Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

Characterizing Malware Samples in the SOREL-20M dataset by Concept Learning

Bachelor Thesis

2023
Tomáš Bisták

Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

Characterizing Malware Samples in the SOREL-20M dataset by Concept Learning

Bachelor Thesis

Study Programme:	Applied Computer Science
Field of Study:	Computer Science
Department:	Department of Applied Informatics
Supervisor:	doc. RNDr. Martin Homola, PhD.

Bratislava, 2023
Tomáš Bisták

Acknowledgments: Tu môžete poďakovať školiteľovi, prípadne ďalším osobám, ktoré vám s prácou nejako pomohli, poradili, poskytli dáta a podobne.

Abstrakt

Slovenský abstrakt v rozsahu 100-500 slov, jeden odstavec. Abstrakt stručne sumarizuje výsledky práce. Mal by byť pochopiteľný pre bežného informatika. Nemal by teda využívať skratky, termíny alebo označenie zavedené v práci, okrem tých, ktoré sú všeobecne známe.

Kľúčové slová: jedno, druhé, tretie (prípadne štvrté, piate)

Abstract

Abstract in the English language (translation of the abstract in the Slovak language).

Keywords:

Contents

List of Figures

List of Tables

1 Introduction

The immense technological advances we had a chance to witness in the past few decades has gradually enabled us to perform many of the previously implausible computationally-intensive tasks in near real time. As a consequence, various artificial-intelligence (AI) methods have been successfully applied in a number of areas where data is abundant.

Although the cyber-security industry can certainly be considered a field belonging into this category, the research regarding malware detection is still conducted predominantly by the commercial sector and the details are often highly confidential. This is due in part to the lack of sufficiently-large publicly-available data sets. Fortunately, in the recent years, a few such sources of information have emerged, e.g. the EMBER [1] and SOREL-20M [14] data sets.

While machine-learning (ML) techniques have long been used in the proprietary tools to protect a user's system against misbehaving software [3], there have been only few attempts to employ symbolic AI to malware detection (for instance, ...). Hence, our main objective is to empirically evaluate the performance of one novel symbolic approach to inductive learning, i.e. learning from examples, called concept learning.

As its name suggests, a concept-learning algorithm in general endeavours to find the most precise descriptions of the given set of positive examples based on the provided knowledge. We will use these algorithms to obtain possible characterisations of malware deduced from the information acquired via static analysis of both malicious and benign files. These data will be extracted from EMBER and SOREL-20M. Since the outcome of a concept-learning process is a human-readable logical formula, we expect to achieve better explainability of malware classification in comparison with the standard statistical methods.

In addition, we propose several changes to the algorithms selected for evaluation. Despite having been tested only in this specific scenario, we believe that the applying our modifications can lead to relevant improvements in other situations as well. We will also justify their relevance formally.

In the remainder of this document, we first discuss ...

2 Description Logics and OWL

Description Logics (DLs) were introduced in the late 1980s as a new family of logical formalisms specifically designed for structured knowledge representation in an effort to automate reasoning over vast amounts of knowledge [4].

Inspired by human conceptual thinking, description logics (DLs) provide us with instruments to classify entities, identify binary relationships between them, and create hierarchies over the classes and associations. Using these tools, we can model the knowledge of an application domain. We refer to such representations as *knowledge bases* in the context of DLs.

Although most of the DLs are essentially fragments of first-order logic (FOL), the theory behind them was carefully developed so that reasoning in DLs, unlike in FOL, remains decidable [20], which is, of course, crucial for automation.

The whole DL family comprises a number of formalisms, differing mainly in the level of expressivity stemming from the richness of syntax. Here, we present the $\mathcal{SROIQ}$ DL and OWL 2 DL.

2.1 $\mathcal{SROIQ}$ DL

The $\mathcal{SROIQ}$ DL [15] is among the most expressive DLs. The next two sections deal with its syntax and semantics, respectively. The definitions we provide in these sections are based on Rudoplh's [20] and Baader et al.'s [4] treatises.

2.1.1 Syntax

As in any logical formalism, we first need to specify a set of distinguishable symbols (*vocabulary*) in order to define a language in which meaningful sentences (*statements*) can be formed, possibly with use of connectors and modifiers. In DLs, we have *concept names*, *role names*, and *individual names* that together determine a language's vocabulary.

Definition 1 (Vocabulary)**.** *The* ***vocabulary*** *of a $\mathcal{SROIQ}$ language $\mathcal{L}$ is a triple $\Sigma = (N_R, N_C, N_I)$, where N_R is a set of* ***role names****, N_C a set of* ***concept names****,*

*and N_I a set of **individual names**. The sets N_R, N_C, N_I are mutually disjoint. None of the symbols in $\{\top, \bot, \neg, \sqcap, \sqcup, \exists, \forall, \text{Self}, \geq, \leq, \circ\}$ and punctuation symbols appears in N_R, N_C, or N_I.*

Intuitively, we use concept names to denote categories, such as `Person`, `Animal`, or `Laptop`, role names as identifiers of binary relationships, e.g., `worksAt`, `livesIn`, or `manufacturedBy`, and individual names to refer to particular entities, for example, `john`, `cat_alice`, or `bobs_newest_ultrabook`.

Given the vocabulary of a $\mathcal{SROIQ}$ language, we can clearly define what constructs are considered valid in that language. However, contrary to monolithic theories in FOL, knowledge bases in $\mathcal{SROIQ}$ are split into three parts: assertional (ABox), terminological (TBox), and relational (RBox). Since each of them has a different purpose, we need to treat the syntax of ABox, TBox, and RBox statements separately.

RBox

The main component of an RBox is the *role hierarchy*, which enables us to define how the binary relationships are connected to one another.

Definition 2 (Roles and Role Inversion)**.** *Let $\mathcal{L}$ be a $\mathcal{SROIQ}$ language. The set of all **roles** in $\mathcal{L}$ is the set $\mathbf{R} = \{r \mid r \in N_R\} \cup \{r^- \mid r \in N_R\} \cup \{u\}$, where u denotes the **universal role**.*

*For any $r \in N_R$, the role r^- is called the **inverse role** of r. The function of role inversion $\text{Inv} : \mathbf{R} \to \mathbf{R}$ is defined for all $r \in \mathbf{R}$ as follows:*

$$\text{Inv}(r) = \begin{cases} r^-, & \text{if } r \in N_R, \\ s, & \text{if } r = s^-, \text{ for some } s \in N_R, \\ u, & \text{if } r = u. \end{cases}$$

Definition 3 (Role Inclusion Axioms and Role Hierarchies)**.** *Let $\mathcal{L}$ be a $\mathcal{SROIQ}$ language. A **role inclusion axiom** (RIA) is any statement of the form $r_1 \circ r_2 \circ \cdots \circ r_n \sqsubseteq r$, where $n \in \mathbb{Z}^+$ and $r_1, r_2, \ldots, r_n, r \in \mathbf{R}$. For any $r, s \in \mathbf{R}$, we call the RIA $r \sqsubseteq s$ a **simple role inclusion** and r a **subrole** of s.*

*A **role hierarchy** ($\mathcal{H}_R$) is any finite set of RIAs.*

For decidability of reasoning in $\mathcal{SROIQ}$, it is inevitable that the role hierarchy in an RBox is regular [20].

Definition 4 (Simple and Non-Simple Roles)**.** *Let $\mathcal{L}$ be a $\mathcal{SROIQ}$ language and $\mathcal{H}_R$ be a role hierarchy in $\mathcal{L}$. The set $\mathbf{R}^n$ of all **non-simple roles** in $\mathcal{L}$ w.r.t. $\mathcal{H}_R$ is the smallest set for which each of the following conditions holds:*

- *if $r_1 \circ r_2 \circ \cdots \circ r_n \sqsubseteq r \in \mathcal{H}_R$, for $n > 1$ and any $r_1, r_2, \ldots, r_n, r \in \mathbf{R}$, then $r \in \mathbf{R}^n$,*

- *if* $r, s \in \mathbf{R}$, $r \in \mathbf{R}^n$, *and* $r \sqsubseteq s \in \mathcal{H}_R$, *then* $s \in \mathbf{R}^n$,
- *if* $r \in \mathbf{R}^n$, *then* $\mathrm{Inv}(r) \in \mathbf{R}^n$.

The set of all ***simple roles*** *in* $\mathcal{L}$ *w.r.t.* $\mathcal{H}_R$ *is the set* $\mathbf{R}^s = \mathbf{R} \setminus \mathbf{R}^n$.

Definition 5 (Regular Role Hierarchy)**.** *Let* $\mathcal{L}$ *be a* $\mathcal{SROIQ}$ *language and* $\mathcal{H}_R$ *be a role hierarchy in* $\mathcal{L}$*. The* ***role hierarchy*** $\mathcal{H}_R$ ***is regular*** *if there exists a strict partial order*[1] $\prec$ *on* $\mathbf{R}^n$ *such that* $r \prec s$ *iff* $\mathrm{Inv}(r) \prec s$*, for any* $r, s \in \mathbf{R}^n$*, and each RIA in* $\mathcal{H}_R$ *has one of the following forms:*

- $r \circ r \sqsubseteq r$,
- $\mathrm{Inv}(r) \sqsubseteq r$,
- $s_1 \circ s_2 \circ \cdots \circ s_n \sqsubseteq r$,
- $r \circ s_1 \circ s_2 \circ \cdots \circ s_n \sqsubseteq r$,
- $s_1 \circ s_2 \circ \cdots \circ s_n \circ r \sqsubseteq r$,

where $n \in \mathbb{Z}^+$, $r \in N_R$, $s_1, s_2, \ldots, s_n \in \mathbf{R}$*, with* $s_i \prec r$*, for every non-simple role* s_i*, where* $i = 1, 2, \ldots, n$.

The rest of the RBox consists of statements through which we can express that some roles have additional properties, such as reflexivity or symmetry. We collectively call these statements *role characteristics*.

Definition 6 (Role Characteristics)**.** *Let* $\mathcal{L}$ *be a* $\mathcal{SROIQ}$ *language and* $\mathcal{H}_R$ *be a role hierarchy in* $\mathcal{L}$*. A* ***role characteristic*** *is any statement of one of the following forms:* $\mathrm{Ref}(r)$ *(****reflexivity****),* $\mathrm{Irref}(r)$ *(****irreflexivity****),* $\mathrm{Sym}(r)$ *(****symmetry****),* $\mathrm{Asym}(r)$ *(****asymmetry****), or* $\mathrm{Disj}(s, s')$ *(****disjointness****), for any* $r \in \mathbf{R}$ *and* $s, s' \in \mathbf{R}^s$.

A $\mathcal{SROIQ}$ RBox can be thus formally defined as follows.

Definition 7 (RBox)**.** *Let* $\mathcal{L}$ *be a* $\mathcal{SROIQ}$ *language. A* $\mathcal{SROIQ}$ ***RBox*** *is any set* $\mathcal{R} = \mathcal{H}_R \cup \mathrm{Char}_R$*, where* $\mathcal{H}_R$ *is a regular role hierarchy and* Char_R *is a finite set of role characteristics.*

TBox

The terminological part of a knowledge base predominantly allows us to create a complex hierarchy over the set of categories we denote by concept names.

Definition 8 (Concept Expressions)**.** *Let* $\mathcal{L}$ *be a* $\mathcal{SROIQ}$ *language and* $\mathcal{R}$ *be an RBox in* $\mathcal{L}$*. The set* $\mathbf{C}$ *of all* ***concept expressions*** *(****concepts****) in* $\mathcal{L}$ *w.r.t.* $\mathcal{R}$ *is the smallest set for which each of the following conditions is satisfied:*

- *every concept name* $C \in N_C$ *is an element of* $\mathbf{C}$ *(****atomic concept****),*
- *the concepts* $\top$ *(****top concept****) and* $\bot$ *(****bottom concept****) belong to* $\mathbf{C}$,

[1] A strict partial order is a transitive and asymmetric binary relation.

- $\{a_1, a_2, \ldots, a_n\} \in \mathbf{C}$, *for any* $n \in \mathbb{Z}^+$ *and* $a_1, a_2, \ldots, a_n \in N_I$*; the concepts of this form are called* ***nominals****,*
- *if* $C, D \in \mathbf{C}$*, then* $\neg C \in \mathbf{C}$ *(****negation****),* $C \sqcap D \in \mathbf{C}$ *(****intersection/conjunction****), and* $C \sqcup D \in \mathbf{C}$ *(****union/disjunction****),*
- *if* $r \in \mathbf{R}$ *and* $C \in \mathbf{C}$*, then* $\exists r.C \in \mathbf{C}$ *(****existential quantification****), and* $\forall r.C \in \mathbf{C}$ *(****universal quantification****),*
- *if* $r \in \mathbf{R}^s$*, then* $\exists r.\text{Self} \in \mathbf{C}$ *(****self restriction****),*
- *if* $r \in \mathbf{R}^s$*,* $C \in \mathbf{C}$*, and* $n \in \mathbb{N}$*, then* $\geq n\ r.C \in \mathbf{C}$ *(****at-least restriction****), and* $\leq n\ r.C \in \mathbf{C}$ *(****at-most restriction****)*[2]*; these concepts are also jointly called* ***cardinality constraints****.*

Definition 9 (General Concept Inclusion Axioms and TBox)**.** *Let* $\mathcal{L}$ *be a* $\mathcal{SROIQ}$ *language and* $\mathcal{R}$ *be an RBox in* $\mathcal{L}$*. A* ***general concept inclusion axiom*** *(GCI) is any statement of the form* $C \sqsubseteq D$*, where* $C, D \in \mathbf{C}$*, read "C is* ***subsumed*** *by D", "C is a* ***subconcept*** *of D", or "D is a* ***superconcept*** *of C".*

A $\mathcal{SROIQ}$ ***TBox*** *(*$\mathcal{T}$*) is any finite set of GCIs.*

ABox

An ABox serves as a tool to express facts about named individuals inside the model of the world built via the RBox and TBox.

Definition 10 (Individual Assertions and ABox)**.** *Let* $\mathcal{L}$ *be a* $\mathcal{SROIQ}$ *language,* $\mathcal{R}$ *be an RBox in* $\mathcal{L}$*, and* $\mathcal{T}$ *be a TBox in* $\mathcal{L}$*. An* ***individual assertion*** *is a statement in any of the following forms:* $C(a)$ *(****concept assertion****),* $r(a, b)$ *(****role assertion****),* $\neg r(a, b)$ *(****negated role assertion****),* $a \approx b$ *(****equality statement****), or* $a \not\approx b$ *(****inequality statement****), where* $C \in \mathbf{C}, r \in N_R$*, and* $a, b \in N_I$*.*

A $\mathcal{SROIQ}$ ***ABox*** *(*$\mathcal{A}$*) is any finite set of individual assertions.*

Knowledge Base

As we mentioned earlier, a $\mathcal{SROIQ}$ knowledge base is simply a union of an ABox, TBox, and RBox.

Definition 11 (Knowledge Base)**.** *Let* $\mathcal{L}$ *be a* $\mathcal{SROIQ}$ *language. A* $\mathcal{SROIQ}$ ***knowledge base*** *is any set* $\mathcal{K} = \mathcal{R} \cup \mathcal{T} \cup \mathcal{A}$*, where* $\mathcal{R}, \mathcal{T}, \mathcal{A}$ *are an RBox, a TBox, and an ABox in* $\mathcal{L}$*, respectively.*

Before we move on to the semantics of $\mathcal{SROIQ}$, we will look at one example of a $\mathcal{SROIQ}$ knowledge base to better grasp how the knowledge described by sentences in a natural language is converted into a knowledge base.

[2]We assume that zero is a natural number as well.

Example 1 (Knowledge Base)**.** *Imagine we would like to translate the English sentences $S_1, \ldots, S_7$ given below into $\mathcal{SROIQ}$ statements and create a knowledge base.*

S_1: *One is a parent of somebody else if and only if the latter is a child of the former.*
S_2: *If someone is a parent of another parent, then the first is a grandparent of all the children the second has.*
S_3: *No one is a parent of themselves.*
S_4: *Every man is a person.*
S_5: *A happy person is either a man with no child or a person who has at least two children.*
S_6: *Alice is a parent of Bob.*
S_7: *Chris is a happy person.*

To accomplish this task, we first define a $\mathcal{SROIQ}$ language $\mathcal{L}$ with

$$N_R = \{\texttt{parentOf}, \texttt{childOf}, \texttt{grandparentOf}\},$$
$$N_C = \{\texttt{Person}, \texttt{Man}, \texttt{HappyPerson}\},$$
$$N_I = \{\texttt{alice}, \texttt{bob}, \texttt{chris}\}.$$

The sentences $S_1, \ldots, S_7$ can be then transformed into the statements $S^{\mathcal{L}}_{1.1}, S^{\mathcal{L}}_{1.2}, S^{\mathcal{L}}_2, \ldots, S^{\mathcal{L}}_7$ in $\mathcal{L}$ as follows:

$S^{\mathcal{L}}_{1.1}$: $\texttt{parentOf} \sqsubseteq \texttt{childOf}^-$,
$S^{\mathcal{L}}_{1.2}$: $\texttt{childOf} \sqsubseteq \texttt{parentOf}^-$,
$S^{\mathcal{L}}_2$: $\texttt{parentOf} \circ \texttt{parentOf} \sqsubseteq \texttt{grandparentOf}$,
$S^{\mathcal{L}}_3$: $\text{Irref}(\texttt{parentOf})$,
$S^{\mathcal{L}}_4$: $\texttt{Man} \sqsubseteq \texttt{Person}$,
$S^{\mathcal{L}}_5$: $\texttt{HappyPerson} \sqsubseteq (\texttt{Man} \sqcap \forall \texttt{parentOf}.\bot) \sqcup (\texttt{Person} \sqcap \geq 2\ \texttt{parentOf}.\top)$,
$S^{\mathcal{L}}_6$: $\texttt{parentOf}(\texttt{alice}, \texttt{bob})$,
$S^{\mathcal{L}}_7$: $\texttt{HapyPerson}(\texttt{chris})$,

where $S^{\mathcal{L}}_{1.1}, S^{\mathcal{L}}_{1.2}, S^{\mathcal{L}}_2, S^{\mathcal{L}}_3$ together constitute an RBox $\mathcal{R}$, $S^{\mathcal{L}}_4, S^{\mathcal{L}}_5$ form a TBox $\mathcal{T}$, and $S^{\mathcal{L}}_6, S^{\mathcal{L}}_7$ represent an ABox $\mathcal{A}$. The corresponding knowledge base $\mathcal{K}$ would thus be a union of $\mathcal{R}, \mathcal{T}$ and $\mathcal{A}$, i.e., $\mathcal{K} = \{S^{\mathcal{L}}_{1.1}, S^{\mathcal{L}}_{1.2}, S^{\mathcal{L}}_2, \ldots, S^{\mathcal{L}}_7\}$.

2.1.2 Semantics

Knowing the rules for writing statements in $\mathcal{SROIQ}$ and understanding their intended use, we can discuss their formal semantics in detail, i.e., how we interpret them and investigate whether they are satisfied.

Interpretations

Since every knowledge base is merely an abstraction of the world it tries to model, we need to define the way we will materialize this abstraction given a set of entities. In DLs, this need is fulfilled by *interpretations*.

