Automated Robotic Process Automation: A Self-Learning Approach

Junxiong Gao[1], Sebastiaan J. van Zelst[1,2], Xixi Lu[3], and Wil M.P. van der Aalst[1,2]

[1] Chair of Process and Data Science, RWTH Aachen University, Aachen, Germany
{jx.gao,s.j.v.zelst,wvdaalst}@pads.rwth-aachen.de

[2] Fraunhofer Institute for Applied Information Technology (FIT), Sankt Augustin, Germany
{sebastiaan.van.zelst, wil.van.der.aalst}@fit.fraunhofer.de

[3] Department of Information and Computing Sciences, Utrecht University, Utrecht, the Netherlands
x.lu@uu.nl

Abstract. Robotic Process Automation (RPA) recently gained a lot of attention, in both industry and academia. RPA embodies a collection of tools and techniques that allow business owners to automate repetitive manual tasks. The intrinsic value of RPA is beyond dispute, e.g., automation reduces errors and costs and thus allows us to increase the overall business process performance. However, adoption of current-generation RPA tools requires a manual effort w.r.t. identification, elicitation and programming of the to-be-automated tasks. At the same time, several techniques exist that allow us to track the exact behavior of users in the front-end, in great detail. Therefore, in this paper, we present a novel end-to-end approach that allows for completely automated, algorithmic RPA-rule deduction, on the basis of captured user behavior. Furthermore, our proposed approach is accompanied by a publicly available proof-of-concept implementation.

Keywords: Robotic Process Automation; Information Systems; User Interaction; Data Mining; Knowledge Discovery

1 Introduction

Business process management [7] (BPM) revolves around the effective scheduling, orchestration and coordination of the different activities and tasks that comprise a (business) process. Indisputably, the end goal, or even holy grail, of any BPM practitioner is to design the most efficient process that achieves the highest possible quality for the end product and/or service. A natural question in this endeavor is related to *automation*, i.e., "What tasks are eligible for automated execution by a computer, rather than a human?".

Within BPM, the challenge of accurately automating a business process is known as *Business Process Automation* (BPA) [15]. Several researchers have

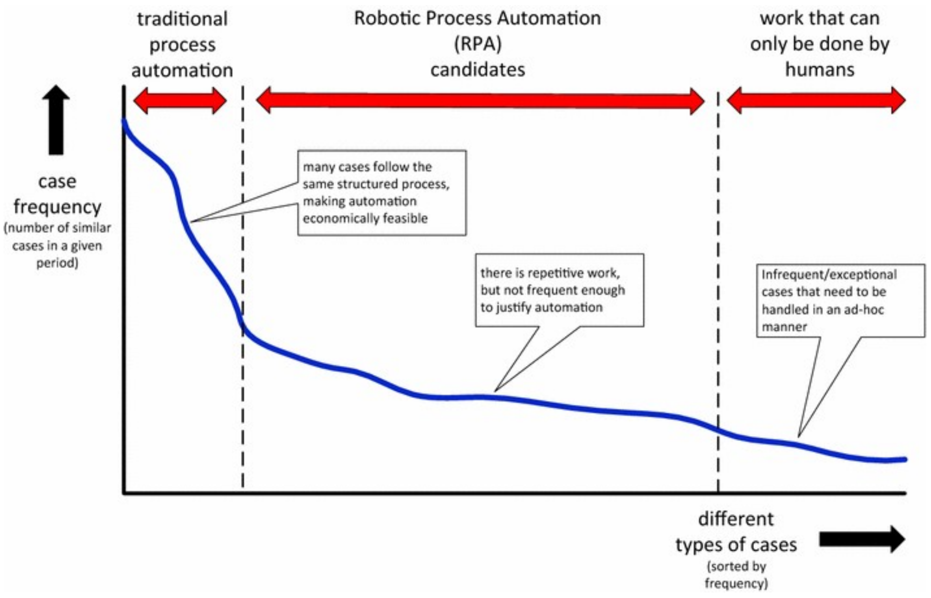

traditional process automation
Robotic Process Automation (RPA) candidates
work that can only be done by humans
case frequency
(number of similar cases in a given period)
many cases follow the same structured process, making automation economically feasible
there is repetitive work, but not frequent enough to justify automation
Infrequent/exceptional cases that need to be handled in an ad-hoc manner
different types of cases
(sorted by frequency)

Fig. 1: Positioning of candidate processes for RPA [2]. The application of RPA is most effective in processes that comprise semi-similar execution flows.

studied BPA, leading to various methodologies and techniques that enable automation in business processes. Techniques in BPA typically focus on the automation of activities and tasks from a *system perspective*, i.e., it requires a change of the configuration, or even redesign of, the information system supporting the process [6][16]. Therefore, applying BPA in practice is timely and costly, and, if applied incorrectly, can have a huge negative impact as well.

For decades, techniques have been developed that allow us to apply automation on the *user interface (UI) level of computer systems*. More recently, these techniques were adopted by *Robotic Process Automation* (RPA) [17], allowing us to mimic user behavior within a process. Since RPA operates on the user interface level, rather than on the system level, it allows us to apply automation without any changes in the underlying information system [4][19]. Hence, the entry barrier of adopting RPA in processes that are already in place, is lower compared to conventional BPA [19]. Furthermore, as motivated in [2], there are ample opportunities for the application of RPA in the context of (business) processes. For example, the application of RPA thrives in processes and process instances that consist of a medium level of process similarity, cf. Figure 1.

Current, (commercial) RPA techniques and products provide recorders and modelers that allow *manual construction of robotic behavior*. However, they are typically defined on low-level streams of user interactions (e.g., mouse clicks and keyboard strokes) [13][18]. Such techniques have three disadvantages: (1) *they require extensive domain knowledge* to identify, from the low-level interactions, which high-level tasks can be automated; (2) *such manual identification is labor*

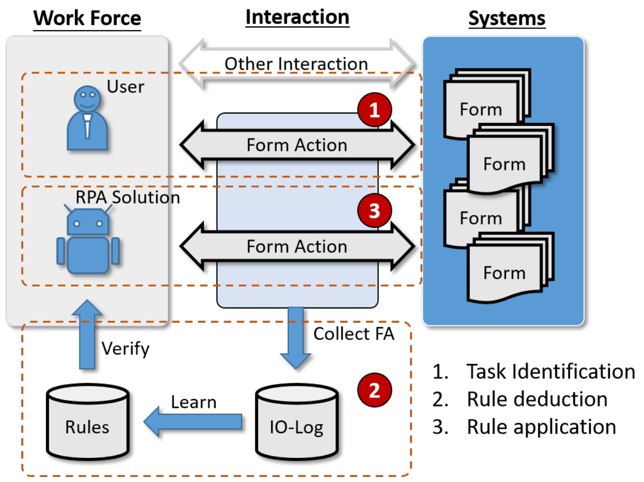


Fig. 2: The F2R approach presented in this paper, consisting of three steps.

intensive [10]; (3) there is *no support for (semi-)automated learning* from these user interactions.

In this paper, we propose a self-learning approach, i.e., the *Form-to-Rule (F2R) approach*, which automatically detects high-level RPA-rules, from captured historical low-level user behavior. Consider Figure 2, in which we schematically present the F2R approach. In the first step, (1) *tasks are identified* by observing the user's interactions with the systems, on a collection of system forms that are defined. Next, it (2) *deduces rules* by learning relations between the different tasks performed. Such *rules* are defined as "`if ... then ...`" statements [17], which is a widely adopted definition in RPA. Finally, it (3) *applies the rules* by instantiating RPA on the basis of the deduced *rules*. The "then" part of the *rule* represents actions that can be automatically executed by the RPA solution. Hence, the RPA solution listens on *forms*, if certain triggering action is observed, it activates and executes the suitable *rule*.

