Adder Designer – Tools for Modelling and Analysis of Rule-based Systems

Marcin Szpyrka

AGH University of Science and Technology, Al. Mickiewicza 30, 30-059 Kraków
email: mszpyrka@agh.edu.pl

Abstract *The paper presents a formal approach to design and analysis of rule-based systems in the form of generalised decision tables. The approach is supported by a computer tool called* Adder Designer*, equipped with decision tables editor and verification algorithms. It allows for designing a reliable rule-based system that finally may be incorporated into an RTCP-net model (a Petri net model). The relevant definitions and main properties of generalised decision tables are presented, and main features of the* Adder Designer *are put forward. (*Supported by KBN Research Project No 4 T11C 035 24*)*

1. Introduction Monitoring and control systems are an important class of embedded systems. They check sensors providing information about the system's environment and take actions depending on the sensor reading. An important part of such a system is a control process that makes decisions based on collected data. The control process may be implemented to use a rule-based system to make decisions. The paper focuses on the design and analysis of such rule-based systems for embedded control systems.

In most basic versions, a rule-based system for control or decision support consists of a single-layer set of rules and a simple inference engine; it works by selecting and executing a single rule at a time, provided that the preconditions of the rule are satisfied in the current state. A rule-based system can be represented as a single decision table with rows labelled with rules' numbers and columns labelled with attributes' names. Each cell in such a decision table contains a single atomic value of the corresponding attribute. Such decision tables are often called *attributive decision tables with atomic values of attributes* (see [1], [3], [4]).

Encoding decision tables with use of atomic values of attributes only is not sufficient for many realistic applications. If the domains of attributes contain more than several values it may be really hard to cope with the number of decision rules. To handle the problem one can use formulae instead of atomic values of attributes. In such a case, a cell in a decision table will contain a formula that evaluates to a boolean value for conditional attributes, and to a single value (that belongs to the corresponding domain) for decision attributes. The result of this approach is a decision table with generalised decision rules (or rules' patterns). Each generalised decision rule covers a set of decision rules with atomic values of attributes (simple decision rules). Therefore, the number of generalised decision rules is significantly less than the number of the simple ones.

Adder Designer allows designing tables with both simple and generalised decision rules. Moreover, the tool is equipped with transformation algorithms that allow users to convert a decision table with generalised decision rules into a table with simple ones and to glue two or more simple rules into a generalised one. Finally, Adder Designer enables users to verify selected qualitative properties of decision tables such as completeness, consistency and optimality.

The paper is organised as follows. Section 2 deals with a short description of decision tables with atomic values of attributes. Decision tables with generalised decision rules and their basic properties are presented in section 3. Adder Designer is described in section 4. The paper ends with concluding remarks and a short summary in the final section.

2. Decision tables with atomic values of attributes The basic form of a decision rule is as follows:

$$\begin{array}{l} IF < preconditions > \\ THEN < conclusions >, \end{array} \tag{1}$$

where $< preconditions >$ is a formula defining when the rule can be applied, and $< conclusions >$ is the definition of the effect of applying the rule; it can be a logical formula, a decision or an action.

Let $\mathcal{A}$ denote a set of attributes selected to describe important features of the system under consideration, i.e., conditions and actions, $\mathcal{A} = \{A_1, A_2, ..., A_n\}$. For any attribute $A_i \in \mathcal{A}$, let D_i denote a domain (finite set of possible values) of A_i. It can be assumed that D_i contains at least two different elements. The set $\mathcal{A}$ is divided into two parts.

$\mathcal{A}_c = \{A_{c_1}, A_{c_2}, \ldots, A_{c_k}\}$ will denote the set of *conditional attributes*, and $\mathcal{A}_d = \{A_{d_1}, A_{d_2}, \ldots, A_{d_m}\}$ will denote the set of *decision attributes*. For the sake of simplicity it will be assumed that $\mathcal{A}_c$ and $\mathcal{A}_d$ are non-empty, finite, and ordered sets. Therefore, a decision rule with atomic values of attributes (a simple decision rule) takes the following form:

$$\begin{aligned}&(A_{c_1} = a_{c_1}) \wedge \ldots \wedge (A_{c_k} = a_{c_k}) \Longrightarrow \\ &(A_{d_1} = a_{d_1}) \wedge \ldots \wedge (A_{d_m} = a_{d_m}),\end{aligned} \quad (2)$$

where $a_{c_i} \in D_{c_i}$, for $i = 1, 2, \ldots, k$ and $a_{d_i} \in D_{d_i}$, for $i = 1, 2, \ldots, m$.

