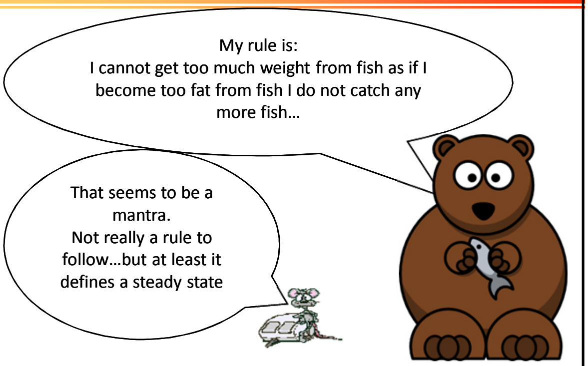
3 Rule-based systems

- In what follows, we give a brief introduction to the mathematical definitions and applied algorithms of rule-based systems
- Rule-based systems comprise
  - a **database that contains the current knowledge** of the system. This may comprise current values or states of variables/parameters/sets
  - a **finite set of rules** that enables the system to derive additional knowledge out of the given one by applying some rule
  - an **interference engine or algorithm** that controls the interaction between the knowledge stored in the data base and the applicable rules
- The definition of the knowledge, the rules, and the applied interference engine is application-dependent and therefore requires a suitable formalization

Rule-based systems



Definition of rule-base systems

3.1 **Definition**

A **rule-based system** P consists of a tuple (D, R) with the **data base** D and a finite **set of rules** R. The elements are also denoted as (known) facts. The elements of D are tuples of parameters and values (denoted as terms). The set of parameters are denoted as $\mathcal{P}(P)$ and the set of values are $\mathcal{V}(P)$. As parameters and values are connected in each tuple by some operator (functioning as a connector) $=, <, \leq$, or $\neq$ to a **term** $t \in D$, of the form $t \in \mathcal{P}(P) \times \{=, \neq, <, \leq\} \times \mathcal{V}(P)$.

A **rule** $r \in R$ possesses the form **IF** C **THEN** t with a **condition** C that is recursively defined through

1. Each term $r \in D$ is a condition
2. For $r, s \in D$ the notations $(r \wedge s)$ and $(r \vee s)$ are conditions

and a term $t \in D$ that defines the **conclusion** (of the rule).

Conjunction and disjunction

- The symbol "∧" represents a **conjunction** of conditions. Hence, the fulfillment of the resulting (possibly partial) condition requires that both partial conditions are fulfilled by the current database entries
- The symbol "∨" represents a **disjunction** of conditions. Hence, the fulfillment of the resulting (possibly partial) condition requires that (at least) one partial condition is fulfilled by the current database entries

Observation

- Conclusions do not comprise conjunctions of terms
  - However, this can be replaced by additional rules
  - Specifically, instead of defining **IF** C **THEN** $t_1 \wedge t_2 \wedge \cdots \wedge t_k$, we insert the k rules **IF** C **THEN** t_1, **IF** C **THEN** t_2,…, **IF** C **THEN** t_k into R
- Note that the well-known **NOT operation** is not valid in Definition 3.1
- In what follows, we define the formal satisfaction of a condition

Satisfaction of conditions in rules

3.2 **Definition**

Given a rule-based system $P = (D, R)$ defined according to Definition 3.1.
Then, a condition C in a rule **IF** C **THEN** t in rule set R is satisfied by the current data base D if one of the following cases applies

1. If C is a single term s and $s \in D$ holds
2. If $C = C_1 \wedge C_2$ and C_1 as well as C_2 are satisfied by the current data base D
3. If $C = C_1 \vee C_2$ and C_1 or C_2 is satisfied by the current data base D
4. If $C = \neg C_1$ and C_1 is not satisfied by the current data base D

No further case exists.

A rule $r =$ **IF** C **THEN** t in set R with a condition C that is satisfied according to Definition 3.2 is denoted as **applicable to data base** D.

If the fourth case is not covered, we say that $P = (D, R)$ is **without negation**.