Definition 12 (Interpretation)**.** *Let $\mathcal{L}$ be a $\mathcal{SROIQ}$ language. An **interpretation** is a pair $\mathcal{I} = (\Delta^\mathcal{I}, \cdot^\mathcal{I})$, where $\Delta^\mathcal{I}$ is a non-empty set, called domain, and $\cdot^\mathcal{I}$ is an interpretation function which maps*

- *each $a \in N_I$ to an element $a^\mathcal{I} \in \Delta^\mathcal{I}$ (**individual**),*
- *each $C \in N_C$ to a subset $C^\mathcal{I} \subseteq \Delta^\mathcal{I}$ (**concept extension**),*
- *each $r \in N_R$ to a subset $r^\mathcal{I} \subseteq \Delta^\mathcal{I} \times \Delta^\mathcal{I}$ (**role extension**).*

The interpretation function $\cdot^\mathcal{I}$ is then extended to all $\mathcal{SROIQ}$ roles and concepts as follows:

- $u^\mathcal{I} = \Delta^\mathcal{I} \times \Delta^\mathcal{I}$,
- $(r^-)^\mathcal{I} = \{(\delta, \hat{\delta}) \in \Delta^\mathcal{I} \times \Delta^\mathcal{I} \mid (\hat{\delta}, \delta) \in r^\mathcal{I}\}$,
- $\top^\mathcal{I} = \Delta^\mathcal{I}$, $\bot^\mathcal{I} = \emptyset$,
- $(\neg C)^\mathcal{I} = \Delta^\mathcal{I} \setminus C^\mathcal{I}$,
- $\{a_1, a_2, \ldots, a_k\}^\mathcal{I} = \{a_1^\mathcal{I}, a_2^\mathcal{I}, \ldots, a_k^\mathcal{I}\}$,
- $(C \sqcap D)^\mathcal{I} = C^\mathcal{I} \cap D^\mathcal{I}$, $(C \sqcup D)^\mathcal{I} = C^\mathcal{I} \cup D^\mathcal{I}$,
- $(\exists s.C)^\mathcal{I} = \{\delta \mid \text{there is } \hat{\delta} \in \Delta^\mathcal{I} \text{ with } (\delta, \hat{\delta}) \in s^\mathcal{I} \text{ and } \hat{\delta} \in C^\mathcal{I}\}$,
- $(\forall s.C)^\mathcal{I} = \{\delta \mid \text{for all } \hat{\delta} \in \Delta^\mathcal{I}, \text{ if } (\delta, \hat{\delta}) \in s^\mathcal{I}, \text{ then } \hat{\delta} \in C^\mathcal{I}\}$,
- $(\exists s.\text{Self})^\mathcal{I} = \{\delta \mid (\delta, \hat{\delta}) \in s^\mathcal{I}\}$,
- $(\geq n\ t.C)^\mathcal{I} = \{\delta \mid |\{\hat{\delta} \in \Delta^\mathcal{I} \mid (\delta, \hat{\delta}) \in t^\mathcal{I} \text{ and } \hat{\delta} \in C^\mathcal{I}\}| \geq n\}$,
- $(\leq n\ t.C)^\mathcal{I} = \{\delta \mid |\{\hat{\delta} \in \Delta^\mathcal{I} \mid (\delta, \hat{\delta}) \in t^\mathcal{I} \text{ and } \hat{\delta} \in C^\mathcal{I}\}| \leq n\}$,

for all $r \in N_R$, $s \in \mathbf{R}$, $t \in \mathbf{R}^s$, $C, D \in \mathbf{C}$, $a_1, a_2, \ldots, a_k \in N_I$, $k \in \mathbb{Z}$, and $n \in \mathbb{N}$.

*Every individual $\delta \in C^\mathcal{I}$ is called an **instance** of C. If $(\delta, \hat{\delta}) \in r^\mathcal{I}$, then we say that $\hat{\delta}$ is an **r-filler** of δ in $\mathcal{I}$.*

For illustration, we will examine one model of the knowledge base constructed in Example 1.

Example 2 (Interpretation)**.** *Let $\mathcal{L}$ be the $\mathcal{SROIQ}$ language and $\mathcal{K}$ be the knowledge base in $\mathcal{L}$ defined in Example 1. Now, let us take the interpretation $\mathcal{I} = (\Delta^\mathcal{I}, \cdot^\mathcal{I})$ where*

$$\Delta^\mathcal{I} = \{A, B, C, D, E, F\},$$
$$\texttt{alice}^\mathcal{I} = A, \quad \texttt{bob}^\mathcal{I} = B, \quad \texttt{chris}^\mathcal{I} = C,$$
$$\texttt{Person}^\mathcal{I} = \{A, B, C, D\}, \quad \texttt{Man}^\mathcal{I} = \{B, C, D\}, \quad \texttt{HappyPerson}^\mathcal{I} = \{A, C, D\},$$
$$\texttt{parentOf}^\mathcal{I} = \{(A, B), (A, D), (E, F)\}, \quad \texttt{childOf}^\mathcal{I} = \{(B, A), (D, A), (F, E)\},$$
$$\texttt{grandparentOf}^\mathcal{I} = \{(E, F)\}.$$

Apparently, $\mathcal{I}$ is a model of $\mathcal{K}$ by Definition 12. Note that not every individual in $\Delta^{\mathcal{I}}$ has to be named, that is, there can be $\delta \in \Delta^{\mathcal{I}}$ for which no $a \in N_I$ exists such that $a^{\mathcal{I}} = \delta$ (for instance, D). Conversely, we could re-define $\mathcal{I}$ so that $\texttt{Bob}^{\mathcal{I}} = \texttt{Chris}^{\mathcal{I}} = B$. Of course, when an individual from $\Delta^{\mathcal{I}}$ is identified by more than one individual name, all individual assertions referring to any of these individual names apply to such individual. Looking at the definition of $\cdot^{\mathcal{I}}$, we can also see the importance of diligence in modeling, since for example in $\mathcal{I}$, E is simultaneously a parent and a grandparent of F, which is possible due to the fact that $\mathcal{K}$ does not require the disjointness of `parentOf` *and* `grandparentOf`*.*

Seeing how interpretations work, it slowly becomes evident that apart from building a concept hierarchy, we are also able to define various constraints with the help of GCIs. For instance, we can assert that all individuals related to another individual (or to themselves) via a given role r have to be instances of a particular concept, called *domain* and referred to as $\mathrm{dom}(r)$. Returning back to Example 1, to set the domain of the role `parentOf` to `Person`, that is, to tell that all parents have to be people, we would use the statement

$$\neg\texttt{Person} \sqcap \exists\texttt{parentOf}.\top \sqsubseteq \bot.$$

Analogously, it is possible to restrict the *range* of r, i.e., to state that every individual to which any individual is connected through r must be an instance of a specific concept, denoted $\mathrm{rng}(r)$. To give an example, we may express that the range of the role `parentOf` has to be `Person`, or equivalently, that one can only be a parent of a person, as follows:

$$\exists\texttt{parentOf}.\neg\texttt{Person} \sqsubseteq \bot.$$

Satisfaction of Statements

Determining if a statement is true is always bound to a specific interpretation, as this process corresponds to examining whether a particular instance of the world conforms to certain constraints. Therefore, the definition of *statement satisfaction* also depends on the provided interpretation.

Definition 13 (Statement Satisfaction)**.** *Let $\mathcal{L}$ be a $\mathcal{SROIQ}$ language and $\mathcal{I}$ be an interpretation. The relation "the **interpretation $\mathcal{I}$ satisfies a statement** α" ($\mathcal{I} \vDash \alpha$) is given by the following definition:*

- $\mathcal{I} \vDash r_1 \circ r_2 \circ \cdots \circ r_n \sqsubseteq r$ *iff* $\{(\delta_0, \delta_n) \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \mid$ *there exist* $\delta_1, \delta_2, \ldots, \delta_{n-1} \in \Delta^{\mathcal{I}}$ *such that* $(\delta_{i-1}, \delta_i) \in r_i^{\mathcal{I}}$*, for* $i = 1, 2, \ldots, n\} \subseteq r^{\mathcal{I}}$*,*
- $\mathcal{I} \vDash \mathrm{Ref}(r)$ *iff* $\{(\delta, \delta) \mid \delta \in \Delta^{\mathcal{I}}\} \subseteq r^{\mathcal{I}}$*,*
- $\mathcal{I} \vDash \mathrm{Irref}(r)$ *iff* $\{(\delta, \delta) \mid \delta \in \Delta^{\mathcal{I}}\} \cap r^{\mathcal{I}} = \emptyset$*,*

- $\mathcal{I} \vDash \mathrm{Sym}(r)$ *iff it holds that* $(\delta, \hat{\delta}) \in r^{\mathcal{I}}$ *implies* $(\hat{\delta}, \delta) \in r^{\mathcal{I}}$*, for any* $\delta, \hat{\delta} \in \Delta^{\mathcal{I}}$*,*
- $\mathcal{I} \vDash \mathrm{Asym}(r)$ *iff it holds that* $(\delta, \hat{\delta}) \in r^{\mathcal{I}}$ *implies* $(\hat{\delta}, \delta) \notin r^{\mathcal{I}}$*, for any* $\delta, \hat{\delta} \in \Delta^{\mathcal{I}}$*,*
- $\mathcal{I} \vDash \mathrm{Disj}(s, t)$ *iff* $s^{\mathcal{I}} \cap t^{\mathcal{I}} = \emptyset$*,*
- $\mathcal{I} \vDash C \sqsubseteq D$ *iff* $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$*,*
- $\mathcal{I} \vDash C(a)$ *iff* $a^{\mathcal{I}} \in C^{\mathcal{I}}$*,*
- $\mathcal{I} \vDash r(a, b)$ *iff* $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in r^{\mathcal{I}}$*,*
- $\mathcal{I} \vDash \neg r(a, b)$ *iff* $(a^{\mathcal{I}}, b^{\mathcal{I}}) \notin r^{\mathcal{I}}$*,*
- $\mathcal{I} \vDash a \approx b$ *iff* $a^{\mathcal{I}} = b^{\mathcal{I}}$*,*
- $\mathcal{I} \vDash a \not\approx b$ *iff* $a \neq b$*,*

where $r_1, r_2, \ldots, r_n, r \in \mathbf{R}$*,* $s, t \in \mathbf{R}^s$*,* $C \in \mathbf{C}$*,* $a, b \in N_I$*, and* $n \in \mathbb{Z}$*.*

The interpretation $\mathcal{I}$ *satisfies (is a* ***model*** *of) a knowledge base* $\mathcal{K}$ *(*$\mathcal{I} \vDash \mathcal{K}$*) if it satisfies all statements in* $\mathcal{K}$*.*

The notation $\mathcal{I} \nvDash \alpha$ *(*$\mathcal{I} \nvDash \mathcal{K}$*) indicates that the interpretation* $\mathcal{I}$ *does not satisfy the statement* α *(the knowledge base* $\mathcal{K}$*).*

Entailment

Having acquired some knowledge, we usually seek to make new inferences from this knowledge. In order to have a formal apparatus to verify that our observations logically follow from a knowledge base, we define the notion of *entailment*.

Definition 14 (Entailment)**.** *Let* $\mathcal{L}$ *be a* $\mathcal{SROIQ}$ *language and* $\mathcal{K}$ *be a knowledge base in* $\mathcal{L}$*. We say that the* ***knowledge base*** $\mathcal{K}$ ***entails a statement*** α *(*$\mathcal{K} \vDash \alpha$*) if for all interpretations* $\mathcal{I}$*, it is true that if* $\mathcal{I} \vDash \mathcal{K}$*, then* $\mathcal{I} \vDash \alpha$*.*

The knowledge base $\mathcal{K}$ *entails a set of statements* S *(*$\mathcal{K} \vDash S$*) if it entails each statement in* S*. The notation* $\mathcal{K} \nvDash \alpha$ *(*$\mathcal{K} \nvDash S$*) indicates that the knowledge base* $\mathcal{K}$ *does not entail the statement* α *(the set of statements* S*).*

Logical Equivalence and Strict Subsumption

We conclude the section on $\mathcal{SROIQ}$ semantics with the definitions of *logical equivalence* and *strict subsumption*, which are convenient ways to detect likeness of two concepts or roles and exclude such pairs from the relation of subsumption, respectively.

Definition 15 (Logical Equivalence)**.** *Let* $\mathcal{L}$ *be a* $\mathcal{SROIQ}$ *language and* $\mathcal{K}$ *be a knowledge base in* $\mathcal{L}$*.* ***Concepts*** $C, D \in \mathbf{C}$ *are* ***logically equivalent*** *w.r.t.* $\mathcal{K}$ *(*$\mathcal{K} \vDash C \equiv D$*) if and only if* $\mathcal{K} \vDash \{C \sqsubseteq D, D \sqsubseteq C\}$*.* ***Roles*** $r, s \in \mathbf{R}$ *are* ***logically equivalent*** *w.r.t.* $\mathcal{K}$ *(*$\mathcal{K} \vDash r \equiv s$*) if and only if* $\mathcal{K} \vDash \{r \sqsubseteq s, s \sqsubseteq r\}$*.*

If in addition $\mathcal{K} = \emptyset$*, the concepts* C, D *(roles* r, s*) are said to be logically equivalent, denoted* $C \equiv D$ *(*$r \equiv s$*).*

Definition 16 (Strict Subsumption)**.** *Let $\mathcal{L}$ be a $\mathcal{SROIQ}$ language and $\mathcal{K}$ be a knowledge base in $\mathcal{L}$. A **concept** $C \in \mathbf{C}$ is **strictly subsumed** by a concept $D \in \mathbf{C}$ w.r.t. $\mathcal{K}$ ($\mathcal{K} \models C \sqsubset D$) if and only if $\mathcal{K} \models \{C \sqsubseteq D\}$ and $\mathcal{K} \not\models C \equiv D$. A **role** $r \in \mathbf{R}$ is **strictly subsumed** by a role $s \in \mathbf{R}$ w.r.t. $\mathcal{K}$ ($\mathcal{K} \models r \sqsubset s$) if and only if $\mathcal{K} \models \{r \sqsubseteq s\}$ and $\mathcal{K} \not\models r \equiv s$.*

*If $\mathcal{K} \models C \sqsubset D$, where $C, D \in \{\top\} \cup N_C$, and there exists no $E \in \mathbf{C}$ such that $\mathcal{K} \models C \sqsubset E$ and $\mathcal{K} \models E \sqsubset D$, we call C a **direct subconcept** of D. If $\mathcal{K} \models r \sqsubset s$, where $r, s \in \{u\} \cup N_R$, and there exists no $t \in \mathbf{C}$ such that $\mathcal{K} \models r \sqsubset t$ and $\mathcal{K} \models t \sqsubset s$, we call r a **direct subrole** of s.*

2.2 OWL 2 DL

The OWL 2 DL [13] is a member of the family of OWL ontology languages, which were proposed by the World Wide Web Consortium (W3C) to back the Semantic Web initiative [2] – an initiative for a next-generation Web where information is semantically organized. Since then, OWL ontology languages have been used in a wide range of areas apart from the Semantic Web, where automated reasoning and semantic retrieval helps with organization of large amounts of data (e.g., medicine [8] or energy management [9]). Ontology languages in general were and still are evolving alongside DLs as a means of structured knowledge representation. Some ontology languages, such as the OWL 2 DL, even take DLs as their formal bases. More precisely, the OWL 2 DL is based on the $\mathcal{SROIQ}$ DL and is essentially its extension. For example, OWL 2 adds the support for data types, so the role fillers can be numbers, strings, etc. There are also several extra-logical features OWL 2 provides beyond the standard DL tools, e.g., annotations. Lastly, the OWL 2 syntax differs from the discussed $\mathcal{SROIQ}$ syntax as well, resulting from the slightly different origins of ontology languages and DLs.

In our research, we use an OWL 2 DL ontology to represent data (a knowledge base in effect, see Table 2.1) but the extensions of $\mathcal{SROIQ}$ are not yet utilized and all OWL 2 statements we present can and will be written in the $\mathcal{SROIQ}$ syntax. Therefore, we do not dive into the details of OWL 2 and its connection to $\mathcal{SROIQ}$, which are, however, thoroughly covered by Rudoplh [20] and Baader et al. [4]. We only introduce a bit of OWL terminology by linking the most fundamental OWL terms to their respective DL counterparts in Table 2.1. In later chapters, we use the corresponding terms from this table interchangeably, albeit we incline to the DL terminology in the chapters closely related to DLs, i.e., Chapters 3 and 5, and to the OWL nomenclature elsewhere.

Table 2.1: Comparison of DL and OWL terminology.

DLs	OWL
knowledge base	ontology
concept	class
role	object property
data-type role	data property
top concept	`owl:Thing`
bottom concept	`owl:Nothing`
universal role	`owl:topObjectProperty`
universal data-type role	`owl:topDataProperty`

3 Learning in Description Logics

Since DLs, especially those supporting role hierarchies, give us the ability to capture various aspects of an application domain, one may choose from several learning objectives to pursue. In this section, we thus first define the DL learning problem a solution to which we aim to find in malware detection. Next, we present the approach to this problem that we decided to employ, together with the selected algorithms.

3.1 Concept-Learning Problem

In our research, we focus on solving the problem of learning concept expressions, which we define below [17].

Definition 17 (Concept-Learning Problem)**.** *Let $\mathcal{L}$ be a DL language and $\mathcal{K}$ be a knowledge base in $\mathcal{L}$. Let $E^+ \subseteq N_I$ be a set of positive examples and $E^- \subseteq N_I$ be a set of negative examples, with $E^+ \cap E^- = \emptyset$.*

Given $\mathcal{K}$, E^+, and E^-, find a concept expression $C \in \mathbf{C}$, also called a description, such that both of the following conditions are satisfied:

$$\mathcal{K} \models C(a), \text{ for all } a \in E^+, \tag{3.1}$$

$$\mathcal{K} \models \neg C(a), \text{ for all } a \in E^-. \tag{3.2}$$

If $\mathcal{K} \models C(a)$ holds for some $a \in E^+ \cup E^-$ and $C \in \mathbf{C}$, we say that C covers a.

This definition can be interpreted as follows: Given some background knowledge and two sets of individuals, positive and negative examples, respectively, the goal of concept learning is to use the background knowledge to find a minimal description of the positive examples, where minimality is achieved by ensuring that none of the negative examples fits the description.

Notice that we always supply additional information about which instances are positive and which are negative, in other words, the learning is supervised. There is also an adaptation of the above definition that allows unlabeled examples as well. This is useful for learning under the *open-world assumption*, i.e., when we do not conclude that a certain observation is false if it does not follow from the knowledge base. Despite

relying on the open-world assumption is a common practice in the field of knowledge representation due to the awareness of the possibility that the current knowledge may be incomplete, we decided to work under the *closed-world assumption*, as we first wanted to assess the learning capabilities of algorithms in these arguably less adverse conditions, in which exhaustive information is supposed to be provided.

3.2 Concept Learning Using Refinement Operators

Definition 17 states that a concept learning problem is considered to be solved once an appropriate description of positive examples is found. Hence, it is possible to cast this problem to a search problem, with the search space being simply the space of all valid concept expressions. One of the approaches to order the elements of this search space is to exploit the standard DL relation of subsumption [17]. Such successor functions are called *refinement operators*.

Definition 18 (Refinement Operator)**.** *Let $\mathcal{L}$ be a DL language and $\mathcal{K}$ be a knowledge base in $\mathcal{L}$ with no ABox. A mapping $\rho : \mathbf{C} \to 2^{\mathbf{C}}$ is a **downward (upward) refinement operator** in the quasi-ordered*[1] *space $(\mathcal{L}, \sqsubseteq)$ if for all $C \in \mathbf{C}$, it holds that $D \in \rho(C)$ implies $\mathcal{K} \models D \sqsubseteq C$ ($\mathcal{K} \models C \sqsubseteq D$). The concept D is called a **refinement** of C and more specifically, a **specialization (generalization)** of C.*

For example, if we let $\rightsquigarrow$ denote a single refinement operation, one branch of a search tree induced by a downward refinement operator may have the following form (with the leftmost concept being in the root node and the rightmost in the leaf node):

$$\top \rightsquigarrow \forall r.\top \rightsquigarrow \forall r.C \rightsquigarrow (\forall r.C) \sqcap D. \tag{3.3}$$

We refer to such a sequence of refinement operations as a *refinement chain*. Usually, the downward-refinement process starts from the most-generic concept, i.e., the top concept, which has the maximum potential to specialize.

In our research, we study the suitability of concept-learning algorithms utilizing the refinement operator called simply ρ, all of which is implemented in DL-Learner [6], a state-of-the-art Java framework for learning in DLs. In the next two sections, we first present the definition of the ρ operator and then discuss the examined algorithms with their peculiarities.

3.2.1 The ρ Refinement Operator

Proposed by Lehmann and Hitzler [17], the ρ refinement operator was the first operator for solving the concept-learning problem in DLs. Before we formally define ρ, it is inevitable to introduce several auxiliary notions.

[1] A quasi-order is a reflexive and transitive binary relation.

