VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

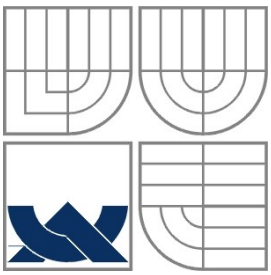
ANALÝZA A IMPLEMENTACE PRAVIDEL PRO SYSTÉM ZPRACOVÁNÍ VIROVÝCH VZORKŮ

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

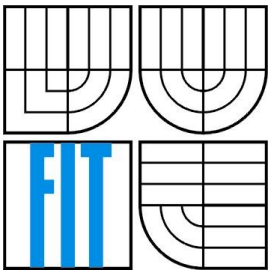
AUTOR PRÁCE
AUTHOR

JIŘÍ JANEČEK

BRNO 2009



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

ANALÝZA A IMPLEMENTACE PRAVIDEL PRO SYSTÉM ZPRACOVÁNÍ VIROVÝCH VZORKŮ

ANALYSIS AND IMPLEMENTATION OF RULES FOR VIRUS SAMPLES PROCESSING SYSTEM

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

JIŘÍ JANEČEK

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. VLADIMÍR JANOUŠEK Ph.D.

BRNO 2009

Abstrakt

Tato bakalářská práce se zabývá analýzou a implementací pravidel pro program, jenž směřuje virové vzorky na jejich pozdější zpracování. V tomto programu je integrován expertní systém k jehož vytvoření by použit nástroj CLIPS.

Současně tato práce popisuje prostředí, ve kterém je tento analyzátor umístěn, a význam takové komponenty v systému zpracování virových vzorků. Analyzátor byl implementován v jazyce C/C++.

Abstract

This bachelor's thesis deals with analysis and implementation of rules for program, which routes virus samples for their later processing. The expert system is integrated in this program. It was created by using tool called CLIPS.

This thesis also describes environment, where program is situated, and importance of component like this in system for virus sample processing. Analyzer has been implemented in C/C++ language.

Klíčová slova

CLIPS, expertní systém, single-slot, virový vzorek, multislot, defrule, deftemplate, defunction, defmodule, deffacts, analyzátor, sandbox

Keywords

CLIPS, expert system, single-slot, virus sample, multislot, defrule, deftemplate, defunction, defmodule, deffacts, analyzer, sandbox

Citace

Jiří Janeček: Analýza a implementace pravidel pro systém zpracování virových vzorků , bakalářská práce, Brno, FIT VUT v Brně, 2009

Analýza a implementace pravidel pro systém zpracování virových vzorků

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Vladimíra Janouška Ph.D.
Další informace mi poskytli Pavel Krčma a Ryan Hicks z AVG Technologies s.r.o.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Jiří Janeček
1.5.2009

Poděkování

Za vedení při tvorbě této zprávy bych chtěl poděkovat Ing. Janouškovi a kolegům z AVG Technologies s.r.o., kteří mi byli oporou při tvorbě softwarové části.

Obsah

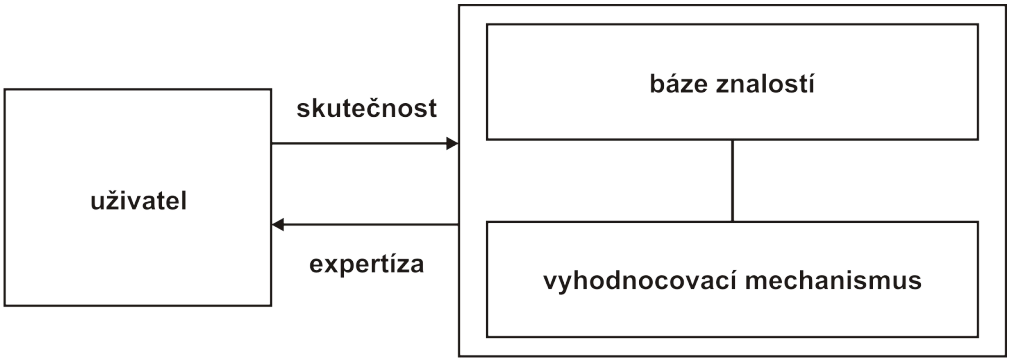
Obsah....1
1 Problematika expertních systémů....3
1.1 Co je to expertní systém....3
1.2 Kde jsou expertní systémy nasazovány....3
1.3 Výhody expertních systémů nad lidskými experty....4
1.4 Nástroje a jazyky na tvorbu expertních systémů....4
1.4.1 Volně šiřitelné nástroje....4
1.4.2 Placené nástroje....5
2 CLIPS....5
2.1 Historie CLIPS....5
2.2 Programovací paradigmata v CLIPS....6
2.3 CLIPS přenosnost a integrace....7
2.4 Formy užití CLIPSu....7
2.5 Dostupné konstrukce v CLIPSu verze 6.3....7
2.5.1 deftemplate....8
2.5.2 defglobal....8
2.5.3 deffunction....8
2.5.4 defrule....9
2.5.5 deffact....9
2.5.6 defmodule....10
2.6 CLIPS, jeho kompilace a užití v uživatelských programech....10
2.6.1 Postup překladu konstrukcí v CLIPSu na struktury jazyka C....11
3 Analýza současného řešení systému pro zpracování virových vzorků....12
3.1 Analýza prostředí současného analyzátoru....12
3.1.1 Zdroje vzorků pro analýzu....12
3.1.2 Výstupy analyzátoru....13
3.2 Význam analyzátoru vzorků....14
3.3 Analýza programu VCB a následná volba integrace expertního systému....14
3.3.1 Stav před zavedením CLIPSu....14
3.3.2 Výhody plynoucí ze zavedení expertního systému....15
3.3.3 Proč jsem zvolil právě CLIPS....15
4 Návrh a implementace řešení nového analyzátoru....16
5 Návrh a implementace části expertního systému....16
5.1 SAMPLE, definice vzorku pro CLIPS....16

5.2 SAMPLE, význam jednotlivých slotů....17
5.2.1 Datové sloty....17
5.2.2 Servisní sloty....20
5.3 Návrh a implementace pravidel analyzátoru....22
5.3.1 Společná část pro všechna pravidla....23
5.3.2 Ukázka implementovaných pravidel v CLIPSu....23
5.3.3 Implementovaný pravidla v analyzátoru – stručný přehled....27
5.3.4 Funkce v části expertního systému....30
6 Návrh a implementace programové části....32
6.1 Loader....32
6.1.1 zdroje informací pro Loader....33
6.2 Expert....34
6.3 Output....35
7 Testování a Statistiky....36
7.1 Testování....36
7.1.1 Způsob testování....36
7.1.2 Data plynoucí z testování....36
7.2 Statistiky....37
8 Závěr....38
8.1 Cíle a využití....38
8.2 Práce na projektu - chronologická data....38
Literatura....39
Příloha A....40
Příloha B....40

1 Problematika expertních systémů

1.1 Co je to expertní systém

Expertní systém [1] je z obecného pohledu software, který se snaží zastoupit experta v podobě člověka v jednom specifickém oboru. Uživatel takového systému očekává, že na svoji otázku v podobě skutečnosti (anglicky je tato skutečnost označována jako `fact` [1]) dostane odpověď. Systém tedy má za úkol emulovat chování skutečného lidského experta a nabídnout tak uživateli tento expertní názor.



obr. 1.1 základní pojetí funkce expertního systému

Obecná podoba expertního systému je složena ze dvou částí. První část nese znalosti (anglicky je tato část označována jako `knowledge base` [1]). Zdroje těchto informací můžeme nalézt v knihách nebo je získat přímo od expertů z daného oboru. Druhou částí je mechanismus pro vyhodnocení skutečnosti (v angličtině označován jako `inference engine` [1]), jehož úkolem je konfrontovat vstupní skutečnost s bází znalostí.

1.2 Kde jsou expertní systémy nasazovány

Využití expertních systémů [1] je velmi různorodé. Tyto systémy můžeme nalézt třeba v medicíně (MYCIN – využívaný ke diagnostice a léčení bakteriálních infekcí), v chemii (DENDRAL – interpretuje molekulární strukturu), v elektrotechnice (ACE – diagnostika chyb v telefonní síti) či dokonce v geologii (PROSPECTOR – interpretuje geologická data s ohledem na naleziště minerálů). Existují ještě mnohé další obory, kde se dnes uplatňují expertní systémy, pro obecné povědomí však postačí těchto pár příkladů.

1.3 Výhody expertních systémů nad lidskými experty

Asi tou nejzásadnější výhodou je fakt, že expertní systém nikdy nezapomíná. Znalosti do něj vložené se neztratí. Další nespornou výhodou je fakt, že mohou pracovat nepřetržitě, na rozdíl od jejich lidských protějšků, a to i v místech, kde je zatím existence člověka nemožná. Jednou z dalších neopomenutelných výhod je fakt, že pracují podstatně efektivněji, než lidé.

1.4 Nástroje a jazyky na tvorbu expertních systémů

1.4.1 Volně šiřitelné nástroje

Na světě existuje poměrně velké množství různých nástrojů [2] na tvorbu expertních systémů. Sice většina těchto nástrojů je použitelná pouze za úplatu, ale najdou se i takové, které je možné využívat zdarma.

Z palety těchto volně šiřitelných nástrojů bych jmenoval třeba CLIPS (C Language Integrated Production System) a jeho rozšíření jako FuzzyCLIPS či DYNACLIPS. FyzzyCLIPS je rozšíření klasického CLIPSu o podporu práce s fuzzy logikou a DYNACLIPS je sada knihoven pro práci s dynamicky se měnící bází znalostí a zahrnuje i tzv. tabulovou architekturu (anglicky Blackboard architecture). O klasickém CLIPSu se tato publikace zmiňuje níže v kapitole o architektuře CLIPS a jeho využití právě pro naši problematiku. Tuto kapitolu jsem nazval CLIPS.

Dalším volně šiřitelným nástrojem na tvorbu expertních systémů je GEST (Generic Expert System Tools). GEST je shell, který je možné využít v mnoha různých problémových doménách (anglicky problém domin). Podporuje jak forward tak backward chaining. Zajímavé je, že tento nástroj podporuje hned několik reprezentací znalostí, a to jak formou rámců (anglicky frames), pravidel (anglicky rules) či procedur (anglicky procedures). GEST je schopen pracovat i s fuzzy logikou či s faktorem určitosti (anglicky Certain factor), stejně jako DYNACLIPS i GEST dává možnost použít tzv. tabulovou architekturu (blackboard architecture).

Jedním z dalším neplacených nástrojů je BABYLON. V tomto případě se ale jedná o modulární, konfigurovatelné a hybridní prostředí pro vývoj expertních systémů. Podobně jako GEST umožňuje opět reprezentaci znalostí v několika formách. Vedle pravidel a rámců je schopen ještě pracovat s logikou (formou Prologu). Babylon vyžaduje pro svůj běh Common LISP. Posledním

členem této skupiny je ES – nástroj pro vývoj expertních systému, který podporuje jak backward tak forward chaining. Dále je možné v jeho podání pracovat s fuzzy logikou.

1.4.2 Placené nástroje

Mnohem větší skupinu tvoří komerční nástroje pro tvorbu expertních systémů [2]. Některé z níže jmenovaných již dále nejsou vyvíjeny a nejsou už ani podporovány. Namátkou bych jmenoval rodinu nástrojů EXSYS, které umožňují reprezentaci znalostí formou rámců nebo pravidel. Dále umožňují pracovat s blackbordingem nebo s fuzzy logikou. Navíc je možné konkrétně EXSYS professional napojit na SQL rozhraní. Další členové rodiny EXSYS (EXSYS RuleBook) umožňují tvorbu expertních systémů pomocí stromových diagramů či dokáží zakomponovat neuronové sítě (EXSYS Linkable Object Modules).

Nástroj ADS (Aion Development System) podporuje backward/forward chaining. Dále umožňuje objektově orientovanou reprezentaci znalostí či grafické rozhraní. Jeho funkce mohou byt volány z ostatních jazyků (C, Pascal…).

Nexpert Object je nástroj s grafickým uživatelským rozhraním, který využívá rozhodovacího mechanismu na bázi pravidel (anglicky Rule base), ale i na bázi objektů (anglicky Object base). Zajímavé je, že umožňuje propojení s databází a ostatními aplikacemi.

Informace z podkapitol 1.1 až 1.3. jsem převzal z [1], kde je současně možné nalézt více podrobností k problematice expertních systémů. Podkapitolu 1.5 jsem zformuloval a napsal na základě informací dostupných na [2], kde je současně možné najít více nástrojů.

2 CLIPS

Informace obsažené v podkapitolách 2.1 až 2.3 jsou čerpány z [3].

2.1 Historie CLIPS

Historie CLIPSu [2] začala v roce 1984. Tou dobou hledala NASA způsob jak integrovat expertní systém pro své účely tak, aby jej bylo možné bez problémů zakompilovat do běžně užívaných procedurálních jazyků, jako je třeba C nebo FORTRAN. Ačkoli prvně zvažovali užití Common LISPu ve spojení s nástroji pro překlad do jazyka C, byla tato řešení příliš nákladná a neefektivní. Proto se NASA rozhodla vytvořit vlastní řešení. Prototyp CLIPSu vyšel v roce 1985.

NASA tedy vytvořila nástroj na bázi jazyka C, který nebylo problémem integrovat do jakéhokoli kódu v tomto jazyce.

Verze 1.0 prokázala proveditelnost prvotního nápadu vytvořit relativně nízko nákladový nástroj na tvorbu expertních systémů. Sice zprvu byl CLIPS vyvíjen jako nástroj pro trénink expertů, ale rok po dalším vývoji, někdy v zimě roku 1986, byl CLIPS vylepšen přenosností, zvýšením výkonu, rozšířením funkcionality a hlavně dokumentací.

V létě roku 1986 vyšla verze 3.0, která již byla šířena pro použití mimo NASA. Tato verze již nabízela užitečný nástroj pro tvorbu expertních systémů. Verze 4.0 a 4.1 byly opět ve znamení navyšování výkonnosti a integrace do ostatních jazyků. To bylo léto roku 1987.

