Lecture 12: Hybrid MPC

I. Modeling of Hybrid Systems: Introduction and Overview

- **Initial Remarks: Notations for Hybrid Systems**

Up to now, we mainly discussed discrete-time linear systems with linear constraints.

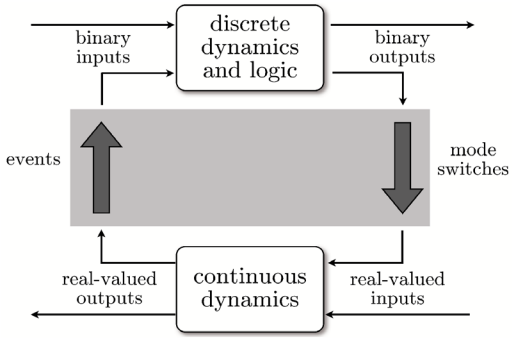
We now consider MPC for systems with:

1. Continuous dynamics: described by one or more difference (or differential) equations; states are continuous-valued.
2. Discrete events: state variables assume discrete values, e.g. Binary digits $\{0,1\}$; $\mathbb{N},\mathbb{Z},\mathbb{Q},\ldots$; finite set of symbols, etc.

And we have the definition for Hybrid systems:

Definition (Hybrid System):

1. Dynamical systems whose state evolution depends on an interaction between continuous dynamics and discrete events.
2. **Logic-based discrete dynamics** and continuous dynamics interact through events and mode switches



- **Examples of Hybrid Systems**

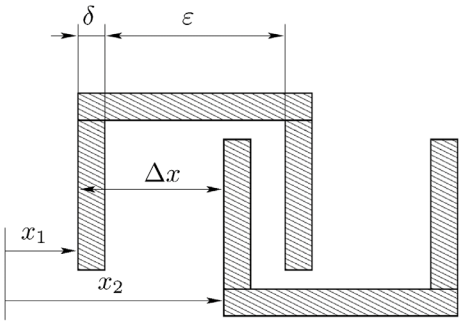
Mechanical System with Backlash

Continuous dynamics: state $x_1, x_2, \dot{x}_1, \dot{x}_2$

Discrete events:

(a) "contact mode": mechanical parts are in contact and the force is transmitted. The condition is:
$[(\Delta x = \delta) \wedge (\dot{x}_1 > \dot{x}_2)] \vee [(\Delta x = \varepsilon) \wedge (\dot{x}_1 < \dot{x}_2)]$

(b) "backlash mode": mechanical parts are not in contact

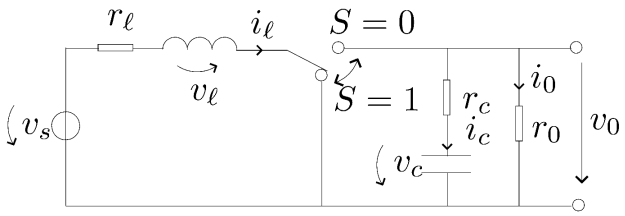


DCDC Converter

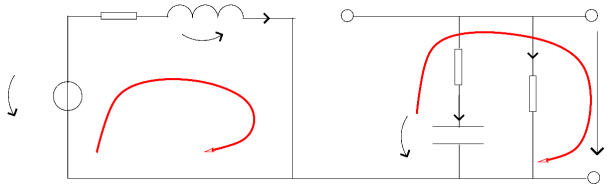
Continuous dynamics: state $v_\ell, i_\ell, v_c, i_c, v_0, i_0$

Discrete events: $S=0, S=1$

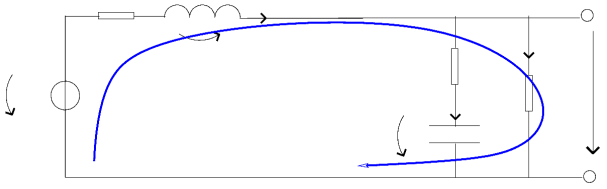
The two modes are shown below:



Mode 1 ($S=1$)



Mode 2 ($S=0$)



- **Piecewise Affine (PWA) Systems**

Piecewise affine systems can approximate nonlinear dynamics arbitrarily well (even the nonlinearity caused by discontinuity), they are the simplest representation that can capture the hybrid nature.

Definition (Piecewise Affine System):

Piecewise affine systems are defined by

Affine dynamics and output in each region:

$$\begin{cases} x(t+1) = A_i x(t) + B_i u(t) + f_i \\ y(t) = C_i x(t) + D_i u(t) + g_i \end{cases} \quad \text{if } (x(t), u(t)) \in \mathcal{X}_i(t)$$

and the polyhedral partition of the (x, u) space:

$$\{\mathcal{X}_i\}_{i=1}^{s} = \{(x.u) \mid H_i x + J_i u \leq K_i\}$$

with $x \in \mathbb{R}^n, u \in \mathbb{R}^m$, and physical constraints on $x(t)$ and $u(t)$ are defined by polyhedra $\mathcal{X}_i$

Definition (Well-Posedness):

Let P be a PWA system and let $\mathcal{X} = \cup_{i=1}^{s} \mathcal{X}_i \subseteq \mathbb{R}^{n+m}$ be the polyhedral partition associated with it. System P is called well-posed if, for all pairs $(x(t), u(t)) \in \mathcal{X}$ there exists only one index $i(t)$ satisfying the membership condition.

Note: That is to say, we hope that the partitions of the PWA system's state space have no overlaps. Apart from well-posedness (no overlaps). We also hope that there are no "holes" in the partitions of the state space, since we do not want to go to a region where control behaviors are not defined.

PWA System Examples

1. linearization of a non-linear system at different operating points: useful as an approximation tool
2. closed-loop MPC system for linear constrained systems (Recall that in the last lecture, we just proved that the controller is PPWA, so it is not hard to see this)
3. When the mode i is an exogenous variable, the partition disappears (i.e. mode is determined externally and assigned as an input, instead of decided internally through the system's space region partition), and we refer to the system as a **Switched Affine System (SAS)**

Note: In the above example and definition for PWA systems, they have all the states and inputs to be continuous. Then why do we say that PWA systems can represent hybrid systems? Actually, the PWA systems can be hybrid because they **have different dynamics in different partitions**.

- **Binary States, Inputs, and Output and Boolean Algebra**

Therefore, the problem comes: how should we mathematically describe such a hybrid nature (different partitions)? Our solution is: formulating PWA systems **including binary state and inputs by treating 0-1 binary variables as**:

Boolean variables p, over which boolean functions are defined (do logic descriptions)

Integer numbers δ, over which arithmetic operations are defined (do numerical translations)

We will use the notation as follows, where c for continuous and ℓ for logic (boolean):

$$x=\begin{bmatrix}x_c\\x_\ell\end{bmatrix}\in\mathbb{R}^{n_c}\times\{0,1\}^{n_\ell},\ n=n_c+n_\ell \quad u\in\mathbb{R}^{m_c}\times\{0,1\}^{m_\ell},\ m=m_c+m_\ell$$

$$y\in\mathbb{R}^{p_c}\times\{0,1\}^{p_\ell},\ p=p_c+p_\ell$$

Boolean Algebra: Basic Definitions and Notation

Definition (Boolean Variable, Boolean Expression, and Boolean Function):

A variable p is a boolean variable if $p\in\{0,1\}$, where $p=0$ means false and $p=1$ means true. A boolean expression is obtained by combining boolean variables through the logic operators:

$$\neg\text{(not)},\ \vee\text{(or)},\ \wedge\text{(and)},\ \leftarrow\text{(implied by)},\ \rightarrow\text{(implies)},\ \leftrightarrow\text{(iff)}$$

A boolean function $f:\{0,1\}^{n-1}\mapsto\{0,1\}$ is used to define a boolean variable p_n as a logic function of other variables $p_1,\ldots,p_{n-1}$:

$$p_n=f(p_1,p_2,\ldots,p_{n-1})$$

Note:

1. All these logical operations and the following translations into integer expressions are exactly the same as those in the field of linear temporal logic (LTL) research. In addition, the LTL community develops more logic expressions like “until”, “eventually” etc.
2. The above $p\in\{0,1\}$ are boolean variables representing Ture-False logic, not arithmetic integers.
3. Though we will use 0-1 binary integer variables to translate a logical structure, there are some other translation methods using the continuous variable, as the following shows.

