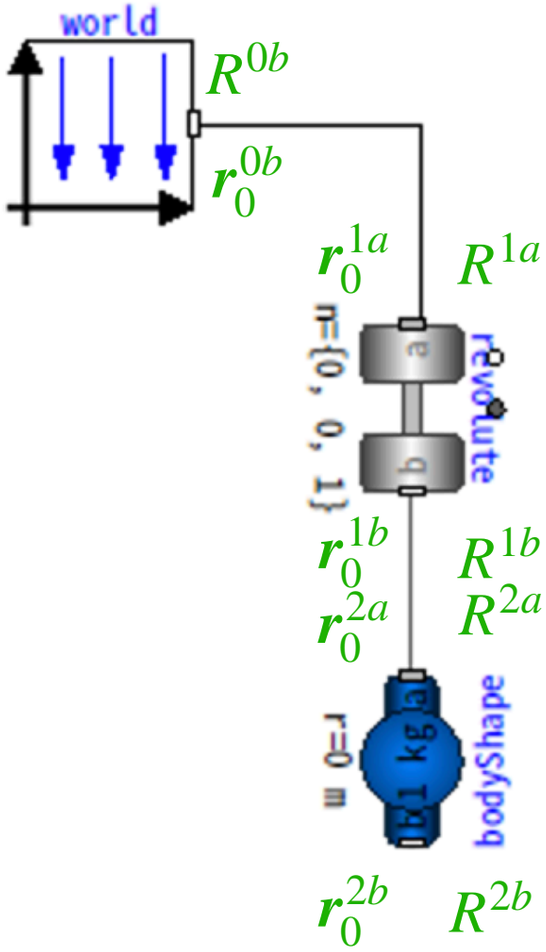
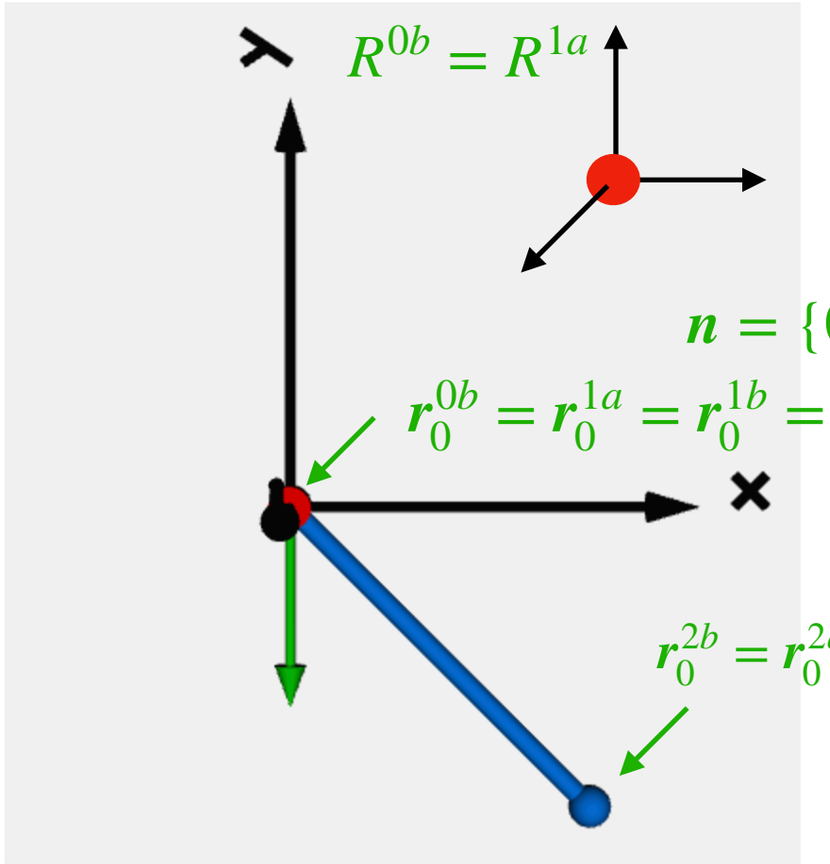
オープンCAEシンポジウム2022 トレーニング
OpenModelicaによる
マルチボディダイナミクス実習
（解説編）





オープンCAE学会　2022.09.10

田中周(アマネ流研), zeta_plusplus(Modelicaライブラリ勉強会)

はじめに

Modelica MultiBody ライブラリは、３次元力学的要素を提供するフリーのModelicaパッケージであり、これらの要素を使用するとロボットや機械装置や車両のような機械系のモデル化ができます。特徴としてすべてのコンポーネントは大きさと色をもったアニメーションの情報をもっています。

ここでは、Multibody ライブラリの考え方やしくみについて、振り子のモデルを使って解説します。

- The purpose of this document is introducing the Modelica.Mechanics.MultiBody package which is included in the Modelica Standard Library (MSL). This document uses libraries, software, figures, and documents included in MSL and those modifications. Licenses and copyrights of those are written in next page.

Contents

OpenModelica の特徴

物理的・工学的な問題　（制御系、機械系、電気系、流体系、伝熱系、…）

テキストベースの
モデリング

グラフィカルなコンポーネントベース
のモデリング

数学的モデル（微分方程式系、微分代数方程式系）

Modelica言語
による記述

変換

Cコード

コンパイル
様々なソルバーとリンク

実行ファイル

シミュレーション実行
スクリプトによるパラメトリックスタディ

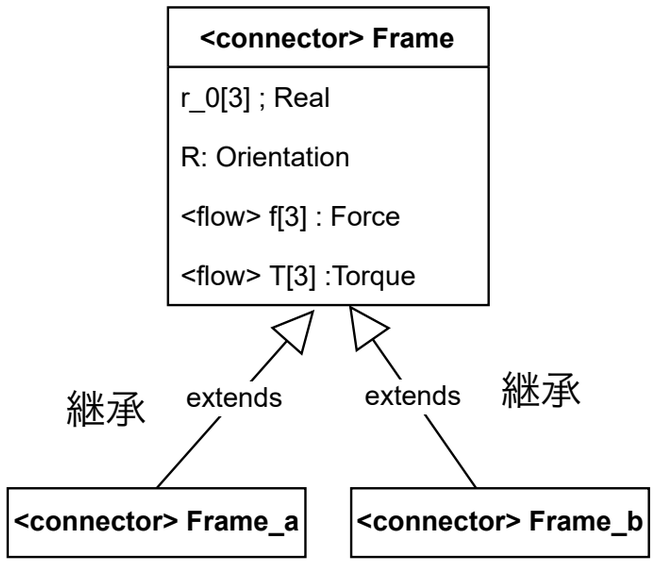
ポスト処理　(2D プロット、3Dマルチボディダイナミクス)

自動化

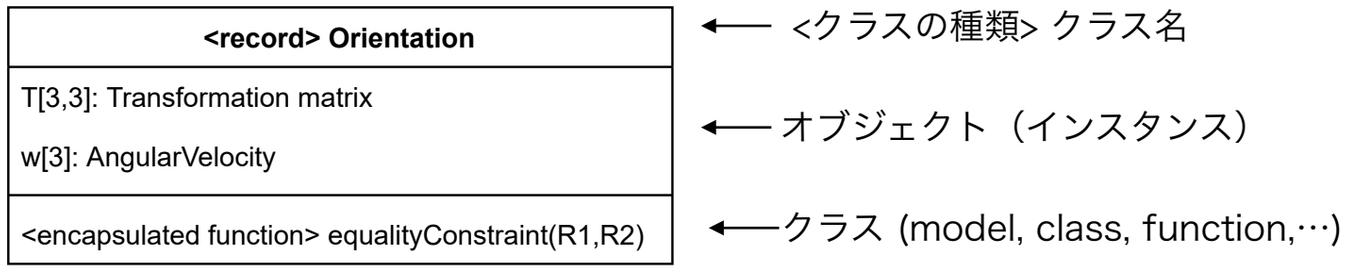
OMEdit
OMShell
OMPython
OMJulia

Frame — MultiBody システムのコネクタ

Frame



Orientation



$R.T = T[3,3]$ ワールド座標系からコネクタ座標系への変換行列

$R.\omega = w[3]$ コネクタ座標系で見た角速度

$$\{\phi_1, \phi_2, \phi_3\} = \text{equalityConstraint}(R_1, R_2)$$

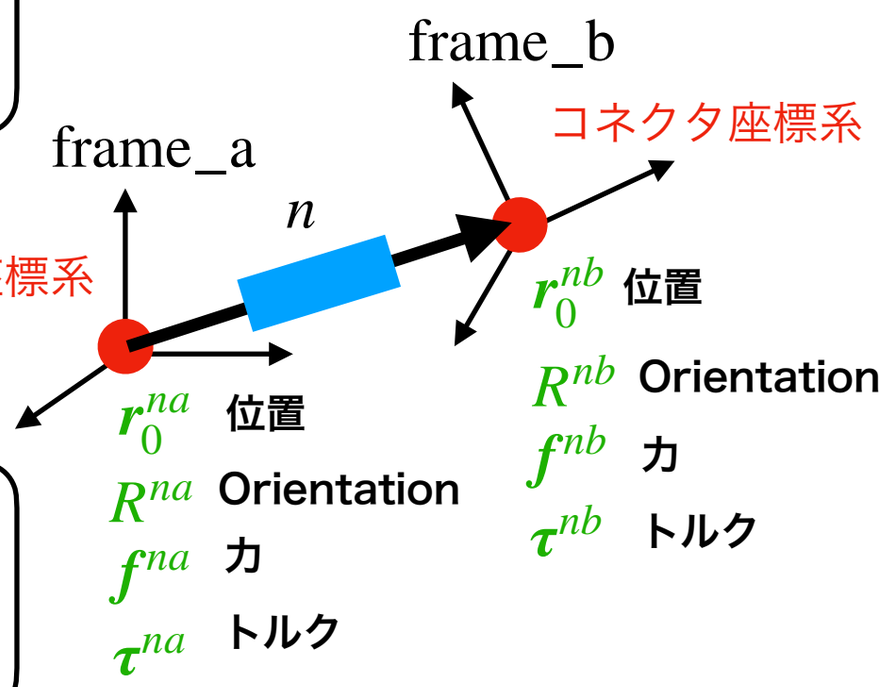
$r_0 = r[3]$ ワールド座標系’で見たコネクタ座標系の原点の位置ベクトル

flow $f = f[3]$ コネクタ座標系で見た力

flow $\tau = t[3]$ コネクタ座標系で見たトルク

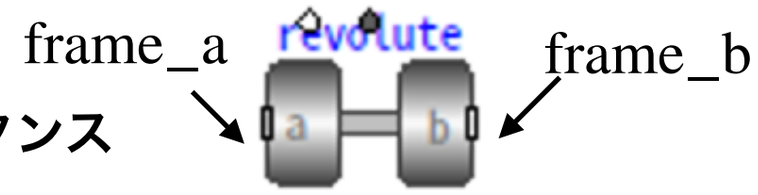
$R = \{R.T, R.\omega\}$ Orientation オブジェクト

コンポーネント



Revolute クラス（コンポーネント）

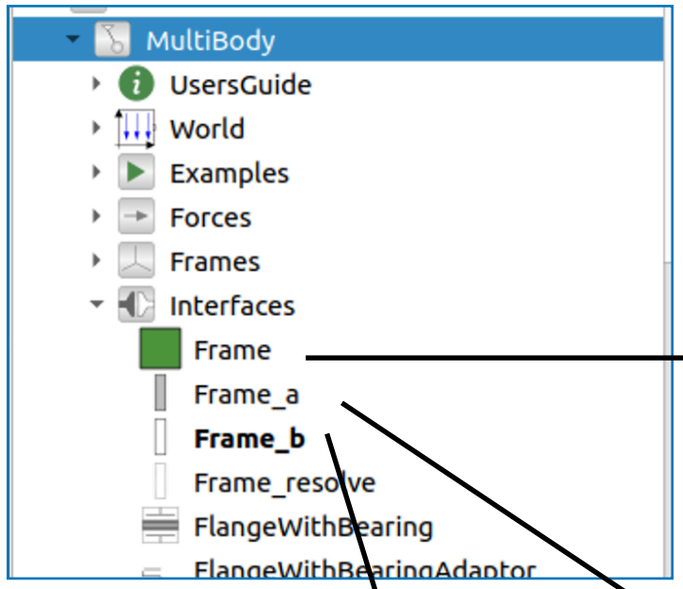
revolute インスタンス



添字 r_0^{na} ⟵ n番目のコンポーネントのコネクタ frame_a

⟵ ワールド座標から見たベクトルを表す。

下付きのないものはコネクタ座標系からみたベクトル



Modelica.Mechanics.Multibody.Interfaces.Frame

```
within Modelica.Mechanics.MultiBody.Interfaces;
connector Frame
  "Coordinate system fixed to the component with one cut-force and cut-torque (no icon)"

  SI.Position r_0[3]
    "Position vector from world frame to the connector frame origin, resolved in world frame";
  Frames.Orientation R
    "Orientation object to rotate the world frame into the connector frame";
  flow SI.Force f[3] "Cut-force resolved in connector frame" annotation (…);
end Frame;
```