CLIPS se ve verzi 4.2 dočkal významného vylepšení, kdy byl celý přepsán do modulární podoby. Dále byl vydán manuál popisující architekturu CLIPSu. Dále byl vydán nástroj pro validaci a verifikaci programů na bázi pravidel (anglicky Rule-based).

Verze 4.3 v létě 1989 opět přidávala další funkcionalitu. Dalším významným zlomem byla verze 5.0 na jaře roku 1991, kdy přibyla dvě nová programovací paradigmata. Jednalo se o procedurální a objektové paradigma. CLIPSu, který využíval objektově orientované paradigma, se říkalo COOL (CLIPS Object-Oriented Language).

Verzí 5.1 na konci roku 1991 přidává NASA podporu pro MS-DOS, X Window a Macintosh. Na jaře roku 1993 je vydána verze 6.0, která již plně integruje pravidlově-objektové rozpoznávání. Dalším významným skokem je verze 6.1 vydaná v roce 1998, která odstraňuje podporu pro starší verzi překladače jazyka C (non-ANSI) a přidává podporu pro překladače jazyka C++. Dále přidává uživatelsky definované funkce.

Poslední stabilní verze 6.24, vydaná na jaře roku 2002, přidává vylepšení rozhraní pro Windows XP a Mac OS. Současně přidává podporu pro práci ve více prostředích (instancích expertního systému). V tuto chvíli je nástroj určen pro volné nekomerční využití a NASA s jeho vývojem již skončila.

2.2 Programovací paradigmata v CLIPS

CLIPS podporuje tři paradigmata [3] pro programování expertních systémů. Paradigma založené na bázi pravidel (rule-based), dále pak paradigma založené na objektech (object-oriented) a nakonec klasické procedurální paradigma.

Programování založené na bázi pravidel umožňuje reprezentování znalostí heuristickou cestou nebo na základě zkušeností (anglicky rule of thumb), které specifikuje množinu akcí, jež budou provedeny na základě dané situace.

Objektově orientované paradigma CLIPSu umožňuje modelovat komplexní systémy jako modulární komponenty. Tyto komponenty mohou být později znovu užity při modelování dalších systémů, nebo k vytváření nových komponent.

Procedurální paradigma v CLIPSu umožňuje psát části expertního systému podobně, jako kdybychom je psali v jazyku C nebo v Javě. Tento typ programování se dá zvláště použít ve funkcích, které je možné integrovat do pravidel.

2.3 CLIPS přenosnost a integrace

Jelikož je celý CLIPS napsán v jazyce C, je celkem dobře přenositelný. Jediné, co je potřeba, je kompilátor. CLIPS byl v minulosti testován na různých operačních systémech, jako jsou Windows XP, Mac OS nebo Unix. Vzhledem k tomu, že je tento nástroj napsán v jazyce C, vyniká také zkompilovaná binární podoba svou rychlostí.

Celý CLIPS je možné stáhnout formou zdrojových kódů, a proto není problém jej zakomponovat a upravit přímo na míru požadavkům uživatele. Další vlastností je možnost zaintegrovat CLIPS do procedurálních jazyků jako je C, JAVA nebo dokonce FORTRAN a ADA.

2.4 Formy užití CLIPSu

První forma užití je konzole. Podobně jako Prolog nebo LISP má i CLIPS svou konzoli. Tuto konzoli je možno používat jako příkazovou řádku reagující na příkazy v jazyce CLIPS. Lze zde vytvářet libovolné konstrukce, které jsou popsány níže v kapitole "Dostupné konstrukce v CLIPSu verze 6.3". Současně lze ale konzoli použít i pro jednoduché logické příkazy. Tuto část plně dokumentuje on-line manuál nazvaný "Basic Programming Guide", někdy též nazývaný lidmi z oboru zkráceně "BPG".

Druhou formou, jak užít CLIPS, je využívání přímo jeho API. Tato varianta se hodí zejména v okamžiku, kdy nechceme používat pouze konzoli coby doplněk uživatele, ale kdy chceme využít služeb CLIPSu v našem programovém řešení. Pokud volíme druhou variantu, pak ze stránek poskytovatele CLIPSu nebudeme stahovat binární instalační balíček, ale použijeme čistě zdrojové kódy, které můžeme kompilovat překladačem jazyka C. O tom, jak vypadá API pro CLIPS, se dočteme v manuálu nazvaném "Advance Programming Guide", někdy též "APG".

2.5 Dostupné konstrukce v CLIPSu verze 6.3

CLIPS má celkem šest konstrukcí. Tyto konstrukce lze sice využít i pro konzolové užití, ale jejich hlavní výhodou, která byla současně použita i v této práci, je fakt, že tyto konstrukce se dají

zakompilovat přímo do programu. O tom, co to znamená pro uživatele CLIPSu a jak je toho docíleno, je podobněji napsáno níže v kapitole "CLIPS, jeho kompilace a užití v uživatelských programech". Obecně o konstrukcích v CLIPSu lze říci, že jsou stavebními prvky pro tvorbu expertních systému.

2.5.1 deftemplate

První taková konstrukce se nazývá `deftemplate`. Tato konstrukce znázorňuje v expertním systému šablonu pro danou skutečnost. Vezmeme-li v potaz, že skutečnost je informace, nebo lépe řečeno fakt z vnějšího světa, se kterým se přicházíme za expertním systémem pro radu, je potřeba ji předat nějakou přesně definovanou formou. `Deftemplate` si můžeme představit jako množinu hodnot (slotů).

Tyto sloty mohou mít dva typy. Buď se jedná o sloty, které nesou jednoduchou hodnotu, jako třeba celé nebo desetinné číslo, potažmo řetězec či adresu, nebo instanci objektu (využíváme-li objektového paradigmatu). Ale vždy je to pouze jedna hodnota. Tomuto typu se v CLIPS říká "single slot". Druhým typem je tzv. "multislot". Jedná se o slot, který si můžeme představit jako pole hodnot stejného typu. Typy těchto jednotlivých hodnot jsou stejné jako v případě single slotu.

```
(deftemplate <název-šablony> [<volitelný-komentář>]
      <definice-slotů>*
)

<definice-slotů>::= (slot <název-slotu>) | (multislot <název-slotu>)
```

obr. 2.1 Obecný formát pro konstrukci deftemplate

2.5.2 defglobal

`Defglobal` je konstrukce, která umožňuje dát jménu hodnotu. Toto jméno je pak vidět ve všech ostatních konstrukcích uvnitř používaného prostředí. Jedná se tedy o globální konstantu. V jazyce C bychom přirovnali `defglobal` k direktivě preprocesoru `#`define.

```
(defglobal [<název-modulu>] <globální-přiřazení>*)

<globální-přiřazení>::= <globální-proměnná> = <výraz>
<globální-proměnná>::= ?*<symbol>*
```

obr. 2.2 Obecný formát pro konstrukci `defglobal`

2.5.3 deffunction

Konstrukce `deffunction` umožňují v CLIPSu, stejně jako funkce v jiných jazycích, zapouzdřit často se opakující kód. Funkce v CLIPSu mohou a nemusí obsahovat vstupní parametry a

stejně tak mohou a nemusí vracet hodnoty. Hodnoty vrácené funkcemi mohou představovat logické hodnoty jako `true` nebo `false`, ale stejně tak mohou vracet i hodnoty číselné.

```
(deffunction <název> [<komentář>]
      (<standardní-parametr>* [<wildcard-parametr>])
      <akce>*
)

<standardní-parametr>::= <jednočlenná-proměnná>
<wildcard-parametr>::= <mnohočlenná-proměnná>
```

obr. 2.3 Obecný formát pro konstrukci `deffunction`

2.5.4 defrule

Pravděpodobně nejdůležitější konstrukce pro tvorbu expertního systému. Zatímco bez šablon, funkcí a globálních konstant bychom se obešli, konstrukce `defrule` je konstrukce pro pravidlo. Jedna konstrukce `defrule` odpovídá přesně jednomu pravidlu v expertním systému.

Každé pravidlo je složeno ze dvou částí. První část je tzv. levá strana pravidla (anglicky označována jako Left-hand side nebo zkratkou LHS). Tato část je povinná, protože nese v sobě podmínky pro spuštění pravidla. Čili v okamžiku, kdy jsou všechny podmínky splněny, je toto pravidlo zařazeno na agendu možných spuštění pro daný fakt.

Druhou částí je tzv. pravá strana pravidla (anglicky označována jako Right-hand side nebo také RHS). Tato část se spustí až v okamžiku, kdy vyhodnocovací mechanismus (inference engine) rozhodne o spuštění daného pravidla. To může být dáno několika věcmi. Dané pravidlo má odpovídající prioritu (vyšší než ostatní) anebo má větší bodové skóre (počet podmínek potřebných pro zařazení pravidla na agendu je větší než u ostatních pravidel).

```
(defrule <název-pravidla> [<komentář>]
      [<deklarace>]                          ;vlastnosti pravidla
      <podmínkové-elementy>                  ;levá strana pravidla (LHS)
      =>
      <akce>*                                ;pravá strana pravidla (RHS)
)
```

obr. 2.4 Obecný formát pro konstrukci `defrule`

2.5.5 deffact

Jedna z variant, jak můžeme vložit do expertního systému skutečnosti, je právě konstrukcí `deffact`. Tato konstrukce vytváří pojmenovanou obálku nad fakty. Těch v dané konstrukci (aby měly nějaký význam) může být od jednoho až po nespočet. CLIPS totiž pracuje se všemi vloženými fakty současně a to tak, že se snaží dosáhnout spuštění některého z pravidel jejich kombinací.

Vložíme-li v konstrukci `deffact` třeba dvě skutečnosti, které ovšem samy o sobě nejsou schopny žádné pravidlo aktivovat, tak se může stát, že dohromady už nějaké pravidlo spustit mohou.

```
(deffacts <název> [<komentář>]
      <skutečnost>*
)
```

obr. 2.5 Obecný formát pro konstrukci `deffact`

2.5.6 defmodule

Poslední konstrukce nazvaná `defmodule` umožňuje zavést do CLIPSu modulární architekturu. Tato konstrukce je vhodná zejména, když se snažíme rozdělit jeden expertní systém do více modulů. Základní modul, který je vždy přítomný, se nazývá MAIN. V této práci nebylo využito práce s moduly, neboť postačil jeden modul, který nese všechna pravidla, globální konstanty a funkce.

2.6 CLIPS, jeho kompilace a užití v uživatelských programech

Na počátku kapitoly o CLIPSu bylo zmíněno, že tento nástroj je dobrý zejména pro jeho snadné zaintegrování do programů uživatele. Tato kapitola tedy popisuje, jak dosáhnout překladu a integrace konstrukcí expertního systému do programu psaném v jazyce C. Ke kompilaci konstrukcí do jazyka C bude potřeba nainstalovaná konzole CLIPSu. Tuto konzoli budeme používat pro zavedení konstrukcí expertního systému. Výhodou tohoto řešení je, že při zavádění konstrukcí do CLIPSu jsou všechny příkazy (celé konstrukce) kontrolovány syntaktickým analyzátorem CLIPSu. Z toho vyplývá, že vám CLIPS nedovolí později zakompilovat do vašeho programu syntakticky špatné konstrukce.

Další nespornou výhodou je testování vašich konstrukcí. Zejména mám na mysli sémantický význam pravidel, neboť když zavedeme do CLIPSu kompletní sadu konstrukcí vašeho expertního systému a pak si vytvoříme skutečnosti uživatelem vytvořené tak, aby odpovídaly konkrétním pravidlům, velice rychle zjistíme, zda expertní systém zareaguje dle našich očekávání. Následující řádky popisují postup pro překlad konstrukcí v CLIPSu na struktury jazyka C.

2.6.1 Postup překladu konstrukcí v CLIPSu na struktury jazyka C

Zavedeme do CLIPSu všechny konstrukce, které chceme v programu používat (`deftemplate`, `defglobal`, `deffunction`, `defrule`, …). Pokud se tento krok obejde bez komplikací, máme v CLIPSu zaveden náš expertní systém.

Aby mohl překlad konstrukcí začít, musíme zapnout ještě dynamickou kontrolu konstrukcí, které můžeme do běžícího programu vkládat. Kontrolu zapneme příkazem

```
(set-dynamic-constraint-checking true)
```

čímž umožníme CLIPSu kontrolu nově vkládaných částí a jejich návaznost na části již zakompilované do programu.

