MASARYKOVA UNIVERZITA
FAKULTA INFORMATIKY

Zpracování událostí vyšších řádů

DIPLOMOVÁ PRÁCE

Martin Juřen

Brno, podzim 2012

Masarykova univerzita
Fakulta informatiky

ZADÁNÍ DIPLOMOVÉ PRÁCE

Student: Martin Juřen

Program, obor (příp. Specializace): Informatika, Umělá inteligence a zpracování přirozeného jazyka (specializace Umělá inteligence)

Garant oboru (příp. Specializace): doc. PhDr. Karel Pala, CSc.

Vedoucí práce: RNDr. Pavel Minařík

Katedra: **KPSK**

Název práce: Zpracování událostí vyšších řádů

Zadání: Úkolem studenta je navrhnout rozšíření existující aplikace pro bezpečnostní monitoring počítačových sítí. Rozšíření by mělo uživateli poskytnout rozhraní dotazovacího jazyka pro definici vlastních vzorů událostí vyšších řádů. Ty budou aplikací následně detekovány. Řád události v tomto případě představuje míru abstrakce nad surovými daty. Za události nultého řádu lze tak považovat přímo neupravená data, za události prvního řádu potom současné výstupy aplikace. Dotazovací jazyk může být inspirován existujícími systémy z oblasti *Complex event processing*, měl by respektovat požadavky dané jeho zamýšleným nasazením, ale zároveň by měl být jednoduše rozšiřitelný pro obecné automatické zpracování velkého množství dat (především pak datových proudů). Dalším důležitým požadavkem na daný jazyk je možnost definovat dotazem detekce využívající jednoduché algoritmy strojového učení (např. naučení a následná kontrola prahových hodnot).

Poznámka: V rámci práce bude student pracovat s citlivými firemními materiály, které zahrnují technické specifikace a zdrojové kódy. Část výsledku práce, zejména zdrojové kódy, které jsou přímo vázané na know-how společnosti AdvaICT, nebudou odevzdány jako součást diplomové práce.

Základní literatura:

LUCKHAM, David. *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems.* Boston: Addison-Wesley, 2001.

Souhlasím se zadáním (podpis, datum)

Juřen D. 18.10.2012

student(ka)

vedoucí diplomové práce

garant oboru

Prohlášení

Prohlašuji, že tato diplomová práce je mým původním autorským dílem, které jsem vypracoval samostatně. Všechny zdroje, prameny a literaturu, které jsem při vypracování používal nebo z nich čerpal, v práci řádně cituji s uvedením úplného odkazu na příslušný zdroj.

Martin Juřen

Vedoucí práce: RNDr. Pavel Minařík

Poděkování

Děkuji.

Vedoucímu této práce, Pavlu Minaříkovi, za to, že mi ponechal volnost a do práce příliš nezasahoval. I za jeho cenné rady.

Okruhu svých nejbližších. Páji Vávrové, díky jejíž trpělivosti, důvěře v mé vědomosti a každodenní podpoře tato práce vůbec vznikla. Má mé největší díky. Jirkovi, Dáši a Vendi Juřenovým a Janě Šustkové, rodině, ve které jsem vyrůstal, která mě při studiu podporovala a které jsem musel působit spoustu stresu neustálým odkládáním práce na tomto textu. Jakubu Kotulovi, za občasné laické konzultace, Davidu Němečkovi za občasné odborné konzultace. Kolegům Honzovi Barienčíkovi, Michalu Vaverkovi a Jardovi Vitekerovi za jejich nadšení pro věc, Jirkovi Ušpákovi za pravidelný odvoz domů, díky kterému jsem měl více času na práci na tomto textu.

Panu docentu Palovi za nečekanou motivaci pár měsíců před odevzdáním. Pánům profesorům, docentům, odborným asistentům i těm netitulovaným, kteří mi během mého studia dali potřebné základy, bez nichž by tato práce nemohla vzniknout. Obzvláště pak profesoru Křetínskému a doktorům Řehákovi a Benešovi za představení světa formálních jazyků a automatů, profesoru Kučerovi za přednášky o formální sématice programovacích jazyků a docentu Polákovi a doktoru Klímovi za nutné základy algebry, bez kterých bych se neobešel ani v této práci.

Za částečný podíl na vzniku této diplomové práce bych rád poděkoval i tiskárně, která tento text vytiskne a sváže. Bohužel ještě nevím, která to bude, takže nemůžu být konkrétnější.

Všem, kteří mi neřekli, jak náročnou cestu jsem si vybral.

Každému, kdo tento seznam dočetl až do konce a zjistil, že jeho jméno v něm chybí přesto, že by chybět nemělo.

Shrnutí

Tato práce se zabývá návrhem dotazovacího jazyka umožňujícího zpracování proudu dat. Tento dotazovací jazyk umožňuje díky možnostem rozšíření použití v libovolném prostředí. Součástí práce je i krátký popis principu fungování překladače tohoto jazyka.

Klíčová slova

Zpracování komplexních událostí, Complex Event Processing, Zpracování datových proudů, Dotazovací jazyk, NetFlow, Caché, Automatické zpracování dat

Obsah

1 Úvod

Současná společnost je příznačná vytvářením obrovského množství digitálních dat. Příkladem mohou být fotografie a videa, ale také automatické výstupy z lékařských měřících přístrojů, výstupy senzorových sítí, záznamy o činnosti výpočetních systémů (tzv. počítačové *logy*) a další. Tato práce se zabývá především záznamy strojově vytvářenými.

Strojově vytvářená data se většinou skládají z velkého množství jednoduchých záznamů, které skoro vždy tvoří posloupnosti v čase. Jednotlivé záznamy (zprávy) také často představují informace buď to nedůležité (např. zpráva o úspěšné aktualizaci systémového času) nebo takové, k jejichž pochopení je potřeba kontext daný dalšími záznamy (např. bez znalosti předchozí výšky hladiny řeky nedokážeme z aktuálního stavu určit, jestli hladina stoupá, klesá, nebo zůstává na stejné úrovni). Do druhé skupiny lze zařadit také zprávy, jejichž informační hodnota roste s jejich dalšími opakováními, případně současným výskytem zpráv nesoucích odlišné informace (např. zpráva o podání léčiv následovaná zprávou o výskytu některých protilátek v krvi může znamenat alergickou reakci pacienta).

Neustále přibývající množství dat již není možné zpracovávat ručně ani dlouhodobě uchovávat. Pro zpracování je tak stále častěji nutné použít automatické systémy, které jsou schopné data zpracovávat, aniž by bylo nutné data dále uchovávat. Tímto přístupem se zabývá oblast zvaná *Zpracování komplexních událostí* (*Complex Event Processing*, dále CEP).

V rámci CEP je vyvíjeno několik projektů, které definují dotazovací jazyk zpracovávající jednotky dat (takzvané události) v datových proudech. Při databázovém přístupu ke zpracování dat jsou do databáze ukládána data a dotazy se nad daty provádějí dodatečně. Oproti tomu při CEP přístupu jsou v databázi uloženy dotazy a data jsou zpracována pouze v okamžiku doručení do systému (případně mohou být dočasně uložena pro navazující výpočty daného dotazu).

Současně s rozšiřováním počítačových technologií mezi širokou veřejnost je také stále více pociťována potřeba vhodného zabezpečení nebo alespoň včasného varování v případě narušení bezpečnosti. Pro potřeby včasného varování je opět nutné vytváření a následné zpra-

cování velkého množství zpráv informujících o stavu konkrétních zařízení případně i o stavu celé sítě. Za tímto účelem vznikla aplikace *Anomaly Detection System* (viz 3.3).

Tato aplikace však stále poskytuje uživateli velké množství informací s omezenou mírou abstrakce, ve kterých je možné přehlédnout některé bezpečnostní incidenty, které mohou být například rozloženy v delším časovém období. Další incidenty pak mohou být popsány několika různými zprávami (viz například 3), mezi kterými nemusí být daná spojitost patrná.

Proto vznikla tato práce, jejímž cílem je primárně návrh systému umožňujícího další zpracování zpráv vytvářených aplikací ADS. Jako vhodný přístup k tomuto problému byl zvolen právě CEP.

Vzhledem k nárokům na rychlost byl vytvořen návrh nového dotazovacího jazyka (částečně inspirovaného jazykem EPL systému Esper [1]), jehož implementaci bude možné přizpůsobovat pro vyšší výkon v konkrétním prostředí. Samotný návrh však uvažuje obecně použitelný systém, nezávislý na prostředí. Nezávislosti jazyka je dosaženo modelem rozšiřování pomocí definic datových typů.

Celá tato práce se zabývá automatizací abstrakce nad jednotkami dat (zprávami, událostmi). Za cíl by se tak dala označit také snaha umožnit počítači i přes samé stromy uvidět les.

Druhá kapitola práce se blíže zabývá datovými proudy a různými přístupy k jejich zpracování. Zdrojem pro tuto kapitolu je zejména kniha Davida Luckhama: *The Power of Events* [2].

Ve třetí kapitole je popsáno prostředí bezpečnostního sledování počítačových sítí, ve kterém budou výsledky této práce použity.

Čtvrtá kapitola slouží k seznámení se s navrhovaným jazykem, může být použita jako jednoduchý průvodce pro uživatele, který má jisté zkušenosti s dotazovacími jazyky.

V páté kapitole je navrhovaný dotazovací jazyk popsán po formální stránce. Je zde uvedena formální gramatika jazyka a formální sémantika některých jazykových konstrukcí. Doplňující informace k této kapitole lze nalézt také v příloze. V této kapitole jsou využity informace čerpané převážně z přednášek Vybrané kapitoly z teorie automatů [3], Sémantiky programovacích jazyků [4] a ze skript Překladače [5].

Šestá kapitola se zabývá návrhem přeložené podoby dotazů jazyka a stručným popisem běhu překladače.

Jednotlivé přílohy obsahují postupně klíčová slova navrhovaného jazyka, výsledky aplikace funkce *FIRST* a funkce *FOLLOW*, rozkladovou tabulku pro LL(1) analýzu, transformační pravidla pro přepis derivačního stromu na strom použitelný pro překlad jazyka, návod pro vytváření rozšíření dotazovacího jazyka a doplňující dotazy jazyka sloužící pro ilustraci některých problémů.

2 Zpracování proudů dat

2.1 Datové proudy, místně a časově příslušná data

Datové proudy (proudy dat) jsou s rozvojem sítě Internet stále častějším formátem výskytu dat. Typickým představitelem datových proudů je vysílání (a sledování) internetových televizních přenosů (samotný televizní přenos je také datovým proudem). Obrazová data jsou průběžně přijímána klientskou stanicí a ihned zobrazována, po zobrazení se k nim není možné vrátit (v případě, že není televizní vysílání zaznamenáváno).

Datové proudy (nebo alespoň jejich části, například jednotlivé televizní pořady) tak mohou být uchovávány i pro další zpracování, to s sebou ale může nést velké režijní náklady, v některých případech by bylo nutné ukládat obrovské množství dat. Dalším důvodem, proč nemusí být vhodné data nesená datovými proudy dlouhodobě uchovávat, je možnost jejich rychlého zastarávání.

Mějme síť automatických barometrů rovnoměrně rozmístěných po celé republice. Tyto barometry vždy v pravidelných hodinových intervalech odesílají na centrální stanici právě naměřené hodnoty atmosférického tlaku. Takto získaná data lze teoreticky použít pro krátkodobou předpověď vývoje počasí, nemá však význam je dlouhodobě uchovávat, protože například pro určení dlouhodobého vývoje klimatu pravděpodobně postačí data agregovaná přes delší časové úseky a oblasti pokryté více senzory (takovouto agregací může být například zjištění trendu vývoje hodnot atmosférického tlaku, zápis křivky popisující denní průběh hodnot a jiné).

Výše zmíněný příklad lze použít také pro demonstraci významné oblasti datových proudů a to místně a časově příslušných dat (*spatio-temporal data*). Tato data se vztahují ke konkrétnímu místu nebo času a bez těchto informací ztrácí svůj význam. Například bez znalosti umístění jednotlivých automatických barometrů lze sestavit větrnou mapu pro danou hodinu jen s velmi malou pravděpodobností (při počtu deseti stanic a za předpokladu, že každá v daný okamžik naměřila jinou hodnotu, odpovídá tato pravděpodobnost hodnotě přibližně $2,8 \times 10^{-7}$).

Při zpracování datových proudů je často nutné zohlednit jejich

specifické vlastnosti. Často není možné vracet se k datům, která byla už zpracována, v některých případech může být obtížné předpovídat obsah dat, která budou zpracovávána, případně je nutné počítat s odlehlými body způsobenými možnou závadou senzoru[1].

Datovými proudy tak mohou být data ze senzorů, senzorových sítí, data popisující chování počítačové sítě, případně jednotlivých zařízení, data o šíření nemoci nebo i lékařské záznamy pacienta a další.

2.2 Zpracování komplexních událostí [6][7][8][9]

Zpracování datových proudů se věnuje oblast informatiky nazývaná *Zpracování komplexních událostí* (CEP). Tato oblast se zabývá oddělováním důležitých dat od ostatních. Důležitá data mohou být představována daty popisujícími vzácně se vyskytující události (odlehlé body) nebo korelací informací z několika zdrojů (případně z posloupnosti záznamů).

Vzácné události (*rare events*, *outliers*) se vyskytují v daleko menší míře než normální události, můžou však mít zásadní význam. Mezi normálními událostmi je obecně možné objevovat jistou podobnost, události, které jsou označovány jako vzácné, se mohou vzájemně znatelně lišit a může být velmi obtížné předpovídat jejich podobu.

Vzácnou událostí je například útok na počítačovou síť, rakovinné buňky na mammogramu, letecká nehoda nebo nakupující zákazník na Amazonu.

Události, které představují vztah mezi několika informacemi, mohou být rozloženy v delším časovém období, případně může jít o posloupnost informací, které samotné neposkytují potřebné vědomosti pro odvozování důsledků.

Příkladem takovéto události může být snaha o získání přístupu k počítači pomocí služby SSH. Je-li daná stanice sledována pomocí protokolu syslog, můžeme mít informaci o každém dílčím neúspěšném pokusu o přihlášení. Aby se však dalo odhalit, že se skutečně jedná o útok, je nutné udržovat počítadlo neúspěšných přihlášení, které při překročení stanoveného prahu vytvoří hlášení o pravděpo-

1. Teplotní senzor v místnosti hlásí prudké zvýšení teploty, znamená to poruchu nebo požár?

dobném útoku na službu SSH.

Dalším příkladem může být například kontrola pracovního postupu, kdy pro každý úkon existuje ve sledovacím systému jedna zpráva a pokud tyto zprávy přijdou ve špatném pořadí, je vytvořeno hlášení o chybě.

Jedním z přístupů, který se v rámci oblasti CEP používá ke zpracování událostí, je zpracování s využitím specializovaného dotazovacího jazyka. Tento přístup musí respektovat povahu proudů dat – není možné data trvale ukládat do databáze a nad těmito daty dodatečně spouštět dotazy. Místo toho je nutné zpracovávat dotazy průběžně, vždy v okamžiku příchodu příslušné události (zprávy) do systému, který provádí sledování. Takovýmto jazykem je například dotazovací jazyk *Event Processing Language* společnosti EsperTech.

Nevýhodou tohoto přístupu je, že bez předchozí znalosti hledaného vzoru nelze napsat dotaz, který by daný vzor odhaloval. Proto je nutné data nejprve analyzovat (například s využitím nástrojů strojového učení) a až následně vytvořit dotazy, podle kterých budou události odhalovány. V případě odhalování vzácných událostí tak může být tento přístup nepoužitelný.

V dalším textu se pod pojmem *CEP* vždy rozumí zpracování komplexních událostí s využitím dotazovacího jazyka.

2.3 Algoritmy strojového učení zpracovávající datové proudy [10]

Druhou oblastí informatiky, která se zabývá zpracováním datových proudů je strojové učení a dobývání znalostí z dat. Tato oblast se snaží o totéž, jako CEP, pouze jinými způsoby.

Pro zpracování datových proudů algoritmy strojového učení platí kromě výše zmíněných obecně platných ještě další skutečnosti. Při použití shlukovacích algoritmů je často nutné předem znát počet shluků, do kterých mají být data rozdělena, to v případě zpracování datových proudů nemusí být možné. Dále není možné kontrolovat validitu jednotlivých shluků (ty se mohou při zpracování dat postupně měnit). Algoritmy strojového učení musí být schopné reagovat na vývoj prostředí (*concept drift*) a postupné zastarávání naučených klasifikátorů.

Jedním z algoritmů pro odhalení odlehlých bodů v datových proudech je algoritmus využívající rozšiřitelné Markovovy modely (*Extensible Markov Models*, EMM [11]). Algoritmus ztotožňuje jednotlivé shluky událostí se stavy Markovova modelu, který je postupně aktualizován podle nově přicházejících dat. Podobné události jsou tedy vždy sloučeny v jednom uzlu. Algoritmus umožňuje učení bez učitele i s učitelem (umělé přidání stavů modelu). Odhalení vzácných událostí je založena buď to na nemožnosti zařadit událost do žádného shluku (stavu modelu) nebo na velmi nízké pravděpodobnosti přesunu události ze stavu modelu předchozí iterace do současného stavu.

Dalším možným přístupem je využití statistických metod předpovídajících průběh hodnot získaných z datového proudu. Pokud se skutečná hodnota od předpovězené příliš liší, je vytvořeno hlášení o nepředpokládaném chování. Obecně jsou tyto metody použitelné pouze pro kvantifikovatelné veličiny (například počet spojení). Zástupcem těchto algoritmů je Holt-Wintersova metoda. V rámci této metody je každá hodnota dána jako součet tří prvků – základny, lineárního trendu a sezónního trendu. [12]

Jak již bylo zmíněno, strojové učení může být také využito k jednorázové analýze nasbíraných dat. Výsledky této analýzy pak lze uplatnit při vytváření dotazů pro CEP.

3 Východiska

Hlavním cílem této práce je navrhnout systém zpracování komplexních událostí v prostředí bezpečnosti počítačových sítí. Narušení bezpečnosti sítě je možné rozdělit na dvě základní oblasti a to na ohrožení z vnější sítě a ohrožení působené některým zařízením připojeným do vnitřní sítě.

Ohrožení z vnější sítě jsou v době vzniku této práce reprezentované obzvláště útoky typu *distribuované odmítnutí služby* (*Distributed Denial of Service*, zkráceně *DDoS*) často používanými uskupením útočníků, kteří si říkají *Anonymous*[1]. K DDoS útoku je využita síť zařízení (botnet[2], případně počítače dobrovolníků), která v jeden časový okamžik začnou posílat velké množství požadavků na služby poskytované cílem útoku, který je následně zahlcen (v případě úspěšného útoku) a nedokáže odpovídat ani na požadavky neútočících klientů. Pro ty se tak stávají služby poskytované cílem útoku nedostupné.

