Indian Journal of Science and Technology, Vol 8(35), DOI: 10.17485/ijst/2015/v8i35/74679, December 2015
ISSN (Print) : 0974-6846
ISSN (Online) : 0974-5645

# Modeling Automatic Generator of Optimal and Minimal Covers of Functional Dependency

**Samad Najjar Ghabel and Saeid Pashazadeh***

Faculty of Electrical and Computer Engineering, University of Tabriz, Tabriz, Iran;
samad.najjar92@ms.tabrizu.ac.ir, pashazadeh@tabrizu.ac.ir

## Abstract

Functional Dependency (FD) rules arise from applications semantic and enforcing integrity of minimal cover of FDs by normalization or triggers is sufficient condition for guaranteeing integrity of database. Formal achievement of minimal and optimal FDs from initial FDs is accomplished in this paper. Process of computing optimal and minimal FDs is presented and modeled by Colored Petri Net (CPN). Proposed model iteratively applies a subset of Armstrong's axioms and infers FDs that are origin of all other FDs. Execution of model automatically is stopped when markings of model's places do not change in iteration. Minimal and optimal FDs are automatically computed and stored in two different places of the model. Although CPN is used in many areas but modeling automatic generation of optimal and minimal cover set using CPN is a novel work. Proposed CPN model in this paper can be considered as automatic proof generator too. Model is designed such that state space graph of model is very small and is generated quickly. Colour sets are declared such that starting from initial markings, step by step proof of inferring optimal and minimal cover FDs can be extracted automatically. Proposed model can be used as a simple tool for automatically computing optimal and minimal FDs of a set of initial FDs.

**Keywords:** Coloured Petri Net, Functional Dependency, Modeling, Optimal Cover, Verification

## 1. Introduction

Enforcing integrity constraints of FDs using triggers decreases performance of Database Management Systems (DBMS). All FDs of a database can be inferred from small set of FDs named original FDs. Integrity constraints of all FDs will be enforced by defining triggers for original FDs. Normalization of tables based on the minimal cover FDs is one of the main responsibilities of Database Administrator (DBA) team. Automatic generation of optimal and minimal covers of FDs help DBA team. A tool that automatically and formally generates optimal and minimal covers of a set of FDs is demanded.

CPN has used as a powerful method for modeling and formal investigation of various concurrent systems[1-4]. Improving performance of workflow systems by proposed novel effective methods of workflow scheduling called phased method is investigated by CPN too[5].

In[6] investigated computing minimum cover (irreducible cover) of a relational database and presented a parallel algorithm for computing third normal form. A better performance was gained in comparison with serial computation and explained problems of database normalization algorithm in parallel computation. Functional dependencies with the mathematical approach were investigated in several studies[7-9].

A formal approach for definition and design of conceptual schemata for database systems was proposed in[10]. In[11] proposed a new approach to the design of relational database schemes. Their approach was combination of synthesis approaches and traditional decomposition. Their approaches automatically correct schemes with lack of certain desirable properties. Fagin introduced fourth normal form that is more powerful than earlier approaches. His novel method is semi-automatic and is useful for huge databases[12]. Other traditional synthesis

*Author for correspondence

approaches have been proposed in various published papers[13-15].

In[9] combined previous algorithms in order to produce an optimal minimal cover in a systematic manner. Automatic computation of new FDs which can be inferred from initial FDs using Armstrong axioms[16] was modeled by CPN[1]. Formal proof of inferring new FDs from existing FDs has been extracted from state space analysis of this model. Various tools for CPN modeling and analysis exists that one of the best implemented tools is CPN tools[17,18] that is used in this paper.

At this part of paper two basic definition of CPN is presented.

## 1.1 Definition 1

A nonhierarchical Colored Petri Net is a nine-tuple CPN = (P, T, A, Σ, V, C, G, E, I), where[19]:

- *P*: A limited set of places.
- *T*: A limited set of transitions *T* such that $P \cap T = \emptyset$.
- $A \subseteq (P \times T) \cup (T \times P)$ is a set of arcs. CPN do not allow parallel arcs.
- Σ: A limited non-Null set of color sets.
- *V*: A limited set of typed variables such that $Type[v] \in \Sigma$ for all variables $v \in V$.
- *C*: $P \rightarrow \Sigma$ denotes a color set function that determines the color set which is assigned for each place.
- *G*: $T \rightarrow EXPR_V$ denotes a guard function for each transition *t* such that $Type[G(t)] = Bool$.
- *E*: $A \rightarrow EXPR_V$ denotes an arc expression function that assigns an arc expression to each arc *a* such that $Type[E(a)] = C(p)_{MS}$, where *p* is jointed place to the arc *a*. $C(p)_{MS}$ is all multi set on $C(p)$.
- *I*: $P \rightarrow EXPR_{\emptyset}$ denotes an initialization function that assigns an initialization expression to each place *p* such that $Type\ [I(p)] = C(p)_{MS}$.