To move through the subsumption hierarchies created over the sets of concept names and role names, we need functions for crawling them both in the upward ($\mathrm{sh}^{\mathbf{C}}_{\uparrow}$ and $\mathrm{sh}^{\mathbf{R}}_{\uparrow}$) and downward ($\mathrm{sh}^{\mathbf{C}}_{\downarrow}$ and $\mathrm{sh}^{\mathbf{R}}_{\downarrow}$) direction.

Definition 19 (Subsumption Hierarchy Crawlers)**.** *Let $\mathcal{L}$ be a DL language and $\mathcal{K}$ be a knowledge base in $\mathcal{L}$ with an ABox $\mathcal{A}$. Let $\mathcal{K}' = \mathcal{K} \setminus \mathcal{A}$. We define the functions $\mathrm{sh}^{\mathbf{C}}_{\uparrow} : \{\top\} \cup N_C \to 2^{\{\top\} \cup N_C}$ and $\mathrm{sh}^{\mathbf{C}}_{\downarrow} : \{\top\} \cup N_C \to 2^{\{\top\} \cup N_C}$ for all $C \in \{\top\} \cup N_C$ as follows:*

$$\mathrm{sh}^{\mathbf{C}}_{\uparrow}(C) = \{D \in \{\top\} \cup N_C \mid \mathcal{K}' \vDash C \sqsubset D \text{ and } C \text{ is a direct subconcept of } D \text{ w.r.t. } \mathcal{K}'\},$$
$$\mathrm{sh}^{\mathbf{C}}_{\downarrow}(C) = \{D \in \{\top\} \cup N_C \mid \mathcal{K}' \vDash D \sqsubset C \text{ and } C \text{ is a direct subconcept of } D \text{ w.r.t. } \mathcal{K}'\}.$$

The functions $\mathrm{sh}^{\mathbf{R}}_{\uparrow} : \{u\} \cup N_R \to 2^{\{u\} \cup N_R}$ and $\mathrm{sh}^{\mathbf{R}}_{\downarrow} : \{u\} \cup N_R \to 2^{\{u\} \cup N_R}$ map each $r \in \{u\} \cup N_R$ to the sets below:

$$\mathrm{sh}^{\mathbf{R}}_{\uparrow}(r) = \{s \in \{u\} \cup N_R \mid \mathcal{K}' \vDash r \sqsubset s \text{ and } r \text{ is a direct subrole of } s \text{ w.r.t. } \mathcal{K}'\},$$
$$\mathrm{sh}^{\mathbf{R}}_{\downarrow}(r) = \{s \in \{u\} \cup N_R \mid \mathcal{K}' \vDash s \sqsubset r \text{ and } s \text{ is a direct subrole of } r \text{ w.r.t. } \mathcal{K}'\}.$$

The presence of domain and range assertions facilitates the ρ operator to avoid generating refinements that certainly do not cover any individual because they do not satisfy a domain or range restriction. However, ρ was designed to work only with *atomic domains* and *ranges*, i.e., those which are either concept names or the top concept.

Definition 20 (Atomic Domains And Ranges)**.** *Let $\mathcal{L}$ be a DL language and $\mathcal{K}$ be a knowledge base in $\mathcal{L}$.*

*The **atomic domain** of a role $r \in \mathbf{R}$ is a concept $\mathrm{ad}(r) \in \{\top\} \cup N_C$ such that $\mathcal{K} \vDash \mathrm{dom}(r) \sqsubseteq \mathrm{ad}(r)$ and there is no $C \in \{\top\} \cup N_C$ with $\mathcal{K} \vDash \mathrm{dom}(r) \sqsubseteq C$ and $\mathcal{K} \vDash C \sqsubset \mathrm{ad}(r)$.*

*The **atomic range** of a role $r \in \mathbf{R}$ is a concept $\mathrm{ar}(r) \in \{\top\} \cup N_C$ such that $\mathcal{K} \vDash \mathrm{rng}(r) \sqsubseteq \mathrm{ar}(r)$ and there is no $C \in \{\top\} \cup N_C$ with $\mathcal{K} \vDash \mathrm{rng}(r) \sqsubseteq C$ and $\mathcal{K} \vDash C \sqsubset \mathrm{ar}(r)$.*

Conversely, given a concept name, we are able to determine the set of *applicable role names*, whose atomic domains are not disjoint from this concept name, based on the information about domains.

Definition 21 (Applicable Role Names)**.** *Let $\mathcal{L}$ be a DL language, $\mathcal{K}$ be a knowledge base in $\mathcal{L}$, and $C \in \{\top\} \cup N_C$ be a concept. The set of all **applicable role names** for the concept C is the set*

$$\mathrm{app}(C) = \{r \in N_R \mid \mathcal{K} \nvDash \mathrm{ad}(r) \sqcap C \equiv \bot\}.$$

*The set of **the most general applicable role names** for the concept C is the set*

$$\mathrm{mgr}(C) = \{r \in \mathrm{app}(C) \mid \text{there is no } s \in \mathrm{app}(C) \text{ such that } \mathcal{K} \vDash r \sqsubset s\}.$$

With the above terms defined, we can now, according to [17], formulate the definition of ρ for the $\mathcal{SROIQ}$ DL without nominals and inverse roles, which we omit because the rules for their refinement were not enabled in our experiments.

Definition 22 (The ρ Refinement Operator)**.** *Let $\mathcal{L}$ be a $\mathcal{SROIQ}$ language without nominals and inverse roles. Let $\mathcal{K}$ be a knowledge base in $\mathcal{L}$. For any $B \in \{\top\} \cup N_C$, let M_B be the set of all* ***top-concept refinements*** *not disjoint from B defined as follows:*

$$\begin{aligned} M_B = \{A \mid A \in N_C, \mathcal{K} \not\models A \sqcap B \equiv \bot, \mathcal{K} \not\models A \sqcap B \equiv B, \text{ and} \\ \text{there is no } A' \in N_C \text{ such that } \mathcal{K} \models A \sqsubset A'\} \\ \cup \{\neg A \mid A \in N_C, \mathcal{K} \not\models \neg A \sqcap B \equiv \bot, \mathcal{K} \not\models \neg A \sqcap B \equiv B, \text{ and} \\ \text{there is no } A' \in N_C \text{ such that } \mathcal{K} \models A' \sqsubset A\} \\ \cup \{\exists r.\top \mid r \in \mathrm{mgr}(B)\} \\ \cup \{\forall r.\top \mid r \in \mathrm{mgr}(B)\} \\ \cup \{\leq \ \mathrm{mf}(r) r.\top \mid r \in \mathrm{mgr}(B)\}, \end{aligned}$$

where $\mathrm{mf}(r) = \max_{a \in N_I} |\{b \mid \mathcal{K} \models r(a,b)\}|$*, for all $r \in N_R$.*

The ***refinement operator*** $\rho : \mathbf{C} \to 2^{\mathbf{C}}$ *maps every $C \in \mathbf{C}$ to a set*

$$\rho(C) = \begin{cases} \{\bot\} \cup \rho_\top(C), & \text{if } C = \top, \\ \rho_\top(C), & \text{otherwise}, \end{cases}$$

where ρ_B is given for each ***domain of refinement*** *$B \in \{\top\} \cup N_C$ as follows:*

$$\rho_B(C) = \begin{cases} \emptyset, & \text{if } C = \bot, \\ \bigcup_{m=1}^{\infty} \{D_1 \sqcup D_2 \sqcup \cdots \sqcup D_m \\ \qquad \mid D_i \in M_B, \text{ for } i = 1, 2, \ldots, m\}, & \text{if } C = \top, \\ \{A' \mid A' \in \mathrm{sh}^{\mathbf{C}}_{\downarrow}(A)\} \\ \quad \cup \{A \sqcap D \mid D \in \rho_B(\top)\}, & \text{if } C = A, \\ \{\neg A' \mid A' \in \mathrm{sh}^{\mathbf{C}}_{\uparrow}(A)\} \\ \quad \cup \{\neg A \sqcap D \mid D \in \rho_B(\top)\}, & \text{if } C = \neg A, \\ \{\exists r.E \mid E \in \rho_{\mathrm{ar}(r)}(D)\} \\ \quad \cup \{(\exists r.D) \sqcap E \mid E \in \rho_B(\top)\} \\ \quad \cup \{\geq 2 r.D\} \\ \quad \cup \{\exists t.D \mid t \in \mathrm{sh}^{\mathbf{R}}_{\downarrow}(r)\}, & \text{if } C = \exists r.D, \\ \{\forall r.E \mid E \in \rho_{\mathrm{ar}(r)}(D)\} \\ \quad \cup \{(\forall r.D) \sqcap E \mid E \in \rho_B(\top)\} \\ \quad \cup \{\forall r.\bot \mid D \in N_C \text{ and } \mathrm{sh}^{\mathbf{C}}_{\downarrow}(D) = \emptyset\} \\ \quad \cup \{\forall t.D \mid t \in \mathrm{sh}^{\mathbf{R}}_{\downarrow}(r)\}, & \text{if } C = \forall r.D, \end{cases}$$

$$\rho_B(C) = \begin{cases} \{\geq n+1\ r.D \mid n < \mathrm{mf}(r)\} & \\ \quad \cup \{\geq n\ s.E \mid E \in \rho_{\mathrm{ar}(s)}(D)\} & \\ \quad \cup \{(\geq n\ s.D) \sqcap E \mid E \in \rho_B(\top)\}, & \text{if } C = \geq n\ s.D, \\ \{\leq n-1\ r.D \mid n > 1\} & \\ \quad \cup \{\leq n\ s.E \mid E \in \rho_{\mathrm{ar}(s)}(D)\} & \\ \quad \cup \{(\leq n\ s.D) \sqcap E \mid E \in \rho_B(\top)\}, & \text{if } C = \leq n\ s.D, \\ \{C_1 \sqcap \cdots \sqcap C_{i-1} \sqcap D \sqcap C_{i+1} \sqcap \cdots \sqcap C_k & \\ \quad \mid D \in \rho_B(C_i),\ \text{for } i = 1, 2, \ldots, k\}, & \text{if } C = C_1 \sqcap C_2 \sqcap \cdots \sqcap C_k, \\ \{C_1 \sqcup \cdots \sqcup C_{i-1} \sqcup D \sqcup C_{i+1} \sqcup \cdots \sqcup C_k & \\ \quad \mid D \in \rho_B(C_i),\ \text{for } i = 1, 2, \ldots, k\} & \\ \quad \cup \{(C_1 \sqcup C_2 \sqcup \cdots \sqcup C_k) \sqcap D \mid D \in \rho_B(\top)\}, & \text{if } C = C_1 \sqcup C_2 \sqcup \cdots \sqcup C_k, \end{cases}$$

for any $n \in \mathbb{N}, k \geq 2, A \in N_C, r \in \mathbf{R}, s \in \mathbf{R}^s, C_1, C_2, \ldots, C_k, C, D \in \mathbf{C}$.

Intuitively, M_B serves as a basis from which all refinements can be constructed by repeatedly applying ρ to the obtained concepts, starting with the top concept. Normally, one would use ρ or $\rho_\top$ to get refinements but it might be also beneficial to employ ρ_B directly if we know that all examples we aim to describe are instances of an atomic concept B.

Although the formal definition of ρ is strict, its implementation in DL-Learner is largely configurable. One of the options allows us to control the use of the *"some-only" rule*, with which we later deal in Sections 5.1.3 and 6.3. If this rule is applied, a universal quantification of a role can occur in a refinement only in conjunction with an existential quantification of the same role. For example, the descriptions of the form $(\forall r.C) \sqcup D$, for any $C, D \in \mathbf{C}$ and $r \in \mathbf{R}$, would be considered invalid, but a concept such as $(\exists r.E \sqcap \forall r.C) \sqcup D$, with any $E \in \mathbf{C}$ not disjoint from C, would be accepted. Taking this precaution precludes the algorithms from over-fitting when no negative example that does not have any r-filler is seen during learning, in which case, the first example description covers exactly the same set of individuals as the second while being significantly shorter and thus preferred by each algorithm.

As the algorithms using ρ learn from examples, we can also choose between *instance-based* and *explicit* checks of concept disjointness for the construction of M_B, that is, whether to infer the disjointness of two concepts from the fact that they have no instance in common, or to rely solely on the available concept-disjointness axioms and the information on disjointness they entail. This raises the necessity to have a tool to check if an individual is an instance of a concept (*instance checking*) or perform reasoning over the set of concept-disjointness axioms. For these purposes, DL-Learner provides various reasoner components, such as the *closed-world reasoner* that is the

most relevant for our use case because it makes the assumption of a closed world.

Other configuration parameters are discussed in Section 6.3 as needed.

3.2.2 Algorithms

In our experiments, we evaluated the performance of four concept-learning algorithms, namely, OCEL, CELOE, ParCEL, and ParCEL-Ex version 2 (hereafter just ParCEL-Ex). To solve the concept-learning problem, each of these algorithms searches the space of all valid concept expressions in a given language ordered by the ρ refinement operator with the guidance of an additional heuristic, measuring mainly how accurate a concept is (exact definition of accuracy is given in Section 6.2). However, notice that the ρ operator is infinite, meaning that there exists a concept C for which the set $\rho(C)$ is not finite (this actually holds for all concepts in the case of ρ). Therefore, the algorithms limit the refinements to only those whose length is lower than or equal to a specified value, where length is defined quite naturally as in Definition 23 below.

Definition 23 (Concept and Role Length)**.** *Let $\mathcal{L}$ be a $\mathcal{SROIQ}$ language. The **length of a role** $r \in \mathbf{R}$, referred to as $\mathrm{len}^{\mathbf{R}}(r)$, is determined as follows:*

$$\mathrm{len}^{\mathbf{R}}(C) = \begin{cases} 1, & \text{if } r \in \{u\} \cup N_R, \\ 2, & \text{otherwise.} \end{cases}$$

*The **length of a concept** $C \in \mathbf{C}$, denoted $\mathrm{len}^{\mathbf{C}}(C)$, is calculated based on this recursive formula:*

$$\mathrm{len}^{\mathbf{C}}(C) = \begin{cases} 1, & \text{if } C \in \{\top, \bot\} \cup N_C \text{ or } C = \{a_1, a_2, \ldots, a_k\}, \\ \mathrm{len}^{\mathbf{C}}(D) + 1, & \text{if } C = \neg D, \\ \mathrm{len}^{\mathbf{C}}(D) + \mathrm{len}^{\mathbf{C}}(E) + 1, & \text{if } C = D \sqcap E \text{ or } C = D \sqcup E, \\ \mathrm{len}^{\mathbf{C}}(D) + \mathrm{len}^{\mathbf{R}}(r) + 1, & \text{if } C = \exists r.D \text{ or } C = \forall r.D, \\ \mathrm{len}^{\mathbf{C}}(D) + \mathrm{len}^{\mathbf{R}}(s) + 1, & \text{if } C = \exists s.\mathrm{Self}, \\ \mathrm{len}^{\mathbf{C}}(D) + \mathrm{len}^{\mathbf{R}}(s) + 1, & \text{if } C = \geq n\ s.D \text{ or } C = \leq n\ s.D, \end{cases}$$

where $r \in \mathbf{R}$, $s \in \mathbf{R}^s$, $D, E \in \mathbf{C}$, $a_1, a_2, \ldots, a_k \in N_I$, $k \in \mathbb{Z}$, and $n \in \mathbb{N}$.

Using this technique the algorithms can proceed iteratively by revisiting an already searched concept whenever they find it promising based on its heuristic score and gradually asking for longer refinements. The example refinement chain 3.3 demonstrates one such course of action, indeed. The general and substantially simplified procedure is shown in Algorithm 1.

We see that the algorithms allow us not only to restrict the domain (D) in which they search for solutions as hinted in Section 3.2.1, but also to choose a concept (C)

Algorithm 1: General concept-learning algorithm using ρ

Input: a start concept $C \in \{\top\} \cup N_C$, a domain of refinement $D \in \{\top\} \cup N_C$, *termination criteria*, *solution criteria*, *minimum-quality criteria*

Output: a set of solutions

```
1  heuristic ← an algorithm-specific heuristic
2  startNode ← a new node with
3                  parent = none, children = ∅, concept = C, length = len^C(C)
4  searchTree ← a new tree with startNode as the root and only node
5  solutions ← ∅
6  while termination criteria are not satisfied do
7      node ← the currently best node from searchTree according to heuristic
8      remove node from searchTree
9      refinements ← {E ∈ ρ_D(best) | node.length − 1 ≤ len^C(E) ≤ node.length}
10     nodesToAdd ← ∅
11     foreach E ∈ refinements do
12         refNode ← a new node with
13                   parent = node, children = ∅, concept = E, length = len^C(E)
14         refNode.score ← heuristic.evaluate(refNode)
15         if refNode does not satisfy minimum-quality criteria then
16             continue
17         end
18         node.children ← node.children ∪ {refNode}
19         nodesToAdd ← nodesToAdd ∪ {refNode}
20         if refNode satisfies solution criteria then
21             solutions ← solutions ∪ {E}
22         end
23     end
24     node.length ← node.length + 1
25     nodesToAdd ← nodesToAdd ∪ {node}
26     add each node from nodesToAdd to searchTree
27 end
28 return solutions
```

other then the top concept with which they should start. The next input parameter, termination criteria, controls when the learning process is stopped, for instance, after a provided period of time. In Definition 17, we clearly stated what concepts are considered to be solutions, nevertheless, in many real-world situations, it is extremely hard to find perfect descriptions having the there-required properties. For this reason, solution criteria can be changed in order to relax the original definition of a solution. Similarly, less strict minimum-quality criteria enable us to determine a lower bound for the quality of concepts that should be kept in the search tree for further refinement, usually, by enforcing the concepts to cover at least a decent share of the positive examples and/or at most a small number of negative examples. In reality, there is a lot more parameters to tweak and we expand upon the ones whose effects we investigated in our experiments in Section 6.3.

Looking at the actual search process represented by the while loop in Algorithm 1, we have to point out that during the evaluation of the heuristic score at line 14, the coverage of positive and negative examples must be implicitly determined to calculate accuracy, so the aforementioned closed-world reasoner is consulted here as well.

Understanding the learning procedure, we can now discuss the primary characteristics of the algorithms with which we experimented.

OCEL and CELOE

OCEL [17] was the first concept-learning algorithm based on a refinement operator. CELOE [16] was introduced as a variation of OCEL attracted to shorter concepts following the principle of Occam's razor [11], i.e., that simpler descriptions may generalize better since they fit the training conditions more loosely. With OCEL and CELOE, the solution and minimum-quality criteria either coincide (CELOE) or are interrelated (OCEL), and thus only the minimum-quality criteria are configurable. Both of these algorithms were designed to work sequentially in a single thread.

ParCEL and ParCEL-Ex

ParCEL [23] parallelizes the learning process by creating a new worker to execute the instructions at lines 9−26 in Algorithm 1 allowing it to immediately continue and spawn multiple such workers, each responsible for refining a different concept. The maximum number of concurrently running workers is one of the hyper-parameters. Besides that, this algorithm does not support minimum-quality criteria and instead of trying to find true solutions, its workers search for partial definitions, i.e., descriptions that cover none or just a few of the negative examples and at least one positive example. The maximum possible proportion of all negative examples covered by a partial definition is ParCEL's solution criterion. If a concept is marked as a partial definition, it is also

excluded from the set of refinements added to the search tree. Furthermore, rather than returning the set of partial definitions (the solutions set in Algorithm 1) this algorithm merges them together into a single disjunction, which we call a generalization process. While focusing on partial definitions helps to fulfill the condition 3.2 from Definition 17, the subsequent generalization deals with the problem of satisfying the property 3.1 to some extent, as a partial definition alone often covers only a handful of positive examples. Since disjunctions are introduced during generalization, the ρ operator is configured so that it does not produce refinements with disjunctions.