A set of simple decision rules can be represented as a *simple decision table*. To construct such a decision table, we draw a column for each conditional and decision attribute. Then, for every possible combination of values of conditional attributes a row should be drawn. We fill cells so as to reflect which actions should be performed for each combination of conditions. Let $\mathcal{R} = \{R_1, R_2, \ldots, R_l\}$ denote the set of all decision rules. A general scheme of such a decision table is as follows:

$$\begin{array}{|c|ccc|ccc|} \hline & A_{c_1} & \ldots & A_{c_k} & A_{d_1} & \ldots & A_{d_m} \\ \hline R_1 & a_{1c_1} & \ldots & a_{1c_k} & a_{1d_1} & \ldots & a_{1d_m} \\ R_2 & a_{2c_1} & \ldots & a_{2c_k} & a_{2d_1} & \ldots & a_{2d_m} \\ \vdots & \vdots & \ldots & \vdots & \vdots & \ldots & \vdots \\ R_l & a_{lc_1} & \ldots & a_{lc_k} & a_{ld_1} & \ldots & a_{ld_m} \\ \hline \end{array} \quad (3)$$

An example of a simple decision table with two conditional and one decision attribute is presented in Table 1. Domains for these attributes are defined as follows:
$D_A =$ int with 1..4,
$D_B =$ with $a|b$,
$D_C =$ bool with $(\mathit{off}, \mathit{on})$.

Table 1. Example of a simple decision table

	A	B	C
R_1	1	a	*off*
R_2	1	b	*on*
R_3	2	a	*off*
R_4	2	b	*on*
R_5	3	a	*on*
R_6	3	b	*on*
R_7	4	a	*on*
R_8	4	b	*on*

3. Decision tables with generalised decision rules

Let's consider the set of decision rules presented in Table 1. If the value of attribute B is equal to b the decision is allways equal to on. Therefore, instead of the rules R_2, R_4, R_6 and R_8, we can take only one rule: $(B = b) \Longrightarrow (C = on)$. The new rule is said to *cover* the rules R_2, R_4, R_6 and R_8. On the other hand, if the value of attribute A is equal to or greater than 3, the decision is also allways equal to on. Thus, instead of the rules $R_4, \ldots, R_8$, we can take the rule: $(A \geq 3) \Longrightarrow (C = on)$.

A formula for an attribute $A_i \in \mathcal{A}$ in a rule $R_j \in \mathcal{R}$ will be denoted by $R_j(A_i)$. To every attribute $A_i \in \mathcal{A}$ there will be attached a *variable* A_i that may take any value belonging to the domain D_i. A formula $R_j(A_i) \equiv A_i$ is a shorthand for $R_j(A_i) \equiv A_i \in D_i$ and it always evaluates to *true*. Table 1 can be represented in the following *condensed* form:

Table 2. Generalised decision table – version 1

	A	B	C
R_1	A	$B = b$	*on*
R_2	$A \leq 2$	$B = a$	*off*
R_3	$A \geq 3$	$B = a$	*on*

The decision table with generalised decision rules presented in Table 2 is not the only possible transformation of the Table 1. Another interesting possibility of transformation is presented in Table 3. The only difference between the tables is the modification of rule R_3. After this modification, both rules R_1 and R_3 can be applied in some states. Such a situation is not treated as a mistake.

Table 3. Generalised decision table – version 2

	A	B	C
R_1	A	$B = b$	*on*
R_2	$A \leq 2$	$B = a$	*off*
R_3	$A \geq 3$	B	*on*

It is evident that transformation of a simple decision table into a corresponding generalised decision table is ambiguous. The final version of such a table is dependent on subjective decisions of a designer. Regardless of this, a generalised decision table has to fulfill the following requirements: Each cell of a decision table should contain a formula, which evaluates to a boolean value for conditional attributes, and to a single value (which belongs to the corresponding domain) for decision attributes.