Po zapnutí kontroly již můžeme započít s vlastní kompilací. Tu provedeme příkazem

```
(constructs-to-c název_obrazu číslo_obrazu)
```

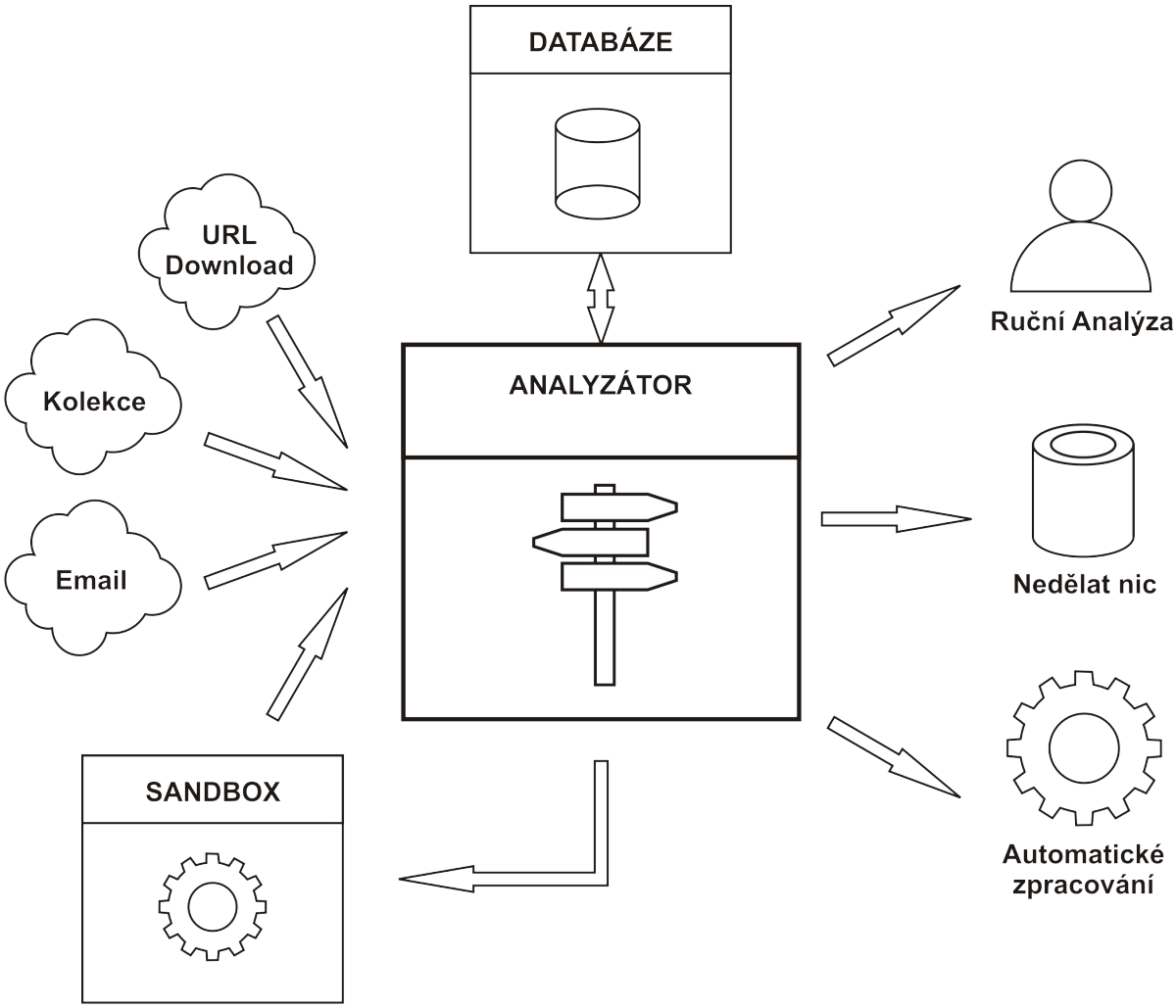
kde název_obrazu je prefix souborů, které budou vytvořeny a obsaženy ve strukturách jazyka C. Tento prefix je doporučeno volit maximálně o délce tři písmena, ale jen pro případ použití v operačním systému MS-DOS, jinak je velikost libovolná. číslo_obrazu je číslo, které použijeme jako součást volání funkce pro zavedení obrazu do programu.

Pokud se překlad a generování souborů do jazyka C podařilo bez problémů (předchozí příkazy proběhly bez komplikací), tak upravíme vlastní CLIPS pro běh mimo konzoli. Toho dosáhneme zapsáním hodnoty 1 pro flag RUN_TIME v hlavičkovém souboru setup.h.

Poslední věc je integrace našich konstrukcí do zdrojových souborů v CLIPSu. Pravděpodobně nejlepším řešením je přepsat celou funkci main. Postupy, jak upravit funkci main, jsou sepsány v pokročilém programovacím manuálu ("APG") na straně 154 v kapitole „Creating a CLIPS run-time program“. Tato publikace se jimi již dále nebude zabývat. Jistou inspiraci lze hledat i ve zdrojových kódech této práce.

3 Analýza současného řešení systému pro zpracování virových vzorků

3.1 Analýza prostředí současného analyzátoru



obr. 3.1 Situační nákres systému pro zpracování vzorku

3.1.1 Zdroje vzorků pro analýzu.

3.1.1.1 Zdroje vzorku v e-mailu

Největším zdrojem vzorků pro analyzátor jsou e-maily obsahující vzorky. Skupina e-mailů zahrnuje hned několik menších podskupin. Největší podskupinou jsou "mass virus center", což jsou

servery sdružující antivirové produkty. Na tyto servery jsou zaslány vzorky od uživatelů, kteří je chtějí prověřit. Na oplátku za poskytnutí produktů od antivirových společností jsou jim nedetekované vzorky zasílány z center na analýzu. Z těch nejznámějších můžeme jmenovat třeba Virus Total. Ovšem kvalita těchto vzorků není příliš vysoká, neboť je zde poměrně velké procento poškozených, čistých či jinak nezajímavých souborů. Druhá poměrně velká skupina jsou partneři AVG Technologies s.r.o. Kvalita těchto vzorků je větší, protože tyto exempláře již přicházejí celkem předtříděné, proto zde nenajdeme velké množství poškozených souborů. V poslední řadě jsou zdrojem e-mailů uživatelé antiviru AVG, ale to je v poměru k celkovému množství přijatých zpráv relativně malý počet.

3.1.1.2 Zdroje vzorků z URL

Druhou skupinou tvořící zdroje pro analýzu jsou adresy URL. Tyto adresy jsou procházeny v nekonečných smyčkách. Inkriminované URL mohou pocházet krom vlastních firemních zdrojů i od zákazníků využívajících antivirus AVG.

3.1.1.3 Zdroje vzorků formou kolekcí

Třetí skupina zdrojů jsou kolekce. Jedná se o sestavy různých vzorků, které přicházejí od různých konkurenčních antivirových společností, jako je třeba Kaspersky, Microsoft nebo Eset. Tyto kolekce se posílají v různých časových intervalech a obvykle jsou to velké objemy dat.

3.1.1.4 Zdroje vzorků pocházející ze sandboxu

Čtvrtou a současně poslední skupinou je sandbox. Sandbox sice není tak zcela zdrojem vzorků, jako spíš instancí, která nám může o vzorku něco málo napovědět. Jedná se totiž o emulátor, ve kterém je vzorek spuštěn. Pokud tedy analyzátor rozhodne o potřebě pustit vzorek do emulátoru a získat tak další informace pro vzorek, začíná nové kolečko a do analyzátoru již vzorek přichází ze sandboxu s novými informacemi. Svým způsobem se tedy jedná o jistou formu vstupu.

3.1.2 Výstupy analyzátoru

3.1.2.1 Manuální zpracování

Tento typ zpracování je využit, když není jistota, že je vzorek virus nebo virem napadený. Sice existují indicie, které říkají, že vzorek je podezřelý, ale na to, aby se na něj udělala definice bez přispění člověka, je to málo. Vzorky s tímto výstupem putují na oddělení virové analýzy, kde jsou prověřeny specialisty. Na jejich úsudku a odbornosti pak záleží, je-li je vzorek přidán do další aktualizace či nikoli.

3.1.2.2 Automatické zpracování

Je-li výstup z analyzátoru kód značící automatické zpracování, pak je jisté, že tento vzorek je virus, nebo je virem napaden. Současně proto, aby mohl být vzorek zařazen do aktualizace. Dále je potřeba, aby existovalo jméno kmene, pod jakým se tento vzorek bude zařazovat do aktualizace.

3.1.2.3 Sandbox

Tato varianta připadá v úvahu, když o daném vzorku není dostatek informací pro rozhodnutí, avšak analyzátor si myslí, že tento vzorek není zcela v pořádku. V takovémto případě putuje do emulátoru. Poté, co se daný vzorek zpracuje emulátorem, je zařazen zpět do fronty na analýzu analyzátorem.

3.2 Význam analyzátoru vzorků

Hlavním významem analyzátoru je tedy provádět rozhodnutí, kam daný vzorek poslat a tím efektivně směrovat proud příchozích vzorků. Vzhledem k počtu příchozích vzorků, které jsou diskutovány v kapitole "Statistiky", je celkem nepředstavitelné, že by taková komponenta chyběla.

Z praktického hlediska není možné všechny vzorky směrovat na ruční analýzu, neboť provádět ruční analýzu nad takovým množstvím souborů by bylo nesmírně časově náročné, nehledě na fakt, že v určitých případech rozhoduje právě rychlost reakce na novou hrozbu. Další aspekt je finanční náročnost, neboť by počet specialistů virové laboratoře musel být obrovský, aby byla zachována efektivita.

Stejně tak není možné všechny příchozí soubory přidávat do aktualizace, aniž by je někdo analyzoval, protože antivirus, který často detekuje čisté soubory (anglicky false), je zcela nepoužitelný, nemluvě o velikosti takové virové databáze.

3.3 Analýza programu VCB a následná volba integrace expertního systému

3.3.1 Stav před zavedením CLIPSu

Předchozí program nazvaný VCB byl celý napsaný čistě v jazyce C. Z toho vyplývá, že pro implementaci pravidel, které rozhodují o směrování vzorků, byly použity konstrukce jazyka C. V tomto případě byla pravidla implementována jako jeden velký blok konstrukcí if a else if.

Tento způsob má jednu poměrně velikou nevýhodu. Při potřebě rozšířit nebo upravit pravidla dle nových skutečností musíme počítat se sekvenčním přístupem k podmínkám. Program je

vyhodnocuje postupně, čili v tomto případě skutečně záleží na pořadí, kam jsou nové podmínky integrovány. Při úpravě některé ze stávajících podmínek musíme opět dbát na to, abychom rozšířením podmínky neudělali příliš velkou „díru“, kudy by proteklo příliš mnoho vzorků, jinými slovy, abychom dbali na správné pořadí.

Z předchozího odstavce tedy vyplývá, že rozšiřitelnost této struktury je obtížná. Na druhou stranu má tento způsob i výhody. Jelikož se počítá se sekvenčním přístupem k pravidlům, není nutné každé pravidlo uvádět v plném znění a můžeme se spolehnout na to, že dílčí podmínky mohou být obsaženy výše v kódu. Není dokonce ani problém, vytvořit na konci této konstrukce poslední sběrné pravidlo, které rozhodne, co se vzorky, které se nezachytily v předchozích pravidlech.

3.3.2 Výhody plynoucí ze zavedení expertního systému

Jelikož expertní systémy jsou komponenty, které na základě vstupní informace vydají názor plynoucí z jejich znalosti problémové domény, jsou současně dobrými adepty na integraci do našeho analyzátoru. Téměř bych řekl, že je to přesně to co je požadováno po tomto analyzátoru.

Vstupní fakt, který předkládáme expertnímu systému, je porovnáván se všemi pravidly, které jsou v systému zaintegrované. Principiálně, i když ke kontrolám nad pravidly dochází postupně, ve skutečnosti je ale vstupní fakt porovnáván znovu od začátku nad celým pravidlem. Není tu tedy nutná žádná návaznost mezi pravidly. Z toho důvodu odpadá sekvenčnost, jaká byla v předchozí verzi, a tedy nutnost myslet na to, jestli daná podmínka už existuje někde výše v kódu. Rozšiřitelnost pravidel je tedy podstatně jednoduší a dovoluje tak jednodušeji tvořit nová pravidla či upravovat stará.

Ovšem má to i své nevýhody. Zde již není možné spoléhat na to, že některé vzorky již byly směrovány, jak tomu bylo v programu VCB. Každé pravidlo v expertním systému musí mít tedy plné znění, což zvyšuje jejich složitost. Nicméně tato nevýhoda se odstraní testováním a za podpory expertů, kteří pravidla tvoří. Proto je zavedení expertního systému rozhodně lepší řešení.

3.3.3 Proč jsem zvolil právě CLIPS

Volba CLIPSu coby nástroje na tvorbu expertního systému, byla celkem jednoduchá. Prvním a současně asi nejdůležitější faktem byla přítomnost kolegy, který již s expertními systémy pracoval. Současně tento člověk přinesl zajímavé reference na CLIPS a hlavně příslib pomoci v okamžiku, kdy jsem se problematikou začal zabývat. Této pomoci jsem hlavně při tvorbě pravidel nejednou využil.

Vzhledem k prvotní nejistotě o úspěchu tohoto projektu bylo zcela vyloučeno, investovat finanční obnosy do komerčních a třeba i lepších nástrojů. Proto fakt, že CLIPS je volně ke stažení, napomohl k jeho pozdějšímu výběru.

Dalším faktem plynoucím z rozpravy s Ryanem Hicksem (kolega, který již s problematikou v tomto oboru pracoval) byl fakt, že CLIPS je nástroj spolehlivý a že je skutečně možné jej využívat

pro praxi. A konečně jedním z posledních faktů, byla opravdu velmi dobrá dokumentace, kterou je možné stáhnout na stránkách projektu.

Vzhledem k situaci popsané v předchozích odstavcích byl CLIPS zvolen jako nástroj pro vytvoření expertního systému.

4 Návrh a implementace řešení nového analyzátoru

Na začátku, před tvorbou vlastního návrhu analyzátoru, jsem se rozhodl tento projekt navrhovat a následně implementovat jako dvě oddělené části. První částí bylo vše kolem expertního systému. Druhou část tvořilo následné zaintegrování zkompilovaných částí expertního systému do programu a tím pádem tvorba vlastní programové části.

Tento postup byl zvolen vzhledem ke skutečnosti, že tou dobou jsem měl již nastudované, co přesně CLIPS je a jak se s ním dá pracovat, abych byl schopen vytvořit fungující expertní systém. Co se programové části týče, zde jsem měl poměrně velkou jistotu a nějaké zkušenosti v programovacích jazycích C/C++. Vzhledem k důležitosti těchto dvou částí jsem pro každou z nich ve zprávě vyčlenil vlastní kapitolu.

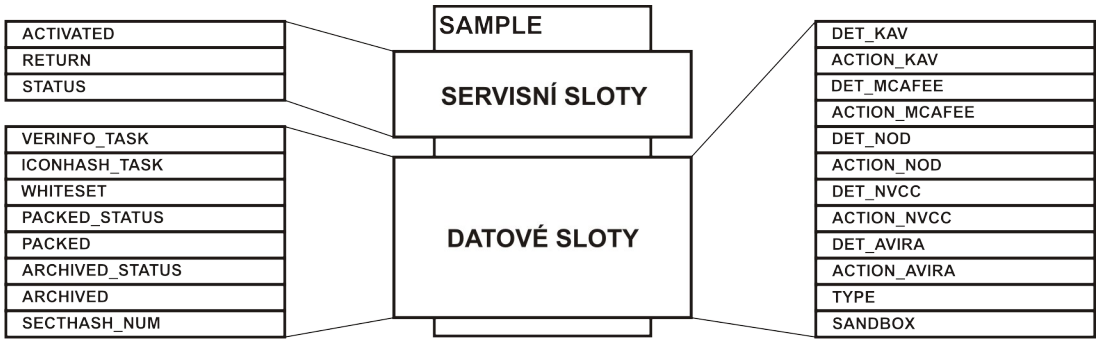
5 Návrh a implementace části expertního systému

Jak již popisují mnohé předchozí odstavce CLIPS má velice obsáhlou dokumentací, která velmi dobře popisuje možnosti pro tvorbu expertního systému. Pro návrh této části mi byl největší inspirací a pomocníkem "Basic Programming Guide“ ("BPG"), který obsahoval popis využitelných konstrukcí v CLIPSu.