Example: Representation of PWA System using Logic and Continuous Interval Translation

Consider the system with logic structure:

$$x_c(t+1)=2x_c(t)+u_c(t)-3u_\ell(t)$$

$$x_\ell(t+1)=x_\ell(t)\wedge u_\ell(t)$$

This example can be represented in the PWA form by associating $x_\ell=0$(FALSE) with $x_\ell\le 1/2$ and $x_\ell=1$(TRUE) with $x_\ell\ge 1/2+\epsilon$ for any $0\le\epsilon\le 1/2$, as the following shows:

$$\begin{bmatrix}x_c(t+1)\\x_\ell(t+1)\end{bmatrix}=\begin{cases}\begin{bmatrix}2x_c(t)+u_c(t)\\0\end{bmatrix} & \text{if}\quad x_\ell\le\frac{1}{2},u_\ell\le\frac{1}{2}\\ \begin{bmatrix}2x_c(t)+u_c(t)-3\\0\end{bmatrix} & \text{if}\quad x_\ell\le\frac{1}{2},u_\ell\ge\frac{1}{2}+\epsilon\\ \begin{bmatrix}2x_c(t)+u_c(t)\\0\end{bmatrix} & \text{if}\quad x_\ell\ge\frac{1}{2}+\epsilon,u_\ell\le\frac{1}{2}\\ \begin{bmatrix}2x_c(t)+u_c(t)-3\\1\end{bmatrix} & \text{if}\quad x_\ell\ge\frac{1}{2}+\epsilon,u_\ell\ge\frac{1}{2}+\epsilon\end{cases}$$

Note: the associating part is the translation process we will discuss in detail in the next section. Here we use the continuous interval to do translation, but we later would only focus on 0-1 binary variables.

II. Modeling of Hybrid Systems: Principles

- **Initial Remarks: Goals and Steps**

Our goal is to describe the hybrid system in a form that is compatible with optimization software, which means, we need the form of:

1. Continuous and boolean variables. 2. Linear equalities and inequalities

Basic Idea: associate to each boolean variable p_i a binary integer variable δ_i:

$$p_i = 1 \Leftrightarrow \text{TRUE} \Leftrightarrow \{\delta_i = 1\}, \quad \neg p_i = 1 \Leftrightarrow \text{FALSE} \Leftrightarrow \{\delta_i = 0\}$$

and embed them into a set of constraints as **Linear (Mixed-) Integer Inequalities.** The idea requires two things to be done:

1. Translation of Logic Rules into Linear Integer Inequalities
2. Translation of continuous and logical components into Linear Mixed-Integer Relations

Result: a compact model with linear equalities and inequalities involving real and binary variables, which is called the **Mixed Logical Dynamical (MLD)** System

- **Step 1: Linear Integer Inequalities: Translating Boolean Formulas**

Our goal for this step, described in mathematical language, is as follows;

Goal (Translating Boolean Formulas (Logic Rules)):

Given a Boolean formula $F(p_1, p_2, \ldots, p_n)$, define a polyhedral set $\mathcal{P}$ such that a set of binary values $\{\delta_1, \delta_2, \ldots, \delta_n\}$ satisfies the Boolean formula F in $\mathcal{P}$:

$$F(p_1, p_2, \ldots, p_n) = \text{TRUE} \quad \Leftrightarrow \quad A\delta \le B, \quad \delta \in \{0,1\}^n \text{ where } \{\delta_i = 1\} \Leftrightarrow p_i = \text{TRUE}$$

Analytic Approach

1. Transform $F(p_1, p_2, \ldots, p_n)$ into a **Conjunctive Normal Form (CNF):**

$$F(p_1, p_2, \ldots, p_n) = \bigwedge_j \left[\bigvee_i p_i \right]$$

2. Translation of a **CNF into algebraic inequalities** using the following table:

Logical relations	Boolean Expression	Linear Constraints
AND $\wedge$	$p_1 \wedge p_2$	$\delta_1 \ge 1, \delta_2 \ge 1$ or $\delta_1 + \delta_2 \ge 2$
OR $\vee$	$p_1 \vee p_2$	$\delta_1 + \delta_2 \ge 1$
NOT $\neg$	$\neg p_1$	$(1 - \delta_1 \ge 1)$ or $\delta_1 = 0$
XOR $\oplus$	$p_1 \oplus p_2$	$\delta_1 + \delta_2 = 1$
IMPLY $\rightarrow$	$p_1 \rightarrow p_2$	$\delta_1 - \delta_2 \le 0$
IFF $\leftrightarrow$	$p_1 \leftrightarrow p_2$	$\delta_1 - \delta_2 = 0$
ASSIGNMENT $p_3 = p_1 \wedge p_2$	$p_3 \leftrightarrow p_1 \wedge p_2$	$\delta_1 + (1 - \delta_3) \ge 1$ $\delta_2 + (1 - \delta_3) \ge 1$ $(1 - \delta_1) + (1 - \delta_2) + \delta_3 \ge 1$

Example:

Given

$$F(p_1,p_2,p_3,p_4)\triangleq[(p_1\wedge p_2)\rightarrow(p_3\wedge p_4)]$$

find the equivalent set of linear integer inequalities.

1. remove implication:

$$F(p_1,p_2,p_3,p_4)=\neg(p_1\wedge p_2)\vee(p_3\wedge p_4)$$

2. using DeMorgan's theorem, obtain CNF:

$$F(p_1,p_2,p_3,p_4)=(\neg p_1\vee\neg p_2\vee p_3)\wedge(\neg p_1\vee\neg p_2\vee p_4)$$

3. introduce $[\delta_i=1]\Leftrightarrow p_i=\text{TRUE}$ and write the inequalities:

$$F(p_1,p_2,p_3,p_4)=\text{TRUE}\Leftrightarrow\begin{cases}\delta_1+\delta_2-\delta_3\leq 1\\ \delta_1+\delta_2-\delta_4\leq 1\\ \delta_{1,2,3,4}\in\{0,1\}\end{cases}$$

- **Step 2: Linear Mixed-Integer Relations: Event Generator**

The above example is for general logic translations, it only contains the logic state and input (Boolean variables). However, in control community, a more common case is that we have logic conditions for different dynamics **that are related to a certain affine constraint on the continuous state and input.**

Note:

Take the systems we mainly discussed as an example: For the PWA system, we have the condition "when $x(t)\in\mathcal{X}_i$ (corresponding $p=\text{TRUE}$)" for locating the partition, where locating criterion $x(t)\in\mathcal{X}_i$ is the affine constraint $a^{\mathrm{T}}x\leq b$. For the SAS system, we have the condition "when $i(t)=s$ (corresponding $p=\text{TRUE}$)" for externally assigning the mode, where we also assume that the mode selection $i(t)=s$ depends on some threshold defined by $a^{\mathrm{T}}x\leq b$.