Inference of terms

3.3 **Definition**

Given two rule-based systems $P = (D, R)$ and $P' = (D', R')$ as defined in Definition 3.1.1. It holds that $(D, R) \vdash_{rs} (D', R')$ if and only if

1. There exists a rule $r \in R$ with $r =$ **IF** C **THEN** t possessing a satisfied condition C and
2. The data base is extended accordingly, i.e., $D' = D \cup \{t\}$

We say that rule $r \in R$ is applicable and term t can be inferred (derived) in $P = (D, R)$ with D

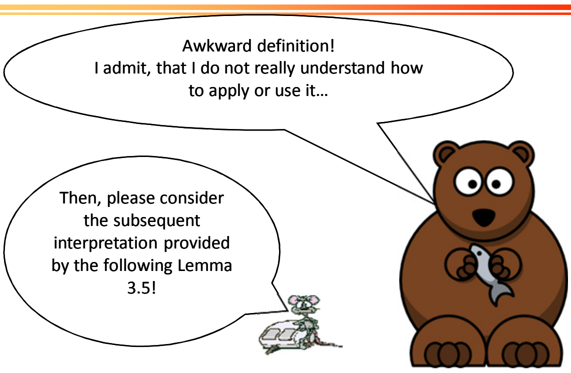
Shortcut: $(D, R) \vdash_{rs} \{t\}$

Commutative rule-based system

3.4 **Definition**

A rule-based system $P = (D, R)$ is denoted as **commutative** if and only if for each data base E that can be inferred (derived) in P with D it holds that each rule that can be applied in P for E can be also applied in F that can be derived from E, i.e., $(D, R) \vdash_{rs} (E, R) \vdash_{rs} (F, R)$ and $\forall r \in R$: r is applicable in $E \Longrightarrow$ r is applicable in F.

Consequences



Consequence

3.5 **Lemma**

A rule based system $P = (D, R)$ is commutative if and only if the following attribute (a) is fulfilled for a data base E that can be inferred in P with D:

(a) Let $R_E \subseteq R$ be a subset of rules given in P that are applicable with data base E. Then, the data set that is inferable by applying these rules is invariant against the sequence in that the rules are applied.

Proof of Lemma 3.5

- Given a commutative rule-base system $P = (D, R)$ and a data base E that can be inferred in P by using D, i.e., we have $(D, R) \vdash_{rs} (E, R)$
- Moreover, we assume that $R_E = \{r_1, \dots, r_n\} \subseteq R$ is the set of applicable rules of set R with data base E
- Consequently, we define the set of terms $T_E = \{t_1, \dots, t_m\}$ that we obtain by applying rules of set R_E
- Due to the assumed commutativity of $P = (D, R)$ and since data base E can be inferred in P by using D, we know that all rules $r \in R_E$ can be also applied in P by using F (instead of E) with $(D, R) \vdash_{rs} (E, R) \vdash_{rs} (F, R)$
- This means that we can apply the respective rules $r \in R_E$ in an arbitrary sequence and the set of inferable terms amounts to $E \cup T_E = E \cup \{t_1, \dots, t_m\}$
- The latter results from the fact that the application of each rule inserts a term of T_E

Proof of Lemma 3.5

- Conversely, we assume that P is not commutative, but attribute (a) is fulfilled for a data base E that can be inferred in P by using D, i.e., $(D, R) \vdash_{rs} (E, R)$
- If two rules in R have identical conclusions we combine them into one rule by a disjunction in the condition. Hence all conclusions are disjoint
- Furthermore, we assume that $R_E = \{r_1, \dots, r_n\} \subseteq R$ is the set of applicable rules of set R with data base E
- Consequently, we define the set of terms $T_E = \{t_1, \dots, t_m\}$ that we obtain by applying rules of set R_E
- As P is not commutative, we assume that E was chosen such that there exists a rule $r_j \in R_E = \{r_1, \dots, r_n\}$ with F as the set of terms that is obtained by applying all applicable rules of set $R_E - \{r_j\}$ while r_j is not applicable in set F, but applicable in set E. By applying all rules of set $R_E - \{r_j\}$, we obtain the set $T_{E,j}$
- Since no other rule implies t_j, we obtain a different data base G if we start with (E, R) and apply r_j first as $t_j \in G$ but $t_j \notin T_{E,j}$. This contradicts attribute (a)