## 1.2 Definition 2

The following concept can be defined for CPN = (*P, T, A, Σ, V, C, G, E, I*)[19]:

- A marking may be defined as a function *M* that maps each place $p \in P$ into a multi set of tokens $M(p) \in C(p)_{MS}$.
- The term initial marking $M_0$ refers to $\forall\ p \in P$: $M_0(p) = I(p)$.
- The variables of a transition *t* can be defined as $Var(t) \subseteq V$ and includes free variables that is appeared in the guard of *t* and in the arc expressions of arcs linked to *t*.
- A binding of a transition *t* may be defined as a function that maps each variable $v \in Var(t)$ into a value $b(v) \in Type[v]$. The term $B(t)$ refers to the set of all bindings for a transition *t*.
- A binding element can be defined as a pair $(t,b)$ such that $t \in T$ and $b \in B(t)$. Whereas $BE(t)$ refers to the set of all binding elements for a transition *t*, which can be defined by $BE(t)=\{(t, b) | \ b \in B(t)\}$. *BE* is the set of all binding elements in a CPN model.
- A step $Y \in BE_{MS}$ refer to a non-empty, finite multi set of binding elements.

Main aim of this paper is automatic computation of optimal and minimal cover of a set of FDs. Following rules are used for computing optimal cover of FD[20]:

- Decompose each FD rule such that left side of each rule involves single attribute.
- Each attribute in the successor part of an FD rule cannot change closure set Z+ (more explanation of Z+ is presented in[20]).
- Remove repeated FDs, and only keep one of them in the set of FDs.

Set of FDs is irreducible (minimal) set, when following conditions holds true[20]:

- This set be optimal.
- FDs that can be concluded using transitivity rule are eliminated.

In order to find optimal FDs, transition and decomposition rules will be used. Activity diagram of the whole process of the proposed model is as shown in Figure 1. Initial dependencies are presented to model at the first step. Then, from these dependencies, optimal dependencies will be computed by applying three main functions named transition, decomposition, and optimization. Afterwards, irreducible dependencies will be achieved using the optimal dependencies and the special functions that their descriptions are presented at follow. If there is dependency between A and B, and there is dependency from B to C, then the Set of Obvious Functional Dependencies (SOFD) function inserts A to C dependency to SOFD List. As is shown in Figure 1, in order to compute all irreducible FD rules a loop is designed. If new sate can be added to state space in compression with

generated sate space after previous processing of FDs, the main loop continues. If new state cannot be added to state space in compression with sate space of previous processing of FDs, loop finishes and model reports irreducible set through Irreducible set place as is shown in Figure 2.

The overall structure of paper contains five sections, with this introductory section. Section two begins with introducing preliminary elements that is used in construction of our proposed model. The third section is concerned with the top level functions of the model and shows the structure charts of model's function. In fourth section, the findings of the research, state space graph and model checking of case study model is presented. Finally, conclusion gives a brief summary and future work of this study.

## 2. Preliminary Elements

### 2.1 Color Sets

All color sets that are used in proposed model are defined as follows:

Colset ATTRIBUTE = with A | B | C | D| E | F | H;

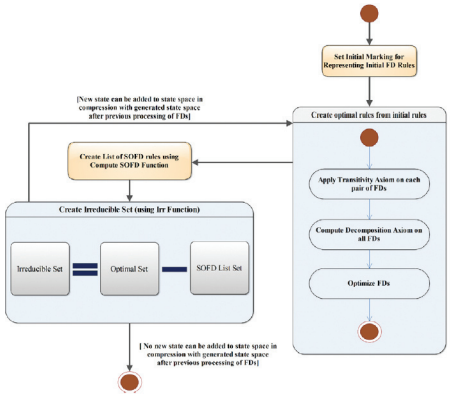


**Figure 1.** Main process of computing optimal and minimal cover set of proposed CPN model.

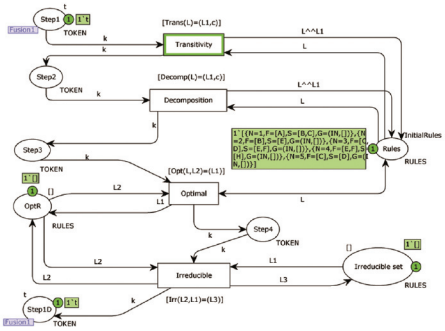


**Figure 2.** Proposed CPN model of the system.

Colset PRODRULE = with IN|DE|TR|BL|OP;
Colset PREDRULELIST = list INT;
Colset ATTRIBUTELIST = list ATTRIBUTE;
Colset RULEGENERATION = product PRODRULE *PREDRULELIST;
Colset FD = record N:INT * F:ATTRIBUTELIST * S: ATTRIBUTELIST * G:RULEGENERATION;
Colset RULES = list FD;
Colset TOKEN = with t;