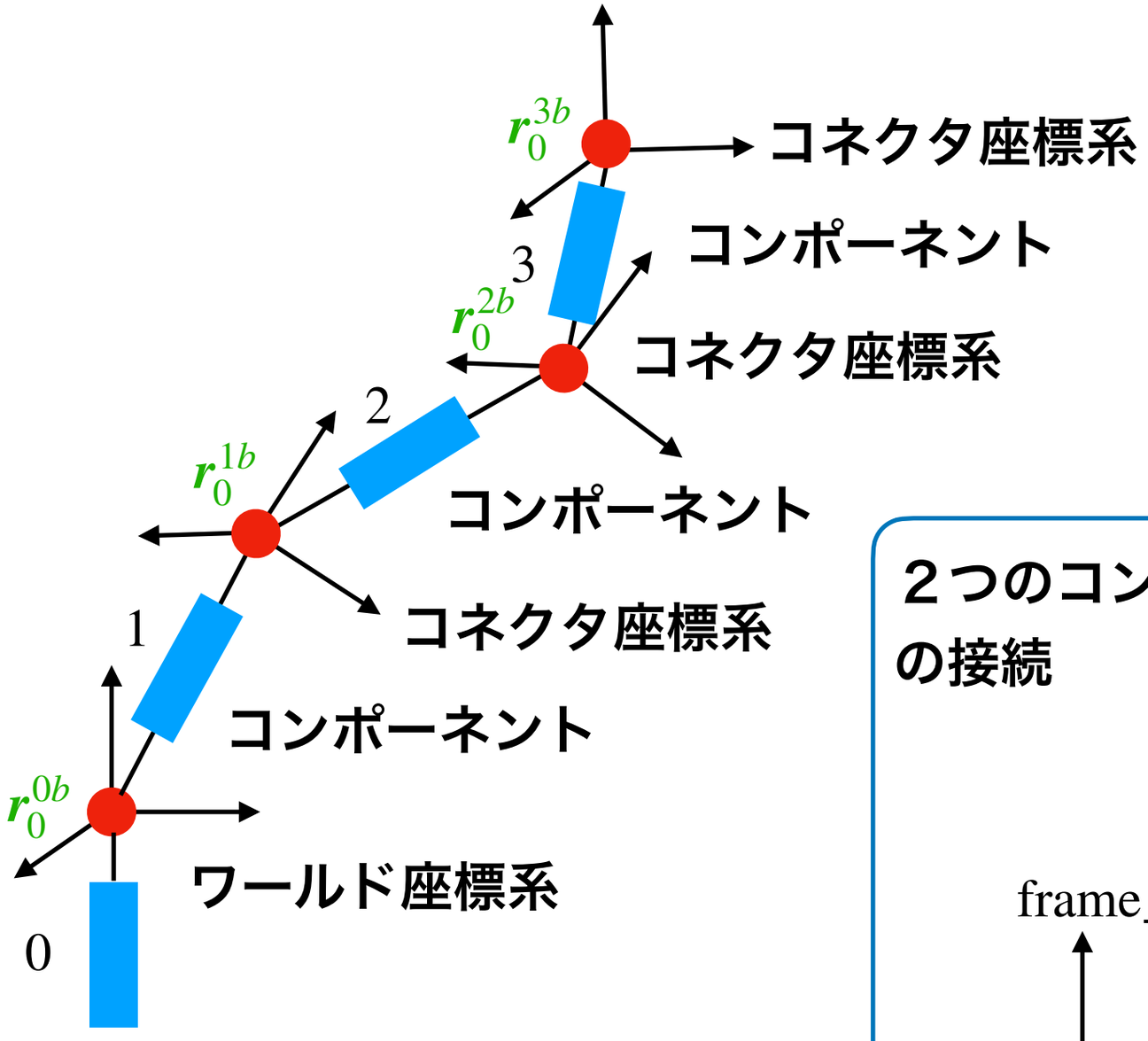
Modelica.Mechanics.Multibody.Interfaces.Frame_a

```
within Modelica.Mechanics.MultiBody.Interfaces;
connector Frame_a
  "Coordinate system fixed to the component with one cut-force and cut-torque (filled rectangular icon)"
  extends Frame;

  annotation (…);
end Frame_a;
```

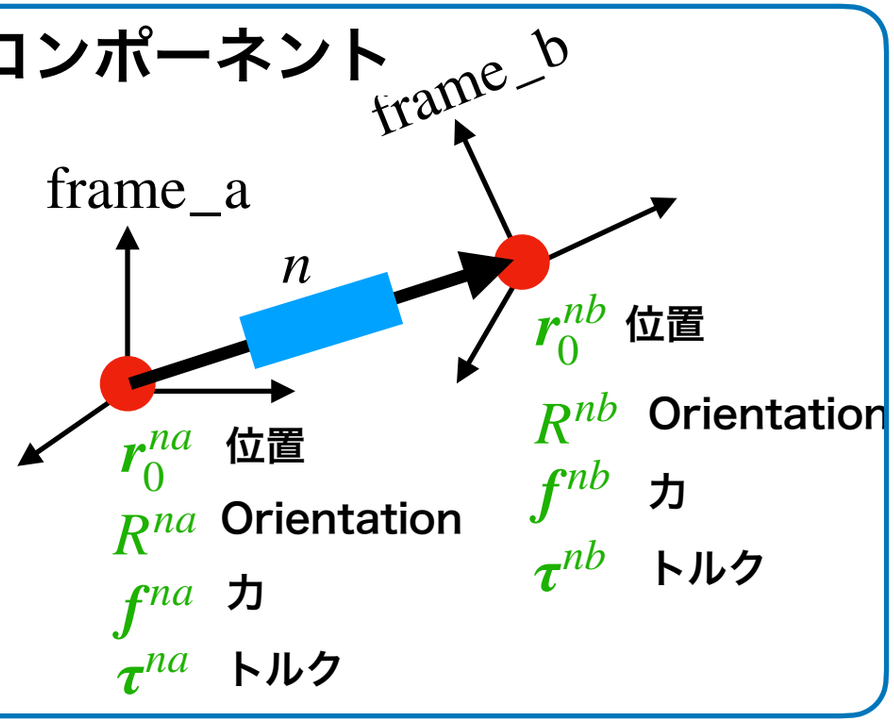
Modelica.Mechanics.Multibody.Interfaces.Frame_b

```
within Modelica.Mechanics.MultiBody.Interfaces;
connector Frame_b
  "Coordinate system fixed to the component with one cut-force and cut-torque (non-filled rectangular icon)"
  extends Frame;

  annotation (…);
end Frame_b;
```

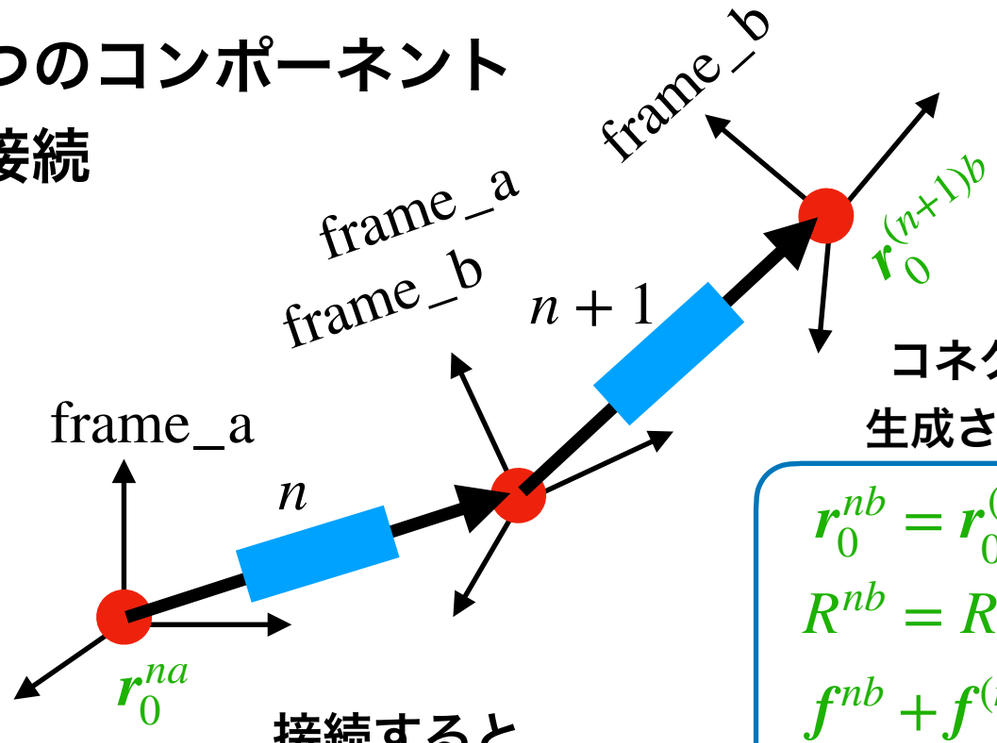
コンポーネントの接続のイメージ



コンポーネント



２つのコンポーネントの接続



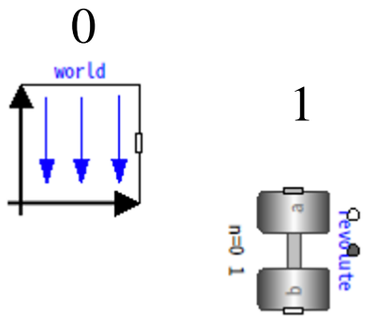
接続すると
コネクタ座標が同じになる

コネクタ接続で
生成される方程式

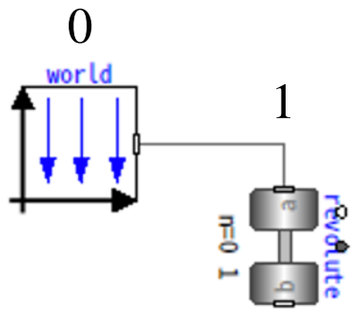
$$r_0^{nb} = r_0^{(n+1)a}$$
$$R^{nb} = R^{(n+1)a}$$
$$f^{nb} + f^{(n+1)a} = 0$$
$$\tau^{nb} + \tau^{(n+1)a} = 0$$

コネクタの接続の例（実際に生成される方程式の確認例）

接続前



接続後



接続によって生成される方程式

$$\boldsymbol{r}_0^{0b} = \boldsymbol{r}_0^{1a}$$
$$R^{0b} = R^{1a}$$
$$\boldsymbol{f}^{0b} + \boldsymbol{f}^{1a} = \boldsymbol{0}$$
$$\boldsymbol{\tau}^{0b} + \boldsymbol{\tau}^{1a} = \boldsymbol{0}$$

potential 変数 - 接続すると値が一致

$\mathrm{world.frame\_b}.R.T = \mathrm{revolute.frame\_a}.R.T$ 9個

$\mathrm{world.frame\_b}.R.\boldsymbol{\omega} = \mathrm{revolute.frame\_a}.R.\boldsymbol{\omega}$ 3個

$\mathrm{world.frame\_b}.\boldsymbol{r}_0 = \mathrm{revolute.frame\_a}.\boldsymbol{r}_0$ 3個

flow 変数 - 接続すると和がゼロ

$\mathrm{revolute.frame\_a}.\boldsymbol{f} + \mathrm{world.frame\_b}.\boldsymbol{f} = 0$ 3個

$\mathrm{revolute.frame\_a}.\boldsymbol{\tau} + \mathrm{world.frame\_b}.\boldsymbol{\tau} = 0$ 3個