The F2R approach allows RPA solutions to continuously and automatically learn from user behavior. Hence, it overcomes the main bottlenecks of adopting RPA, as mentioned before. A prototypical implementation of the F2R approach and a corresponding screencast are publicly available via `https://github.com/FrankBGao/F2R_approach_RPA`.

The remainder of this paper is structured as follows. In Section 2, we review related work. In Section 3, we present basic the notations used throughout the paper. Section 4 describes the core elements of the F2R approach. Section 5 presents the corresponding algorithm and implementation for the F2R approach. Section 6 concludes the paper and provides directions for future work.

2 Related Work

RPA has received much attention in recent work in the BPM community. Here, we provide a brief overview of related work in the field.

Some techniques focus on the identification of tasks and/or process properties that are eligible to be automated using RPA, i.e., focusing on *what to automate*. In [12], the authors of [12] propose to use natural language processing techniques to identify candidate tasks for automation, based on business process descriptions. In [8], the authors propose to leverage technology from the domain of process mining [1], i.e., a specific type of data analysis tailored towards analyzing execution logs of business processes, for RPA candidate task identification. In [10], the authors collect the mouse clicks and keyboard strokes of users combined with simultaneous screen captures, which they define as an UI-Log. They use *process discovery algorithms* [1] on the UI-Log to mine an end-to-end process model, in order to help the business analyst to determine what tasks to automate. In [5], the authors adopt UI-Logs to automate the testing of RPA tool deployment.

Other work focuses more on the impact and/or application of RPA w.r.t. business processes. In [3], a case study with a business process outsourcing provider is presented. The authors divided employees, who performed similar tasks, into two groups, the comparison group was working with RPA, whereas the control group was without RPA. The comparison group showed a 21% productivity improvement. In [14], the authors reflect on the impact of RPA, alongside with machine learning and blockchain technology, on the human perspective of business processes. In particular, the impact on individual tasks is discussed, i.e., repetitive tasks are identified as most suitable for RPA as well as the impact on the organizational/coordination perspective.[4]

The main deficiency of mainstream research in the domain of RPA is the lack of focus on *how to automate* RPA itself. Furthermore, it is difficult to deduce *how to automate* on the basis of the outcomes of the currently available studies. In this research, we try to fix these deficiencies by empowering RPA with the ability of self-learning.

3 Notation

In this section, we present the basic notation used throughout the paper.

Given an arbitrary set X, $|X|$ returns the size of set X, and, we let $\mathcal{P}(X)=\{X' \mid X' \subseteq X\}$ denote its *power set*. A *sequence* is an enumerated collection of objects. We write a sequence as $\langle a, b, c, a, c \rangle$, i.e., a is the first and fourth element of the sequence, d is the second element, etc. We write X^* to denote the set of all possible sequence over an arbitrary set X.

[4] Ref. [14] summarizes a panel discussion on the extent to which the emergence of recent technologies will reduce the "‘human factor” in business process management.

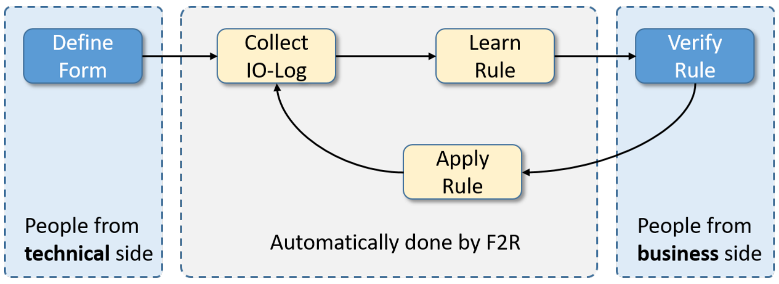

Define Form
Collect IO-Log
Learn Rule
Verify Rule
Apply Rule
People from **technical** side
Automatically done by F2R
People from **business** side

Fig. 3: The RPA lifecycle in the F2R approach: less human intervention in RPA by adopting the F2R approach.

Given arbitrary sets X_1, X_2, ..., X_n, we let $X_1 \times X_2 \times \cdots \times X_n = \{(x_1, x_2, ..., x_n) \mid x_1 \in X_1, x_2 \in X_2, ..., x_n \in X_n\}$ denote the Cartesian product over sets X_1, X_2, ..., X_n, i.e., the set of all ordered pairs consisting of elements of X_1, X_2, ..., X_n.

Let X, Y be two no-empty sets, $f \in X \to Y$ is a total function, the domain $dom(f)$ of f is X, and the range $rng(f)$ of f is defined as $rng(f) = \{f(x) \mid x \in X\} \subseteq Y$. $f \in X \nrightarrow Y$ denotes a partial function, the domain of f is a subset of X, $dom(f) \subseteq X$. $\mathbb{B} = \{\texttt{True}, \texttt{False}\}$ represents the set of Boolean values, $\perp$ is the null value, $\mathbb{N}$ is the set of natural numbers. A multiset allows its elements to occur multiple times, e.g., $[a^2, b]$ consists of 2 times element a and one time element b. We let $\mathcal{B}(X)$ represent the set of all possible multisets for X.

4 The F2R Approach to Automate RPA

In this section, we discuss the F2R approach in detail. Section 4.1 describes the life-cycle of adopting the F2R approach and provides a running-example which is used in the remainder of this paper. Section 4.2 describes how to deduce user actions and tasks on the basis of inspecting form interactions. Section 4.3 defines the notion of a *rule* and details on how to apply multiple enabled *rules*. Section 4.3 provides examples of instantiation for *rule*'s components, it makes the definition concrete.

4.1 The F2R Lifecycle

By adopting the F2R approach, we enable RPA with an algorithmic self-learning ability. Consider Figure 3, in which we present the new basic F2R life-cycle. In the life-cycle, there are only two points in which human interaction is required, i.e., *form definition* and *rule verification*. The *Form definition* step concerns identifying *forms* in the IT system, i.e., in order for F2R to be able to learn from users interacting with these forms. The *Rule verification* step concerns human validation of the rules learned by F2R, i.e., to determine whether a newly learned rule is in line with business/domain knowledge.

To ease the overall understanding of F2R and its definitions, we use a running example (the “SDAP scenario”), on the basis of the F2R life-cycle. We consider a "swivel chair" process, which concerns transferring information from one system to another. Such a process is a typical RPA adoption scenario [11]. Consider an imaginary company, SDAP Co., which has two systems for employment, i.e. an *interview system* and an *HR System*. The main business process is as follows.

1. An interviewee fills in the *Interviewee Form* (F_I) in the interview system. Form F_I has 4 fields: "Name", "Age", "Gender",and "Result".
2. The human resource (HR) department of SDAP Co. evaluates the interviewee, and updates the evaluation in Interviewee Form.
3. If the HR department's evaluation equals "Pass", an employee of the HR department subsequently fills in the *New Employee Form* (F_N) in the HR system. The F_N from has 3 fields, i.e., "Name", "Age", "Gender", which the employee copies from the F_I form. If the HR department's evaluation equals "Fail", the HR-department does not perform any action.

