TomClass Manual

preleminary version, 12.11.2012

Contents

1 Constraints

Constraints are used to formulate biological assumptions precisely. A constraint is a statement involving predefined predicates. Typical examples for predicates are edge labels like "Activating" or "Inhibiting". Predicates may be connected in any way using the boolean operators `and,or,not,=>,xor`. Here `=>` represents *implies*.

1.1 Edge Labels

Biologically, edge labels represent specific effects of a regulator on its target. Formally, they enforce specific inequalities to hold between pairs of parameters of the target component.

An edge label predicate consists of up to five arguments: *label, regulator, target, threshold* and *restriction*. The *label* determines the kind of inequalities that must hold. A detailed description follows below. The pair *(regulator, target)* must be an edge in the interaction graph and *threshold* must be one of the thresholds of that edge.

The *restriction* is an optional argument that may be left out. It specifies under which conditions the label must hold. With it, we can specify, for example, that an activation exists *given* that some regulator is not present. Formally, it is a boolean expression involving components and activities, e.g `A=B or D>1`.

Note: A restriction of an edge label must not involve the regulator, because its activities are already set to be just below or above the given threshold.

Currently implemented labels are:

Label	Inequalities
Activating	$>$ somewhere
ActivatingOnly	$\geq$ everywhere and $>$ somewhere
Inhibiting	$<$ somewhere
InhibitingOnly	$\leq$ everywhere and $<$ somewhere
NotActivating	$\leq$ everywhere
NotInhibiting	$\geq$ everywhere
Observable	$>$ or $<$ somewhere
NotObservable	$=$ everywhere

Figure 1: Examples for edge labels

`Activating(A, B, 1)`
An example of an edge label predicate without restriction. It enforces that the edge (A,B,1) is activating somewhere.

`NotObservable(B, A, 2, A=D or D=0)`
An example with restriction. States that the edge $(B, A, 2)$ must not be observable when `A=D or D=0`.

1.2 Inequalities

Parameter inequalities are the most basic constraints. They consist of a *lhs operand*, an *operator* and a *rhs operand*. An operand is either the *name of a parameter* or an *activity*. The naming convention for parameters is `K_x_y`, where `x` is a component name and `y` denotes the threshold levels of `x`'s regulators. The levels are sorted lexicographically by the regulators names. For example: If `A` has two regulators named `XZY` and `XZ` with thresholds `(1,)` and `(1,3)` respectively, then the parameter where `XYZ` is below thresholds `1` and `XZ` above threshold `3` is called `K_A_03`. An input component A that is not regulated has only one parameter, denoted by `K_A_`.

1.3 Generalized Inequalities

Syntax: `All(restriction, component, operator, value)`

Figure 2: Examples for parameter inequalities.

`K_A_02>1`
Enforces that the value of `K_A_02` is greater than 1.

`K_A_00!=K_A_11`.
Enforces that the value of `K_A_00` is different from the value of `K_A_11`.

This predicate is a shorthand for a conjunction or disjunction of parameter inequalities. The parameters to which this constraint applies are chosen by the *restriction*. It enforces that all/some parameters of *component* satisfy the inequality given by "parameter operator value". The *operator* is one of `>,<,>=,<=,=,!=`.

Figure 3: Examples for generalized parameter inequalities.

`All(A=1 or B>0, A > 1)`
Enforces that all parameters of `A` that intersect the description `A=1 or B>0` are greater than 1.

`Some(A>=0, A = 0)`
Enforces that some parameters of A that intersect the description `A>=0` are equal to 0. Note that the description `A>=0` is true in every state which means that the set of parameters to which this constraint is applied is the complete set of parameters of A.

1.4 Boolean Components

Syntax: `Boolean(formula, component)`.

This predicate enforces that all parameters of *component* that intersect the *formula* are set to 1. All other are set to 0.

Note that this is, in general, not the same as defining the component's behavior to be exactly given by *formula*. It is, however, the same, if the interactions and threshold are consistent with the formula.

For example: Suppose `A` has two regulators `B` and `C` with thresholds `(1,)` and `(1,3)`. A consistent formula is for example `B=1 or C=0`. An inconsistent formula is for example `A>0 and (B=1 or C>=2)`. It is inconsistent for two reasons: The interaction (A, A) is not present, and 2 is not a threshold of (C, A). After intersecting the formula with A's parameters, it will be equivalent to `B=1 or C>=1`.

