INTELLIGENT CONTROL SYSTEMS

Time-Dependent Rules and Rule Bases

Katalin Hangos

Department of Electrical Engineering and Information Systems

Oct 2022

Lecture overview

1. Time dependent rules
   - Rules
   - Rules as dynamic system models
   - Example: Rules for the coffee machine
   - Reasoning with rules for control and diagnosis

2. Datalog Rule Sets
   - Analysis of datalog rule sets

3. Verification of rule bases
   - Contradiction freeness
   - Completeness

Recall - Logical expressions

Atomic formulas

- logical constants (**true**, **false**)
- logical variables
- predicates: relations with Boolean values, e.g. $x > y\ ,\ x, y \in \mathbb{R}$

Basic logical operations:

- **and** ($\wedge$), **or** ($\vee$),
- **neg** ($\neg$),
- **imp** ($\rightarrow$)

Recall - Logical operations

Operation table of the **implication** ($\rightarrow$) operation:

- used for describing **rules**

$a \rightarrow b$ $a \downarrow \quad b \rightarrow$	**false**	**true**
false	**true**	**true**
true	**false**	**true**

Recall - Canonical forms of logical expressions

- the *disjunctive normal form* or *DNF* is disjunction of conjunctions of atomic formulas or their negations, e.g.
  $(\neg a \wedge b) \vee (c \wedge \neg d)$
- the *conjunctive normal form* or *CNF* is conjunction of disjunctions of atomic formulas or their negations, e.g.
  $(\neg a \vee b) \wedge (c \vee \neg d)$
- the *implicative normal form* or *INF* is an implication with the conjunction of atomic formulas on the left and disjunctions of atoms on the right, e.g.
  $(\neg a \wedge b) \to (c \vee \neg d)$

Extension: the **unknown** value

unknown can be interpreted as "either **true** or **false**", i.e.

$$\textbf{unknown} = \textbf{true} \vee \textbf{false}$$

Extended Boolean value set

$$\overline{\mathbb{B}} = \{\textbf{true}, \textbf{false}; \textbf{unknown}\}$$

Operation table of the extended **or** operation

$a \vee b$ $a \downarrow \quad b \rightarrow$	**false**	**true**	**unknown**
false	**false**	**true**	**unknown**
true	**true**	**true**	**true**
unknown	**unknown**	**true**	**unknown**

Rules - syntax

Rule formats

if condition **then** conclusion;

$$\text{condition} \quad \rightarrow \quad \text{conclusion;}$$

where both "condition" and "conclusion" are logical expressions.

Logical expressions
syntactical elements

- logical constants: **true**, **false**
- **predicates**: atomic logical expression with the value **true** or **false**
- logical operations: $\wedge$ (**and**), $\vee$ (**or**), $\neg$ (**not**), $\rightarrow$ (implication)

Time dependent predicates

Arithmetic predicates based on signal values
syntactical elements

- constants: numerical (e.g. 0.0) or qualitative (e.g. **high** or **open**)
- arithmetic **relation symbols**: $=, \neq, \leq, <, \geq, >$
- signal identifier: denoting the time dependent value of a signal, e.g. the level of a tank $\ell(t)$, or the status of a valve $v_1(t)$

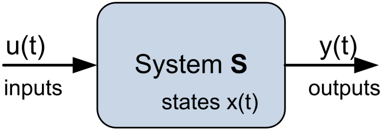
Examples

$$\begin{array}{cc} p_1 = (\ell \ = \ \mathbf{high}) & p_2 = (\ell \ \geq \ 1.0) \\ p_3 = (v_1 \ = \ \mathbf{open}) & p_4 = (v_1 \ \neq \ \mathbf{closed}) \end{array}$$

Time dependent rules contain time dependent predicates

$$(p_1 \wedge p_2) \quad \rightarrow \quad p_3$$

Time dependent predicates and rules



Signals: the value of ime dependent predicates (Boolean valued ($\in \overline{\mathbb{B}}$) discrete time signals)