Following the pipeline above, we first want to connect the logic boolean variable p with the binary integer variable δ. Then we need to think of ways to link the continuous state for further translation. Hence we have the following definition of the event generator:

Definition (Event Generator):

An event generator is defined by the function $f_{\mathrm{EG}}:\mathcal{X}_{\mathrm{c}}\times\mathcal{U}_{\mathrm{c}}\times\mathbb{N}_0\rightarrow\mathcal{D}$:

$$\delta_e(t)=f_{\mathrm{EG}}(x_c(t),u_c(t),t)$$

which **generates a binary integer variable** of event conditions **according to the satisfaction of the linear or affine constraint threshold condition.**

In other words, the event generator generally works as the following:

if $x\in\mathcal{X}$ satisfies a certain linear or affine constraint (corresponding $p=\text{TRUE}$), then the event generator should output $\delta=1$, else (corresponding $p=\text{FALSE}$) the event generator should output $\delta=0$. i.e., we are considering the boolean expression consisting of a boolean variable p, binary

integer variable δ and continuous variable $x \in \mathbb{R}^n$ such that:

$$p = \text{TRUE} \Leftrightarrow a^{\mathrm{T}} x \leq b \Leftrightarrow \delta = 1$$

Where $a \in \mathbb{R}^n, b \in \mathbb{R}, x \in \mathcal{X} \subset \mathbb{R}^n$

Big-M Method for IFF Affine Satisfaction

To do the logic translation for the event generator, we here use a method called **"big-M method"**. First, we assume that the linear constraint is bounded on the set of all x (or conversely saying, we need to know the set $\mathcal{X}$ in which the constraint is bounded, otherwise the translation can not be done)

$$\mathcal{X} = \{x |\ a^{\mathrm{T}} x - b \in [m, M]\ \}$$

The translated linear integer inequalities are:

$$a^{\mathrm{T}} x - b \leq M(1 - \delta)$$

$$a^{\mathrm{T}} x - b > m\delta$$

We can do the sanity check as follows:

When $\delta = 1$, we have $\begin{cases} a^{\mathrm{T}} x - b \leq 0 \\ a^{\mathrm{T}} x - b > m \end{cases}$, from the first constraint we have $a^{\mathrm{T}} x \leq b$ (True), the second constraint $a^{\mathrm{T}} x - b > m$ does not matter since it always holds due to the bounded condition.

When $\delta = 0$, we have $\begin{cases} a^{\mathrm{T}} x - b \leq M \\ a^{\mathrm{T}} x - b > 0 \end{cases}$, from the second constraint we have $a^{\mathrm{T}} x > b$ (False), the first constraint $a^{\mathrm{T}} x - b \leq M$ does not matter since it always holds due to the bounded condition.

Note:

1. The event generator represents an "if and only if" logic. Note that we also have "if and only if" logic $p_1 \leftrightarrow p_2$ translated as $\delta_1 - \delta_2 = 0$ shown in the previous section (step 1), but it only connects the boolean variable and integer variable and forms **linear integer inequalities**. The big difference now is that we are tying p not only to δ, but also to an affine constraint related to x. The big-M translation connects boolean variables, integer variables and continuous states variables altogether to form **linear mixed-integer inequalities**.
2. It is worth mentioning that for event generator translation, the methods and resulting mixed-linear inequalities **are not unique**. With different translations, the complexity of the expression of hybrid dynamics and the difficulty of the optimization problem to be formulated would also be different. The translation is tricky, but the big-M method is the most widely used technique. Though sometimes a better translation does exist, we mainly focus on the big-M method.
3. Even for the big-M method, the choice of m, M can actually not be unique, but here we recommend using the theoretical bounds (sup and inf on $\mathcal{X}$) for better numerical accuracy.
4. In addition, for better numerical properties in real applications, we can replace $a^{\mathrm{T}} x - b > m\delta$ with $a^{\mathrm{T}} x - b \geq \epsilon + (m - \epsilon)\delta$ with $\epsilon > 0$ to be small enough (e.g. machine precision).

- **Step 3: Linear Integer & Linear Mixed-Integer Relations: Compact Hybrid Dynamics**

XOR for Mode Selection and Compact Dynamics

Now, we have the method to describe the trigger depending on linear or affine satisfaction, i.e., we can now use arithmetic numbers (binary integer variable and continuous variable) to form linear mixed-integer inequalities and describe "When $x \in \mathcal{X}_i$" or "When $i(t) = s$" for PWA or SAS system.

However, there is one last challenge (gap) that needs to overcome in modeling the whole hybrid system in a compact way: with a given condition, we still need to write out the mathematical descriptions for "selecting" the corresponding dynamics.

In fact, this can be done through rewriting the state update functions of the PWA (or SAS) system:

$$z_1(t) = \begin{cases} A_1 x_c(t) + B_1 u_c(t) + f_1 & \text{if } x_c(t) \in \mathcal{X}_1 \ \ (\text{or } i(t) = 1) \\ 0 & \text{otherwise} \end{cases}$$

$$\vdots$$

$$z_s(t) = \begin{cases} A_s x_c(t) + B_s u_c(t) + f_s & \text{if } x_c(t) \in \mathcal{X}_s \ \ (\text{or } i(t) = s) \\ 0 & \text{otherwise} \end{cases}$$

$$x_c(t+1) = \sum_{i=1}^{s} z_i(t)$$

In other words, we introduce new variables $z_i(t) = \delta_i(t)\left[A_i x_c(t) + B_i u_c(t) + f_i\right]$ and add them up to form the final compact dynamic, in which the selection would be imposed by the exclusive or (XOR) logic that only one δ_i can be 1, and the others would all be 0, i.e. $\sum_{i=1}^{s} \delta_i = 1$.

Big-M Method for IF-Then-Else Structure

The subproblem (each $z_i(t) \in \mathbb{R}^n$) is an **"If-Then-Else"** structure, and again can be translated by the big-M method. E.g., consider suitable dimension problems (say, one entry of $z_i(t)$, denoted as z, so $a_i^{\mathrm{T}}, b_i^{\mathrm{T}}$ are some rows of matrix A_i, B_i above, f here is slightly abused in notation), $z \in \mathbb{R}$, $x \in \mathbb{R}^n, u \in \mathbb{R}^m$ and the affine satisfaction condition for event generator is $c^{\mathrm{T}} x + d \le 0$, we have:

$$\begin{array}{ll} \text{IF } p \text{ THEN} & (m_2 - M_1)\delta + z \le a_2^{\mathrm{T}} x + b_2^{\mathrm{T}} u + f_2 \\ \quad z = a_1^{\mathrm{T}} x + b_1^{\mathrm{T}} u + f_1 & (m_1 - M_2)\delta - z \le -a_2^{\mathrm{T}} x - b_2^{\mathrm{T}} u - f_2 \\ \text{ELSE} \quad \Longleftrightarrow & (m_1 - M_2)(1-\delta) + z \le a_1^{\mathrm{T}} x + b_1^{\mathrm{T}} u + f_1 \\ \quad z = a_2^{\mathrm{T}} x + b_2^{\mathrm{T}} u + f_2 & (m_2 - M_1)(1-\delta) - z \le -a_1^{\mathrm{T}} x - b_1^{\mathrm{T}} u - f_1 \end{array}$$

