 16
Island[4], 15, 11, 4, 0
Number of redundant islands: 2        #    displayed when redundant islands exist.


========================================
```

The information regarding the best individual is displayed every generation on the screen. After completing the search, the result is displayed. The bit-string GA indicates the real value sequences, not the bit sequences. Indication of results every generation:

```
generation[0], Obtain: 0, Fitness: -26.652798, Gene: (-1.982981, -4.013446, 0.087129, -1.089640, 0.110301)
generation[1], Obtain: 1, Fitness: -22.533149, Gene: (-0.890659, -1.025080, -2.211625, 0.958039, -1.936774)
generation[2], Obtain: 2, Fitness: -14.719590, Gene: (0.074704, -0.078463, -0.083288, 1.033160, -2.918250)
            (omission)
(when the target fitness or the maximum generation number is achieved.)
******Search Result******
```

```
Best gene
Obtained Generation      : 65
Fitness                  : -0.000000
Gene                     : (-0.000000, -0.000000, 0.000000, 0.000000, 0.000000)
*************************
```

4.4 Output file

The output file is appointed as an argument on a command line. The results regarding the entire individuals every generation are written in the form of "csv".

```
generation,0
Best gene:
Obtained generation, fitness, variable[0], variable[1], variable[2], variable[3], variable[4]
0, -46.007195, -3.995016, -0.149753, 3.052604, 3.074872, -2.078418
ID, fitness, variable[0], variable[1], variable[2], variable[3], variable[4]
[0][0], -46.007195, -3.995016, -0.149753, 3.052604, 3.074872, -2.078418
[0][1], -65.409291, -2.949895, 4.444091, 1.857122, 1.830248, -1.036918
[0][2], -83.227264, 1.431177, 2.992203, 3.875130, 4.705202, 0.037506
  (omission)
[1][7], -100.625396, -4.595544, 3.022847, -4.746842, -3.206425, -0.934593
[1][8], -110.044951, 2.700800, 4.610274, -0.096368, -4.767272, 1.675321
[1][9], -135.758812, 3.813671, -4.647205, -3.593073, -4.125371, 5.059261
```

The brackets [][] indicate [island number][individual number within islands]. The individuals within each island are sorted according to the values of the fitness. A smaller number indicates a higher fitness.

5. Detailed specification of functions

5.1 Encode method

This optimization program presents the real-coded GA (RGA) and the bit-string GA (BGA) as the method that encodes design variables. For the bit-string GAs, the binary coding or gray coding is employed as the method that exchanges between integers and bit-string (0,1).

5.1.1 Bit-string GA

The bit-string GA encodes the design variables as the bit string that consists of 0 and 1. The bit-string GA divides the range of variables into 2^n regions, and assigns the integer of $0,\ldots,2^n-1$ for each region in a numerical order, thereby converting the integer to the bit-string. Two encoding methods as follows:

- Binary coding: converting an integer by binary arithmetic.
- Gray coding: converting an integer with a binary code under the condition that the Haming distance between two codes to continue is always kept 1. The Haming distance is defined as the number of the figures with

different values.

Table F3 Binary and gray codes

Integer	Binary code	Gray code
0	000	000
1	001	001
2	010	010
3	011	011
4	100	110
5	101	111
6	110	101
7	111	100

5.1.2 Real GA

The real GAs employ the sequence of real values as the chromosome.

5.2 GA type

This module presents two types of GAs, the distributed GA (DGA, island model) and the distributed and integrated GA (DIGA).

5.2.1 Distributed GA

The distributed GA divides the population into multiple islands. The individuals of each island evolve within the island, and some individuals immigrate mutually among the islands. The immigration method can be selected from random swap, random ring, and no immigration.

- Random swap: selecting immigrants randomly out of randomly selected islands under the given immigration rate in order to exchange two individuals between the islands. The elites of each island are not allowed to immigrate.
- Random ring: determining the destination of immigrants so that the way between the destination and origin of the immigrants forms a ring. The elites of each island are not allowed to immigrate.

5.2.2 Distributed and integrated GA (DIGA)

The DIGA divides the population into multiple islands as well as DGA. Each island evolves independently until all the islands are integrated to a new island at the generation number for integration. At their integration, the elites of each island are gathered to the new one, and the remaining individuals are selected randomly. The DIGA is not allowed to operate the immigration.

5.3 Generation alternation