Dalším typem útoků realizovaných obzvláště z vnější sítě může být skenování portů stanic, které je často první fází rozsáhlejšího útoku. Po skenování portů může následovat pokus o zneužití služeb vzdáleného připojení (např. *telnet*, *Secure Shell* a *Remote Desktop*), které byly ve skenované síti objeveny. V případě úspěšného napadení služby vzdáleného připojení může být na oběť útoku nahrán škodlivý kód, který se touto cestou šíří.

Podobné chování vykazuje například červ *Morto* [13], který ke svému šíření využívá službu Windows Remote Desktop. Červ nejprve s využitím skenování portu číslo 3389 (na tomto portu je služba Windows Remote Desktop poskytována [14]) objeví stanice umožňující vzdálenou správu pomocí této služby. S časovým odstupem se pokusí s využitím slovníku přihlašovacích údajů na tyto stanice přihlásit. V případě úspěchu umístí na napadenou stanici svou ko-

---

1. Neformální hnutí s volnou organizací (každý se může prohlásit za jednoho z *Anonymous*). Síťovými útoky zastrašují organizace, které podle jejich názoru omezují svobodu šíření informací. Více viz `http://cs.wikipedia.org/wiki/Anonymous_(skupina)`.
2. Síť zařízení infikovaných škodlivým programem, který umožňuje útočníkovi zadávat jednoduché příkazy, které infikovaná zařízení následně vykonají.

pii, která se po časové prodlevě aktivuje. Tato kopie využívá pro získání dalších instrukcí vzdálené servery, se kterými komunikuje prostřednictvím portu číslo 53, který je standardně používán pro poskytování služby překladu doménových jmen [14] (*Domain Name Service*, DNS), proto tato komunikace nemusí být na první pohled podezřelá. Obvykle se po obdržení instrukcí červ pokouší dále šířit. Proto začne napadená stanice skenovat další stanice (především ve svém okolí) na portu číslo 3389 a následně se pokusí pomocí služby Windows Remote Desktop na tyto stanice přihlásit.

Mezi útoky, které jsou vedeny z prostředí vnitřní sítě lze zařadit vynášení důvěrných informací, napadení sítě škodlivým programem s využitím nezabezpečeného bezdrátového připojení či porušení bezpečnostních politik nebo dalších pravidel pro práci s výpočetní technikou platných ve společnosti využívající danou síť. Případným porušením pravidel dané společnosti může být například využívání aplikací okamžité komunikace (*instant messaging*, dále IM), případně jejich webových verzí. Za vážné porušení nejen bezpečnostních politik, ale potažmo i zákona o právu autorském[3] lze považovat použití některé z aplikací sítí *peer-to-peer* (*BitTorrent*, *DC++*, *eMule* a podobné). Tyto aplikace jsou používány zejména pro stahování nelegálních kopií dat, která jsou často současně sdílena s dalšími uživateli.

3.1 Sledování provozu na síti

V současnosti existuje několik možností, jak sledovat provoz na síti a provoz jednotlivých stanic k síti připojených. Jako příklad lze uvést protokol *syslog* [15], který je využíván pro záznam činnosti koncových stanic na síti a protokol *SNMP* [16], který umožňuje sběr statistik o provozu na síti a generování jednoduchých strukturovaných zpráv.

Dalším protokolem, poskytujícím informace o síťovém provozu je protokol *NetFlow* [17]. Záznamy tohoto protokolu (síťové toky) představují agregaci posloupností paketů majících stejnou zdrojovou a cílovou adresu, stejné zdrojové a cílové porty a využívající stejný protokol transportní vrstvy (např. ICMP, TCP, UDP, ...). Do jednoho toku jsou navíc agregovány pouze ty pakety, mezi kterými nejsou

3. Zákon č.121/2000 Sb. Autorský zákon.

```
Date flow start      Duration Proto Src IP:Port  Dst IP:Port Packets Bytes
2013-01-01 06:23:15.861 0.000 UDP 10.0.1.9:67 -> 10.0.1.1:68 2       656
```

Obrázek 3.1: Příklad NetFlow záznamu vypsaného programem *nfdump* [21]. IP adresy byly z důvodů zachování anonymity pozměněny.

příliš velké časové rozestupy (nastavitelné, tzv. *neaktivní časový limit*), a které dohromady nepřekračují nastavené časové omezení (tzv. *aktivní časový limit*).

Mějme například exportér NetFlow záznamů, který má neaktivní časový limit nastavený na 30 sekund a aktivní časový limit 300 sekund. Pokud se mezi dvěmi stanicemi pošle každých 40 sekund jeden paket (např. paket sloužící pro udržení ustanoveného spojení), je pro každý tento paket vytvořen jeden síťový tok. Pokud se mezi stanicemi posílá nepřetržitý proud dat (např. stahování velkého souboru) po dobu 820 sekund, budou tyto pakety rozděleny do tří po sobě jdoucích síťových toků. První dva toky budou mít trvání 300 sekund, třetí tok bude mít trvání 220 sekund.

Každý NetFlow záznam může být dále tvořen statistickými informacemi o paketech, které jsou do toku zahrnuty. NetFlow záznam tak může obsahovat informace o celkovém množství přenesených dat (vzhledem k implementaci opět dochází při dosažení celkové hodnoty větší než 4 GiB k rozdělení na více po sobě jdoucích toků), počtu paketů tvořících jeden záznam, časové známce prvního paketu toku, trvání toku (rozdíl mezi časovou známkou prvního a posledního paketu), v případě protokolu TCP i o jednotlivých TCP příznacích (*TCP flags*) a dalších[4].

Přiložený NetFlow záznam (viz 3.1) obsahuje v uvažovaném prvním poli časovou známku počátku síťového toku, dále trvání toku (v sekundách), použitý protokol, zdrojovou IP adresu a číslo zdrojového portu, cílovou IP adresu a číslo cílového portu, počet paketů v toku a celkový počet bytů.

V prostředí sledování síťového provozu se používá několik odliš-

4. Více informací o NetFlow viz `http://www.cisco.com/go/netflow`, `http://en.wikipedia.org/wiki/NetFlow` nebo archiv závěrečných prací Masarykovy univerzity, `https://is.muni.cz/thesis/`.

ných implementací formátu záznamů o síťových tocích. Kromě NetFlow verze 5 a verze 9 se jedná například o protokol IPFIX[5], protokol sFlow[6] nebo protokol JFlow vyvinutý společností Juniper[7].

Jistě existují i další způsoby sledování provozu na síti a jeho vyhodnocování, NetFlow protokol však představuje kompromis mezi snahou maximalizovat množství informací určených ke zpracování a snahou snížit množství dat tak, aby byly zpracovatelné v reálném čase.

3.2 Analýza chování síťových zařízení

Behaviorální analýza počítačové sítě (*Network Behavioral Analysis*, *NBA*) je disciplína zabývající se odvozováním faktů, které nemusí být na první pohled zřejmé, z informací o chování zařízení na síti. Jako vstupní data pro tuto analýzu typicky slouží právě výše zmíněné záznamy protokolu NetFlow.

Behaviorální analýza umožňuje díky sledování profilů síťových zařízení určit nákazu škodlivým programem i v případě, že daný program ještě není známý a v databázích antivirových programů tak nejsou uloženy jeho signatury. Pokud se jedná o škodlivý program využívající ke své činnosti počítačovou síť, je možné jej odhalit právě pomocí behaviorální analýzy (nakažená stanice začne vykazovat odchylky ve svém síťovém chování).

Máme-li například výše zmíněný červ Morto (viz 3) a pominemeli počáteční skenování portu číslo 3389 (v případě nakažení stanicí z vnější sítě není tento sken odchylkou v chování stanic ve sledované síti), projeví se tento červ díky komunikaci na portu číslo 53 se servery, které nakažená stanice dosud nepoužila pro překlad doménových jmen (počítače tradičně používají jeden až dva servery poskytující službu DNS, proto je jakákoliv odchylka od tohoto chování podezřelá). Po komunikaci s řídícími servery navíc červ začne skenovat další stanice v okolí.

Stanici infikovanou červem *Morto* je tak možné objevit pouze na základě podezřelých změn v chování i v případě, že samotný červ

5. Více viz `http://datatracker.ietf.org/wg/ipfix/`.
6. Více viz `http://blog.sflow.com/2011/01/presentation.html`.
7. Více viz `http://www.juniper.net`.

není dosud znám.

Behaviorální analýza se neomezuje pouze na odhalování odchylek v chování stanic, tato disciplína se zabývá i hledáním známých vzorů chování, jakými se na síti projevují některé specifické programy. Takovými programy mohou být třeba klienti různých protokolů určených pro IM (např. OSCAR[8], XMPP[9]). Odhalení skenování je také z velké části založeno na hledání známých vzorů.

Jelikož jsou pro behaviorální analýzu počítačové sítě často dostupná data popisující chování všech stanic na síti (chování celé sítě), je možné pro odhalení bezpečnostního incidentu použít i metod odhalování odlehlých bodů (*Outlier Detection*) používaných v oblasti strojového učení. V okamžiku, kdy jedna stanice (případně skupina stanic) vykazuje výrazné odlišnosti chování oproti ostatním stanicím sledované sítě, je možné, že byla narušena bezpečnost. Rozdílnost chování může být způsobena i v odlišné konfiguraci daných stanic. Pokud však správce sítě o těchto rozdílech v konfiguraci neví, je pro něj informace o odchylce v chování důležitá.

3.3 Aplikace ADS

Aby byla behaviorální analýza v praxi použitelná, je nutná její automatizace. Za tímto účelem je vyvíjena aplikace *Anomaly Detection System* [20] (dále jen *ADS*) společnosti AdvaICT, a.s. Tato aplikace sbírá NetFlow záznamy ze sledované sítě a v pětiminutových intervalech provádí jejich vyhodnocování prostřednictvím takzvaných detekčních metod.

Detekční metody jsou podprogramy používané pro odhalování bezpečnostních anomálií na základě NetFlow záznamů případně na základě dlouhodobých statistik. Jednotlivé metody mohou průběžně vytvářet profily jednotlivých stanic ve sledované síti nebo statistiky sítě, případně mohou generovat události popisující odhalené bezpečnostní incidenty.

Událost je v aplikaci ADS reprezentována n-ticí tvořenou kódem detekční metody, která událost odhalila, IP adresou původce (zdroje) události, IP adresami cílů událostí, časovou známkou, ke které se

8. Protokol používaný službami ICQ a AIM [18].
9. Protokol používaný službami Jabber a Google Talk [19].

událost vztahuje (časová známka události většinou odpovídá časové známce prvního síťového toku, který se k dané události vztahuje, výjimkou jsou události detekčních metod založených na souhrnných statistikách za delší časové období), identifikátorem zdroje dat, který vygeneroval relevantní NetFlow záznamy, seznamem doplňujících atributů a jejich textovou reprezentací určenou obzvláště pro zobrazení uživateli (pro následné strojové zpracování slouží samotný seznam atributů).

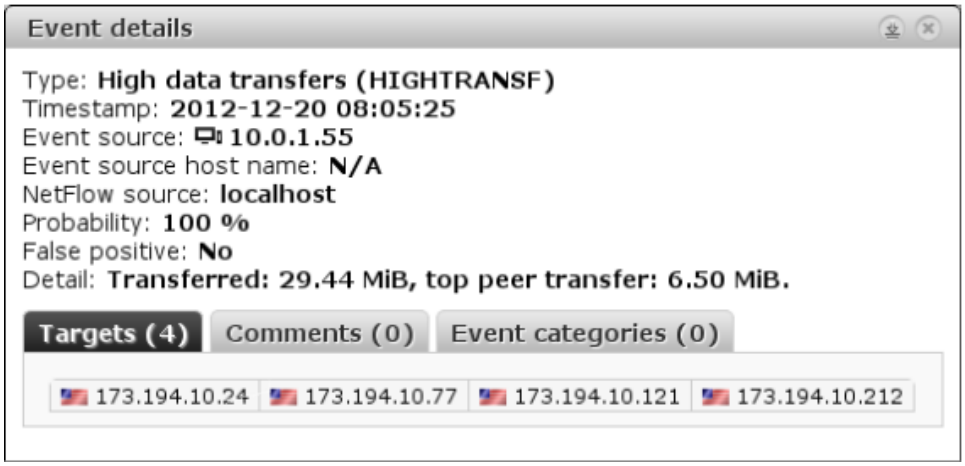
Kód metody slouží k jednoznačné identifikaci detekčních metod, je tvořen řetězcem nejvýše deseti velkých písmen a číslic. Časová známka události je ve formátu „RRRR-MM-DD hh:mm:ss“, zdroj dat je identifikován identifikačním číslem záznamu v databázi představujícího daný zdroj. Seznam atributů obsahuje pouze jejich hodnoty, sémantika atributu v každém poli seznamu je popsána pro každou metodu v rámci jejího záznamu v databázi.

Atributy mohou blíže specifikovat typ události (např. u detekční metody odhalující použití klienta některého z protokolů okamžité komunikace rozlišuje hodnota atributu *typ události* právě použitý protokol), objem dat přenesených danou stanicí (např. pro popis události upozorňující na zvýšený přenos dat), čísla portů (vhodné například při zjištěném DoS útoku pro specifikaci služby, která byla cílem útoku) a další.

Jako příklad si můžeme uvést událost vytvořenou detekční událostí sloužící pro odhalení skenování. Tato detekční metoda má kód *SCANS*, událost byla odhalena 23. listopadu 2012 přesně o půl sedmé večer („2012-11-23 18:30:00“) z NetFlow záznamů vygenerovaných zdrojem, jehož identifikační číslo v databázi je dva. Původcem události byla stanice s IP adresou 10.0.1.10, událost měla deset cílů a to stanice s IP adresami 10.0.1.11 až 10.0.1.20. U těchto stanic byl skenován port číslo 3389. Takovéto skenování se označuje jako *horizontální*. Pro skenování byly použity pakety, kterým nebyl nastaven žádný TCP příznak (jednalo se tedy o tzv. *NULL* skenování). Na třech ze skenovaných stanic je spuštěna služba naslouchající na daném portu, na ostatních je port s daným číslem filtrován firewallem. Seznam atributů dané události potom vypadá následovně:

```
(horizontal, NULL, 3389, 3, 7),
```

sémantika jednotlivých polí je zřejmá z popisu události výše. Pro



Obrázek 3.2: Zobrazení detailu události v aplikaci ADS [20].

uživatele se na základě výše zmíněných atributů vygeneruje následující zpráva: *Horizontal NULL scan (port: 3389, scans with response: 3, scans without response: 7)*. Tato zpráva slouží pouze pro čitelné zobrazení hodnot atributů, ostatní informace (IP adresa původce, IP adresy cílů, časová známka a identifikace zdroje dat) jsou zobrazeny odděleně.

Cílem této práce je navrhnout řešení, které by bylo schopné na základě událostí generovaných aplikací ADS odvozovat další informace. Příkladem takovéto odvozené informace (a vyšší úrovně abstrakce) může být například odhalení již zmíněného červa Morto (viz 3). Podklady pro odhalení nákazy tímto červem představují události vygenerované detekčními metodami *SCANS* a *DNSANOMALY*[10]. Pokud je těmito detekčními metodami vytvořena posloupnost událostí s příslušnými atributy odpovídající šíření červa Morto, může být tato posloupnost považována za dostatečný důkaz jeho výskytu.

Událost informující o nákaze červem Morto může uživateli sdělit mnohem více než zmíněná posloupnost událostí, které samy o sobě nejsou dostatečně významné, aby jim byl věnován dostatek pozor-

10. Detekční metoda *DNSANOMALY* slouží ke zjišťování nepředpokládaného využití služby DNS. Tím je i odeslání DNS požadavku na dosud nepoužívaný server.

nosti.

Data jsou v aplikaci ADS zpracovávána cyklicky. Cyklus začíná nahráním dávky NetFlow záznamů do databáze aplikace. Následně jsou data předzpracována (je provedeno filtrování záznamů a korelace síťových toků procházejících přes proxy server).

Po předzpracování dat dochází ke spouštění samostatných detekčních metod. Detekční metody jsou spouštěny paralelně, nejvyšší počet výpočetních vláken je však omezen licencí a uživatelským nastavením. Detekční metody jsou spouštěny vždy v daném pořadí, proto se může stát, že událost jedné metody, může být do databáze uložena dříve, než událost jiné metody, přesto, že má druhá událost nižší časovou známku. Na tuto skutečnost je nutné brát ohled při návrhu systému pro případ definice události vyššího řádu, jejíž součástí je posloupnost událostí detekčních metod s přesně daným pořadím. Události detekčních metod tedy nelze zpracovávat navrhovaným systémem okamžitě po jejich vložení do databáze.

Po zpracování NetFlow záznamů detekčními metodami jsou vytvořeny e-mailové zprávy informující o událostech s nejvyšší přiřazenou prioritou, dále jsou informace o událostech odeslány pomocí protokolů *SNMP* a *syslog* do případných nadřazených systémů sledujících stav sítě. Nakonec jsou smazány NetFlow záznamy příslušející do dané dávky.

Nahrání dat a následně celé zpracování probíhá v pětiminutových intervalech pro každý zdroj NetFlow záznamů.

3.4 Databázový systém Caché a programovací jazyk Caché ObjectScript

Aplikace ADS je implementována v programovacím jazyce Caché ObjectScript a pro svůj běh využívá databázový stroj Caché [22]. Kvůli jednomu z cílů této práce (použití navrhovaného řešení pro zpracování události vyšších řádů jako rozšíření aplikace ADS) by měl výsledný návrh respektovat specifické vlastnosti této platformy.

3.4.1 Databáze Caché

Databáze Caché bývá označována jako databáze postrelačního typu [23], jedná se o objektovou databázi s přímou projekcí do relačního systému, je tak možné použít dotazovací jazyk Caché SQL (jazyk SQL rozšířený o některé funkce programovacího jazyka Caché ObjectScript). Instance tříd jsou ukládány do stromové struktury zvané *globál*, ke které je možné přistupovat přímo (ne jen zprostředkovaně pomocí objektů). O Caché tak lze uvažovat také jako o hierarchické databázi.

Globál je řídké n-rozměrné pole, ale je vhodnější představovat si ho jako n-ární strom následujících vlastností. Každý uzel globálu může mít libovolný konečný počet potomků. Každý uzel globálu s výjimkou kořene má přiřazen klíč, ten musí být unikátní pouze mezi potomky stejného uzlu (tj. mezi sourozenci). Klíč může být reprezentován libovolným datovým typem, dokonce i seznamem a jinými binárními strukturami. Prázdný řetězec není povolená hodnota klíče globálu. Není nutné, aby byly klíče v jednom globálu stejného datového typu. Každý uzel může mít přiřazenu hodnotu, uzly, které jsou listy stromu, musí mít přiřazenu hodnotu. Všechny uzly, které mají společného rodiče, tvoří oboustranně zřetězený seznam seřazený podle klíčů. K procházení tohoto seznamu je určená funkce `$Order`. Funkce `$Order` přijímá dva argumenty, prvním je uzel globálu, druhým je směr průchodu pole (hodnota 1 nebo žádná pro nalezení klíče následujícího uzlu, hodnota -1 pro nalezení klíče předchozího uzlu). Funkce vrací klíč následníka (předchůdce) zadaného uzlu.