where $x \in \mathcal{X}$, with:

$$\sup_{x \in \mathcal{X}} a_i^{\mathrm{T}} x + b_i^{\mathrm{T}} + f_i \le M_i, \qquad \inf_{x \in \mathcal{X}} a_i^{\mathrm{T}} x + b_i^{\mathrm{T}} + f_i \ge m_i, \ i = 1, 2 \qquad m_2 \ne M_1, \ m_1 \ne M_2$$

and δ is derived from:

$$c^{\mathrm{T}} x - d \le M(1-\delta)$$

$$c^{\mathrm{T}} x - d > m\delta$$

and note that the linear or affine satisfaction trigger is bounded:

$$\mathcal{X}=\{x|\ c^{\mathrm{T}}x-d\in[m,M]\ \}$$

Now, we know how to model the whole hybrid dynamics.

Note:

1. In this part, we use both linear integer inequalities (for XOR Translation) and linear mixed-integer inequalities (for "If-Then-Else" Translation) to get a compact form of the hybrid dynamics.
2. For cases that $s\geq 3$, we had better just introduce s binary variables, use the general formulation, and just let the solver deal with it, and note that a_2,b_2,f_2,M_2,m_2 would then be 0 in each subproblem ("If-Then-Else" structure) $z_i(t)$.
3. In fact, consider selecting 1 from s cases, since we have $\sum_{i=1}^{s}\delta_i=1$. There essentially only need $s-1$ binary variables, which means in the most compact form, the system can be written as $s-1$ "If-Then-Else" subproblem. For $s=2$, though it looks like we have 2 subproblem structures $z_1(t),z_2(t)$ to translate, it can be eliminated to a single basic "If-Then-Else". We can show the equivalence through the following, first, consider for $s=2$, the general formulation is.

$$z_1=\begin{cases}a_1^{\mathrm{T}}x+b_1^{\mathrm{T}}u+f_1 & \text{if } \delta_1=1\\ 0 & \text{otherwise}\end{cases}\qquad z_2=\begin{cases}a_2^{\mathrm{T}}x+b_2^{\mathrm{T}}u+f_2 & \text{if } \delta_2=1\\ 0 & \text{otherwise}\end{cases}$$

$$x^+=z_1+z_2=\delta_1(a_1^{\mathrm{T}}x+b_1^{\mathrm{T}}u+f)+\delta_2(a_2^{\mathrm{T}}x+b_2^{\mathrm{T}}u+f_2)$$

Translating z_1:

$$-M_1\delta_1+z_1\leq 0$$
$$m_1\delta_1-z_1\leq 0$$
$$m_1(1-\delta_1)+z_1\leq a_1^{\mathrm{T}}x+b_1^{\mathrm{T}}u+f_1$$
$$-M_1(1-\delta_1)-z_1\leq -a_1^{\mathrm{T}}x-b_1^{\mathrm{T}}u-f_1$$

Translating z_2:

$$-M_2\delta_2+z_2\leq 0$$
$$m_2\delta_2-z_2\leq 0$$
$$m_2(1-\delta_2)+z_2\leq a_2^{\mathrm{T}}x+b_2^{\mathrm{T}}u+f_2$$
$$-M_2(1-\delta_2)-z_2\leq -a_2^{\mathrm{T}}x-b_2^{\mathrm{T}}u-f_2$$

We have $\delta_1+\delta_2=1$, the above translations becomes:

Translating z_1:

$$-M_1\delta_1+z_1\leq 0 \quad ①$$
$$m_1\delta_1-z_1\leq 0 \quad ②$$
$$m_1(1-\delta_1)+z_1\leq a_1^{\mathrm{T}}x+b_1^{\mathrm{T}}u+f_1 \quad ③$$
$$-M_1(1-\delta_1)-z_1\leq -a_1^{\mathrm{T}}x-b_1^{\mathrm{T}}u-f_1 \quad ④$$

Translating z_2:

$$-M_2(1-\delta_1)+z_2\leq 0 \quad ⑤$$
$$m_2(1-\delta_1)-z_2\leq 0 \quad ⑥$$
$$m_2\delta_1+z_2\leq a_2^{\mathrm{T}}x+b_2^{\mathrm{T}}u+f_2 \quad ⑦$$
$$-M_2\delta_1-z_2\leq -a_2^{\mathrm{T}}x-b_2^{\mathrm{T}}u-f_2 \quad ⑧$$

Therefore, define $z=z_1+z_2=\delta_1[(a_1^{\mathrm{T}}-a_2^{\mathrm{T}})x+(b_1^{\mathrm{T}}-b_2^{\mathrm{T}})u+(f_1-f_2)]+a_2^{\mathrm{T}}x+b_2^{\mathrm{T}}u+f_2$

$$z=\begin{cases}a_1^{\mathrm{T}}x+b_1^{\mathrm{T}}u+f_1 & \text{if } \delta_1=1\\ a_2^{\mathrm{T}}x+b_2^{\mathrm{T}}u+f_2 & \text{otherwise}\end{cases}$$

$$①+⑦\quad (m_2-M_1)\delta_1+z\leq a_2^{\mathrm{T}}x+b_2^{\mathrm{T}}u+f_2$$
$$②+⑧\quad (m_1-M_2)\delta_1-z\leq -a_2^{\mathrm{T}}x-b_2^{\mathrm{T}}u-f_2$$
$$③+⑤\quad (m_1-M_2)(1-\delta_1)+z\leq a_1^{\mathrm{T}}x+b_1^{\mathrm{T}}u+f_1$$
$$④+⑥\quad (m_2-M_1)(1-\delta_1)-z\leq -a_1^{\mathrm{T}}x-b_1^{\mathrm{T}}u-f_1$$

It returns to the basic single "If-Then-Else" structure. Also, note that z happens to be x^+.

- **Example: Hybrid Modelling of PWA system with big-M**

Consider the following PWA system with constraints: $|x| \le 10, |u| \le 10$:

$$x_{t+1} = \begin{cases} 0.8x_t + u_t & \text{if } x_t \ge 0 \\ -0.8x_t + u_t & \text{if } x_t < 0 \end{cases}$$

Model it into a compact form of Mixed Logical Dynamical System (MLDS)

According to the pipeline above, this is a PWA system with an event generator, so we need to do two things: event generator translation and compact dynamics writing.