The conceptual description for adopting the F2R approach in the SDAP scenario is as follows:

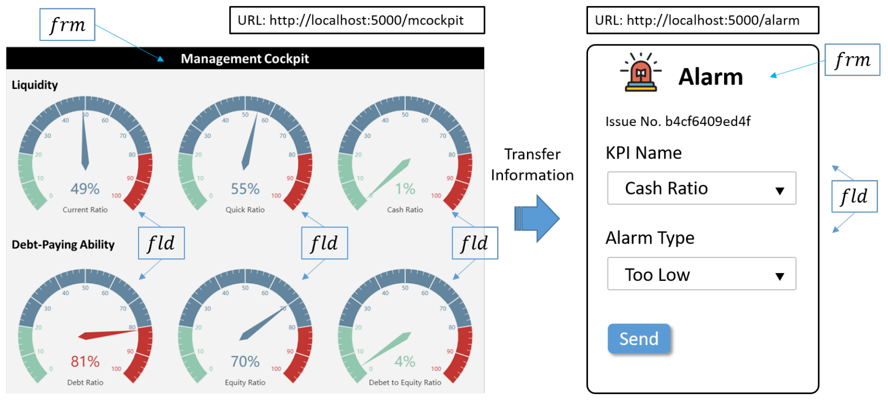
1. Define two *forms* F_I and F_N, and collect the user interactions (captured in an *Input/Output (IO)-Log*) on these two *forms*.
2. Learn *rules* from the IO-Log. The rules represent statements of the form "`if` F_I's Result field is updated to "Pass", `then` copy the content of F_I's to F_N".
3. An employee verifies the *rule*. If the rule is verified, it is adopted into the RPA solution, i.e., the rule is applied automatically.

In the following sections, we provide more detailed descriptions and formal definitions of the F2R approach. Moreover, the algorithm for learning rules, i.e. instantiating F2R, is presented as well.

4.2 Task Detection Using Forms

The cornerstone concept of the F2R approach is the notion of a *form*. A form is an abstracted and standardized representation of any IT system's data container. For example, consider a simple excel sheet, a web form, a form in SAP, etc., all these data containers can essentially be seen as a form. A *form instance* represents a filled form. A form can be filled multiple times, hence, a form can be instantiated into one or multiple *form instances*. A form instance is distinguished by its *form instance identifier*. A *form state* is the set of values of a form instance at a point of time, i.e. it contains the filled information of the form instance. A *form action* describes a manipulation of a form's state, i.e. it describes two form states, collected at the points in time in which the user is starting to edit (enter) the form instance until the user closes (leaves) the form instance. Finally, an *IO-Log* represents a set of collected form actions. The F2R approach leverages the notion of an IO-Log to find behavioral rules that are eligible for automation through RPA.

As indicated, the notion of a form is the cornerstone of the F2R approach. For example, a purchase order form in SAP, in essence, is a simple form. However,



Alarm

Issue No. b4cf6409ed4f

KPI Name: Cash Ratio

Alarm Type: Too Low

(a) A page could be seen as Form

(b) A Normal Form

Fig. 4: Two examples of a form, where *frm* points to the names of the forms and *fld* to the fields.

the concept of a form is more general. Not only the actual forms in an IT system are defined as a form, but also any UI which has a fixed structure and position in the system, is represented by a form, e.g., tables, BI dashboards, etc.. Any form has two elements, i.e., a *form name* and a set of *form fields*. The form fields refer to all the UI elements which are able to hold information.

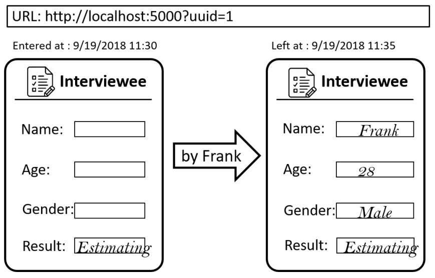
Definition 1 (Form). *Let $\mathcal{U}_{frm}$ denote the universe of* form names *and $\mathcal{U}_{fld}$ the universe of* form fields*. A form F is a pair $F = (frm, Q) \in \mathcal{U}_{frm} \times \mathcal{P}(\mathcal{U}_{fld})$, where frm is the name of the form F and $Q \subseteq \mathcal{U}_{fld}$ is a set of form fields. Let $\mathcal{U}_F$ denote the set of all possible forms in the system, and $\mathtt{ID} \in \mathcal{U}_{fld}$ be the form instance identifier, for which we enforce:*

- $\forall F = (frm, Q) \in \mathcal{U}_F \; (\mathtt{ID} \in Q)$*, i.e., every form has the form instance identifier in Q.*
- $\forall F = (frm, Q), F' = (frm', Q') \in \mathcal{U}_F \left(frm = frm' \implies Q = Q' \right)$*, i.e., the name of a* form *uniquely determines the set of fields associated with it.*

Figure 4 shows two examples of a form; Figure 4 (a) is a management dashboard, which is representable as a form, i.e., it is formalized as $F_{MC} = (MC, \{\mathtt{ID}, CR, QR, CashR, DR, ER, DER\})$. Figure 4 (b) is a more classical *form*, which is formalized as $F_A = (Alarm, \{\mathtt{ID}, KPI, AlarmType\})$.

A *form instance* is a filled form with a unique id assigned to the omnipresent field $\mathtt{ID}$. We assume that there always exists a unique form instance ID $\mathtt{id} \in \mathtt{IID}$, which allows us to identify a single instance of a form. Observe that, in some cases, an IT system readily defines an instance identifier for a form. In other cases, such an instance identifier is derived, e.g., from the URL of a web form.

Definition 2 (Form Instance Identifier, Form Instance). *Let $\mathcal{U}_{Val}$ denote the universe of possible form-field values and let $\mathtt{IID} \subseteq \mathcal{U}_{Val}$ denote the set of all possible form instance ids. A form instance $f = (F, \mathtt{id}) \in \mathcal{U}_F \times \mathtt{IID}$ of a form F*



URL: http://localhost:5000?uuid=1

Entered at : 9/19/2018 11:30

Interviewee

Name:

Age:

Gender:

Result: Estimating

Left at : 9/19/2018 11:35

Interviewee

Name: Frank

Age: 28

Gender: Male

Result: Estimating

$$\begin{aligned} F_I &= (I, \{\texttt{ID}, Name, Age, Gender, Result\}) \\ \overrightarrow{F} &= \{F_I, \\ &\quad \{(\texttt{ID}, 1), (Result, Estimating)\}, \\ &\quad \{(\texttt{ID}, 1), (Name, Frank), (Age, 28), \\ &\quad (Gender, Male), (Result, Estimating)\}, \\ &\quad 9/19/2018\ 11{:}\ 30,\ 9/19/2018\ 11{:}\ 35, \\ &\quad Frank\} \end{aligned} \tag{1}$$

Fig. 5: An example of form action, a form action captures two form states when the user enters the form instance and leaves the instance.

is uniquely identified by $\texttt{id}{\in}IID$. *Let* $\mathbf{ids}_F{\subseteq}IID$ *denote all possible id values for form* F. *We assume* $\forall F, F'{\in}\mathcal{U}_F\ (F{\neq}F' \implies \mathbf{ids}_F \cap \mathbf{ids}_{F'}{=}\emptyset)$

Since users typically do not fill out an instance of a form in one go, i.e., the user typically reopens the form and/or changes values, hence, the *state* of the form changes over time. Therefore, we define the notion of a *form state* ,i.e., representing the actual information contained in a form instance at a specific point of time.