Remark

- The derived definition of commutativity is analogously characterized by Nilsson (1982). He gives the following three attributes
  - Each rule that is applicable for a given database D stays applicable for each database that is derivable from D
  - Each condition that is fulfilled by D is also fulfilled by each database that is derivable from D
  - Each database that can be derived from D is invariant against the sequence of the applied rules

Observation

3.6 **Theorem**

Rule-based systems without negation are commutative.

Proof of Theorem 3.6

- We consider a rule based system $P = (D, R)$ without negation
- Let E be a data base with $(D, R) \vdash_{rs} (E, R)$
- $R_E = \{r_1, \dots, r_n\} \subseteq R$ is the set of applicable rules of set R with data base E, while it holds that $r_i =$ **IF** C_i **THEN** t_i, $\forall i \in \{1, \dots, n\}$
- Hence, C_i is true (i.e., fulfilled) in E
- Since there are only connectors of the form $\wedge$ or $\vee$, the satisfaction of a condition only depends on the fact whether a specific term t is in the data base E
- As each rule application only adds additional terms to the data base, such a satisfaction does not change
- Hence, P is commutative

Conclusion

- Due to Theorem 3.6, for rule-based systems without negation, we know that the sequence of applied rules has no impact on the resulting set of terms in the derived data base
- Therefore, an applied inference algorithm do not need a specific selection rule for choosing the next applicable rule to be executed

Derivable knowledge

3.7 **Definition**

Given a commutative rule-based system $P = (D, R)$. Then, the set $D^*(P)$ is denoted as the maximum set of derivable terms in P if and only if it holds that

1. $(D, R) \vdash_{rs} (D^*(P), R)$ and
2. $\forall E$ with $(D, R) \vdash_{rs} (E, R)$ it holds that $E \subseteq D^*(P)$

Types of reasoning strategies

In order to check whether a specific data can be derived from a given rule-based system, two reasoning strategies are proposed:

Forward chaining

- Starts with the available data currently stored in the data base
- It iteratively executes the rules that are applicable in order to derive additional knowledge
- It terminates when a predefined goal (sought term) is reached
- If no predefined goal is given the algorithm stops when no further knowledge can be obtained from applying rules

Types of reasoning strategies

Backward chaining

- This strategy works in opposite direction to the forward chaining
- Namely, it starts with the goal term that is sought
- This term is inserted into the set of goal terms G
- As long as the set of goal terms G is not empty do
  - Take some term t out of G
  - Consider the condition C of each rule of the form IF C THEN t
  - Depending on the condition C insert new terms into G (this may include recursive function calls or the iterative constitution of different sets G and will be specified later on)

3.8 Algorithm – Forward Chaining in Pseudo Code

Input: Database D_0, set of rules R (no goal)

begin

- $D^* := D_0$
- repeat
  - $D := D^*$ /* keeping the former state */
  - $R^* := \{ IF\ C\ THEN\ t \in R \mid C\ is\ true\ for\ D \}$
  - $D^* := D^* \cup \{ t \mid IF\ C\ THEN\ t \in R^* \}$
- until $D = D^*$ /* if a goal is given, we can check it here */
- Output: D^*