Color set ATTRIBUTE is of enumerated type and represents name of attributes that are used in FDs. Colour set ATTRIBUTELIST is defined as list of attributes that is used in the successor and predecessor parts of each FD. Colour set PRODRULE is defined to represent name of Armstrong's production rule is used for generating each new FD. Table 1 displays brief description of each value of enumerated color set PRODRULE. Colour set PREDRULELIST determines a list of the number of FDs index (integer value) that a new FD rule was deduced using these FDs. Colour set PRODRULE and PREDRULELIST are only used in model checking.

Color set FD shows comprehensive form of an FD rule that contains four fields. The first field named N indicates the index of current FD. Second and third fields indicate successor (denoted by F) and predecessor (denoted by S) part of an FD rule, respectively. Fourth field is of color set RULEGENERATION shows which FDs are used for deduction of current FD rule and is denoted by G. Colour set TOKEN is defined for controlling serial execution of the transitions for decreasing size of state space of the model.

### 2.2 Initial Marking and Variables

Proposed model is presented with a case study. Initial markings of model that presents case study are as follows:
val InitialAttribs = 1`[ A,B,C,D,E,F,H];

**Table 1.** Description of abbreviations that are used in PRODRULE color set.

| Abbreviation | Description |
|---|---|
| IN | Initial FD |
| DE | Decomposition |
| TR | Transitivity |
| BL | Black List |
| OP | Optimal |

```
val InitialRules = [{N=1,F=[A],S=[B,C],G=(IN,[])},
                    {N=2,F=[B],S=[E],
    G=(IN,[])},{N=3,F=[C,D],S=[E,F],G=(IN,[])},
    {N=4, F=[E,F],S=[H],G=(IN,[])},
    {N=5, F=[C],S=[D],G=(IN,[])}];
```

InitialAttribs shows the list of the attributes that are used in the case study. Colour set of InitialAttribs is ATTRIBUTELIST. InitialRules declares the initial marking of place Rules. This initial marketing with color set RULES represents the five following FDs of case study.

1: A → BC
2: B → E
3: CD → EF
4: EF → H
5: C → D

Variable of the mode is declared as follows:

```
var L,L1,L2,L3 : RULES;        var c : BOOL;
var al: ATTRIBUTELIST;         var k: TOKEN;
```

## 2.3 Proposed Model of System

Proposed CPN model of system is as shown in Figure 2. Place Rules shows all current rules that we want to work on them, and after running, it gives us transitivity and decomposition features. Place OptR gives optimal rules of FDs that exist in Rule's place. Place irreducible shows minimal FDs of initial FDs. Places step1 and step1D are fusion places in order to generate main loop, and places step2, step3 and step4 can help us to enforce sequential firing of transitions for decreasing size of state space of mode. This model contains four main functions that exist in guard conditions of transitions. Each function calls some preliminary function. All these functions are explained in the following section, and structure chart of these functions is presented in Figure 3.

CPN models are usually faced with state space explosion problem[21]. Against this problem, few effective techniques are used in design of the model. In CPN, permutations of a list member are considered as different tokens.

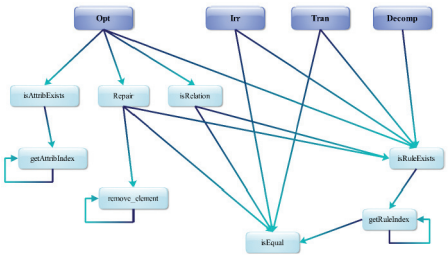


**Figure 3.** Structure chart of model's functions.

The first proposed technique is; order of members in the list is considered in a way that multiple permutations of list members and different tokens are not generated. As a result, a single meaning (marking) of model does not generate multiple states in the state space graph of the model. Second technique is defining more functionality of model using coding in ML language[22]. Coding in high level ML language caused the number of model places decreases significantly. This technique respectively decreases the size of state space of the model. Third proposed technique is sequencing enabling of transitions to decrease size of state space graph of system.

## 2.4 Preliminary Function of Model

Several preliminary functions are needed for writing the main functions of the model. At first, preliminary functions of the model are introduced and explained. Of course, structure chart of the functions is provided in the next section for better understanding of the relationship between the main functions and preliminary functions. In all of the functions, it is assumed that attributes in left and right side of FDs are not repeated. Moreover, sub-attributes can be related to the list of attributes in the same class, but in rules where left side sub-attribute is more than 2 attributes, two or more sub-attributes cannot be found from left side with a relation to other attributes in the same class. This limitation can be eliminated by improving optimization functions, which can be a task for the future studies.