Příklad 3.1. Mějme globál *^VzorovyGlobal* definovaný následovně:

$$
\begin{aligned}
\hat{}VzorovyGlobal &= 1 \\
\hat{}VzorovyGlobal(1) &= 2 \\
\hat{}VzorovyGlobal(2) &= 3 \\
\hat{}VzorovyGlobal(\text{"}a\text{"}) &= 4 \\
\hat{}VzorovyGlobal(1,1) &= 5 \\
\hat{}VzorovyGlobal(1,5) &= 6 \\
\hat{}VzorovyGlobal(1,10,1,2) &= 7
\end{aligned}
$$

Potom můžeme funkci `$Order` použít takto:

$$\begin{aligned} \$Order(\hat{}VzorovyGlobal(1)) &= 2 \\ \$Order(\hat{}VzorovyGlobal(\text{"}a\text{"}), -1) &= 2 \\ \$Order(\hat{}VzorovyGlobal(1, \text{""})) &= 1 \\ \$Order(\hat{}VzorovyGlobal(1, \text{""}), -1) &= 10 \end{aligned}$$

V příkladu lze vidět, že hledáme-li první (případně poslední) klíč daného oboustranně zřetězeného seznamu, musíme hledat následníka (případně předchůdce) nedefinovaného uzlu s klíčem odpovídajícím prázdnému řetězci. Globály se dají využít i k řazení dat, požadované uspořádání však musí odpovídat uspořádání, ve kterém jsou řazeny klíče uzlů se společným rodičem (řetězce jsou řazeny v pořadí číslice, velká písmena, malá písmena).

3.4.2 Programovací jazyk Caché ObjectScript

Součástí databázového serveru Caché je i aplikační server, který slouží jako interpret programovacího jazyka Caché ObjectScript. Tento jazyk patří do třídy slabě typovaných objektově orientovaných programovacích jazyků.

Důležitou vlastností jazyka, kterou je možné využít při návrhu systému zpracovávajícího události vyšších řádů, je možnost měnit zdrojový kód programu v rámci jeho běhu, tzv. *reflexivita programovacího jazyka*. Reflexivita jazyka umožní, aby byly uživatelem zadané definice událostí vyšších řádů přeloženy do programovacího jazyka Caché ObjectScript a v této podobě dále zpracovávány. Není tak nutné při každé zpracovávané dávce dat definici opakovaně interpretovat (navíc v již interpretovaném jazyku).

4 Obecný popis jazyka

Jednou z cest zpracování velkého množství dat (obzvláště datových proudů) je výše zmíněná oblast CEP (viz 2.2), ve které se využívá dotazovací jazyk k abstrakci nad surovými daty. Vzhledem k požadavkům na rychlost daných zamýšleným nasazením implementace návrhu této práce nelze využít žádný existující systém zpracování komplexních událostí, volání externího programu by způsobovalo časové prodlevy. Proto je součástí této práce návrh dotazovacího jazyka DAQL[1] (*Data Abstraction Query Language*) a příslušného překladače, který bude integrován do výše popsané aplikace ADS (viz 3.3).

Ačkoliv bude zamýšlené nasazení navrhovaného jazyka poměrně specifické, je jazyk navržen tak, aby byl jednoduše rozšiřitelný pro libovolné použití. Tato a následující kapitoly se zabývají popisem základu jazyka DAQL a jeho konkrétním rozšířením pro zpracování událostí vyššího řádu v prostředí bezpečnosti počítačových sítí. Rozšiřující prvky jazyka jsou zřetelně odlišeny od základní sady konstrukcí[2]. Způsob rozšiřitelnosti jazyka je popsán níže (viz příloha F).

V této kapitole jsou nejdříve definovány některé termíny, které jsou dále používány. Dále je krátce vysvětlen princip fungování vyhodnocení dotazu, jsou popsány základní formulace jazykem používané a následně jsou představeny konstrukce projekce a konstrukce sledů včetně způsobu zápisu podmínek.

4.1 Definice pojmů

Před samotným popisem jazyka DAQL je nutné definovat několik pojmů, které jsou v následujícím textu používány. Definice jsou záměrně uvedeny nezávisle na rozšíření.

---

1. Zkratka a název byly zvoleny tak, aby nedocházelo k záměně s jiným dotazovacím jazykem. Zkratku lze zjednodušeně vyslovovat jako *dakil*.
2. Pro jednotlivé lexikální jednotky jazyka DAQL je v textu použit výraz *konstrukce*, protože jednotlivé dotazy *konstruují* množinu, jejíž prvky jsou daným dotazem popsány.

4.1.1 Událost

Definice 4.1. Událost je zpráva o ději, který má jednoznačně určenu neprázdnou množinu původců, množinu cílů (ne nutně neprázdnou), množinu dvojic atribut-hodnota a vztahuje se k jednomu časovému okamžiku. Tyto informace jsou obsaženy v dané zprávě.

Pro praktické potřeby je vždy třeba uvažovat události s neprázdnou množinou dvojic atribut a hodnota. Máme-li například událost, jejímž původcem je Jan a má časovou známku odpovídající třetímu lednu roku 2012, devíti hodinám a dvaceti minutám, a víme, že událost je vygenerována vždy v okamžiku, kdy zaměstnanec přichází do práce, poskytuje tato událost všechny požadované informace. Pokud však nemáme znalost kontextu vzniku události, nezjistíme z dané události žádné informace.

Příklady splňující definici:

NetFlow záznam Zdrojová IP adresa identifikuje původce dané události, cílová IP adresa určuje cíl události, čas události je dán časovou známkou záznamu. Dvojice „číslo portu“ a konkrétní hodnota čísla portu představuje jednu z dvojic atribut-hodnota dané události.

Událost aplikace ADS Zdrojová IP adresa identifikuje původce dané události. IP adresy cílů jsou současně identifikátory cílů události, časová známka odpovídá časové známce události. Seznam atributů rozšířený o dvojice *nfcapd* a identifikace zdroje NetFlow dat a *code* a kód metody, která událost vygenerovala, tvoří atributy dané události.

Jedna hodnota elektrokardiogramu Pacient, kterému je EKG měřeno, představuje původce události, čas události odpovídá času daného měření a dvojice atribut-hodnota je reprezentována dvojicí „Hodnota“ a aktuálně naměřená hodnota.

Původcem této události může být také senzor, které dané měření zaznamenal, identifikace pacienta by pak měla být součástí událostí v podobě jedné z dvojic atribut-hodnota.

Měření aktuální výšky hladiny Zdrojem události je senzor, který naměřil danou hodnotu a odeslal danou zprávu, čas události odpovídá času měření a hodnota měření spolu s pojmenováním (např. opět „Hodnota“) zastupují dvojici atribut a hodnota.

4.1.2 Řád události

Definice 4.2. Řád události reprezentuje míru abstrakce nad zdrojovými daty. Událost, která odpovídá zdrojovým datům je událost nultého řádu. Událost vytvořená nad událostmi nejvýše řádu n je řádu $n + 1$. Řád události může nabývat několika různých hodnot, obecně je uvažována nejvyšší z nich.

Vytvořením události je myšlen proces, který za použití dat z existujících událostí vytvoří událost novou. Vytvořením události může být vytvoření kopie stávající události a její následná modifikace (rozšíření této kopie o dodatečné dvojice atribut-hodnota, odebrání některých atributů), případně sloučení několika stávajících událostí do jedné některým ze způsobů popsaných dále v této kapitole (viz například konstrukce sledů, 4.1 a 4.2.8). Pokud jsou události, ze kterých je odvozena nová událost, řádu n, je nově vytvářená událost řádu $n + 1$.

Příklady splňující definici

NetFlow záznam V prostředí, kde jsou NetFlow záznamy zdrojovými daty a neexistuje možnost, jak získat podrobnější informace, je NetFlow záznam událostí nultého řádu. V případě, že jsou zdrojovými daty jednotlivé pakety, je NetFlow záznam událostí prvního řádu. V dalších příkladech budeme uvažovat prostředí, ve kterém jsou NetFlow záznamy zdrojovými daty.

Událost *Vysoký přenos dat* Událost představuje abstrakci nad několika záznamy NetFlow se společnou zdrojovou (případně cílovou) IP adresou, u kterých součet přenesených dat překročil stanovený práh. Jelikož se jedná o událost vytvořenou pouze na základě NetFlow záznamů, je tato událost řádu jedna.

Událost *Nepoučitelný stahovač* Tato událost informuje o zařízení, které v síti pravidelně přenáší (stahuje) velké množství dat.

Událost může být založena přímo na NetFlow datech, nebo také na opakovaném výskytu události *Vysoký přenos dat*, proto je řád této události jedna a současně i dva (dále je uvažována pouze nejvyšší hodnota, tedy dva).

4.1.3 Sled událostí

Definice 4.3. Sled událostí je libovolná posloupnost událostí určená jejich časy.

Věta 4.1. *Sled událostí je událost.*

Tato věta je dokázána v následující kapitole (viz 5). Z věty a předchozích definicí vyplývá, že sled je jedním ze způsobů vytváření událostí a má tak přiřazen řád události. Z této věty dále plyne, že je možné vytvořit sled, jehož dílčí událostí je opět sled.

Příklady splňující definici:

Náhodná posloupnost NetFlow záznamů Podle definice sledu musí být tato posloupnost uspořádaná podle časových známek.

Elektrokardiogram za poslední hodinu

Soubor měření výšky hladiny vždy v poledne za poslední měsíc

4.2 Základní konstrukce jazyka

Navržený dotazovací jazyk je záměrně podobný jazyku SQL [24]. Podobnost umožní uživatelům se základními znalostmi relačních databází snazší pochopení navrhovaného jazyka. Nevýhodou této podobnosti je možná záměna klíčového slova *SET* jazyka SQL (přiřazení hodnoty) a *SET* navrhovaného jazyka DAQL (konstrukce množiny, viz 4.2.8).

Zásadním rozdílem mezi jazyky SQL a DAQL je reprezentace dat, se kterými dané jazyky manipulují. Jazyk SQL pracuje nad definovaným systémem relací reprezentovaným tabulkami, jazyk DAQL zpracovává teoreticky nekonečný proud událostí (podle formální definice se opět jedná o relace, viz 5.1). Obecně platí, že tyto události

nejsou nikde trvale ukládány[3], proto je nutné je zpracovat v okamžiku doručení do systému (více viz 2.2). Vzhledem k tomuto rozdílu se jazyk DAQL v dotazech neodkazuje na logické datové struktury (relace, tabulky), ale v průběhu dotazu vytváří vlastní abstraktní datové struktury s využitím konstrukcí *SET* (viz 4.2.8) a *SEQUENCE* (viz 4.2.8).

Tento text si neklade za cíl být ucelenou učebnicí jazyka DAQL, ale spíše průvodcem pro uživatele se znalostmi dotazovacích jazyků (například SQL). Proto je zde tento jazyk pouze popsán na úrovni jednotlivých konstrukcí. Syntaxe jazyka je důsledně popsána v následující kapitole (5.2).

4.2.1 Dotaz v jazyce DAQL

Dotaz jazyka DAQL představuje projekci ze sledu událostí (reprezentovaného konstrukcemi *SET* a *SEQUENCE*) do nově vytvářené události. O příslušnosti události do daného sledu rozhodují vstupní podmínky definované dotazem. Projekce ze sledu do události je provedena pouze v případě, že jsou splněny podmínky ukončení sledu (výstupní podmínky a podmínka ukončení, viz 4.2.8).

Dotaz může být chápán jako stavový automat: nekonečný proud dat (událostí) lze reprezentovat jako nekonečné vstupní slovo, které automat čte. Vždy, když některá událost splní vstupní podmínku platnou v daném stavu, posune se automat do dalšího stavu. Vstupní podmínky tak představují přechodovou funkci tohoto automatu. Jakmile jsou splněny všechny podmínky ukončení, přejde automat do koncového stavu, kde jsou vyhodnoceny výstupní podmínky a případně je provedena projekce události. Pokud se v průběhu výpočtu automatu objeví událost splňující vstupní podmínku platnou v prvním stavu automatu, je vytvořen automat paralelně zpracovávající stejný dotaz (více viz kapitola o překladači jazyka DAQL, 6). Tyto automaty jsou v této kapitole nazývány jako *instance dotazu*.

Definice 4.4. Posloupnost událostí, které splnily vstupní podmínky, je po dobu zpracování dotazu uložena v paměti. Tuto posloupnost si pro další potřeby nazvěme jako *posloupnost splňujících událostí*.

3. V případě aplikace ADS jsou události po omezenou dobu uloženy v databázi.

4.2.2 Datové typy

Datové typy představují základní prvek pro rozšiřování dotazovacího jazyka DAQL. V základu jazyka není dostupný žádný datový typ, veškeré datové typy jsou rozšířením. Součástí popisu datového typu je definice formátu zápisu možných hodnot daného datového typu, definice n-árních funkcí, které jsou na hodnoty daného datového typu aplikovatelné (například funkce sinus, exponenciální funkce, funkce zřetězení, rozdíl mezi dvěmi časovými známkami a podobné), definice relací ($=$, $\leq$, atp.) a definice agregačních funkcí aplikovatelných na hodnoty datového typu (například suma, průměr, seznam různých hodnot, atp.).

Veškeré datové typy, které se dále v textu vyskytnou, jsou součástí rozšíření pro použití v prostředí bezpečnosti počítačových sítí.

4.2.3 Konstrukce *EVENT*, *EVENT(n)* a *EVENT(m,n)*

Konstrukce *EVENT*, *EVENT(n)* a *EVENT(m,n)* jsou speciální proměnné, pomocí kterých se lze odkazovat na zpracovávané události, respektive na události splňující vstupní podmínky dotazu.

Proměnná *EVENT* obsahuje poslední, tj. aktuálně zpracovávanou událost. Proměnné *EVENT(n)* a *EVENT(m,n)* umožňují přístup k posloupnosti splňujících událostí. Proměnná *EVENT(n)* tak obsahuje n-tý prvek dané posloupnosti, proměnná *EVENT(m,n)* obsahuje podposloupnost splňující posloupnosti, od m-tého po n-tý prvek, včetně. Místo čísla n lze použít také symbol „*“ (hvězdička). Použití hvězdičky odkazuje na poslední prvek seznamu. Hvězdičku lze kombinovat také s odečtením konstantní hodnoty. Máme-li tedy například proměnnou *EVENT(*−1)*, odkazuje se tato proměnná na předposlední prvek posloupnosti splňujících událostí.

Místo proměnné *EVENT(1,*)* lze použít makro příkaz *EVENTS*. Je třeba poznamenat, že proměnné *EVENT(*)* a *EVENT* neobsahují stejnou událost.

4.2.4 Přístup k datům událostí

Pro potřeby dotazu je nutné přistupovat přímo k datům, které popisují danou událost. K projekci jednotlivých vlastností (původci,

cíle, časová známka a hodnoty jednotlivých atributů) je využita takzvaná *tečková syntaxe*, která se používá zejména v objektově orientovaných programovacích jazycích. Pro projekci jednotlivých vlastností lze použít klíčová slova *sources*, *targets*, *timestamp*, pro projekci hodnot atributů je nutné použí identifikátor konkrétního atributu.

Z předchozího také plyne, že klíčová slova *sources*, *targets* a *timestamp* nesmějí být použity jako identifikátory atributů, aby se zabránilo nejednoznačnosti.

Tečkovou syntaxi je možné v případě sledu událostí splňujícího ukončující podmínky využít i pro přístup k atributům jednotlivých událostí, ze kterých se sled skládá. Vzhledem k formální definici sledu (viz 5.3) není možné rozlišit příslušnost původců a cílů k jednotlivým událostem. Dále není možné získat časové známky jednotlivých událostí. Více viz 4.2.8.

Takto je možné sestoupit na úroveň jednotlivých událostí i v případě *sledu sledů* a dále (induktivně).

Dalším využitím tečkové syntaxe je aplikace agregačních funkcí na hodnoty atributů dílčích událostí sledu. Mějme sled *sled*, jehož dílčí události obsahují atribut *atribut*. Pro datový typ daného atributu je definována agregační funkce *f*. Aplikaci funkce na hodnoty atributů dílčích událostí lze zapsat pomocí řetězce *sled.f(atribut)* (viz dotaz 4.7, řádek 6).

V případě, že máme k-tici sledů *sled(m,n)* (zápis má totožný význam jako zápis *EVENT(m,n)*), aplikace funkce, pro jejíž zápis byl použit řetězec *sled(m,n).f(atribut)*, vrací k-tici hodnot. Ta může být opět zpracována agregační funkcí (viz dotaz 4.7, řádek 24).

Příklad 4.1. Mějme v proměnné *EVENT* událost, jejíž původce je identifikován IP adresou *10.0.1.2*, cíl je identifikován adresou *10.0.1.1*, časová známka je dána ve formátu POSIX[4] jako *1352309353*. Tato událost má dva atributy: kód události (*code*), jehož hodnotou je řetězec „ATTACK“, a počet dílčích útoků (*count_of_attacks*), jehož hodnota je rovna šesti.

4. Počet sekund od 1. ledna 1970 00:00:00 UTC

Jednotlivé projekce potom vypadají následovně:

$$\begin{aligned} EVENT.sources &= 10.0.1.2 \\ EVENT.targets &= 10.0.1.1 \\ EVENT.timestamp &= 1352309353 \\ EVENT.code &= ATTACK \\ EVENT.count\_of\_attacks &= 6 \end{aligned}$$

Příklad 4.2. Mějme sled dvou událostí pojmenovaný (viz 4.2.5) jako *sled* splňující podmínky ukončení. První událost (shodná s událostí v předchozím příkladu) je v rámci sledu přejmenována na *attack_1*, druhá událost, která se liší pouze časovou známkou (*1352309396*) a hodnotou atributu počet dílčích útoků (*11*), je přejmenována na *attack_2*.

Vybrané projekce potom vypadají následovně:

$$\begin{aligned} sled.attack\_1.count\_of\_attacks &= 6 \\ sled.attack\_2.count\_of\_attacks &= 11 \end{aligned}$$

4.2.5 Pojmenování

Pro intuitivnější zápis a lepší čitelnost výsledného dotazu je vhodné v rámci sledů používat pojmenování jednotlivých událostí, na které by se měl dotaz později odkazovat (viz 4.1). K tomu slouží funkce *AS* zapisovaná infixově.

Formát jmen, která jsou událostem (respektive atributům) přiřazována, musí respektovat následující požadavky. Jméno může obsahovat alfanumerické znaky (bez diakritiky), dále může obsahovat znaky „–“ a „_“. První znak jména musí být písmeno. Pojmenování je citlivé na velikost písmen, tudíž jména *name* a *Name* jsou brána jako rozdílná. Jako jméno nelze použít žádné z klíčových slov jazyka, včetně klíčových slov použitých rozšíření jazyka (např. jména funkcí definovaných pro dané datové typy).

Pokud se dotaz odvolává na nepřiřazené jméno, je překladačem označený jako chybný a příslušná chyba je označena.