Figure 4: Example for boolean formula constraints.

`Boolean(A=1 and B=1, D)`

If the only interactions targeting `D` are `(A,(1,)),(B,(1,))`, then `D` is governed by the boolean equation `A=1 and B=1`.

1.5 Set Inclusion

Syntax: `AllInSet(restriction, component, set)`

An InSet constraint enforces that all or some parameters must lie within the given *set*. The parameters to which this constraint applies are chosen by the *restriction*. The InSet constraint must be quantified giving rise to two types: `AllInSet` and `SomeInSet`.

Figure 5: Examples for InSet constraints.

`AllInSet(A=1 or B>0, A,` $\{0,2\}$`)`
Enforces that all parameters of A that also satisfy the restriction `A=1 or B>0` lie within the set $\{0,2\}$.

`SomeInSet(A=1 or B>0, A,` $\{0,2\}$`)`
An example with restriction. States that the edge $(B,A,2)$ must not be observable when `A=D or D=0`.

1.6 Equality

Syntax: `AllEqual(restriction, component)`

An Equal constraint groups parameters of a *component* together by enforcing that their values must be equal. The parameters to which this constraint applies are chosen by the *restriction*. It is different from an All/Some or InSet constraint in that the parameter value must not be given or restricted. The predicate must be quantified, giving rise to two types: `AllEqual` and `SomeEqual`.

Figure 6: Examples for equality constraints.

`AllEqual(A=1, A)`
In the case that `A=1` references at least two different parameters of A, this constraint enforces that all of them must take the same value.

`SomeEqual(C=1 or B>0, cA)`
In the case that `C=1 or B>0` references at least two different parameters of A, this constraint enforces that at least two of them must take the same value.

1.7 Multiplexes

Syntax: `Multiplex(formulas, component)`

A multiplex reduces the parametrizations of a *component* by partitioning its parameters into sets of identical intersections with the *formulas*. For each parameter it computes which formulas are intersected. Parameters with the same intersection pattern are grouped together. An AllEqual constraint is applied to every group.

Multiplexes are, for example, useful when modeling components that form compounds. For example: Suppose that a component A has exactly two regulators, B and C, that must both be present in order to change the activity of A. A multiplex constraint `Multiplex(B>=1 and C>=3, A)` then forces A to behave like a component with two artificial parameters: either B and C are present, or at one of them is not. A multiplex therefore reduces the regulatory complexity of a component.

Figure 7: Examples for multiplex constraints.

`Multiplex(A=1 and B=1, C)`
C is regulated by the complex consisting of A and B.

`Multiplex(A=1 and B=1, D=1, C)`
C is regulated by the complex consisting of A and B and another component D.

1.8 Paths

Syntax: `Path(states)`
Enforces that the path consisting of the given *states* exists in the asynchronous unitary transition graph. The activities of a state or sorted lexicographically. Suppose, for example, that a system consists of 3 components A, B and C with maximal activities $1, 1, 2$. The state $A = 0, B = 1, C = 2$ is then denoted by 012.

Figure 8: Example for path constraints.

`Path(000,001,011)`

Enforces the transitions $000 \to 001$ and $001 \to 011$.

1.9 Fixpoints

Syntax: `Fixpoint(state)`
Enforces *state* to be a fixpoint.

Figure 9: Example for fixpoint constraints.

`Fixpoint(101)`
Enforces the transition $101 \to 101$.

2 Instantiation

2.1 NewByCP

NewByCP is an script the enumerates and stores parametrizations that satisfy biological assumptions in a database. All assumptions are combined into a boolean expression over parameter predicates, see Sec. 1 for definitions. This constraint is then passed to a CP solver that enumerates every acceptable model. All models are stored in a database.

- **DB_name** = `'AB.sqlite'`

  The file name (string) of the database where parametrizations will be added.

- **Components** = `[('A',1), ('B',2)]`

  A list of tuples containing component names (string) and maximal activities (int).

- **Interactions** = `[('A','B',(1,)),('B','A',(1,2))]`

  A list of tuples containing regulator name (string), target name (string) and thresholds (tuple of ints). *Note*: The Python syntax for tupels containing only one element x is (x,).

- **Constraint** = `'Inhibiting(A,B,1) and Activating(B,A,1)'`

  A constraint (string) as defined in Sec. 1.