ParCEL-Ex [24] is a modification of ParCEL that requires the partial definitions to cover solely positive examples, so no solution criterion can be specified. Moreover, ParCEL-Ex records counter-partial definitions as well, i.e., the partial definitions of negative examples. Anytime a refinement is processed that is neither a partial nor a counter-partial definition, the algorithm tries to combine this refinement with some of the negated counter-partial definitions into a conjunction in order to create a partial definition. By appending the negations of counter-partial definitions, all the negative examples covered by any of these counter-partial definitions are removed from the set of negative examples that particular refinement covers. Although the combination can be postponed to happen after the entire search ends, the version 2 of ParCEL-Ex that we decided to employ performs it on-the-fly as described above. This method is suitable when a large search tree is expected to be generated, which definitely is our case based on the conducted preliminary tests. Due to the use of negations during combination, the ρ operator is further banned from producing refinement in which negated concepts occur.

4 Data and Representation

Our aim is to characterize malware targeted at the family of Windows operating systems. Therefore, we begin this chapter with a brief overview of the Windows-specific Portable Executable format. Then, we present the SOREL-20M dataset as the source of malicious- and benign-software examples for our experiments. In the end, we discuss the PE Malware Ontology, which we used during data preprocessing that was inevitable for us to be able to supply the learning algorithms with semantic input.

4.1 Portable Executable Files

The Portable Executable (PE) standard [5] defines a common format for executables (EXEs), dynamic link libraries (DLLs), and object files in modern Windows operating systems, starting with Windows NT. The PE format extends the ubiquitous Common Object File Format (COFF) [22], initially introduced to unify the structure of executable, object, and shared-library files in Unix-based operating systems. The basic organization of a PE file is shown in Figure 4.1. We shortly describe its constituent parts below.

The **MS-DOS header** is a legacy header added purely for compatibility with MS-DOS 2.0 and newer, as well as earlier versions of Windows [7]. For this reason, we omit the details about the information it contains.

The **MS-DOS stub** can be found only in PE image files, i.e., EXEs and DLLs. It is an MS-DOS program which outputs a warning message saying that that image cannot be run in DOS mode when a user tries to run a PE image in an MS-DOS environment.

The **PE signature** is a string of four bytes indicating that the file is an image file in the PE format. This field is not present in PE object files.

The **COFF file header** contains details about the class of machines for which the file is targeted, the total number of sections in the file, the size of the optional header that follows, the file's type (e.g., EXE or DLL), and more.

The **optional header** has received its name on the grounds of the fact that it is used exclusively in image files. If present, this header provides us with further information about the file, such as linker version, address of the entry point, image

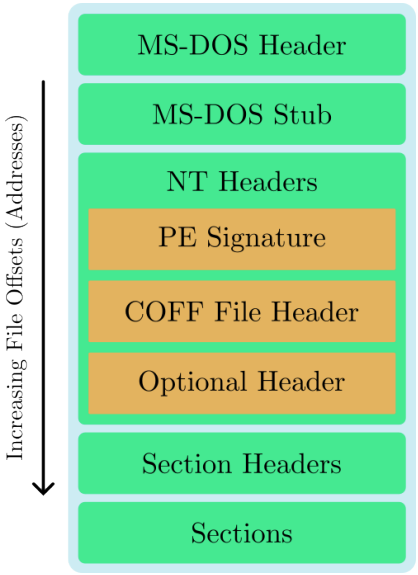


Figure 4.1: Simplified structure of a PE file.

base, data directories etc. Here, we focus on the data located in and referred to by data directories since they are the most relevant for our use case. Each of these directories act as a pointer to a table, a group of related tables, or a string with the actual data. Normally, every table (or group) resides in a separate section. The contents of the most notable tables are summarized in Table 4.1.

The array of **section headers** stores the key characteristics of all the sections in the PE file. For every section, there is a dedicated header bearing information about the section's name, entropy (a degree of randomness in the section's data), the type of the data it contains (code, initialized data, uninitialized data), access permissions (whether it can be read, modified, executed, or shared among multiple processes), etc.

4.2 SOREL-20M

The SOREL-20M dataset [14] is one of the first public large-scale datasets for static malware analysis. This dataset is composed of almost 20 million samples amassed from the beginning of 2017 until the middle of 2019, each representing either a malicious or a benign PE file (EXE or DLL). Alongside PE metadata extracted directly from the binary files, SOREL-20M contains format-agnostic data useful for malware analysis.

For each of the samples with PE metadata available (approx. 98% of all samples), there is a JSON object holding all the information in the headers and data-directory tables of the corresponding PE file, except for the actual certificates, fonts, graphics, and other resources.

The format-agnostic data are provided for each sample. This data include:

- the date and time the file associated with a given sample was first detected,

Table 4.1: Contents of data-directory tables.

Table(s)	Description of Contents
Import Tables	identifiers of imported functions, constants, etc. and the respective DLLs from which they should be imported
Export Tables	public identifiers of exported functions, constants, etc. and their respective addresses
Resource Table	fonts, graphics and other file resources
Certificate Table	certificates indicating that the file is digitally signed
Base Relocation Table	pointers to addresses which need to be relocated if the image is loaded at a base address different form the desired
Debug Table	debug information (symbols, line numbers, strings, etc.)
TLS Table	information on how to initialize and clean up thread-local storage (TLS)[1]

- the total number of times the file was seen in the span of data collection, and
- for malicious binaries only, a list of malware categories into which it may belong, such as adware, spyware or ransomware, as hinted by the results of behavioral analysis.

Besides SOREL-20M, there are very few datasets for static PE-malware analysis, for example, Microsoft Malware Classification Challenge Dataset [19], BODMAS [25], and EMBER [1], the majority of which does not include such comprehensive information about PE files extracted from headers. The EMBER dataset is an exception to this and hence, the previous experiments were carried on data from EMBER. As we were looking for an alternative source to validate the studied approach, we selected SOREL-20M for being largely compatible with EMBER and relatively up to date, which together with its extraordinary size allows for creating a realistic picture of the current state of the PE-malware world with much of the diversity captured.

4.3 PE Malware Ontology

Since our approach uses concept learning requiring semantic input, the data from SOREL-20M need to be first transformed into a knowledge base or, as in our case, into an ontology. We decided to base this ontology on the PE Malware Ontology [21],

[1]Thread-local storage is a special kind of static (automatic) memory local to each thread. The data in this memory are initialized before a thread starts and emptied on its termination.

which defines a class hierarchy and both object- and data-property hierarchies[2] suitable for static malware detection. The main reason behind this choice is the fact that the PE Malware Ontology is an extended and refined version of the ontology used in the proof-of-concept work whose outcomes we aim to validate and that to the best of our knowledge, no other relevant ontology for analysis of malicious software was published at the time of conducting this research.

Figure 4.2 depicts the core structure of the PE Malware Ontology. Below, we describe the class hierarchy and the available object properties by going through all the most-generic classes (direct subclasses of `owl:Thing`).

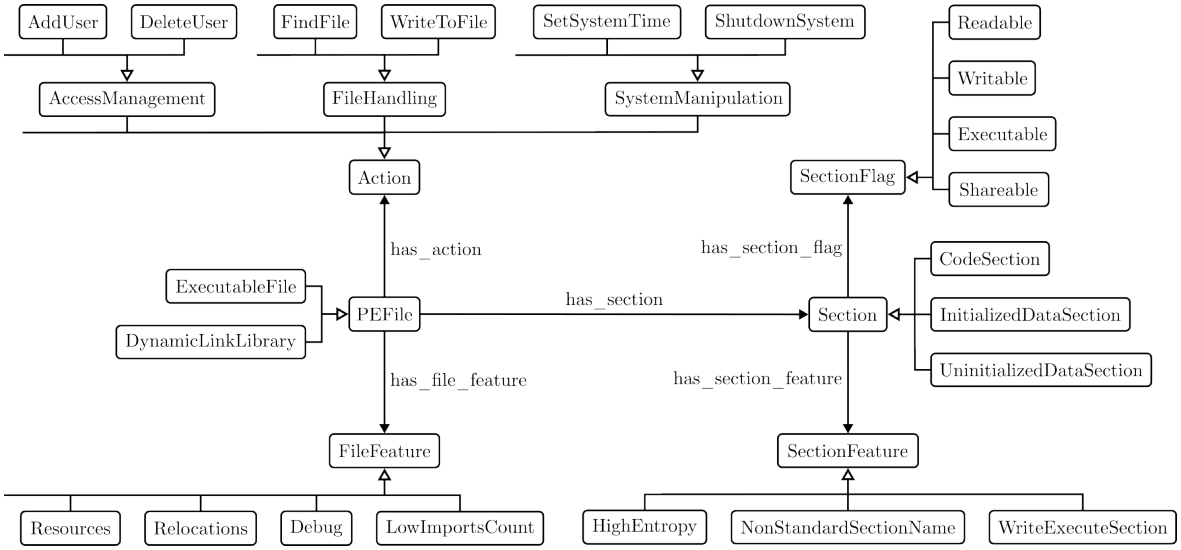


Figure 4.2: Core structure of the PE Malware Ontology. Class names are enclosed in rounded boxes. The fact that C is a direct subclass of D is represented by an empty arrow beginning at C and pointing towards D. A free end of a subclass-relation arrow indicates that there are more subclasses of that particular class which are not shown in the figure. A filled arrow connects the domain of the object property whose name is given above or next to that arrow to the property's range.

`PEFile`. This class is meant to represent all instances of PE files that are in the ontology. We distinguish between EXEs and DLLs, for which there are two separate subclasses of `PEFile` – `ExecutableFile` and `DynamicLinkLibrary`, respectively.

`FileFeature`. The subclasses of the `FileFeature` class symbolize features of PE files that are interesting from the perspective of malware analysis, such as, the presence of a data directory (e.g., `Resources` and `Relocations`), the fact that the file's entry point lies within a non-executable section (`NonexecutableEntryPoint`), or that the file contains COFF debug symbols (`Symbols`). For each of the features a given PE

[2]Here, by *hierarchies*, we mean the orderings induced by $\sqsubseteq$ on the sets of class names, object-property names and data-property names, respectively.

file has, a link between the associated `PEFile` instance and the prototypical instance of the relevant `FileFeature` subclass is created using the `has_file_feature` object property.

`Action.` This class is the root of a three-level hierarchy of action-related classes. At the first level, there is the class `Action`. The second level consists of the direct subclasses of `Action`, which can be conceived of as action categories, e.g., `FileHandling` and `Networking`. Finally, the classes at the third level, i.e., the subclasses of the action-category classes, represent individual actions PE files may take, especially those that are potentially harmful, for example, `FindFile` and `CreateSocket`. The set of actions to which a PE file has access is inferred from the list of imports. To indicate that a PE file can perform a certain action, the corresponding `PEFile` instance and the representative of that particular action are connected through the `has_action` object property.

`Section.` In the PE Malware Ontology, each section of a PE file is also supposed to be modeled as an instance. The role of the `Section` class is to group together all these section instances. Similarly to `PEFile`, `Section` is further divided into three subclasses, `CodeSection`, `InitializedDataSection`, and `UninitializedDataSection`, to differentiate between sections based on the type of data they contain. A PE file (`PEFile` instance) is linked to its sections (`Section` instances) via the `has_section` object property.

`SectionFlag.` Section flags are simply access permissions, for which there are dedicated subclasses of the `SectionFlag` class: `Readable`, `Writable`, `Executable`, and `Shareable`. When a section has a particular flag, it is signified with the help of the `has_section_flag` object property the same way the connections between `PEFile` and `FileFeature` instances are created.

`SectionFeature.` The subclasses of `SectionFeature` enable us to record some purportedly valuable attributes of a section, e.g., high entropy (`HighEntropy`). Once again, the fact that a section has a given feature is signalized by defining a relationship between the corresponding `Section` instance and `SectionFeature` instance (the only instance of the matching `SectionFeature` subclass) through the `has_section_feature` object property.

The object-property hierarchy consists only of the already presented properties, all of which are direct subproperties of `owl:topObjectProperty`.

As mentioned earlier, there are also several data properties declared in the PE Malware Ontology, for instance, to store a section's name, exact value of entropy, or the number of imported/exported symbols for a PE file. However, since we do not

utilize the data properties throughout our experimentation, we will not discuss the data-property hierarchy any further.

5 Corrections and Enhancements

Prior to beginning the experimentation described later in Section 6, we made several modifications to the theoretical definition of the ρ refinement operator, as well as to its implementation and the implementation of the examined learning algorithms in the official distribution of DL-Learner[1], to improve the performance of the learners and resolve a couple of issues that arouse as we were performing preliminary tests. The updated version of DL-Learner which we used in our experiments is accessible through our GitHub repository[2]. In this chapter, we discuss the most pivotal changes.

5.1 Refinement Operator

5.1.1 At-Most Restrictions

After noticing some inexplicable differences in the accuracy of two logically equivalent descriptions reported by OCEL (for the definition of accuracy, see Section 6.2), we found out that the ρ refinement operator was handling at-most restrictions inappropriately. More specifically, we realized that the operation

$$\leq n\ r.D \rightsquigarrow \leq n\ r.E, \tag{5.1}$$

for any concept D, any simple role r, $n \in \mathbb{N}$, and $E \in \rho_{\mathrm{ar}(r)}(D)$, does not produce downward refinements. Moreover, this operation actually generates upward refinements if we assume that $E \sqsubseteq D$, which is expected to be satisfied for a downward refinement operator like ρ. We verify our claims by proving Lemma 1 below.

Lemma 1. *Let $\mathcal{L}$ be a DL language which allows cardinality constraints and $\mathcal{K}$ be a knowledge base in $\mathcal{L}$. Let n be a natural number, $r \in \mathbf{R}^s$ be a role, and D, E be such concepts from $\mathbf{C}$ that $E \sqsubseteq D$. Then, the operation*

$$\leq n\ r.D \rightsquigarrow \leq n\ r.E, \tag{5.2}$$

is an upward refinement operation.

[1] `https://github.com/SmartDataAnalytics/DL-Learner/releases/tag/1.5.0`

[2] `https://github.com/mousetom-sk/DL-Learner/tree/v3`

Proof. Let $\mathcal{L}$ be a DL language which allows qualified number restrictions and $\mathcal{K}$ be a knowledge base in $\mathcal{L}$. In order to prove that (5.2) is an upward refinement operation, it is now sufficient to show that for all concepts $D, E \in \mathbf{C}$ such that $E \sqsubseteq D$, all roles $r \in \mathbf{R}^s$, and any $n \in \mathbb{N}$, the proposition $\leq n\ r.D \sqsubseteq \leq n\ r.E$ holds, i.e., that for every model $\mathcal{I}$ of $\mathcal{K}$ it is also true that

$$\mathcal{I} \models \leq n\ r.D \sqsubseteq \leq n\ r.E. \tag{5.3}$$

Therefore, let D and E be arbitrary concepts in $\mathbf{C}$ with $E \sqsubseteq D$, let $r \in \mathbf{R}^s$ be an arbitrary role, and let n be a natural number. Let us further take an arbitrary model $\mathcal{I}$ of $\mathcal{K}$. As we aim to prove (5.3) which holds if and only if $(\leq n\ r.D)^{\mathcal{I}} \subseteq (\leq n\ r.E)^{\mathcal{I}}$, we first construct the sets $(\leq n\ r.D)^{\mathcal{I}}$ and $(\leq n\ r.E)^{\mathcal{I}}$ based on the definition of the interpretation function:

$$(\leq n\ r.D)^{\mathcal{I}} = \{a \in \Delta^{\mathcal{I}} \mid |\{b \in \Delta^{\mathcal{I}} \mid (a,b) \in r^{\mathcal{I}} \text{ and } b \in D^{\mathcal{I}}\}| \leq n\},$$
$$(\leq n\ r.E)^{\mathcal{I}} = \{a \in \Delta^{\mathcal{I}} \mid |\{b \in \Delta^{\mathcal{I}} \mid (a,b) \in r^{\mathcal{I}} \text{ and } b \in E^{\mathcal{I}}\}| \leq n\}.$$

Taking into account our premise that $E \sqsubseteq D$, we can also conclude that for any $b \in \Delta^{\mathcal{I}}$, it has to be true that if $b \in E^{\mathcal{I}}$, then $b \in D^{\mathcal{I}}$, since $E \sqsubseteq D$ iff $E^{\mathcal{I}} \subseteq D^{\mathcal{I}}$. Consequently, the inclusion

$$\{b \in \Delta^{\mathcal{I}} \mid (a,b) \in r^{\mathcal{I}} \text{ and } b \in E^{\mathcal{I}}\} \subseteq \{b \in \Delta^{\mathcal{I}} \mid (a,b) \in r^{\mathcal{I}} \text{ and } b \in D^{\mathcal{I}}\}$$

and the corresponding inequality

$$|\{b \in \Delta^{\mathcal{I}} \mid (a,b) \in r^{\mathcal{I}} \text{ and } b \in E^{\mathcal{I}}\}| \leq |\{b \in \Delta^{\mathcal{I}} \mid (a,b) \in r^{\mathcal{I}} \text{ and } b \in D^{\mathcal{I}}\}|$$

must be satisfied for any $a \in \Delta^{\mathcal{I}}$. Thus, when an individual $a \in \Delta^{\mathcal{I}}$ belongs to $(\leq n\ r.D)^{\mathcal{I}}$, it has to be an element of $(\leq n\ r.E)^{\mathcal{I}}$ as well, because given such $a \in \Delta^{\mathcal{I}}$, from our previous observations it follows that

$$\begin{aligned} n &\geq |\{b \in \Delta^{\mathcal{I}} \mid (a,b) \in r^{\mathcal{I}} \text{ and } b \in D^{\mathcal{I}}\}| \\ &\geq |\{b \in \Delta^{\mathcal{I}} \mid (a,b) \in r^{\mathcal{I}} \text{ and } b \in E^{\mathcal{I}}\}|. \end{aligned}$$

This brings us to the desired conclusion that $(\leq n\ r.D)^{\mathcal{I}} \subseteq (\leq n\ r.E)^{\mathcal{I}}$. □

Refining at-most restrictions through the operation (5.1) is thus not in accordance with the intention to design ρ as a downward refinement operator. To cope with this problem, we redefined this operation as follows:

$$\leq n\ r.D \rightsquigarrow \leq n\ r.(\neg E), \tag{5.4}$$

i.e., we inverted the direction in which the constrained concept is refined (from downwards to upwards). With the help of Lemma 1, it can be easily shown by the principle

of structural induction for the selected language $\mathcal{L}$ that when we change (5.1) to (5.4) in the definition of ρ, the ρ operator becomes a downward refinement operator. The key is to leverage the fact that the validity of Lemma 1 also implies that the inverse of (5.2) is an operation of specialization.

Of course, this modification also requires the refinement process of at-most restrictions to start from the concept $\leq n\ r.\bot$ instead of $\leq n\ r.\top$ as in Definition 22.

5.1.2 Negations And Universal Quantification

Studying the definition of the ρ refinement operator more carefully, we later detected at least two other places for potential improvement.

Fristly, we prevented the operator from generating the concept-negation refinements having the form $\neg C$, where $C \in \mathbf{C}$ is a super-concept of the domain in which the refinement is performed, for instance, the range of a role. Clearly, this does not decrease the expressive power of the operator because such negations cannot cover any individual from the given domain, yet it helps to reduce the number of nodes in the search tree.

Secondly, we decided to shorten the refinement path from the top concept to the universal quantification concepts of the following kind: $\forall r.\bot$. Originally, a concept $\forall r.C$, with $C \in N_C$ a most-specific concept name, i.e., one for which no $D \in N_C$ exists such that $D \sqsubseteq C$, had to be reached in order to produce $\forall r.\bot$ (see Definition 22). Proceeding in this manner, we could observe a phenomenon that $\forall r.\bot$ is never introduced during refinement, which may happen if, for example, the universal quantifications with most-specific concept names are not promising enough to be further refined. We think that $\forall r.\bot$ should be searched irrespective of whether a universal quantification of r with some most-specific concept name is valuable, as it bears a slightly different meaning, e.g., in our context, $\forall$`has_section_feature`.$\bot$ represents all sections with none of the special features. Contrarily, a concept of the form $\forall r.\bot$ could also quite unnecessarily appear multiple times in a search tree because it was a refinement of every concept $\forall r.C$, where C is most-specific. To address these issues, we redefined the ρ operator in a way that makes $\forall r.\bot$ a direct refinement of $\forall r.\top$.

5.1.3 "Some-Only" Rule

As mentioned in Section 3.2.1, previously, the "some-only" rule allowed the generation of a universal quantification only if put in a conjunction with an existential quantification of the same role.