Decision rules providing a decision or conclusion will be called *positive rules*. Sometimes it is necessary to state in an explicit way that the particular combination of input values (values of conditional attributes) is impossible or not allowed. Such combinations of input values are represented as *negative rules*. For negative rules values of decision attributes are omitted. When necessary, $\mathcal{R}^+$ will be used to denote the subset of positive rules, and $\mathcal{R}^-$ will be used to denote the subset of negative rules.

To be usefull a generalised decision table should satisfy some qualitative properties such as completeness, consistency (determinism) and optimality. A decision table is considered to be *complete* if for any possible input situation at least one rule can produce a decision. A decision table is *deterministic* if no two different rules can produce different results for the same input situation. The last property means that any dependent rules were removed. Formal definitions of these properties are presented below.

Definition 1. A *transition function* is a function φ that satisfies the following conditions: $\varphi\colon \mathcal{A} \to D_1 \cup D_2 \cup \cdots \cup D_n \wedge \forall A_i \in \mathcal{A}\colon \varphi(A_i) \in D_i$.

Definition 2. A transition function φ is said to *satisfy* the conditional part of a rule $R_j \in \mathcal{R}$ ($\varphi \cong R_j|\mathcal{A}_c$) iff each formula $R_j(A_i)$, where $A_i \in \mathcal{A}_c$, evaluates to *true* for values the function φ assigns to conditional attributes.

A transition function φ is said to *satisfy* a rule $R_j \in \mathcal{R}^+$ ($\varphi \cong R_j$) iff φ satisfies the conditional part of the rule R_j and each formula $R_j(A_i)$, where $A_i \in \mathcal{A}_d$, evaluates to $\varphi(A_i)$.

Any transition function cannot be said to satisfy a negative rule.

Definition 3. The set $\mathcal{R}$ is *complete* iff for any transition function φ there exists a rule $R_i \in \mathcal{R}$ such that φ satisfies the conditional part of the rule R_i, i.e.: $\forall\varphi\, \exists R_i \in \mathcal{R}\colon \varphi \cong R_i|\mathcal{A}_c$.

Let Φ^+denote the set of all transition functions such that for any $\varphi \in \Phi^+$ and for any decision rule $R_j \in \mathcal{R}^-$, the transition function φ does not satisfy the conditional part of R_j.

Definition 4. The set $\mathcal{R}$ is *consistent* iff for any transition function $\varphi \in \Phi^+$, and any two rules $R_i, R_j \in \mathcal{R}^+$ if φ satisfies the rule R_i, and φ satisfies the conditional part of the rule R_j, then φ satisfies the rule R_j, i.e.: $\forall\varphi \in \Phi^+\, \forall R_i, R_j \in \mathcal{R}^+\colon (\varphi \cong R_i \wedge \varphi \cong R_j|\mathcal{A}_c) \Rightarrow \varphi \cong R_j$.

Definition 5. Let $\mathcal{R}$ be a complete and consistent set of decision rules. The rule $R_i \in \mathcal{R}^+$ is *independent* iff the set $\mathcal{R}-\{R_i\}$ is not complete. The rule $R_i \in \mathcal{R}^+$ is *dependent* if and only if the rule is not independent. The set $\mathcal{R}$ is *semi-optimal* if and only if any rule belonging to the set $\mathcal{R}$ is independent.

The semi-optimality should be verified after a set of rules is complete and consistent. The verification algorithm is presented in Fig. 1.