end

(Very simple) example

RULE 1: IF AUDIO=croaks ∧ NUTRITION=insects - **THEN** ANIMAL=frog

RULE 2: If AUDIO=öök ∧ NUTRITION=insects - **THEN** ANIMAL=toad

RULE 3: If ANIMAL=frog - **THEN** COLOR=green

RULE 4: If ANIMAL=toad - **THEN** COLOR=brown

Applying forward chaining

- Starting set $D_0 = \{AUDIO = croaks, NUTRTION = insects\}$
- Hence, $D^* = \{AUDIO = croaks, NUTRTION = insects\}$
- Thus, we obtain
  $R^* =$ {**IF** AUDIO=croaks ∧ NUTRITION=insects – **THEN** ANIMAL=frog}
- And therefore
  $D^* = \{AUDIO = croaks, NUTRTION = insects, ANIMAL = frog\}$
- $R^* = \left\{\begin{matrix}\textbf{IF}\text{ AUDIO=croaks} \land \text{NUTRITION=insects} - \textbf{THEN}\text{ ANIMAL=frog,} \\ \textbf{IF}\text{ ANIMAL=frog} - \textbf{THEN}\text{ COLOR=}green\end{matrix}\right\}$
- Finally, we obtain $D^* = \left\{\begin{matrix}AUDIO = croaks, NUTRTION = insects, \\ ANIMAL = frog,\ COLOR = green\end{matrix}\right\}$

Forward chaining

3.9 **Theorem**

By being applied to a commutative rule-based system $P = (D, R)$, the algorithm forward chaining is correct and works in quadratic time of the size of the given rule-based system $P = (D, R)$

Proof of Theorem 3.9

Termination

- Clearly, the forward chaining algorithm (Algorithm 3.8) always terminates
- This results from the fact that R is assumed to be finite and therefore the derivable knowledge (terms located after a THEN statement) is finite
- Hence, the number of extensions of D^* is limited by the number of rules in R
- Specifically, it holds that
  $D \subseteq D^*(P) \subseteq D \cup \{t \mid IF\ C\ THEN\ t \in R\}$
- Since at least one term is added during each iteration of the algorithm, we have at most $|R|$ iterations

Proof of Theorem 3.9

Correctness

- We first show that if the Algorithm 3.8 is called without a goal term it terminates with the output $D^* = D^*(P)$
- We first show that $D^*(P) \subseteq D^*$
- For this purpose, we assume that $\exists s \in D^*(P) - D^*$. Due to $D^*(P) \subseteq D \cup \{t \mid IF\ C\ THEN\ t \in R\}$ and the finiteness of D, s can be derived within a finite number of rule applications
- Furthermore, s is defined such that its shortest derivation in $(D,R) \vdash_{rs} (D^*(P), R\ )$ requires a minimum number of rule applications. This minimum number is denoted as i. With other words, no other term in $D^*(P) - D^*$ can be derived with a smaller number of applied rules
- In what follows, the existence of s is disproven by induction over the number of iterations i
- With other words, we prove that after i iterations D^* contains all terms of set $D^*(P)$ that are derivable by the application of at most i rules

Proof of Theorem 3.9

- Start of induction with $i = 0$. In this case, we have $D^*(P) = D$ as no application of a rule is allowed
- Hence, as the Algorithm 3.8 sets $D^* = D$, we have $D^*(P) = D = D^*$ and s does not exist with $i = 0$
- Therefore, it remains to consider the case $i > 0$
- Then, by induction and the definition of term s, after conducting $i - 1$ iterations of Algorithm 3.8, the set D^* contains all terms of set $D^*(P)$ derivable by at most $i - 1$ rule applications
- Consequently, as $D^*(P) \subseteq D \cup \{t \mid IF\ C\ THEN\ t \in R\}$ holds and since s is not derivable within $i - 1$ rule applications, we conclude due to the commutativity of $P = (D, R)$ there must be a rule $IF\ C\ THEN\ s$ in set R such that the terms of $D^*(P)$ that are derivable within at most $i - 1$ rule applications fulfill C

Proof of Theorem 3.9

- However, as, by induction, this set (the terms of $D^*(P)$ that are derivable within at most $i - 1$ rule applications) is subset of D^*
- Therefore, $IF\ C\ THEN\ s$ is applicable during the i-th iteration of the Algorithm 3.8 and s is also inserted into D^*
- Hence, s does not exist
- As $D^*(P) \subseteq D \cup \{t \mid IF\ C\ THEN\ t \in R\}$ holds and D is finite, each term $s \in D^*(P) - D^*$ is derivable in a finite number of rule applications
- Thus, $D^*(P) - D^* = \emptyset$ holds as $s \in D^*(P) - D^*$ exists, otherwise and that was excluded before
- This proves $D^*(P) \subseteq D^*$