We extended this rule by permitting a universal quantification in conjunction with an at-least restriction as well since semantic-wise, at-least restrictions are just more constrained existential quantifications. If the "some-only" rule is applied, this relax-

ation thus opens the way for the algorithms to more finer concepts, which may also be more precise descriptions, while preserving the cautiousness discussed in Section 3.2.1.

In addition, we found out that these "some-only" checks were not performed conscientiously in the DL-Learner implementation. For example, the top concept refinements of the form $\forall r.C$ would be generated despite the activated "some-only" rule. As this erroneous behavior was canceling much of the efforts to prevent over-fitting, we felt compelled to correct it.

5.2 Reasoner Component

5.2.1 Cardinality Constraints in the Closed-World Reasoner

Redefining the ρ operator as suggested in Section 5.1.1 did not settle all the problems with accuracy. We discovered this while comparing the output of OCEL and CELOE, which disagreed in the accuracy of the same concepts containing at-most restrictions. After further investigation, we came to the conclusion that the problem was caused by the employed closed-world reasoner, which was not always answering the queries asking whether a certain individual belongs to the extension of a given concept with cardinality constraints correctly due to simple programmatic errors.

This problem was fixed by rectifying the closed-world reasoner.

5.2.2 Concurrent Closed-World Reasoner

Experimenting with the parallel algorithms, we noticed that when opting for the explicit method of concept-disjointness checks, some worker threads terminated due to an unexpected concurrent modification of the closed-world reasoner's internal data structures. We deduced that this was a consequence the fact that all workers were consulting the same reasoner instance.

To resolve this issue, we first tried creating one standalone copy of the closed-world reasoner for each worker but this solution was causing a bottleneck in the early phases of learning since each copy had to be initialized separately. As a remedy to this problem, we thus eventually implemented a concurrent closed-world reasoner maintaining a pool of the original (*base*) closed-world reasoners sharing the read-only internals whose construction demands the majority of time during initialization.

5.3 Heuristic in OCEL

As mentioned in Section 3.2.2, OCEL uses a heuristic to determine which parts of the search space to explore first. The role of one undocumented component of this

heuristic function was to encourage the algorithm to refine concepts containing $\forall r.\top$ or $\leq n\ r.\top$ by artificially bettering their heuristic score. Such concepts might be interesting for further refinement due the fact that $\forall r.\top$ is logically equivalent to $\top$ and that $\leq n\ r.\top$ is usually too broad, which means that on their own, they provide no or only little information. In addition, $\forall r.\top$ and $\leq n\ r.\top$ were the starting points on the paths to all universal-quantification and at-most-restriction refinements for the role r, respectively, so, naturally, there is a greater potential to replace these less informative expressions with some more specific ones through refinement. A clear advantage of their refinement is that we can gain the extra expressivity without much increase in the length and complexity of the whole concept.

However, the implementation of this component of the heuristic was not deterministic and the adjustment to the final score did not grow with each occurrence of any of the above two expressions in the evaluated concept, which we consider crucial for a proper assessment of a concept's capacity to specialize. Moreover, after the changes presented in Section 5.1.1, it makes more sense to favor $\leq n\ r.\bot$ rather than $\leq n\ r.\top$, as it is the former that now begins every refinement chain of at-most restrictions on r.

Hence, we altered this part of OCEL's heuristic to ensure that its output is deterministic, that the final score correction accumulates with the increasing number of appearances of the expressions in question, and that it supports the refinement of concepts containing $\leq n\ r.\bot$ instead of those with $\leq n\ r.\top$.

5.4 ParCEL and ParCEL-Ex

5.4.1 Same-Length Refinements

When refining a concept expression via the ρ operator, both parallel algorithms, or at least their implementations in DL-Learner, required the direct refinements to be longer than the expression from which they resulted. Apparently, this was not the intention of the authors, since it deprives the algorithms of many valuable concepts. For instance, the descriptions of the form $\forall r.C$, with $C \neq \top$, were completely unreachable with this restriction imposed because the only path to them leads through $\forall r.\top$, of which they are direct refinements but with no greater length.

We resolved this issue by allowing ParCEL and ParCEL-Ex to accept also those direct refinements whose length is equal to the length of the refined concept.

5.4.2 Accuracy Calculation in ParCEL

The process of calculating the accuracy of a concept expression for the purposes of ParCEL's heuristic and determining if the concept is a partial definition was originally

taking into account the coverage of the hitherto found partial definitions as well. More specifically, the positive examples covered by the already learned partial definitions were excluded from the set of positive examples considered during the calculation.

We suppose that the rationale behind this change to the standard definition of accuracy was to promote those expressions that can potentially contribute to the overall coverage of all positive examples the most, which is a reasonable goal to strive for. Nevertheless, the way this objective was attained brought a bit of uncertainty into the selection of concepts for refinement, possibly developing a preference for otherwise less suitable concepts, as under these circumstances, the accuracy reflects not only the actual quality of a concept from the global viewpoint, but also when it was evaluated. This would require us to recompute the accuracy of each searched concept every time a new partial definition is found to always have an up-to-date measurement of their aptitude for refinement. Despite that, the calculation was performed only at the time of the discovery of a given concept by virtue of being a resource-intensive task.

Therefore, we decided to re-implement the procedure of accuracy calculation so that the basic definition, provided in Section 6.2, is used. In order to draw the algorithm's attention to the promising concepts, we added the following rules. When a node of the search tree is revisited to seek for new refinements, we first check whether the concept it represents still covers at least one positive example that is not an instance of any of the already found partial definitions. If no such example exists, the node is removed from the search tree and the corresponding concept is never refined again. Likewise, a description now needs to cover strictly more of the yet uncovered positive examples than uncovered negative examples in order to be deemed a partial definition.

5.4.3 Acceleration of Accuracy Calculation

Previously, the accuracy of a concept was always calculated from scratch in ParCEL and ParCEL-Ex. Inspired by the implementation of OCEL, we sped up this calculation by keeping and exploiting the information on which positive and negative examples are covered by the searched concept expressions. These data are stored in the respective nodes of the search tree. To compute the accuracy of a given concept's refinement, it is then sufficient to determine what subsets of positive and negative examples covered by the original concept are also covered by that particular refinement because we know that ρ is a downward refinement operator, and thus each refinement can cover only the examples covered by its direct predecessor. In combination with other enhancements, this modification enabled the parallel algorithms to triple the number of concept expressions they were able to search per unit of time, which also forced us to devise a compressed representation of the sets of the covered examples to fit the entire search tree into the available memory.

Motivated by this rise in the number of searched expressions, we tried to improve the performance of CELOE in a similar manner. However, our experiments showed that CELOE can barely benefit from the accelerated accuracy calculation and that the necessary compression and decompression sometimes even outweighs the effects of the more efficient computation of accuracy in terms of time consumption. We assume that this stems from the fact that CELOE endeavors to find a single description as a whole, whereas the parallel algorithms search for partial definitions that often cover significantly less examples so much more time is saved with this optimization of accuracy calculation. OCEL, which uses this technique out of the box, copes with the problem of the enlarged search tree by means of periodic branch pruning in place of compression. Since this would be a quite fundamental intervention in the implementation of CELOE, we eventually decided not to alter the way CELOE computes accuracy.

5.4.4 Search Tree Organization

Although the change we discuss in this section is a bit more technical when compared to the ones mentioned above, we find it important to present it here due to its surprisingly immense impact despite its simplicity.

The problem we are about to describe consisted in how the search tree was accessed and maintained by ParCEL and ParCEL-Ex. Both investigated parallel algorithms use Java's `ConcurrentSkipListSet` to store all the nodes of the search tree in the order determined by the values assigned to the concepts they embody by their heuristics. Previously, each time ParCEL or ParCEL-Ex was ready to refine a concept expression it polled the last element of this set – the node with the currently best concept according to the heuristic.

In our efforts to make the search tree representation more compact, induced by the modification proposed in Section 5.4.3, we noticed that the official documentation for `ConcurrentSkipListSet` emphasizes that ascending iterators over this data structure allows faster access than the descending iterators. Therefore, we closely examined the difference in the time required to empty a set of this type by repeatedly polling the last element versus the first. As retrieving the first element proved to be on average almost two times faster during our tests, we reversed the order of nodes in the search tree and instructed the algorithms to poll the first element instead of the last. It also has to be noted that storing the elements in the reversed order does not affect the insertion time.

6 Experiments

In this chapter, we describe our experimentation process in detail. We expand upon the conditions under which we evaluated the performance of the learning algorithms, including the input data we prepared, what methodology we used for performance evaluation, and what configurations of the algorithms were tested.

6.1 Test Cases and Ontology Preparation

We performed our experiments in three settings differing in the nature of input data.

6.1.1 Test Case 1 – Mixed Data

In the first test case, we studied the capabilities of the learning algorithms in the most general situation where no prior categorization of samples takes place.

For the purposes of this test case, we prepared four mutually disjoint datasets. Each dataset consists of 1000 samples evenly split between malicious and benign, as well as EXEs and DLLs (see Table 6.1). These samples were selected from the entire SOREL-20M dataset at random. The decision on dataset size was largely affected by the time and memory requirements of the algorithms and the resources we had at our disposal. Augmenting the PE Malware Ontology by adding the knowledge about the extracted samples, we then created the ontology that was later provided to the learning algorithms together with the information on which samples should be treated as positive and which as negative – in our context, malware and benign-software samples, respectively.

Table 6.1: Number of malicious/benign samples by file type in the mixed datasets.

File Type	Malicious	Benign
EXE	250	250
DLL	250	250

In reality, the number of benign-software samples exceeds the number of malware samples by an order of magnitude, usually, 80 benign binaries are observed per each piece of malware that appears [18]. In spite of that, we were encouraged to balance the datasets in this regard because with a dataset of 1000 samples, ensuring even a 10% share of malware does not have to sufficiently motivate the algorithms to learn as in such a case, it is still enough to precisely cover a few malicious samples to achieve a favorable heuristic score due to high accuracy (we define accuracy in Section 6.2). Moreover, our goal is to also find descriptions that can discriminate malware from benign software in general, which is hardly possible when we give the algorithms only a limited amount of information on features of malicious binaries, lacking diversity.

Likewise, we did not want the algorithms to learn that the chance a file of a certain type is harmful is too low to bother describing it. To this end, the one-to-one ratio of EXEs to DLLs among both the malicious and benign samples was established, which we think was particularly important for the objectivity of the obtained results. We hold this opinion because of our experience with undesirable results in an unbalanced setting. Initially, we constructed one dataset of 500 malicious and 500 benign samples randomly drawn from SOREL-20M without any restrictions on the proportion of EXEs to DLLs. Inspecting this dataset, we discovered that only about 20% of all samples were DLLs and that malicious DLLs barely accounted for 10% of all DLLs in this dataset. The non-parallel algorithms were thus able to achieve high accuracy, recall, and F1 score (see Dafinition 24 in Section 6.2) but merely in the wake of the scarcity of malicious DLLs and the fact that the expressions they found were covering mostly EXEs.

Finally, it has to be noted that we also added a number of axioms to the resulting ontology to explicitly express the obvious fact that each class in the PE Malware Ontology is disjoint from every other class therein, with the exception of the subclasses of the class `Section`. We refrained from asserting disjointness here because we noticed that it is possible to attach multiple content-type labels to a PE-file section. As this is unusual, it may be an indication of malware as well. We further clarify why we introduced these class-disjointness axioms in Section 6.3.

6.1.2 Test Cases 2 and 3 – Separation of Concerns

The test cases number 2 and 3 are tightly related to one another because in the former, we let the algorithms to focus exclusively on EXE binaries while in the latter, they were only confronted with DLLs. We expected that learning the attributes of malicious EXEs and DLLs separately could yield better results as finer descriptions might be found. Furthermore, having to characterize both at once may overwhelm especially the non-parallel algorithms, i.e., OCEL and CELOE, which try to find a description

as a whole rather than producing it in a distributed manner, for example, through partial definitions. OCEL and CELOE thus almost necessarily have to search for disjunctions with one disjunct per file type but such concepts are inherently longer and less attractive for the learners. Additionally, since we are able to determine whether a file is an EXE or DLL directly by inspecting the file's metadata, we consider first finding out a file's type and then using a type-specific classifier a feasible approach to malware detection.

For each of these two test cases, we again created four non-overlapping datasets comprising 1000 samples. As mentioned above, each dataset for the second test case contains only the samples representing EXE files, of which 500 are malicious and 500 benign. Similarly, each dataset prepared for the third test case consists of 500 malicious and 500 benign DLL samples. The calibration and the first validation dataset for both of these test cases were constructed from the samples present in the datasets for the first test case. The other validation datasets are composed of randomly selected samples from the remainder of SOREL-20M.

The final ontologies provided to the learning algorithms were created as previously, i.e., by combining the PE Malware Ontology with sample data and class-disjointness axioms.

6.2 Evaluation

In each test case, we first investigated various configurations of the learning algorithms to optimize some hyper-parameter values as we describe in Section 6.3. Among the four datasets prepared for the corresponding test case, we reserved one (*calibration*) dataset for this stage, on which the performance of all configurations was evaluated. Then, we selected the overall best configurations of each algorithm and evaluated their performance on the other three (*validation*) datasets for that particular case. The purpose of this further validation was to counteract the smaller dataset volume and verify that similar results can be produced on broader range of data.

For the evaluation of each configuration, whether during the optimization or the subsequent validation stage, we employed k-fold stratified cross-validation [10]: *With the dataset partitioned into k subsets (folds) of equal size preserving the original proportions of selected classes, k experiments are performed, each time leaving a different fold aside as the test set while using the remaining $k-1$ folds as the training set.* We chose $k = 5$ and our stratification resided in retaining the ratio of malicious samples to benign samples, as well as EXEs to DLLs within these categories in the first test case.

Experiments were run on two identical Ubuntu 20.04-1 machines, both equipped with a 12-core AMD Ryzen 9 5900X CPU and 64 GB of RAM. The duration of each

experiment was limited by a specified amount of time, which was also the only termination criterion used apart from finding a perfect solution (see Definition 17). For OCEL and CELOE configurations, we fixed this learning timeout to 2 hours of user time, i.e., the actual time the CPU spent executing the corresponding process. This usually corresponds to approx. 2 real-time hours as both OCEL and CELOE are intended to work in a single execution thread. For ParCEL and ParCEL-Ex configurations, the time limit was set to 24 hours of user time in the first test case and then reduced to 12 hours in the second and third test case on the grounds of our observations described in Section ... Since we allowed a maximum of 12 concurrent workers to be utilized by the parallel algorithms (see Section 6.3) and conducted the experiments on 12-core machines, 24 hours of user time should amount to roughly 2 hours of real time here. The initial time limits were determined after some preparatory tests which showed that the above values are generous enough for the algorithms to unleash their potential. We decided to keep the time measurement relative to the utilization of CPU to ensure fairness regardless of the system load.

For evaluation purposes, we used the standard metrics [12], i.e., *accuracy*, *precision*, *recall*, *specificity*, *false-positive (FP) rate*, *false-negative (FN) rate*, and *F1 score*, which can be defined in the context of concept learning as follows.

Definition 24 (Evaluation Metrics)**.** *Let $\mathcal{L}$ be a DL language and $\mathcal{K}$ be a knowledge base in $\mathcal{L}$. Let $E^+ \subseteq N_I$ be a set of positive examples and $E^- \subseteq N_I$ be a set of negative examples, with $E^+ \cap E^- = \emptyset$. Let $C \in \mathbf{C}$ be a concept expression, then*

$$\begin{aligned}
\textbf{\textit{accuracy}}(C) &= \frac{|TP| + |TN|}{|E^+| + |E^-|}, & \textbf{\textit{FP rate}}(C) &= \frac{|FP|}{|E^-|} = 1 - \textit{specificity}(C),\\
\textbf{\textit{precision}}(C) &= \frac{|TP|}{|TP| + |FP|}, & \textbf{\textit{FN rate}}(C) &= \frac{|FN|}{|E^+|} = 1 - \textit{recall}(C),\\
\textbf{\textit{recall}}(C) &= \frac{|TP|}{|E^+|}, & \textbf{\textit{F1 score}}(C) &= 2 \cdot \frac{\textit{precision}(C) \cdot \textit{recall}(C)}{\textit{precision}(C) + \textit{recall}(C)},\\
\textbf{\textit{specificity}}(C) &= \frac{|TN|}{|E^-|}, &&
\end{aligned}$$

where

$$\begin{aligned}
TP &= \{a \in E^+ \mid \mathcal{K} \models C(a)\}, & FP &= \{a \in E^- \mid \mathcal{K} \models C(a)\},\\
TN &= \{a \in E^- \mid \mathcal{K} \models \neg C(a)\}, & FN &= \{a \in E^+ \mid \mathcal{K} \models \neg C(a)\}.
\end{aligned}$$

From the success measures in Definition 24, we decided to pay special attention to FP rate and precision due to our preference for strong descriptions of malware even at the cost of, e.g., lower recall, meaning that only a certain category of malicious software is covered. This problem may be simply mitigated by repeating the learning process

multiple times in succession while always removing the already covered instances of malware, or even better, labeling them as negative examples. All the learned descriptions can be then combined into a single disjunction uniting the sets of malware samples they cover separately.

6.3 Tested Configurations

We identified several hyper-parameters of the studied learning algorithms that we presumed can potentially impact their performance. These parameters were subject to optimization throughout our experimentation in each test case. Before we delve into the details, we present the modifications we made to the default configuration of the algorithms which were applied in all experiments.

Firstly, we instructed the algorithms to seek refinements exclusively in the domain of `PEFile`, i.e., to consider only those class expressions which are subsumed by `PEFile` when searching for solutions. This means that the expressions which cannot characterize any (malicious) PE file, such as $\texttt{CodeSection} \sqcap \exists \texttt{has\_section\_flag}.\texttt{Readable}$, do not appear in the search space on their own, which helps the algorithms to learn more efficiently. The default for both the domain of refinement and the start class is `owl:Thing`, however, we did not alter the latter to allow the generation of disjunctions at the top-most level. Next, we changed the process of determining whether two classes are mutually disjoint form being instance based to being based on explicit assertions, since we think that class disjointness should not be deduced from the instance data, but rather from the general knowledge about the classes, hence also the need for the supplementary class-disjointness axioms. To make the learning easier, we forbade the use of data properties as mentioned earlier. For the purposes of instance checking, OCEL and CELOE were provided with the corrected closed-world reasoner, while ParCEL and ParCEL-Ex were requested to consult the new concurrent version of this reasoner. We set up both reasoners to use the standard semantics of universal quantification because by default, $\forall r.C$ is interpreted as $\forall r.C \sqcap \exists r.C$, resulting in an implicit application of the "some-only" rule, with which we decided to experiment during the hyper-parameter optimization. In order to spare computing time, we disabled the simplification of class expressions in the set of solutions based on logical equivalence, for example, $(C \sqcup D) \sqcap E$, where $C \sqsubseteq D$, would be transformed to $D \sqcap E$. This is normally performed by CELOE to output shorter and less complicated descriptions. Lastly, ParCEL and ParCEL-Ex were limited to 12 concurrent workers to fully exploit the benefits of parallelization w.r.t. the available resources (the number of CPU cores on the evaluation machines described in Section 6.2). With regard to this choice, we configured the concurrent closed-world reasoner to maintain a pool of 13 base reasoners, so that there is always

at least one idle reasoner ready to handle incoming queries.

Now, we look at the hyper-parameters with which we were experimenting.

Noise. Noise represents the minimum-quality criteria for OCEL and CELOE, the solution criteria for ParCEL, and the termination criteria for ParCEL-Ex. The respective roles of these criteria were discussed in Section 3.2.2. In CELOE, noise directly translates to an upper bound for the percentage of the false-negative rate of a class expression that should be considered for further refinement, i.e., for such expressions, following must hold: $FN\ rate < noise/100$. OCEL's formula is more intricate but can be simplified to $FN\ rate < 2 \cdot noise/100$ in our case. For ParCEL, the noise value is the maximum percentage of the FP rate of a partial definition, so any expression has to satisfy the inequality $FP\ rate \leq noise/100$ to be accepted as a partial definition. ParCEL-Ex treats noise as the FP-rate percentage of the entire disjunction of partial definitions which is enough to reach to terminate the learning process. Since we decided to stop learning after a specified a period of time, we did not optimize the value of noise for ParCEL-Ex, leaving it always at 0. Tweaking the value of noise, our goal was to optimize the level of FP rate in combination with other metrics.