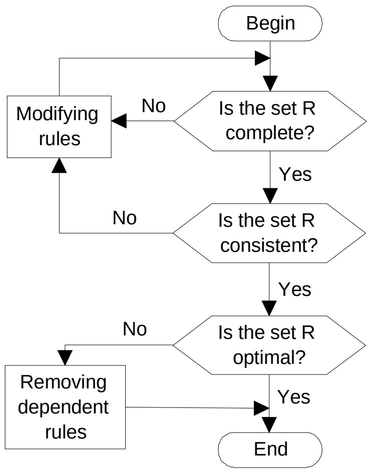
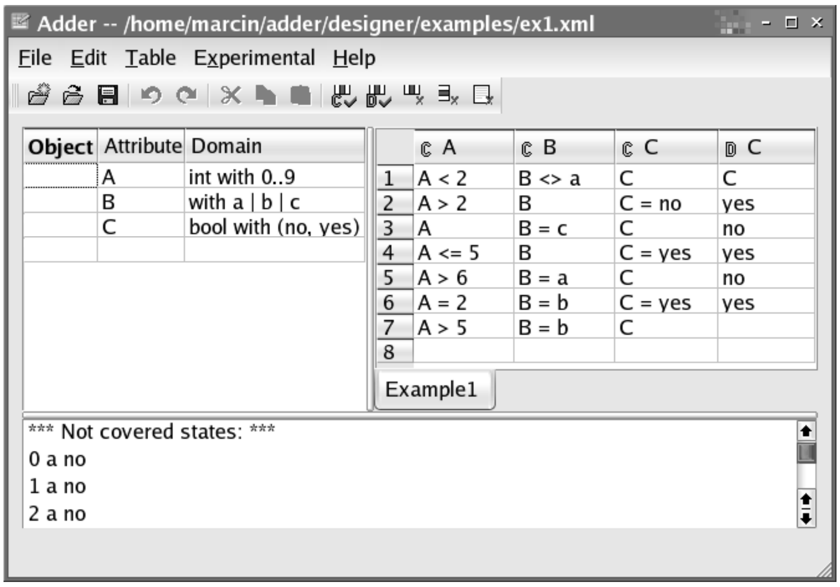


Figure 1. Scheme block of the verification procedure

4. Adder Designer Manual analysis of a decision table can be time-consuming even for very small sets of decision rules. *Adder Designer* supports design and analysis of both simple and generalised decision tables. Adder Designer is a free software covered by the GNU Library General Public License. It is being implemented in the GNU/Linux environment by the use of the Qt Open Source Edition. Qt is a comprehensive C++ application development framework. It includes a class library and tools for cross-platform development and internationalisation. The Qt Open Source Edition is freely available for the development of Open Source software for Linux, Unix, Mac OS X and Windows under the GPL license. Code written for either environment compiles and runs with the other ones. *Adder Tools home page*, hosting information about current status of the project, is located at *http://adder.ia.agh.edu.pl*. An example of Adder Designer session is shown in the Fig. 2. The decision table presented in the figure contains three conditional and one decision attribute. Moreover, it contains six positive and one negative rule.

The proposed approach to designing of decision tables consists of a few steps. It is first necessary to define attributes selected to describe important features of the system under consideration. There are possible three types of domains: integer, boolean and enumerated data type. Moreover, a new domain may be defined as an alias for already defined one. Secondly, it is necessary to choose conditional and decision attributes. Each attribute can be used twice. Finally, the set of decision rules should be defined. For positive rules, each cell in the corresponding row must be filled. On the other hand, for negative rules the cells that correspond to decision attributes must stay empty.


Adder -- /home/marcin/adder/designer/examples/ex1.xml
File Edit Table Experimental Help
Example1

Object	Attribute	Domain
	A	int with 0..9
	B	with a | b | c
	C	bool with (no, yes)

	ℂ A	ℂ B	ℂ C	𝔻 C
1	A < 2	B <> a	C	C
2	A > 2	B	C = no	yes
3	A	B = c	C	no
4	A <= 5	B	C = yes	yes
5	A > 6	B = a	C	no
6	A = 2	B = b	C = yes	yes
7	A > 5	B = b	C	
8				

```
*** Not covered states: ***
0 a no
1 a no
2 a no
```

Figure 2. Example of Adder Designer session

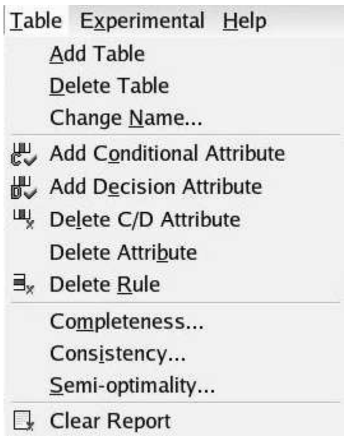


Figure 3. Table menu

All commands used for designing of decision tables are gathered in *Table* and pop-up menu. The menus are shown in Fig. 3 and Fig. 4 respectively.