Some preliminary functions such as isEqual, gerRuleIndex, isRuleExists, getAttribIndex and isAttribExists of proposed model is reused from our previous published paper and complete explanation and ML code of them were presented in that paper[1]. Function isEqual takes two attribute lists and compare these two lists. If they are equal, it returns true and otherwise returns false[1]. Function gerRuleIndex finds the position of a special FD in the list of rules[1]. Function getAttribIndex takes two parameters. First, a special attribute that searches the attribute list which is taken as the second parameter[1]. Function isRuleExists is used with gerRuleIndex to find special rules in the list of rules and returns result with Boolean type. Function isAttribExists takes two parameters in order to find special attributes in the attribute list[1]. Function remove_element takes a list of attributes as the first parameter and a special attribute as the second parameter. This recursive function returns a list of attributes that remove the attribute (second parameter) from it.

```
fun remove_element(list:ATTRIBUTELIST, element: ATTRIBUTE):ATTRIBUTELIST = case list of [] => []
 | list_head::list_tail => let val a = remove_element(list_tail, element)
    in
      if list_head = element then a
      else list_head::a
    end
```

Function isRelation takes a list of rules that exist in rules place as the first parameter and a list of attributes that we want to optimize as the second parameter. This function searches in the attributes list to find an attribute that has a relation with sub attributes in this list.

```
fun isRelation (L: RULES,LA:ATTRIBUTELIST):bool=
    let val rsu=ref 0
        val Brsu=ref false
        val n1=List.length(LA)
        val nr = ref 0
        val i = ref 0
        val j = ref 0
    in
    while !i < n1 do (
     let val XX= List.nth(LA,!i)
      in  j:= 0;
       while !j < n1 do (
         let val ZZ = List.nth(LA,!j)
           val t1={N=(!nr),F= [XX],S=[ZZ],G=(OP,[])}
            in  if isRuleExists(t1,L)andalso
              not(isEqual([XX],[ZZ]))then
           rsu := !rsu + 1
        else ()
       end;
            j := !j + 1) (* while j *)
        end;
      i := !i + 1); (* while i *)
      if !rsu = 0 then
        Brsu := true
      else();
    !Brsu
   end;
```

Function repair takes a list of attributes. It compares all paired attributes using two nested loops. If functional dependency was found for this pair of attributes (from FDs taken from input), the attribute which is found in the right side of this dependency will be eliminated in order to optimize attributes.

```
fun repair(f1:FD,L: RULES, LA:ATTRIBUTELIST) :FD=
  let val rsu=ref 0
      val Brsu=ref false
      val n1=List.length(LA)
      val nr = ref 1
      val i = ref 0
      val j = ref 0
      val L4 = ref []
      val n7=List.length(L)
  in
     while !i < n1 do (
       let val XX= List.nth(LA,!i)
       in  j:= 0;
          while !j < n1 do (
            let val ZZ = List.nth(LA,!j)
               val t1 = { N= #N f1,F = [XX],
                     S=[ZZ],G=(OP,[#N f1])}
               val RE=remove_element(#F f1,ZZ)
             in  if isRuleExists(t1,L)andalso
                    not(isEqual([XX],[ZZ]))
               then
               L4 := !L4^^[{N= #N f1,F = RE, S=(#S f1),
                   G=(OP,[#N f1,getRuleIndex(t1,L)+1])}]
               else ()
             end;
           j := !j + 1) (* while j *)
        end;
      i := !i + 1); (* while i *)
      if !rsu = 0 then
          Brsu := true
      else();
    (List.nth(!L4,0))
end;
```

## 3. Top Level Functions of the Model

The main functions of the model in the current paper are investigated in this section which includes two rules from Armstrong rules implemented in[1]. Function Tran and Decomp are retrieved from the previous work[1]. In addition, some functions will be used for optimization and finding irreducible relationships. These functions are guard conditions of transitions in Figure 2 which include a list of rules, and finally provide a list of rules in due places. In Armstrong functions, a C variable of binary type is transformed which indicates whether a new rule is inferred or not.