Definition 3 (Form State). *Let* $\mathcal{U}_{Val}$ *denote the universe of possible form-field values, and let* $\mathcal{U}_T$ *denote the universe of time. Let* $F{=}(frm, Q){\in}\mathcal{U}_F$ *be a form, let* $\texttt{id}{\in}\mathbf{ids}_F$ *and let* $t{\in}\mathcal{U}_T$. *A state of the instance of a form* F*, identified by instance-id* $\texttt{id}$*, at point* t *in time, is defined by a partial function* $\phi_{(F,\texttt{id},t)}\colon Q{\nrightarrow}\mathcal{U}_{Val}$*, for which:* $\phi_{(F,\texttt{id},t)}(ID){=}\texttt{id}, \forall t{\in}\mathcal{U}_T$*, i.e., the instance-id is always present.*

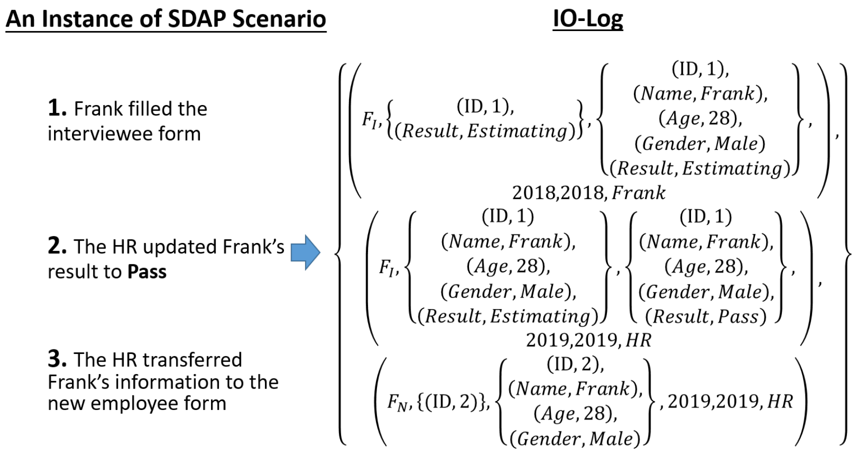
A *form action* (FA) records an interaction a user has performed on a form instance. More specifically, it records two form states at the points of time when the user is entering and leaving the form instance, respectively.

Definition 4 (Form Action, IO-Log). *Let* $\mathcal{U}_U$ *denote the universe of all possible users, let* $F = (frm, Q){\in}\mathcal{U}_F$ *be a form, let* $t_b, t_f{\in}\mathcal{U}_T$ *s.t.* $t_b{\leq}t_f$ *be two timestamps, let* $u{\in}\mathcal{U}_U$ *be a user and* $\texttt{id}{\in}IID$ *be an instance ID. Furthermore, let* $\overline{\phi}, \hat{\phi}\colon Q{\nrightarrow}\mathcal{U}_{Val}$. *A form action* $\overrightarrow{F}{=}(F, \overline{\phi}, \hat{\phi}, t_b, t_f, u)$ *describes a manipulation of an instance of form* F *by user* u*, entering in the instance at time* t_b *and leaving it at time* t_f. *Observe that* $\phi_{(F,\texttt{id},t_b)} = \overline{\phi}$ *and* $\phi_{(F,\texttt{id},t_f)} = \hat{\phi}$. *We let* $\mathcal{U}_{\overrightarrow{F}}$ *denote a set of all possible form actions. An IO-Log* $L \subseteq \mathcal{U}_{\overrightarrow{F}}$ *is a set of form actions.*

The first step in the SDAP scenario is illustrated in Figure 5, with an example of a form action. Moreover, a complete example of the SDAP scenario is illustrated using the IO-Log shown in Figure 6.

4.3 Defining and Applying Rules

As an IO-Log is a collection of FAs, it records manipulations of form values corresponding to the tasks executed by users. We use the collection of FAs to


An Instance of SDAP Scenario
IO-Log
1. Frank filled the interviewee form
2. The HR updated Frank's result to Pass
3. The HR transferred Frank's information to the new employee form

$$\left\{ \begin{array}{c} \left(F_I, \left\{ \begin{array}{c} (\mathrm{ID}, 1), \\ (Result, Estimating) \end{array} \right\}, \left\{ \begin{array}{c} (\mathrm{ID}, 1), \\ (Name, Frank), \\ (Age, 28), \\ (Gender, Male) \\ (Result, Estimating) \end{array} \right\}, \\ 2018, 2018, Frank \right), \\ \left(F_I, \left\{ \begin{array}{c} (\mathrm{ID}, 1) \\ (Name, Frank), \\ (Age, 28), \\ (Gender, Male), \\ (Result, Estimating) \end{array} \right\}, \left\{ \begin{array}{c} (\mathrm{ID}, 1) \\ (Name, Frank), \\ (Age, 28), \\ (Gender, Male), \\ (Result, Pass) \end{array} \right\}, \\ 2019, 2019, HR \right), \\ \left(F_N, \{(\mathrm{ID}, 2)\}, \left\{ \begin{array}{c} (\mathrm{ID}, 2), \\ (Name, Frank), \\ (Age, 28), \\ (Gender, Male) \end{array} \right\}, 2019, 2019, HR \right) \end{array} \right\}$$

Fig. 6: An example of IO-Log, it is a set of FAs. This example describes that an interviewee Frank filled his form. HR updated Frank's result as "Pass", and copied Frank's information to an F_N instance.

deduce their relations, i.e., we identify whether certain input(s)/modification(s) in a form are likely to lead to other input(s)/modifications(s). We define such relations by using *rules*. For the notion of a rule we adopt a widely accepted RPA definition suggesting to describe RPA actions using "`if ... then ...`" statements [17]. When the `if` condition of a rule holds for an observed FA, our RPA solution applies the `then` part of the rule and generates a new FA as a response. Finally, the RPA solution translates this new FA into a sequence of low-level movements (e.g., mouse clicks or keyboard strokes) on the corresponding form instance.

A *rule* consists of two parts, a condition and a response. A *condition* checks whether an FA satisfies some requirements, i.e., it defines a function that maps an FA to true or false. For example, in the SDAP scenario, we have a condition that requires the Result field (of the Interview form) to be changed into a "Pass". The second FA shown in Figure 6 satisfies this condition. Based on the condition, a *response* then generates a new FA, e.g., such an FA copies the current values in the "Name", "Age", and "Gender" fields to a form instance of the new employee form. In the following, we formally define the conditions, responses, and rules.

Definition 5 (Condition). *A condition* $\mathbf{c}$ *is a function* $\mathbf{c} \colon \mathcal{U}_{\vec{F}} \to \mathbb{B}$*. Let* $\mathcal{U}_{Cond}$ *be the set of all possible* conditions*.*

Definition 6 (Response). *A response* $\mathbf{r}$ *is a function* $\mathbf{r} \colon \mathcal{U}_{\vec{F}} \to \mathcal{U}_{\vec{F}}$*, which generates a new FA based on the observed FA. Let* $\mathcal{U}_{Resp}$ *be the set of all possible* responses*.*

Definition 7 (Rule). *Given a condition* $\mathbf{c} \in \mathcal{U}_{Cond}$ *and response* $\mathbf{r} \in \mathcal{U}_{Resp}$*, let* $\mathcal{U}_R$ *be the set of all possible* rules*, a rule* $\mathbf{R} \in \mathcal{U}_R$ *is characterized as:*

$$\begin{aligned} \mathbf{R}\colon \mathcal{U}_{\overrightarrow{F}} &\nrightarrow \mathcal{U}_{\overrightarrow{F}} \\ \overrightarrow{F} &\mapsto \begin{cases} \mathbf{r}(\overrightarrow{F}) & \textit{if } \mathbf{c}(\overrightarrow{F}) \\ \perp & \textit{otherwise} \end{cases} \end{aligned}$$

Essentially, the RPA solution of our F2R approach learns a set of *rules*, which we refer to as a *rule base.* Moreover, we define a mechanism for selecting a set of suitable rules for one observed FA, this mechanism is called the *rule selector.* It is possible that a single, observed FA satisfies multiple conditions and, therefore, potentially triggers multiple rules present in a rule base. For example, consider the first FA of the SDAP scenario in Figure 6, which satisfies the following two conditions: `if` the Age field is filled in, or `if` the Gender field is filled in. Therefore, we define a *rule selector*, which is a mechanism to select and apply a rule.