1. To do event generator translation, associate $\{\delta_t = 1\} \Leftrightarrow \{x_t \ge 0\}$ and use big-M method:

   First, decide the bounds of the affine constraints

   $$|x| \le 10 \Rightarrow -x_t \in [-10, 10] \Rightarrow m = -10 = -M$$

   Then write the mixed integer inequalities

   $$\begin{matrix} -x_t \le M(1-\delta_t) \\ -x_t > \epsilon + (m-\epsilon)\delta_t \end{matrix} \xrightarrow{M=-m} \begin{matrix} m(1-\delta_t) \le x_t \\ \epsilon + (-M-\epsilon)\delta_t < -x_t \end{matrix} \Rightarrow \begin{matrix} -m\delta_t \le x_t - m \\ -(M+\epsilon)\delta_t < -x_t - \epsilon \end{matrix}$$

   where ϵ is a small positive number (e.g. machine precision)

2. To write the compact dynamics, by defining $z_{it} = \delta_{it}[A_i x_t + B_i u_t + f_i]$ and $x_{t+1} = \sum_{i=1}^{s} z_{it}$ we can write the state update equation, also note that this is the simplest 2-case choice, so we can use a single δ_t to express everything:

   $$x_{t+1} = z_{1t} + z_{2t} = \delta_t[0.8x_t + u_t] + (1-\delta_t)[-0.8x_t + u_t] = 1.6\delta_t x_t - 0.8x_t + u_t$$

   To write the corresponding big-M translation for "If-Then-Else" structure, there are two options to do the translation:

   a. Introduce auxiliary variable

   $Z_t = x_{t+1} = 1.6\delta_t x_t - 0.8x_t + u_t$

   "If-Then-Else":

   IF $p(\delta = 1)$ THEN

   $Z_t = 0.8x_t + u_t$

   ELSE

   $Z_t = -0.8x_t + u_t$

   b. Introduce auxiliary variable

   $Z_t = \delta_t x_t$, then $x_{t+1} = 1.6Z_t - 0.8x_t + u_t$

   "If-Then-Else":

   IF $p(\delta = 1)$ THEN

   $Z_t = x_t$

   ELSE

   $Z_t = 0$

   Clearly, the second choice ($Z_t = \delta_t x_t$) would result in a simpler translation, so we take it and do the translation. Note that the two pieces are bounded: $x_t \in [-10, 10] \Rightarrow m_1 = -10 = -M_1$ $0 \in [0, 0] \Rightarrow m_2 = 0 = M_2$, so the translation is:

   $$\begin{matrix} (m_2 - M_1)\delta + Z_t \le a_2^{\mathrm{T}} x_t + b_2^{\mathrm{T}} u_t + f_2 \\ (m_1 - M_2)\delta - Z_t \le -a_2^{\mathrm{T}} x_t - b_2^{\mathrm{T}} u_t - f_2 \\ (m_1 - M_2)(1-\delta) + Z_t \le a_1^{\mathrm{T}} x_t + b_1^{\mathrm{T}} u_t + f_1 \\ (m_2 - M_1)(1-\delta) - Z_t \le -a_1^{\mathrm{T}} x_t - b_1^{\mathrm{T}} u_t - f \end{matrix} \Rightarrow \begin{matrix} Z_t \le M_1 \delta_t \\ Z_t \ge m_1 \delta_t \\ Z_t \le x_t - m_1(1-\delta_t) \\ Z_t \ge x_t - M_1(1-\delta_t) \end{matrix}$$

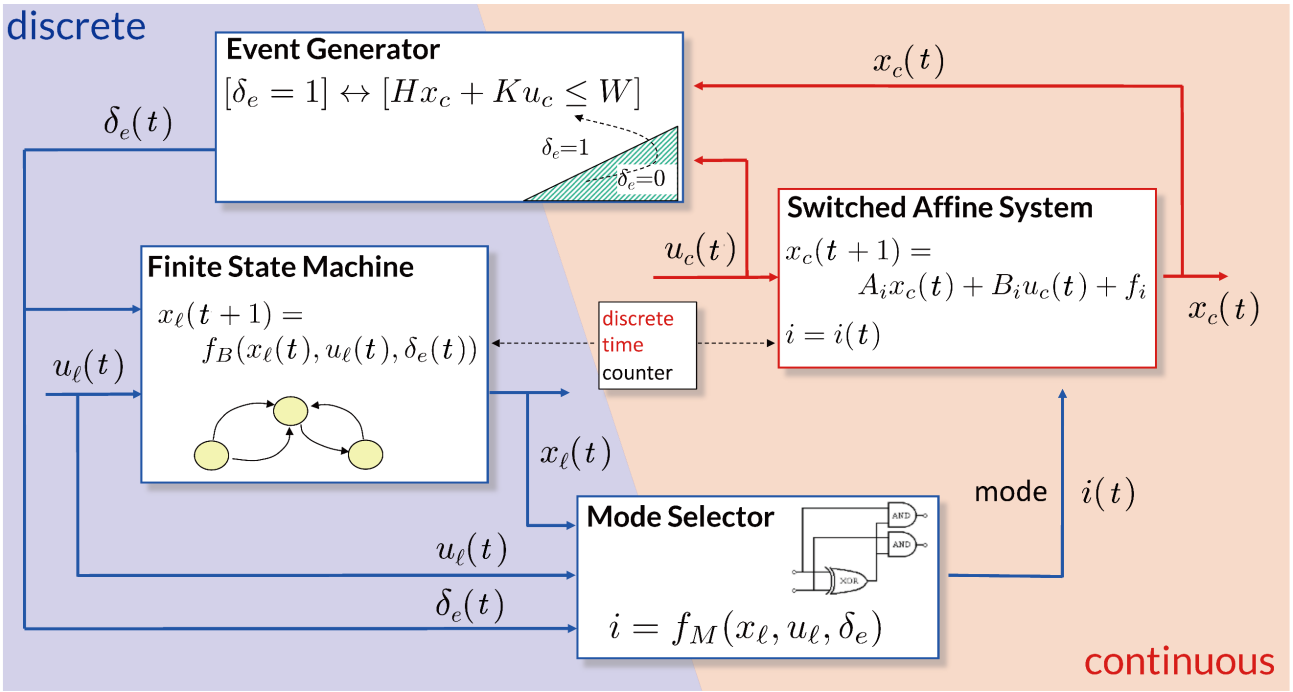
Note:

1. This is the nonlinear system $x^+ = 0.8|x| + u$
2. Why do we introduce auxiliary variables? It is because in the mixed-integer formulation, the binary integer δ and continuous state x are both system variables, so their product δx is bi-linear (bi-linear is also a kind of nonlinearity). Therefore, we need to introduce auxiliary variables to make the system linear, or all our efforts would be meaningless.
3. In the example above, we set the auxiliary variable $Z_t = \delta_t x_t$ because the coefficient of the input b_1^{T} and b_2^{T} happens to cancel out, and only the bi-linear coupling term is $\delta_t x_t$. This choice, compared with using $Z_t = x_{t+1} = 1.6\delta_t x_t - 0.8x_t + u_t$, would make the translation a lot easier (we do not need to consider bounds for affine expressions caused by input). In the most general case, we minimally need to define auxiliary variables $Z_t = \delta_t[(a_1^{\mathrm{T}} - a_2^{\mathrm{T}})x_t + (b_1^{\mathrm{T}} - b_2^{\mathrm{T}})u_t]$ to cancel all the bi-linear relationships between the variables.

III. Modeling of Hybrid Systems: DHA, MLD and Summary

- **Discrete Hybrid Automata (DHA) System**

In fact, apart from PWA (SAS) system, a more general description of the hybrid system is the Discrete Hybrid Automata (DHA) which can be concluded as the following figure:



where $x_\ell \in \{0,1\}^{n_\ell}$ is the binary logical boolean state, $u_\ell \in \{0,1\}^{n_\ell}$ is the binary logical boolean input, $\delta_e \in \{0,1\}^{n_e}$ is the binary integer variable, $x_c \in \mathbb{R}^{n_c}$ is the real-valued continuous state, and $u_c \in \mathbb{R}^{m_c}$ is the real-valued continuous input, $i \in \{1, \ldots, s\}$ is the mode selector.