計 21個

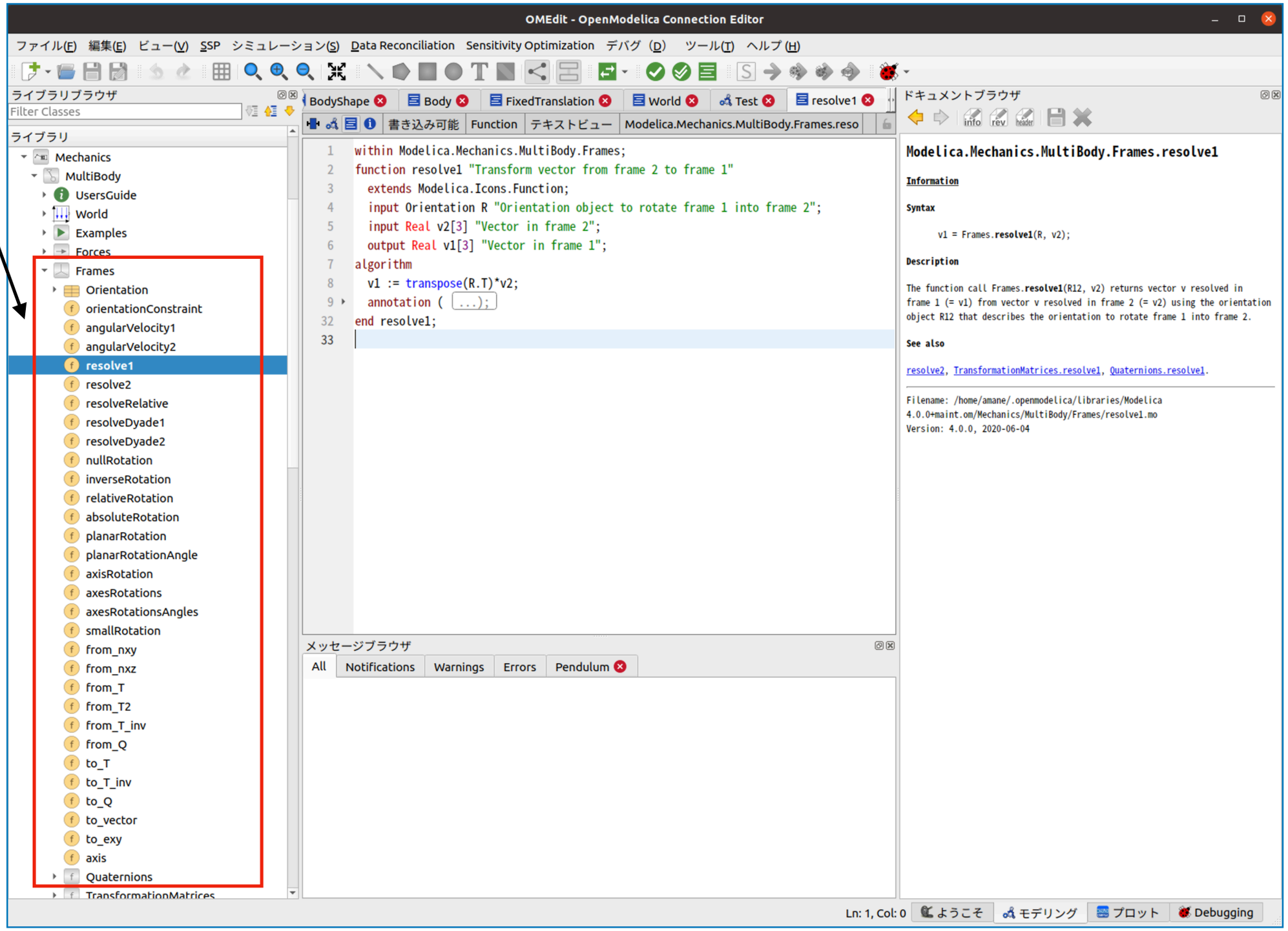
接続前と接続後の[モデルのインスタンス化]の結果の差分（diff)

```
amane@finback ~ % diff before.txt after.txt
341,352c341,361
<    world.frame_b.f[1] = 0.0;
<    world.frame_b.f[2] = 0.0;
<    world.frame_b.f[3] = 0.0;
<    world.frame_b.t[1] = 0.0;
<    world.frame_b.t[2] = 0.0;
<    world.frame_b.t[3] = 0.0;
<    revolute.frame_a.f[1] = 0.0;
<    revolute.frame_a.f[2] = 0.0;
<    revolute.frame_a.f[3] = 0.0;
<    revolute.frame_a.t[1] = 0.0;
<    revolute.frame_a.t[2] = 0.0;
<    revolute.frame_a.t[3] = 0.0;
---
>    world.frame_b.R.T[1,1] = revolute.frame_a.R.T[1,1];
>    world.frame_b.R.T[1,2] = revolute.frame_a.R.T[1,2];
>    world.frame_b.R.T[1,3] = revolute.frame_a.R.T[1,3];
>    world.frame_b.R.T[2,1] = revolute.frame_a.R.T[2,1];
>    world.frame_b.R.T[2,2] = revolute.frame_a.R.T[2,2];
>    world.frame_b.R.T[2,3] = revolute.frame_a.R.T[2,3];
>    world.frame_b.R.T[3,1] = revolute.frame_a.R.T[3,1];
>    world.frame_b.R.T[3,2] = revolute.frame_a.R.T[3,2];
>    world.frame_b.R.T[3,3] = revolute.frame_a.R.T[3,3];
>    world.frame_b.R.w[1] = revolute.frame_a.R.w[1];
>    world.frame_b.R.w[2] = revolute.frame_a.R.w[2];
>    world.frame_b.R.w[3] = revolute.frame_a.R.w[3];
>    world.frame_b.r_0[1] = revolute.frame_a.r_0[1];
>    world.frame_b.r_0[2] = revolute.frame_a.r_0[2];
>    world.frame_b.r_0[3] = revolute.frame_a.r_0[3];
>    revolute.frame_a.f[1] + world.frame_b.f[1] = 0.0;
>    revolute.frame_a.f[2] + world.frame_b.f[2] = 0.0;
>    revolute.frame_a.f[3] + world.frame_b.f[3] = 0.0;
>    revolute.frame_a.t[1] + world.frame_b.t[1] = 0.0;
>    revolute.frame_a.t[2] + world.frame_b.t[2] = 0.0;
>    revolute.frame_a.t[3] + world.frame_b.t[3] = 0.0;
401d409
<    assert(false, "Connector frame_a of revolute joint is not connected");
```

計 21個

座標系関連の関数

Modelica.Mechanics.MultiBody.Frames



座標変換

座標変換

変換行列 (translational matrix)

$$R^{na}.T = \begin{bmatrix} T_{11} & T_{21} & T_{31} \\ T_{21} & T_{22} & T_{32} \\ T_{31} & T_{32} & T_{33} \end{bmatrix}$$

コネクタ座標系への変換

$$\boldsymbol{r}^{na} = \text{Frames}.\text{resolve2}(R^{na}, \boldsymbol{r}_0^{na})$$
$$\boldsymbol{f}^{na} = \text{Frames}.\text{resolve2}(R^{na}, \boldsymbol{f}_0^{na})$$
$$\boldsymbol{\tau}^{na} = \text{Frames}.\text{resolve2}(R^{na}, \boldsymbol{\tau}_0^{na})$$
$$\boldsymbol{\omega}^{na} = \text{Frames}.\text{resolve2}(R^{na}, \boldsymbol{\omega}_0^{na})$$

$R^{na}.\boldsymbol{\omega}$ 角速度ベクトル

$$\boldsymbol{v} = (R^{na}.T)\boldsymbol{v}_0$$

ワールド座標への変換

$$\boldsymbol{r}_0^{na} = \text{Frames}.\text{resolve1}(R^{na}, \boldsymbol{r}^{na})$$
$$\boldsymbol{f}_0^{na} = \text{Frames}.\text{resolve1}(R^{na}, \boldsymbol{f}^{na})$$
$$\boldsymbol{\tau}_0^{na} = \text{Frames}.\text{resolve1}(R^{na}, \boldsymbol{\tau}^{na})$$
$$\boldsymbol{\omega}_0^{na} = \text{Frames}.\text{resolve1}(R^{na}, \boldsymbol{\omega}^{na})$$

$$\boldsymbol{v}_0 = (R^{na}.T)^T\boldsymbol{v}$$

Frame.resolve2

```
within Modelica.Mechanics.MultiBody.Frames;
function resolve2 "Transform vector from frame 1 to frame 2"
  extends Modelica.Icons.Function;
  input Orientation R "Orientation object to rotate frame 1 into frame 2";
  input Real v1[3] "Vector in frame 1";
  output Real v2[3] "Vector in frame 2";
algorithm
  v2 := R.T*v1;
  annotation ( …);
end resolve2;
```

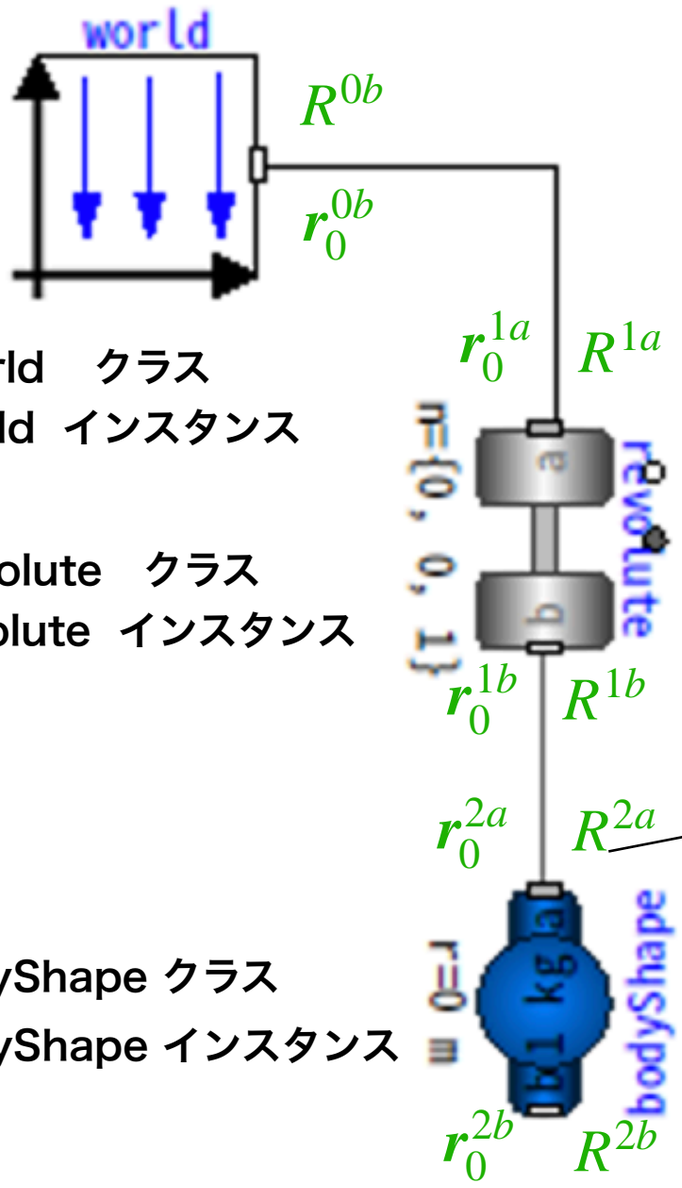
$$\boldsymbol{v}_2 = (R.T)\boldsymbol{v}_1$$

Frame.resolve1

```
within Modelica.Mechanics.MultiBody.Frames;
function resolve1 "Transform vector from frame 2 to frame 1"
  extends Modelica.Icons.Function;
  input Orientation R "Orientation object to rotate frame 1 into frame 2";
  input Real v2[3] "Vector in frame 2";
  output Real v1[3] "Vector in frame 1";
algorithm
  v1 := transpose(R.T)*v2;
  annotation ( …);
end resolve1;
```

$$\boldsymbol{v}_1 = (R.T)^T\boldsymbol{v}_2$$

座標の追跡



World　クラス
world　インスタンス

Revolute　クラス
revolute　インスタンス

BodyShape クラス
bodyShape インスタンス

ワールド座標で見たコネクタの位置

$$r_0^{0b} = \text{world . flame\_b . } r_0$$

$$r_0^{1a} = \text{revolute . frame\_a . } r_0$$
$$r_0^{1b} = \text{revolute . frame\_b . } r_0$$

$$r_0^{2a} = \text{bodyShape . frame\_a . } r_0$$
$$r_0^{2b} = \text{bodyShape . frame\_b . } r_0$$

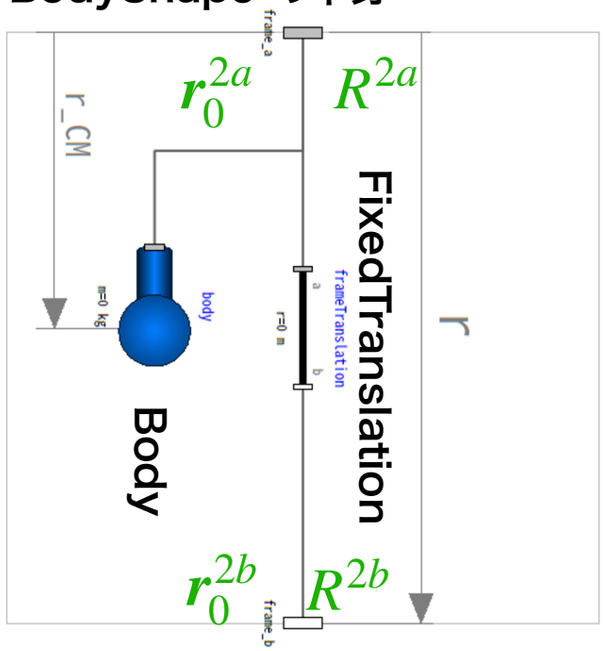
Orientation オブジェクト

$$R^{0b} = \text{world . frame\_b . } R$$

$$R^{1a} = \text{revolute . frame\_a . } R$$
$$R^{1b} = \text{revolute . frame\_b . } R$$