Proof of Theorem 3.9

- We show that $D^* \subseteq D^*(P)$
  - This results directly from the fact that the Algorithm 3.8 only adds terms by applying rules of set R
  - Consequently, it holds that $(D, R) \vdash_{rs} (D^*, R)$
  - This implies $D^* \subseteq D^*(P)$
- Therefore, we obtain $D^* = D^*(P)$
- This completes the proof of the correctness

Proof of Theorem 3.9

Worst case running time

- Due to $D^* = D^*(P) \subseteq D \cup \{t \mid IF\ C\ THEN\ t \in R\}$, there are at most $|R|$ iterations
- In each iteration at most each term in D^* and each rule has to be enumerated. By using a sophisticated data structure this is possible in time $\mathcal{O}(|R|)$
- Thus, all in all, we obtain an asymptotic running time of $\mathcal{O}(|R|^2)$

Observation

- The average running time of the Algorithm 3.8 can be improved by erasing each applied rule from set R

3.10 Algorithm – Forward Chaining with goal term

Input: Database D_0, set of rules R, goal term is t^*

begin

- $D^* \coloneqq D_0$
- repeat
  - $D \coloneqq D^*$ /* keeping the former state */
  - $R^* \coloneqq \{IF\ C\ THEN\ t \in R \mid C\ is\ true\ for\ D\}$
  - $D^* \coloneqq D^* \cup \{t \mid IF\ C\ THEN\ t \in R^*\}$
- until $D = D^*$ or $t^* \in D^*$
- Output: If $t^* \in D^*$ then write (“t^* is derivable”) else write (“t^* is NOT derivable”)

end

Excluding disjunctions

- By analyzing a rule-based systems, we can state that disjunctions can be excluded without restricting the knowledge and rules in a rule-based systems
- This results from the fact that we can replace a rule
  - IF $t_1 \vee t_2$ THEN t_3

  by the two following rules
  - IF t_1 THEN t_3
  - IF t_2 THEN t_3

  that are equivalent, i.e., the set of derivable terms is unchanged

Examples

- IF $(A = 1 \wedge B = 1) \vee C = 0$ THEN $X = 1$

Is equivalent to the two rules

- IF $(A = 1 \wedge B = 1)$ THEN $X = 1$
- IF $C = 0$ THEN $X = 1$

- IF $(A = 1 \vee B = 1) \wedge C = 0$ THEN $X = 1$

Is equivalent to

- IF $\big((A = 1 \wedge C = 0) \vee (B = 1 \wedge C = 0)\big)$ THEN $X = 1$

And equivalent to the two rules

- IF $(A = 1 \wedge C = 0)$ THEN $X = 1$
- IF $(B = 1 \wedge C = 0)$ THEN $X = 1$

Comment

- As each formula in propositional logic can be transformed in a equivalent formula in so-called Disjunctive Normal Form (DNF), i.e., into the form $F = \bigvee_{i=1}^{n} \left(\bigwedge_{j=1}^{m_i} L_{i,j} \right)$, with $L_{i,j} \in \{A_1, A_2, \dots\} \cup \{\neg A_1, \neg A_2, \dots\}$, we can always exclude all disjunctions in a set of rules
- Thus, for the backward chaining algorithm, we solely consider rule-based systems without disjunctions

3.11 Backward Chaining with goal term and DFS

```
function depth(t: list of terms): boolean;
begin
if t = NIL then return(true)  /* Nothing to check anymore, goal is attainable */
else /* There are still terms (i.e., conditions) to check */
        set t* to the first term in list t /* first term to be checked */
        define cl as a list of conditions of rules (i.e., list of list of terms) with conclusion "THEN t*"
        if t* ∈ D
        then cl: = append(NIL − list, cl) /* NIL-list is an empty (i.e., true) condition */
        stop = false
        while (cl ≠ NIL) and (not stop) do
                set cl* to the first condition in cl
                newgoal ≔ append(cl*, rest(t)) /* This has to be checked next (DFS) */
                if depth(newgoal) then stop:=true else cl ≔ rest(cl) end if
        end while /* cl is a list of conditions. One of them has to be fulfilled to fulfill t* */
        if stop then return(true) else return(false) end if
        end if
end if
```