Actually, what we discussed before can all be fit into this DHA framework. To make a simple illustration and correspondence, we can see the DHA is composed of:

1. Affine dynamics depend on the current mode $i(t)$: $x_c(t+1) = A_{i(t)}x_c(t) + B_{i(t)}u_c(t) + f_{i(t)}$ (corresponds to the **SAS** mentioned previously)
2. Event variable δ_e generated by affine threshold conditions (corresponds to the **Event Generator** mentioned previously)
3. The binary state of the finite state machine evolves according to a boolean state update function (corresponds to the **Boolean Formulas** mentioned previously)
4. The active mode $i(k)$ is selected by a Boolean function, also called the mode selector of the current binary (corresponds to the **XOR logic** for selecting the dynamics piece mentioned above)

It is not hard to observe that the PWA system is a special case of a DHA whose threshold events and mode selector function are defined by the PWA partition.

- **DHA system, MLD system and HYSDEL**

DHA to MLD for Hybrid MPC

And the translations we discussed in the previous sections show that a DHA can be converted into the following MLD model:

$$x(t+1) = Ax(t) + B_1u(t) + B_2\delta(t) + B_3Z(t)$$
$$y(t) = Cx(t) + D_1u(t) + D_2\delta(t) + D_3Z(t)$$
$$E_2\delta(t) + E_3Z(t) \le E_4x(t) + E_1u(t) + E_5$$

Where $x \in \mathbb{R}^{n_c} \times \{0,1\}^{n_\ell}, u \in \mathbb{R}^{m_c} \times \{0,1\}^{m_l}, y \in \mathbb{R}^{p_c} \times \{0,1\}^{p_\ell}, \delta \in \{0,1\}^{r_\ell}$ and $Z \in \mathbb{R}^{r_c}$ and the physical constraints on continuous variables:

$$\mathcal{C} = \left\{ \begin{bmatrix} x_c \\ u_c \end{bmatrix} \in \mathbb{R}^{n_c+m_c} \mid Fx_c + Gu_c \le H \right\}$$

It should be noted that the inequality $E_2\delta(t) + E_3Z(t) \le E_4x(t) + E_1u(t) + E_5$ is the integer or mixed-integer linear inequalities we introduced in the process of doing translation, and the whole dynamics looks "linear" in $x(t), u(t), \delta(t)$ and the auxiliary variable $Z(t)$. It is not truly linear because there are binary boolean and integer components in $x(t), u(t), \delta(t)$ and $Z(t)$, but this is good enough for mature optimization solvers to solve.

To sum up, by converting logic relations into mixed-integer linear inequalities, a DHA can be rewritten as the MLD system. MLD allows solving MPC, verification, state estimation, and fault detection problems via mixed-integer programming.

HYbrid System DEscription Language (HYSDEL)

In practice, an efficient way to generate the MLD is to use the HYbrid System DEscription Language (HYSDEL), which is based on DHA and enables the description of discrete-time hybrid systems in a compact way. It can deal with the following:

Automata & propositional logic; Continuous dynamics; A/D & D/A conversion; Constraint definition

And it automatically generates MLD models for MATLAB

- **Well-Posedness, Other Hybrid Models and Model Equivalence**

MLD Prediction and Well-Posedness

For MLD description, we want to make sure that it makes sense for most control purposes, meaning it should at least uniquely predict the evolution of the system. This motivates the following definition:

Definition (Well-Posedness):

For MLD system, if given a state-input pair $x=[x(t)\ \ u(t)]^{\mathrm{T}} \Rightarrow x(t+1)$ and $y(t)$ are uniquely determined, then the MLD system is called well-posed.

Well-posedness is sufficient for the computation of the state and output prediction.

Note: There are algorithms that can check the well-posedness of the MLD system, but we do not go into the details here.

Other Hybrid Models, Model Equivalence and Complete Well-Posedness

In the lecture we introduced PWA (and the more general DHA), which is most suitable for modeling the hybrid system intuitively, and we also introduced MLD, which is most suitable for solving MPC and optimal control, and we learned how to do the translation from PWA (DHA) to MLD. As mentioned before, given a hybrid system, there actually are more methods we can model hybrid systems. Here we list two other existing hybrid models:

1. Linear complementarity systems (LCS)

$$\begin{aligned} x(t+1) &= Ax(t)+B_1u(t)+B_2w(k) \\ y(t) &= Cx(t)+D_1u(t)+D_2w(t) \\ v(t) &= E_1x(t)+E_2u(t)+E_3w(t)+E_4 \\ 0 &\le v(t)\ \perp\ w(t)\ge 0 \end{aligned}$$

   This model is widely used in mechanical systems with frictional contact and is now popular in the robotics community.

2. Min-max-plus-scaling (MMPS) systems

$$\begin{aligned} x(t+1) &= M_x(x(t),u(t),w(t)) \\ y(t) &= M_y(x(t),u(t),w(t)) \\ 0 &\ge M_c(x(t),u(t),w(t)) \end{aligned}$$

   Where $M_{(\cdot)}$ are MMPS functions defined by the grammar:

$$M = x_i \mid \alpha \mid \max(M_1,M_2) \mid \min(M_1,M_2) \mid M_1+M_2 \mid \beta M_1$$

   Example: $x(k+1)=2\max(x(k),0)+\min(0.5u(k),1)$

A question then arises: do all these hybrid models can be equivalently written as MLD? This motivates us to propose a stronger version of well-posedness

Definition (Complete Well-Posedness):

If the MLD system is well-posed, and in addition, it can uniquely determine the binary variable $\delta(t)$

and auxiliary variable $Z(t)$, $\forall x = [x(t) \quad u(t)]^{\mathrm{T}}$, then it is called complete well-posed.

Complete well-posedness allows transformation into equivalent hybrid models. There are some research results on the equivalence of the hybrid models:

1. MLD and PWA systems are equivalent (Bemporad, Ferrari-Trecate, Morari, 2000)
2. Efficient conversion algorithms from MLD to PWA form exist. (Geyer, Torrisi, Morari, 2003)
3. Further equivalences exist with other classes of hybrid dynamical systems, such as LCS (Heemels, De Schutter, Bemporad, 2001)

Note: If one derives MLD system from the methods told in Section II above, then the MLD system is guaranteed to be complete well-posedness, and that is how HYSDEL works.

IV. Optimal Control of Hybrid Systems

- **Optimal Control for Hybrid Systems: General Formulation**

The CFTOC problem formulation for the hybrid system looks exactly the same as what we got before:

$$J^*(x(t)) = \min_{U_0} \; p(x_N) + \sum_{k=0}^{N-1} q(x_k, u_k, \delta_k, Z_k)$$

$$\text{s.t.} \begin{cases} x_{t+1} = Ax_t + B_1 u_t + B_2 \delta_t + B_3 Z_t \\ E_2 \delta_t + E_3 Z_t \le E_4 x_t + E_1 u_t + E_t \\ x_N \in \mathcal{X}_f \\ x_0 = x(t) \end{cases}$$

where $U_0 = \{u_0, \ldots, u_{N-1}\}$ is the control sequence.