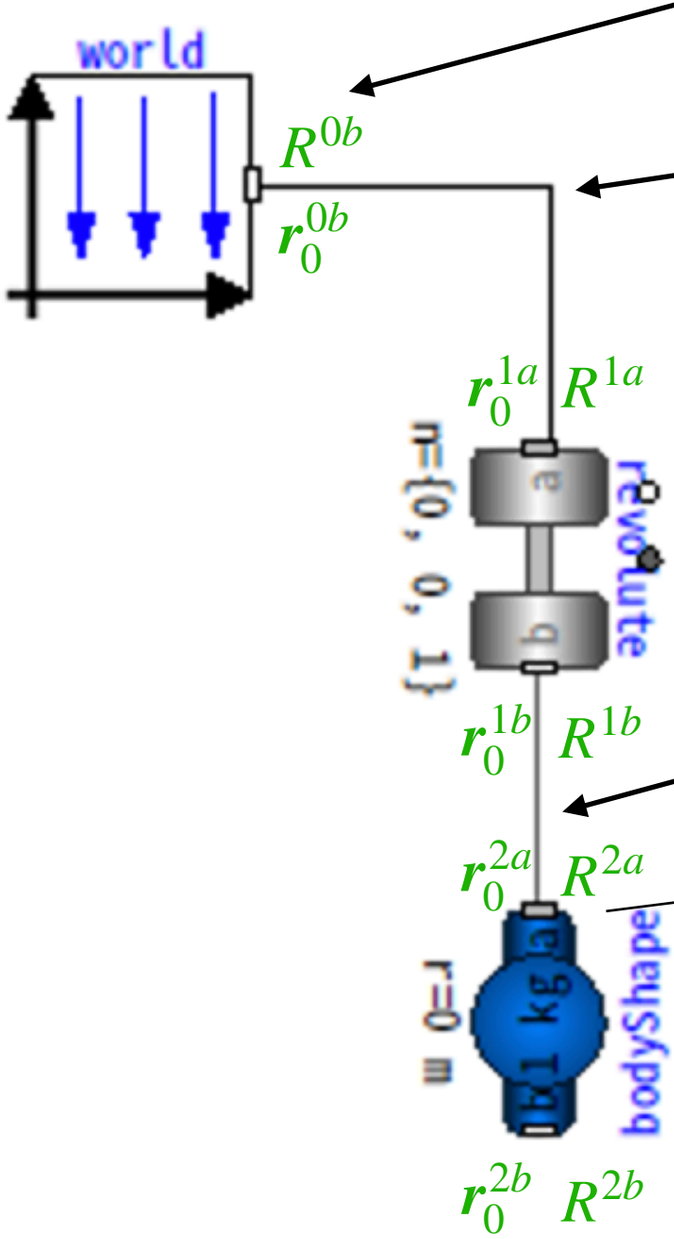
$$R^{2a} = \text{bodyShape . frame\_a . } R$$
$$R^{2b} = \text{bodyShape . frame\_b . } R$$

BodyShape の中身



Body と
FixedTransltation をまとめたもの

方程式を追っていくと…



world

$r_0^{0b} = \{0,0,0\}$
$R^{0b} = \text{nullRotation}()$

nullRotation()

$$R^{0b}.T = \begin{bmatrix} 1, & 0, & 0 \\ 0, & 1, & 0 \\ 0, & 0, & 1 \end{bmatrix}$$

$R^{0b}.\omega = \{0,0,0\}$

コネクタ接続

$r_0^{1a} = r_0^{0b}$
$R^{1a} = R^{0b}$

revolute

$r_0^{1b} = r_0^{1a}$
$e = \text{normalizedWithAssert}(n)$
$R_{rel} = \text{planarRotation}(e, \varphi, \omega)$
$R^{1b} = \text{absoluteRotation}(R^{1a}, R_{rel})$

$n = \{0,0,1\}$ 回転軸の方向ベクトル

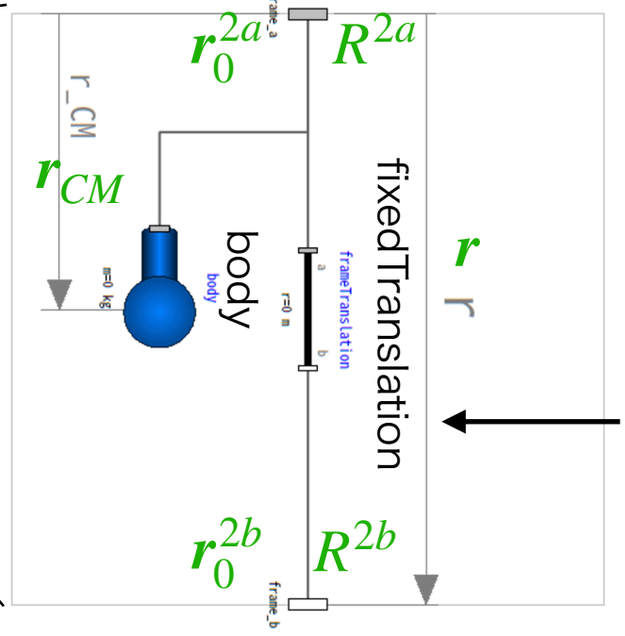
スカラー
- φ 回転角度
- $\omega = \dot{\varphi}$ 角速度
- $a = \dot{\omega}$ 角加速度

微分

コネクタ接続

$r_0^{2a} = r_0^{1b}$
$R^{2a} = R^{1b}$

bodyShape



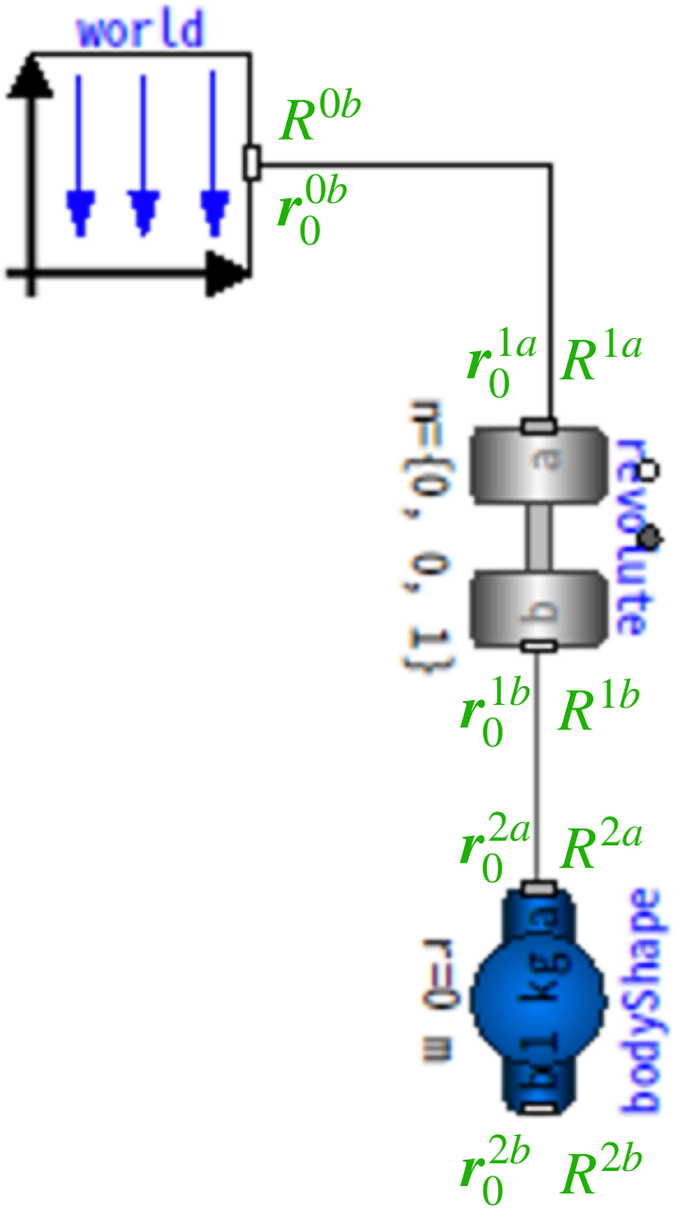
bodyShape

r_{CM} frame_a から重心までの位置ベクトル
r frame_a から frame_b までの位置ベクトル
$r = r_{CM} = \{0.5,0,0\}$

fixedTranslation

$r_0^{2b} = r_0^{2a} + \text{resolve1}(R^{2a}, r)$
$R_{2a} = R_{1b}$

まとめると…



$R^{0b} = R^{1a}$　　$R^{1b} = R^{2a} = R^{2b}$

φ

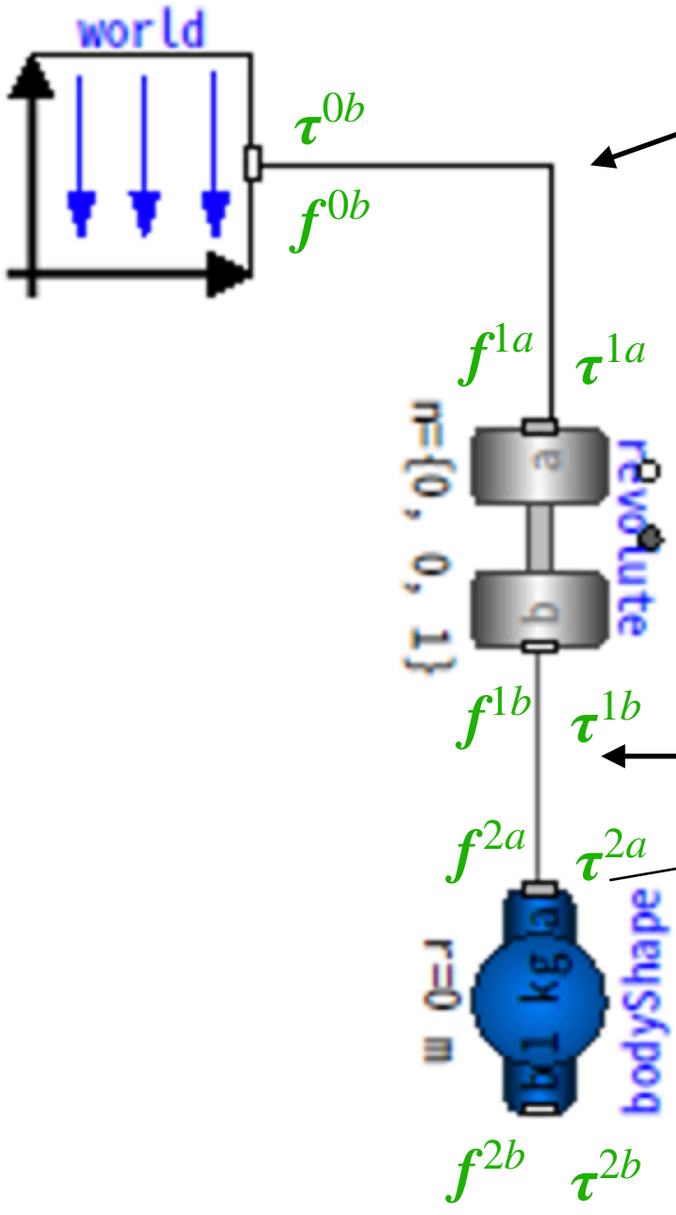
$\boldsymbol{n} = \{0,0,1\}$　　$\boldsymbol{r} = \boldsymbol{r}_{CM} = \{0.5,0,0\}$

回転軸　　(回転後の座標系でのベクトル)

$r_0^{0b} = r_0^{1a} = r_0^{1b} = r_0^{2a} = \{0,0,0\}$

$r_0^{2b} = r_0^{2a} + \mathrm{resolve1}(R^{2a}, \boldsymbol{r})$

力とトルクの追跡



コネクタ接続

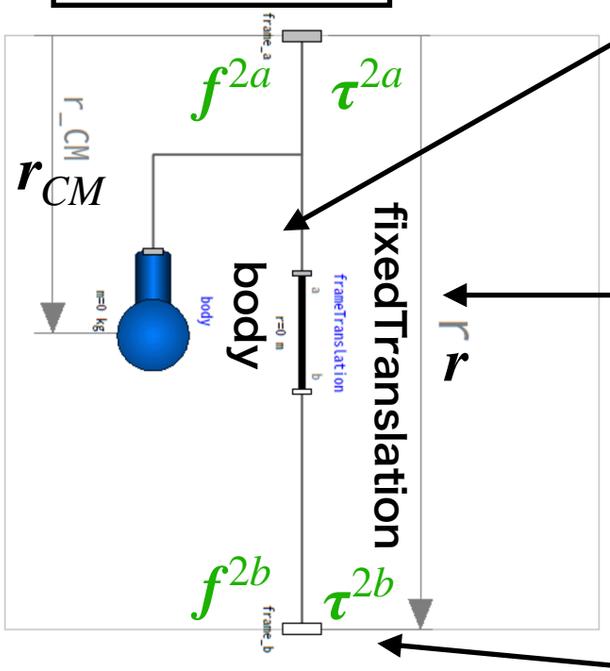
$$f^{0b} + f^{1a} = \mathbf{0}$$
$$\tau^{0b} + \tau^{1a} = \mathbf{0}$$

revolute

$$f^{1a} = -\,\mathrm{resolve1}(R_{rel}, f^{1b})$$
$$\tau^{1a} = -\,\mathrm{resolve1}(R_{rel}, \tau^{1b})$$
$$\tau = -\,\tau^{1b} \cdot \boldsymbol{e}$$
$$\tau = \begin{cases} 0, & \mathrm{useAxisFlange} = \mathit{false}, \\ \mathrm{axis}\,.\,\tau, & \mathrm{useAxisFlange} = \mathit{true} \end{cases}$$