Definition 8 (Rule Selector). *A* rule selector $\zeta\colon \mathcal{P}(\mathcal{U}_R) \times \mathcal{U}_{\overrightarrow{F}} \nrightarrow \mathcal{U}_{\overrightarrow{F}}$*, selects and applies a rule present in a given rule base on an observed form action.*

Instantiating Conditions Using First-Order Logic In order to make the concept of conditions more tangible, we provide a detailed example of how the conditions can be formally instantiated using First-Order Logic Conditions (FOLC). A FOLC is a set of propositions linked by logic connectives. A FOLC evaluates the values of the proposition variables and returns `True` or `False`. In our case, the propositions are defined on the observed FA. Each proposition describes the range of values that is acceptable for an observed FA's field value.

Definition 9 (First-Order Logic Condition). *Let* $F = (frm, Q) \in \mathcal{U}_F$*,* $\overrightarrow{F} = (F, \overline{\phi}, \hat{\phi}, t_b, t_f, u)$ *be an FA on an instance of form F; let* $\overline{\gamma}_1, ..., \overline{\gamma}_n, \hat{\gamma}_1, ...,$ $\hat{\gamma}_m$ *be functions of the form* $Q \nrightarrow \mathcal{P}(\mathcal{U}_{Val}) \backslash \{\emptyset\}$*. And, let*

- $\overline{\mathbf{c}}_i \equiv \forall q \in dom(\overline{\gamma_i}) \left(q \in dom(\overline{\phi}) \wedge \overline{\phi}(q) \in \overline{\gamma_i}(q)\right)$*, for* $1 \leq i \leq n$*;*
- $\hat{\mathbf{c}}_i \equiv \forall q \in dom(\hat{\gamma}_i) \left(q \in dom(\hat{\phi}) \wedge \hat{\phi}(q) \in \hat{\gamma}_i(q)\right)$*, for* $1 \leq i \leq m$*.*

An FOLC function is formed by a Boolean expression over $\overline{\mathbf{c}}_1, ..., \overline{\mathbf{c}}_n, \hat{\mathbf{c}}_1, ..., \hat{\mathbf{c}}_m$*.*

For example, let us consider the third step in the SDAP scenario, where an FA, e.g., copying the field values, is performed, if the result field is changed from an "Estimating" to a "Pass" value. A corresponding FOLC $\mathbf{c}'_\gamma$ for this `if` condition is defined as follows. Let $F_I = (I, \{\mathtt{ID}, Name, Age, Gender, Result\})$ be the interviewee form, and $\overrightarrow{F}' = (F_I, \overline{\phi}, \hat{\phi}, t_b, t_f, u) \in \mathcal{U}_{\overrightarrow{F}}$ be an arbitrary FA on a F_I's instance. Let $\overline{\gamma} = \{(Result, \{Estimating\})\}$ denote the range of the pairs of the fields and their values that are acceptable for the form-state $\overline{\phi}$, and $\hat{\gamma} = \{(Result, \{Pass\})\}$ the range of the pairs that are acceptable for $\hat{\phi}$. Using FOLC, we can define the `if` condition $\mathbf{c}'_\gamma$ as follows: $\mathbf{c}'_\gamma(\overrightarrow{F}') = (\overline{\phi}(Result) \in \{Estimating\}) \wedge (\hat{\phi}(Result) \in \{Pass\})$.

Instantiating Responses Using Transform Functions To exemplify the concept of responses, we discuss a detailed type of response, namely the *transform* response. A transform response transforms, e.g., copies, the field values of an observed FA into the corresponding fields of a newly created form instance. Taking again the third step in the SDAP scenario as an example, the transform response in this case copies the "Name", "Age", and "Gender" values into a new form instance of the new employee form.

Definition 10 (Transform Response). *Let $F = (frm, Q), F' = (frm', Q') \in \mathcal{U}_F$ be two forms, let $\lambda \colon Q \nrightarrow Q'$ be a field mapping, and, given $q \in Q$, let $\delta_q \colon \mathcal{U}_{Val} \to \mathcal{U}_{Val}$ be the transform function. Furthermore, let $\overrightarrow{F} = (F, \overline{\phi}, \hat{\phi}, t_b, t_f, u) \in \mathcal{U}_{\overrightarrow{F}}$ be an FA on an instance of form F. Let $\hat{\phi}_{F'} = \{(q', v) \in Q' \times \mathcal{U}_{Val} \mid \exists q \in dom(\lambda) \colon \lambda(q) = q' \wedge \delta_q(\hat{\phi}(q)) = v\}$. Let $t_r \in \mathcal{U}_T, t_r > t_f$, $\mathtt{id}' \in \mathtt{IID}$, the transform response function $\mathbf{r}_{tf}$ is characterized as:*

$$\mathbf{r}_{tf} \colon \mathcal{U}_{\overrightarrow{F}} \to \mathcal{U}_{\overrightarrow{F}}$$
$$(F, \overline{\phi}, \hat{\phi}, t_b, t_f, u) \mapsto (F', \phi(F', \mathtt{id}', t_f), \hat{\phi}_{F'}, t_f, t_r, \mathcal{R}).$$

In essence, the transform response creates a new form instance (F', id') of the form F'. Moreover, it creates two states: state $\phi(F', \mathtt{id}', t_f)$ is the default form state of the new form instance, and state $\hat{\phi}_{F'}$ is the state where its field values are transformed. These two states are then embedded in the returned FA. Besides, the entering time of this FA is t_f, which means it will be executed right after observing the previous FA. The leaving time would be t_f. Hence, the period between t_f and t_r allows the RPA solution to execute this new FA and the system latency. We use $\mathcal{R}$ to denote this new FA is done by the RPA solution.

A special case of the transform response is the *copy* response. If the transform function $\delta_q \colon \mathcal{U}_{Val} \to \mathcal{U}_{Val}$ returns the input itself for all fields $q \in Q$, it then basically copies the input values, which we called a *copy* response.

Considering the step 3 of the SDAP scenario, where the information of name, age, and gender is copied from the Interview form into the New Employee form, this can now be performed by applying a copy response $\mathbf{r}'_{\mathtt{tf}}$. Let $F_N = (N, \{\mathtt{ID}, Name', Age', Gender', Result'\})$ be the New Employee form. Let $t_r \in \mathcal{U}_T$ and $t_r > t_f$ be the two timestamps of the new FA. Let $\mathtt{id}' \in \mathtt{IID}$ be the id of the new form instance. Let $\lambda = \{(Name, Name'), (Age, Age'), (Gender, Gender')\}$ be the mapping. The response $\mathbf{r}'_{\mathtt{tf}}(\overrightarrow{F}') = (F_N, \phi(F', \mathtt{id}', t_f), \{(q', v) \mid (q, q') \in \lambda \wedge \hat{\phi}(q) = v\}, t_f, t_r, \mathcal{R})$ is a copy response. A concrete example is shown by the second and the third FAs in Figure 6, respectively as the input and the output of this copy response.

Simple Rules We have explained that a rule consists of a condition and a response. We have shown a concrete instantiation of the conditions using FOLC and a concrete type of response defined as the transform response. Based on the SDAP scenario shown in Figure 6, we can define a *rule* $\mathbf{R}'_{\mathbf{c.r}}$ which given an input

FA $\overrightarrow{F}'$, it returns the new FA $\mathbf{r}'_{\mathtt{tf}}(\overrightarrow{F}')$ (i.e., the result of the copy response) if the condition $\mathbf{c}'_{\gamma}(\overrightarrow{F}')$ on the FA $\overrightarrow{F}'$ holds, i.e., $\mathbf{R}'_{\mathbf{c}.\mathbf{r}} = \mathbf{r}'_{\mathtt{tf}}(\overrightarrow{F}')$ if $\mathbf{c}'_{\gamma}(\overrightarrow{F}')$. We call this type of rules, which comprise a FOLC as its condition and a copy response, the *simple rules*.