Identifikátor, který je pomocí funkce pojmenování přiřazen události, může být použit obdobně jako proměnná *EVENT*. Máme-li událost (ve skutečnosti konstrukci události, vstupní podmínku, viz 4.2.8)

pojmenovanou jako *udalost* a při zpracování dotazu vyhoví její definici n událostí, potom lze použít zápis *udalost(i)* pro odkázání se na i-tou událost, která vyhověla definici. Zápisy *udalost(*)*, *udalost(i,j)* a *udalost(i,*)* jsou také povoleny. Ekvivalent proměnné *EVENTS* neexistuje, je nutné použít zápis *udalost(1,*)*.

Příklady využití jsou ukázány u následujících konstrukcí.

4.2.6 Funkce

V základu jazyka DAQL jsou, vzhledem k neexistenci datových typů, definovány pouze tři funkce, jejichž definičním oborem je množina jednotlivých událostí, respektive množina sledů událostí.

Mezi tyto základní funkce patří již zmíněné funkce projekce (viz 4.2.4) a pojmenování (viz 4.2.5). Třetí funkcí je funkce určení mohutnosti množiny událostí (množina událostí je zde reprezentována sledem). Zápis této funkce je *COUNT(p)*, kde *p* je zápisem podmínky (viz 4.2.8). Podmínka musí být funkci předána jako řetězec, tedy uzavřená do jednoduchých uvozovek (apostrofů).

Funkce vrací počet událostí ve sledu nejvyšší úrovně (viz 5.3 a 4.2.8), které splňují podmínku předanou funkci jako argument. Podmínku lze nahradit symbolem „*“ (hvězdička), do výpočtu jsou pak zahrnuty všechny události daného sledu. Další možností zápisu podmínky je použití identifikátoru, který byl funkcí pojmenování přiřazen události. V tomto případě funkce vrátí počet všech událostí zpracovaných danou instancí dotazu, které vyhovují definici dané události.

Funkce *COUNT(p)* je agregační funkce, proto může být aplikována s využitím tečkové syntaxe i na dílčí události sledu (viz 4.2.4).

4.2.7 Konstrukce *SELECT*

Konstrukce *SELECT* slouží k definici události, která je výstupem daného dotazu. V rámci konstrukce je nutné definovat vždy alespoň původce události a časovou známku. Veškeré hodnoty popisující vytvářenou událost je nutné pojmenovat pomocí funkce *AS*. Pro pojmenování základních vlastností události (původci, cíle, časová známka) je třeba použít klíčová slova *sources*, *targets* a *timestamp*.

Možná podoba popisu výstupní události dotazu je ukázána ve vzorovém dotazu 4.1.

Dotaz 4.1 Popis výstupní události

```
1: SELECT
2:     MIN(timestamp) AS timestamp,
3:     attack_1.sources AS sources,
4:     attack_2.sources AS targets,
5:     'ATTACK' AS code,
6:     COUNT(*) AS attack_count
7: FROM
8: ...
```

Výstupní událost dotazu 4.1 tak má časovou známku odpovídající minimální časové známce sledu (řádek 2), za původce události jsou označeni původci jedné z dílčích událostí sledu, která je pojmenována jako *attack_1* (řádek 3) a cíle této události jsou shodné s původci dílčí události sledu, která je pojmenována jako *attack_2* (řádek 4). Takovéto přiřazení je možné pouze v rozšířeních jazyka, ve kterých datový typ použitý pro identifikaci původců události odpovídá datovému typu použitému pro identifikaci cílů události. Jelikož je dotaz aplikován na nejvyšší úrovni vždy právě na jeden sled (ten se však může skládat z dalších sledů, viz 4.2.8), je možné odvolávat se přímo na *attack_1* místo na *„sled“.attack_1*. Ze stejného fůvodu je možné zapisovat aplikaci agregační funkce jako *MIN(timestamp)* (funkce se následně provede nad všemi událostmi sledu).

Takto definovaná událost má přiřazeny dva atributy: atribut *code* (řádek 5 dotazu 4.1) s přiřazenou hodnotou *'ATTACK'* a atribut *attack_count* (řádek 6 dotazu 4.1), kterému je jako hodnota přiřazen výstup funkce *COUNT(*)*, která vrací počet dílčích událostí sledu, na který je dotaz aplikován.

Rozšíření jazyka

Dalším způsobem rozšíření jazyka (mimo datových typů) je definice povinných atributů popisujících událost. V případě rozšíření pro zpracování událostí z prostředí síťové bezpečnosti jsou takovýmito povinnými atributy atribut *code* a atribut *template*.

Hodnota atributu *code* slouží k identifikaci dotazu, který danou

událost vygeneroval. Množinu možných hodnot (datový typ) atributu *code* představuje množina všech řetězců velkých písmen délky nejvýše deset. Pro tento datový typ je definována pouze relace ekvivalence „=“ a agregační funkce *LIST* a *DISTINCT*, které pro množinu událostí vrátí čárkami oddělený seznam hodnot atributu, respektive čárkami oddělený seznam, ve kterém se žádná hodnota neopakuje.

Hodnota atributu *template* slouží pro výpis informací o události tak, aby byly lidsky čitelné. Datový typ je představován množinou řetězců alfanumerických znaků bez diakritiky a znaků „-“, „:“, „(“, „)“, „=“, „.“, „!“, „?“, „,“, „;“, „_“ (mezera) a „_“. Dále je možné použít znak „%“ zřetězený s názvem některého z dalších atributů. Tímto se definuje proměnná, která je při výpisu nahrazena hodnotou odpovídajícího atributu události. Pro tento datový typ je definována pouze relace ekvivalence „=“.

4.2.8 Konstrukce sledů

Dotazovací jazyk DAQL pracuje pouze s jedním vstupním datovým proudem a s jedním proudem výstupním (případné rozdělení do více datových proudů je možné navazujícím systémem v rámci dodatečného zpracování). Proto nejsou možné operace analogické ke kartézskému součinu tabulek, vnitřnímu spojení a dalším, které jsou známé z relačních databází. Jako náhrada těchto operací, umožňujících definici závislostí mezi více událostmi (záznamy, v případě relačních databází), slouží konstrukce sledů. Pomocí konstrukcí sledů lze definovat vzájemné závislosti mezi jednotlivými událostmi zvolené posloupnosti.

V jazyce DAQL jsou definovány dvě konstrukce konstruující sledy událostí. První konstrukce, *SET*, konstruuje množinu událostí, tj. libovolnou permutaci n-tice událostí. Druhou konstrukcí je konstrukce posloupnosti (*SEQUENCE*) konstruující n-tici událostí, u kterých je přesně dáno jejich pořadí. Konstrukci posloupnosti je možné definovat i pomocí konstrukce množiny (viz 4.5), avšak pro lepší přehlednost výsledného dotazu a uživatelskou přívětivost jazyka DAQL byla zavedena i konstrukce posloupnosti.

U obou zmíněných konstrukcí sledů lze pro jednotlivé události definovat vstupní podmínky, podmínky ukončení sledu a výstupní podmínky. Vstupní podmínky slouží k určení příslušnosti události

k danému sledu (výběr podposloupnosti z posloupnosti všech událostí), podmínka ukončení definuje okamžik, ve kterém je sled úplný a je možné ho předat k dalšímu zpracování. Výstupní podmínky definují podmínky platné pro úplný sled. V případě splnění podmínky ukončení a současném nesplnění výstupních podmínek končí výpočet dané instance dotazu neúspěchem.

Místo jednotlivých událostí v rámci konstrukce sledu lze použít i další konstrukce sledů (díky platnosti věty 4.1). Sledu, který není součástí jiného sledu, říkáme *sled nejvyšší úrovně* (možno zaměňovat výrazy *množina nejvyšší úrovně*, případně *posloupnost nejvyšší úrovně*). Jelikož výstupem dotazu jazyka DAQL je opět událost, lze jako dílčí událost daného sledu použít i vnořený dotaz.

Při přístupu k hodnotám dílčích událostí sledu nejvyšší úrovně není nutné používat tečkovou syntaxi.

V případě, že sled (například *sled1*) reprezentuje dílčí událost jiného sledu (tj. mimo sled nejvyšší úrovně), pohlíží se na *sled1* jako na událost. Nelze tak rozlišit, ke které dílčí události sledu *sled1* se vztahují jednotliví původci a cíle. Celý sled může mít přiřazenu pouze jednu časovou známku. Množina původců (cílů) události *sled1* je sjednocením množin původců (cílů) všech dílčích událostí sledu. Časová známka události *sled1* je definována jako minimální časová známka všech dílčích událostí sledu. Atributy lze rozlišit podle jejich příslušnosti k dílčím událostem sledu (viz 5.2), proto lze použít aplikaci agregačních funkcí pomocí tečkové syntaxe (viz 4.2.4).

Podmínky, logické výrazy

Pro zápis podmínek v jazyce DAQL slouží klauzule predikátové logiky prvního řádu. Univerzem pro zápis podmínek je množina všech událostí. Termem logického výrazu podmínky může být prvek některého z datových typů, funkce aplikovaná na proměnnou odkazující se na událost (samotná proměnná není termem) nebo funkce aplikovaná na n termů.

Dobře utvořené formule jsou tvořeny zápisem relací, definovaných nad danými datovými typy. V rámci definic relací nad datovými typy je nutné definovat alespoň jednu relaci ekvivalence. Pokud je φ formule, pak i negace této formule, *NOT* φ, je formule. Pokud φ a ψ jsou formule, pak i φ *AND* ψ a φ *OR* ψ jsou formule. Je-li φ for-

mule a *X* je proměnná odkazující na posloupnost událostí (například proměnné *EVENTS* a *EVENT(m,n)*), potom i výrazy zapsané jako *FOR ALL X* φ a *FOR SOME X* φ jsou formule. Dále platí, že je-li φ formule, potom i *(*φ*)* je formule. Pokud je za *X* dosazeno libovolné *Y(1,*)*, je možné zápis indexů vynechat.

Formulemi jsou i výrazy zapsané jako *SAME sources FOR ALL X*, *SAME targets FOR ALL X* a v případě, že *t* je term reprezentující funkci aplikovanou na časovou značku (například identita, projekce časové známky pouze do hodin a jiné), pak i výraz se zápisem *SAME t FOR ALL X* je formulí. V případě prvních dvou výrazů (*SAME sources* a *SAME targets*) lze klíčové slovo *SAME* rozšířit na klíčovou frázi *STRICT SAME*.

Formuli *SAME sources FOR ALL X* (*SAME targets FOR ALL X*) lze použít pouze jako součást vstupní podmínky. Tato formule je událostí splněna, má-li množina původců (cílů) dané události neprázdný průnik s průnikem množin původců (cílů) všech *X* patřících do splňující posloupnosti. Musí tak existovat alespoň jeden původce, který je původcem každé z událostí *X* patřící do splňující posloupnosti. V případě použití fráze *STRICT SAME* musí být množiny původců (cílů) všech *X* patřících do splňující posloupnosti totožné (tj. včetně právě zpracovávané události).

Formuli *SAME t FOR ALL X* lze opět použít pouze v rámci vstupní podmínky. Tato formule omezí události patřící do posloupnosti splňujících událostí pouze na ty, které mají stejné *t*.

V rámci formulí využívajících fráze *FOR ALL* a *FOR SOME* je možné zapisovat projekci *EVENT.x* pouze jako *x*.

Vyhodnocování podmínek Pravdivost formulí je založena na definici použitých relací. Formule obsahující pouze zápis relace je pravdivá právě tehdy, když je daná k-tice v této relaci. Pokud se tato formule odkazuje na atribut, který nemá právě zpracovávaná událost definovaný, je tato formule nepravdivá[5].

Formule *NOT* φ je pravdivá právě tehdy, je-li φ nepravdivá. Formule φ *AND* ψ je pravdivá právě tehdy, jsou-li pravdivé obě formule φ i ψ. Formule φ *OR* ψ je nepravdivá právě tehdy, jsou-li obě

5. I v případě, že je formule obecně tautologií – například porovnání hodnoty atributu (relace $\geq$) s nejmenší hodnotou daného datového typu

formule, φ a ψ, nepravdivé. Logické spojky *AND* a *OR* jsou vyhodnocovány líně, tj. pokud je formule φ ve formuli φ *OR* ψ pravdivá, není formule ψ vyhodnocována. Formule ψ není vyhodnocována ani v případě, kdy je formule φ ve formuli φ *AND* ψ nepravdivá.

Formule *FOR ALL X* φ je pravdivá právě tehdy, když platí formule φ pro každé *X*, které patří do posloupnosti splňujících událostí (*X* může být například proměnná *EVENT*, *EVENT(i,j)*, nebo také identifikátor, který je přiřazen událostem funkcí pojmenování). Formule *FOR SOME X* φ je pravdivá právě tehdy, platí-li φ pro alespoň jedno *X* patřící do posloupnosti splňujících událostí.

Rozšíření jazyka Veškeré relace používané v podmínkách jsou definovány současně s datovými typy. V příkladech v této práci jsou použity relace $=$ (relace ekvivalence), relace $<=$ a $>=$ (relace uspořádání, menší nebo rovno a větší nebo rovno), dále ternární relace *BETWEEN* ($(X, Y, Z) \in BETWEEN \Leftrightarrow X \geq Y \wedge X \leq Z$) a relace $<$ a $>$ (asymetrické a tranzitivní relace, je menší a je větší).

Vstupní podmínky Vstupní podmínky představují omezení definující události, které můžou patřit do daného sledu (můžou být prvky jeho posloupnosti splňujících událostí). Vstupní podmínka konstruuje události, které danou podmínku splňují. V jazyce DAQL jsou rozlišeny dva způsoby zápisu vstupní podmínky.

Prvním způsobem zápisu je zápis podmínky aplikovatelné pouze na jednu událost. Tato podmínka se zapisuje pomocí fráze *EVENT FILTERED BY* φ (viz dotaz 4.2, řádek 5), kde φ je logická formule, která neobsahuje aplikaci agregačních funkcí, neobsahuje formule využívající fráze *FOR ALL* a *FOR SOME*, ani neobsahuje proměnné odkazující se na jinou než právě zpracovávanou událost (z toho důvodu je možné zapisovat projekci *EVENT.x* pouze jako *x*). Výstupem aplikace tohoto typu vstupní podmínky je událost, na kterou byla daná podmínka aplikovaná (v případě splnění vstupní podmínky), nebo nedefinovaná hodnota (jinak, například *NULL*).

Tento zápis vstupní podmínky představuje konstrukci události. Pro jednoznačnost zápisu je nutné použít funkci přejmenování (*AS*).

Druhou možností zápisu vstupní podmínky je zápis podmínky vztahující se ke všem událostem sledu. Tuto podmínku lze zapsat

frází *„konstrukce sledu“ WHERE* φ (viz dotaz 4.2, řádek 7), kde φ představuje zápis logické formule, která neobsahuje aplikaci agregačních funkcí (s výjimkou funkce *COUNT*). Další omezení pro tuto formuli neplatí. Je nutné využívat celý zápis projekcí hodnot popisujících danou událost, tj. včetně identifikace události.

Lze konstatovat, že podmínka zapsaná pomocí fráze *FILTERED BY* je zapsatelná také pomocí fráze *WHERE*. Použití fráze *FILTERED BY* bylo zvoleno pro lepší čitelnost výsledného dotazu a snazší zápis některých podmínek (srovnej 4.10 a G.2).

Dotaz 4.2 Příklad zápisů vstupních podmínek

```
1: ...
2:     EVENT
3:         FILTERED BY
4:             '192.168.3.12' IN sources
5:     AS my_events
6: WHERE
7:     SAME hour(timestamp) FOR ALL EVENT
```

Výstupní podmínky Výstupní podmínky představují omezení platná pro celý sled, slouží k určení, jestli je posloupnost splňujících událostí sledu zároveň i definovaným sledem. V jazyce DAQL lze zapsat výstupní podmínku frází *„konstrukce sledu“ HAVING* φ (viz dotaz 4.3, řádek 3), kde φ představuje zápis logické formule, která neobsahuje formule využívající fráze *FOR ALL* a *FOR SOME*.

Dotaz 4.3 Příklad zápisu výstupní podmínky

```
1: ...
2: HAVING
3:     AVG(EVENTS.latency) > 1000
```

Rozšíření jazyka Vstupní podmínky sledů lze využít k dalšímu rozšíření jazyka DAQL. Tímto rozšířením jsou makro příkazy využívající vstupní podmínku vztahující se pouze k právě zpracovávané události, takzvané *pojmenované filtry* V případě rozšíření pro prostředí bezpečnosti počítačových sítí jsou vytvořeny příkazy odpoví-

dající kódům detekčních metod. Zápis makra pro každý takový kód *X* vypadá následovně:

```
X: EVENT FILTERED BY code = 'X'.
```

Při překladu jazyka však nejsou takto definované symboly pouze nahrazeny svou definicí. Ta je automaticky rozšířena, aby bylo možné takto definované proměnné používat stejným způsobem jako proměnnou *EVENT* (ekvivalent proměnné *EVENTS* se automaticky nevytváří, je nutné využít zápis odpovídající zápisu *EVENT(1,*)*). Tyto makropříkazy tak odpovídají použití identifikátorů přiřazených událostem pomocí funkce pojmenování.

Je třeba upozornit, že vstupní podmínka na řádcích 3-5 dotazu 4.4 nemusí být nutně splněna stejnou událostí, jako podmínka na řádku 7.

První podmínka je splněna událostí, která je právě pátá v posloupnosti splňujících událostí, její kód je roven *X* a hodnota jejího atributu *type* je rovna řetězci *unexpected*.

Oproti tomu druhá podmínka je splněná událostí s kódem *X* a hodnotou atributu *type* rovnou řetězci *unexpected*, kterou v posloupnosti splňujících událostí předchází právě čtyři události, jejichž kód je také roven *X*.

Dotaz 4.4 Odlišnosti při použití proměnné definované jako makro

```
1: ...
2: WHERE
3:     EVENT(5).code = 'X'
4:     AND
5:     EVENT(5).type = 'unexpected'
6:     AND
7:     X(5).type = 'unexpected'
```

Konstrukce množiny

Konstrukce množiny definuje n-tici událostí nezávisle (pokud není v rámci podmínek řečeno jinak) na jejich pořadí. Pro definici podmínky ukončení jsou v jazyce DAQL použity příkazy *WHILE* a *UNTIL* následované logickým výrazem. V případě použití podmiňovacího příkazu *WHILE* je daný sled ukončen a předán k dalšímu zpracování v okamžiku, kdy daný logický výraz přestane platit. Naopak,

v případě použití příkazu *UNTIL*, je daný sled ukončen a předán k následnému zpracování v okamžiku, ve kterém je logický výraz poprvé pravdivý.

Dílčí události množiny konstruované příslušnými vstupními podmínkami lze oddělit bílým znakem (nejlépe novým řádkem). Události splňující jednotlivé vstupní podmínky je nutné přejmenovat (s využitím klíčového slova *AS*) pro případné odkazování v dotazu a pro jednoznačnost dotazu.