Call of the procedure – main program

Input: Database D_0, set of rules R (no disjunction), goal term is t^*

```
begin
        if depth([t*]) then write("t* is derivable")
                else write("t* is NOT derivable")
        end if
end
```

Corresponding graph

3.12 **Definition**

Given a rule-based system $P = (D, R)$. For the set of rules, we define the following corresponding graph $\mathcal{G}(R) = \big(\mathcal{V}(R), E(R)\big)$ as follows:

1. For each term t occurring in a condition or conclusion of a rule $r \in R$ there exists a corresponding node $v_t \in \mathcal{V}(R)$
2. For each rule $r \in R$ there exists a corresponding node $v_r \in \mathcal{V}(R)$
3. For each rule $r = IF\ c_1 \wedge \cdots \wedge c_n\ THEN\ t \in R$ there exist n corresponding edges $\big(v_{c_i}, v_t\big) \in E(R), \forall i \in \{1, \dots, n\}$ and an additional edge $(v_r, v_t) \in E(R)$
4. Aside from the results by applying the preceding steps 1,2, and 3, there are no further nodes and arcs in $E(R)$

Acyclic rule-based systems

3.13 **Definition**

A given rule-based system $P = (D, R)$ is denoted as acyclic if and only if the corresponding graph $\mathcal{G}(R) = \big(\mathcal{V}(R), E(R)\big)$ is acyclic.

3.14 **Comment**

Given a directed graph $\mathcal{G} = (V, E)$. The test of whether graph $\mathcal{G}$ is acyclic can be done in linear time of the size of the set of arcs.

Correctness of the algorithm

3.15 **Theorem**

Algorithm 3.11 is correct for acyclic rule-based systems

Proof of Theorem 3.15

- We assume that a given term t can be derived by a rule based system $P = (D, R)$
- Then there exists a shortest existing derivation $(D, R) \vdash_{rs} (D^*, R)$ with $t \in D^*$ and we prove by induction of the number of applied rules l in the above shortest derivation that $depth([t]) =$ true, i.e., Algorithm returns the correct result
- Start of induction $l = 0$
  - In this case no rule is necessary for the derivation of $t \in D^*$
  - Hence, it holds that $t \in D$
  - In this case $first(t) \in D$ holds and the first entry of cl is the empty list
  - Therefore, $newgoal$ becomes to NIL and $depth(NIL)$ is called
  - Then, $depth(newgoal)$ is true and $stop$ is set to $true$
  - Consequently, the Algorithm 3.11 returns the correct result "t^* is derivable"

Proof of Theorem 3.15

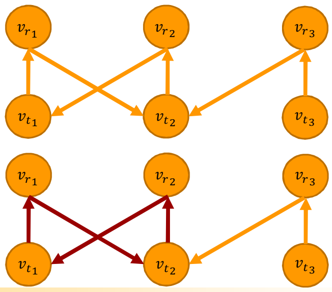
- We consider the case $l > 0$
  - Hence, as the considered derivation is a shortest one, there exists a rule IF $c_1 \wedge \cdots \wedge c_n$ THEN t in set R that was used by the considered derivation $(D, R) \vdash_{rs} (D^*, R)$ with $t \in D^*$
  - Hence, by induction we have $\forall i \in \{1, \ldots, n\}: depth([c_i]) = true$
  - As $r \in R$ holds, cl is extended by appending the list $[c_1, \ldots, c_n]$
  - As $P = (D, R)$ is assumed to be acyclic, in the considered case, the Algorithm will either terminate before reaching this part of the list cl (other proving is possible) or after checking $depth([c_1, \ldots, c_n])$. The latter results from the assumption of the induction