Use of Negations. By means of this parameter, we can control the use of negations in the generated class expressions. Disallowing negations not only narrows the search space, but we hypothesized that it may also encourage the algorithms to find expressions of higher quality, i.e., $\exists \texttt{has\_action}.(\texttt{AddUser} \sqcap \texttt{Encrypt})$ rather than $\exists \texttt{has\_action}.(\neg \texttt{DeleteUser})$, and reduce the number of negative test examples classified as positive, which the negated expressions are more likely to cover. Even though ParCEL-Ex was originally designed to work with negations disabled due to the introduction of counter-partial definitions, we studied the effects of enabling them on the performance of ParCEL-Ex as well on account of the fact that ParCEL and ParCEL-Ex use the same heuristic, which is optimized for finding partial definitions only.

"Some-Only" Universal Quantification. In spite of the fact that the default behavior is to perform "some-only" checks, which could prevent the algorithms from over-fitting, we theorized that deactivating this rule might give the algorithms access to valuable parts of the search space. For example, it might be the case that an expression of the form $\forall r.C$ alone generalizes well, i.e, that the individuals which do not have an r-filler are mostly or exclusively positive examples. Hence, by forcing such an expression to appear only in conjunction with an existential quantification or an at-least restriction on r, we loose the ability to explore loads of promising descriptions.

Restrictions on Cardinality Constraints. We experimented with two parameters restricting the construction of cardinality constraints – *allowedRolesInCardinalityRestrictions* and *maxCardinalityLimit*. With the former, we can define the set of properties that are allowed to be used to produce cardinality constraints. The latter

represents an upper limit for the quantity in at-most restrictions. Our goal was to guide the search towards more explicit and simpler descriptions via these two parameters, respectively.

For example, the expression ≥ 3 `has_action.FileHandling` is a relatively vague characterization of a PE-file category and this applies to a lot of other cardinality-constraint expressions on any of the available object properties in the PE Malware Ontology, except for `has_section`, which we view as a relation where cardinality naturally arises, since a PE file is often composed of multiple sections with different contents. Therefore, we investigated whether allowing only the `has_section` object property to be constrained via cardinality constraints (this option is hereafter also referred to as HS) enables the algorithms to discover more specific descriptions, which would also justify our conjecture that cardinality constraints on other properties merely distract the learners.

Similarly, we assumed that for an at-most restriction of the form $\leq n\ r.C$, where $n \geq 2$, to be a decent description, C needs to be fairly complex. Moreover, at-most restrictions in general rather prohibit individuals from having too many properties, but the PE Malware Ontology was clearly developed to characterize malware by listing the features that make malware distinct from benign software. Nonetheless, expressions like $\leq 1\ r.(C \sqcup D) \sqcap \exists r.(C \sqcup D)$ concisely mimic the standard logical exclusive or, which may be convenient in some situations. To reconcile these two aspects of at-most restrictions in our efforts to focus on simpler expressions, we tried limiting the maximum quantity in at-most restrictions to 1 (hereafter also abbreviated as M1).

However, neither of these two parameters was included in the official distribution of DL-Learner and the support for them was implemented by us, taking an inspiration from the existing parameter for limiting the quantity in all cardinality constraints.

In order to test as much reasonable combinations of values for the aforementioned hyper-parameters, we split the experimentation in each test case into three phases. At the end of each phase, the best configurations of each algorithm were chosen to proceed to the next phase.

In the first phase, we optimized the selection of noise. For OCEL, we started with the value of 5, which we were gradually increasing by 5 until we stopped at the noise of 25. This translates to the range of admissible minimum recall on the training set from 0.9 to 0.5 (see the part describing the noise parameter). To align CELOE configurations with those of OCEL in terms of the tested values of minimum recall, the first, the step, and the last value for CELOE were 10, 10, and 50, respectively. We did not try to set the minimum recall above 0.9 or below 0.5 as we deemed such a high recall to be a too strict criterion and such a low recall to be unacceptable. With ParCEL, we tried the noise values of $0, 1, 2$, and 3 since for ParCEL, noise is just an upper bound for the FP

rate of a single partial definition on the training set, meaning that the final FP rate can easily grow when they are all combined. Although we aimed to minimize FP rate, we experimented with higher values of noise to allow for a more loose representation of training data and perhaps better generalization.

The second phase was devoted to investing the effects of disabling negations and "some-only" checks on the overall performance. Both were enabled in the first phase and here, we tested all the remaining combinations.

In the last phase, we used the two newly-added parameters to control the generation of cardinality constraints through either the HS or the M1 rule. These restrictions on cardinality constraints were not applied in the previous phases.

We decided to optimize the selected hyper-parameters in this order based on our experience with what interdependencies there are between the individual parameters from the perspective final performance, showing that the value of noise is the most crucial.

7 Results

After describing our experimentation process, evaluation methodology, and the outcomes we expected, in this chapter, we share the results we obtained throughout the optimization and validation in each of the three arranged test cases discussed in Section 6.1.

All tables of results in this chapter adhere to the following format. In the columns for evaluation metrics, we give the final scores represented as the *average value* $\pm$ *the standard deviation* across all 5 iterations of the cross-validation. For test data, the tables contain the score for each of the success measures from Definition 24, except for the FN rate and specificity that can be easily computed from recall and FP rate, respectively. For training data, we present only the final accuracy as the value for this metric is one of the main components of the heuristics guiding the studied algorithms. The Configuration column describes to what configuration of which algorithm the scores in a particular raw belong in the form algorithm[noise/use of negations/"some-only" checks/restrictions on cardinality constraints]. The symbols T, F, and "-" in a configuration description mean enabled, disabled, and none, respectively.

The provided accuracy-against-time and FP-rate-against-time plots display the average score of the most accurate description on the training set discovered until a given point in the user-time space. For brevity, we do not present the learning progression of every configuration in the main text, but all are available in Appendix ...

7.1 Test Case 1

In this section, we first go through the respective phases of optimization. Then, we summarize the validation results and present some examples of learned descriptions.

7.1.1 Noise Optimization

OCEL and CELOE

The evaluation results of the final descriptions found by OCEL and CELOE configurations in the first phase of optimization are displayed in Table 7.1. We see that

in general, it holds that the higher the noise, the lower the FP rate, which we also expected, since allowing the algorithms to cover less positive examples, i.e., to reach lower recall, enables them to find more specific descriptions. Moreover, FP rate was decreasing at a slightly faster pace than recall on average, leading to a considerable improvement in precision, although not enough to offset the lowering recall (see the F1 scores in Table 7.1). Interestingly, setting the value of noise to 50 caused CELOE to behave erratically, producing results comparable to those achieved by CELOE at the noise of 30 but with much higher variance. This may indicate that there is an inflection point between the values 40 and 50 beyond which the noise becomes irrelevant for CELOE in these conditions, forcing it to determine minimum-quality criteria by itself. Looking at the final results, we can also conclude that OCEL and CELOE configurations with the matching lower bounds for recall seemed to perform very similarly, of course, with the exception of CELOE[50/T/T/-].

The learning progression of OCEL and CELOE configurations in terms of accuracy is outlined in Figures 7.1 and 7.2, respectively. Examining these figures, we can notice that OCEL configurations were consistently getting towards more accurate descriptions, whereas CELOE configurations reached their maximum much quicker. We assume that this is an implication of CELOE's endeavor to search for the shortest descriptions possible, making it hard to achieve major improvements in the later phases of learning, when more complex expressions are generated.

Since we decided to put extra emphasis on FP rate, we analogously studied how the algorithms performed with respect to this metric. Figures 7.3 and 7.4 show that the progression of FP rate on the test set copied the progression on the training set, which signalizes that both OCEL and CELOE where able to eliminate negative examples equally well in the training and the unseen test data. However, Figure 7.3 also reveals that the OCEL[25/T/T/-] configuration over-fit, reaching the lowest FP rate quite early and then almost doubling it until the end.

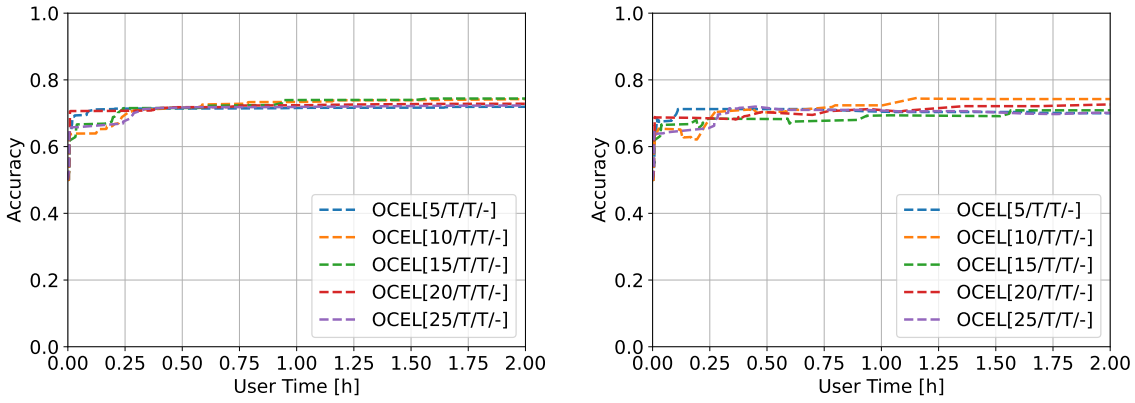
Prioritizing low FP rate, OCEL[20/T/T/-], OCEL[25/T/T/-], CELOE[30/T/T/-], and CELOE[40/T/T/-] were chosen to proceed to the next phase. We decided to pick two configurations of each algorithm to ascertain that a configuration with a better noise setting, performance-wise, cannot be later surpassed by a configuration performing worse at first.

ParCEL and ParCEL-Ex

Optimizing the noise value for ParCEL, we likewise obtained results that are in accordance with our intuition. As we see in Table 7.2, any increase in noise negatively affected ParCEL's performance in every aspect apart from recall, so no improvement in generalization occurred. The lowering precision and the fact that the test accuracy

Table 7.1: Test Case 1: OCEL and CELOE Noise Optimization.

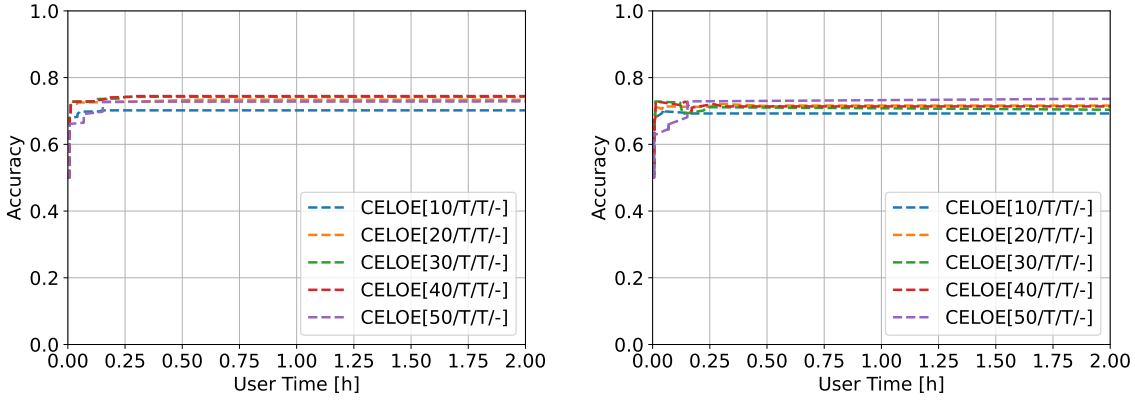
Configuration	Training	Test				
	Accuracy	Accuracy	Precision	Recall	FP Rate	F1 Score
OCEL[5/T/T/-]	0.72 ± 0.01	0.70 ± 0.04	0.64 ± 0.03	0.91 ± 0.04	0.51 ± 0.04	0.75 ± 0.03
OCEL[10/T/T/-]	0.74 ± 0.00	0.75 ± 0.03	0.72 ± 0.03	0.82 ± 0.03	0.32 ± 0.05	0.76 ± 0.03
OCEL[15/T/T/-]	0.74 ± 0.01	0.71 ± 0.02	0.73 ± 0.02	0.68 ± 0.03	0.25 ± 0.03	0.70 ± 0.02
OCEL[20/T/T/-]	0.73 ± 0.01	0.72 ± 0.02	0.76 ± 0.04	0.66 ± 0.07	0.21 ± 0.06	0.70 ± 0.04
OCEL[25/T/T/-]	0.72 ± 0.00	0.70 ± 0.01	0.78 ± 0.03	0.56 ± 0.06	0.16 ± 0.04	0.65 ± 0.03
CELOE[10/T/T/-]	0.70 ± 0.01	0.69 ± 0.03	0.63 ± 0.02	0.94 ± 0.04	0.55 ± 0.02	0.75 ± 0.03
CELOE[20/T/T/-]	0.73 ± 0.01	0.72 ± 0.04	0.69 ± 0.03	0.80 ± 0.05	0.35 ± 0.03	0.74 ± 0.04
CELOE[30/T/T/-]	0.74 ± 0.00	0.71 ± 0.03	0.71 ± 0.03	0.72 ± 0.07	0.29 ± 0.05	0.71 ± 0.04
CELOE[40/T/T/-]	0.74 ± 0.00	0.71 ± 0.02	0.76 ± 0.03	0.63 ± 0.08	0.20 ± 0.06	0.69 ± 0.03
CELOE[50/T/T/-]	0.73 ± 0.01	0.73 ± 0.04	0.74 ± 0.06	0.74 ± 0.10	0.27 ± 0.10	0.73 ± 0.05



(a) Training accuracy.

(b) Test accuracy.

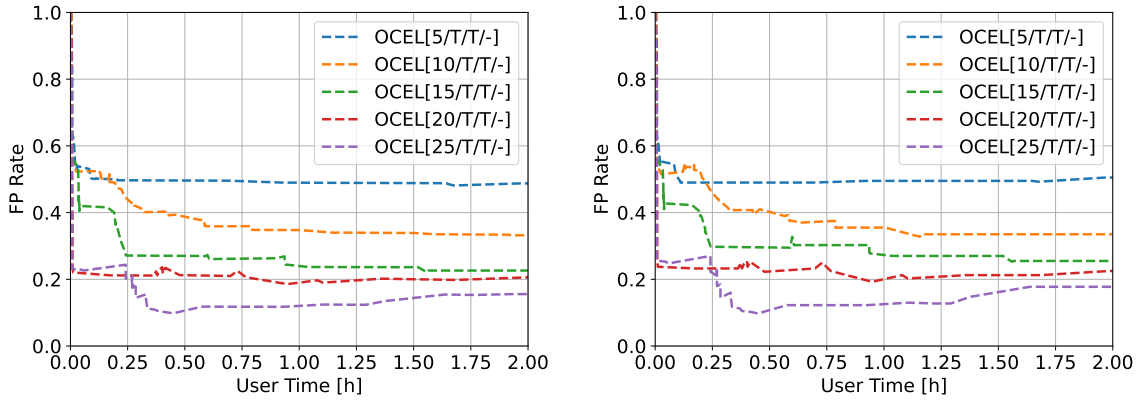
Figure 7.1: Test case 1: OCEL noise optimization – Accuracy progression.



(a) Training accuracy.

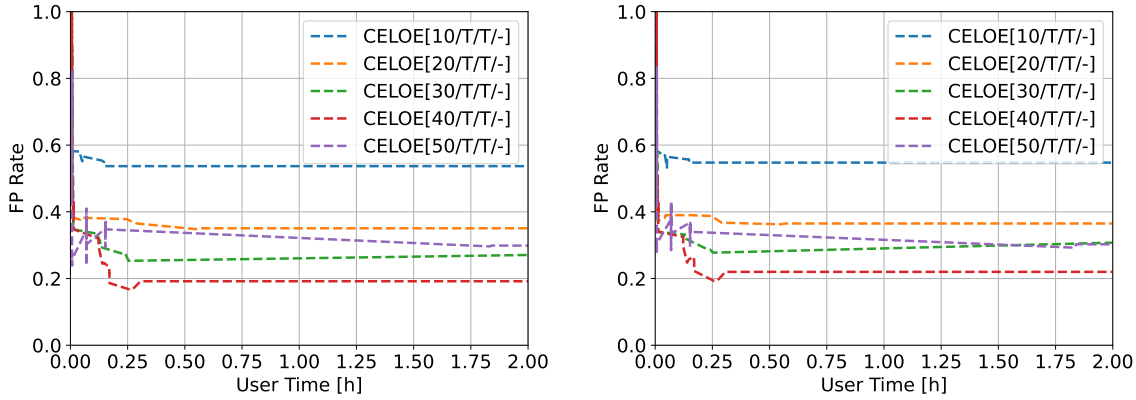
(b) Test accuracy.

Figure 7.2: Test case 1: CELOE noise optimization – Accuracy progression.



(a) Training FP rate. (b) Test FP rate.

Figure 7.3: Test case 1: OCEL noise optimization – FP rate progression.



(a) Training FP rate. (b) Test FP rate.

Figure 7.4: Test case 1: CELOE noise optimization – FP Rate progression.

Table 7.2: Test Case 1: ParCEL and ParCEL-Ex Noise Optimization.

Configuration	Training	Test				
	Accuracy	Accuracy	Precision	Recall	FP Rate	F1 Score
ParCEL[0/T/T/-]	0.90 ± 0.01	0.79 ± 0.02	0.85 ± 0.02	0.70 ± 0.04	0.12 ± 0.02	0.77 ± 0.03
ParCEL[1/T/T/-]	0.86 ± 0.01	0.79 ± 0.03	0.80 ± 0.05	0.77 ± 0.03	0.19 ± 0.05	0.79 ± 0.02
ParCEL[2/T/T/-]	0.84 ± 0.01	0.79 ± 0.02	0.79 ± 0.03	0.79 ± 0.03	0.21 ± 0.03	0.79 ± 0.02
ParCEL[3/T/T/-]	0.83 ± 0.01	0.78 ± 0.04	0.76 ± 0.05	0.82 ± 0.03	0.27 ± 0.07	0.79 ± 0.03
ParCEL-Ex[0/T/T/-]	0.89 ± 0.02	0.78 ± 0.02	0.83 ± 0.02	0.71 ± 0.03	0.14 ± 0.02	0.76 ± 0.02

was oscillating around 0.79 among all ParCEL configurations suggest that progressing from the noise value of 0 up to 3, the FP rate was growing relatively faster than recall but by roughly the same absolute amount. The average test FP rate of 0.12 for ParCEL[0/T/T/-] might indicate that the training sets were not representative enough of the entire dataset, since no false positives were allowed during learning here.

Figures 7.5 and 7.6 show that all ParCEL configurations made the most tangible progress in the first user-time hour of training, that is, the first five minutes in the real time. This means that even ParCEL struggled with detecting the common features of malicious software after covering the most obvious examples. Note also that the test FP rate for ParCEL configurations with non-zero noise value differed from the training FP rate significantly less than in the case of ParCEL[0/T/T/-]. We can thus infer that when we let ParCEL bring some noise into the partial definitions, it is more tolerant to the lack of information in training data.