- input predicate: depends on an input signal $u(t)$ of the system
- state predicate: depends on a state signal $x(t)$
- output predicate: depends on an output signal $y(t)$

Model equations are the rules

State space is spanned by the state predicates: $\overline{\mathbb{B}}^n$

Constructing time dependent rules

Time dependent rules sets as dynamic system models can be constructed

- from common sense heuristically,
- from confluences
- from discrete time qualitative DAEs

Recall – Rule generation from confluences

The **rows of the truth table of a confluence can be interpreted as a rule** if one reads them from right to left.
For example

$$\delta h = [\eta_I] \ominus_S [\eta_O]$$

with the combination $\eta_I = 0$, $\eta_I = +$ gives $\delta h = -$

$\Longrightarrow$

$$\textbf{if} \;\; (\eta_I = \textbf{closed}) \textbf{ and } (\eta_O = \textbf{open}) \;\; \textbf{then} \;\; (h = \textbf{decreasing})$$

Important

Rule sets can be generated from the truth table of a confluence.
The generated rules are datalog rules.
The generated rule set is contradiction-free by construction ,
but it may not be complete.

Recall – Rule generation from QDEs

The **rows of the solution table of a QDE can be interpreted as a rule** if one reads them from right to left.
For example

$$v^m = v + \chi \cdot E$$

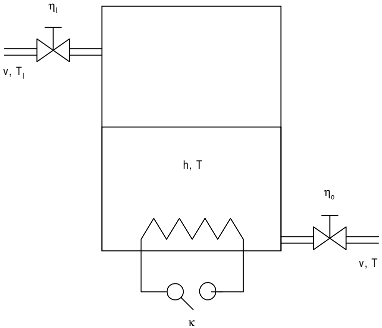
with the combination $[v] = N$, $\chi = -1$ gives $[v]^m = L$
$\Longrightarrow$

$$\textbf{if} \;\; (\chi = \textbf{neg fault}) \textbf{ and } ([v] = \textbf{normal}) \;\; \textbf{then} \;\; ([v]^m = \textbf{low})$$

Important

Rule sets can be generated from the truth table of a static QDE in a datalog form.
The generated rules are contradiction-free and complete.

The operation of the coffee machine



Engineering model equations

$$\begin{aligned} \frac{dh}{dt} &= \frac{v}{A}\eta_I - \frac{v}{A}\eta_O && \text{(mass balance)} \\ \frac{dT}{dt} &= \frac{v}{Ah}(T_I - T)\eta_I + \frac{H}{c_p \rho h}\kappa && \text{(energy balance)} \end{aligned} \qquad (1)$$

Rules describing the operation of the coffee machine

Rules originate from the **mass balance** and common sense

Predicates:

- input: $p_{Isz} = (\eta_I = 1)$, $p_{Osz} = (\eta_O = 1)$
- state: $p_{hinc} = (\Delta h > 0)$, $p_{hstd} = (\Delta h = 0)$, $p_{hsmall} = (h < 0.1\ cm)$, $p_{hnormal} = (13\ cm < h < 15\ cm)$

Rules:

$$IF\ (p_{Isz} \wedge \neg p_{Osz})\ THEN\ p_{hinc}$$
$$IF\ (\neg p_{Isz} \wedge p_{Osz})\ THEN\ \neg p_{hinc}$$
$$IF\ (\neg p_{Isz} \wedge \neg p_{Osz})\ THEN\ \neg p_{hinc}$$

– –

$$IF\ (p_{hsmall} \wedge p_{hinc})\ THEN\ p_{hnormal}$$
$$IF\ (p_{hnormal} \wedge \neg p_{hinc})\ THEN\ p_{hsmall}$$

Reasoning with rules

Depends on the goal (method) of reasoning

Prediction: we use forward chaining:

1. logical expression *condition* is checked
2. when **true** the rule "fires" (!! conflict resolution may be needed)
3. executing a rule: its *consequence* is *made* **true** by changing the value of the corresponding predicates

Diagnosis: we use backward chaining

Important

Reasoning changes the state (i.e. the values of the state predicates) in each step.