The GA module presents two methods for generation alternation, the ordinary generation alternation method (employing elite conservation method), and Minimal Generation Gap (MGG). The ordinary method is sometimes called the simple GA, but such a name may cause users to misunderstand the methods. Here, we name the ordinary

generation alternation method.

5.3.1 Ordinary generation alternation method

The ordinary method replaces all the parents by children. Actually, the parents except the elites are replaced, because the elite conservation strategy is employed. If users want to replace all the parents including the elites, set "the number of elites" as zero.

The roulette, tournament, and random selections are presented as the methods for selecting the parents to crossover. The roulette is made based on the difference:

(difference) = (fitness of an individual) – (the lowest fitness of the island).

5.3.2 Minimal Generation Gap (MGG)

The MGG is the most desirable model that can avoid early convergence and suppress evolutionary stagnation. The procedure of MGG can be summarized in the following box.

1. Create an initial population randomly
2. Sample two individuals randomly without replacement from the population
3. Generate children from the selected parents and characterize them
4. Replace two individuals (parents) by the best individual and the roulette-selected individual, resulting in the next generation.

The above method is the original MGG. Some crossover methods require more than two parents. Thus, the crossover is defined by the rules:

- BLX-α, n-point mutation: The parents to replace are the same ones to crossover.
- UNDX, UNDX-m, SPX: Two parents to replace are selected randomly out of all the parents that have been employed to generate children.

5.4 Crossover method

The crossover method to apply depends on the encoding methods, the bit-string GA (BGA) and the real GA (RGA). In this section, [RGA] indicates that the crossover is available for RGA, [BGA] for BGA.

5.4.1 BLX-α [RGA]

The BLX is a simple algorithm among the real-coded GAs, but its performance is not guaranteed in the variable-dependent functions.

$$c_i = u(\min(p_i^1, p_i^2) - \alpha I_i, \max(p_i^1, p_i^2) + \alpha I)$$
$$I_i = | p_i^1 - p_i^2 |$$

,

where $\mathbf{p}^1 = (p_1^1, \ldots, p_n^1), \mathbf{p}^2 = (p_1^2, \ldots, p_n^2)$ are the parents, $\mathbf{c} = (c_1, \ldots, c_n)$ is the child, and $u(\cdot)$ represents the function that generates a uniform random number within the closed region of the parentheses.

5.4.2 UNDX / UNDX-m [RGA]

The UNDX-m has been proposed to improve a search performance on an optimization problem with poorly scale coordinate systems. The UNDX-m generates offspring vector values by sampling values from the m-dimensional space that the m+1 parents span around their middle point. The UNDX is the same as the UNDX-m at m = 1 except some respects. The prototype algorithm of the UNDX-m in the n-dimensional parameter space is provided by:

(1) Select m +1 parents $\boldsymbol{x}^1,\ldots,\boldsymbol{x}^{m+1}$ randomly from the population.

(2) Let the center of mass of these parents

$$\boldsymbol{p} = \frac{1}{m+1}\sum_{i=1}^{m+1}\boldsymbol{x}^i ,$$

and let the difference vector between $\boldsymbol{x}^j$ and $\boldsymbol{p}$ be $\boldsymbol{d}^i = \boldsymbol{x}^i - \boldsymbol{p}$.

(3) Select another parent $\boldsymbol{x}^{m+2}$ from the population randomly

(4) Let *D* be the length of the component of $\boldsymbol{d}^{m+2} = \boldsymbol{x}^{m+2} - \boldsymbol{p}$ orthogonal to $\boldsymbol{d}^1,\ldots,\boldsymbol{d}^m$.

(5) Let $\boldsymbol{e}^1,\ldots,\boldsymbol{e}^{n-m}$ be orthogonal bases of the subspace orthogonal to the vectors $\boldsymbol{d}^1,\ldots,\boldsymbol{d}^m$.

(6) Generate the child $\boldsymbol{x}^c$ by:

$$\boldsymbol{x}^c = \boldsymbol{p} + \sum_{i=1}^{m} w_i \boldsymbol{d}^i + \sum_{i=1}^{n-m} v_i D\boldsymbol{e}^i ,$$

where w_i, v_i are the random numbers that follow normal distributions $N(0,\sigma_\xi^2), N(0,\sigma_\eta^2)$, respectively. The parameters, σ_ξ and σ_η, are provided by:

$$\sigma_\xi = \alpha/\sqrt{m}, \qquad \sigma_\eta = \frac{1}{\sqrt{n-m}}\frac{\sqrt{m+1}}{\sqrt{m+2}}\frac{\sqrt{3}}{\sqrt{2}}\beta$$

where α = 1.0 and β = 0.35 are recommended.

The UNDX is carried out as follows:

(1) Select three parents $\boldsymbol{x}^1, \boldsymbol{x}^2, \boldsymbol{x}^3$ randomly.

(2) Let the center of mass of these parents:

$$\boldsymbol{x}^p = (\boldsymbol{x}^1 + \boldsymbol{x}^2)/2 ,$$

and let the difference vector $\boldsymbol{d} = \boldsymbol{x}^1 - \boldsymbol{x}^2$.

(3) Let *D* to be the distance between the third parent $\boldsymbol{x}^3$ and the line connecting $\mathbf{x}^1$ to $\mathbf{x}^2$.

(4) Generate a child $\mathbf{x}^c$ by the following equation:

$$\mathbf{x}^c = \mathbf{x}^p + \xi\mathbf{d} + D\sum_{i=1}^{n-1}\eta_i\mathbf{e}_i ,$$

where ξ and η_i are the random numbers that follow normal distributions $N(0,\sigma_\xi^2)$ and $N(0,\sigma_\eta^2)$, respectively. The parameters, σ_ξ and σ_η, are given by:

$$\sigma_\xi = \alpha, \qquad \sigma_\eta = \beta/\sqrt{n}$$

where α = 0.5 and β = 0.35 are recommended.

Difference between UNDX and UNDX-m

The recommended value of α is a half of that of the UNDX-m, because the UNDX employs the difference vector between the parents instead of the difference vector based on the center of mass. Actually, both are the same. On the other hand, for the sub-search component, the UNDX-m provides:

$$\sigma_\eta = \beta / \sqrt{n-1} \quad \text{at m = 1,}$$

where the difference is indicated clearly between them.

Table F4 Difference between UNDX and UNDX-m

	Revision on sub-search component	
UNDX	$\sigma_\xi = \alpha$	$\sigma_\eta = \beta / \sqrt{n}$
UNDX-m	$\sigma_\xi = \alpha / \sqrt{m}$	$\sigma_\eta = \frac{1}{\sqrt{n-m}} \frac{\sqrt{m+1}}{\sqrt{m+2}} \frac{\sqrt{3}}{\sqrt{2}}$ $\beta(m=1, \beta/\sqrt{n-1})$

Note: Rank cancellation

In the UNDX-m, the space that the difference vectors span is the main search component, and the orthogonal subspaces are the minor search components. If the difference vector sets that the main search component belongs to have any dependent element, the dimension of the main search space is less than the number of the vectors that span it, canceling the rank.

When the rank is cancelled, the optimization module does not change the main search component, and lets the minor search component to be the subspace orthogonal to the space (***P***) that independent vectors of the mains search components span. The parameter *D* is set as the component of the vector $\boldsymbol{d}^{m+2}$ orthogonal to the space (***P***).

5.4.3 SPX [RGA]

SPX generates offspring vector values by uniformly sampling values from the simplex formed by multiple parent vectors.

(1) Seletct m+1 parents $\vec{P}_0, \ldots, \vec{P}_n$ randomly from the population.

(2) Let the center of mass of the parents $\vec{G}$ be

$$\vec{G} = \sum_{i=0}^{n} \vec{P}_i \ .$$

(3) The vector $\vec{x}_k, \vec{C}_k$ are determined as follows:

$$\vec{x}_k = \vec{G} + \varepsilon(\vec{P}_k - \vec{G}) \quad (k = 0,\ldots,n)$$
$$\vec{C}_k = \begin{cases} \vec{0} & (k = 0) \\ r_{k-1}(\vec{x}_{k-1} - \vec{x}_k + \vec{C}_{k-1}) & (k = 1,\ldots n) \end{cases},$$

where ε is the extension coefficient (ε > 0), r_k is provided by random uniform function:

$$r_k = (u(0,1))^{1/(k+1)} \quad (k = 0,\ldots,n-1).$$

The coefficient ε is provided by:

$$\varepsilon = \sqrt{n+2}$$

(4) The child is provided by

$$\vec{C} = \vec{x}_n + \vec{C}_n$$

5.4.4 N point crossover [BGA]

N points to crossover are selected at a chromosome randomly, and the parameters between the crossing points are replaced alternately by the corresponding parameters of a parental chromosome.

5.5 Mutation method

The mutation methods to apply depend on BGA or RGA. In this section, the mutation method for RGA is explained. The individuals that have been selected out of an island at a given mutation rate are mutated according to the various methods, where $(x_1,\ldots,x_n)$ is the coordinate of an individual, and $(z_1,\ldots,z_n)$ is the coordinate that shows the best individual in all of the individuals within the island.

5.5.1 Uniform mutation within the region [RGA]

The vector of the selected individual is changed under the uniform random distribution within the given region.

5.5.2 Uniform mutation with fixed width [RGA]

A mutated vector is given by the function:

$$u(x_i - w, x_i + w),$$

which generates a uniform random number between $x_i - w$ and $x_i + w$, where w is the width that users give arbitrarily.

5.5.3 Normal mutation with fixed width [RGA]

A mutated vector is provided by the function:

$$u(x_i - w, x_i + w),$$

which generates a normal random number between $x_i - w$ and $x_i + w$, where w is the width that users give arbitrarily. The coordinate of the mutated vector follows a normal distribution:

$N(x_i, s^2)$,

where *s* is the standard deviation that users can give arbitrarily.

5.5.4 Uniform mutation with variable width [RGA]

A mutated vector is provided by the function:

$$u(x_i - w_v, x_i + w_v),$$

which generates a uniform random number between $x_i - w_v$ and $x_i + w_v$, where w_v is given by:

$$w_v = | z_i - x_i |.$$

5.5.5 Normal mutation with variable width [RGA]

A mutated vector is provided by the function:

$$u(x_i - w, x_i + w),$$

which generates a normal random number between $x_i - w$ and $x_i + w$, where *w* is the width that users give arbitrarily. The coordinate of the generated vector follows a normal distribution:

$N(x_i, s_v^2)$,

where the standard deviation is $s_v = | z_i - x_i |$.

5.5.6 Bit reverse mutation [BGA]

A vector is mutated by reversing the bit according to the given mutation rate.

6. Flow chart

6.1 Single without MPI