コネクタ接続

$$f^{1b} + f^{2a} = \mathbf{0}$$
$$\tau^{1b} + \tau^{2a} = \mathbf{0}$$



body

$$r_{CM} = \{0.5,0,0\}$$
$$f^{2a} = m\left(a_a + z_a \times r_{CM} + \omega_a \times (\omega_a \times r_{CM})\right)$$
$$\tau^{2a} = Iz_a + \omega_a \times I\omega_a + r_{CM} \times f^{2a}$$

fixedTranslation

$$r = \{0.5,0,0\}$$
$$\mathbf{0} = f^{2a} + f^{2b}$$
$$\mathbf{0} = \tau^{2a} + \tau^{2b} + r \times f^{b}$$

コネクタ未接続

$$f^{2b} = \mathbf{0}$$
$$\tau^{2b} = \mathbf{0}$$

Body の力とトルクの方程式

コネクタ座標から見た値

$a_a = \mathrm{resolve2}(R^a, a_0 - g_0)$ frame_a の原点の加速度

$\omega_a = \mathrm{angularVelocity2}(R^a)$ 角速度

微分 → $z_a = \dot{\omega}_a$ 角加速度

ワールド座標絡みた値

$r_0 = r_0^a$ frame_a の原点の位置

$v_0 = \dot{r}_0$ 速度

$a_0 = \dot{v}_0$ 加速度

$g_0 = \mathrm{world.gravityAcceleration}(r_0^a + \mathrm{resolve1}(R^a, r_{CM}))$ 重心位置の重力加速度

$f^a = f_{CM}$ frame_aと重心の力のバランス方程式

$\tau^a = \tau_{CM} + r_{CM} \times f_{CM}$ frame_aと重心のトルクバランス方程式

$a_{CM} = a_a + \dot{\omega}_a \times r_{CM} + \omega_a \times (\omega_a \times r_{cm})$ 重心の加速度

$f_{CM} = m(a_{CM} - g_a)$ 重心の力

$\tau_{CM} = I\dot{\omega}_a + \omega_a \times I\omega_a$ 重心周りのトルク

重心周りの慣性テンソル

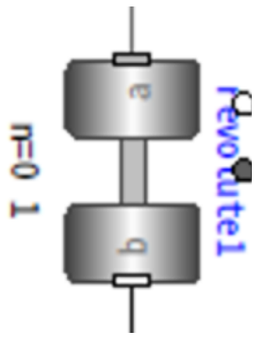
$$I = \begin{bmatrix} I_{11} & I_{21} & I_{31} \\ I_{21} & I_{22} & I_{32} \\ I_{31} & I_{32} & I_{33} \end{bmatrix}$$

$f^a = m\left(a_a + z_a \times r_{CM} + \omega_a \times (\omega_a \times r_{CM})\right)$ frame_a の力

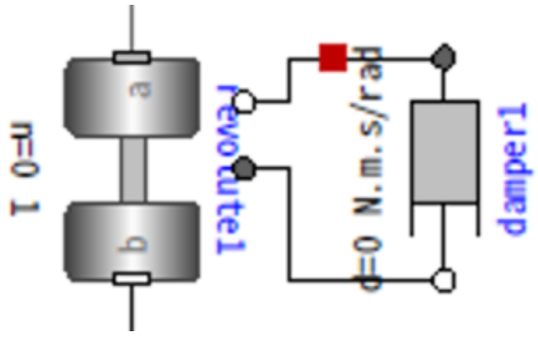
$\tau^a = Iz_a + \omega_a \times I\omega_a + r_{CM} \times f^a$ frame_a のトルク

Revolute のオプション①　useAxisFlange

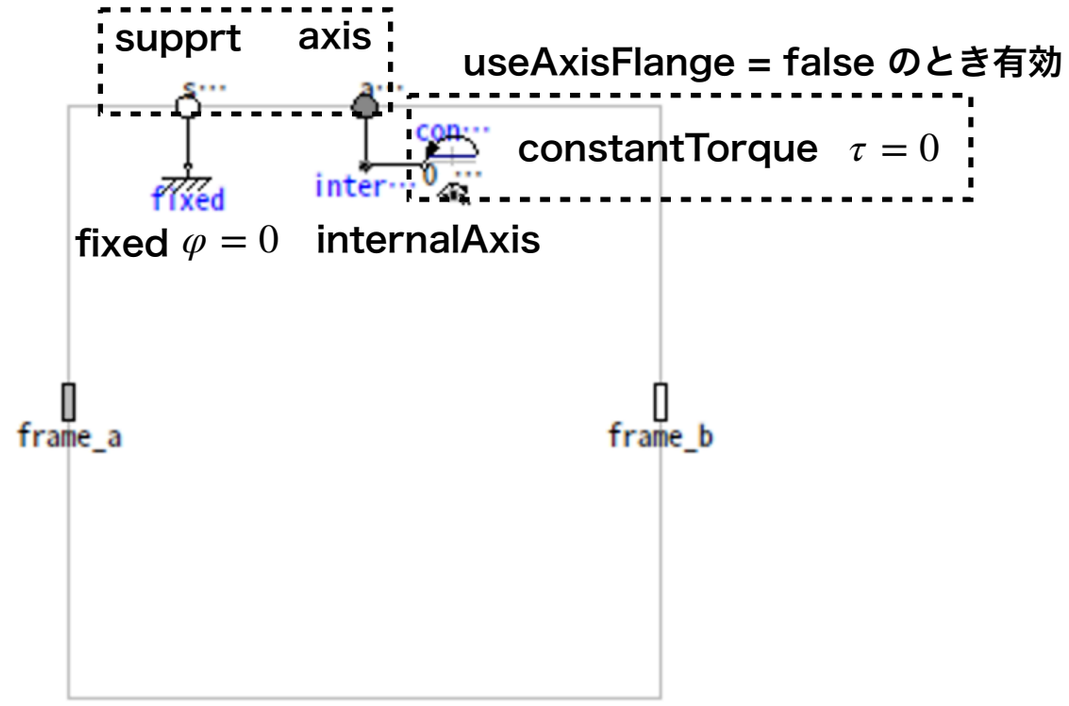
useAxisFlange = false の場合



useAxisFlange = true の場合



useAxisFlange = true のとき有効



Revolue の中身（ダイアグラムビュー）

useAxisFlange = true とした場合、コネクタ support と axis にModelica.Mechanics.Rotational のコンポーネントを接続して、トルク τ を制御することができる。false にした場合 $\tau = 0$

Revolute のオプション②　stateSelect

Modelica の State Selection (状態変数選択)

微分代数方程式の最も一般的な形
陰的微分方程式

$$\mathbf{F}(t, \mathbf{y}, \mathbf{y}') = 0$$

物理学的、工学的現象の数学的モデル

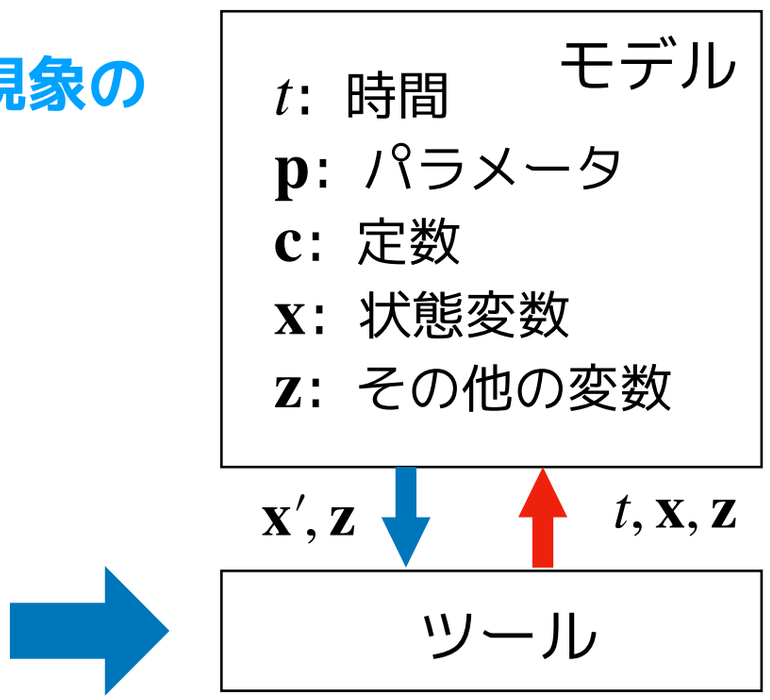
ツールによる translation

半陽的微分代数方程式
制約付きの常微分方程式

数値解析に適したモデル

$$\begin{cases} \mathbf{x}' &= \mathbf{f}(t, \mathbf{x}, \mathbf{z}) \\ \mathbf{0} &= \mathbf{g}(t, \mathbf{x}, \mathbf{z}) \end{cases}$$

微分方程式
代数方程式（制約条件）



ツールによる simulation

OpenModelicaなどのツールが
どの変数を状態変数(state variables)として微分方程式を構成するかを選択します。

例

$$x = y + c \quad \Leftrightarrow \quad y = x - c$$

同値

$$\frac{dy}{dt} = f(x, y) \quad \Leftrightarrow \quad \frac{dx}{dt} = f(x, y)$$

x, y のどちらに対して微分方程式を構成するかツールが選択します。

Revolute の stateSelect パラメータ

回転角度φや角速度ωを状態変数する選択のプライオリティを調整する。

```
 parameter StateSelect stateSelect=StateSelect.prefer
   "Priority to use joint angle phi and w=der(phi) as states" annotation(Dialog(tab="Advanced"));

 SI.Angle phi(start=0, final stateSelect=stateSelect)
   "Relative rotation angle from frame_a to frame_b"
    annotation ( …);
SI.AngularVelocity w(start=0, stateSelect=stateSelect)
   "First derivative of angle phi (relative angular velocity)";
```

stateSelect

実数変数(Real) に対するパラメータで、ソルバーが実数変数を状態変数として選択するプライオリティを設定する。言語仕様に含まれる。

Modelica Lanuage Specification 3.5, 4.8.7.1, p.57　https://modelica.org/documents/MLS.pdf

低

プライオリティ

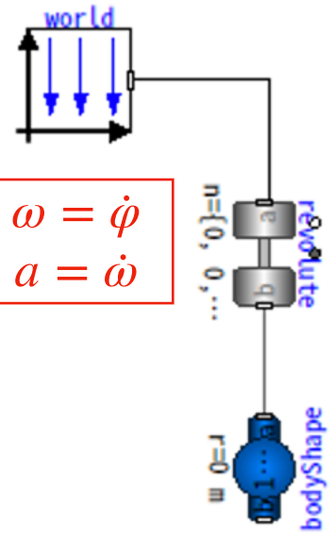
高

```
type StateSelect = enumeration(
  never "Do not use as state at all.” ,
  avoid "Use as state, if it cannot be avoided (but only if variable appears
         differentiated and no other potential state with attribute default,
         prefer, or always can be selected).” ,
  default "Use as state if appropriate, but only if variable appears differentiated.” ,
  prefer "Prefer it as state over those having the default value
         (also variables can be selected, which do not appear differentiated). ",
  always "Do use it as a state.”
);
```