5 Learning Simple Rules from IO-Logs

As discussed above, an RPA solution operates based on a rule base (i.e., a set of rules). Training such an RPA solution is basically learning a rule base from the IO-log and building up such a rule base. In this section, we discuss an algorithm to learn the simple rules. We call it the *simple rule learner (SRL)*.

The SRL algorithm consists of two phases, first learning the conditions and then learning the responses. Next, we explain the learning of conditions and responses in Section 5.1 and 5.2, respectively. In Section 5.3, we discuss how to use the obtained responses and conditions to construct the rule base and the rule selector for the RPA solution. In Section 5.4, we briefly explain the implementation of the RPA solution.

5.1 The Condition Learner in the SRL Algorithm

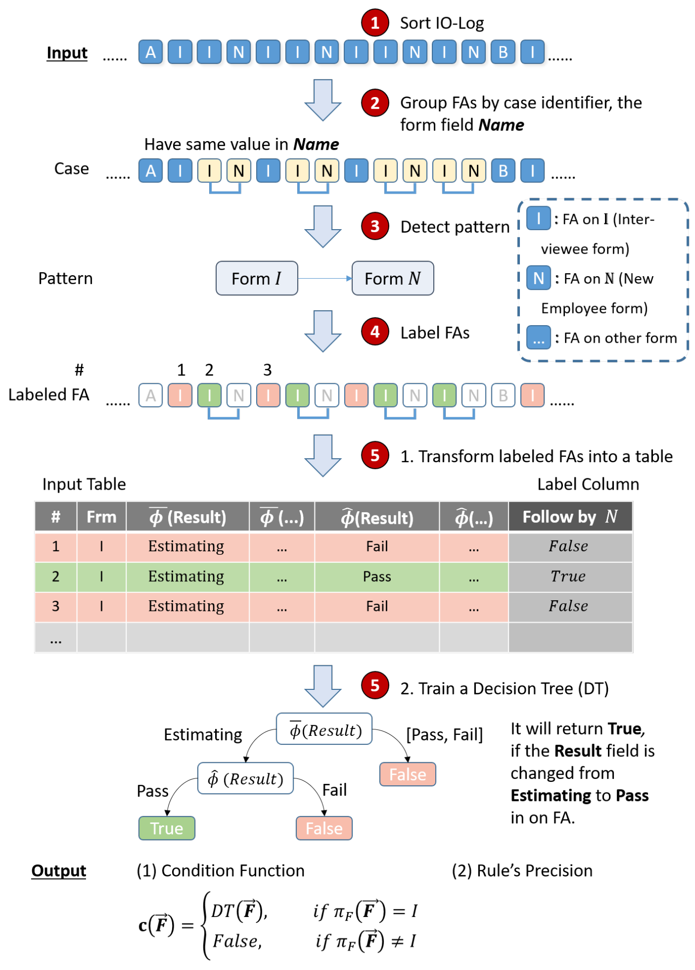
As discussed, based on the input IO-log, we propose the SRL algorithm to learn a set of rules in order to obtain our rule base. As defined in Def. 5, a condition of a rule checks whether an FA meets some properties. For learning such properties, we provide an algorithm which consists of 5 steps. An overview of these steps together with an example is shown in Figure 7.

In step (1), we simply sort the input IO-Log $L \in \mathcal{P}(\mathcal{U}_{\overrightarrow{F}})$ to obtain a sequence $S \in L^*$ of FAs which is ordered by the timestamps t_b of the FAs. As shown by the output of step (1) in Figure 7, where each rounded rectangle represents an FA, the sorted IO-log comprises $\langle \ldots \overrightarrow{F}_A, \overrightarrow{F}_I, \overrightarrow{F}_I, \overrightarrow{F}_N, \overrightarrow{F}_I, \overrightarrow{F}_I, \overrightarrow{F}_N, \ldots \rangle$.

In step (2), we find *cases* from the sorted IO-log S. A *case* $cs \in \mathcal{U}_{\overrightarrow{F}} \times \mathcal{U}_{\overrightarrow{F}}$ is a pair of FAs. In this paper, we assume there is a form field which captures the case identifier, i.e., the value in this field is the case ID. Every two FAs, which have the same case ID, are paired as a case. We define a function FC to find the cases, i.e., $FC \colon L^* \times \mathcal{U}_{\mathtt{fld}} \nrightarrow \mathcal{P}(\mathcal{U}_{\overrightarrow{F}} \times \mathcal{U}_{\overrightarrow{F}})$.

For instance, in the SDAP scenario, let $F_I = (I, \{\mathtt{ID}, Name, Age, Gender, Result\})$, $F_N = (N, \{\mathtt{ID}, Name, Age, Gender\})$ be the two forms, the *Name* field in F_I and F_N could be the case identifier. Based on the values in the *Name* field, the second and the third FAs in Figure 6 are grouped into one case, because they have the same name. In the output of step (2) in Figure 7, the cases are highlighted in the original IO-log. For example, the third FA $\overrightarrow{F}_I$ is paired with the fourth FA $\overrightarrow{F}_N$.

In step (3), we use the set of cases to detect *patterns*. A *pattern* $p \in \mathcal{U}_F \times \mathcal{U}_F$ is a pair of forms. A pattern indicates that the FAs happened on the pattern's forms are always paired as cases, and the order of these two FAs in a case is


1 Sort IO-Log
Input ...... A I I N I I N I I N I N B I ......
2 Group FAs by case identifier, the form field Name
Have same value in Name
Case ...... A I I N I I N I I N I N B I ......
3 Detect pattern
I : FA on I (Interviewee form)
N : FA on N (New Employee form)
... : FA on other form
Pattern
Form I
Form N
4 Label FAs
# 1 2 3
Labeled FA ...... A I I N I I N I I N I N B I ......
5 1. Transform labeled FAs into a table
Input Table
Label Column
5 2. Train a Decision Tree (DT)
Estimating
[Pass, Fail]
Pass
Fail
True
False
False
It will return True, if the Result field is changed from Estimating to Pass in on FA.
Output
(1) Condition Function
(2) Rule's Precision

#	Frm	$\overline{\phi}$ (Result)	$\overline{\phi}$ (...)	$\hat{\phi}$(Result)	$\hat{\phi}$(...)	Follow by N
1	I	Estimating	...	Fail	...	*False*
2	I	Estimating	...	Pass	...	*True*
3	I	Estimating	...	Fail	...	*False*
...						

$\overline{\phi}(Result)$

$\hat{\phi}\,(Result)$

$$\mathbf{c}(\vec{\boldsymbol{F}}) = \begin{cases} DT(\vec{\boldsymbol{F}}), & if\ \pi_F(\vec{\boldsymbol{F}}) = I \\ False, & if\ \pi_F(\vec{\boldsymbol{F}}) \neq I \end{cases}$$

Fig. 7: Adopting learning algorithm of *condition* functions at the SDAP scenario.

corresponding with the pattern. For example, in the SDAP scenario, the pattern could be $p = (F_I, F_N)$, it means after an arbitrary FA $\vec{F}$ happened on F_I, it is often observed that another FA, which has the same case ID with $\vec{F}$, happened on F_N. The main points of this step are (i) extracting candidates of patterns from cases; (ii) calculating the confidence for each candidate, if the confidence passes a threshold, the candidate will be selected as a pattern.