V rámci logického výrazu podmínky ukončení je možné použít hodnoty popisující všechny dosud zpracované události splňující vstupní podmínku (vzhledem k případné paměťové složitosti není možné uchovávat v paměti i události, které vstupní podmínku nesplňují). Formule logického výrazu mohou obsahovat funkce o *n* proměnných i agregační funkce aplikované na podposloupnost splňujících událostí. Ve spojení s příkazem *WHILE* je možné použít i formuli uvozenou klíčovými slovy *FOR ALL*, ve spojení s příkazem *UNTIL* lze potom použít formuli uvozenou klíčovými slovy *FOR SOME* (s platnými výjimkami zmíněnými výše, viz 4.2.8).

Vyhodnocení podmínky ukončení probíhá po zpracování události, která splňuje vstupní podmínku. V tomto okamžiku už je zpracovaná událost posledním prvkem posloupnosti splňujících událostí a nelze se na ni odkazovat použitím proměnné *EVENT*, ta je v tomto případě nedefinovaná[6].

Pokud definice množiny obsahuje více dílčích událostí s odlišnými vstupními podmínkami (pojmenované například jako *udalost1*, *udalost2* a *udalost3*) a v rámci podmínky ukončení je použita formule obsahující funkci *COUNT(*)*, není zaručeno, že každá z dílčích událostí bude alespoň jednou zahrnuta ve splňující posloupnosti. Máme-li dotaz 4.5, je úspěšným splněním podmínky ukončení i výskyt pěti událostí s hodnotou atributu *code* rovnou řetězci *X*. Z tohoto důvodu je nutné použít funkci *COUNT* s argumentem odpovídajícím identifikátoru, kterým byla daná dílčí událost pojmenována (viz G.1).

Příklad 4.3. Uvažujme rozšíření pro zpracování událostí vyšších řádů v prostředí síťové bezpečnosti, ve kterém jsou definovány datové

6. Výjimku představuje použití fráze *FOR ALL EVENT*, kdy tento zápis odpovídá zápisu *FOR ALL EVENT(1,*)*.

Dotaz 4.5 Použití funkce *COUNT(*)* v rámci podmínky ukončení

```
 1: SET WHILE
 2:     COUNT(*) = 5
 3:     EVENT
 4:         FILTERED BY
 5:             code = 'X'
 6:     AS udalost1
 7:     EVENT
 8:         FILTERED BY
 9:             code = 'Y'
10:     AS udalost2
11:     EVENT
12:         FILTERED BY
13:             code = 'Z'
14:     AS udalost3
```

typy „množství dat“, „časová známka“ a datový typ „IP adresa“, který je použitý pro identifikaci původců a cílů událostí. Datový typ „množství dat“ umožňuje zpracovávat a porovnávat čísla doplněná o jednotku určující objem přenesených dat, datový typ „časová známka“ poskytuje funkci *TIMEDIFF*, která spočítá počet sekund mezi dvěmi zadanými časovými známkami (vždy vrací nezápornou hodnotu). Datový typ „IP adresa“ poskytuje funkci *LIST* (*DISTINCT*) umožňující vytvoření seznamu (seznamu bez opakujících se prvků) zápisů IP adres.

Jak bylo zmíněno výše (viz 4.2.7), každá událost daného rozšíření má atribut *code*. V případě události informující o nadměrném množství přenesených dat je hodnota atributu *code* rovna řetězci „HIGHTRANSF“. Události s daným kódem dále mají atributy *Uploaded* a *Downloaded*, které obsahují množství dat odeslaných a stažených zařízením, jehož IP adresa je uvedena jako zdroj události. Vzhledem k architektuře aplikace ADS (viz 3.3) poskytuje událost informaci pouze o datech přenesených během pěti minut a pro odhalení uživatelů, kteří dlouhodobě stahují nebo odesílají velká množství dat, je třeba kontrolovat opakující se výskyt této události.

Pro kontrolu, jestli některá ze stanic během poslední hodiny odeslala více než 1 GiB dat nebo přijala více než 4 GiB dat, lze použít dotaz 4.6. Dotaz vytváří událost s kódem *BIGDATA*, která v zobrazovaném detailu informuje o celkovém množství přenesených dat.

Nově vytvářená událost je založena na množině událostí s kódem *HIGHTRANSF* (řádek 17), které mají stejného původce a byly vygenerovány v rámci jedné hodiny (výstupní podmínka, řádek 21). Událost je vytvořena v okamžiku, kdy celkový objem odeslaných dat přesáhne 1 GiB nebo celkový objem přijatých dat přesáhne 4 GiB (podmínka ukončení, viz řádek 13).

Původcem události *BIGDATA* je původce všech relevantních událostí *HIGHTRANSF*. Cíle události jsou všechny cíle relevantních událostí. Jako časová známka události je použita časová známka první relevantní události.

Dotaz 4.6 Příklad zápisu konstrukce množiny

```
 1: SELECT
 2:     MIN(timestamp) AS timestamp,
 3:     ht(1).sources AS sources,
 4:     DISTINCT(targets) AS targets,
 5:     'BIGDATA' AS code,
 6:     'Total uploaded: %totalU, total downloaded: %totalD' AS template,
 7:     SUM(Uploaded) AS totalU,
 8:     SUM(Downloaded) AS totalD
 9: FROM
10:     SET WHILE
11:         (SUM(EVENTS.Uploaded) < '1 GiB'
12:         AND
13:         SUM(EVENTS.Downloaded) < '4 GiB')
14:         EVENT
15:             FILTERED BY
16:                 code = 'HIGHTRANSF'
17:         AS ht
18:     WHERE
19:         SAME sources FOR ALL EVENTS
20:     HAVING
21:         TIMEDIFF(ht(1).timestamp, ht(*).timestamp) < 3600
```

Modifikace konstrukce množiny

Uvažujeme-li každý započatý výpočet dotazu jako jeho instanci, je nová instance dotazu vytvářena vždy, když některá událost ve sledovaném datovém proudu splní vstupní podmínku definovanou pro událost, která může být prvním prvkem splňující posloupnosti. Chce-

me-li tak pomocí dotazu jazyka DAQL sledovat změny například v množství přenesených dat v kontextu posledního dne a je-li granularita použitých dat 5 minut, může pro jednoho sledovaného původce existovat až 288 instancí.

Pro takovéto případy jsou v jazyce DAQL zahrnuty modifikace konstrukce množiny a to konstrukce „iterativní množiny“ a konstrukce „klouzavé množiny“.

Konstrukce iterativní množiny umožňuje definovat dotaz, pro který je vytvořena pro každého původce (každou množinu původců) v daném čase nejvýše jedna jeho instance. Další instance dotazu je tak vytvořena vždy až po ukončení výpočtu předchozí instance (pro dané původce). Při zápisu konstrukce iterativní množiny stačí použít spojení klíčových slov *ITERATIVE SET*.

Konstrukce klouzavé množiny umožňuje podobně jako konstrukce iterativní množiny definici dotazu, pro který existuje v daném čase a pro dané původce nejvýše jedna instance. Konstrukce klouzavé množiny konstruuje strukturu analogickou takzvanému „klouzavému okénku“ (*sliding window*). V případě, že je splněna podmínka ukončení klouzavé množiny a zároveň není splněna výstupní podmínka, jsou postupně odebírány nejstarší události, dokud nedojde k zneplatnění podmínky ukončení. Konstrukce klouzavé množiny se zapisuje pomocí klíčových slov *SLIDING SET*.

Obzvláště díky konstrukci klouzavé množiny je možné v jazyce DAQL formulovat dotazy umožňující aplikaci jednoduchých algoritmů strojového učení, jako je například průběžné učení prahových hodnot respektujících posun konceptu (*concept drift*) a jejich následná kontrola.

Příklad 4.4. Při sledování sítě může být zajímavou informací také objevení výrazné odchylky ve vybraných statistických hodnotách oproti hodnotám za určitý časový úsek odchylce předcházející. Pokud je například medián přenesených dat z DNS serverů během každých pěti minut za předchozí hodinu třetinový oproti množství dat přenesených z DNS serverů za posledních pět minut, může to znamenat snahu o zahlcení sítě (za účelem odepření služby) zesílenou DNS servery[7].

Takovýto útok lze detekovat pomocí dotazu 4.7. Tento dotaz vy-

7. *DNS amplified attack* je jedním z útoků typu *Denial of Service*. Tento útok

užívá konstrukci iterativní množiny pro vytváření pětiminutových dávek z důvodů diskretizace času. Pětiminutový interval byl zvolen podle intervalu cyklického zpracování dat aplikací ADS. Konstrukce klouzavého okénka je v dotazu využita pro průběžnou kontrolu množství přenesených dat z DNS serverů na jednotlivé stanice monitorované sítě. Při překročení prahové hodnoty, která je v dotazu definována jako trojnásobek mediánu objemů dat každé z předchozích jedenácti dávek, je vygenerována událost. Události je přiřazen kód *DNSDOS*, časová známka je shodná s časovou známkou dávky, ve které došlo k překročení prahové hodnoty (tato časová známka odpovídá časové známce prvního toku dané dávky). Za původce události jsou označeny DNS servery, které v dané dávce odeslaly na oběť útoku nějaká data. Cílem události je oběť tohoto útoku. Dalším atributem události je množství dat, které bylo odesláno oběti útoku v dané dávce a příslušná prahová hodnota.

Konstrukce posloupnosti

Konstrukce posloupnosti definuje n-tici událostí s přesně daným pořadím. Mezi jednotlivými událostmi lze definovat časové rozestupy. Podmínka ukončení je definovaná pomocí časového rozdílu mezi první a poslední událostí daného sledu. V jazyce DAQL se konstrukce posloupnosti zapisuje pomocí příkazu *SEQUENCE LIMITED BY duration* kde *duration* představuje řetězec tvořený dekadickým zápisem přirozeného čísla a zápisem jednotky času. Takovýto řetezec může být například *8 DAYS* pro posloupnosti trvající nejdéle osm dní, *12 HRS* pro posloupnosti trvající nejdéle dvanáct hodin, *18 MINS* pro posloupnosti trvající nejdéle osmnáct minut a *100 SECS* pro posloupnosti trvající nejdéle sto sekund. Jiné jednotky trvání než *DAYS*, *HRS*, *MINS* a *SECS* (nebo jejich dlouhé alternativy *HOURS*, *MINUTES* a *SECONDS*) nejsou podporovány.

Podmínka ukončení je současně i výstupní podmínkou konstrukce posloupnosti. Ke splnění dojde v okamžiku zpracování události, která splňuje vstupní podmínky poslední dílčí události posloupnosti

---

zneužívá návrh protokolu DNS, který umožňuje formulovat dotazy tak, aby byla příslušná odpověď mnohonásobně větší než daný dotaz. Za současného podvržení zdrojové IP adresy dotazu lze dosáhnout zahlcení oběti s využitím malého množství prostředků útočníka.[25]

Dotaz 4.7 Odhalení DoS útoku zesíleného DNS servery

```
 1: SELECT
 2:     batch(*).timestamp AS timestamp,
 3:     batch(*).sources AS sources,
 4:     batch(*).targets AS targets,
 5:     'DNSDOS' AS code,
 6:     batch(*).SUM(bytes) AS dataSum,
 7:     3 * MED(batch(1,* − 1).SUM(bytes)) AS threshold,
 8:     'Final amount of data: %dataSum, the threshold: %threshold' AS template
 9: FROM
10:     SLIDING SET WHILE
11:         COUNT(*) < 12
12:         ITERATIVE SET WHILE
13:             TIMEDIFF(dns_flow(1).timestamp, dns_flow(*).timestamp) < 300
14:             NETFLOW
15:                 FILTERED BY
16:                     source_port = 53
17:             AS dns_flow
18:         WHERE
19:             SAME sources FOR ALL dns_flow
20:         AS batch
21:     WHERE
22:         SAME sources FOR ALL batch
23:     HAVING
24:         batch(*).SUM(bytes) > (3 * MED(batch(1,* − 1).SUM(bytes)))
```

definované danou konstrukcí. Zároveň musí platit, že rozdíl mezi časovými známkami této události a první události posloupnosti nesmí být větší než je definováno podmínkou ukončení.

Dílčí události posloupnosti mohou být odděleny bílým znakem, v případě, že je potřeba definovat časové rozestupy mezi jednotlivými po sobě jdoucími dílčími událostmi, lze použít oddělovač zapisovaný pomocí řetězce *GAP specifier*, kde *specifier* může mít podobu řetězců *UP TO duration*, *BETWEEN* $duration_1$ *AND* $duration_2$ a nebo *STRICT BETWEEN* $duration_1$ *AND* $duration_2$, kde proměnné *duration*, $duration_1$ a $duration_2$ představují stejný řetězec jako výše. Podmínka *UP TO duration* popisuje časovou mezeru, jejíž trvání je nejvýše *duration*. Podmínka *BETWEEN* $duration_1$ *AND* $duration_2$ definuje časovou mezeru, která trvá alespoň dobu definovanou zápisem $duration_1$ a nejvýše dobu zapsanou jako $duration_2$. Události splňující vstupní podmínku následující dílčí události posloupnosti

jsou ignorovány, jestliže se vyskytnou dříve, než může časová mezera skončit. Podmínka *STRICT BETWEEN* $duration_1$ *AND* $duration_2$ je shodná s předchozí podmínkou, pouze v případě výskytu události splňující vstupní podmínku následující dílčí události posloupnosti dříve, než může být časová mezera ukončena, dojde k ukončení výpočtu dotazu, aniž by byla vytvořena událost. Časovou mezeru omezenou touto podmínkou tak lze současně považovat za vstupní i výstupní podmínku.

Posloupnost splňujících událostí konstruovaná danou konstrukcí vždy obsahuje právě jednu dílčí událost vyhovující příslušným vstupním podmínkám. Další události vyhovující dané vstupní podmínce jsou danou instancí dotazu ignorované.

Příklad 4.5. Uvažujme šíření viru Morto, tak jak bylo popsáno výše (viz 3). Dotaz odhalující jeho úspěšné šíření monitorovanou sítí kontroluje výskyt posloupnosti skenování stanice na portu 3389, použití neobvyklého DNS serveru a skenování dalších zařízení na portu 3389 napadenou stanicí. Lze očekávat, že celá tato posloupnost proběhne během osmi hodin, a že mezi prvním skenováním a začátkem komunikace maskované za DNS dotazy bude nejvíce šest hodin. Tyto hodnoty mohou být i vyšší, ovšem při návrhu dotazů je nutné brát v ohled také možnou výpočetní a paměťovou náročnost (při každém skenování portu číslo 3389 je vytvořena nová instance dotazu, která je v systému uložena po celou dobu, během které je reálné jeho splnění).

Při návrhu dotazu pro odhalení šíření viru Morto je vhodné uvažovat více případů komunikace maskované za DNS dotazy, dále je nutné najít vztah mezi jednotlivými dílčími událostmi. Ten je zde představován úspěšně napadenou stanicí, která je cílem prvního skenování a původcem následujících událostí.

Samotný dotaz pro odhalení šíření viru Morto může vypadat například jako dotaz 4.10.

Přepis konstrukce posloupnosti pomocí konstrukce množiny

Pro přepis konstrukce posloupnosti na konstrukci množiny je nutné uvědomit si jejich vzájemné rozdíly. Rozdíly jsou uváděny vždy z pohledu konstrukce posloupnosti. Prvním z nich je nutnost alespoň jed-

noho výskytu každé dílčí události, přičemž pouze první z nich je zahrnuta do posloupnosti splňujících událostí, ostatní jsou ignorovány. Tato vlastnost musí být postihnuta ve vstupní podmínce, podmínce ukončení.

Druhým rozdílem je existence časových mezer použitých jako oddělovače. Časové mezery omezené podmínkami *UP TO duration* a *BETWEEN* $duration_1$ *AND* $duration_2$ lze přepsat pomocí vstupní podmínky, časovou mezeru omezenou pomocí podmínky *STRICT BETWEEN* $duration_1$ *AND* $duration_2$ je nutné přepsat současně do vstupních i výstupních podmínek.

Samotné omezení *LIMITED BY duration* představuje pouze jiný způsob zápisu podmínky ukončení. Je však vhodnější toto omezení přepsat mezi výstupní podmínky konstrukce množiny. Při přepisu je nutné převést časové jednotky na sekundy.

Pomocí zde popsaných pravidel tak lze přepsat konstrukci posloupnosti 4.8 na konstrukci množiny 4.9.

Dotaz 4.8 Konstrukce posloupnosti

```
1: SEQUENCE LIMITED BY 6 DAYS
2:     EVENT
3:     AS event1
4:     GAP UP TO 2 HOURS
5:     EVENT
6:     AS event2
7:     GAP BETWEEN 3 HOURS AND 4 HOURS
8:     EVENT
9:     AS event3
10:    GAP STRICT BETWEEN 5 HOURS AND 6 HOURS
11:    EVENT
12:    AS event4
```

Dotaz 4.9 Posloupnost 4.8 zapsaná pomocí konstrukce množiny

```
1: SET WHILE
2:     COUNT(event1) = 1
3:     AND
4:     COUNT(event2) = 1
5:     AND
6:     COUNT(event3) = 1
7:     AND
8:     COUNT(event4) = 1
9:     EVENT
10:    AS event1
11:    EVENT
12:    AS event2
13:    EVENT
14:    AS event3
15:    EVENT
16:    AS event4
17: WHERE
18:    COUNT(event1) = 1
19:    AND
20:    COUNT(event2) = 1
21:    AND
22:    COUNT(event3) = 1
23:    AND
24:    COUNT(event4) = 1
25:    AND
26:    event1.timestamp < event2.timestamp
27:    AND
28:    event2.timestamp < event3.timestamp
29:    AND
30:    event3.timestamp < event4.timestamp
31:    AND
32:    TIMEDIFF(event1.timestamp, event2.timestamp) <= 7200
33:    AND
34:    TIMEDIFF(event2.timestamp, event3.timestamp) BETWEEN 10800 AND
   14400
35:    AND
36:    TIMEDIFF(event3.timestamp, event4.timestamp) <= 21600
37: HAVING
38:    TIMEDIFF(evnt3, event4) >= 18000
39:    AND
40:    TIMEDIFF(event1.timestamp,event2.timestamp) <= (6 * 86400)
```

Dotaz 4.10 Zjištění šíření viru Morto ve sledované síti

```
 1: SELECT
 2:     MIN(timestamp) AS timestamp,
 3:     first_scan.sources AS sources,
 4:     dns_set.sources AS targets,
 5:     'MORTO' AS code,
 6:     'The target device was infected by MORTO malware.' AS template
 7: FROM
 8:     SEQUENCE LIMITED BY 8 HOURS
 9:         SCANS
10:             FILTERED BY
11:                 port = 3389
12:         AS first_scan
13:         GAP UP TO 6 HOURS
14:         SET UNTIL
15:             (COUNT(EVENTS) > 12
16:             AND
17:             COUNT(DISTINCT(targets)) > 5)
18:             DNSANOMALY
19:                 FILTERED BY
20:                     type = 'unexpected'
21:                     AND
22:                     COUNT(sources) = 1
23:             AS anomaly
24:         WHERE
25:             SAME sources FOR ALL EVENTS
26:         AS dns_set
27:         GAP UP TO 30 MINUTES
28:         SCANS
29:             FILTERED BY
30:                 port = 3389
31:         AS second_scan
32:     WHERE
33:         dns_set.sources IN first_scan.targets
34:         AND
35:         dns_set.sources = second_scan.sources
```

5 Formální definice jazyka

Před formální definicí slovníku jazyka, jeho syntaxe a sémantiky je nutné řádně definovat některé dříve uvedené pojmy.