```
+-main
    +-initialize parameters
    +-display the employed parameters
    +-generate the initial population and characterize the fitness
    +-if(ORDINARY )
        +-NormalExecute()
            +-for (final generation)
                +-if (DGA && immigration generation) immigrate
                +-if (DIGA && integration generation) integrate
                +-for (all the islands)
                    +-for (non-elite individuals)
                    +-select parents based on the crossover method
                    +-generate children by crossover
                    +-replace parents by children (except elites)
```

```
                                        +-for (non-elite individuals) mutate
                                        +-characterize the fitness of non-elites
                                        +-sort the individuals according to their fitness
                                +-conserve the best individual of all the islands
                                +-if (termination condition) break;
                        +-display the results
        +-else if (MGG)
                +-MGGExecute()
                        +-for(final generation)
                                +-if (DGA && immigration generation) immigrate
                                +-if (DIGA && integration generation) integrate
                                +-for (all the islands)
                                        +-sampling parents without replacement
                                        +-generate a specific number of children
                                        +-characterize the fitness of children
                                        +-select the best individual and another by roulette selection
                                        +-replace parents by children
                                +-conserve the best individual of all the islands
                                +-if (termination condition) break;
                        +-display the results
```

6.2 MPI

6.2.1 Master-slave model

This optimization module presents a master-slave model for parallel computing, where the master manages the whole processes and send jobs to the slaves. Concretely speaking, the master manages the whole population, and displays the results. On the other hand, the slaves execute the evolution within the islands.

Master:

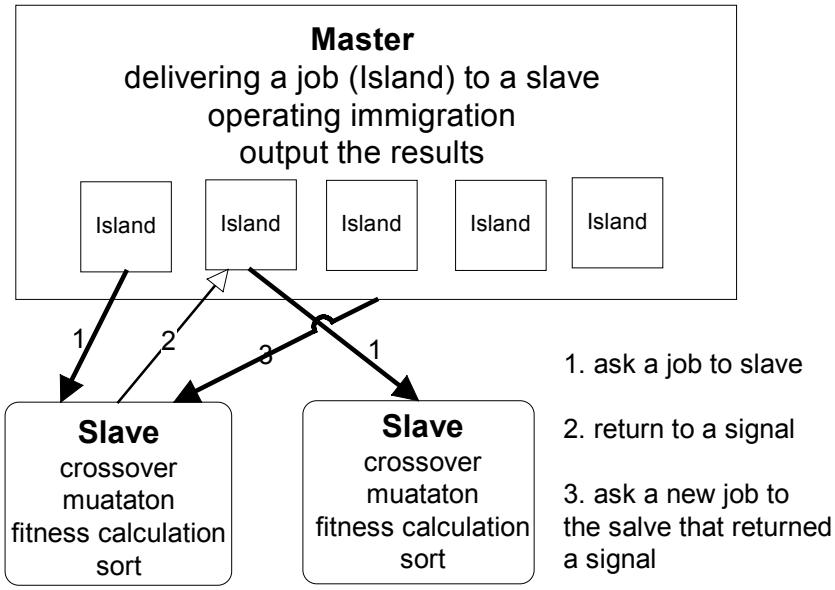
- Generating the whole population
- Sending jobs to slaves.
- The queue manages jobs and asks the slave that has terminated the task to do another job
- Managing immigration or integration
- Displaying the results

Slaves:

- Independently processing a job
- Crossovering and mutating chromosomes
- Characterizing the fitness and sorting the individuals.

The speed of parallel computation mainly depends on the grain of calculation provided for each CPU. The small grain increases the overhead of communication, resulting in decreasing the advantage of parallel computation. Delivering a job to a slave by an island unit increases the grain of calculation. The calculation rate of a simple parallel system is limited by the slowest CPU. By contrast, the use of the master-slave model loads many jobs to the CPU with high-performance, thus the calculation rate is enhanced more than the simple parallel system.

The master-salve model is shown in the following diagram.



6.2.2 A flow char in MPI