2.2 NewByPerturbation

NewByPerturbation is an script that perturbes the parameter values of a set of given parametrizations and stores the results in a database. It is helps investigating properties of models that are "close" to a well established model. The script takes a *depth* parameter that determines how many parameter values may be perturbed at once. Perturbations are limited to adding or subtracting 1 to a parameter value.

The initial set of parametrizations, typically just one, is defined by a constraint as in Sec. 2.1.

NewByPerturbation automatically creates a property column recording, for every model, the distance from an initial model.

Note: The distance is not well defined if there are more than one initial models.

- **DB_name** = `'perturbed.sqlite'`

  The file name (string) of the database where parametrizations will be added.

- **Perturbation_depth** = `2`

  The maximal number parameter values that may be perturbed simultaneously.

- **Property_name** = `'P'`

  The name under which the distance to an initial model is recorded in the database.

- **Components** = `[('A',1), ('B',2)]`

  A list of tuples containing component names (string) and maximal activities (int).

- **Interactions** = `[('A','B',(1,)),('B','A',(1,2))]`

  A list of tuples containing regulator name (string), target name (string) and thresholds (tuple of ints). *Note*: The Python syntax for tupels containing only one element x is (x,).

- **Constraint** = `'Inhibiting(A,B,1) and Activating(B,A,1)'`

  A constraint (string), see Sec. 1, that defines the set of initial models.

2.3 NewBySampling

3 Annotation

3.1 AnnotateByGT

AnnotateByGT is a general interface to the synchronous or asynchronous transition graphs of a model. It is a script that requires some programming by the user. performs annotation by a *graph traversal* (GT) algorithm.

- **DB_name** = `'AB.sqlite'`
- **Restriction** = `'K_v2_0=K_v2_1'`
- **Property_name** = `'GT1'`
- **Property_type** = `'int'`
- **Description** = `'Checks the out-degree of 00.'`

3.2 AnnotateByLTL

- **DB_name** = `'AB.sqlite'`

  The file name (string) of the database where parametrizations will be annotated.

- **Restriction** = `''`

  A SQL statement (string) defining the parametrizations to be annotated. The empty string denotes all parametrizations.

- **Property_name** = `'LTL1'`

  The name (string) of the property to be added to the database. The name will be the header of the column containing the labels.

- **Property_type** = `'int'`

  A SQLite datatype (string) like `'int'` or `'text'`.

- **Description** = `'Checking a simple G property.'`

  A description (string) of the property. Used only as a long version of the short-hand `Property_name`.

- **Formula** = `'G(A=0 | B!=1)'`

  A LTL specification (string) in NuSMV syntax.

- **Initial_states** = `'A!=0 | B=0'`

  An initial states specification (string) in NuSMV syntax.

- **Verification_type** = `'Forall'`

  Determines wheter the LTL formula should hold `'Forall'` or `'Forsome'` initial states.

3.3 AnnotateByCTL

- **DB_name** = `'AB.sqlite'`

  The file name (string) of the database where parametrizations will be annotated.

- **Restriction** = `''`

  A SQL statement (string) defining the parametrizations to be annotated. The empty string denotes all parametrizations.

- **Property_name** = `'CTL1'`

  The name (string) of the property to be added to the database. The name will be the header of the column containing the labels.

- **Property_type** = `'int'`

  A SQLite datatype (string) like `'int'` or `'text'`.

- **Description** = `'Checking a simple EF property.'`

  A description (string) of the property. Used only as a long version of the short-hand `Property_name`.

- **Formula** = `'EF(A=0 & B!=1)'`

  A CTL specification (string) in NuSMV syntax.

- **Initial_states** = `'A!=0 | B=0'`

  An initial states specification (string) in NuSMV syntax.

- **Verification_type** = `'Forall'`

  Determines wheter the CTL formula should hold `'Forall'` or `'Forsome'` initial states.

4 Analysis

4.1 AnalyzeClasses

- **DB_name** = `'AB.sqlite'`

  The file name (string) of the database to be analyzed.

- **Properties** = `['CTL1', 'LTL1']`

  The properties (list of strings) for which occuring combinations should be counted.

- **Restriction** = `''`

  A SQL statement (string) defining the parametrizations to be annotated. The empty string denotes all parametrizations.