Datalog rules - definition

Datalog rule sets have the following properties

D1. *There is no function symbol in the arguments of the rules' predicates.*

D2. *There is no negation* $\neg$ *applied to the predicates and the rules are in the following form:*

$$(p_{i_1} \wedge \cdots \wedge p_{i_n}) \quad \rightarrow \quad q_i;$$

D3. *The rules should be "safe rules", that is their value should be evaluated in finite number of steps.*

Transformation to **datalog** form

General rule sets are transformed to datalog form by

M1. *Remove function symbols* for requirement D1.
functions are computed by infinite series

M2. *Remove negations and disjunctions ($\neg$ and $\vee$ operations)* for requirement D2.
implicative normal form + negation of the relation in predicates

$$\neg(a > b) = (a \leq b) \quad , \quad \neg(a = b) = (a \neq b) \quad etc.$$

$$\neg(a \vee b) = \neg a \wedge \neg b$$

$$\begin{array}{rcccc}
(s_0) & : & (p_{i_1} \wedge \cdots \wedge p_{i_n}) & \rightarrow & (q_{i_1} \vee \cdots \vee q_{i_m}); \\
(s_0') & : & (p_{i_1} \wedge \cdots \wedge p_{i_n}) & \rightarrow & (\neg q_{i_1} \wedge \cdots \wedge \neg q_{i_m}); \\
 & & & becomes & \\
(s_{i_1}) & : & (p_{i_1} \wedge \cdots \wedge p_{i_n}) & \rightarrow & \neg q_{i_1}; \\
 & & & \cdots & \\
(s_{i_n}) & : & (p_{i_1} \wedge \cdots \wedge p_{i_n}) & \rightarrow & \neg q_{i_m};
\end{array}$$

M3. *Use finite digit realization of real numbers* for requirement D3.

Dependence graph of datalog rules

Dependence graph $D = (V_D, E_D)$: **directed** graph

1. The vertex set of the graph is the set of the predicates in the rule set

$$V_D = P$$

2. Two vertices p_i and p_j are connected by a directed edge $(p_i, p_j) \in E_D$ if there is a rule in the rule set such that p_i is present in the *condition* part and p_j is the *consequence*.
3. Label the edges (p_i, p_j) by the rule identifier they originate from.

Analysis of datalog rule sets

The dependence graph shows how the predicate values depend on each other.

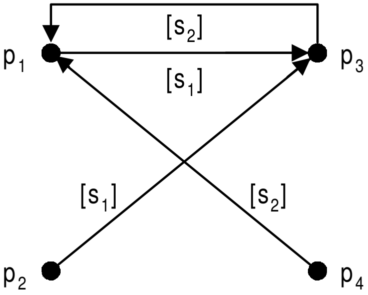
- The *set of entrances of the dependence graph* are the **root predicates**: they should be given if we want to compute the value of the others.
- *Directed circles* show that the result of the computation may depend on the computation order.

If there is no directed circle in the dependence graph then we obtain the same reasoning (evaluation) result regardless of the computation order.

Dependence graph – example

Set of predicates: $P = \{p_1, p_2, p_3, p_4\}$
The implication form of the rule set

$$(s1): \quad (p_1 \wedge p_2) \quad \rightarrow \quad p_3; \qquad (s2): \quad (p_3 \wedge p_4) \quad \rightarrow \quad p_1;$$



Testing knowledge bases

We can test a knowledge base in two principally different ways.

- Either we *validate* it by comparing its content with additional knowledge of a different type,
- or we *verify* it by checking the knowledge elements against each other to find conflicting or missing items.

Properties to be checked during verification

- contradiction freeness
- completeness

Definition of contradiction freeness

Reliable knowledge bases have a unique primary or inferred knowledge item, if they have any, irrespectively of the way of reasoning.

Definition:
A rule-based knowledge base with *datalog rules* is contradiction free if the value of any of the non-root predicates is uniquely determined by the rule-base using the rules for forward chain reasoning.