φ や ω に対してデフォルトで通常より高いプライオリティに設定されていることになります。

振り子モデルで state (状態変数)の候補となる変数

方程式内で微分オペレータ(der)が付けられた変数



revolute.phi $\omega = \dot{\varphi}$

revolute.w $a = \dot{\omega}$

body.w_a $z_a = \dot{\omega}_a$

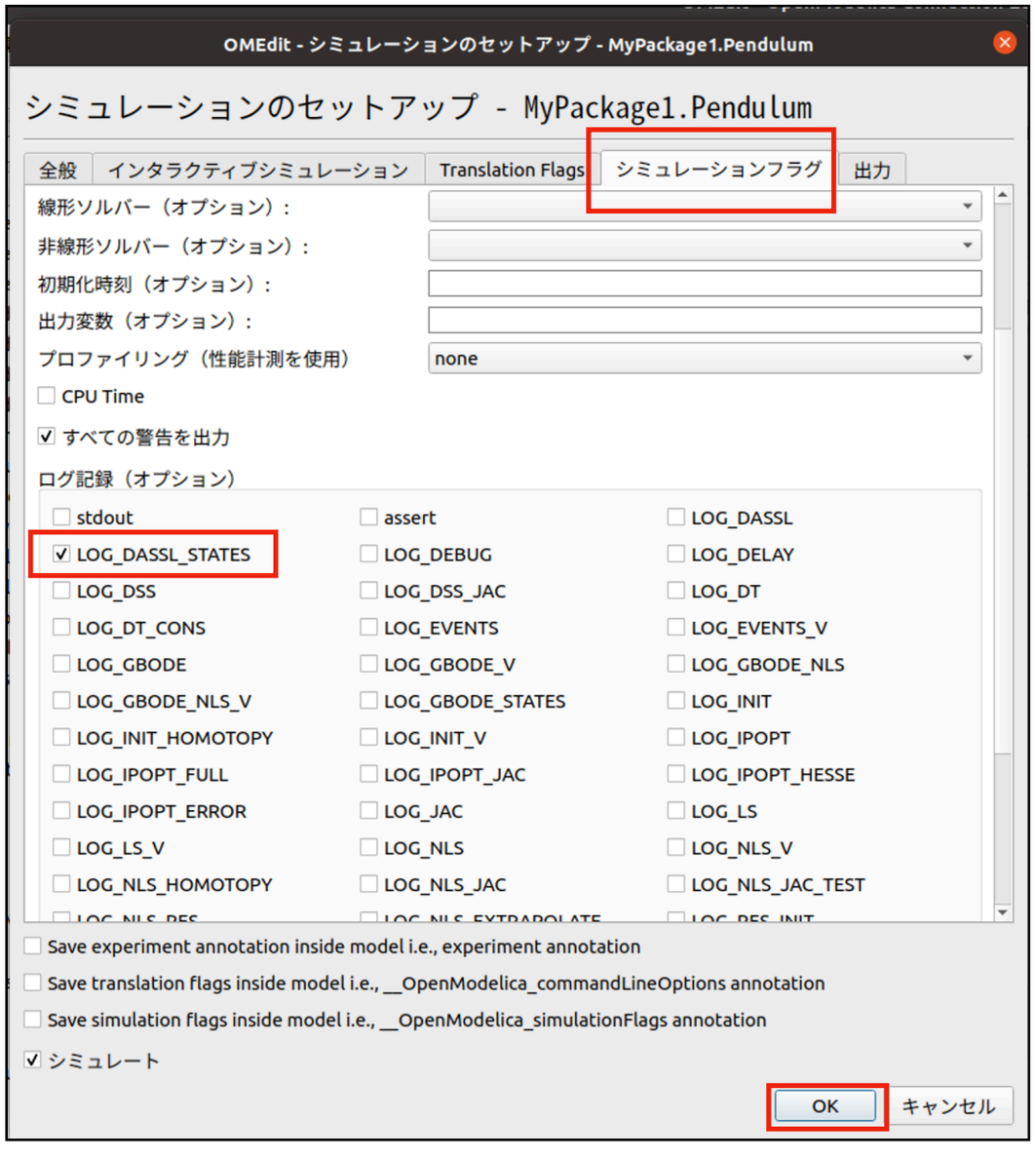
body.frame_a.r_0 $v_0 = \dot{r}_0$

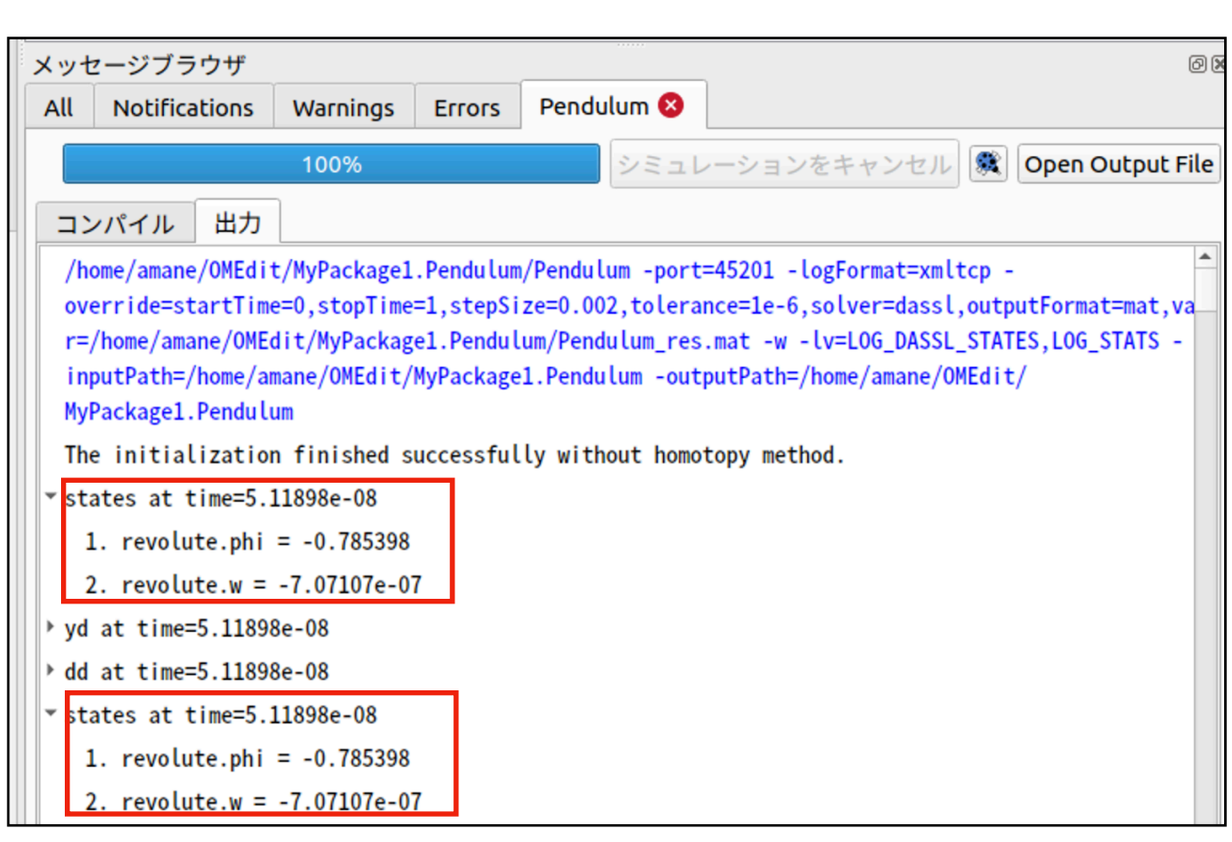
body.v_0 $a_0 = \dot{v}_0$

state (状態変数)の確認

シミュレーション > シミュレーションのセットアップ

シミュレーションフラグ LOG_DASSL_STATES をチェックするして実行する。





revolute.ph と revolute.w が
state (状態変数）になってることがわかる。

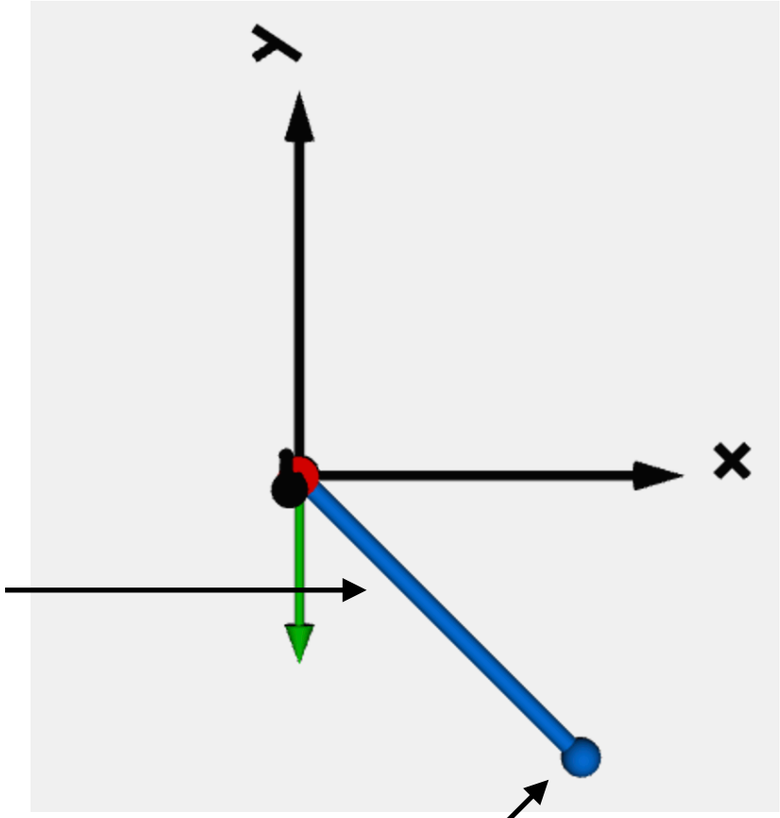
アニメーションのためのコード

BodyShapeのアニメーションに関するコード

```
protected
  outer Modelica.Mechanics.MultiBody.World world;
  Visualizers.Advanced.Shape shape1(
    shapeType=shapeType,
    color=color,
    specularCoefficient=specularCoefficient,
    length=length,
    width=width,
    height=height,
    lengthDirection=lengthDirection,
    widthDirection=widthDirection,
    r_shape=r_shape,
    extra=extra,
    r=frame_a.r_0,
    R=frame_a.R) if world.enableAnimation and animation;
  Visualizers.Advanced.Shape shape2(
    shapeType="sphere",
    color=sphereColor,
    specularCoefficient=specularCoefficient,
    length=sphereDiameter,
    width=sphereDiameter,
    height=sphereDiameter,
    lengthDirection={1,0,0},
    widthDirection={0,1,0},
    r_shape=r_CM - {1,0,0}*sphereDiameter/2,
    r=frame_a.r_0,
    R=frame_a.R) if world.enableAnimation and animation and animateSphere;
equation
```

shapeType = “cylinder”

円筒の描画



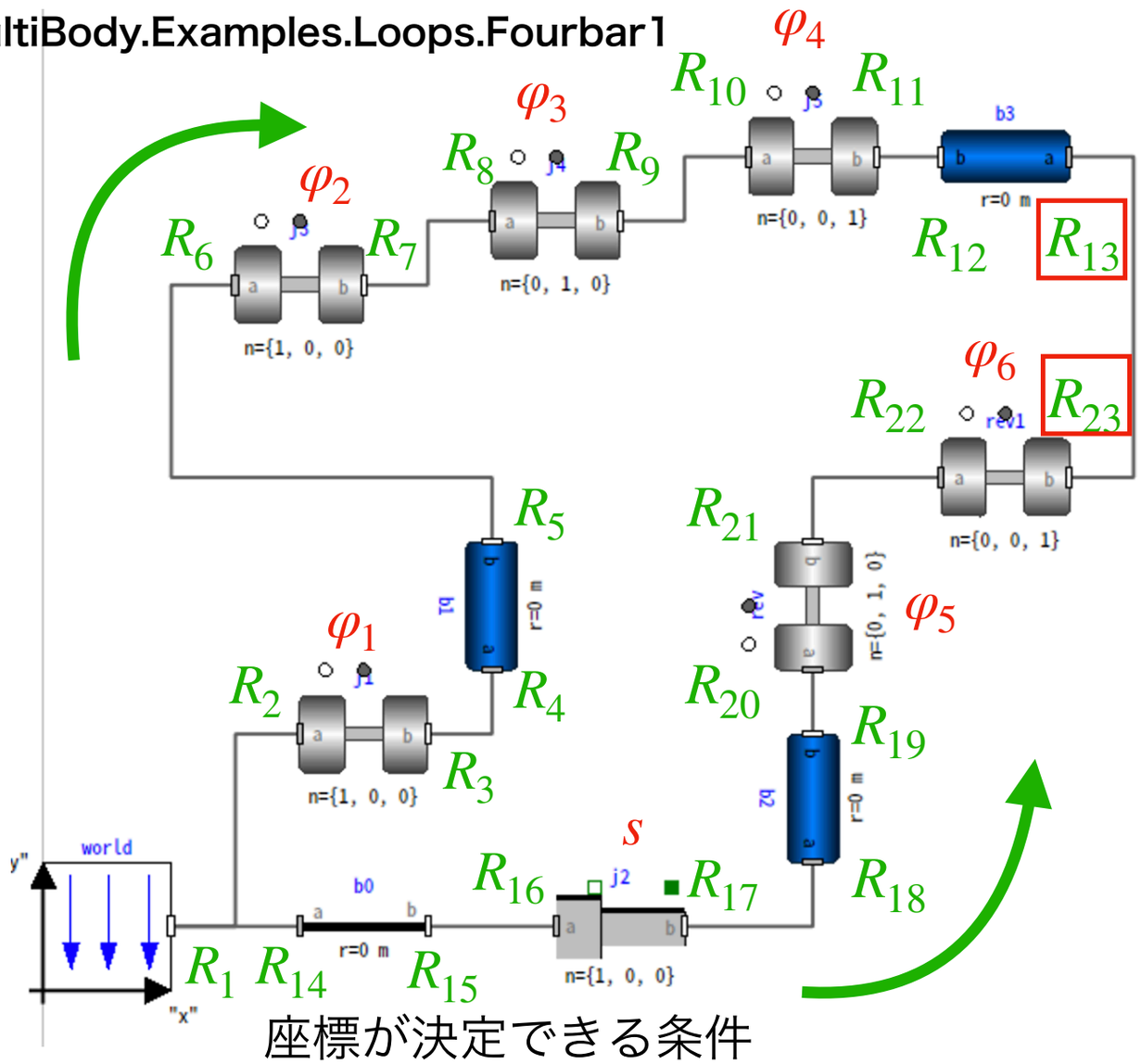
球の描画

Overconstrained Connection Loop 問題

overconstrained connection loop 制約条件が多すぎ

overdetermined DAEs　方程式が多すぎ

MultiBody.Examples.Loops.Fourbar1



座標が決定できる条件

$\varphi_1, \varphi_2, \varphi_3, \varphi_4, s, \varphi_5, \varphi_6$　７個の変数

右回りにたどる

$R_1 = \text{identity}(3)$
$R_2 = R_1$
$R_3 = R_{32}(\varphi_1)R_2$
$R_4 = R_3$
$R_5 = R_4$
$R_6 = R_5$
$R_7 = R_{76}(\varphi_2)R_6$
$R_8 = R_7$
$R_9 = R_{98}(\varphi_3)R_8$
$R_{10} = R_9$
$R_{11} = R_{11,10}(\varphi_4)R_{10}$
$R_{12} = R_{11}$
$R_{13} = R_{12}$