For the formal description, we define a function $mop\colon \mathcal{P}(\mathcal{U}_{\vec{F}} \times \mathcal{U}_{\vec{F}}) \to \mathcal{B}(\mathcal{P}(\mathcal{U}_F \times \mathcal{U}_F))$ for transforming a set of cases to a multiset of the pairs of forms, and a function $mof\colon \mathcal{P}(\mathcal{U}_{\vec{F}}) \to \mathcal{B}(\mathcal{P}(\mathcal{U}_F))$ for returning a multiset of forms by the same giving IO-Log, which is adopted to generate the set of cases. Let $q \in \mathcal{U}_{\mathtt{fld}}$ be the case identifier, $X = mop(FC(S, q))$ be a multiset of the pattern candidates, if $pc = (F, F') \in X$ is a pattern candidates, s.t. $F \in \mathcal{U}_F, F' \in \mathcal{U}_F$, $F \neq F'$. Let $Y = mof(L)$ be a multiset of forms; and let M be a multiset. The function

$Count_M \colon M \to \mathbb{N}^+$ returns the counting number for an element in multiset M. The function $conf_{PT} \colon \mathcal{U}_F \times \mathcal{U}_F \to [0,1]$ computes the confidence of a giving pattern candidate, its formula is Equation 2.

$$\begin{aligned} pc &= (F, F') \\ conf_{PT}(pc) &= \frac{Count_X(pc)}{Count_Y(F)} \end{aligned} \tag{2}$$

If the pattern candidate's confidence is greater than a certain threshold, it will be a pattern, and this threshold is set by the user.

In step (4), we label FAs by a certain pattern. A labeled FA $lfa \in \mathcal{U}_{\overrightarrow{F}} \times \mathbb{B}$ is a pair of an FA and a Boolean value. Firstly, we label cases by a pattern. If a case's the first FA and the second FA happened on the pattern's corresponding forms, the first FA will be signed the `True` label, otherwise, the `False` label, and the second FA is ignored in the next step. Secondly, we label all other FAs that do not belong to a case, by the same pattern. If the FA occurred on the pattern's first form, it is labeled as `False`, otherwise, it will be ignored in the next step and the steps in the response learner. Meanwhile, the first element of a case is labeled as `True` by a pattern, we call this case *is aligned* with this pattern.

As shown by the output of step (4) in Figure 7, the rounded rectangle with the green background is labeled as `True` by the pattern (F_I, F_N), the one with rose color is labeled as `False`, and the one with white is ignored.

In step (5), we train a condition function from a sequence of *Labeled FAs*. This is a typical supervised learning scenario, we adopt the Decision Tree (DT) [9] algorithm in this research. There are two points in this step, (i) we transform the sequence of labeled FAs to the *input table* of DT, the attribute columns are the form fields in the "enter" and "leave" form states of FA, the label column is the Boolean label which is generated in step 4. (ii) We train a DT model based on the input table. The condition function will apply the trained DT model to giving the Boolean result if the input FA happened on the pattern's the first form; otherwise, it will return `False`. In this step, we not only gain the DT model, but also the precision of the model. And the DT model's precision is the rule's precision.

As illustrated in Figure 7, each row of the input table corresponds a labeled FA in step (4). Based on this input table, we train a DT model. In the SDAP scenario, the DT model should able to represent the logic of "it will return `True` if the Result field is changed from Estimating to Pass".

5.2 The Response Learner in the SRL Algorithm

Using the SDAP scenario, we explained the *copy* response to illustrate one type of the response function. A copy response means that the function δ in Def.11 is an *identity function* (i.e., the input equals to its output). The main issue of this part is gaining the form field mapping λ, which is defined in Def.11. There are two steps in this part, and the learning process of the SDAP scenario's response function could be illustrated as Figure 8.

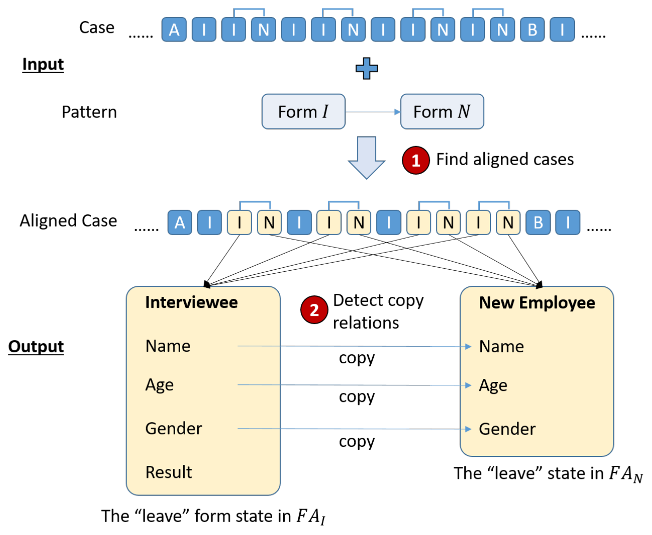


Fig. 8: Adopting learning algorithm of *response* functions at the SDAP scenario.

In step (1), given all the cases found by the condition learner, we retain the set of cases that is aligned with a given pattern. Let $p = (F, F')$ be a pattern, and $ac = (\overrightarrow{F}, \overrightarrow{F}')$ a case. Case ac is aligned with p if and only if $\pi_F(\overrightarrow{F}) = F \wedge \pi_F(\overrightarrow{F}') = F'$. The output is, thus, a set of cases $AC \subseteq \mathcal{U}_{\overrightarrow{F}} \times \mathcal{U}_{\overrightarrow{F}}$ that are aligned with p (i.e., $\forall ac \in AC : ac$ is *aligned* with p). For instance, as shown in Figure 8, the cases that are aligned with the given pattern $p = (F_I, F_N)$ are highlighted after the step (1).

In step (2), we detect the *copy relation* between the form fields of a set of aligned cases. A *copy relation* $cr \in \mathcal{U}_{\texttt{fld}} \times \mathcal{U}_{\texttt{fld}}$ is a pair of form fields, where the values of these two fields are the same, i.e., is copied, and the two fields are not in the same form. This step is similar with the step (3) in the condition learner, which is for detecting the pattern. The two main substeps of this step are (i) extracting candidates for the copy relation from the set of aligned cases and (ii) computing a confidence for each candidate. If the confidence passes a threshold, the candidate is said to be a copy relation. A set of copy relations together with the identity function build a copy response.

For substep (i), a candidate is extracted from an aligned case, the first element of the candidate is a field in the aligned case's the first FA's "leave" state; similarly, the second element of this candidate is a field in the aligned case's the second FA's "leave" state. Moreover, these two fields are holding the same value. We define a function $fcr : \mathcal{P}(\mathcal{U}_{\overrightarrow{F}} \times \mathcal{U}_{\overrightarrow{F}}) \nrightarrow \mathcal{B}(\mathcal{P}(\mathcal{U}_{\texttt{fld}} \times \mathcal{U}_{\texttt{fld}}))$ which detects the candidates. Let $\overrightarrow{F} = (F, \overline{\phi}, \hat{\phi}, t_b, t_f, u) \in \mathcal{U}_{\overrightarrow{F}}$, $\overrightarrow{F}' = (F', \overline{\phi}', \hat{\phi}', t'_b, t'_f, u) \in \mathcal{U}_{\overrightarrow{F}}$ be two arbitrary FAs on F and F'. The multiset of candidates are generated as $fcr(AC) = \left[(q, q') \mid (\overrightarrow{F}, \overrightarrow{F}') \in AC \wedge (q, q') \in dom(\hat{\phi}) \times dom(\hat{\phi}') \wedge \hat{\phi}(q) = \hat{\phi}'(q')\right]$. For instance, an output of the SDAP scenario is [(Name,Age)^2, (Name,Name)^{97}, (Name,Gender)^1].