```
+-main
    +-initialize MPI
    +-if (MASTER)
        +-input the GA parameter setting file and variable range setting file
        +-send the data on parameter and range to all the slaves
        +-display the employed parameters
    +-if (SLAVES)
        +-receive the data from the master
        +-register the data
    +-if (MASTER)
        +-generate the initial population of all the islands
        +-for (SLAVES)
            +-send the data of the initial population
            +-if (All the islands are sent) break;
        +-for (infinite loop)
```

```
                +-receive the response from a slave
                +-if (there is an island to send data) send the island to the slave
                +-if (All the islands are received) break;
        +-for (All the SLAVES) send the command to exit the loop
+-if (SLAVE)
        +-for (infinite loop)
                +-receive data from the master
                +-if (Exiting the loop is commanded) break;
                +-characterize the fitness of all the individuals
                +-send the data to the master
+-if (ORDINARY)
        +-normalExecute()
                +-if (MASTER)
                        +-for (final generation)
                                +-if (DGA && immigration generation) immigrate
                                +-if (DIGA && integration generation) integrate
                                        +-for (All the SLAVES)
                                        +-send the data of all the islands
                                        +-if (All the islands are sent) break;
                                +-for (infinite loop)
                                        +-receive the response from a slave
                                        +-if (there is a island to send)
                                        send the island to the slave
                                        +-if (all the islands are received) break;
                                +-conserve the best of all the islands
                                +-if (termination condition)
                                        +-send the command for exiting the loop, break;
                        +-display the results
                +-if (SLAVE)
                        +-for (infinite loop)
                                +-receive the data from the master
                                +-if (Exiting the loop is commanded) break;
                                +-for (non-elite individuals)
                                        +-select parents based on the crossover method
                                        +-generate a child by crossover
                                +-replace parents by children (except elites)
                                +-for (non-elite individuals) mutate
                                +-characterize the fitness of non-elites
```

```
                                    +-sort the individuals according to the fitness
                                    +-send data to the master
```

```
+-else if (MGG)    /* MGG */
        +-MGGExecute()
                +-if (MASTER)
                        +-for (final generation)
                                +-if (DGA && immigration generation) immigrate
                                +-if (DIGA && integration generation) integrate
                                        +-for (All the SLAVES)
                                        +-send the population data of all the islands
                                        +-if (All the islands are sent) break;
                                +-for (infinite loop)
                                        +-receive the response from a slave
                                +-if (there is an island to send) send the island to the slave
                                        +-if (All the islands are received) break;
                                +-conserve the best individual of all the islands
                                +-if (termination condition) break;
                                        +-send the command for termination to the slaves,
                                                break;
                        +-display the results
                +-if (SLAVE)
                        +-for (infinite loop)
                                +-receive the data from the master
                                +-if (Termination is commanded) break;
                                +-select parents randomly
                                +-generate a specific number of children from parents
                                +-characterize the fitness of children
                                +-select the best individual and another by roulette selection
                                +-replace parents by children
                                +-if (Mutation)
                                        +-mutate
                                        +-characterize the fitness and sort
                                +-send data to the master.
```

G. CLIENT-SERVER MODEL

The CADLIVE Simulator is a client-server model, supporting Internet Explorer. Users access a server through php, and operate on Web pages. The system functions on LINUX (Red Hat Version 7.1), and employs Xerces2 Java Parser for parsing the regulator-reaction equations in XML, postgreSQL for managing database, the C programming language for calculating differential and algebraic equations and matrixes, and Message-Passing Interface (MPI) for parallel computing.