左回りにたどる

$R_1 = \text{identity}(3)$
$R_{14} = R_1$
$R_{15} = R_{14}$
$R_{16} = R_{15}$
$R_{17} = R_{17,16}(s)R_{16}$
$R_{18} = R_{17}$
$R_{19} = R_{18}$
$R_{20} = R_{19}$
$R_{21} = R_{21,20}(\varphi_5)$
$R_{22} = R_{21}$
$R_{23} = R_{23,22}(\varphi_6)R_{22}$

ループが閉じる条件

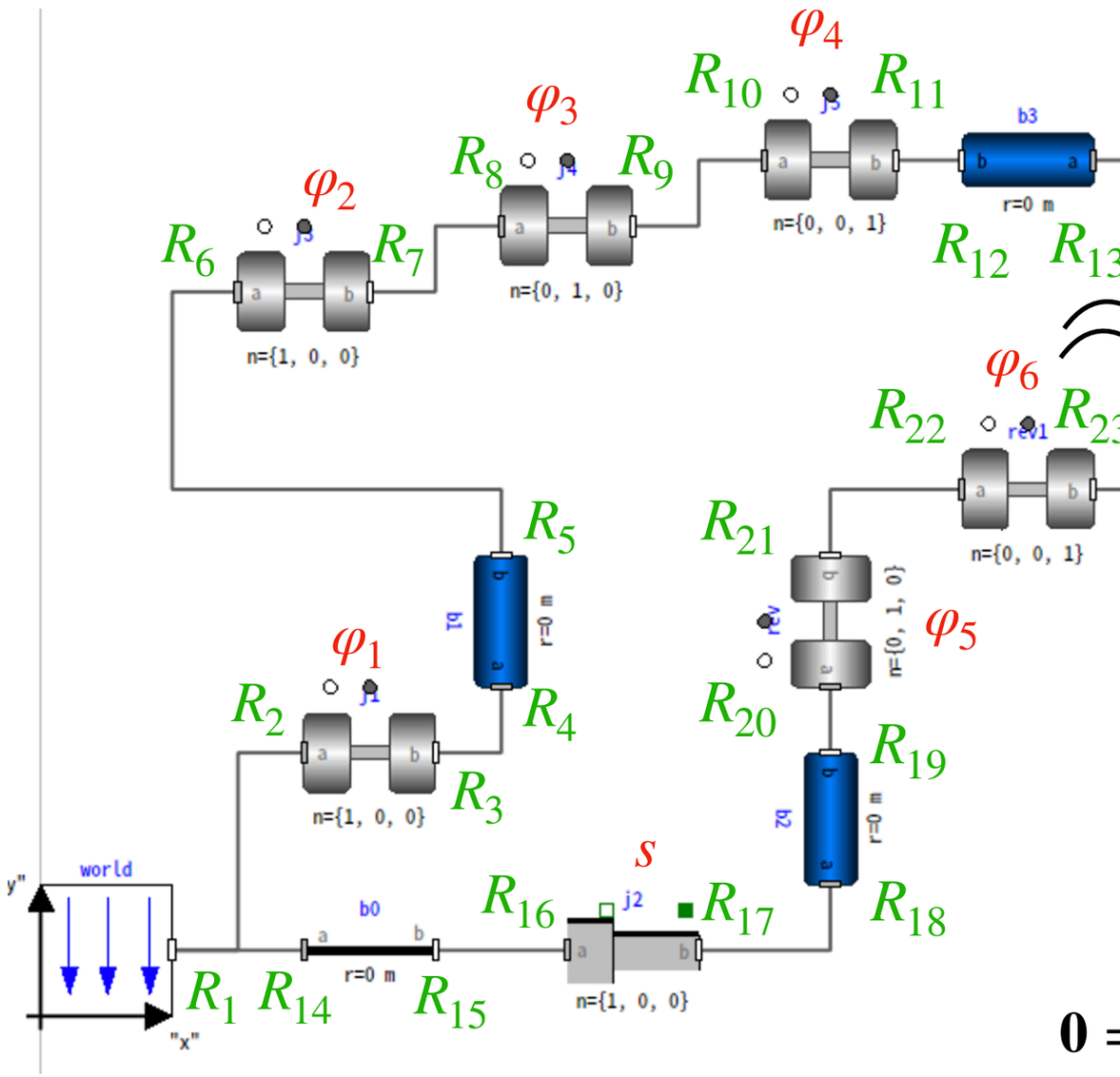
$R_{13} = R_{23}$

$\Rightarrow R_{13}.T[3.3] = R_{23}.T[3,3]$　９個の方程式

コンポーネントを接続してループができてしまうと…　　変数の数 < 方程式の数

Principals of Object-Oriented Modeling ans Simulation With Modelica 3.3, 2nd Ed., Peter Fritzson, Chapter8, 8.5

方程式を減らす



①ループを切断 (break) する。

②切断した部分のコネクタ接続の方程式置き換える

$R_{13}\,.\,T[3.3] = R_{23}\,.\,T[3,3]$　方程式９個

$\mathbf{0} = \text{equalityConstraint}(R_{13}, R_{23})$　方程式３個

equalityConstraint 関数

$$\{\varphi_1, \varphi_2, \varphi_3\} = \text{equalityConstraint}(R_1, R_2)$$

$$R_{rel}.T = (R_2.T)(R_1.T)^T \sim \begin{bmatrix} 1 & \varphi_3 & -\varphi_2 \\ -\varphi_3 & 1 & \varphi_1 \\ \varphi_2 & -\varphi_1 & 1 \end{bmatrix}$$

$$\mathbf{0} = \text{equalityConstraint}(R_1, R_2) \quad \Rightarrow R_1 \sim R_2$$

Modelica.Mechanics.MultiBody.Frames.Orientation.equalityConstraint

```
encapsulated function equalityConstraint
  "Return the constraint residues to express that two frames have the same orientation"

  import Modelica;
  import Modelica.Mechanics.MultiBody.Frames;
  extends Modelica.Icons.Function;
  input Frames.Orientation R1
    "Orientation object to rotate frame 0 into frame 1";
  input Frames.Orientation R2
    "Orientation object to rotate frame 0 into frame 2";
  output Real residue[3]
    "The rotation angles around x-, y-, and z-axis of frame 1 to rotate frame 1 into frame 2 for a small rotation (should be zero)";
algorithm
  residue := {
     Modelica.Math.atan2(cross(R1.T[1, :], R1.T[2, :])*R2.T[2, :],R1.T[1,:]*R2.T[1,:]),
     Modelica.Math.atan2(-cross(R1.T[1, :],R1.T[2, :])*R2.T[1, :],R1.T[2,:]*R2.T[2,:]),
     Modelica.Math.atan2(R1.T[2, :]*R2.T[1, :],R1.T[3,:]*R2.T[3,:])};
  annotation(…);
end equalityConstraint;
```

Connections オペレータ

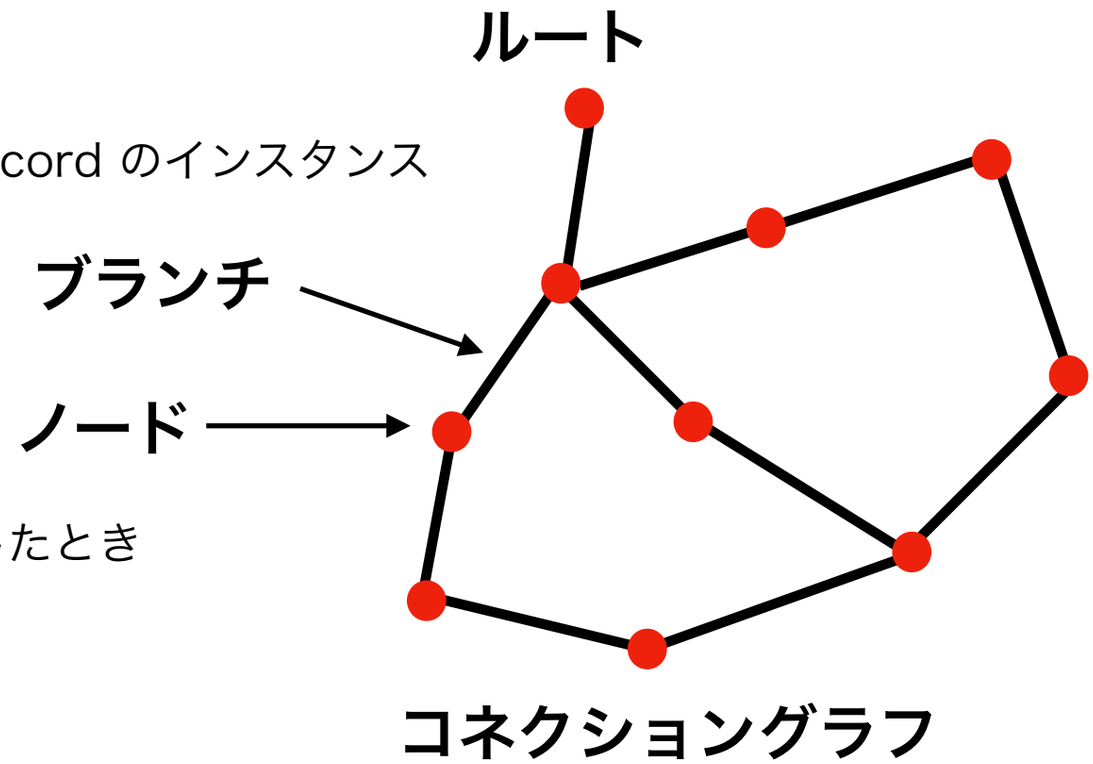
Overconstrained Connection Loop 問題を解決するため、Modelica 言語の仕様レベルで用意されたオペレータ

R : overdetermined 変数

equalityConstraint 関数が定義された type または record のインスタンス

A, B : overdetermined コネクタ

R を要素にもつ コネクタ



$\mathrm{Connections.root}(A.R)$

コネクショングラフのルートを定義する。ツリーに分解したときルート（根元）になる。

$A.R$ が、パラメータなどで設定されている必要がある。

$\mathrm{Connections.branch}(A.R, B.R)$

コネクショングラフの　non-breakable branch (切断できないブランチ）を定義する。

$A.R$ と $B.R$ の間に代数方程式 (algebraic equation) が存在する。

$B.R = f(A.R, \cdots)$ のような関係になる。

connect オペレータの位置に置くことができる。

$\mathrm{connect}(A, B)$

コネクショングラフの breakable branch (切断可能なブランチ）をあらわす。

コネクタ間の標準的な接続に用いる。

$$\text{Connections . potentialRoot}(A\,.\,R, priority)$$

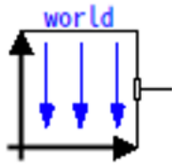
Connections.root()が設定されていないコネクションサブクラスでは、最も priority の数値が低いノードがルートとして選択される。
priority は非負の整数でデフォルトはゼロ。
der(A.R)が現れるところには設定する。。

$$b = \text{Connections . isRoot}(A\,.\,R)$$

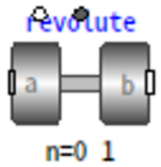
$A\,.\,R$ がルートとして選択されているかを返す。

$$b = \text{Connections . Rooted}(A\,.\,R)$$

$\text{Connections . branch}(A\,.\,R, B\,.\,R)$ とともに使う。$B\,.\,R$とくらべて$A\,.\,R$ の方がスパニングツリーのルートに近いときtrue を返す。