Function Opt has the responsibility of generating optimal FD rules using current FD rules. This function

takes all rules and updates the rules in OptR place. This function calls the functions repair, isRelation, isRuleExists and isAttribExists.

```
fun Opt(L: RULES,L4:RULES) : RULES  =
  let val L2 = ref []
     val n = List.length(L)
     val i = ref 0
     val k=ref 0
  in
     while !i < n do (
       let val F1 = List.nth(L,!i)
          val Dd= List.nth(#S F1,0)
       in
          if List.length(#S F1) = 1 andalso List.length(#F
               F1)=1 andalso not(isEqual(#S F1,#F F1))
          then
            L2:= !L2^^[F1]
          else  if List.length(#S F1) = 1
                 andalso List.length(#F F1) > 1 then (
                 if not (isAttribExists(Dd,#F F1)) andalso
                     isRelation(L,#F F1) then
                       L2:= !L2^^[F1]
                 else if not(isRelation(L,#F F1)) then
                       L2:= !L2^^[repair(F1,L,#F F1)]
                    else()
               )
              else()
    end;
     i := !i + 1 ); (* while i *)
  !L2
  end;
```

Function *Irr* provides a list of rules which should not be in irreducible rules from optimal rules, the rules which are present in optimal list and are not present in this list will be returned as irreducible rules.

```
fun Irr(OptR: RULES , BR:RULES) : RULES  =
 let val L2 = ref []
val L3=ref []
val n = List.length(OptR)
val i = ref 0
val k=ref 0
val j = ref 0
val cs = ref 0
val Found = ref false
val Gen = ref false
val nr = ref 0
 in
nr := n+1;
while !i< n do (
      let  val F1 = List.nth(OptR,!i)
 in j := 0;
   while !j < n do (
if !i < !j then
       let val F2 = List.nth(OptR,!j)
         val ta={N=(!nr),F=(#F F1),S=( #S F2),G=(BL,[#N
F1,#N F2])}
         val tb={N=(!nr),F=(#F F2), S=(#S F1),G=(BL,[#N
F1, #N F2])}
         in  if isEqual(#S F1 , #F F2) then
              ( cs :=1;
               Gen := true )
            else if isEqual(#S F2, #F F1)then
                ( cs := 2;
                  Gen := true )
               else ( cs := 0;
                    Gen := false);
                    case (!cs) of
                                  1=>  if  !Gen=true  andalso
not(isRuleExists(ta,!L2))
                 then
                      ( L2 := !L2 ^^ [ta] ;
                       nr := !nr +1;
                        Found := true )
                  else ()
     | 2 =>if !Gen=true andalso not(isRuleExists(tb,!L2))
                 then
                      ( L2 := !L2 ^^ [tb];
                       nr := !nr +1;
                       Found := true )
                    else ()
              |0 => ()
              end
            else ( );
        j := !j + 1 ) (* while j *)
     end;
    i := !i + 1 ); (* while i *)
    while !k< n do (
     let val FF = List.nth(OptR,!k)
     in
        if  not(isRuleExists(FF,!L2))  then
            L3 := !L3 ^^ [FF]
        else()
     end;
    k := !k + 1 ); (* while k *)
!L3
end;
```

Figure 3 illustrates the structure chart of model's functions. Top row of the Figure shows high level functions that call preliminary functions which are shown below. Source of arrow shows caller function.

## 4. State Space Graph and Model Checking of Case Study Model

A sample case study that was explained in section 2.2 is modeled in this paper. Report of state space of the model is as follows:

State Space
Nodes: 11
Arcs: 11
Secs: 0
Status: Full
Scc Graph
Nodes: 8
Arcs: 7
Secs: 0
Home Properties

---

Home Markings
[8,9,10,11]
Liveness Properties

---

Dead Markings
None
Dead Transition Instances
None
Live Transition Instances
All

This report shows that our proposed model has small state space graph and runs very quickly. Figure 4 displays complete state space graph of the model.

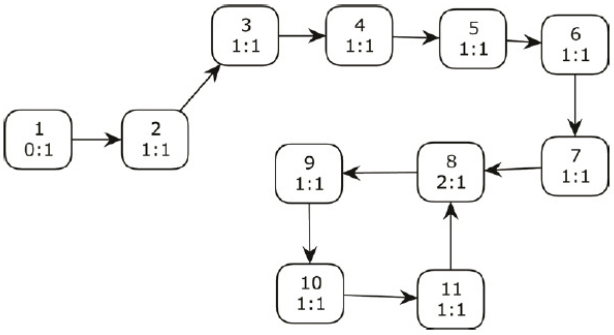


**Figure 4.** State spaces graph of CPN model for case study's markings.

After generation of state space of model, full description of each state space node can be extracted using some build-in functions of CPN. Following code extracts complete description of state n = 8 of state space.

```
let
  val file_id = TextIO.openOut("d:/outputfile.txt")
 in
  TextIO.output(file_id, NodeDescriptor 8);
  TextIO.closeOut(file_id)
end;
```

Full description of a state space node displays marking of the system at that state. Full description of state number 8 of Figure 4 is as follows.