Definice 5.1. Mějme množinu identifikátorů původců událostí (A), množinu identifikátorů cílů událostí (P), uspořádanou množinu časových známek (T, relace $\leq$), množinu identifikátorů atributů (At) a množinu množin možných hodnot (datových typů, Dt).

Definujme funkci $f : At \longrightarrow Dt$. Tato funkce přiřadí každému identifikátoru atributu množinu možných hodnot, které může daný atribut nabývat (datový typ).

Funkce $\mathcal{P}$ přiřadí každé množině její potenční množinu.

Definujme množinu $\mathcal{P}^*(At \times Dt)$ takto:

$$A \in \mathcal{P}(\{(at, x) | at \in At \wedge x \in f(at)\})$$

$$A \in \mathcal{P}^*(At \times Dt) \iff \forall (at_1, x_1), (at_2, x_2) \in A : at_1 \neq at_2 \vee x_1 = x_2$$

Množina všech událostí (Ev) je potom definována takto:

$$Ev = (\mathcal{P}(A) - \emptyset) \times T \times \mathcal{P}(P) \times \mathcal{P}^*(At \times Dt)$$

Množina všech událostí je uspořádatelná podle množiny T a relace $\leq$.

Definice 5.2. Nejprve definujme binární operaci $\cup^* : \mathcal{P}^*(At \times Dt) \times \mathcal{P}^*(At \times Dt) \longrightarrow \mathcal{P}^*(At \times Dt)$. Tato operace odpovídá sjednocení množin X a Y, které splňují následující podmínku:

$$(at, x) \in X \iff \nexists y : (at, y) \in Y$$

Pokud tato podmínka není splněna, musí nejprve dojít k přejmenování atributů v obou množinách tak, aby byla podmínka splněna[1], pak až dojde k jejich sjednocení

Nyní definujme binární operaci $\circ_{\min} : Ev \times Ev \longrightarrow Ev$ takto:

$$A_1, A_2 \in (\mathcal{P}(A) - \emptyset), P_1, P_2 \in \mathcal{P}(P), t_1, t_2 \in T, AV_1, AV_2 \in \mathcal{P}^*(At \times Dt)$$

1. Bude implementováno jako indexace atributů.

$$ev_1 = (A_1, t_1, P_1, AV_1)$$
$$ev_2 = (A_2, t_2, P_2, AV_2)$$
$$ev_1 \circ_{\min} ev_2 = (A_1 \cup A_2, \min\{t_1, t_2\}, P_1 \cup P_2, AV_1 \cup^* AV_2)$$

Vzhledem k možné nutnosti přejmenování atributů v rámci operace $\cup^*$ není operace $\circ_{\min}$ obecně komutativní ani asociativní. Analogicky lze definovat také operaci $\circ_{\max}$.

Definice 5.3. Mějme posloupnost událostí $ev_1, ev_2, \ldots, ev_n$ seřazenou podle času vzniku. Událost $ev = ((\ldots(ev_1 \circ_{\min} ev_2)\ldots) \circ_{\min} ev_n)$ se nazývá *minimální sled událostí* (zkráceně pouze sled).

Analogicky je možné definovat *maximální sled událostí*, pokud však nebude řečeno jinak, sledem je vždy myšlen *minimální sled událostí*.

Důkaz věty o sledu událostí (viz 4.1) Pravdivost věty plyne z formální definice sledu událostí.

5.1 Slovník jazyka DAQL

Prvky jazyka DAQL lze podle jejich vzniku a použití rozdělit do pěti skupin. První skupinou jsou klíčová slova a klíčové fráze jazyka DAQL, která jsou dána definicí tohoto dotazovacího jazyka. Tato slova jsou v abecedním pořadí vypsána v příloze A. Za součást této skupiny lze dále považovat symboly „(“, „)“, „,“, „.“, znak „=“, který je v jazyku použit pro označení relace ekvivalence a znak „‘“ (apostrof), který je použit pro uvozování nečíselných konstant (viz níže).

Druhá skupina prvků jazyka je tvořena všemi řetězci, které je možné použít jako identifikátory (viz 4.2.5). Tyto řetězce se zapisují bez použití uvozovek.

Třetí skupinou prvků jsou konstanty dané definicí jazyka DAQL. Mezi tyto konstanty patří dekadický zápis nezáporného čísla (použitý například jako index nebo při trvání posloupnosti událostí, případně časové mezery), symbol „*“ a zápis „* − #num“, kde #num je dekadickým zápisem přirozeného čísla.

Zbývající dvě skupiny prvků jazyka DAQL nejsou dané definicí tohoto jazyka, ale definicí jeho rozšíření.

První takto definovanou skupinou jsou klíčová slova definovaná v rámci rozšíření. Do této skupiny patří identifikátory povinných atributů rozšíření, makro příkazy použité jako pojmenované filtry a slova (případně symboly) označující funkce a relace. Tato skupina může mít s první skupinou neprázdný průnik, který je dán formální gramatikou jazyka DAQL (gramatika musí zůstat LL(1)). Příkladem takového překryvu jsou například klíčová slova *BETWEEN* a *AND*, která lze společně použít pro zápis ternární relace.

Poslední skupinou jsou zápisy konstant daných jednotlivými datovými typy. Tyto konstanty je nutné zapisovat do jednoduchých uvozovek (apostrofů). Výjimkou jsou zápisy celých čísel využívající arabské číslice a symbol „–“ k rozlišení záporných čísel a zápisy racionálních čísel využívající arabské číslice a symbol „.“ k oddělení desetinných míst.

5.2 Formální syntaxe dotazovacího jazyka DAQL

Syntaxi jazyka DAQL lze zapsat pomocí LL(1) gramatiky (důkazem, že je gramatika LL(1), budiž existence jednoznačné rozkladové tabulky pro jednotlivé terminální symboly, viz 5.2.1), která je rozepsána níže.

Neterminály této gramatiky jsou uzavřeny lomenými závorkami ($\langle\rangle$), znak ε je použit pro prázdné slovo, symboly uvozené symbolem „#“ slouží k identifikaci řetězců rozpoznaných a označených při lexikální analýze. Ostatní symboly jsou terminály gramatiky. Mezi dvěmi symboly s alfanumerickým zápisem je nutné zapisovat libovolný nenulový počet bílých znaků (mezera, tabulátor, nový řádek). Mezi zápisem funkce (případně relace) zapisované prefixově a symbolem levé závorky nesmí být žádná mezera. Mezi ostatními dvojicemi symbolů může být libovolný počet bílých znaků.

Gramatika pro jazyk DAQL včetně rozšíření může být vygenerována na základě níže popsaných gramatických pravidel. Pro konstanty definované datovými typy se automaticky přidají pravidla $\langle CONSTANT\rangle \longrightarrow \#datovy\_typ$, pro funkce zapisované prefixově se přidají pravidla $\langle FUNCTOR\rangle \longrightarrow \#zapis\_funkce$, dále obdobně

pro funkce zapisované infixově ($\langle INFFUNCTOR \rangle$), relace zapisované prefixově ($\langle PRREL \rangle$), binární relace, které jsou zapisované infixově ($\langle BINARY \rangle$), ternární relace, které jsou zapisované infixově ($\langle TERNARY1 \rangle$ a $\langle TERNARY2 \rangle$) a makro příkazy použité jako pojmenované filtry ($\langle EVENT \rangle \longrightarrow \#makro \langle FILTER \rangle$).

Gramatika níže popisuje pro větší názornost jazyk DAQL s rozšířením, které obsahuje datový typ #num, infixově zapisovanou funkci sčítání značenou symbolem „+“, relaci „match“ zapisovanou prefixově a infixově zapisovanou ternární relaci „BETWEEN ... AND ...“. Symbol „*“ může být pro potřeby této gramatiky považován za prvek datového typu #num.

V pravidlech 5.46-5.64 lze vidět omezení použití závorek, které zajistí jednoznačnost definovaného dotazu.

$$\langle S' \rangle \longrightarrow \langle S \rangle \tag{5.1}$$
$$\langle S \rangle \longrightarrow SELECT \langle PROJECTION \rangle FROM \langle STREAM \rangle \tag{5.2}$$
$$\langle PROJECTION \rangle \longrightarrow \langle APPLICATION \rangle AS\ \#idstring \langle NEXTPROJ \rangle \tag{5.3}$$
$$\langle NEXTPROJ \rangle \longrightarrow , \langle APPLICATION \rangle AS\ \#idstring \langle NEXTPROJ \rangle \tag{5.4}$$
$$\langle NEXTPROJ \rangle \longrightarrow \varepsilon \tag{5.5}$$
$$\langle APPLICATION \rangle \longrightarrow \langle FUNCTOR \rangle (\langle APPLICATION \rangle \langle NEXT \rangle) \langle APPLINF \rangle \tag{5.6}$$
$$\langle APPLICATION \rangle \longrightarrow \langle COMPLEXID \rangle \langle APPLINF \rangle \tag{5.7}$$
$$\langle APPLICATION \rangle \longrightarrow (\langle APPLICATION \rangle) \langle APPLINF \rangle \tag{5.8}$$
$$\langle APPLICATION \rangle \longrightarrow \langle CONSTANT \rangle \langle APPLINF \rangle \tag{5.9}$$
$$\langle APPLICATION' \rangle \longrightarrow \langle FUNCTOR \rangle (\langle APPLICATION \rangle \langle NEXT \rangle) \langle APPLINF \rangle \tag{5.10}$$
$$\langle APPLICATION' \rangle \longrightarrow \langle COMPLEXID \rangle \langle APPLINF \rangle \tag{5.11}$$
$$\langle APPLICATION' \rangle \longrightarrow \langle CONSTANT \rangle \langle APPLINF \rangle \tag{5.12}$$
$$\langle APPLINF \rangle \longrightarrow \langle INFFUNCTOR \rangle \langle APPLICATION \rangle \tag{5.13}$$
$$\langle APPLINF \rangle \longrightarrow \varepsilon \tag{5.14}$$
$$\langle NEXT \rangle \longrightarrow , \langle APPLICATION \rangle \langle NEXT \rangle \tag{5.15}$$
$$\langle NEXT \rangle \longrightarrow \varepsilon \tag{5.16}$$
$$\langle COMPLEXID \rangle \longrightarrow \#idstring \langle INDEXED \rangle \langle NEXTCOMP \rangle \tag{5.17}$$
$$\langle NEXTCOMP \rangle \longrightarrow . \langle APPLY \rangle \tag{5.18}$$
$$\langle NEXTCOMP \rangle \longrightarrow \varepsilon \tag{5.19}$$
$$\langle APPLY \rangle \longrightarrow \langle FUNCTOR \rangle (\langle SIMPLEID \rangle) \tag{5.20}$$
$$\langle APPLY \rangle \longrightarrow \#idstring \langle INDEXED \rangle \langle NEXTCOMP \rangle \tag{5.21}$$
$$\langle SIMPLEID \rangle \longrightarrow \#idstring \langle INDEXED \rangle \langle NEXTID \rangle \tag{5.22}$$
$$\langle NEXTID \rangle \longrightarrow .\#idstring \langle INDEXED \rangle \langle NEXTID \rangle \tag{5.23}$$
$$\langle NEXTID \rangle \longrightarrow \varepsilon \tag{5.24}$$
$$\langle INDEXED \rangle \longrightarrow (\langle INDEX \rangle) \tag{5.25}$$
$$\langle INDEXED \rangle \longrightarrow \varepsilon \tag{5.26}$$
$$\langle INDEX \rangle \longrightarrow \#num \langle LINDEX \rangle \tag{5.27}$$

$$\langle INDEX\rangle \longrightarrow *-\#num\langle LINDEX\rangle \tag{5.28}$$
$$\langle INDEX\rangle \longrightarrow * \tag{5.29}$$
$$\langle LINDEX\rangle \longrightarrow \varepsilon \tag{5.30}$$
$$\langle LINDEX\rangle \longrightarrow ,\langle LINDEX'\rangle \tag{5.31}$$
$$\langle LINDEX'\rangle \longrightarrow \#num \tag{5.32}$$
$$\langle LINDEX'\rangle \longrightarrow *-\#num \tag{5.33}$$
$$\langle LINDEX'\rangle \longrightarrow * \tag{5.34}$$
$$\langle CONSTANT\rangle \longrightarrow \#num \tag{5.35}$$
$$\langle INFFUNCTOR\rangle \longrightarrow + \tag{5.36}$$
$$\langle FUNCTOR\rangle \longrightarrow COUNT \tag{5.37}$$
$$\langle STREAM\rangle \longrightarrow \langle STREAMX\rangle\langle WHERE\rangle\langle HAVING\rangle \tag{5.38}$$
$$\langle STREAMX\rangle \longrightarrow ITERATIVE\langle SET\rangle \tag{5.39}$$
$$\langle STREAMX\rangle \longrightarrow SLIDING\langle SET\rangle \tag{5.40}$$
$$\langle STREAMX\rangle \longrightarrow \langle SET\rangle \tag{5.41}$$
$$\langle STREAMX\rangle \longrightarrow SEQUENCE\ LIMITED\ BY\ \#num\langle TSPEC\rangle\langle SEQBODY\rangle \tag{5.42}$$
$$\langle SET\rangle \longrightarrow SET\langle SETCONSTR\rangle\langle SETBODY\rangle \tag{5.43}$$
$$\langle SETCONSTR\rangle \longrightarrow WHILE\langle CONSTR\rangle \tag{5.44}$$
$$\langle SETCONSTR\rangle \longrightarrow UNTIL\langle CONSTR\rangle \tag{5.45}$$
$$\langle CONSTR\rangle \longrightarrow NOT\langle CONSTR\rangle \tag{5.46}$$
$$\langle CONSTR\rangle \longrightarrow (\langle AFTERBR\rangle\langle ANDOR\rangle \tag{5.47}$$
$$\langle CONSTR\rangle \longrightarrow FOR\ ALL\ \#idstring\langle INDEXED\rangle\langle CONSTR\rangle \tag{5.48}$$
$$\langle CONSTR\rangle \longrightarrow FOR\ SOME\ \#idstring\langle INDEXED\rangle\langle CONSTR\rangle \tag{5.49}$$
$$\langle CONSTR\rangle \longrightarrow SAME\ sources\ FOR\ ALL\ \#idstring\langle INDEXED\rangle\langle ANDOR\rangle \tag{5.50}$$
$$\langle CONSTR\rangle \longrightarrow SAME\ targets\ FOR\ ALL\ \#idstring\langle INDEXED\rangle\langle ANDOR\rangle \tag{5.51}$$
$$\langle CONSTR\rangle \longrightarrow SAME\langle TAPPL\rangle FOR\ ALL\ \#idstring\langle INDEXED\rangle\langle ANDOR\rangle \tag{5.52}$$
$$\langle CONSTR\rangle \longrightarrow \langle RELATION\rangle\langle ANDOR\rangle \tag{5.53}$$
$$\langle AFTERBR\rangle \longrightarrow NOT\langle CONSTR\rangle) \tag{5.54}$$
$$\langle AFTERBR\rangle \longrightarrow (\langle AFTERBR\rangle\langle ANDOR\rangle) \tag{5.55}$$
$$\langle AFTERBR\rangle \longrightarrow FOR\ ALL\ \#idstring\langle INDEXED\rangle\langle CONSTR\rangle) \tag{5.56}$$
$$\langle AFTERBR\rangle \longrightarrow FOR\ SOME\ \#idstring\langle INDEXED\rangle\langle CONSTR\rangle) \tag{5.57}$$
$$\langle AFTERBR\rangle \longrightarrow SAME\ sources\ FOR\ ALL\ \#idstring\langle INDEXED\rangle\langle ANDOR\rangle) \tag{5.58}$$
$$\langle AFTERBR\rangle \longrightarrow SAME\ targets\ FOR\ ALL\ \#idstring\langle INDEXED\rangle\langle ANDOR\rangle) \tag{5.59}$$
$$\langle AFTERBR\rangle \longrightarrow SAME\langle TAPPL\rangle FOR\ ALL\ \#idstring\langle INDEXED\rangle\langle ANDOR\rangle) \tag{5.60}$$
$$\langle AFTERBR\rangle \longrightarrow \langle PRREL\rangle(\langle APPLICATION\rangle\langle NEXT\rangle)\langle ANDOR\rangle) \tag{5.61}$$
$$\langle AFTERBR\rangle \longrightarrow \langle APPLICATION'\rangle\langle BRORNOT\rangle\langle ANDOR\rangle \tag{5.62}$$
$$\langle BRORNOT\rangle \longrightarrow)\langle INFREL\rangle \tag{5.63}$$
$$\langle BRORNOT\rangle \longrightarrow \langle INFREL\rangle\langle ANDOR\rangle) \tag{5.64}$$
$$\langle ANDOR\rangle \longrightarrow AND\langle CONSTR\rangle \tag{5.65}$$
$$\langle ANDOR\rangle \longrightarrow OR\langle CONSTR\rangle \tag{5.66}$$
$$\langle ANDOR\rangle \longrightarrow \varepsilon \tag{5.67}$$
$$\langle TAPPL\rangle \longrightarrow timestamp\langle TAPPL'\rangle \tag{5.68}$$
$$\langle TAPPL\rangle \longrightarrow (\langle TAPPL\rangle)\langle TAPPL'\rangle \tag{5.69}$$
$$\langle TAPPL\rangle \longrightarrow \langle FUNCTOR\rangle(\langle TAPPL\rangle)\langle TAPPL'\rangle \tag{5.70}$$