Testing contradiction freeness

The algorithmic problem

Testing Contradiction Freeness

Given:

- *A rule-based knowledge base* with its datalog rule structure.

Question:
Is the rule-base contradiction free?

Testing contradiction freeness – 2

Solution:

1. *Determine the set of root predicates* (polynomial)
   by analyzing the dependence graph or by collecting all predicates which do not appear on the consequence part of any rule.

2. *Construct the set of all possible values for the root predicates* (to be stored in the set S_{rp}, non-polynomial)
   by considering the possible values **true**, **false** for every root predicate. The number of the elements in this set is $2^{n_{rp}}$.

3. *For every element in S_{rp} perform forward chaining and compute the value of the non-root predicates in every possible way* (NP-complete)

4. Finally, *check that the computed values for each of the non-root predicates are the same*. If yes then the answer to our original question is *yes*, otherwise *no*.

A simple example – contradiction freeness

Set of predicates $P = \{p_1, p_2, p_3, p_4, p_5\}$ so that $p_5 = \neg p_4$ holds. This is described by a "virtual" rule pair:

$$(r_{01}): \quad p_5 \quad \rightarrow \quad \neg p_4; \qquad (r_{02}): \quad p_4 \quad \rightarrow \quad \neg p_5;$$

The implication form of the rule set

$$\begin{array}{lclcl} (r_1) & : & (p_1 \wedge p_2) & \rightarrow & p_4; \\ (r_2) & : & (p_3 \wedge p_1) & \rightarrow & p_5; \\ (r_3) & : & (p_1 \wedge p_2) & \rightarrow & p_3; \end{array}$$

The set of root predicates is $P_{root} = \{p_1, p_2\}$
With the following values for the root predicates: $p_1 =$ **true** , $p_2 =$ **true** we get for p_4 the following values

- **true** from (r_1)
- **false** from (r_3), (r_2), (r_{01})

Definition of completeness

Rich enough knowledge bases have an answer (even this answer is not unique) to every possible query or question.

Definition:
A rule-based knowledge base with *datalog rules* is complete if any non-root predicate gets a value when performing forward chain reasoning with the rules.

Testing completeness

The algorithmic problem

Testing Completeness

Given:

- *A rule-based knowledge base* with its datalog rule structure.

Question:
Is the rule-base complete?

Testing completeness – 2

Solution:

1. *Determine the set of root predicates* (polynomial)
   by analyzing the dependence graph or by collecting all predicates which do not appear on the consequence part of any rule.

2. *Construct the set of all possible values for the root predicates* (to be stored in the set S_{rp}, non-polynomial)
   by considering the possible values **true**, **false** for every root predicate. The number of the elements in this set is $2^{n_{rp}}$.

3. *For every element in S_{rp} perform forward chaining and and generate a reasoning tree* (NP-complete) until either all non-root predicates appear at least once or all the rules have been applied in every possible order.

4. Finally, *check that each of the non-root predicates gets at least one value in every possible case*. If yes then the answer to our original question is *yes*, otherwise *no*.

A simple example – completeness

Set of predicates $P = \{p_1, p_2, p_3, p_4, p_5\}$ so that $p_5 = \neg p_4$ holds. This is described by a "virtual" rule pair:

$$(r_{01}) : \quad p_5 \quad \rightarrow \quad \neg p_4; \qquad (r_{02}) : \quad p_4 \quad \rightarrow \quad \neg p_5;$$

The implication form of the rule set

$$\begin{array}{lclcl}
(r_1) & : & (p_1 \wedge p_2) & \rightarrow & p_4; \\
(r_2) & : & (p_3 \wedge p_1) & \rightarrow & p_5; \\
(r_3) & : & (p_1 \wedge p_2) & \rightarrow & p_3;
\end{array}$$

The set of root predicates is $P_{root} = \{p_1, p_2\}$
With the values for the root predicates $p_1 =$ **true** , $p_2 =$ **false**, we have no applicable rule from the rule set therefore the non-root predicates p_3, p_4 and p_5 are undetermined in this case.