8:
FD'Step1 1: empty
FD'Step2 1: empty
FD'Step1D 1: empty
FD'Rules 1: 1`[{N=1,F=[A],S=[B,C],G=(IN,[])},{N=2,F=[B],S=[E],G=(IN,[])},{N=3,F=[C,D],S=[E,F],G=(IN,[])},{N=4,F=[E,F],S=[H],G=(IN,[])},{N=5,F=[C],S=[D],G=(IN,[])},{N=6,F=[C,D],S=[H],G=(TR,[3,4])},{N=7,F=[A],S=[B],G=(DE,[1])},{N=8,F=[A],S=[C],G=(DE,[1])},{N=9,F=[C,D],S=[E],G=(DE,[3])},{N=10,F=[C,D],S=[F],G=(DE,[3])},{N=11,F=[A],S=[E],G=(TR,[2,7])},{N=12,F=[A],S=[D],G=(TR,[5,8])}]
FD'OptR 1: 1`[{N=2,F=[B],S=[E],G=(IN,[])},{N=4,F=[E,F],S=[H],G=(IN,[])},{N=5,F=[C],S=[D],G=(IN,[])},{N=6,F=[C],S=[H],G=(OP,[6,5])},{N=7,F=[A],S=[B],G=(DE,[1])},{N=8,F=[A],S=[C],G=(DE,[1])},{N=9,F=[C],S=[E],G=(OP,[9,5])},{N=10,F=[C],S=[F],G=(OP,[10,5])},{N=11,F=[A],S=[E],G=(TR,[2,7])},{N=12,F=[A],S=[D],G=(TR,[5,8])}]
FD'Step3 1: empty
FD'Step_4 1: 1`t
FD'Irreducible_set 1: 1`[{N=2,F=[B],S=[E],G=(IN,[])},{N=4,F=[E,F],S=[H],G=(IN,[])},{N=5,F=[C],S=[D],G=(IN,[])},{N=6,F=[C],S=[H],G=(OP,[6,5])},{N=7,F=[A],S=[B],G=(DE,[1])},{N=8,F=[A],S=[C],G=(DE,[1])},{N=9,F=[C],S=[E],G=(OP,[9,5])},{N=10,F=[C],S=[F],G=(OP,[10,5])}]

Full description of state 11 of Figure 4 is as follows.

11:
FD'Step1 1: empty
FD'Step2 1: empty
FD'Step1D 1: empty
FD'Rules 1: 1`[{N=1,F=[A],S=[B,C],G=(IN,[])},{N=2,F=[B],S=[E],G=(IN,[])},{N=3,F=[C,D],S=[E,F],G=(IN,[])},{N=4,F=[E,F],S=[H],G=(IN,[])},{N=5,F=[C],S=[D],G

=(IN,[])},{N=6,F=[C,D],S=[H],G=(TR,[3,4])},{N=7,F=[A],S=[B],G=(DE,[1])},{N=8,F=[A],S=[C],G=(DE,[1])},{N=9,F=[C,D],S=[E],G=(DE,[3])},{N=10,F=[C,D],S=[F],G=(DE,[3])},{N=11,F=[A],S=[E],G=(TR,[2,7])},{N=12,F=[A],S=[D],G=(TR,[5,8])}]

FD'OptR 1: 1`[{N=2,F=[B],S=[E],G=(IN,[])},{N=4,F=[E,F],S=[H],G=(IN,[])},{N=5,F=[C],S=[D],G=(IN,[])},{N=6,F=[C],S=[H],G=(OP,[6,5])},{N=7,F=[A],S=[B],G=(DE,[1])},{N=8,F=[A],S=[C],G=(DE,[1])},{N=9,F=[C],S=[E],G=(OP,[9,5])},{N=10,F=[C],S=[F],G=(OP,[10,5])},{N=11,F=[A],S=[E],G=(TR,[2,7])},{N=12,F=[A],S=[D],G=(TR,[5,8])}]

FD'Step3 1: 1`t

FD'Step_4 1: empty

FD'Irreducible_set 1: 1`[{N=2,F=[B],S=[E],G=(IN,[])},{N=4,F=[E,F],S=[H],G=(IN,[])},{N=5,F=[C],S=[D],G=(IN,[])},{N=6,F=[C],S=[H],G=(OP,[6,5])},{N=7,F=[A],S=[B],G=(DE,[1])},{N=8,F=[A],S=[C],G=(DE,[1])},{N=9,F=[C],S=[E],G=(OP,[9,5])},{N=10,F=[C],S=[F],G=(OP,[10,5])}]

Comparing full description of states 8,9,10, and 11 shows that, tokens of places OptR, Rules and Irreducible_set in all of these states are exactly the same. Only tokens of places Step1, Step2, Step3 and Step4 differ from each other. As a sample, full descriptions of states 8 and 11 are shown. This means that the process of computing optimal and irreducible FDs was converged and yields a single result. Irreducible set of FDs that is computed automatically by the model and can be extracted from marking of place Irreducible_set of states 8-11 is summarized as follows:

B → E
EF → H
C → D
C → H
A → B
A → C
C → E
C → F

Computed optimal FD set of the model that is stored in place *OptR* of Figure 2 in states 8-11 is as follows:

B → E
EF → H
C → D
C → H
A → B
A → C
C → E
C → F
A → E
A → D

Model is designed such that if proof of each minimal FDs will be required, it can be extracted by model checking of state space graph of model. Our proposed model can be used as automatic proof generator. Table 2 shows marking of three important places of the model in state one up to four of the system. For more clarity, new FDs that are added to each place of a state are shown with bold, italic, and underline font. Brief name of rule and FDs that are used for inferring new FD is shown at first appearance of new FD rule. Transitions from states 1 up to 4 are result of firing transitions Transitivity, Decomposition and Optimal respectively.

Table 3 shows marking of three important places of the model in state 5 up to 8 of the system. Transitions from states 4 up to 8 are result of firing transitions Irreducible, Transitivity, Decomposition, and Optimal respectively. As is shown in Table 3 and Table 4, all information for extracting automatic proof of each new FDs exists in the state space of mode. Extracting proof of FDs was presented in other paper[1].

**Table 2.** Brief description of states 1 up to 4 of state space.

| Places | State Number | | | |
|---|---|---|---|---|
| | 1 | 2 | 3 | 4 |
| Rules | 1. A→BC<br>2. B→E<br>3. CD→EF<br>4. EF→H<br>5. C→D | 1. A→BC<br>2. B→E<br>3. CD→EF<br>4. EF→H<br>5. C→D<br>*6. TR(3,4): CD→H* | 1. A→BC<br>2. B→E<br>3. CD→EF<br>4. EF→H<br>5. C→D<br>6. CD→H<br>*7. DE(1): A→B*<br>*8. DE(1): A→C*<br>*9. DE(3): CD→E*<br>*10. DE(3):CD→F* | 1. A→BC<br>2. B→E<br>3. CD→EF<br>4. EF→H<br>5. C→D<br>6. CD→H<br>7. A→B<br>8. A→C<br>9. CD→E<br>10. CD→F |
| OptR | | | | *2. B→E*<br>*4. EF→H*<br>*5. C→D*<br>*6. Op(5,6): C→H*<br>*7. A→B*<br>*8. A→C*<br>*9. Op(5,9): C→E*<br>*10. Op(5,10): C→F* |
| Irreducible_ set | | | | |

**Table 3.** Brief description of states 5 up to 8 of state space.

| Places | State Number | | | |
|---|---|---|---|---|
| | 5 | 6 | 7 | 8 |
| Rules | 1. A→BC | 1. A→BC | 1. A→BC | 1. A→BC |
| | 2. B→E | 2. B→E | 2. B→E | 2. B→E |
| | 3. CD→EF | 3. CD→EF | 3. CD→EF | 3. CD→EF |
| | 4. EF→H | 4. EF→H | 4. EF→H | 4. EF→H |
| | 5. C→D | 5. C→D | 5. C→D | 5. C→D |
| | 6. CD→H | 6. CD→H | 6. CD→H | 6. CD→H |
| | 7. A→B | 7. A→B | 7. A→B | 7. A→B |
| | 8. A→C | 8. A→C | 8. A→C | 8. A→C |
| | 9. CD→E | 9. CD→E | 9. CD→E | 9. CD→E |
| | 10. CD→F | 10. CD→F | 10. CD→F | 10. CD→F |
| | | *11. TR(2,7): A→E* | 11. A→E | 11. A→E |
| | | *12. TR(5,8): A→D* | 12. A→D | 12. A→D |
| OptR | 2. B→E | 2. B→E | 2. B→E | 2. B→E |
| | 4. EF→H | 4. EF→H | 4. EF→H | 4. EF→H |
| | 5. C→D | 5. C→D | 5. C→D | 5. C→D |
| | 6. C→H | 6. C→H | 6. C→H | 6. C→H |
| | 7. A→B | 7. A→B | 7. A→B | 7. A→B |
| | 8. A→C | 8. A→C | 8. A→C | 8. A→C |
| | 9. C→E | 9. C→E | 9. C→E | 9. C→E |
| | 10. C→F | 10. C→F | 10. C→F | 10. C→F |
| | | | | *11. A→E* |
| | | | | *12. A→D* |
| Irreducible_ set | *2. B→E* | 2. B→E | 2. B→E | 2. B→E |
| | *4. EF→H* | 4. EF→H | 4. EF→H | 4. EF→H |
| | *5. C→D* | 5. C→D | 5. C→D | 5. C→D |
| | *6. C→H* | 6. C→H | 6. C→H | 6. C→H |
| | *7. A→B* | 7. A→B | 7. A→B | 7. A→B |
| | *8. A→C* | 8. A→C | 8. A→C | 8. A→C |
| | *9. C→E* | 9. C→E | 9. C→E | 9. C→E |
| | *10. C→F* | 10. C→F | 10. C→F | 10. C→F |