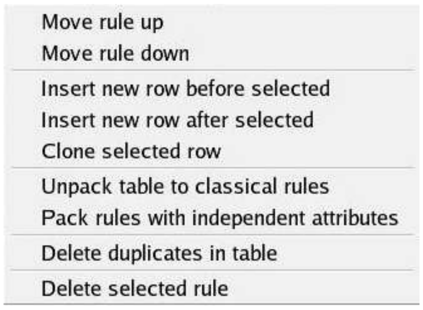

Move rule up
Move rule down
Insert new row before selected
Insert new row after selected
Clone selected row
Unpack table to classical rules
Pack rules with independent attributes
Delete duplicates in table
Delete selected rule

Figure 4. Pop-up menu

The verification stage is included into the design process. At any time, during the designing stage, users can check whether a decision table is complete, consistent or it contains some dependent rules.

Let $\mathcal{R} = \{R_1, R_2, \ldots, R_l\}$ be a set of decision rules and Φ denote the set of all transition functions. Let $\sim_c$ be an equivalence relation on Φ, such that:

$$\varphi_1 \sim_c \varphi_2 \Leftrightarrow \forall A_i \in \mathcal{A}_c \colon \varphi_1(A_i) = \varphi_2(A_i). \qquad (4)$$

Restriction of the set Φ to the set of conditional attributes is defined as follows:

$$\Phi|\mathcal{A}_c = \{\varphi|\mathcal{A}_c \colon \varphi \in \Phi\}. \qquad (5)$$

Let $\psi \in \Phi|\mathcal{A}_c$, $\varphi \in \Phi$ and $\psi = \varphi|\mathcal{A}_c$.

$$\psi \cong R_i|\mathcal{A}_c \Leftrightarrow \varphi \cong R_i|\mathcal{A}_c. \qquad (6)$$

Adder Designer uses methods based on colour Petri nets theory to check completeness, consistency and semi-optimality (see [7]). A simplified representations of these algorithms are as follows:

Completeness

```
$\Psi := \emptyset$;
for all $\psi \in \Phi|\mathcal{A}_c$ do
  covered := false;
  for all $R_i \in \mathcal{R}$ do
    if $\psi \cong R_i|\mathcal{A}_c$ then
      covered := true;
    end if
  end for
  if covered = false then
```

```
        $\Psi = \Psi \cup \{\psi\}$;
      end if
    end for
    if $|\Psi| > 0$ then
      Not covered states: $\Psi$;
    end if
```

The result of completeness analysis is a list of input states (combinations of values of conditional attributes) that are not covered by decision rules. For the decision table presented in Fig. 2 the report of completeness analysis is as follows:

```
*** Not covered states: ***
0 a no
1 a no
2 a no
2 b no
6 a yes
Table is not complete.
```

Let $\psi \in \Phi|\mathcal{A}_c$ and let $R_i \in \mathcal{R}^+$ be a positive decision rule such that $\psi \cong R_i|\mathcal{A}_c$. $\psi_i^* \in \Phi$ will be used to denote a transition function such that:

$$\forall A \in \mathcal{A}_c \colon \psi(A) = \psi_i^*(A) \wedge \psi_i^* \cong R_i. \qquad (7)$$

Consistency

```
  $\Theta := \emptyset$;
  for all $\psi \in \Phi|\mathcal{A}_c$ do
    $\mathcal{R}' := \emptyset$;
    $\Psi := \emptyset$;
    for all $R_i \in \mathcal{R}$ do
      if $\psi \cong R_i|\mathcal{A}_c$ then
        $\mathcal{R}' := \mathcal{R}' \cup \{R_i\}$;
        $\Psi := \Psi \cup \{\psi_i^*\}$;
      end if
    end for
    if $|\mathcal{R}'| > 1 \wedge \mathcal{R}' \cap \mathcal{R}^- = \emptyset \wedge |\Psi| > 1$ then
      $\Theta = \Theta \cup \{\mathcal{R}'\}$;
    end if
  end for
  if $|\Theta| > 0$ then
    Not consistent sets of rules: $\Theta$;
  end if
```

After consistency analysis users receive a list of sets of inconsistent rules. Each such a set of rules is labelled with an input state that is covered by the rules and results of applying these rules for the state are also presented. A part of consistency analysis report for the table presented in Fig. 2 is as follows:

```
*** Not consistent sets of rules: ***
State:  0 c yes
R1:   yes
R3:   no
R4:   yes
State:  1 c yes
R1:   yes
R3:   no
R4:   yes
...
```