5.1 SAMPLE, definice vzorku pro CLIPS

První otázka při návrhu zněla, jak dostat do expertního systému fakt z vnějšího světa. Tedy přesněji, jak by měla vypadat konstrukce se skutečností, která bude expertní systémem zpracovatelná. Z VCB jsem měl velice dobrou představu, jaká data by měla tato zatím neznámá konstrukce

obsahovat, ale až BPG mi nabídl variantu, kterou jsem nakonec využil. Byla jí šablona (konstrukce CLIPSem nazvaná deftemplate). Tuto šablonu jsem pojmenoval SAMPLE a stala základní jednotkou, která obsahuje informace o vzorku, který bude do expertního systému vkládán.



obr. 5.1 Grafické znázornění deftemplate SAMPLE

SAMPLE má z pohledu využití dva typy slotů. Prvním typem slotů jsou ty datové. Jedná se o položky, které nesou data vypovídající o vzorku. Význam těchto slotů je popsán níže. Druhou skupinu slotů nazývám servisní. Jedná se položky, které mají za úkol mimo jiné držet návratovou hodnotu dle spuštěného pravidla, nebo zaručit, že nebude spuštěno více, než jedno pravidlo. Popis i těchto slotů následuje níže v podkapitole „SAMPLE, význam jednotlivých slotů“.

5.2 SAMPLE, význam jednotlivých slotů

Tato kapitola popisuje význam slotů využitých v šabloně SAMPLE.

5.2.1 Datové sloty

Datové sloty byly převzaty z programu VCB, takže ve své podstatě nepodléhaly návrhu. jejich přítomnost byla vynucena zkušeností z předchozího analyzátoru, avšak jejich současná podoba byla upravena, tak aby co nejlépe vyhovovala potřebám a efektivitě pravidel.

5.2.1.1 Slot DET_*

Jedná se o single sloty, které nesou řetězce s názvy detekcí ostatních antivirů. V analyzátoru totiž mimo jiné bereme v potaz, jak se k příchozímu vzorku staví konkurenční antivirové produkty, a jelikož na těchto poznatcích jsou založena i pravidla, je nutné mít k dispozici názvy detekcí.

V tuto chvíli pracuje analyzátor s celkem pěti detekcemi konkurenčních antivirů. Jedná se o produkty Kaspersky antivirus (slot s jeho případnou detekcí se jmenuje DET_KAV), Norton antivirus

(slot DET_NVCC), Nod32 (slot DET_NOD), McAfee antivirus (slot DET_MCAFEE) a antivirus od Aviry (slot DET_AVIRA).

5.2.1.2 Slot ACTION_*

Opět se jedná o single slot, tentokráte celočíselného typu, který obsahuje hodnotu akce, která uvádí, jaký je postoj ke vzorku na základě názvu detekce. Momentálně se v databázi nachází sedm akcí s následujícími významy:

- akce 1 - udělej automatický crc
- akce 2 - pošli na ruční analýzu
- akce 3 - nechtěný soubor
- akce 4 - ignoruj tento vzorek
- akce 5 - ignoruj pouze tuto detekci
- akce 6 - ignoruj tento vzorek, ale ulož ho pro pozdější zkoumání
- akce 7 - udělej common crc nad nerozbaleným souborem

Důležitý je fakt, že ačkoli z významu jednotlivých akcí může být na první pohled jasno, kam by se měl daný vzorek zaslat, hlavní slovo mají až pravidla, neboť ta jsou mnohem komplexnější a snaží se tak situaci posoudit nejen na základě akcí. Akce pro určité detekce jsou tedy jen dílčím ukazatelem, co se bude s daným vzorek dít ve skutečnosti.

Jelikož jsou akce vázány na detekce (když není detekce, není akce, ale může se stát, že když je detekce, nemusí být na ni akce), tak opět existuje uvnitř SAMPLE pro každý konkurenční antivirus právě jeden single slot.

5.2.1.3 Slot TYPE

Jedná se o celočíselný single slot, který nese informaci typu vzorku. Opět je, podobně jako v případě akcí, i zde několik možností.

- typ 0 - neznámý
- typ 1 - soubor formátu PE
    - podtyp 1 - DLL
    - podtyp 0 - zbytek
- typ 2 - soubor formátu ELF
- typ 3 - skript
    - podtyp 1 - VBS soubor
    - podtyp 2 - JS soubor
    - podtyp 3 - BAT soubor
    - podtyp 4 - PHP soubor
    - podtyp 5 - HTML soubor

- podtyp 6 - Perl

Hodnota slotu TYPE je vždy vyplněná. Nikdy se nemůže stát, že by vzorek tuto hodnotu neměl.

5.2.1.4 Slot VERINFO_TASK

Single slot nesoucí celočíselnou hodnotu. Tato hodnota může reflektovat přítomnost určitého řetězce umístěného do resources daného vzorku. Pokud je tento řetězec nalezen uvnitř databáze, tak k němu existuje i číselná reprezentace určující, jaký na vzorek máme názor. Hodnoty a jejich popis jsou uvedeny na následujících řádcích:

- task 1 - soubor je čistý
- task 2 - nechtěný (unwanted), čili se vzorkem se dále pracovat nebude
- task 3 - nedělat crc, ale pošli rovnou na manuální analýzu

Opět stejně jako v případě slotů ACTION_* není tato hodnota vždy přímo aplikovatelná a je třeba užit komplexnějších pravidel. Navíc jen malý zlomek ze vzorků, které prochází analyzátorem, je odchycen pravidly na VERINFO_TASK. Z toho vyplývá, že tento slot může zůstávat prázdný.

5.2.1.5 Slot ICONHASH_TASK

Podobně jako v předchozím případě, i zde se jedná o informace z resources daného vzorku. Tentokrát se jedná o hash ze všech ikon, které jsou obsaženy v této sekci vzorku. Princip je prakticky úplně stejný jako v případě VERINFO_TASK, opět existuje databáze nesoucí na jedné straně tyto hash a na straně druhé jejich úlohy. Jejíchž znění je následující:

- task 1 - soubor je čistý
- task 2 - soubor nechceme detekovat
- task 3 - nedělat crc, ale poslat rovnou na manuální analýzu
- task 4 - podezřelé i když jinak vypadá v pořádku

5.2.1.6 Slot SECTHASH_NUM

Single slot nesoucí celočíselnou hodnotu. Tato hodnota je počtem souborů, které mají stejnou hash ze sekcí. Tato hash je zjišťována pouze u portable executable (PE) souborů, tedy všech těch, které mají typ 1.

Jen pro představu, co je to secthash, tak se jedná o md5 nad vybranými sekcemi uvnitř těchto binárních souborů. Pokud soubor není typu portable executable, pak není možné tuto hash vypočítat. Proto také tento slot v tomto případě, není naplněn.

5.2.1.7 Slot WHITESET

Whiteset je název pro rozsáhlou databázi, která nese seznam vybraných čistých souborů. Tento seznam se sestává zejména ze systémových souborů různých operačních systémů Microsoft

Windows a souborů obsažených v programu Microsoft Development Network (MSDN). Slot WHITESET nabývá tedy pouze dvou hodnot. Buď je daný vzorek obsažen v této databázi, a pak má slot hodnotu 1, nebo vzorek v databázi není, a pak má slot hodnotu `nil`.

5.2.1.8 Slot SANDBOX

Single slot obsahující názor emulátoru na daný vzorek. Tento bude mít některou z následujících hodnot jen v okamžiku, kdy projde emulátorem. Do té doby bude jeho hodnota `nil`. Vzorek prochází emulátorem pouze v případě, že si o to analyzátor sám požádá, což je vysvětleno v předešlé kapitole nazvané „Výstupy z analyzátoru“, konkrétně pak v podkapitole sandbox.

Hodnoty, které může tento slot obsahovat na základě výstupu ze sandbox:

- hodnota 0 – sandbox si myslí, že vzorek je čistý
- hodnota 1 – sandbox si myslí, že vzorek je virus

5.2.1.9 Slot PACKED_STATUS

Flag celočíselného typu v podobě single slotu, který pouze říká, jestli byl daný vzorek zabalen nějaký softwarem. Tento slot ovšem nic neříká o tom, kterým programem byl zabalen. Obvykle nabývá hodnot `nil`, není-li vzorek zabalen, nebo obsahuje hodnotu jedna v případě, že vzorek zabalen byl.

5.2.1.10 Slot PACKED

Jedná se o multislot, který obsahuje seznam programů (jedná se tedy o multislot s řetězcovými typy), kterými byl daný vzorek zabalen. Tento slot v případě, že vzorek zabalen vůbec nebyl, bude mít hodnotu `nil`. V opačném případě bude obsahovat názvy programů.

5.2.1.11 Slot ARCHIVED_STATUS

Podobně jako v případě PACKED_STATUS je i tento slot využíván jako flag pro určení, jestli je daný vzorek archivován některým z archivačních programů. Proto i zde platí fakt, že jestli vzorek archivován byl, bude zde hodnota jedna, jinak bude hodnota `nil`.

5.2.1.12 Slot ARCHIVED

Multislot nesoucí jména archivačních programů v případě, že vzorek archivován byl. V opačném případě bude jeho hodnota `nil`.

5.2.2 Servisní sloty

Tyto sloty, již podléhaly vlastnímu návrhu nového analyzátoru, neboť v původním programu VCB jich není třeba.

5.2.2.1 Slot ACTIVATED

Jedná se o single slot, který je využíván jako pojistka spuštění více pravidel. V některých případech se totiž může stát, že daný vzorek bude odpovídat více levým stranám pravidel, a tak by se je expertní systém pokusil spustit všechny. To je ovšem celkem nežádoucí v případě, že od expertního systému očekáváme jeden konkrétní výsledek. Proto byl zaveden slot ACTIVATED, který má za hodnotu symbol. Tento symbol má při spuštění expertního systému hodnotu nastavenou na `no`. Pokud je pravidlo umístěno na agendu a poté i spuštěno, změní jeho pravá strana hodnotu symbolu na `yes`. Jelikož ostatní pravidla ke svému spuštění vyžadují hodnotu symbolu na `no`, tak již dále žádné nespustí.

5.2.2.2 Slot RETURN

Jedná se o celočíselný single slot, který nese návratovou hodnotu po spuštění pravidlem. Tento slot je tu proto, abychom zjistili z programové části, jaké pravidlo bylo spuštěno a dále pak reagovat na tuto skutečnost v programové části. Hodnoty pro návratové kódy z expertního systému jsou popsány souboru `defglobal.clp`, který současně nese všechny používané globální konstanty. Návratové kódy z CLIPSu mohou být následující:

- hodnota 0 – systém si nedokázal se vzorkem poradit, takže nebylo spuštěno žádné pravidlo
- hodnota 1 – nedělej nic
- hodnota 2 – nedělej nic, ale zjisti jestli vzorek není určen pro pozdější zkoumání
- hodnota 3 – zašli vzorek na ruční zpracování
- hodnota 4 – zašli vzorek do sandboxu pro získání více informací
- hodnota 5 – udělej nad vzorkem obecný common crc
- hodnota 6 – udělej ze vzorku crc a jméno kmene je z Kasperského detekce
- hodnota 7 – udělej ze vzorku crc a jméno kmene je z detekce McAfee
- hodnota 8 – udělej ze vzorku crc a jméno kmene je z detekce NOD32
- hodnota 9 – udělej ze vzorku crc a jméno kmene je z detekce Nortonu
- hodnota 10 – udělej ze vzorku crc a jméno kmene je z detekce Aviry
- hodnota 11 – udělej ze vzorku crc a jméno kmene vezmi z obecného kmene pro sandbox
- hodnota 12 – proveď common crc nad vzorkem a vezmi jméno kmene z obecného názvu pro balící programy
- hodnota 13 – proveď common crc nad vzorkem a vezmi jméno kmene z obecného názvu pro archivační programy
- hodnota 14 – proveď common crc nad vzorkem a jméno kmene je z detekce Kasperkého
- hodnota 15 – proveď common crc nad vzorkem a jméno kmene je z detekce McAfee
- hodnota 16 – proveď common crc nad vzorkem a jméno kmene je z detekce NOD32

- hodnota 17 – proveď common crc nad vzorkem a jméno kmene je z detekce Nortonu
- hodnota 18 – proveď common crc nad vzorkem a jméno kmene je z detekce Aviry
- hodnota 19 – proveď common crc nad vzorkem a jméno kmene vezmi z obecného kmene pro sandbox

5.2.2.3 Slot STATUS

Opět celočíselný single slot, který upřesňuje, jaký status bude mít vzorek po průchodu analyzátorem. Hodnoty jsou opět formou globální pojmenovaných konstant uložených v souboru `defglobal.clp` a mohou nabývat následujících hodnot:

- hodnota 0 – vzorek je i po průchodem analyzátoru neznámý
- hodnota 1 – vzorek je čistý
- hodnota 2 – vzorek je infikovaný
- hodnota 3 – tento vzorek nechceme detekovat
- hodnota 4 – ve zpracování a čeká na ruční analýzu
- hodnota 5 – vzorek je infikovaný a bude detekován v příští aktualizaci

5.3 Návrh a implementace pravidel analyzátoru

Po návrhu a následné implementaci šablony pro vzorek bylo již vše připraveno k tomu, abych začal navrhovat vlastní pravidla. Tato situace byla do jisté míry zjednodušena skutečností, že daná pravidla již existovala implementována v programu VCB. Konstrukce v jazyce C sice nijak nepomohly implementaci pravidel do CLIPS, ale přínos byl v tom, že pravidla již po významové stránce byla navržena. To umožňovalo podstatně rychlejší implementaci, protože v případě, že by neexistovalo nic z VCB, bylo by nutné, aby mi při návrhu asistoval člověk, který problematiku analýzy ovládá.