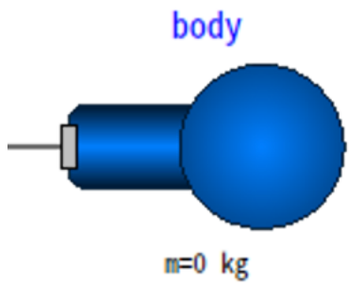



```
equation
  Connections.root(frame_b.R);

  assert(Modelica.Math.Vectors.length(n) > 1e-10,
    "Parameter n of World object is wrong (length(n) > 0 required)");
  frame_b.r_0 = zeros(3);
  frame_b.R = Frames.nullRotation();
  annotation (…);
end World;
```




```
equation
  Connections.branch(frame_a.R, frame_b.R);
…
 frame_b.r_0 = frame_a.r_0;

  if Connections.rooted(frame_a.R) then
    R_rel = Frames.planarRotation(e, phi, w);
    frame_b.R = Frames.absoluteRotation(frame_a.R, R_rel);
    frame_a.f = -Frames.resolve1(R_rel, frame_b.f);
    frame_a.t = -Frames.resolve1(R_rel, frame_b.t);
  else
    R_rel = Frames.planarRotation(-e, phi, w);
    frame_a.R = Frames.absoluteRotation(frame_b.R, R_rel);
    frame_b.f = -Frames.resolve1(R_rel, frame_a.f);
    frame_b.t = -Frames.resolve1(R_rel, frame_a.t);
  end if;
…
end Revolute;
```

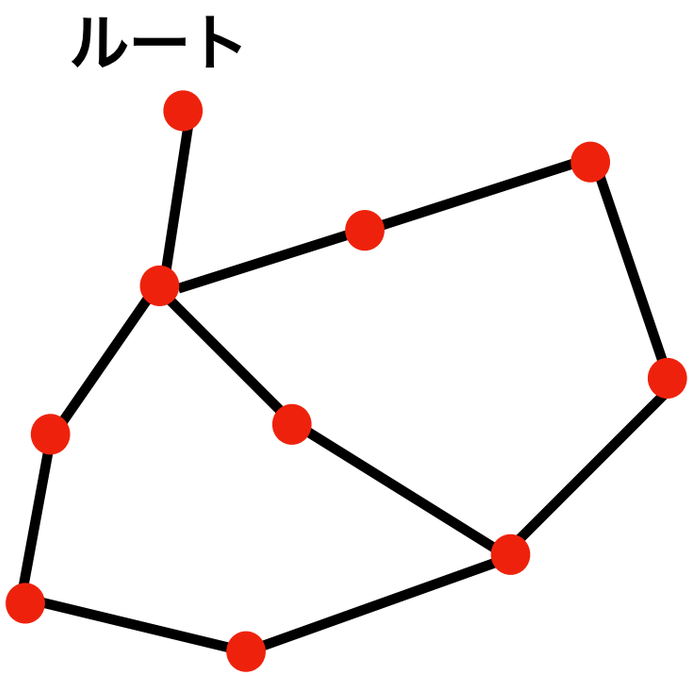

```
equation
  Connections.branch(frame_a.R, frame_b.R);
  assert(cardinality(frame_a) > 0 or cardinality(frame_b) > 0,
    "Neither connector frame_a nor frame_b of FixedTranslation object is connected");

  frame_b.r_0 = frame_a.r_0 + Frames.resolve1(frame_a.R, r);
  frame_b.R = frame_a.R;

  /* Force and torque balance */
  zeros(3) = frame_a.f + frame_b.f;
  zeros(3) = frame_a.t + frame_b.t + cross(r, frame_b.f);
  annotation (…);
nd FixedTranslation;
```



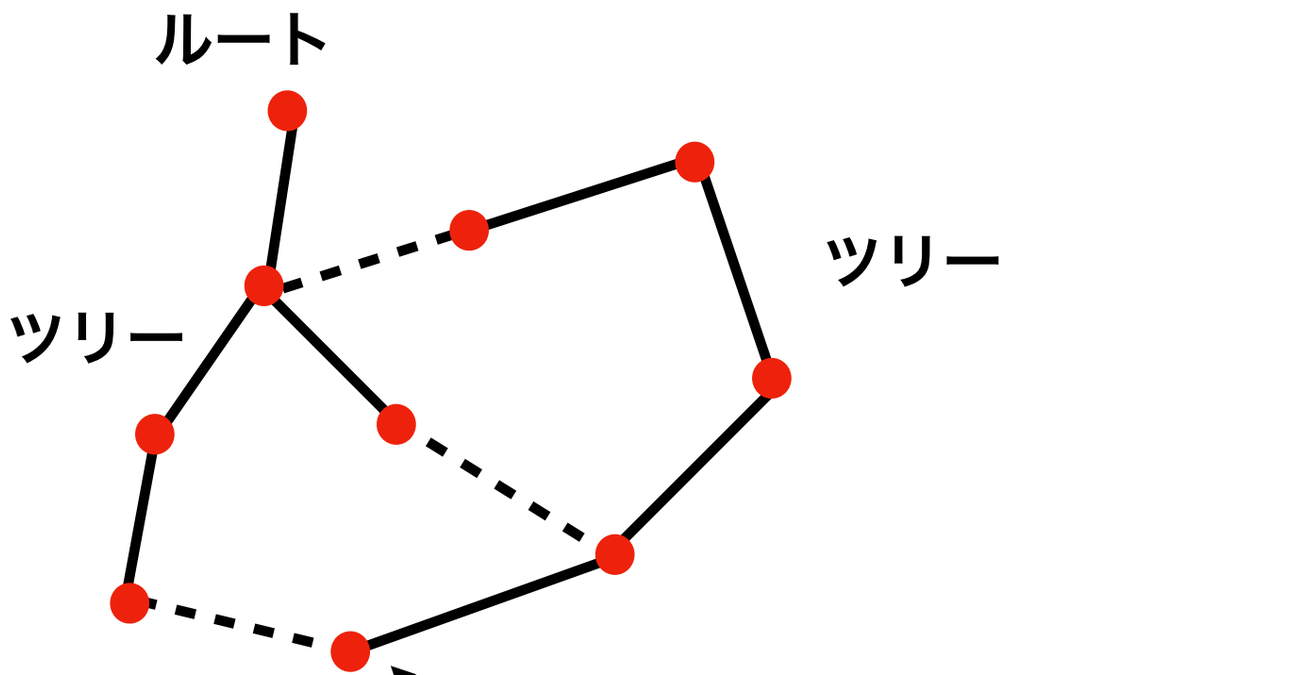

```
equation
  if enforceStates then
    Connections.root(frame_a.R);
  else
    Connections.potentialRoot(frame_a.R);
  end if;
  r_0 = frame_a.r_0;

  if not Connections.isRoot(frame_a.R) then
    // Body does not have states
    // Dummies
    Q = {0,0,0,1};
    phi = zeros(3);
    phi_d = zeros(3);
    phi_dd = zeros(3);
  elseif useQuaternions then
    // Use Quaternions as states (with dynamic state selection)
    frame_a.R = Frames.from_Q(Q, Frames.Quaternions.angularVelocity2(Q, der(Q)));
    {0} = Frames.Quaternions.orientationConstraint(Q);

    // Dummies
    phi = zeros(3);
    phi_d = zeros(3);
    phi_dd = zeros(3);
  else
    // Use Cardan angles as states
    phi_d = der(phi);
    phi_dd = der(phi_d);
    frame_a.R = Frames.axesRotations(
        sequence_angleStates,
        phi,
        phi_d);

    // Dummies
    Q = {0,0,0,1};
  end if;
...
end Body;
```

コネクショングラフの切断と方程式の変換

- コネクテョングラフがツリーに分解されるまで、connect(A, B) のブランチを切断する。
- ツリーが得られたら、ツリーの中のブランチには通常のルールの方程式を生成する。
- 切断されたブランチの方程式は、 $\mathbf{0} = \text{equalityConstraint}(\text{A . R}, \text{B . R})$ に変換する。R が通常のタイプであれば通常のコネクタ接続の方程式を生成する。
- Connections . root()が定義されているノードはすべてルートとする。
- コネクショングラフが未接続のサブグラフの集合の場合、少なくとも 1 個は、Connections . Root() または Connections . potentialRoot() が定義されたノードが無くてはならない。
- サブグラフに Connection . root() が定義されたノードがない場合は、Connections . pottentialRoot() が定義されたノードのうち最も *priority* の数値が少ないノードがルートとして選択する。



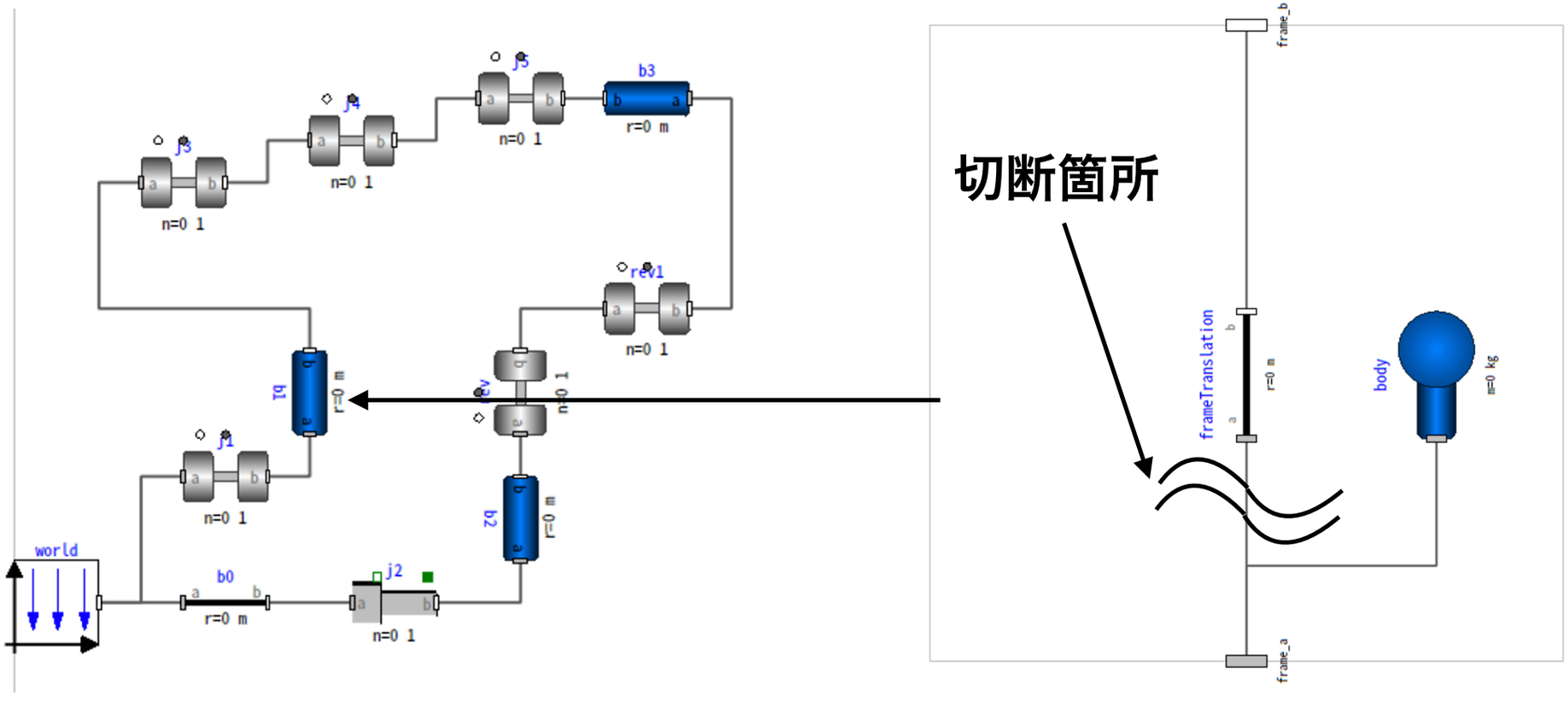
コネクショングラフ



ループの切断箇所の確認

Modelica.Mechanics.MultiBody.Examples.Loop.Fourbar1でモデルのインスタンス化を実行する。

equalityConstraint を検索する



参考文献

Principals of Object-Oriented Modeling ans Simulation With Modelica 3.3, 2nd Ed., Peter Fritzson, Chapter8, 8.5

The New Modelica MultiBody Library, Proc. 3rd International Modelica Conference, Peter Fritzson, 2003.
https://modelica.org/events/Conference2003/papers/h37_Otter_multibody.pdf