The only big difference is we now have $x \in \mathbb{R}^{n_c} \times \{0,1\}^{n_b}, u \in \mathbb{R}^{m_c} \times \{0,1\}^{m_b}, y \in \mathbb{R}^{p_c} \times \{0,1\}^{p_b}$ $\delta \in \{0,1\}^{r_b}$ and $Z \in \mathbb{R}^{r_c}$ all composed of either continuous or binary variables. Resulting in the final optimization problem being **mixed-integer optimization.**

Note: In the following notation, we use subscript b to represent the binary part of the variables. Since we link the boolean p and binary integer δ, this subscript b is equivalent to subscript ℓ before.

- **Mixed Integer Programming**

Mixed Integer Linear Programming (MILP)

Consider the following MILP:

$$\inf_{[z_c, z_b]} \; c_c^{\mathrm{T}} z_c + c_b^{\mathrm{T}} z_b + d$$

$$\text{subj. to} \quad G_c z_c + G_b z_b \le W$$

$$z_c \in \mathbb{R}^{s_c}, \; z_b \in \{0,1\}^{s_b}$$

MILP are nonconvex, in general. For a fixed $\bar{z}_b$ the MILP becomes a linear program:

$$\inf_{z_c} \; c_c^{\mathrm{T}} z_c + (c_b^{\mathrm{T}} \bar{z}_b + d)$$

$$\text{subj. to} \quad G_c z_c \le W - G_b \bar{z}_b$$

$$z_c \in \mathbb{R}^{s_c}$$

This also means we have the **Brute force approach** to the solution: enumerating the 2^{s_b} integer values of the variable z_b and solving the corresponding LPs. By comparing the 2^{s_b} optimal costs, one can find the optimizer and the optimal cost of the MILP.

Mixed Integer Quadratic Programming (MIQP)

Consider the following MIQP:

$$\begin{aligned} \inf_{[z_c, z_b]} \ & \frac{1}{2} z^{\mathrm{T}} H z + q^{\mathrm{T}} z + r \\ \text{subj. to} \ & G_c z_c + G_b z_b \leq W \\ & z_c \in \mathbb{R}^{s_c}, \ z_b \in \{0,1\}^{s_b} \\ & z = [z_c, z_b], \ s = s_c + s_d \end{aligned}$$

where $H \succeq 0, z_c \in \mathbb{R}^{s_c}, z_b \in \{0,1\}^{s_b}$

MIQP are nonconvex, in general. For a fixed $\overline{z}_b$, the MILP becomes a quadratic program:

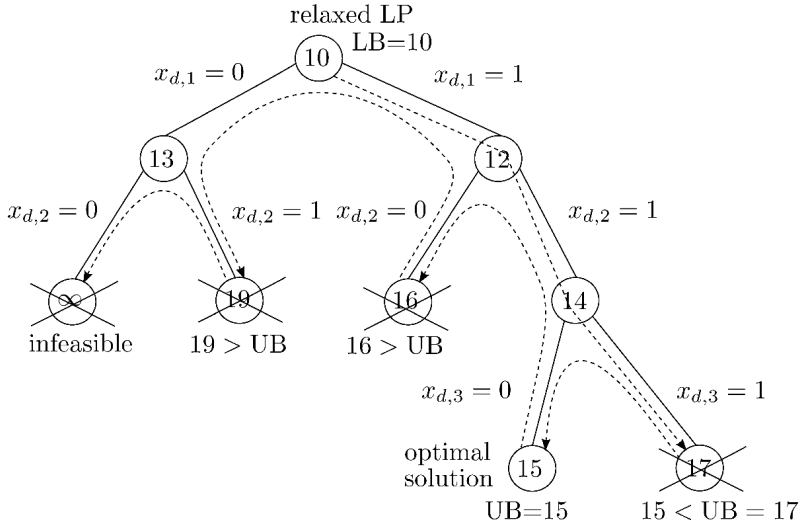
$$\begin{aligned} \inf_{z_c} \ & \frac{1}{2} {z_c}^{\mathrm{T}} H_c z_c + {q_c}^{\mathrm{T}} z_c + k \\ \text{subj. to} \ & G_c z_c \leq W - G_b \overline{z}_b \\ & z_c \in \mathbb{R}^{s_c} \end{aligned}$$

Similarly, we have the **Brute force approach** to the solution: enumerating the 2^{s_b} integer values of the variable z_b and solving the corresponding QPs. By comparing the 2^{s_b} optimal costs, one can find the optimizer and the optimal cost of the MIQP.

Branch and Bound (B&B)

A common solution method for Mixed Integer Programming, is a technique that is based on the relaxations of binaries $\{0,1\} \to [0,1]$ and the construction of a decision tree. It is known as branch and bound (B&B) method. The following example shows the pipeline of this approach.

Consider a system with three integer variables: $[x_{d,1} \ x_{d,2} \ x_{d,3}] \in \{0,1\}^3$



Step 1: Firstly, relax the integer constraints, take $[x_{d,1}\ x_{d,2}\ x_{d,3}]\in[0,1]^3$ and solve the normal LP. Since the constraint is relaxed, the cost we get is the lower bound of the actual cost, here $\mathrm{LB}=10$.

Step 2: Then, pick an arbitrary combination of the integers and, grow the tree to the end, calculate the cost. Here we take $[x_{d,1}\ x_{d,2}\ x_{d,3}]=[1\ 1\ 1]$ and get the cost for this combination, note that though this is imposing the integer constraint, so the calculating cost would be the upper bound for the actual optimal cost, here $\mathrm{UB}=17$.

Step 3: Then we go back to check the case when $[x_{d,1}\ x_{d,2}\ x_{d,3}]=[1\ 1\ 0]$, by calculating the cost, we can see that it is lower than $\mathrm{UB}=17$ before, so we set the new $\mathrm{UB}=15$ and throw out the previous combination $[1\ 1\ 1]$

Step 4: Then we go one node back, check the case when $[x_{d,1}\ x_{d,2}\ x_{d,3}]=[1\ 0\ *]$, and calculate the cost, here $*$ means that we only relax the third one $x_{d,3}\in[0,1]$, so it would be a lower bound for combinations $[1\ 0\ 0]$ and $[1\ 0\ 1]$. The solution is 16, even higher than the $\mathrm{UB}=15$ before, so it can be judged that any further growth and calculation on the $[1\ 0\ *]$ branch would be meaningless since with further tree growth, the cost would only be higher. We can directly neglect this whole branch

Step 5: Again, go one step to check the case when $[x_{d,1}\ x_{d,2}\ x_{d,3}]=[0\ *\ *]$, the result is 13, smaller than $\mathrm{UB}=15$, so we keep growing the tree, the cost lower bound for $[x_{d,1}\ x_{d,2}\ x_{d,3}]=[0\ 1\ *]$ is $19>\mathrm{UB}=15$, so $[0\ 1\ *]$ branch can be neglected, the cost lower bound for $[x_{d,1}\ x_{d,2}\ x_{d,3}]=[0\ 0\ *]$ is ∞, meaning infeasible, so we also do not need to further grow the tree.

To sum up, for mixed integer programming problem:

1. The optimal cost of a solution to a modified problem where some binaries are relaxed is a **lower bound** on optimal cost.
2. Any feasible solution to the original problem is **upper bound** on optimal cost.
3. The branch and bound method uses these information as a heuristic to systematically do efficient solution search. We can see from the example above, it effectively cuts off the combinations that are useless for finding the optimal solution. For large-scale problems, this can lead to a huge improvement in computation efficiency.