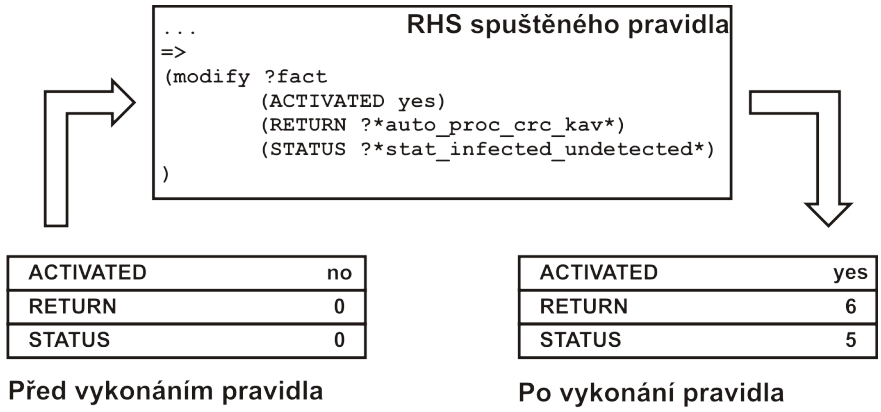
Problém který se během raných fází implementace dostavil, byl problém se sekvenčností. Jak již uvádí kapitola o výhodách a nevýhodách pojetí analyzátoru coby programu v jazyce C a v CLIPS, došlo na fáze, kdy byla nezbytná úprava pravidel tak, aby obsahovala své plné znění, tzn. aby pravidla počítala s podmínkami, které byly obsaženy v programu VCB nad nimi. Tato překážka byla postupem času odstraňována během testování, kdy se upravovala forma pravidel, tak aby co nejlépe odpovídala skutečnosti.

V době psaní této zprávy je v analyzátoru implementováno třicet dva pravidel. Některá z nich jsou pro názornost popsána níže v podkapitole "Ukázka pravidel v CLIPSu“, zbytek je možno nalézt v souboru `defrule.clp`.

5.3.1 Společná část pro všechna pravidla

Všechna pravidla mají konstrukčně stejnou pravou stranu, tj. část, která se provádí až v okamžiku, kdy je pravidlo spuštěno z agendy. Je dána celkem třemi řádky. První část nastavuje hodnotu slotu ACTIVATED z `no` na `yes`, čímž zabraňuje spuštění dalšího pravidla. Druhý řádek nastavuje hodnotu návratového kódu (slot RETURN), aby se v programové části dalo zjistit, které pravidlo bylo spuštěno. A konečně třetí řádek nastavuje hodnotu slotu STATUS, která vypovídá o tom, co si o daném vzorku myslí analyzátor. Všechny tyto změny jsou realizovány modifikací příchozí skutečnosti (příkaz `modify`), čili ve své podstatě se jedná o změnu tvaru příchozího faktu po spuštění pravidla.

Kromě těchto tří řádků na pravé straně pravidla existuje ještě jedna část na levé straně pravidla, která kontroluje fakt jestli, již bylo pravidlo aktivováno. Současně se skutečnost (fakt) navazuje na proměnnou, abychom ji byli později schopni změnit na pravé straně pravidla tak, jak to popisuje předchozí odstavec. Tato část kontroluje hodnoty servisních slotů vůči jejich původní hodnotě. Tzn. že je kontrolován slot ACTIVATED na hodnotu `no` a slot RETURN na hodnotu 0.



obr. 5.2 grafické znázornění změny servisních slotů
před a po spuštění pravidla

5.3.2 Ukázka implementovaných pravidel v CLIPSu

Tato podkapitola ukazuje a popisuje význam dvou pravidel v CLIPSu. Všechna ostatní pravidla implementovaná v analyzátoru jsou po syntaktické stránce velmi podobná. Ovšem stručný význam jednotlivých pravidel je popsán v podkapitole "Pravidla v analyzátoru – stručný přehled".

5.3.2.1 Pravidlo pro whiteset

První ukázkou je pravidlo zabývající se kontrolou, jestli daný vzorek je obsažen ve whiteset, čili jestli je daný prvek čistý. Jedná se současně asi o jedno z nejjednodušších pravidel implementovaných v analyzátoru.

```
(defrule whiteset_exist "check if sample exists in white set"     ;1
      ?fact <- (SAMPLE (ACTIVATED no)(RETURN 0))                  ;2
      (SAMPLE (WHITESET ?whiteset &: (neq ?whiteset nil)))        ;3
      =>                                                          ;4
      (modify ?fact                                               ;5
            (ACTIVATED yes)                                       ;6
            (RETURN ?*no_proc*)                                   ;7
            (STATUS ?*stat_clean*)                                ;8
      )
)
```

obr. 5.3 pravidlo pro existenci vzorku ve whiteset.

Konstrukce defrule v CLIPSu

Na obrázku jsou označeny jednotlivé řádky čísly. Je to z důvodu, aby se dalo na jednotlivé příkazy odkazovat. Řádek s číslem jedna vytváří konstrukci defrule, kterou je v CLIPSu označováno pravidlo. Tato konstrukce ještě obsahuje identifikátor pravidla, v našem případě `whiteset_`exists, který je pro každé pravidlo povinný. A nakonec ještě řádek obsahuje třetí část a tou je komentář. Tato část je volitelná, ale vzhledem vzhledem k faktu, že pravidla mohou být a obvykle jsou podstatně složitější, je lepší ji využít. Řádek s číslem dva provádí onu funkci kontroly, bylo-li pravidlo již spuštěno. Pokud tedy CLIPS vyhodnotí tuto skutečnost kladně, naváže na proměnou vstupní fakt, neboli náš sledovaný vzorek.

Řádek s číslem tři je jádrem tohoto pravidla. De facto říká, že se má CLIPS podívat do příchozího faktu, který by měl vypadat jako šablona SAMPLE a v ní najít slot WHITESET. Pokud je tento předpoklad správný, pak by měl tuto hodnotu navázat na proměnnou whiteset (otazník před názvem značí proměnnou). Je-li i tohle splněno, pak se může přistoupit k vyhodnocení, které říká vrátím pravdu, pokud je hodnota proměnné různá od `nil`, což ve své podstatě znamená, že je-li v programové části zjištěno, že vzorek není ve databázi čistých souborů, nebude se o tom faktu vůbec zmiňovat a tato hodnota bude tedy `nil` v rámci původního nastavení CLIPSu. V případě, že by vzorek skutečně v databázi byl, pak bude mít hodnotu jedna a toto pravidlo bude spuštěno.

Řádek čtyři je zde jako pomyslná hranice mezi levou stranou a pravou stranou. To co je nad tímto řádkem se musí rovnat příchozímu faktu, aby bylo dané pravidlo zařazeno na agendu. Vše ostatní, co se nachází pod tímto řádkem, se bude provádět, pokud pravidlo bude spuštěno.

Řádek pět je právě oním místem, kde se upravuje předkládaná skutečnost tak, aby nebylo možné spustit další pravidlo. Příkaz modify říká, že fakt bude upravován. Fakt je nyní navázán na proměnnou fact z prvního řádku.

Řádek šest ukazuje, na jakou hodnotu je měněn slot ACTIVATED. Je to hodnota `yes`, to protože tento fakt již aktivovalo některé z pravidel, a proto již není žádoucí, aby docházelo k dalším aktivacím.

Řádek sedm nastavuje novou hodnotu návratového kódu, která je uložena v konstrukcích defglobal. Hodnota má název `no_proc` a říká v tomto případě, že daný vzorek je pro další zpracování komponentami virového oddělení nezajímavý.

Řádek osm nastavuje do skutečnosti nový status pro tento vzorek. Je to opět utvořeno globální konstantou, která se v tomto případě jmenuje `stat_`clean, což znamená, že vzorek je považován za čistý.

5.3.2.2 Pravidlo pro sandbox

V kontrastu s předchozím pravidlem jsem se rozhodl, popsat v této zprávě jedno z nejsložitějších pravidel, která jsou v analyzátoru implementována. Je to pravidlo, v jehož důsledku je příchozí vzorek spuštěn v emulátoru. Vzhledem k předchozí ukázce jsem se rozhodl popsat pouze části, kterou jsou něčím zajímavé a hlavně které nejsou obsaženy v pravidle předchozím. Vzhledem k velikost pravidla jsem se jen rozhodl rozložit jej na tři sekce.

```
(defrule sandbox_require "require sandbox for more information"
       ;sekce 1
       ?fact <- (SAMPLE (ACTIVATED no)(RETURN 0))
       (SAMPLE (TYPE ?type &:(eq ?type 1)))                                    ;1
       (SAMPLE (ACTION_KAV ?a_kav))                                            ;2
       (SAMPLE (ACTION_MCAFEE ?a_mcafee))
       (SAMPLE (ACTION_NVCC ?a_nvcc))
       (SAMPLE (ACTION_NOD ?a_nod))
       (SAMPLE (ACTION_AVIRA ?a_avira))
       (test
              (not(f_unwanted_det ?a_kav ?a_mcafee ?a_nvcc ?a_nod ?a_avira))   ;3
       )
       ...
```

obr. 5.4 pravidlo pro požadavek na sandbox. 1.část

Začneme sekcí jedna. Řádek označen číslem jedna je podmínkou, jež požaduje, aby typ příchozího vzorku byl portable executable (formát PE). Řádek označený číslem dvě provádí navazování slotů jednotlivých akcí (sloty ACTION_*) konkurenčních antivirů na proměnné. To proto, abychom je poté mohli předat coby parametry do funkce, která je uvedena na řádku označeném sekcí tři.

Tento řádek zasílá hodnoty v proměnných do funkce na zjištění, jestli některá z nich neodpovídá nechtěné detekci. Znamená to, že kdyby jen jedna akce měla tuto specifickou hodnotu,

pak je celkem zbytečné toto pravidlo spouštět, neboť my tento vzorek vůbec detekovat nechceme. Čili vrátí-li se z funkce hodnota false (vstupní fakt nemá žádnou akci odpovídající nechtěné detekci), pokračuje CLIPS ve vyhodnocování pravidla.

```
...
;sekce 2
(SAMPLE (DET_KAV ?det_kav))                                                    ;1
(SAMPLE (DET_MCAFEE ?det_mcafee))
(SAMPLE (DET_NOD ?det_nod))
(SAMPLE (DET_NVCC ?det_nvcc))
(SAMPLE (DET_AVIRA ?det_avira))
(test
       (or                                                                     ;2
              ( <=                                                             ;3
                     (-
                            (f_num_detected ?det_kav ?det_mcafee ?det_nod ?det_nvcc ?det_avira)
                            (f_num_ignore ?a_kav ?a_mcafee ?a_nvcc ?a_nod ?a_avira)
                     )
                     ?*minimal_occurrence_number*
              )
              (f_no_unignorable_action ?a_kav ?a_mcafee ?a_nod ?a_nvcc ?a_avira)   ;4
       )
)
(or                                                                            ;5
       (SAMPLE (ARCHIVED_STATUS ?arch &:(eq ?arch nil)))
       (SAMPLE (ARCHIVED $?archived     &:(f_content_pattern "Thinstall" $?archived)
                                                  |:(f_archiver_1 $?archived))
       )
)
...
```

obr. 5.5 pravidlo pro požadavek na sandbox. 2.část

Druhá sekce (znázorněná na obrázku 5.5) obsahuje podmínky pro nejmenší počet neignorovaných detekcí a také podmínky pro případ, že by vzorek byl archivován. Řádek jedna ukazuje navazování jednotlivých názvů detekcí na proměnné.

Řádek dva zavádí příkazem or možnost shodovat se buď první blokem podmínek definovaným od řádku tři níže, nebo druhým blokem, který je na řádku označeném číslem čtyři.

Pro shodu na řádku tři je potřeba, aby rozdíl počtu detekcí (vracený funkcí `f_num_detected`) a počtu akcí s hodnotou pro ignoraci konkrétní detekce (vracený funkcí `f_num_ignore`) byl menší nebo roven minimálnímu nenulovému počtu výskytů detekcí, uváděným v globální konstantě `minimal_occurrence_number`. Jinými slovy je požadováno, aby počet neignorovaných detekcí nebyl větší než jedna.

Testování ovšem ukázalo, že je nutné ještě brát v potaz fakt, kdy na danou detekci neexistuje akce. V tomto případě pak je jedno, kolik takových detekcí existuje. A proto byla vytvořena funkce, která vrací `true` nebo `false` v závislosti na existenci takové detekce.

Posledním úsekem v této sekci jsou podmínky pro archivované vzorky, které začínají na řádku označeném pětkou. To je opět realizováno konstrukcí or, neboť máme opět několik možností, které jsou přijatelné, aby bylo pravidlo spuštěno. První z nich je, že vzorek vůbec archivovaný není. Druhou možností je, že vzorek je archivován aplikací Thinstall, což je zjištěno funkcí

f_content_pattern nad multislotem softwaru pro archivaci ARCHIVED. Třetí a poslední možností je fakt, že vzorek je archivován alespoň jednou aplikací pro archivaci definovanou funkcí f_archiver_1.

```
	...
	;sekce 3
	(SAMPLE (WHITESET ?white &:(eq ?white nil)))                                    ;1
	(SAMPLE (SANDBOX ?sandbox &:(eq ?sandbox nil)))                                 ;2
	(SAMPLE (PACKED_STATUS ?stat &:(neq ?stat nil)))                                ;3
	(SAMPLE (VERINFO_TASK ?verinfo &:(neq ?verinfo 1) &:(neq ?verinfo 2)))          ;4
	(SAMPLE (ICONHASH_TASK ?iconhash &:(neq ?iconhash 1) &:(neq ?iconhash 2)))      ;5
	=>
	(modify ?fact
		(ACTIVATED yes)
		(RETURN ?*sandbox_proc*)
		(STATUS ?*stat_unknown*)
	)
)
```

obr. 5.6 pravidlo pro požadavek na sandbox. 3. část

Třetí sekce, již pouze dotváří pravidlo pro odeslání vzorku do sandboxu. Proto bych ji shrnul ve stručnosti. Řádek jedna říká, že vzorek nesmí být v databázi whiteset. Řádek dva vyžaduje, aby vzorek ještě neměl žádný log ze sandboxu, což je logické vzhledem k tomu, že se ho toto pravidlo snaží do sandboxu zaslat. Řádek tři vyžaduje prostřednictvím podmínky, aby byl vzorek zabalen. Řádek čtyři spolu s řádkem pět požadují, aby hodnota úlohy pro verinfo a iconhash nebyla ani jedna, ani dva, což by znamenalo, že jsou buď čisté, nebo nechtěné.