Semi-optimality

```
  $\mathcal{R}' := \mathcal{R}$;
  for all $\psi \in \Phi|\mathcal{A}_c$ do
    $S := \emptyset$;
    for all $R_i \in \mathcal{R}$ do
      if $\psi \cong R_i|\mathcal{A}_c$ then
        $S := S \cup \{R_i\}$;
      end if
    end for
    if $|S| = 1$ then
      $\mathcal{R}' := \mathcal{R}' - S$;
    end if
  end for
  if $|\mathcal{R}'| > 0$ then
    Dependent rules: $\mathcal{R}'$;
  end if
```

The result of semi-optimality analysis is a set of dependent rules. For the considered decision table such a set contains rule R_6 only.

In addition to this the commands *Unpack table to classical rules* and *Pack rules with independent attributes* are used to convert a generalised decision table into a simple one and vice versa.

Adder Designer uses XML format to store projects. A piece of XML code describing a decision table is presented below:

```
<attribute name="A"
  domain="int with 0..9"/>
...
<table name="Example1">
  <conditionalAttributes>
    <attribute name="A"/>
    <attribute name="B"/>
    <attribute name="C"/>
  </conditionalAttributes>
  <decisionAttributes>
    <attribute name="C"/>
  </decisionAttributes>
  <rules>
    <rule>
      <formula
        expression="A &lt; 2"/>
      <formula
        expression="B &lt;&gt; a"/>
      <formula expression="C"/>
      <formula expression="C"/>
    </rule>
    ...
  </rules>
</table>
```

The XML format may be used by another tools to generate input files for Adder Designer. For example, it allows users to interchange data between Adder Designer and *Mirella* tool ([5]). Mirella is a tool for support of visual design, verification and implementation of rule-based systems. Mirella uses eXtended Tabular Trees for knowledge specification and formal properties of the developing system are verified by the external modules integrated with Prolog-based inference engine.

5. Summary *Adder Designer*, a tool for design and analysis of rule-based systems in the form of generalised decision tables, has been presented in the paper. The tool is equipped with a decision table editor and verification procedures. A survey of decision tables properties and the corresponding verification algorithms has been also presented.

Some of the algorithms are based on typical Petri nets analysis methods. The considered decision tables may be automatically transformed into an equivalent Petri net form called D-net. A D-net is a non-hierarchical coloured Petri net. D-nets may be used both to specify external system behaviour and to model a rule-based system. In the second case, D-nets constitute the bottom layer of an RTCP-net model (a Petri net model). For more details see [6].

Development of the tool is still in progress. Our future plans will focus on development of another analysis methods. Especially, we are interested in implementing faster algorithms for verification decision tables' properties. An example of such an algorithm, for completeness analysis, is placed in *Experimental* menu.

Finally, Adder Designer is released under the GPL license and everyone may develop his own verification procedures and include them into the tool.

References

[1] Davis. A.M.: *A comparison of techniques for the specification of external system bahavior*. Communication of the ACM, Vol. 31, No 9, pp. 1098–1115, 1988

[2] Gibek M.: *Wykorzystanie kolorowanych sieci Petriego do modelowania i analizy systemów regułowych*. Praca magisterska, Wydział EAIiE, AGH Kraków, 2005 (promotor: M. Szpyrka)

[3] Ligęza A.: *Logical Foundations for Rule-Based Systems.* Uczelniane Wydawnictwa Naukowo-Dydaktyczne AGH, 2005

[4] Macaulay, L. A.: *Requirements Engineering.* Springer Verlag, Berlin, 1996

[5] Nalepa G. J., Szpyrka M.: *Two Formal Approaches to Design and Verification of Embedded Rule-based Systems.* Proc. of 29th IFAC/IFIP Workshop on Real-Time Programming WRTP 2004. September 6-8, Istanbul, Turkey, 2004

[6] Szpyrka M.: *Fast and flexible modelling of real-time systems with RTCP-nets*. Computer Science, Vol. 6, pp. 81–94, 2004

[7] Szpyrka M., Szmuc T.: *D-nets – Petri Net Form of Rule-based Systems*. (submitted to Journal on Foundation of Computing and Decision Science)