$$\langle TAPPL' \rangle \longrightarrow \varepsilon \tag{5.71}$$
$$\langle TAPPL' \rangle \longrightarrow \langle INFFUNCTOR \rangle \langle TAPPL \rangle \tag{5.72}$$
$$\langle RELATION \rangle \longrightarrow \langle PRREL \rangle (\langle APPLICATION \rangle \langle NEXT \rangle) \tag{5.73}$$
$$\langle RELATION \rangle \longrightarrow \langle APPLICATION' \rangle \langle INFREL \rangle \tag{5.74}$$
$$\langle PRREL \rangle \longrightarrow match \tag{5.75}$$
$$\langle INFREL \rangle \longrightarrow \langle BINARY \rangle \langle APPLICATION \rangle \tag{5.76}$$
$$\langle INFREL \rangle \longrightarrow \langle TERNARY1 \rangle \langle APPLICATION \rangle \langle TERNARY2 \rangle \langle APPLICATION \rangle \tag{5.77}$$
$$\langle BINARY \rangle \longrightarrow = \tag{5.78}$$
$$\langle TERNARY1 \rangle \longrightarrow BETWEEN \tag{5.79}$$
$$\langle TERNARY2 \rangle \longrightarrow AND \tag{5.80}$$
$$\langle WHERE \rangle \longrightarrow WHERE \langle CONSTR \rangle \tag{5.81}$$
$$\langle WHERE \rangle \longrightarrow \varepsilon \tag{5.82}$$
$$\langle HAVING \rangle \longrightarrow HAVING \langle CONSTR \rangle \tag{5.83}$$
$$\langle HAVING \rangle \longrightarrow \varepsilon \tag{5.84}$$
$$\langle SETBODY \rangle \longrightarrow \langle ASEVENT \rangle \langle NEXTSETB \rangle \tag{5.85}$$
$$\langle NEXTSETB \rangle \longrightarrow \langle ASEVENT \rangle \langle NEXTSETB \rangle \tag{5.86}$$
$$\langle NEXTSETB \rangle \longrightarrow \varepsilon \tag{5.87}$$
$$\langle ASEVENT \rangle \longrightarrow \langle EVENT \rangle AS \#idstring \tag{5.88}$$
$$\langle EVENT \rangle \longrightarrow \langle S \rangle \tag{5.89}$$
$$\langle EVENT \rangle \longrightarrow \langle STREAM \rangle \tag{5.90}$$
$$\langle EVENT \rangle \longrightarrow EVENT \langle FILTER \rangle \tag{5.91}$$
$$\langle FILTER \rangle \longrightarrow FILTERED\ BY \langle CONSTR \rangle \tag{5.92}$$
$$\langle FILTER \rangle \longrightarrow \varepsilon \tag{5.93}$$
$$\langle TSPEC \rangle \longrightarrow DAYS \tag{5.94}$$
$$\langle TSPEC \rangle \longrightarrow HRS \tag{5.95}$$
$$\langle TSPEC \rangle \longrightarrow MINS \tag{5.96}$$
$$\langle TSPEC \rangle \longrightarrow SECS \tag{5.97}$$
$$\langle TSPEC \rangle \longrightarrow HOURS \tag{5.98}$$
$$\langle TSPEC \rangle \longrightarrow MINUTES \tag{5.99}$$
$$\langle TSPEC \rangle \longrightarrow SECONDS \tag{5.100}$$
$$\langle SEQBODY \rangle \longrightarrow \langle ASEVENT \rangle \langle NEXTSEQB \rangle \tag{5.101}$$
$$\langle NEXTSEQB \rangle \longrightarrow GAP \langle GAPSPEC \rangle \langle ASEVENT \rangle \langle NEXTSEQB \rangle \tag{5.102}$$
$$\langle NEXTSEQB \rangle \longrightarrow \langle ASEVENT \rangle \langle NEXTSEQB \rangle \tag{5.103}$$
$$\langle NEXTSEQB \rangle \longrightarrow \varepsilon \tag{5.104}$$
$$\langle GAPSPEC \rangle \longrightarrow UP\ TO\ \#num \langle TSPEC \rangle \tag{5.105}$$
$$\langle GAPSPEC \rangle \longrightarrow STRICT\ BETWEEN\ \#num \langle TSPEC \rangle AND\ \#num \langle TSPEC \rangle \tag{5.106}$$
$$\langle GAPSPEC \rangle \longrightarrow BETWEEN\ \#num \langle TSPEC \rangle AND\ \#num \langle TSPEC \rangle \tag{5.107}$$

5.2.1 Syntaktická analýza shora dolů

Pro nalezení příslušného syntaktického stromu pro daný dotaz jazyka DAQL lze použít syntaktickou analýzu shora dolů s využitím rozkladových tabulek. Výsledky aplikace funkcí *FIRST* a *FOLLOW* jsou vypsány v příloze (viz kapitoly B a C, terminální symbol $ je použit k označení konce dotazu). Samotná rozkladová tabulka je popsána v kapitole D přílohy. Do tabulky je přidán neterminální symbol $\langle\#\rangle$ značící dno zásobníku. Řádky tabulky popisují neterminální symboly, které jsou na vrcholu zásobníku automatu provádějícího výpočet, sloupce představují právě čtené terminální symboly. V řádcích tabulky jsou vynechány terminální symboly – při čtení stejného terminálního symbolu jako je na vrcholu zásobníku se použije operace *pop* zásobníkového automatu. Políčka tabulky obsahují číslo pravidla gramatiky, podle kterého se přepíše vrchol zásobníku v případě dané kombinace neterminálního symbolu na vrcholu zásobníku a terminálního symbolu na vstupu automatu. Pokud se při výpočtu automatu objeví kombinace symbolu ze zásobníku a symbolu na vstupu, pro kterou není v tabulce přiřazeno přepisovací pravidlo ani operace *pop*, dojde k selhání syntaktické analýzy, protože dotaz neodpovídá dané gramatice.

5.3 Formální sémantika dotazovacího jazyka DAQL

V této části textu je formálně popsána denotační sémantika funkcí přejmenování[2] a projekce, dále sémantika konstrukcí projekce a množiny včetně modifikací. Vzhledem k možnosti zápisu konstrukce posloupnosti pomocí konstrukce množiny zde není popsána její formální sémantika.

Pro potřeby definice sémantiky je třeba definovat několik pojmů.

Definice 5.4. *Posloupnost splňujících událostí* množiny je taková posloupnost událostí $(e_1, \ldots, e_n)$, pro kterou platí, že pokud $i \leq j$, potom i $e_i \preceq e_j$ (událost e_j nenastala dříve než událost e_j). Pokud je $\mathcal{S}$ posloupnost splňujících událostí a $\mathcal{M}$ množina vstupních podmínek, potom nutně platí $\{\mathcal{S}\} \models \mathcal{M}$.

2. Formálně se jedná o konstrukci. Nezaměňovat s přejmenováním v rámci operace $\cup^*$.

Posloupnost splňujících událostí značíme $\mathcal{S}$.

Definice 5.5. Buď $\mathcal{E} = (e_1, \ldots, e_i)$ posloupnost událostí a b událost, která nenastala dříve než událost e_i. Potom $\mathcal{E} \Cup b = (e_1, \ldots, e_i, e_{i+1})$, kde $e_{i+1} = b$. Operaci $\Cup$ nazvěme rozšířením posloupnosti událostí.

Definice 5.6. Říkáme, že událost e patří do posloupnosti událostí $\mathcal{E} = (e_1, \ldots, e_n)$ právě tehdy, když existuje $i \in \{1, \ldots, n\}$ takové, že $e_i = e$.

Definice 5.7. Mějme množinu zápisů aplikací funkcí $Aexp$. Potom můžeme pro všechny posloupnosti splňujících událostí $\mathcal{S}$ a všechny datové typy D definovat funkci $\mathcal{A} :\to Aexp(\mathcal{S} \to D)$. Obdobně můžeme definovat funkci $\mathcal{B}$ pro množinu všech zápisů podmínek (logických výrazů). Oborem hodnot funkce $\mathcal{B}$ je množina pravdivostních hodnot.

Funkce $\mathcal{C}$ je obdobně definovaná pro množinu všech konstrukcí. Oborem hodnot této funkce je množina posloupností událostí.

5.3.1 Přístup k datům událostí, aplikace funkcí

Mějme konstantu x a její zápis x. Potom pro všechna $\mathcal{S}$ platí:

$$\mathcal{A}[\![\mathrm{x}]\!]\mathcal{S} = x.$$

Dále mějme množinu původců A_1, množinu cílů P_1, časovou známku t_1 a množinu dvojic atribut-hodnota Atr_1. Potom platí následující rovnosti:

$$\begin{aligned}
\mathcal{A}[\![sources]\!]((A_1, t_1, P_1, Atr_1)) &= A_1 \\
\mathcal{A}[\![targets]\!]((A_1, t_1, P_1, Atr_1)) &= P_1 \\
\mathcal{A}[\![timestamp]\!]((A_1, t_1, P_1, Atr_1)) &= t_1 \\
\forall (x, y) \in Atr_1 : \mathcal{A}[\![x]\!]((A_1, t_1, P_1, Atr_1)) &= y
\end{aligned}$$

Mějme posloupnost splňujících událostí $a_i = (A_i, t_i, P_i, Atr_i)$, kde $i \in \{1, \ldots, n\}$ a právě zpracovávanou událost b. Dále mějme množiny $X = \{1, \ldots, (n-1)\} \cup \{* - 1, \ldots, * - (n-1)\}$ a $Y = \{2, \ldots, n\} \cup \{*, * - 1, \ldots, * - (n-2)\}$ a agregační funkci f se zápisem f. Potom

platí následující rovnosti:

$$
\begin{aligned}
&\mathcal{A}[\![EVENT.Pr]\!]\mathcal{S} \Cup b && = \mathcal{A}[\![Pr]\!](b) \\
&\mathcal{A}[\![EVENT(*).Pr]\!]\mathcal{S} && = \mathcal{A}[\![Pr]\!]((A_n, t_n, P_n, Atr_n)) \\
&\forall x \in \{1, \ldots, n\} : && \\
&\quad \mathcal{A}[\![EVENT(x).Pr]\!]\mathcal{S} && = \mathcal{A}[\![Pr]\!]((A_x, t_x, P_x, Atr_x)) \\
&\forall x \in \{0, \ldots, n-1\} : && \\
&\quad \mathcal{A}[\![EVENT(*-x).Pr]\!]\mathcal{S} && = \mathcal{A}[\![Pr]\!]((A_{n-x}, t_{n-x}, P_{n-x}, Atr_{n-x})) \\
&\forall x \in X, y \in Y, x < y : && \\
&\quad \mathcal{A}[\![EVENT(x,y).Pr]\!]\mathcal{S} && = (\mathcal{A}[\![EVENT(x).Pr]\!]\mathcal{S}, \ldots, \\
&&& \quad \mathcal{A}[\![EVENT(y).Pr]\!]\mathcal{S}) \\
&\forall x \in X, y \in Y, x < y : && \\
&\quad \mathcal{A}[\![EVENT(x,y).\mathbf{f}(Pr)]\!]\mathcal{S} && = f(\mathcal{A}[\![EVENT(x).Pr]\!]\mathcal{S}, \ldots, \\
&&& \quad \mathcal{A}[\![EVENT(y).Pr]\!]\mathcal{S})
\end{aligned}
$$

Mějme posloupnost splňujících událostí $\mathcal{S}$ a sled událostí b, který patří do posloupnosti splňujících událostí. Tento sled je odkazován zápisem $\mathbf{b}$. Potom platí následující rovnost:

$$\mathcal{A}[\![\mathbf{b}.Rest]\!]\mathcal{S} = \mathcal{A}[\![Rest]\!](b)$$

Mějme n-ární funkci f zapisovanou jako $\mathbf{f}$. Bez újmy na obecnosti předpokládejme, že je tato funkce zapisovaná prefixově. Potom platí následující rovnost:

$$\mathcal{A}[\![\mathbf{f}(arg_1, \ldots, arg_n)]\!]\mathcal{S} = f(\mathcal{A}[\![arg_1]\!]\mathcal{S}, \ldots, \mathcal{A}[\![arg_n]\!]\mathcal{S})$$

5.3.2 Pravdivostní výrazy

V základu jazyka DAQL neexistují konstanty pro popis pravdivostních hodnot, v případě nutnosti je tedy nutné použít tautologii (například $1 = 1$) případně kontradikci (například $1 = 0$).

Mějme n-ární relaci ρ zapisovanou jako $\mathbf{r}$. Bez újmy na obecnosti předpokládejme, že je tato relace zapisována prefixově. Potom platí následující rovnosti:

$$\mathcal{B}[\![\mathbf{r}(arg_1, \ldots, arg_n)]\!]\mathcal{S} = \begin{cases} \text{true} & \Leftrightarrow \quad (\mathcal{A}[\![arg_1]\!]\mathcal{S}, \ldots, \mathcal{A}[\![arg_n]\!]\mathcal{S}) \in \rho \\ \text{false} & \quad \text{jinak} \end{cases}$$

$$\mathcal{B}[\![b_1 \text{ AND } b_2]\!]\mathcal{S} = \begin{cases} \text{true} & \Leftrightarrow \ \mathcal{B}[\![b_1]\!]\mathcal{S} = \text{true} \wedge \mathcal{B}[\![b_2]\!]\mathcal{S} = \text{true} \\ \text{false} & \text{jinak} \end{cases}$$

$$\mathcal{B}[\![b_1 \text{ OR } b_2]\!]\mathcal{S} = \begin{cases} \text{true} & \Leftrightarrow \ \mathcal{B}[\![b_1]\!]\mathcal{S} = \text{true} \vee \mathcal{B}[\![b_2]\!]\mathcal{S} = \text{true} \\ \text{false} & \text{jinak} \end{cases}$$

$$\mathcal{B}[\![NOT\ b]\!]\mathcal{S} = \neg\mathcal{B}[\![b]\!]\mathcal{S}$$

5.3.3 Konstrukce

Konstrukce přejmenování

Mějme událost $e = (A_1, t_1, P_1, Atr_1)$, která patří do posloupnosti splňujících událostí $\mathcal{S}$. Dále mějme událost e', která je definována čtveřicí $(A_1, t_1, P_1, Atr_1 \cup \{(\%name,' jmeno')\})$, kde *%name* je systémový atribut, pro který neplatí omezení $((an, av_1) \in Atr \wedge (an, av_2) \in Atr) \Leftrightarrow av_1 = av_2$ (může tedy nabývat více hodnot). Potom platí následující rovnost:

$$\mathcal{C}[\![\text{e AS } jmeno]\!]\mathcal{S} = \mathcal{S}[e/e']$$

Zápis $\mathcal{S}[e/e']$ v předchozí rovnosti znamená nahrazení události e událostí e' v posloupnosti splňujících událostí.

Konstrukce projekce

Buď c zápisem dotazu 5.1 a *toset* funkce, která ze seznamů (případně jednotlivých entit) vytvoří množiny. Potom platí následující rovnost:

$$\begin{aligned}\mathcal{C}[\![c]\!]\mathcal{S} =& ((toset(\mathcal{A}[\![atr_2]\!]\mathcal{S}), \mathcal{A}[\![atr_1]\!]\mathcal{S}, toset(\mathcal{A}[\![atr_3]\!]\mathcal{S}), \\ & \{(atribut\_1, \mathcal{A}[\![atr_4]\!]\mathcal{S}), \ldots, (atribut\_n, \mathcal{A}[\![atr_{n+3}]\!]\mathcal{S})\}))\end{aligned}$$

Dotaz 5.1 Konstrukce projekce

SELECT
atr_1 **AS** timestamp,
atr_2 **AS** sources,
atr_3 **AS** targets,
atr_4 **AS** atribut_1,
...,
atr_{n+3} **AS** atribut_n

Vstupní podmínky, konstrukce události

Buď c zápisem dotazu 5.2, $\mathcal{S}$ posloupnost splňujících událostí a e právě zpracovávaná událost. Potom platí rovnost:

$$\mathcal{C}[\![c]\!]\mathcal{S} = \begin{cases} \mathcal{S} \Cup e & \Leftrightarrow \quad (\mathcal{B}[\![b_W]\!]\mathcal{S} \Cup e \wedge (\mathcal{B}[\![b_1]\!]\mathcal{S} \Cup e \vee \cdots \vee \mathcal{B}[\![b_n]\!]\mathcal{S} \Cup e)) \\ \mathcal{S} & \quad \text{jinak} \end{cases}$$

Dotaz 5.2 Konstrukce události

```
1: ...
2:     EVENT
3:         FILTERED BY
4:             b_1
5:     AS jmeno_1
6:     ...
7:     EVENT
8:         FILTERED BY
9:             b_n
10:    AS jmeno_n
11: WHERE
12:    b_W
```

Konstrukce množiny a příslušných modifikací

Denotační sémantika těchto konstrukcí bude bez újmy na obecnosti uvedena pouze s ukončující podmínkou uvozenou klíčovým slovem *WHILE*. Pro případy, při kterých dojde ke splnění podmínky ukončení a zároveň k nesplnění výstupních podmínek, je nutné definovat speciální návratovou hodnotu FAIL. Pokud dojde při zpracování dotazu k obdržení této návratové hodnoty, je zpracování dané instance ukončeno aniž by byla vygenerována událost.

Buď c zápisem dotazu 5.3, $\mathcal{S}$ posloupností splňujících událostí a $\bigcirc_{\min} : Ev^n \longrightarrow Ev$ funkce taková, že platí $\bigcirc_{\min}((e_1, \ldots, e_n)) = (e_1 \circ_{\min} \cdots \circ_{\min} e_n)$. Potom platí také následující rovnost:

$$\mathcal{C}[\![c]\!]\mathcal{S} = \begin{cases} (\bigcirc_{\min}(\mathcal{S})) & \Leftrightarrow \quad \mathcal{B}[\![b_W]\!]\mathcal{S} \wedge \mathcal{B}[\![b_H]\!]\mathcal{S} \\ \mathcal{S} & \Leftrightarrow \quad \neg\mathcal{B}[\![b_W]\!]\mathcal{S} \\ \text{FAIL} & \quad \text{jinak} \end{cases}$$

Dotaz 5.3 Konstrukce množiny

```
1: SET WHILE
2:     b_W
3:     ...
4: HAVING
5:     b_H
```

Buď c zápisem dotazu 5.4 a $\mathcal{S}$ posloupností splňujících událostí. Potom platí také následující rovnost:

$$\mathcal{C}[\![c]\!]\mathcal{S} = \begin{cases} (\bigcirc_{\min}(\mathcal{S})) & \Leftrightarrow \quad \mathcal{B}[\![b_W]\!]\mathcal{S} \wedge \mathcal{B}[\![b_H]\!]\mathcal{S} \\ \mathcal{S} & \Leftrightarrow \quad \neg\mathcal{B}[\![b_W]\!]\mathcal{S} \\ () & \quad \text{jinak} \end{cases}$$

Dotaz 5.4 Konstrukce iterativní množiny

```
1: ITERATIVE SET WHILE
2:     b_W
3:     ...
4: HAVING
5:     b_H
```

Buď c zápisem dotazu 5.5 a $\mathcal{S}$ posloupností splňujících událostí. Dále mějme funkce *split* a *second*. Aplikace funkce $split(\mathcal{E}, Bexp_1)$ rozdělí posloupnost událostí $\mathcal{E} = (e_1, \ldots, e_i, e_{i+1}, \ldots, e_n)$ na posloupnosti $\mathcal{E}_1 = (e_1, \ldots, e_i)$ a $\mathcal{E}_2 = (e_{i+1}, \ldots, e_n)$ právě tehdy, když platí následující rovnosti:

$$\mathcal{B}[\![Bexp_1]\!]\mathcal{E}_2 = \text{false}$$
$$\mathcal{B}[\![Bexp_1]\!](e_i, \ldots, e_n) = \text{true}$$

Přičemž $\mathcal{E}_1$ je nejmenší taková posloupnost, pro které dané rovnosti platí.

Funkce *second* se aplikuje na dvojici a vrací její druhý prvek.