5.3.3 Implementovaný pravidla v analyzátoru – stručný přehled

V této podkapitole je stručný popis významu jednotlivých pravidel, aby čtenář získal přehled o tom, jaká mohou existovat pravidla, zejména co všechno se sleduje pro rozhodování v rámci jednotlivých vzorků. Rozbor, jak byla v CLIPSu implementovaná, není obsahem této podkapitoly a ani této zprávy, ale v případě zájmu jsou všechna pravidla implementována v souboru defrule.clp.

5.3.3.1 Pravidla VERINFO

Rodina pravidel verinfo_noProc_clean a verinfo_noProc_unwanted, jejichž jádro je založeno na kontrole informací z verinfo sekcí.

5.3.3.2 Pravidla na ICONHASH

Rodina pravidel spouštěných na základě hodnoty iconhash task, iconhash_noProc_clean, iconhash_noProc_unwanted, iconhash_Arch_Or_noPE a iconhash_pe_noArch.

5.3.3.3 Pravidlo pro WHITESET

Pravidlo spuštěné v okamžiku, kdy je vzorek obsažen v databázi whiteset.

5.3.3.4 Pravidlo pro Archivované vzorky

Jedná se o pravidlo `archivers_1`, které je spuštěno v okamžiku, kdy je vzorek archivován alespoň jedním ze sledovaných programů pro archivaci. Dále pro spuštění tohoto pravidla je nezbytné, aby existovaly alespoň dvě neignorované detekce a alespoň jedna neignorovaná akce. Dále vzorek nesmí obsahovat nechtěnou detekci.

5.3.3.5 Pravidlo pro nechtěné detekce

Pravidlo se jmenuje `unwanted_det`. Obsahuje-li alespoň jedna detekce akci s hodnotou 3, čili nechceme detekovat tento vzorek, pak je spuštěno toto pravidlo. Mimo jiné vzorek nesmí být archivován, maximálně aplikací Thinstall. A musí jít o vzorek typu jedna, čili o portable executable formát.

5.3.3.6 Pravidla pro common crc

Tvoří pětici pravidel začínající názvem `common_crc_*`, kde místo hvězdičky je název konkurenčního antiviru. Každé z pěti pravidel pro jeho spuštění očekává hodnotu 7 v akci daného antiviru. U akcí ostatních antivirů je požadována absence akce s hodnotou 3 (nechtěné detekce). Mimo jiné je požadováno, aby existovala alespoň jedna neignorovaná detekce navíc. V případě, že je jedno z těchto pravidel spuštěno, bude vytvořen tzv. common crc, což je výpočet redundantního součtu nad celým vzorkem.

5.3.3.7 Pravidla pro automatické zpracování

Opět se jedná o pětice pravidel stejně jako v případě common crc pravidel, ale s tím rozdílem, že nyní je požadována hodnota akce jedna místo sedm. V tomto případě je potřeba pro splnění levé strany ještě ověřit, že vzorek není zabalen specifickou sadou softwaru definovanou ve funkci `f_packer_1`, dále vzorek nesmí být archivován (maximálně tak aplikací Thinstall). Stejně jako v předchozím případě, i tady je potřeba na spuštění mít další neignorovanou detekci z jiného antiviru.

Posledním přírůstkem ve vývoji této sady pravidel je podmínka, zda-li se nenachází v ostatních akcích i hodnota dva. To by potom znamenalo, že existuje někde požadavek na manuální zpracování. Pokud je někde požadavek na ruční zpracování, pak dostane přednost, neboť je z nějakého dobrého důvodu potřeba analýza specialisty.

5.3.3.8 Pravidla pro manuální zpracování

Sada pěti pravidel založena na principu předchozích dvou skupin. Opět je požadována určitá akce, tentokrát se jedná o hodnotu dva. Dále je potřeba mít další, alespoň jednu detekci z jiného antiviru. Samozřejmě žádná z akcí nesmí poukazovat na nechtěnou detekci, podmínky pro archivace jsou stejné jako v minulých sadách pravidel. A nakonec, i tady je požadován formát portable executable či absence vzorku ve whiteset.

5.3.3.9 Pravidlo pro SECTHASH

Celý název pravidla je `secthash_count` a pro jeho spuštění je potřeba mít alespoň 100 vzorků se stejnou secthash za posledních dvacet čtyři hodin. Dále je požadována alespoň jedna neignorovaná detekce a žádná z detekcí nesmí být nechtěná. Nakonec už zbývají jen podmínky pro archivaci a typ souboru zakončený absencí ve whiteset.

5.3.3.10 Pravidla pro SANDBOX

Jedná se o tři pravidla, která nějakým způsobem pracují se sandboxem, nebo z jejich spuštění vyplývá zaslaní vzorku do sandboxu. Prvním pravidlem je `sandbox_require`, které je již popsáno výše.

Dalším pravidlem pro sandbox je `sandbox_opinion_virus_crc`. Toto pravidlo se spustí v okamžiku, kdy existuje log ze sandboxu, který vzorek označuje jako virus. Důležité v tomto případě je, že výstupem tohoto pravidla je pouze crc ze specifických oblastí vzorku. To vyžaduje, aby vzorek nebyl archivován.

Třetím pravidlem spojeným se sandboxem je `sandbox_opinion_common`. Rozdíl tohoto a předchozího pravidla je v archivaci. Je-li toto pravidlo spuštěno, pak byl vzorek archivován některým z archivačních softwarů sledovaného funkcí `f_archiver_1`. Výstupem z pravidla je common crc, čili redundantní součet přes celý nerozbalený obsah.

5.3.3.11 Souhrnné pravidlo REMAIN

Implementace tohoto pravidla byla celkově hodně složitá, neboť se jedná o tzv. souhrnné pravidlo. Toto pravidlo se ve VCB nachází na konci else-if konstrukcí, čili v okamžiku, kdy žádné pravidlo není naplněno, spustí se právě toto pravidlo.

Samozřejmě v tomto pravidle jsou obsažené podmínky, takže se nejedná o něco jako část default ve switch konstrukci jazyka C. Jak již je zmiňováno v kapitole o sekvenčnosti, implementace souhrnu může být pro nesekvenční přístup obtížná.

Výsledná levá strana pro souhrnné pravidlo vypadá následovně. Proto aby bylo spuštěno postačí, aby existovala jedna neignorovaná detekce u konkurenčních antivirových produktů vyjma produktu od Aviry.

Další věc je fakt, že detekce u některých z těchto čtyř antivirů nesmí obsahovat určité podřetězce. Pro společnost McAfee jsou to podřetězce „New Malware“ nebo „Generix.dx“ a u Norton Antiviru se jedná o podřetězce „Suspicious_“ a „Trojan W32/Packed“.

Dále pak musí vzorek buď obsahovat log ze sandboxu, nebo nesmí být zabalen. V případě archivace, je vzorek možno mít nearchivovaný, archivovaný softwarem definovaným ve funkci `f_archiver_1` nebo archivovaný aplikací Thinstall. Nakonec pro spuštění postačí už jen absence ve whiteset a fakt, že žádná definice není nechtěná.

5.3.3.12 Pravidlo pro podezřelou ICONHASH a VERINFO

Ačkoli pravidla pro iconhash a verinfo byla již popsána na začátku, toto jedno pravidlo jsem do té skupiny nezařadil. Jedná se o pravidlo, které očekává, že iconhash nebo verinfo budou mít hodnotu tři, čili budou z nějakého důvodu podezřelé. Proto je výstupem z toho pravidla manuální zpracování.

Dále pro jeho spuštění je nutné mít alespoň dvě detekce a z toho aspoň jednu neignorovanou akci. Ostatní části, jako absence whiteset, nutnost, aby vzorek byl typu jedna, či archivace maximálně aplikací Thinstall, zůstávají stejné jako u ostatních pravidel.

5.3.3.13 Pravidlo pro zabalené soubory

Kromě existence alespoň dvou neignorovaných detekcí a jedné akce je potřeba, aby byl vzorek zabalen některým ze softwarů definovaných ve funkci `f_packer_1`. Výstupem z tohoto pravidla je automatický common crc nad nerozbaleným vzorkem. Zbytek pravidla se opět zabývá archivací, existencí ve whiteset a typem portable executable (hodnota jedna).

5.3.3.14 Pravidlo pro typově neznámé vzorky

Toto pravidlo je připraveno na spuštění v okamžiku, kdy vzorek má hodnotu typu nula. V takovém případě nevíme, co je vzorek zač. Takže se třeba může jednat i o poškozený soubor.

5.3.4 Funkce v části expertního systému

Pro často se opakující rozměrnější kód v CLIPSu bylo použití funkcí zcela jednoznačné. CLIPS umožňuje pro funkce použít konstrukci deffunction. Dále pak CLIPS zavádí i procedurální funkce, které se přibližují klasickým procedurálním programovacím jazyků. Jedná se zejména o cykly pro průchod vícehodnotových proměnných.

Tato kombinace byla zejména vhodná pro konstrukce funkcí, které procházejí seznamem detekcí nebo akcí a snaží se zjistit, jestli je daný vzorek alespoň někým detekován nebo jestli jsou akce ve vzorku chtěné či nechtěné.

Tento prvek v CLIPSu do značné míry zlepšil přehlednost pravidel. A dále umožnil plynule rozšiřovat seznamy různých softwarů. Podobně jako v pravidlech tato zpráva neobsahuje implementační detaily funkcí, ale pouze popisuje jejich význam pro rozhodování.

5.3.4.1 Funkce nesoucí seznam specifických balících aplikací

Tato funkce se jmenuje `f_archiver_1`. Podobně jako v předchozím případě se jedná o funkci, která dostane nula až n vstupních parametrů a snaží se je porovnat s názvy specifických archivačních aplikací. Pokud existuje shoda aspoň jednoho z nich, pak je vrácena hodnota `true`, jinak `false`.

5.3.4.2 Funkce nesoucí seznam specifických archivačních aplikací

Tato funkce se jmenuje `f_archiver_1`. Podobně jako v předchozím případě se jedná o funkci, která dostane nula až n vstupních parametrů a snaží se je porovnat s názvy specifických archivačních aplikací. Pokud existuje shoda aspoň jednoho z nich, pak je vrácena hodnota `true`, jinak `false`.

5.3.4.3 Funkce hodnotící akce

Jedná se o celkem tři funkce (`f_unwanted`, `f_manual`, `f_common_crc`). Každá z těchto funkcí očekává na vstupu multislot s jednotlivými akcemi.

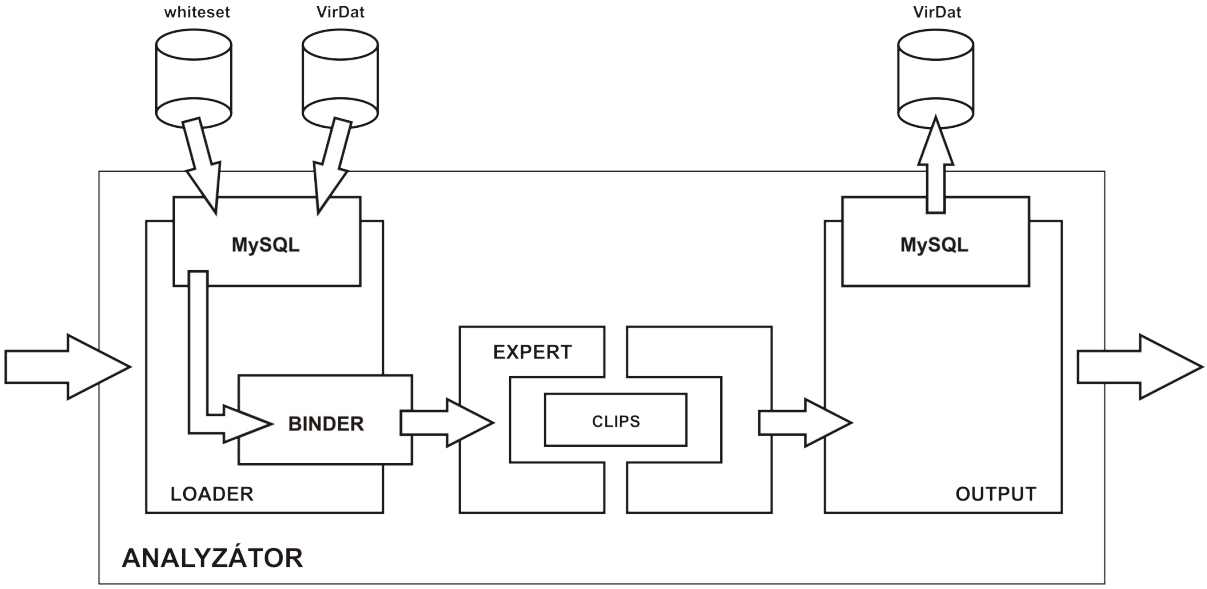
První z těchto funkcí sleduje, jestli některá ze vstupních akcí nemá hodnotu tři, čtyři nebo šest. Jestli je tato hodnota nalezena, vrací `true`, jinak `false`. Druhá funkce prověřuje, jestli se někde v akcích daných na vstupu nenachází hodnota dvě značící manuální zpracování. Je-li tato akce nalezena, pak vrací funkce hodnotu `true`, jinak vrací hodnotu `false`. Třetí funkce slouží pro zjištění, zda-li součástí multislotu není akce na common crc, tedy hodnota sedm. Pokud je tato hodnota nalezena, je vráceno `true`, pokud ne, je vráceno `false`.