Note: Mixed integer programming has developed a lot over the past decades by the operation research community. Nowadays, all the mature solvers use B&B, some smarter heuristics are also proposed. We need to think twice if we want to dig into the research of MIP. We had better treat ourselves as a tool user and figure out how to smartly formulate the problem rather than invent a new MIP solver.

IV. Model Predictive Control and Explicit MPC of Hybrid Systems

- **Hybrid MPC Formulation**

As for linear MPC, at each sample time:

1. Measure / estimate current state $x(t)$
2. Find the optimal input sequence for the entire planning window (horizon) N:
$$U_t^* = \{u_t^*, u_{t+1}^*, \ldots, u_{t+N-1}^*\}$$
3. Implement only the first control action u_t^*

The key difference for Hybrid MPC: Requires online solution of a MILP or MIQP

- **Explicit Hybrid MPC**

Also, it is possible to do the explicit MPC for hybrid systems, as the following theorem shows.

Theorem: (Explicit MPC for Hybrid Systems - Quadratic Case)

The solution to the CFTOC problem based on an MLD model and with the quadratic cost is **time-varying PWA feedback law** of the form:
$$u_t^*(x_t) - F_t^i x_t + G_t^i \text{ if } x_t \in \mathcal{R}_k^i$$
where $\{\mathcal{R}_t^i\}_{t=1}^{R_t}$ are regions partitioning the set of feasible states $\mathcal{X}_t^*$ and the closure $\bar{\mathcal{R}}_t^i$ of the sets $\bar{\mathcal{R}}_t^i$ of the sets $\mathcal{R}_t^i$ has the following form:
$$\bar{\mathcal{R}}_t^i = \{x:\ x_t{}^{\mathrm{T}} L(j)_t^i x_t + M(j)_t^i x_t \le N(j)_t^i,\ j = 1, \ldots, n_t^i,\ t = 0, \ldots, N-1\}$$
And need to note that the partitions have quadratic and linear boundaries: **they are not polyhedra!**

The pipeline for quadratic explicit hybrid MPC is as follows:

1. Denote by $\{v_i\}_{i=1}^{s^N}$ the set of all possible switching sequences over the horizon N
2. Fix a certain v_i and constrain the state to switch according to the sequence v_i
3. The problem becomes a **CFTOC for a linear time-varying system**. The solution is:
$$u^i(x(0)) = \tilde{F}^{i,j} x(0) + \tilde{g}^{i,j},\ \forall x(0) \in \mathcal{T}^{i,j},\ j = 1, \ldots, N^{r^i} \text{ where } \mathcal{D}^i = \bigcup_{i=1}^{N^{r^i}} \mathcal{T}^{i,j}$$
Note that the set $\mathcal{X}_0 = \bigcup_{i=1}^{s^N} \mathcal{D}^i$ in general, is not convex. And the set $\mathcal{D}_i$ can, in general, overlap. i.e., some initial state is feasible for more than one switching sequence.

Theorem: (Explicit MPC for Hybrid Systems – 1 and ∞ Norm Case)

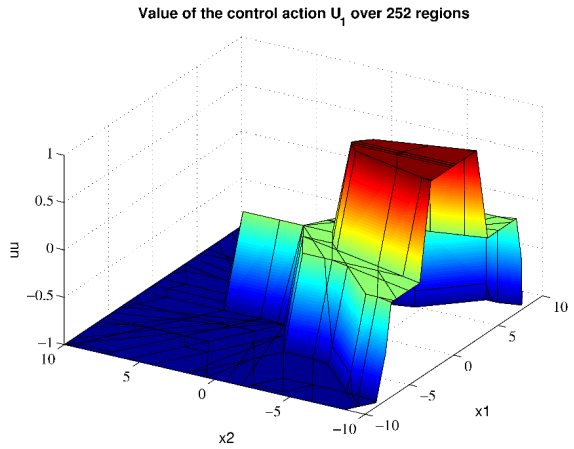
The solution to the CFTOC problem based on an MLD model and with cost based on norms $\{1, \infty\}$ is the **time-varying PWA feedback law** of the form:
$$u_t^*(x_t) - F_t^i x_t + G_t^i \text{ if } x_t \in \mathcal{R}_k^i$$
where $\{\mathcal{R}_t^i\}_{t=1}^{R_t}$ are **polyhedral regions** partitioning the set of feasible states $\mathcal{X}_t^*$.

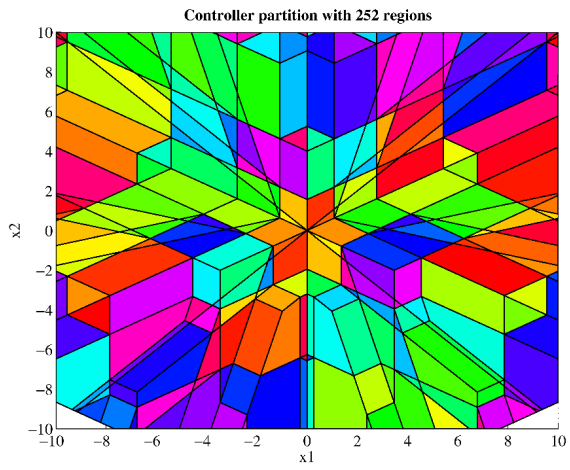
Explicit MPC for Hybrid Systems – Example

Given the system:
$$\begin{cases} x_{t+1} = 0.8 \begin{bmatrix} \cos\alpha_t & -\sin\alpha_t \\ \sin\alpha_t & \cos\alpha_t \end{bmatrix} x_t + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u_t \\ \alpha_t = \begin{cases} \pi/3 & \text{if } [1 \ \ 0] x_t \ge 0 \\ -\pi/3 & \text{if } [1 \ \ 0] x_t < 0 \end{cases} \end{cases}$$
with constraints $x_t \in [-10,\ 10] \times [-10,\ 10],\ u_t \in [-1,\ 1]$

with MPC design: $N=12,\ P_N=Q=I,\ R=1,\ \infty$ norm cost

The explicit solution is shown below:





Note:

1. No matter 2-norm or $1,\infty$-norm, the explicit solution in general is intractable and too complex, but in theorem at least $1,\infty$-norm case would be better since the partitions are polyhedra and can be dealt with some regular tools, as is shown above.
2. All in all, in practice, the best solution would not be explicitly solving this, but doing online computation (or, one can turn to reinforcement learning for a possible solution).

V. Summary

- **Hybrid Systems and Problem Formulation**

Hybrid Systems: a mixture of continuous and discrete dynamics

Many important systems fall into this class

Many tricks involved in modeling - automatic systems available to convert to consistent form

Big Difference: The optimization problem becomes a mixed-integer linear / quadratic program

NP-hard (exponential time to solve)

Advanced commercial solvers available

- **Hybrid MPC Theory and Complexity**

MPC Theory:

Invariance, stability, etc applies, but…

Computing invariant sets is usually extremely difficult

Computing the optimal solution is extremely difficult (sub-optimal ok)

MPC complexity in Practice:

Complexity strongly depends on the problem structure and the initial setup and:

Mixed-integer programming is HARD

Efficient general-purpose solvers for MILP/MIQP: CPLEX, XPRESS-MP) based on Branch-And-

Bound, Branch-And-Cut methods + lots of heuristics

On-line optimization is good for applications allowing large sampling intervals (typically minutes), requires expensive hardware and (even more) expensive software

For very small and deterministic problems requiring fast sampling rate: can turn to explicit solution