Proof of Theorem 3.15

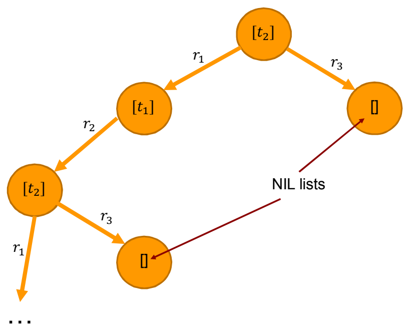
- We assume that a given term t cannot be derived by a rule-based system $P = (D, R)$
- Then, there is no $(D, R) \vdash_{rs} (D^*, R)$ with $t \in D^*$
- This, in turn, means that there is no possibility to trace back the term t to the initial entries of set D
- Therefore, $depth(NIL)$ cannot be reached throughout the computation
- Since $P = (D, R)$ is assumed to be acyclic, each rule is chosen once and the Algorithm 3.11 will terminate after finite time as $depth$ is not called in a recursion more than once for a list starting with the same term. Hence, the number of calls is bounded and the algorithm never reaches an empty list
- Thus Algorithm 3.11 returns the correct result "t^* is NOT derivable"

3.16 Example with cycle in R

- We consider the following rule-based system $P = (D, R)$ with
- $D = \{t_3\}, R = \{R_1: IF\ t_1\ THEN\ t_2, R_2: IF\ t_2\ THEN\ t_1, R_3: IF\ t_3\ THEN\ t_2\}$
- There is a cycle as the corresponding graph reveals



Applying the Algorithm 3.11 with $depth([t_2])$



Complexity

3.17 **Lemma**

The worst case running time of the Algorithm 3.11 is not polynomial even for acyclic rule-based systems $P = (D, R)$

Proof of Lemma 3.17

- We consider the following rule-based system
  - $P_n = (D, R_n)$ with
  - $D = \{t_0\}$ and
  - $R_n = \left\{ \begin{array}{l} IF\ c_{i,1} \wedge c_{i,2}\ THEN\ t_i, \\ IF\ t_{i-1}\ THEN\ c_{i,1} \\ IF\ t_{i-1}\ THEN\ c_{i,2} \end{array} \middle| \ 1 \le i \le n \right\}$
- The rule-based system $P_n = (D, R_n)$ comprises $3n$ rules and terms
- We count the number of calls $\mathcal{A}(n)$ of the function $depth$ with the goal term t_n

Proof of Lemma 3.17 – $n = 0, n = 1$

- $\mathcal{A}(0) = 2$ as $t_0 \in D$ and after calling $depth([t_0])$, we have a second and final call $depth([])$ that is successful
- $\mathcal{A}(1) = 5$ as shown below
  - $depth([t_1])$
  - $depth([c_{1,1}, c_{1,2}])$
  - $depth([t_0, c_{1,2}])$
  - $depth([c_{1,2}])$
  - $depth([t_0])$

Proof of Lemma 3.17 – $n > 1$

- We have always the situation that
  - $depth([t_n])$
  - $depth([c_{n,1}, c_{n,2}])$
  - $depth([t_{n-1}, c_{n,2}])$
  - …
  - $depth([c_{n,2}])$
  - …
  - $depth([t_0])$

Proof of Lemma 3.17 – Conclusion

- For $n > 1$, it holds that $\mathcal{A}(n) = 3 + 2 \cdot \mathcal{A}(n-1)$
- Therefore, we conclude that
  - $\mathcal{A}(n) > 2^n$ since it holds that
  - $\mathcal{A}(0) = 2 > 2^0 = 1$,
  - $\mathcal{A}(1) = 5 > 2^1 = 2$, and
  - $\mathcal{A}(n) = 3 + 2 \cdot \mathcal{A}(n-1) > 3 + 2 \cdot 2^{n-1} = 3 + 2^n > 2^n$
- This completes the proof

Remark

- The exponential running time and the problems with cyclical rule sets can be avoided by using breath first search
- However, this may lead to exhaustive memory consumptions