5.3.4.4 Funkce vracející počet

Trojice funkcí (`f_num_detected`, `f_num_ignored`, `f_num_unignored`) která vrací počet detekcí nebo akcí.

První z trojice vrací počet detekcí. Vstupním multislotem jsou obvykle detekce a funkce jím prochází. Jakmile najde nějaký obsah, navýší čítač o jedničku. Druhá funkce vrací počet ignorovaných akcí v rámci vstupních parametrů. Ze vstupních parametrů je jejich hodnota porovnávána s hodnotou pět. A třetí funkce vrací počet neignorovaných akcí v rámci vstupních parametrů. V tomto případě porovnáváme vstupní parametry s hodnotou jedna nebo dva.

6 Návrh a implementace programové části



obr. 6.1 schématický nákres kompletní programové části analyzátoru

Mým hlavním úmyslem při návrhu programové části nového analyzátoru byla snaha o vytvoření programu rozděleného do jednotlivých logických celků. Celý analyzátor je tedy pomyslně rozdělen do tří hlavním částí. První část získává informace o faktu. Druhá část je pak předkládá expertnímu systému a zjišťuje z něj odpověď. Třetí a současně poslední část musí umět vhodně přeložit výstupy z expertního systému na programové výstupy. K lepší představě o mém původním a současně implementovaném úmyslu by měl dopomoci obrázek 6.1.

6.1 Loader

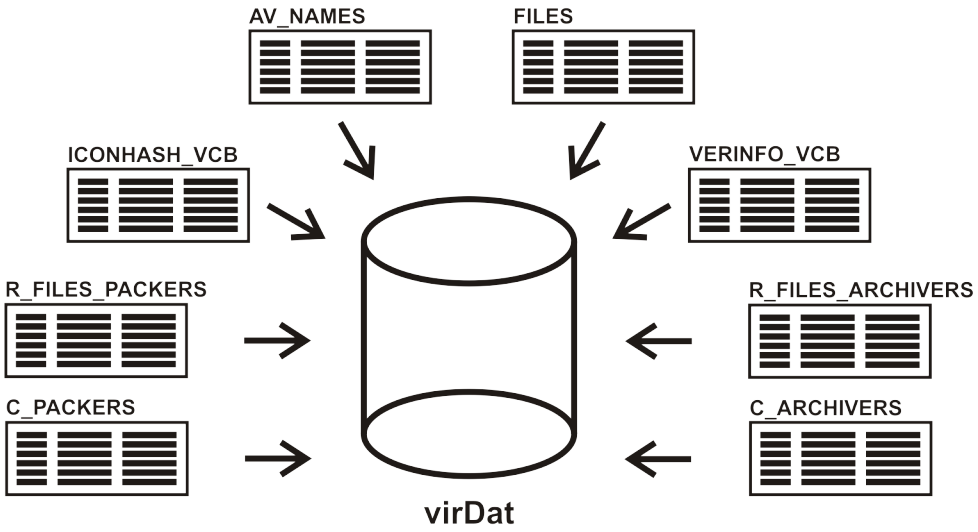
Jedná se o první část tříčlenné sestavy. Tato část je zodpovědná za získání a sestavení skutečnosti tak, aby jej bylo možno předložit expertnímu systému. Vzhledem k faktu, že všechna dostupná data, na kterých je rozhodování postaveno, jsou uložena v databázi MySQL, bylo nezbytné implementovat modul pro spojení s touto databází. Tento modul je implementován ve třídě `MysqlOper` umístěné v souborech `mysqlOper.cc/.h`.

Druhým úkolem této části je sestavení faktu, tak aby jej přijal CLIPS. To je implementováno pomocí metod třídy `Binder`, která je umístěna v souborech `binder.cc/.h`. Nakonec podle

hlavního úkolu načíst data o vzorku dostala třída název `Loader` a její implementace je v souborech `loader.cc/.h`.

6.1.1 zdroje informací pro Loader

Data jsou získávána ze dvou databází.



obr. 6.2 situační nákres využívaných tabulek v databázi VirDat

První databází využitou analyzátorem je VirDat, která obsahuje mimo jiné tabulku `files`. V této tabulce jsou informace o názvu jednotlivých detekcí a informace pro získání obsahu VERINFO. Stejně tak zde najdeme hodnoty ICONHASH, SECTHASH či typ vzorku. Na základě názvů detekcí se získávají akce a kmenová jména v tabulce `av_names`. Výsledné akce pro VERINFO jsou uloženy v tabulce `verinfo_vcb`. Akce pro ICONHASH jsou uloženy v tabulce `iconhash_vcb`. Pro získání informace, čím byl daný vzorek archivován, je použito spojení tabulek `r_files_archivers` a `c_archivers`. Podobně je tomu při vyhledávání informace o balících aplikacích, které jsou k dispozici při spojení tabulek `r_files_packers` a `c_packers`. Pro informace o výsledku ze sandboxu je použita tabulka `sandbox_output`. Druhou databází užitou v programu je již výše zmíněný `whiteset`. V této databázi je využito spojení tabulek `f_file_group a files`. Přesná forma dotazů tak, jak jsou kladeny, je k vidění ve zdrojových kódech, konkrétně v souboru `loader.cc`. Všechny dotazy byly převzaty z původní verze analyzátoru VCB.

6.2 Expert

Druhým pomysleným stupněm analyzátoru je modul Expert. Tento modul vznikl během návrhu jako potřeba zaobalit zdrojové kódy CLIPSu tak, aby bylo možné jednoduše zavolat požadavek na expertízu pouze s předáním správně sestaveného faktu.

Tento modul je tvořen třídou `Expert` (třída je implementována v souborech `expert.cc/.h`), která prostřednictvím svých metod pracuje s API CLIPSu. Postup pro získaní expertízy je složen z několika volání funkcí. V první řadě je potřeba volat funkci na inicializaci prostředí, ve kterém jsou obsaženy naše zkompilované konstrukce. Tím dojde k zavedení našeho expertního systému do paměti.

Poté proběhne restart expertního systému tak, aby bylo zaručeno, že nebudou obsaženy žádné skutečnosti. Tento analyzátor pracuje pouze s jednou skutečností a tou je vzorek. Všechny ostatní jsou nechtěné, neboť by docházelo k jejich kombinaci.

Po restartu expertního systému se do něj vloží `Loaderem` vytvořený fakt. Současně během vkládání faktu je kontrolována jeho struktura a současně existence slotů nad existující zavedenými šablonami. Není tedy možné, že by program pokračoval, aniž by fakt nebyl syntakticky a sémanticky správně.

Teď už je možné zavolat funkci agenda, která začne vyhodnocovat levé strany pravidel vzhledem k existujícím faktům. Pravidla vhodná pro spuštění jsou řazena sestupně podle počtu prvků, které obsahují. Čili obsahuje-li pravidlo deset dílčích podmínek na spuštění zatímco jiné pravidlo pouze pět, pak lze říct, že šance na spuštění má spíše pravidlo z deseti dílčími položkami, protože je výše v agendě.

Jakmile jsou pravidla na agendě, je spuštěno jejich vykonání pomocí funkce run, která začne provádět jejich levé a pravé strany. Vzhledem ke konstrukci pravidel a hlavně jejich pravých stran (jak je diskutováno výše v této publikaci) spustí se pouze jedno pravidlo. Po vykonání běhu již pouze přečteme hodnoty výsledku z expertního systému (sloty RETURN a STATUS). A tyto hodnoty jsou předloženy třetímu modulu.

Jen bych nakonec této podkapitoly ještě rád doplnil, že postup, který je popsán ve výše uvedených odstavcích, je prakticky identický s postupem, který uživatel provede při práci s konzolí CLIPSu. API, které CLIPS nabízí je napsáno, tak aby přechod z konzolového CLIPSu na ten programový bylo co nejsnadnější.

6.3 Output

Posledním modulem implementovaným v programové části analyzátoru je Output. Tento modul musí splňovat jeden z nejdůležitějších požadavků na analyzátor a to zanechat návratové kódy nezměněny vůči předchozí verzi. Na tento nástroj jsou napojeny další softwarové nástroje, které s těmito hodnotami pracují. Tento modul byl tedy pojmenován Output a je implementován ve stejnojmenné třídě, která je umístěná v souborech `output.cc/.h`.

Ve finální fázi návrhů pravidel obsahuje expertní systém přesně dvacet návratových hodnot (výčet a význam těchto návratových hodnot je nastíněn v sekci expertního systému v podkapitole "slot RETURN") a těchto dvacet hodnot je potřeba namapovat na původních šest, které vystupují z programu VCB. Těchto šest hodnot bylo ještě během implementace nového analyzátoru rozšířeno na sedm. Jejich aktuální význam je následující:

- hodnota 0 – nedostatek informací o vzorku
- hodnota 1 – nedělej nic
- hodnota 2 – udělej crc
- hodnota 3 – pošli vzorek na manuální zpracování
- hodnota 4 – zašli vzorek do sandboxu
- hodnota 5 – proveď na vzorkem common crc
- hodnota 6 – nedělej nic, ale vzorek je zajímavý pro pozdější analýzu

Možnost, jak namapovat těchto dvacet hodnot do šesti, spočívá ve faktu, že existuje ještě název kmene. To umožní nevracet celých dvacet hodnot z CLIPSu, ale pouze oněch sedm. Pro úspěšné namapování je potřeba znát název kmene, pod kterým daný vzorek půjde do aktualizace.

Pro lepší představu uvedu příklad. Existuje pět pravidel pro automatický crc. Těchto pět pravidel vrací z CLIPSu pět návratových hodnot, ale ve skutečnosti se na vzorku udělá vždy jen automatický crc. Rozdíl je jen v tom, že program musí vědět, že v případě spuštění pravidla pro NOD32 si má stáhnout a do souboru zapsat název kmene, který získal od NOD32, nikoli třeba od Kaspersky antivirus. Do takových pravidel počítáme ta pro common crc, manuální detekce (ačkoli tady chybí zpracování na kmen, protože vzorek bude dále analyzován) či právě automatické crc. Vlastní implementace je pak řešena prostřednictvím konstrukce switch, tvořené návratovými kódy z CLIPSu. Ty jsou definovány ve výčtovém typu zvaném `OutputCLIPS`.

7 Testování a Statistiky

7.1 Testování

7.1.1 Způsob testování

Celý proces testování probíhal po dobu dvaceti tří dnů. Pro potřeby testování bylo využito původního VCB, které fungovalo jako referenční program. Jak CLIPS, tak VCB byli spuštěni současně na jeden vzorek. Srovnávaly se výstupní hodnoty z obou programů. Došlo-li k rozdílu, byla do testovacího logu přidána ještě podoba vzorku (databázových informací) v době, kdy byl vzorek podroben analýze. Bylo to z toho důvodu, že určité informace mohou být pozměněny v důsledku zaslání do sandboxu.

Po získání logu probíhala analýza rozdílů a následná úprava pravidel. Před vypuštěním dalšího vydání probíhalo testování lokálně nad vzorky, které dříve tvořily rozdíly. Tímto způsobem bylo nakonec dosaženo absolutní shody mezi novým analyzátorem a původním VCB.

7.1.2 Data plynoucí z testování

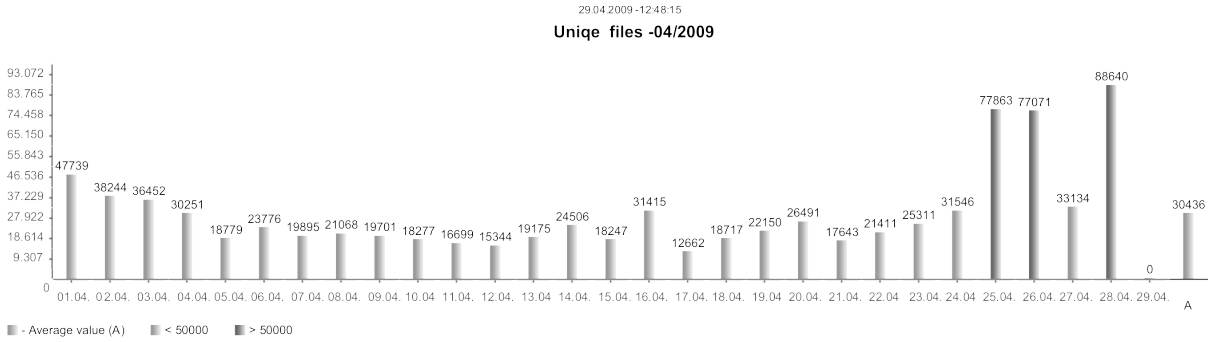
První log z testování byl obdržen a analyzován 23. 3. 2009 a obsahoval 73 vzorků, než bylo testování zastaveno. Při tomto testování bylo nalezeno v logu celkem 29 rozdílů v rozhodování analyzátorů. Log obsahoval zejména dva druhy rozdílů. První, větší skupina byla v rozdílném názoru na zaslání vzorku do sandboxu (tento rozdíl byl řešen nejméně dvě třetiny celého času na testování). A druhý rozdíl byl spojen s faktem, že analyzátor byl napojen na zrcadlo databáze, nikoli na vlastní databázi. Proto se stávalo, že vzorek ještě než byl zrcadlen na tuto druhou databázi určenou pouze pro čtení, už byl požadován programem, a proto docházelo k pádům.

Během následujících více než tří týdnů bylo provedeno celkem osmnáct testování. Po ukončení každého z nich nastala fáze úpravy pravidel v části expertního systému, aby bylo dosaženo naprosté shody s analyzátorem. Celým testováním prošlo 4370 vzorků, které byly analyzovány oběma programy.