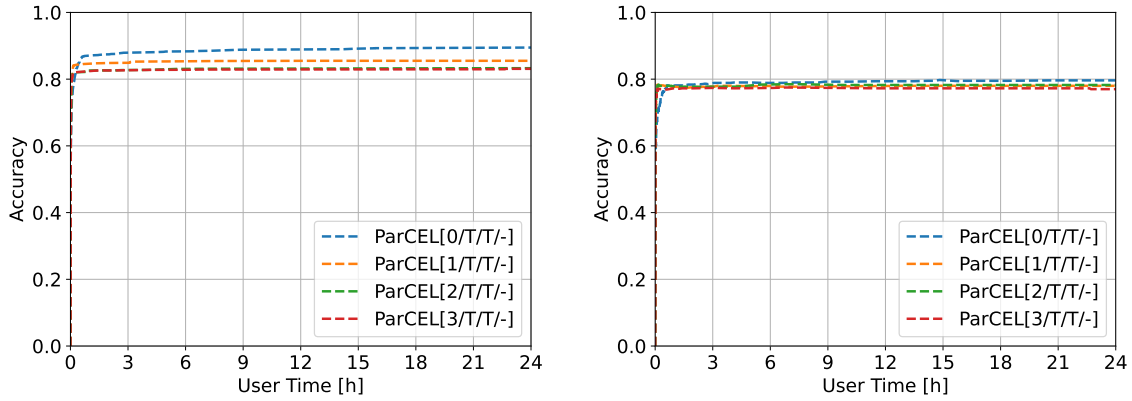
As mentioned before, we did not optimize the value of noise for ParCEL-Ex, hence, we tested only the configuration ParCEL-Ex[0/T/T/-]. Due to the similarities between ParCEL and ParCEL-Ex, we expected ParCEL-Ex[0/T/T/-] to perform as good as ParCEL[0/T/T/-]. In reality, ParCEL-Ex[0/T/T/-] proved to be able to produce comparable descriptions with just slightly worse FP rate and precision, which may result from the use of negated counter-partial definitions. The learning progression curves for ParCEL-Ex[0/T/T/-] are provided together with other ParCEL-Ex configurations in the next section.

For the second phase of optimization, ParCEL[0/T/T/-], ParCEL[1/T/T/-], and ParCEL-Ex[0/T/T/-] were selected.

7.1.2 Use of Negations/"Some-Only" Optimization

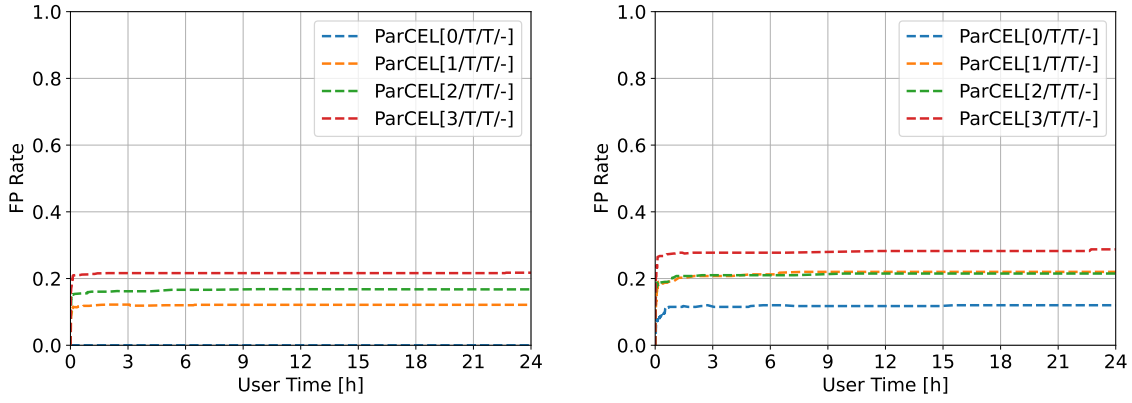
OCEL and CELOE

In the second phase, the performance of both OCEL configurations improved in all aspects after disabling exclusively the "some-only" checks, but worsened when we forbade negations (see Table 7.3). Our hypothesis regarding the "some-only" rule was thus



(a) Training accuracy. (b) Test accuracy.

Figure 7.5: Test case 1: ParCEL noise optimization – accuracy progression.



(a) Training FP rate. (b) Test FP rate.

Figure 7.6: Test case 1: ParCEL noise optimization – FP rate progression.

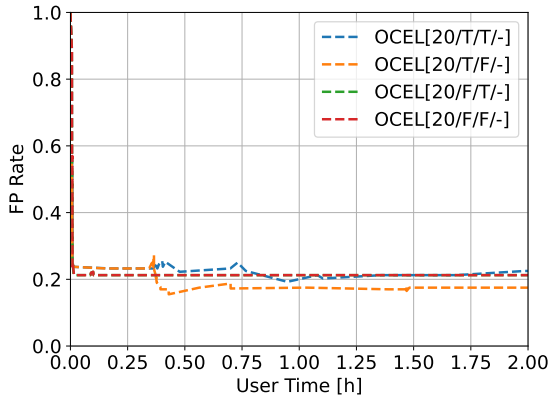
Table 7.3: Test Case 1: OCEL and CELOE Use of Negations/"Some-Only" Optimization.

Configuration	Training	Test				
	Accuracy	Accuracy	Precision	Recall	FP Rate	F1 Score
OCEL[20/T/T/-]	0.73 ± 0.01	0.72 ± 0.02	0.76 ± 0.04	0.66 ± 0.07	0.21 ± 0.06	0.70 ± 0.04
OCEL[20/T/F/-]	0.77 ± 0.00	0.76 ± 0.01	0.80 ± 0.02	0.69 ± 0.03	0.18 ± 0.02	0.74 ± 0.01
OCEL[20/F/T/-]	0.72 ± 0.01	0.70 ± 0.02	0.76 ± 0.04	0.60 ± 0.04	0.20 ± 0.04	0.67 ± 0.02
OCEL[20/F/F/-]	0.72 ± 0.01	0.71 ± 0.02	0.75 ± 0.03	0.62 ± 0.02	0.21 ± 0.03	0.68 ± 0.02
OCEL[25/T/T/-]	0.72 ± 0.00	0.70 ± 0.01	0.78 ± 0.03	0.56 ± 0.06	0.16 ± 0.04	0.65 ± 0.03
OCEL[25/T/F/-]	0.75 ± 0.00	0.74 ± 0.02	0.83 ± 0.04	0.61 ± 0.02	0.13 ± 0.04	0.70 ± 0.02
OCEL[25/F/T/-]	0.73 ± 0.01	0.70 ± 0.01	0.77 ± 0.04	0.56 ± 0.04	0.17 ± 0.04	0.65 ± 0.02
OCEL[25/F/F/-]	0.73 ± 0.01	0.70 ± 0.01	0.77 ± 0.04	0.56 ± 0.04	0.17 ± 0.04	0.65 ± 0.02
CELOE[30/T/T/-]	0.74 ± 0.00	0.71 ± 0.03	0.71 ± 0.03	0.72 ± 0.07	0.29 ± 0.05	0.71 ± 0.04
CELOE[30/T/F/-]	0.74 ± 0.00	0.71 ± 0.03	0.71 ± 0.03	0.72 ± 0.07	0.29 ± 0.05	0.71 ± 0.04
CELOE[30/F/T/-]	0.74 ± 0.00	0.71 ± 0.03	0.71 ± 0.03	0.72 ± 0.07	0.29 ± 0.05	0.71 ± 0.04
CELOE[30/F/F/-]	0.74 ± 0.00	0.71 ± 0.03	0.71 ± 0.03	0.72 ± 0.07	0.29 ± 0.05	0.71 ± 0.04
CELOE[40/T/T/-]	0.74 ± 0.00	0.71 ± 0.02	0.76 ± 0.03	0.63 ± 0.08	0.20 ± 0.06	0.69 ± 0.03
CELOE[40/T/F/-]	0.74 ± 0.00	0.71 ± 0.02	0.76 ± 0.03	0.63 ± 0.08	0.20 ± 0.06	0.69 ± 0.03
CELOE[40/F/T/-]	0.74 ± 0.00	0.72 ± 0.02	0.76 ± 0.04	0.63 ± 0.08	0.20 ± 0.06	0.69 ± 0.03
CELOE[40/F/F/-]	0.74 ± 0.00	0.72 ± 0.02	0.76 ± 0.04	0.63 ± 0.08	0.20 ± 0.06	0.69 ± 0.03

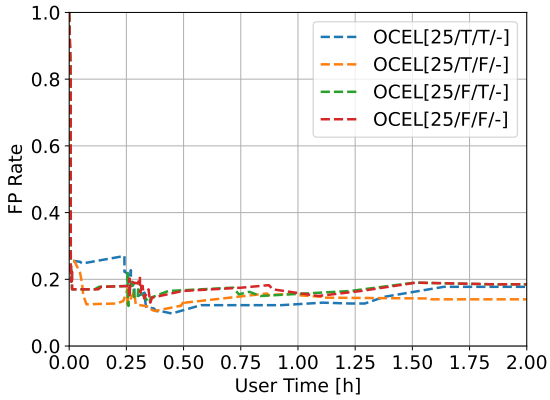
confirmed in this case, while banning negations did not lead to the awaited increase in the quality of descriptions, which may be due in part to the higher complexity of quality expressions without negations. For example, to describe PE files which do not have a debug data-directory table (class `Debug`) and are not digitally signed (class `Signature`), we would have to name all the remaining file features in the PE Malware Ontology that such a file can have in place of using the simple expression $\forall$`has_file_feature`.($\neg$`Debug` $\sqcap$ $\neg$`Signature`). We deliberately provided this example because OCEL[20/T/T/-] actually included this short expression in some of the most promising descriptions it found.

The progression of FP rate (Figure 7.7) shows that the key to the success of OCEL[20/T/F/-] and OCEL[25/T/F/-] from the perspective of evaluation scores was either the ability to lower the FP rate (OCEL[20/T/F/-]) or to stay close to the minimum reached early on (OCEL[25/T/F/-]).

Contrary to OCEL, CELOE was not affected by changes to the here-studied two hyper-parameters in terms of evaluation metrics at all. Moreover, for each level of noise, the new CELOE configurations mostly came to the same descriptions as the respective configuration which was selected in the first phase. We think that there was no reaction to the prohibition of negations since CELOE was designed to favor shorter descriptions really aggressively, which discourages it from even considering negations. In fact, CELOE[30/T/T/-] and CELOE[40/T/T/-] already outputted descriptions containing negations only rarely. Disabling the "some-only" rule probably did not deliver

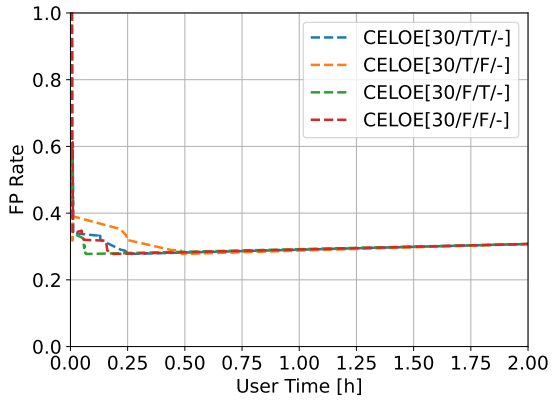


(a) OCEL with noise of 20.

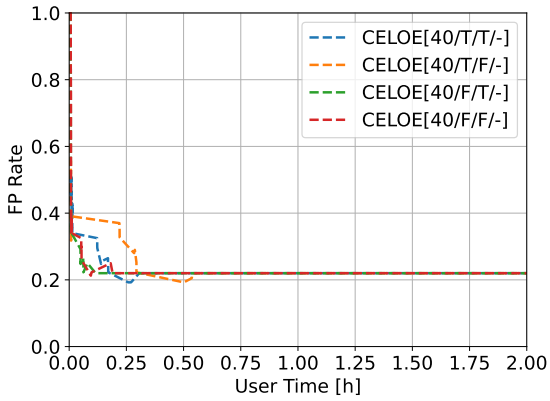


(b) OCEL with noise of 25.

Figure 7.7: Test case 1: OCEL use of negations/"some-only" optimization – Test FP rate progression.



(a) CELOE with noise of 30.



(b) CELOE with noise of 40.

Figure 7.8: Test case 1: CELOE use of negations/"some-only" optimization – progression of test FP rate.

any improvement because an expression of the form $\forall r.C$ may be convenient merely to complement a strong, complex description. This is where also much of the superiority of OCEL[20/T/F/-] and OCEL[25/T/F/-] over OCEL[20/T/T/-] and OCEL[25/T/T/-], respectively, resided.

The only difference we noticed between the configurations of CELOE was the learning progression (see Figure 7.8). More precisely, the configurations with the search space depleted of negations reached the best descriptions faster than CELOE[30/T/T/-] and CELOE[40/T/T/-], accordingly, whereas those with the greatest expressive power, i.e., with negations enabled and the "some-only" rule disabled, were not far from being two times slower. We see this as a direct consequence of the extent to which the search was restricted.

Table 7.4: Test Case 1: ParCEL and ParCEL-Ex Use of Negations/"Some-Only" Optimization.

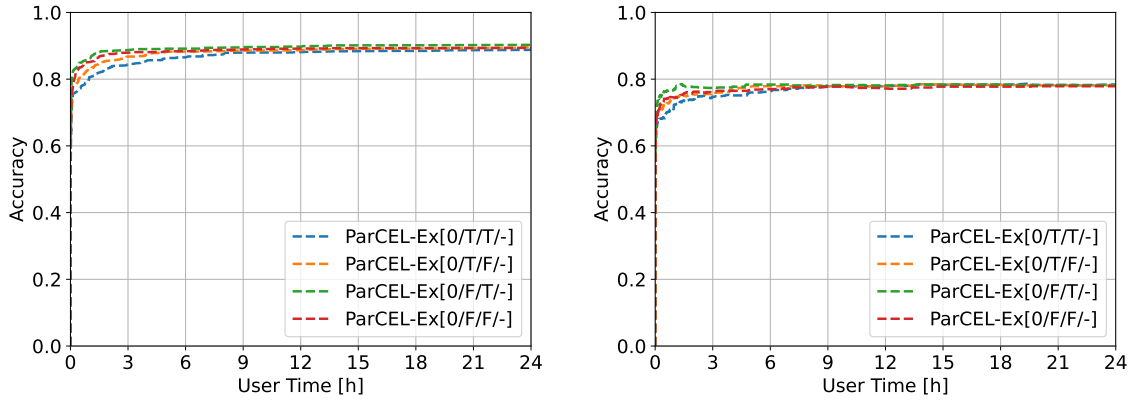
Configuration	Training	Test				
	Accuracy	Accuracy	Precision	Recall	FP Rate	F1 Score
ParCEL[0/T/T/-]	0.90 ± 0.01	0.79 ± 0.02	0.85 ± 0.02	0.70 ± 0.04	0.12 ± 0.02	0.77 ± 0.03
ParCEL[0/T/F/-]	0.90 ± 0.01	0.79 ± 0.02	0.85 ± 0.03	0.70 ± 0.05	0.12 ± 0.03	0.77 ± 0.03
ParCEL[0/F/T/-]	0.90 ± 0.01	0.79 ± 0.02	0.85 ± 0.03	0.70 ± 0.04	0.13 ± 0.03	0.77 ± 0.02
ParCEL[0/F/F/-]	0.90 ± 0.01	0.79 ± 0.02	0.86 ± 0.02	0.70 ± 0.05	0.11 ± 0.02	0.77 ± 0.03
ParCEL[1/T/T/-]	0.86 ± 0.01	0.79 ± 0.03	0.80 ± 0.05	0.77 ± 0.03	0.19 ± 0.05	0.79 ± 0.02
ParCEL[1/T/F/-]	0.86 ± 0.00	0.79 ± 0.02	0.81 ± 0.04	0.76 ± 0.03	0.18 ± 0.05	0.79 ± 0.01
ParCEL[1/F/T/-]	0.86 ± 0.01	0.78 ± 0.02	0.80 ± 0.04	0.76 ± 0.03	0.19 ± 0.05	0.78 ± 0.02
ParCEL[1/F/F/-]	0.86 ± 0.00	0.77 ± 0.02	0.77 ± 0.04	0.76 ± 0.04	0.23 ± 0.06	0.76 ± 0.01
ParCEL-Ex[0/T/T/-]	0.89 ± 0.02	0.78 ± 0.02	0.83 ± 0.02	0.71 ± 0.03	0.14 ± 0.02	0.76 ± 0.02
ParCEL-Ex[0/T/F/-]	0.90 ± 0.01	0.78 ± 0.02	0.82 ± 0.02	0.71 ± 0.04	0.16 ± 0.02	0.76 ± 0.02
ParCEL-Ex[0/F/T/-]	0.90 ± 0.00	0.78 ± 0.02	0.83 ± 0.03	0.71 ± 0.06	0.15 ± 0.03	0.77 ± 0.03
ParCEL-Ex[0/F/F/-]	0.90 ± 0.01	0.78 ± 0.03	0.82 ± 0.03	0.71 ± 0.06	0.16 ± 0.04	0.76 ± 0.04

Despite the prolonged way to the final descriptions, we chose CELOE[30/T/F/-] and CELOE[40/T/F/-] for the next phase since firstly, these configurations have access to the richest search space, which we planned to narrow in the third phase of optimization, and secondly, they had still plenty of time left to further learn after getting to the same level as their respective competitors. Among OCEL configurations, OCEL[20/T/F/-] and OCEL[25/T/F/-] were selected.

ParCEL and ParCEL-Ex

Based on the evaluation of the final descriptions constructed by various configurations of ParCEL (see Table 7.4), we can conclude that it depended on the level of noise which combination of settings suited this algorithm the most. With the noise value of 0, there were only minor differences between the configurations. The deactivation of the "some-only" rule helped only if negations were disabled as well. Since ParCEL relies too much on training data for noise equal to 0, we suppose that this may correlate with the fact that a learner is more prone to over-fitting even when solely negations are enabled or the "some-only" rule is disabled. However, with noise set to 1, ParCEL not only avoided this pitfall, but also took advantage of the new opportunities brought by loosening the restrictions on universal quantification, as the partial definitions already do not have to describe the seen malware examples precisely in this case. This lead to a moderate performance gain of the ParCEL[1/T/F/-] configuration in comparison with ParCEL[1/T/T/-] from the previous phase.

The results of our experimentation with ParCEL-Ex in the second phase show that it was fruitful to study how ParCEL-Ex would behave with negations enabled despite being disabled by default because ParCEL-Ex[0/T/T/-] delivered the best results in



(a) Training accuracy.

(b) Test accuracy.

Figure 7.9: Test case 1: ParCEL-Ex use of negations/"some-only" optimization – accuracy progression.

terms of FP rate (see Table 7.4). Analyzing the output of ParCEL-Ex[0/T/T/-], we found out that it leveraged the presence of negations to include them in both partial and counter-partial definitions. Nonetheless, allowing negations proved to be beneficial exclusively in the situation when the "some-only" rule was applied, which we think is connected to the above mentioned amplification of the risks to over-fit if we permit negations but do not perform the "some-only" checks.

Examining the progression of accuracy, displayed in Figure 7.9, we noticed a quite significant contrast between the pace at which the individual configurations of ParCEL-Ex learned. Clearly, the redundancy introduced by negations, stemming from the fact that the negated counter-partial definitions are used for the same purpose, caused ParCEL-Ex to slow down.

Moreover, the impact of negations was much more severe if the generation of universal quantification was controlled by the "some-only" rule. This was a consequence of the fact that negations were overly distracting ParCEL-Ex[0/T/T/-]. For example, the expression $\texttt{ExecutableFile} \sqcap \neg\texttt{DynamicLinkLibrary}$, in spite of being equivalent to $\texttt{ExecutableFile}$, frequently appeared in partial and counter-partial definitions found by this configuration since it was still shorter than, e.g.,

$$\exists\texttt{has\_file\_feature}.\top \sqcap \forall\texttt{has\_file\_feature}.\texttt{LowImportsCount},$$

and thus more likely to be refined. Contrarily, the configuration with the "some-only" rule disabled, i.e., ParCEL-Ex[0/T/F/-], could simply opt for

$$\forall\texttt{has\_file\_feature}.\texttt{LowImportsCount}.$$

On the grounds of our observations, we decided to continue with the optimization of ParCEL[0/F/F/-], ParCEL[1/T/F/-], and ParCEL-Ex[0/T/T/-].

Table 7.5: Test Case 1: OCEL and CELOE Cardinality Optimization.