For the substep (ii), we compute the confidence for each candidate to determine if it is a copy relation, by using the relative frequency of the candidate in the obtained multiset. For a formal description, let $MC = fcr(AC)$ be a multiset of the copy relation candidates; $cc = (q, q') \in MC$ be a candidate; $conf_{CR} : \mathcal{U}_{\texttt{fld}} \times \mathcal{U}_{\texttt{fld}} \rightarrow [0, 1]$ be a total function for calculating the confidence, its formula is in Equation 3.

$$\begin{aligned} &cc = (q, q') \in MC \\ &conf_{CR}(cc) = \frac{Count_{MC}(cc)}{|MC|} \end{aligned} \tag{3}$$

If a candidate's confidence is above a certain threshold, the candidate is selected as a copy relation. Meanwhile, this threshold is set by the user. For example, given the multiset returned by the previous step, $conf_{CR}$(Name,Age)=0.02, $conf_{CR}$(Name,Name)=0.97, $conf_{CR}$(Name,Gender)=0.01. If the user set the confidence to 0.9. The (Name,Name) is selected as a copy relation.

As illustrated in Figure 8, the step (2) detects the copy relations from the set of aligned cases. The values, which are holding by the fields "Name", "Age", "Gender" in the "leave" state on F_I's FAs, are always equivalent to the values, which are in the fields "Name", "Age", "Gender" of the "leave" state in F_N's FAs. Thus, these threes copy relations are detected. Furthermore, the set of these threes copy relations is the field mapping λ, the identity function is the value transfer function δ, they build the copy response function for the SDAP scenario.

5.3 The Rule Base and The Rule Selector in The SRL Algorithm

In the rule selector, the rules are picked based on a certain criterion, in the SRL algorithm, we use the precision as the criterion. In principle, each rule classifies an observed FA `True` or `False`; since we are in the training phase, we know the true positive and false positive, which we use to calculate the precision measure [9]; as a result, if an observed FA satisfies multiple rules' condition, we will select the rule with the highest precision measures from the SRL rule base.

5.4 Implementation

The implementation is a proof of concept for the F2R approach and SRL algorithm, which supports the RPA solution to interact with web-pages. We developed a Chrome-extension (running in Chrome's back-end) that collects IO-Log and a web-service that receives IO-Log from the Chrome-extension, holds the rule base, and runs the SRL and the rule selector. We adopted Selenium (Web-page automation software) to translate each FA into a sequence of low-level movements on UI. The code together with screencasts for the F2R approach is publicly available at GitHub (`https://github.com/FrankBGao/F2R_approach_RPA`).

6 Conclusion and Future Work

In conclusion, this research provides the skeleton for our future studies in the area of RPA. We discussed the limitations in the current RPA techniques. For solving these issues, we provide the F2R approach, it formalizes the critical notions such as form, IO-Log, and rule. These notions allow the F2R approach to automatically identify tasks, deduce rules, and apply rules. We provided algorithms to learn FOLC as the conditions of rules and copy responses. It makes the concepts of the F2R approach much more concrete.

For future work, we consider dropping certain assumptions that are made in this paper. Firstly, in the definition of FA, an FA consists of two form states which are collected at the points in time at which the user enters and leaves the form instance. In the future, we will define the running FAs, which collects more form states at different timestamps.

Secondly, in the definition of Transform Response, we define $\lambda\colon Q \nrightarrow Q'$, $\delta_q\colon \mathcal{U}_{Val} \to \mathcal{U}_{Val}$, which are two "one to one" mappings. We plan to define other types of λ and δ, i.e., "one to many" or "many to one" mappings.

Thirdly, in learning simple rules, we assume there is a field that is the case identifier. However, this assumption does not always hold. Thus, detecting the case without the identifier is a research topic for the future.

Finally, in order to perform a response, we use the rule to check the condition and return an FA. For future work, we aim to develop a language to specify the rule's response. This language would extend the current definition of rule such that it can describe more complex tasks. The RPA solution can then be trained to execute this language to deliver more sophisticated FAs.

References

1. van der Aalst, W.M.P.: Process Mining - Data Science in Action, Second Edition. Springer (2016). https://doi.org/10.1007/978-3-662-49851-4
2. van der Aalst, W.M.P., Bichler, M., Heinzl, A.: Robotic Process Automation (2018)
3. Aguirre, S., Rodriguez, A.: Automation of a Business Process Using Robotic Process Automation (RPA): A Case Study. In: WEA. pp. 65–71. Springer (2017)
4. Asatiani, A., Penttinen, E.: Turning Robotic Process Automation into Commercial Success–Case Opuscapita. Journal of Information Technology Teaching Cases **6**(2), 67–74 (2016)
5. Chacón-Montero, J., Jiménez-Ramírez, A., Enríquez, J.: Towards a Method for Automated Testing in Robotic Process Automation Projects. In: Proceedings of the 14th International Workshop on AST. pp. 42–47. IEEE Press (2019)
6. Cichocki, A., Ansari, H.A., Rusinkiewicz, M., Woelk, D.: Workflow and Process Automation: Concepts and Technology, vol. 432. Springer Science & Business Media (2012)
7. Dumas, M., La Rosa, M., Mendling, J., Reijers, H.A.: Fundamentals of Business Process Management, Second Edition. Springer (2018)
8. Geyer-Klingeberg, J., Nakladal, J., Baldauf, F., Veit, F.: Process Mining and Robotic Process Automation: A Perfect Match. In: BPM (Dissertation/Demos/Industry). pp. 124–131 (2018)

9. Han, J., Pei, J., Kamber, M.: Data Mining: Concepts and Techniques. Elsevier (2011)
10. Jimenez-Ramirez, A., Reijers, H.A., Barba, I., Del Valle, C.: A Method to Improve the Early Stages of the Robotic Process Automation Lifecycle. In: CAiSE. pp. 446–461. Springer (2019)
11. Lacity, M., Willcocks, L.: Robotic Process Automation: The Next Transformation Lever for Shared Services. London School of Economics Outsourcing Unit Working Papers **7**, 1–35 (2015)
12. Leopold, H., van der Aa, H., Reijers, H.A.: Identifying Candidate Tasks for Robotic Process Automation in Textual Process Descriptions. In: Enterprise, Business-Process and Information Systems Modeling. pp. 67–81. Springer (2018)
13. Linn, C., Zimmermann, P., Werth, D.: Desktop Activity Mining - A New Level of Detail in Mining Business Processes. In: Workshops of the INFORMATIK 2018-Architekturen, Prozesse, Sicherheit und Nachhaltigkeit (2018)
14. Mendling, J., Decker, G., Richard, H., Hajo, A., Ingo, W., et al.: How do Machine Learning, Robotic Process Automation, and Blockchains Affect the Human Factor in Business Process Management? Communications of the Association for Information Systems **43**(Art. 19), 297–320 (2018)
15. Scheer, A.W., Abolhassan, F., Jost, W., Kirchmer, M.: Business Process Automation. Springer (2004)
16. Ter Hofstede, A.H., Van der Aalst, W.M., Adams, M., Russell, N.: Modern Business Process Automation: YAWL and Its Support Environment. Springer Science & Business Media (2009)
17. Tornbohm, C.: Market Guide for Robotic Process Automation Software. Gartner.com (2017)
18. Tripathi, A.M.: Learning Robotic Process Automation: Create Software Robots and Automate Business Processes with the Leading RPA Tool–Uipath. Packt Publishing Ltd (2018)
19. Willcocks, L.P., Lacity, M., Craig, A.: The IT Function and Robotic Process Automation (2015)