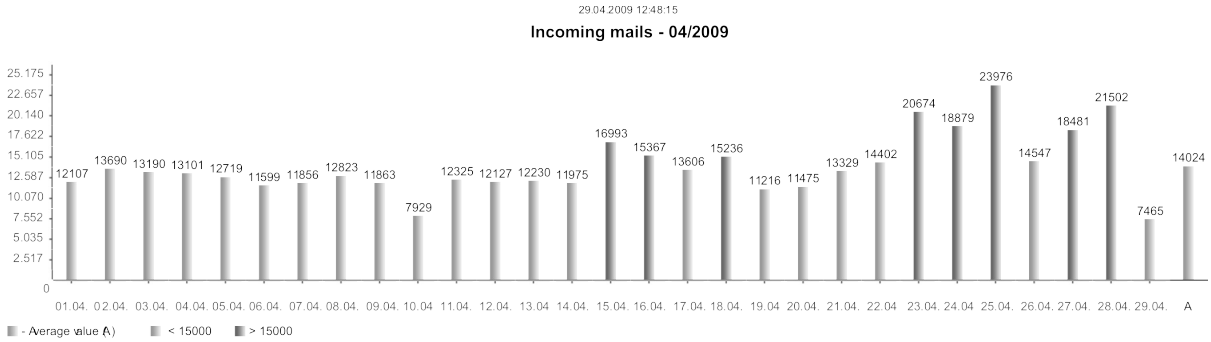
Poslední testování proběhlo 15. 4. 2009 a log z tohoto testování obsahoval jediný rozdíl z 1458 vzorků. Navíc rozdíl byl dán pravděpodobně zpožděním mezi spuštěním CLIPSu a VCB. Tímto logem bylo testování ukončeno a byly současně předány zdrojové kódy a základní programová dokumentace.

7.2 Statistiky

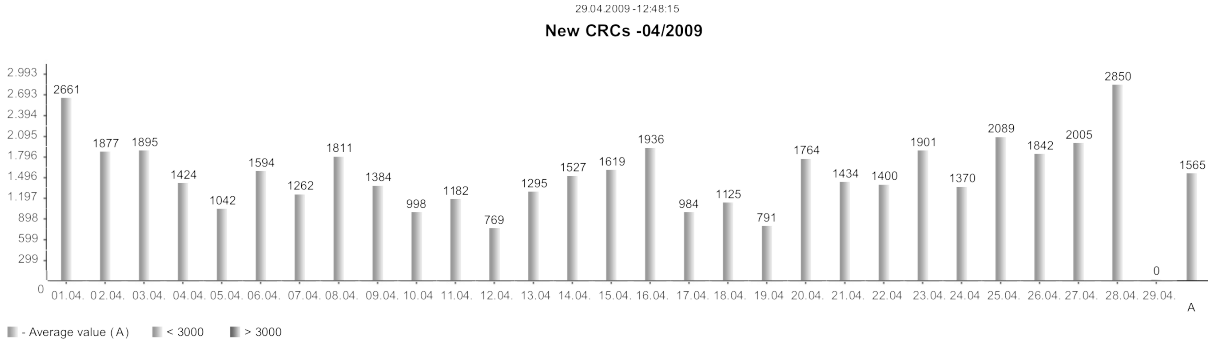
V této kapitole jsou ke zhlédnutí některé statistiky, které popisují počty e-mailů, unikátních souborů, vytvořených definic a počet chybně detekovaných souborů, které do AVG Technologies s.r.o. přijdou. Všechny obrázky jsou pak ještě ve své originální velikosti publikovány v příloze A.



obr. 7.1 grafické znázornění unikátní souborů, přijatých v AVG Technologies s.r.o. za měsíc duben, rok 2009



obr. 7.2 Grafické znázornění příchozích e-mailů v AVG technologies s.r.o. za měsíc duben, rok 2009



obr. 7.3 Grafické znázornění počtu vytvořených detekcí v AVG Technologies s.r.o. za měsíc duben, rok 2009

8 Závěr

8.1 Cíle a využití

Cílem této podkapitoly je shrnout, jestli se podařilo splnit cíle, které tato práce zadávala a jestli se pro tuto práci našlo reálné využití. Práce měla za úkol implementovat pravidla do CLIPSu a vytvořit tak nový a lépe rozšiřitelný analyzátor, než byl VCB, který byl psaný čistě v jazyce C. Tento bod se podařilo splnit už během testování, kdy bylo potřeba daná pravidla pozměňovat a rozšiřovat oproti VCB. Současně se tedy podařilo prokázat i fakt, že použití specializovaného nástroje, jako je CLIPS, je lepším řešením než implementace v procedurálním jazyce. I přes fakt, že úplnou verzi pravidel je obtížnější sestavit, neboť nejedno pravidlo bylo rozšiřováno až během testování. Podmínkou zvládnutí byla též naprostá shoda výstupů analyzátoru VCB a CLIPSu. To bylo prokázáno při posledním testování, kterým byla tato práce ukončena.

Jelikož požadavek na nový analyzátor vzešel z AVG Technologies s.r.o., s využitím by neměl být problém. Na základě odezvy od nadřízených po ukončení testování se dá očekávat, že bude analyzátor nasazen do normálního provozu.

8.2 Práce na projektu - chronologická data

První informace začaly být shromažďovány o CLIPSu 21. 9. 2008 z publikace [1], kde byly sepsány základní principy a části expertního systému. Učení se principům expertních systémů probíhalo přerušovaně až do 16. 1. 2009, kdy jsem přešel na základní programovací techniky obsažené v manuálu "BPG - Basic Programming Guide".

V tuto dobu současně začínají první experimenty s CLIPSem. Postupně tyto experimenty přechází k implementaci jednotlivých pravidel definovaných ve VCB a implementaci navržených konstrukcí jako deftemplate. Současně se v tuto dobu také řeší, jak přeložit konstrukce CLIPSu do jazyka C.

23. 2. 2009 se začínají tvořit první návrhy na programovou část, které jsou záhy implementovány. Implementace programové části a poslední úpravy pravidel jsou prováděny až do 20. 3. 2009. V této fázi se upotřebil hlavně manuál "APG - Advance Programming Guide".
Testování začalo 23. 3. 2009 a končí 15. 4. 2009, kdy je projekt prohlášen za dokončený a kdy jsou odevzdány zdrojové kódy.

Literatura

[1] Giarratano Joseph, Riley Gary: Expert system Principles and programming Third edition, PWS, ITP, 1998, ISBN 0-534-95053-1

[2] Expert system tools, [online], http://www.cs.cofc.edu/~manaris/ai-education-repository/expert-systems-tools.html

[3] CLIPS: A Tool For Building Expert Systems, [online], http://clipsrules.sourceforge.net/

[4] CLISP: CLIPS Reference Manual, Volume I: Basic Programming Guide, [online], http://clipsrules.sourceforge.net/documentation/v630/bpg.pdf

[5] CLIPS: CLIPS Reference Manual, Volume II: Advance Programming Guide, [online], http://clipsrules.sourceforge.net/documentation/v630/apg.pdf

Příloha A

Tato příloha obsahuje grafické znázornění statistik, které jsou zobrazeny v podkapitole 7.2. Tato znázornění, ale obsahují plnou velikost, která umožňuje lépe přečíst obsah.

Příloha B

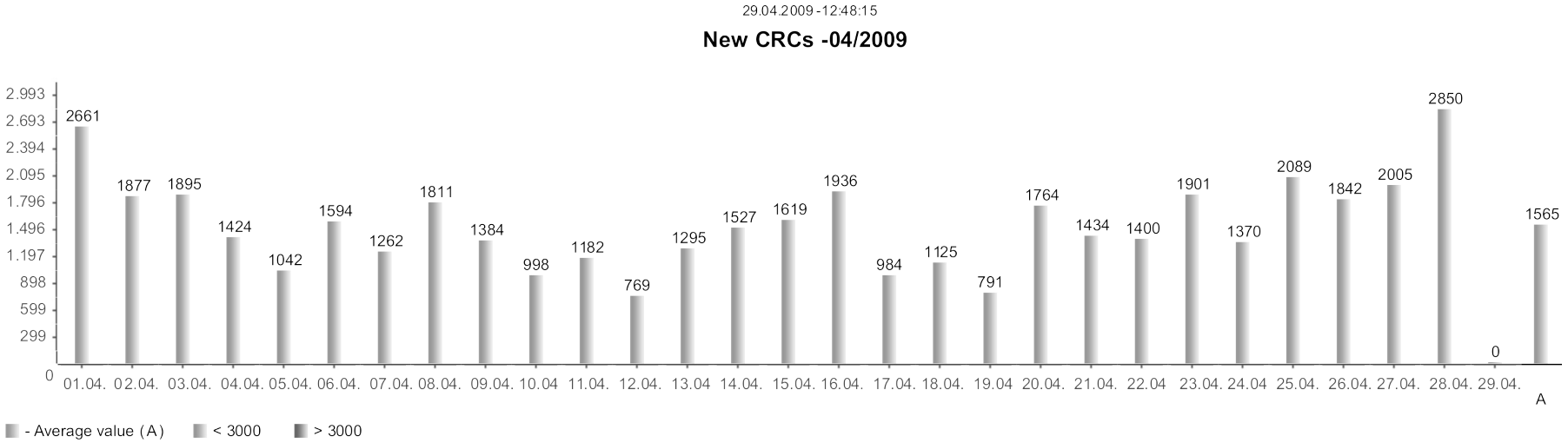
Tato příloha popisuje obsah CD nosiče, na kterém jsou umístěny zdrojové kódy a technická zpráva k této práci.

Složka construct obsahuje konstrukce implementované v CLIPS.

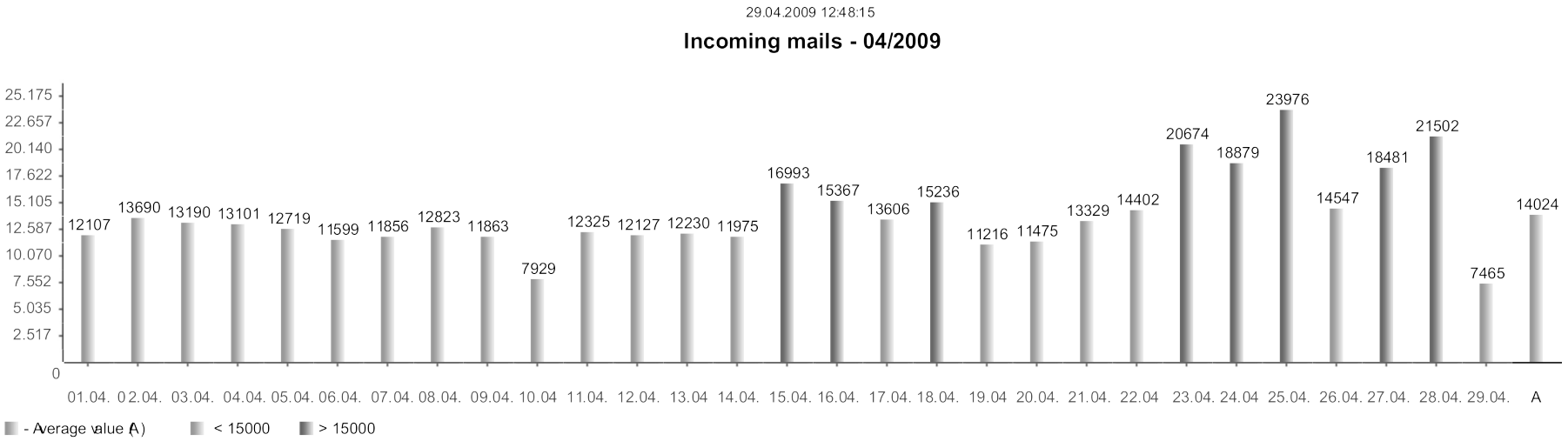
Složka sources obsahuje zdrojové kódy, které obsahují tuto práci. Současně je v této složce obsažena i složka clips, která nese zdrojové kódy CLIPSu. V těchto kódech jsou obsaženy i soubory zkompilovaných konstrukcí. Tyto soubory mají prefix smp.

Složka debug obsahuje logy z testování.

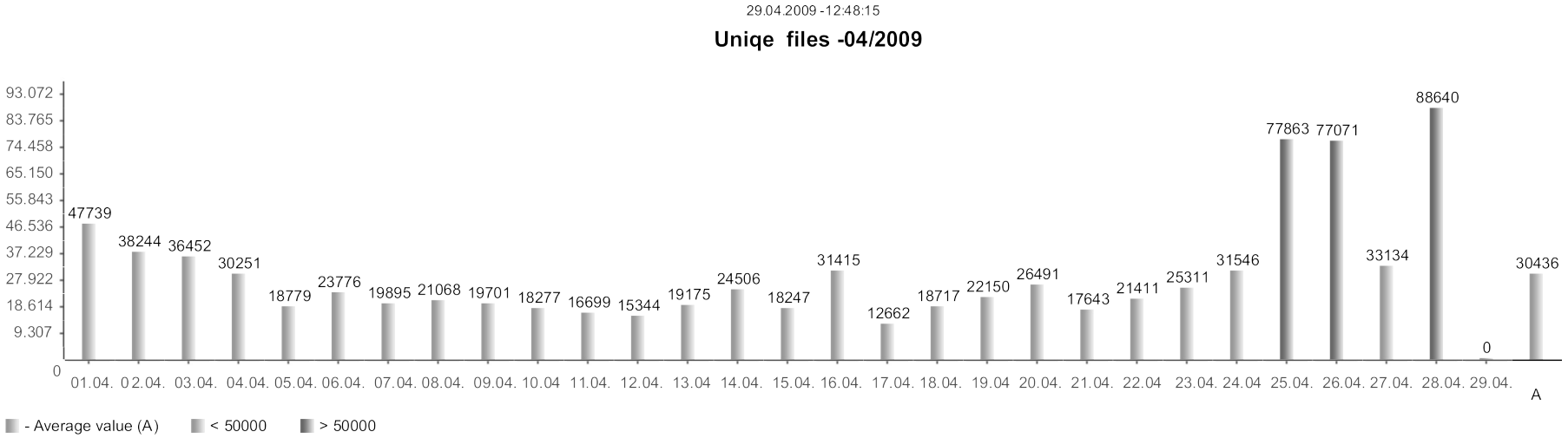
Složka documentation obsahuje technickou zprávu pod názvem BP.pdf



A.1 grafické znázornění počtu vytvořených detekcí v AVG Technologies s.r.o.
za měsíc duben, rok 2009



A.2 grafické znázornění příchozích e-mailů v AVG technologies s.r.o.
za měsíc duben, rok 2009



A.3 grafické znázornění unikátní souborů, přijatých v AVG Technologies s.r.o.
za měsíc duben, rok 2009