Configuration	Training	Test				
	Accuracy	Accuracy	Precision	Recall	FP Rate	F1 Score
OCEL[20/T/F/-]	0.77 ± 0.00	0.76 ± 0.01	0.80 ± 0.02	0.69 ± 0.03	0.18 ± 0.02	0.74 ± 0.01
OCEL[20/T/F/HS]	0.74 ± 0.02	0.72 ± 0.02	0.76 ± 0.03	0.66 ± 0.04	0.21 ± 0.03	0.71 ± 0.03
OCEL[20/T/F/M1]	0.76 ± 0.00	0.76 ± 0.01	0.80 ± 0.02	0.69 ± 0.03	0.18 ± 0.02	0.74 ± 0.01
OCEL[25/T/F/-]	0.75 ± 0.00	0.74 ± 0.02	0.83 ± 0.04	0.61 ± 0.02	0.13 ± 0.04	0.70 ± 0.02
OCEL[25/T/F/HS]	0.75 ± 0.00	0.74 ± 0.02	0.83 ± 0.04	0.61 ± 0.02	0.13 ± 0.04	0.70 ± 0.02
OCEL[25/T/F/M1]	0.75 ± 0.00	0.74 ± 0.02	0.83 ± 0.04	0.61 ± 0.02	0.13 ± 0.04	0.70 ± 0.02
CELOE[30/T/F/-]	0.74 ± 0.00	0.71 ± 0.03	0.71 ± 0.03	0.72 ± 0.07	0.29 ± 0.05	0.71 ± 0.04
CELOE[30/T/F/HS]	0.74 ± 0.01	0.71 ± 0.02	0.69 ± 0.03	0.76 ± 0.07	0.34 ± 0.07	0.72 ± 0.03
CELOE[30/T/F/M1]	0.74 ± 0.00	0.72 ± 0.03	0.73 ± 0.02	0.71 ± 0.06	0.27 ± 0.01	0.72 ± 0.04
CELOE[40/T/F/-]	0.74 ± 0.00	0.71 ± 0.02	0.76 ± 0.03	0.63 ± 0.08	0.20 ± 0.06	0.69 ± 0.03
CELOE[40/T/F/HS]	0.74 ± 0.01	0.71 ± 0.03	0.74 ± 0.06	0.67 ± 0.08	0.25 ± 0.10	0.70 ± 0.03
CELOE[40/T/F/M1]	0.74 ± 0.00	0.71 ± 0.02	0.76 ± 0.03	0.63 ± 0.08	0.20 ± 0.06	0.69 ± 0.03

7.1.3 Cardinality Optimization

OCEL and CELOE

As discussed in Section 6.3, in the last phase of optimization, we aimed to motivate the algorithms to explore more valuable and accessible areas of the search space.

Regarding OCEL, the results in Table 7.5 hint that these goals were not achieved. Restricting the use of cardinality constraints to `has_section` even induced an increase in FP rate for the configuration with the noise value of 20. The problem, however, did not consist in the lack of expressive capabilities as the descriptions found by OCEL[20/T/F/-] were already free of any cardinality constraints on other object properties. Hence, OCEL needed such expressions just to populate the search tree, helping its heuristic to provide better guidance in this particular situation.

The OCEL[20/T/F/M1] configuration reached almost the same final descriptions as OCEL[20/T/F/-], taking a very similar path to that of OCEL[20/T/F/-]. This was also true for OCEL[25/T/F/-] and the configurations derived from it. Therefore, the only visible change in OCEL's performance we registered when limiting the maximum cardinality in at-most restrictions to 1 was a $3-10\%$ reduction in the total number of expressions searched.

For CELOE configurations, it was especially hard to cope with the restriction on cardinality constraints allowing no property except for `has_section` to be used. We discovered that the main reason behind these difficulties was that CELOE[30/T/F/-] and CELOE[40/T/F/-] often arrived at the following final description:

$$\leq 1 \ \texttt{has\_file\_feature}.(\texttt{CLR} \sqcup \texttt{Debug} \sqcup \texttt{Resources} \sqcup \texttt{Signature}). \tag{7.1}$$

Although we do not view this as a satisfactory characterization of malware (ergo, the

Table 7.6: Test Case 1: ParCEL and ParCEL-Ex Cardinality Optimization.

Configuration	Training	Test				
	Accuracy	Accuracy	Precision	Recall	FP Rate	F1 Score
ParCEL[0/F/F/-]	0.90 ± 0.01	0.79 ± 0.02	0.86 ± 0.02	0.70 ± 0.05	0.11 ± 0.02	0.77 ± 0.03
ParCEL[0/F/F/HS]*	0.89 ± 0.01	0.79 ± 0.02	0.87 ± 0.03	0.68 ± 0.03	0.10 ± 0.03	0.76 ± 0.03
ParCEL[0/F/F/M1]*	0.89 ± 0.00	0.79 ± 0.02	0.86 ± 0.01	0.69 ± 0.04	0.11 ± 0.01	0.76 ± 0.03
ParCEL[1/T/F/-]	0.86 ± 0.00	0.79 ± 0.02	0.81 ± 0.04	0.76 ± 0.03	0.18 ± 0.05	0.79 ± 0.01
ParCEL[1/T/F/HS]	0.86 ± 0.01	0.79 ± 0.02	0.80 ± 0.05	0.76 ± 0.03	0.19 ± 0.06	0.78 ± 0.02
ParCEL[1/T/F/M1]	0.85 ± 0.01	0.79 ± 0.03	0.81 ± 0.05	0.76 ± 0.04	0.18 ± 0.06	0.78 ± 0.03
ParCEL-Ex[0/T/T/-]	0.89 ± 0.02	0.78 ± 0.02	0.83 ± 0.02	0.71 ± 0.03	0.14 ± 0.02	0.76 ± 0.02
ParCEL-Ex[0/T/T/HS]	0.88 ± 0.01	0.76 ± 0.02	0.82 ± 0.03	0.67 ± 0.03	0.15 ± 0.03	0.74 ± 0.02
ParCEL-Ex[0/T/T/M1]	0.89 ± 0.01	0.78 ± 0.02	0.82 ± 0.02	0.70 ± 0.04	0.15 ± 0.03	0.76 ± 0.02

* Some experiments run out of memory.

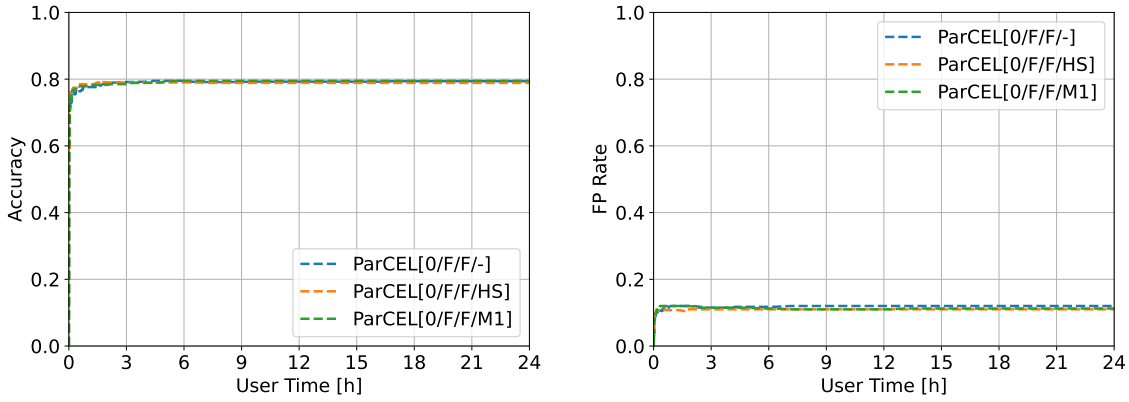
higher final FP rate), it managed to exceed 0.70 test accuracy most of the time and apparently, CELOE was unable to find an equally successful alternative in those situations. On the contrary, setting the upper limit for quantities in at-most restrictions to 1 did not degrade the performance of CELOE and contributed to an improvement in FP rate in the case of CELOE with the noise of 30 (see CELOE[30/T/F/M1] in Table 7.5), which at least marginally supports our theory that searching "higher-order" at-most restrictions is unnecessary in this scenario. Similarly to OCEL, CELOE[40/T/F/M1] benefited from this limitation just in terms of the size of the search tree, which shrunk by about 15%.

Since there were no signs of negative effects of restricting the quantity in at-most restrictions, we selected OCEL[20/T/F/M1], OCEL[25/T/F/M1], CELOE[30/T/F/M1], and CELOE[40/T/F/M1] as the best configurations.

ParCEL and ParCEL-Ex

Looking at the results in Table 7.6, we see that despite our efforts to make the search tree representation as compact as possible, some of the experiments with the configurations ParCEL[0/F/F/HS] and ParCEL[0/F/F/M1] were terminated prematurely by the out-of-memory killer. As this happened sometime between 12 and 24 user-time hours of learning, the last known statistics should give us a decent picture of what the final scores would be if these experiments had not been interrupted, because from that point onward, only few partial definitions are added to the resulting description.

We assume that the enormous surge in the memory required may mean that ParCEL could not later decide on which expressions would be more valuable, and thus kept them all, since the restrictions on cardinality constraints prevented it from traversing less relevant parts of the search space. Nevertheless, ParCEL[0/F/F/HS] and ParCEL[0/F/F/M1] reached an FP rate comparable to or slightly better than the FP



(a) Accuracy progression.

(b) FP rate progression.

Figure 7.10: Test case 1: ParCEL with noise of 0 cardinality optimization – progression on test sets.

rate of the description created by ParCEL[0/F/F/-] until the corresponding runs ended (see Figure 7.10).

With noise set to 1, ParCEL was able to approach the learning task a bit more appropriately in the unconstrained settings as well. This might suggest that on the smaller scale at which partial definitions operate, i.e., when it is enough to precisely cover a handful of positive examples, cardinality constraints on other properties than `has_section` and at-most restrictions with higher quantities than 1 can be interesting. For instance, ParCEL[1/T/F/-] included the following expressions in partial definitions:

$$\exists \texttt{has\_section}.(\geq 3\ \texttt{has\_section\_feature}.\top),$$
$$\leq 2\ \texttt{has\_action.ProcessHandling}.$$

The performance of ParCEL-Ex was influenced by the restrictions on the use of cardinality constraints in a similar way. We noticed that when forming counter-partial definitions, ParCEL-Ex[0/T/T/-] also exploited the negative connotation of at-most restrictions, that is, the fact that they forbid individuals from having too many features. The other two configurations were substantially limited in this regard.

ParCEL[0/F/F/-], ParCEL[1/T/F/-], and ParCEL-Ex[0/T/T/-] were chosen as the ultimately best configurations.

7.1.4 Validation

As we expected, the performance of the optimized learning algorithms varied across the three prepared validation datasets. Nevertheless, the results of validation in Table ... show that the majority of configurations stayed at approx. the same level as during optimization. The most evident exceptions were CELOE[40/T/F/M1], which surpris-

$$(\texttt{ExecutableFile} \sqcap \exists \texttt{has\_file\_feature}.\texttt{MultipleExecutableSections}) \sqcup (\exists \texttt{has\_action}.(\neg \texttt{SendHttpConnectRequest}) \sqcap \forall \texttt{has\_file\_feature}.(\neg \texttt{Signature} \sqcap \neg \texttt{Symbols})) \quad (7.2)$$

$$(\texttt{DynamicLinkLibrary} \sqcap \exists \texttt{has\_action}.\texttt{FileHandling}) \sqcup (\exists \texttt{has\_file\_feature}.\texttt{MultipleExecutableSections}) \quad (7.3)$$

$$\exists \texttt{has\_file\_feature}.\texttt{LowImportsCount} \sqcap \geq 3\ \texttt{has\_section}.(\texttt{InitializedDataSection} \sqcap \exists \texttt{has\_section\_feature}.\texttt{NonstandardSectionName}) \quad (7.4)$$

$$\texttt{DynamicLinkLibrary} \sqcap \exists \texttt{has\_action}.\texttt{OpenProcess} \sqcap \neg(\texttt{DynamicLinkLibrary} \sqcap \exists \texttt{has\_action}.\texttt{OpenFileMapping} \sqcap \exists \texttt{has\_file\_feature}.\texttt{Signature}) \quad (7.5)$$

Figure 7.11: Test Case 1: Examples of learned descriptions and partial definitions.

ingly improved, and CELOE[30/T/F/M1], whose performance deteriorated. Overall, we can thus conclude that the quality of the output we obtained throughout the optimization proved to be a strong indicator of how a configuration would behave when trained on other data, although the volume of the data may still affect an algorithm's abilities to learn. As such, we verified that it is sufficient to concentrate on optimizing solely the best configuration from the first phase, i.e., that the selection of noise plays an overarching role in an algorithm's performance and tweaking the other parameters does not help a configuration with a worse setting of noise to outperform a configuration with a better choice of noise.

7.1.5 Learned Descriptions

The main advantage of concept-learning is its explainability, which we demonstrate on a few examples of learned descriptions and partial definitions in Figure 7.11.

OCEL was able to identify that a top-level disjunction is necessary to deal with distinct categories of malware separately. One of the most accurate descriptions found by OCEL[25/T/F/M1], with an accuracy of 0.75 and FP rate equal to 0.15, was the expression (7.2), which can be interpreted as follows: *A PE file is malicious if it is an EXE that has multiple sections which can be executed, or if it simultaneously (i) cannot*

send an HTTP client request for a connection to a server, (ii) is not digitally signed, and (iii) does not contain COFF debug symbols. Except for the part requiring a PE file to be unable to connect via HTTP to a server, we consider this a reasonable, albeit simple characterization of malware.

On account of preferring shorter descriptions, CELOE struggled with finding disjunctions and the most promising it discovered were still quite broad, such as, (7.3). Besides these attempts to search in multiple directions at the same time, it often outputted descriptions similar to the expression (7.1) presented in Section 7.1.3.

The final descriptions constructed by ParCEL and ParCEL-Ex are disjunctions by design, so taking a case-wise approach posed no problem for them. In our scenario, their descriptions usually consisted of above 50 partial definitions targeted at a specific subset of malicious samples. For instance, ParCEL[0/F/F/-] incorporated the partial definition (7.4) into the final description to cover 10 positive examples in the training data (3 in the test data). An extensive use of counter-partial definitions in ParCEL-Ex was also visible as many of the partial definitions learned by ParCEL-Ex, e.g., (7.5), resulted from a combination of a mediocre expression with negated counter-partial definitions.

7.2 Test Cases 2 and 3

Now, we analyze the results from the test cases 2 and 3, mostly highlighting the differences in the behavior and performance of the corresponding configurations.

Again, we discuss first, and draw conclusions in the end.between these two test cases and comparing them to the results obtained in the first test case.

To speed up the optimization in these test cases, we set the time limit for ParCEL and ParCEL-Ex to 12 hours of user time since we noticed that in the first test case, ParCEL, ParCEL-Ex, and CELOE configurations reached their maximum, or got at least close enough, during the first quarter of training. However, the timeout for CELOE remained unchanged as it occupies only one execution thread, which allows us to run multiple experiments with CELOE concurrently. Additionally, in the test cases 2 and 3, where we always proceeded with a single configuration of each algorithm.

7.2.1 Noise Optimization

OCEL and CELOE

Bibliography

[1] Hyrum S. Anderson and Phil Roth. EMBER: An Open Dataset for Training Static PE Malware Machine Learning Models. *ArXiv e-prints*, April 2018. arXiv: 1804.04637 [cs.CR].

[2] Grigoris Antoniou and Frank Van Harmelen. *A semantic web primer*. MIT press, 2004.

[3] Ömer Aslan and Refik Samet. Investigation of possibilities to detect malware using existing tools. In *2017 IEEE/ACS 14th International Conference on Computer Systems and Applications (AICCSA)*, pages 1277–1284, 2017.

[4] Franz Baader, Ian Horrocks, Carsten Lutz, and Uli Sattler. *An Introduction to Description Logic*. Cambridge University Press, 2017.

[5] Karl Bridge et al. PE format. 2023. URL: https://learn.microsoft.com/en-us/windows/win32/debug/pe-format (visited on 03/24/2023).

[6] Lorenz Bühmann, Jens Lehmann, and Patrick Westphal. Dl-learner—a framework for inductive learning on the semantic web. *Journal of Web Semantics*, 39:15–24, 2016. ISSN: 1570-8268.

[7] Alfonso Caruso. The old useless DOS header of Windows PE. 2022. URL: https://quercialabs.com/blog/dos-header-portable-executable-windows/ (visited on 03/24/2023).

[8] Werner Ceusters, B Smith, and Louis J. Goldberg. A terminological and ontological analysis of the nci thesaurus. *Methods of Information in Medicine*, 44:498–507, 2005.

[9] Pierre Chaussecourte, Birte Glimm, Ian Horrocks, Boris Motik, and Laurent Pierre. The energy management adviser at edf. In *The Semantic Web – ISWC 2013*, pages 49–64, Berlin, Heidelberg. Springer Berlin Heidelberg, 2013. ISBN: 978-3-642-41338-4.

[10] N.A. Diamantidis, D. Karlis, and E.A. Giakoumakis. Unsupervised stratification of cross-validation for accuracy estimation. *Artificial Intelligence*, 116(1):1–16, 2000. ISSN: 0004-3702.

[11] Pedro M. Domingos. The role of occam's razor in knowledge discovery. *Data Mining and Knowledge Discovery*, 3:409–425, 1999.

[12] Tom Fawcett. An introduction to roc analysis. *Pattern Recogn. Lett.*, 27(8):861–874, June 2006. ISSN: 0167-8655.

[13] Bernardo Grau, Ian Horrocks, Boris Motik, Bijan Parsia, Peter Patel-Schneider, and Uli Sattler. OWL 2: the next step for OWL. *Web Semantics: Science, Services and Agents on the World Wide Web*, 6:309–322, November 2008.

[14] Richard E. Harang and Ethan M. Rudd. SOREL-20M: a large scale benchmark dataset for malicious PE detection. *ArXiv e-prints*, December 2020. arXiv: `2012.07634 [cs.CR]`.

[15] Ian Horrocks, Oliver Kutz, and Ulrike Sattler. The even more irresistible sroiq. In *International Conference on Principles of Knowledge Representation and Reasoning*, 2006.

[16] Jens Lehmann, Sören Auer, Lorenz Bühmann, and Sebastian Tramp. Class expression learning for ontology engineering. *Journal of Web Semantics*, 9(1):71–81, 2011. ISSN: 1570-8268.

[17] Jens Lehmann and Pascal Hitzler. Concept learning in description logics using refinement operators. *Machine Learning*, 78:203–250, 2009.

[18] Bo Li, Kevin Roundy, Chris Gates, and Yevgeniy Vorobeychik. Large-scale identification of malicious singleton files. In *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy*, CODASPY '17, pages 227–238, Scottsdale, Arizona, USA. Association for Computing Machinery, 2017. ISBN: 9781450345231.

[19] Royi Ronen, Marian Radu, Corina Feuerstein, Elad Yom-Tov, and Mansour Ahmadi. Microsoft malware classification challenge. *ArXiv e-prints*, 2018. arXiv: `1802.10135 [cs.CR]`.

[20] Sebastian Rudolph. *Foundations of description logics*. In *Reasoning Web. Semantic Technologies for the Web of Data: 7th International Summer School 2011, Galway, Ireland, August 23-27, 2011, Tutorial Lectures*. Springer, Berlin, Heidelberg, 2011, pages 76–136.

[21] Peter Švec, Štefan Balogh, Martin Homola, and Ján Kľuka. Knowledge-based dataset for training PE malware detection models. *ArXiv e-prints*, December 2022. arXiv: `2301.00153 [cs.CR]`.

[22] Texas Instruments Inc. Common Object File Format. Technical report SPRAAO8–April 2009, Texas Instruments, April 2009.

[23] An C. Tran, Jens Dietrich, Hans W. Guesgen, and Stephen Marsland. An approach to parallel class expression learning. In *Rules on the Web: Research and Applications*, pages 302–316, Berlin, Heidelberg. Springer, 2012.

[24] An Cong Tran, Jens Dietrich, Hans W. Guesgen, and Stephen R. Marsland. Two-way parallel class expression learning. In *Asian Conference on Machine Learning*, 2012.

[25] Limin Yang, Arridhana Ciptadi, Ihar Laziuk, Ali Ahmadzadeh, and Gang Wang. BODMAS: an open dataset for learning based temporal analysis of PE malware. In *2021 IEEE Security and Privacy Workshops (SPW)*, pages 78–84, 2021.