## 5. Conclusion

A novel CPN model of automatically computing optimal and irreducible FD rules is presented with complete description. This model takes initial FD rules as an input and computes optimal FD rules of them. Then irreducible FD rules are generated using computed optimal FD rules. The model is designed such that proof of all results exists in state space of the model. All proofs can be extracted automatically by application dependent model checking of the system. State space explosion of model is eliminated by using novel techniques in this paper. First technique is paying attention to not generating different permutations of list members in whole functions and parts of the model. Second technique is decreasing the number of model places and defining most functionality of the model in terms of ML functions. This proposed model can be used as a toll for computing optimal and minimal cover of a set of FDs. Also, this model can be used as automatic proof generator of optimal and minimal FDs.

## 6. References


1. Pashazadeh S, Pashazadeh M. Modelling an automatic proof generator for functional dependency rules using Colored Petri Net. IJFCST. 2012; 2(5):31-47.
2. Pashazadeh S. Modelling a resource management method using hierarchical Colored Petri Net. Proceedings of the International Conference on Computer and Knowledge Engineering (ICCKE2011); Ferdowsi University of Mashhad, Mashhad, Iran. 2011 Oct 13-14. p. 34-9.
3. Kim Y, Song Y, Lee J. Vehicular authentication security mechanism modelling using Petri net. Indian Journal of Science and Technology. 2015; 8(S7):443-7.
4. Meher Taj S, Kumaravel A. Survey on fuzzy Petri nets for classification. Indian Journal of Science and Technology. 2015; 8(14):1-8.
5. Xiao Z, Ming Z. A method of workflow scheduling based on Colored Petri Nets. Data and Knowledge Engineering. 2011; 70(2):230-47.
6. Omiecinski ER. A parallel algorithm for relational database normalization. IEEE Transactions on Parallel and Distributed Systems. 1990; 1(4):415-23.
7. Matus F. Abstract functional dependency structures. Theoretical Computer Science. 1991; 81(1):117–26.
8. Lovrencic A, Cubrilo M, Kisasondi T. Modelling functional dependencies in databases using mathematical logic. Proceedings of 11th International Conference on Intelligent Engineering Systems (INES 2007); Budapest, Hungary. 2007 June 29-July 2. p. 307-12.
9. Waseem AA, Hussain SJ, Shaikh ZA. An extended synthesis algorithm for relational database schema design. Proceedings of International Conference on Information Systems and Design of Communication (ISDOC'13); Lisbon, Portugal. USA: ACM New USA. 2013 July 11. p. 94-100.
10. Zaniolo C, Melkaoff M. A formal approach to the definition and the design of conceptual schemata for data-based systems. ACM Transactions on Database Systems. 1982; 7(1):24-59.
11. Beeri C, Kifer M. An integrated approach to logical design of relational database schemes. ACM Transactions on Database Systems. 1986; 11(2):134-58.
12. Fagin R. The decomposition versus synthetic approach to relational database design. Proceedings of the 3rd International Conference on Very Large Databases (VLDB'77); Tokyo, Japan. 1977 Oct. p. 441-6.

13. Ram S, Curran SM. The synthesis approach for relational database design: an expanded perspective. IEEE Proceedings of the 21st Annual Hawaii International Conference on System Sciences; Kailua-Kona, HI, USA. 1988 Jan 5-8. p. 571-80.
14. Silberschatz A, Korth HF, Sudarshan S. Database system concepts. 6th ed. Boston: McGraw-Hill; 2010. p. 338-61.
15. Maier D. Theory of relational databases. Rockville: Computer Science Press; 1983.
16. Armstrong WW. Dependency structures of database relationships. Proceedings of IFIP Congress; Stockholm, Sweden. 1974 Aug 5-10. p. 580-3.
17. CPN Tools groups. 2015 Jan 10. Available from: http://cpn-tools.org/.
18. Department of Computer Science of AU. Cpnets. 2015 Jan 10. Available: http://cs.au.dk/CPnets/
19. Jensen K, Kristensen LM. Coloured Petri Nets: modelling and validation of concurrent systems. Berlin: Springer; 2009.
20. Date CJ. An introduction to database systems. 8th ed. Pearson; 2003. p. 333-45.
21. Pashazadeh S. Modeling and verification of deadlock potentials of a concurrency control mechanism in distributed databases using hierarchical colored Petri Net. IJIET. 2012; 2(2):77-82.
22. Paulson LC. ML for the working programmer. USA: Cambridge University Press; 1996.