Potom platí také následující rovnost:

$$\mathcal{C}[\![c]\!]\mathcal{S} = \begin{cases} (\bigcirc_{\min}(\mathcal{S})) & \Leftrightarrow \quad \mathcal{B}[\![b_W]\!]\mathcal{S} \wedge \mathcal{B}[\![b_H]\!]\mathcal{S} \\ \mathcal{S} & \Leftrightarrow \quad \neg\mathcal{B}[\![b_W]\!]\mathcal{S} \\ second(split(\mathcal{S}, b_W)) & \quad \text{jinak} \end{cases}$$

Dotaz 5.5 Konstrukce klouzavé množiny

1: **SLIDING SET WHILE**
2: b_W
3: ...
4: **HAVING**
5: b_H

6 Překlad dotazovacího jazyka

6.1 Zpracování dotazu

Jak již bylo zmíněno, každý dotaz jazyka DAQL je popisem stavového automatu, jehož vstupním řetězcem je proud událostí. Zároveň je možné ukázat, že přechodová funkce daného automatu není v okamžiku formulace (a následného překladu) dotazu nutně zcela definovaná. Tato částečná nedefinovanost je dána možnou vzájemnou závislostí mezi jednotlivými událostmi posloupnosti.

Mějme dotaz 4.10. Pro jednoduchost považujme množinu událostí *DNSANOMALY* (řádek 17) za jednu událost. Tuto událost nazvěme *dns_set*. Automat popsaný tímto dotazem je tvořen čtyřmi stavy (vstupní stav, stav po přečtení události *first_scan*, stav po přečtení události *dns_set* a stav po přečtení události *second_scan*, který je zároveň i koncovým stavem).

V okamžiku překladu je však plně definován pouze přechod z prvního do druhého stavu (automat přejde do druhého stavu, objeví-li se na vstupu událost, jejíž atributy *kód metody* a *číslo portu* nabývají po řadě hodnot *SCANS* a *3389*). Přechody mezi následujícími stavy vyžadují znalost parametrů události *first_scan*, což není v čase překladu možné.

Uvažme tedy událost *first_scan* jako událost definovanou následující n-ticí:

$$(\{10.0.10.10\}, 1353164370, \{192.168.1.113\}, \{('code','SCANS'),\\ ('port','3389')\})$$

Z podmínky *dns_set.sources IN first_scan.targets* a nejvíce šestihodinové časové mezery meti událostmi *first_scan* a *dns_set* plyne, že automat přejde do dalšího stavu pod událostí, jejímž původcem je stanice identifikovaná IP adresou 192.168.1.113 a jejíž časová známka není větší než 1353185970 ($1353185970 = 6 * 3600 + 1353164370$). Další omezení pro tento přechod plynou přímo z dotazu (nezávisle na parametrech popisujících událost *first_scan*) a jsou tak známa už v čase překladu.

Podobně lze postupovat i při definici dalšího přechodu.

6.1.1 Implementace

Zpracování dotazu jazyka DAQL bude implementováno ve dvou základních částech. První část představuje správce dotazů, druhá část je tvořena samotným překladačem a reprezentací přeložených dotazů.

Správce dotazů poskytuje jednotlivým dotazům rozhraní pro přístup k událostem. Vzhledem k různé reprezentaci událostí v jednotlivých prostředích (respektive datových modelech) je toto rozhraní nutné implementovat v rámci rozšíření jazyka. V textu níže je popsáno rozhraní pro zpracování událostí generovaných aplikací ADS. Správce dotazů také spravuje instance jednotlivých dotazů (např. udržuje databázi jednotlivých instancí dotazů a informace o stavech dosažených danými instancemi dotazu, ukládá data o splňujících událostech, poskytuje parametry pro změnu příslušné přechodové funkce, ...). Správce dotazů dále kontroluje aktuální nedosažitelnost koncových stavů z některého již dosaženého stavu dané instance přechodového systému reprezentujícího dotaz. V případě nedosažitelnosti je tato instance dotazu včetně doplňujících informací odstraněna z databáze správce.

Překladač dotazu překládá dotaz jazyka DAQL do přechodové funkce automatu popsaného dotazem, dále poskytuje informace o chybách v nepřeložitelném dotazu a o nedosažitelnosti koncového stavu, optimalizuje dotaz podle statistik daného prostředí a využívá definovaná rozšíření jazyka DAQL.

Následující specifikace jednotlivých komponent překladu a zpracování dotazů jazyka DAQL je určena pro objektově orientované programovací jazyky s využitím některých aspektů programovacího jazyka Caché ObjectScript a databázového systému Caché. U použitých českých termínů jsou v závorce uvedeny odpovídající termíny používané po řadě v jazycích Caché ObjectScript a Java.

6.1.2 Reprezentace událostí

Množina událostí je reprezentována třídou, jejíž instance představují jednotlivé události. Objekt třídy je plně popsán čtyřmi datovými poli (*Properties*, *Fields*) odpovídajícími jednotlivým prvkům čtveřice definující událost (z formální definice). Datová pole *sources* a *targets*

mohou být tvořeny pouze seznamy (*List*, *Collection list*) identifikátorů (datový typ v závislosti na rozšíření), datové pole *timestamp* nabývá hodnot z přiřazeného datového typu a datové pole *attributes* je tvořeno objektem pole (třída *%Library.ArrayOfDataTypes*, v jazyce Java odpovídá této třídě kolekce *SortedMap*), ve kterém slouží identifikátory atributů jako klíče, pod kterými jsou dostupné hodnoty příslušných atributů.

V rámci třídy jsou dále implementovány instanční metody (*Method*, *Instance Method*) pro aplikaci funkcí projekce (viz 4.2.4) a pojmenování atributů (pojmenování událostí je implementováno v rámci překladače). Třída obsahuje i metody třídy (*Static/Class Methods*) implementující funkce $\circ_{\min}$ a $\circ_{\max}$, kontrolu rovnosti dvou událostí a seřazení seznamu událostí podle časových známek jednotlivých událostí.

Dvě události jsou si rovny právě tehdy, když si jejich jednotlivá datová pole odpovídají. V případě datových polí obsahujících seznamy je nutné porovnat prvky seznamu nezávisle na jejich pořadí v seznamu, který zde reprezentuje množinu (implementace množiny pomocí seznamu byla zvolena vzhledem k neexistenci třídy popisující množinu v Caché ObjectScript).

Metody implementující funkce $\circ_{\min}$ a $\circ_{\max}$ přijímají na vstupu seznam událostí. Tyto události jsou v rámci metody nejprve seřazeny podle svých časových známek, následně jsou atributy všech událostí seznamu přejmenovány. Nové identifikátory atributů jsou vytvářeny zřetězením řetězce „event_i.“, kde i představuje proměnnou nahrazovanou pořadím prvku v seřazeném seznamu, s řetězcem dosavadního identifikátoru atributu. Dále jsou ze seznamů původců a cílů vytvořeny nové seznamy, které obsahují všechny původce a cíle právě jednou. Z dosavadně vypočítaného je vytvořena nová událost s časovou známkou určenou příslušnou funkcí ($\max$, $\min$). Tato událost je vrácena jako návratová hodnota metody.

Metoda pro seřazení seznamu událostí je implementována pomocí globálů (v jazyce Java lze implementovat například pomocí algoritmu řazení haldou).

6.1.3 Reprezentace dotazu

Jak již bylo zmíněno, dotaz jazyka DAQL popisuje přechodový systém (stavový automat), jehož vstupním slovem je posloupnost událostí. Jednotlivé přechodové systémy budou implementovány vlastními třídami. Třídy nebudou mít žádné vlastnosti (*Properties*, *Fields*), ale pouze metody třídy (*Static/Class Methods*) definující přechodovou funkci a poskytující rozhraní správci událostí.

Metoda popisující stavy automatu

Tato metoda poskytuje správci dotazů rozhraní, které pro každé číslo stavu daného přechodového systému předá návratovou hodnotou seznam parametrů potřebných pro definici přechodu z daného stavu do následujícího. Takovýmto parametrem upravující přechodovou funkci může být například identifikátor původce předchozích událostí z posloupnosti splňujících událostí.

Seznam odpovídajících hodnot parametrů je následně předán při samotném zpracování události.

Metody popisující přechody mezi stavy

Tyto metody jsou generovány pro každý stav automatu zvlášť. Společně popisují přechodovou funkci automatu. Metody tohoto typu očekávají na vstupu dva argumenty – seznam hodnot parametrů modifikujících přechodovou funkci a objekt právě zpracovávané události. Výstupem daných metod je vždy seznam stavů, které byly pod danou událostí z daného stavu dosaženy. Pokud nedojde k přechodu do dalšího stavu, obsahuje seznam pouze identifikaci aktuálně dosaženého stavu. V případě, že je seznam více než jednoprvkový, je nutné pro každý další stav vytvořit novou instanci dotazu, která je kopií současné instance lišící se pouze v dosaženém stavu. Při ukončení dotazu metoda vrací jednoprvkový seznam obsahující pouze řetězec *FIN* (úspěšné ukončení), nebo řetězec *FAIL* (neúspěšné ukončení).

Některé dotazy mohou při své reprezentaci využívat pouze jeden stav pro samotný výpočet, průběh výpočtu se potom projeví pouze jako změny hodnot parametrů modifikujících přechodovou funkci.

Metody dále odkazem předávají události, které mají rozšířit posloupnost splňujících událostí a je tak nutné je zapsat do příslušné datové struktury.

Metody pro kontrolu podmínek ukončení a výstupních podmínek

Tyto metody jsou generovány zvlášť pro každou konstrukci sledu v dotazu. Metody využívají rozhraní správce dotazů pro získání dodatečných dat o posloupnosti splňujících událostí. Volány jsou pouze metodami popisujícími přechodovou funkci. Pro správce dotazů je generována jejich modifikace sloužící ke kontrole možné splnitelnosti dotazu (kvůli případnému mazání instancí dotazů).

Reprezentace instance dotazu

Každá z instancí dotazu je v databázi uložena pomocí globálu. První klíč tohoto pole je tvořen identifikátorem dotazu, druhý klíč odpovídá identifikátoru dané instance. Na další úrovni je listový uzel identifikovaný klíčem *state*, který obsahuje informaci o posledním dosaženém stavu. Klíčem *events* je určený podstrom, který obsahuje posloupnost splňujících událostí.

Jednotlivé události jsou identifikovány podle pořadí svého výskytu v rámci svého sledu. Uzly, jejichž klíčem je právě číslo události, obsahují jako hodnotu rozlišení, jestli se jedná o událost (*event*) nebo dosud neuzavřený sled (*sequence*).

V případě, že se jedná o událost, jsou na další úrovni uzly, které mají jako klíč řetězce *sources*, *targets*, *timestamp* a identifikátory jednotlivých atributů události. Tyto uzly pak mají jako hodnotu přiřazena příslušná data.

Pokud se jedná o dosud neuzavřený sled, jsou pod tímto uzlem opět čísla identifikující jednotlivé dílčí události sledu. V okamžiku splnění podmínky ukončení a výstupní podmínky (tj. „uzavření“ sledu), dojde k přejmenování atributů dílčích událostí a celý podstrom je transformován do podoby odpovídající jedné události.

6.1.4 Překladač

Překladač jazyka DAQL provádí postupně několik kroků. Prvním krokem je syntaktická analýza, během které se při čtení dotazu současně provádí i analýza lexikální. Lexikální analýza slouží k vyhledání a označení klíčových slov, označení identifikátorů, označení konstant podle příslušnosti k datovým typům a označení funkcí a relací podle arity a požadovaných datových typů.

Výsledkem tohoto prvního kroku je derivační strom, který v listových uzlech obsahuje označkované terminální řetězce dotazovacího jazyka. Tento strom je pomocí několika pravidel transformován do stromu obsahujícího pouze označkované terminální řetězce a pomocné symboly. Transformační pravidla jsou popsána v příloze E.

Dalším krokem překladu je sémantická analýza výsledného stromu. Tato analýza prochází strom a kontroluje aritu a příslušné datové typy funkcí nebo relací a jejich argumentů a omezení daná definicí jazyka (nemožnost použití agregačních funkcí s výjimkou funkce *COUNT* jako součást vstupních podmínek, nemožnost odkazovat se v rámci *FILTERED BY* podmínky na jinou než právě zpracovávanou událost a podobně). Sémantická analýza kontroluje také úplnost konstrukce projekce.

Posledním krokem před samotným překladem je nahrazení zápisu funkcí, u kterých jsou všechny argumenty konstantami, výsledkem jejich aplikace. Řetězce specifikující časové trvání (například *9 HOURS*) jsou nahrazeny ekvivalentní hodnotou počtu sekund. Je rozhodnuta pravdivost formulí založených na konstantních hodnotách.

Po těchto krocích už je možné začít se samotným překladem. Použité identifikátory jsou využity k vygenerování příslušných seznamů parametrů modifikujících přechodovou funkci, samotné identifikátory jsou následně nahrazeny odkazy na tyto seznamy, které budou jednotlivým metodám předávány jako jeden z argumentů. Funkce a relace jsou nahrazeny voláním jejich implementací v příslušných definicích datových typů.

Dále jsou podle jednotlivých podmínek ukončení a výstupních podmínek vytvořeny metody obsluhující kontroly sledů. V těchto metodách je zahrnuta také obsluha klouzavých a iterujících množin. V metodě obsluhující sled nejvyšší úrovně je zahrnuta také konstruk-

ce projekce.

Posledním krokem překladu je vytvoření metod popisujících přechodovou funkci. K jejich generování je využit strom vytvořený podle zmíněných transformačních pravidel z derivačního stromu. Při generování těchto metod se postupuje od nejvíce zanořeného sledu po sled nejvyšší úrovně, nakonec je vygenerována metoda popisující přechodovou funkci z iniciálního stavu. Pro každou konstrukci události v daném sledu je vytvořena jedna metoda. Pokud je daný sled množinou, jsou s využitím vstupních podmínek vytvořeny přechody mezi všemi těmito metodami. Pokud se jedná o posloupnost, musí přechodová funkce odpovídat pořadí událostí, které je dané dotazem.

Je-li v množině zanořena množina, je nutné doplnit přechody mezi všemi událostmi (stavy) obou množin. Je-li v množině zanořena posloupnost, opět budou doplněny přechody mezi všemi stavy, přechodová funkce (ve směru do stavů popisujících události posloupnosti) však musí respektovat pořadí událostí posloupnosti, proto musí tyto stavy jako modifikátory přechodové funkce vyžadovat také identifikace stavů dosažených v rámci dané posloupnosti.

Je-li zanořena množina v posloupnosti, je nutné doplnit přechody ze stavu popisujícího předchozí událost posloupnosti do všech stavů popisujících události dané množiny. Do metody pro kontrolu ukončení dané množiny je potom nutné doplnit informaci o přechodu do stavu popisujícího následující událost posloupnosti. V případě zanoření posloupnosti do posloupnosti je nutné doplnit přechod mezi stavem popisujícím předchozí událost posloupnosti a prvním stavem zanořené posloupnosti. Metoda pro kontrolu ukončení vnořené posloupnosti je doplněna stejně jako v případě vnořené množiny.

Pro vytvoření přechodové funkce z iniciálního stavu je využit stejný postup jako při přechodu do sledu zanořeného v posloupnosti.

Po úspěšném překladu je nutné dotaz zaregistrovat do tabulky dotazů (pro případné úpravy, aktivaci a deaktivaci).

6.1.5 Správce dotazů

Správce dotazů je tvořen třemi základními komponentami, z toho jedna musí být implementována pro konkrétní použití. Touto do-

datečně vytvářenou komponentou je transformace událostí generovaných systémem do formátu, v jakém je očekává komponenta zpracovávající dotazy a zase zpět. Třída implementující tuto funkcionalitu musí být pojmenována *Connector* a musí patřit do balíčku tříd *DAQL.Manager*. Pro předání události ke zpracování musí použít metodu *Process* třídy *Processor* balíčku *DAQL.Manager*. Tato metoda přijímá jako argument objekt události a vrací chybová hlášení o formátu události, případně číslici jedna, pokud se žádná chyba nevyskytne.

Třída *Connector* musí pro komponentu zpracovávající dotazy poskytovat metodu *Output* přijímající jako argument objekt události. Tato metoda zajistí potřebné modifikace události a zpracování systémem, do kterého je správce dotazů zapojený. Takovýmto následným zpracováním může být například uložení v databázi.

Další komponenta správce dotazů slouží pro údržbu databáze instancí dotazů. Z databáze v pravidelných cyklech maže instance dotazů, které jsou ve stavu, ze kterého již není dosažitelný koncový stav (takovýmto stavem může být například vypršení časového limitu omezujícího konstrukci posloupnosti). Instance dotazů, u kterých po danou dobu nedojde k rozšíření posloupnosti splňujících událostí, jsou také smazány. Dále jsou mazána data o událostech posloupnosti splňujících událostí, která už nejsou v dané instanci dotazu odkazována.

Nejdůležitější komponentou správce dotazů je komponenta zpracovávající dotazy. Úkolem této komponenty je ověření správnosti předané události (například nesmí chybět časová známka události). Následně se pro všechny aktivní instance dotazů zjistí jejich dosažený stav a pro každý takový stav se získá seznam požadovaných parametrů modifikujících přechodovou funkci. Podle požadavků jsou v databázi nalezena příslušná data a ta předána spolu s událostí ke zpracování jednotlivými dotazy. Proměnná obsahující událost je ke zpracování předána odkazem.

Pokud metoda zpracovávající událost vrátí jako návratovou hodnotu seznam obsahující pouze řetězec *FAIL*, je instance dotazu včetně veškerých uložených dat smazána. Pokud je po zpracování události vrácen seznam obsahující řetězec *FIN*, je událost uložená v proměnné, která byla zpracovávající metodě předána odkazem, odeslána komponentě sloužící pro spojení správce událostí s okolním systémem

(třída *Connector*). Pokud seznam obsahuje jiné hodnoty (čísla stavů), uloží správce dotazů událost obsaženou v předávané proměnné do posloupnosti splňujících událostí. Při uložení se použije systémový atribut *%identifier*, který určí, kam se má událost ve stromu uložit (hodnota atributu představuje plný název události s využitím tečkové syntaxe). Pokud je proměnná prázdná, neukládá se nic. Po tomto kroku převede správce dotazů instanci dotazu do nového stavu, případně vytvoří její kopie.

Kromě samotného ověření správnosti formátu události vše ostatní probíhá na pozadí.

Rozšíření

Na tomto místě je stručně popsán návrh komponenty pro konverzi událostí pro aplikaci ADS. Aplikace ADS má veškeré potřebné informace o událostech uloženy ve třech tabulkách. První tabulkou je tabulka `event`, která obsahuje zdrojovou IP adresu v desítkovém zápisu (v případě IPv6 adresy je použit plný formát zápisu, ve kterém jsou vynechány rozdělovací dvojtečky), identifikaci zdroje NetFlow dat, časovou známku, kód metody, která událost vygenerovala, detail události v čitelné podobě a seznam atributů. V této tabulce také může být zaznamenáno doménové jméno původce událostí, které mu bylo přiřazeno v době vygenerování události. Další tabulkou je tabulka `participant`, která obsahuje IP adresy cílů událostí (ve stejném formátu jako zdrojová IP adresa v tabulce `event`). Kvůli použitému datovému modelu je v některých konkrétních případech role cílů a původců událostí přiřazena opačně, je tedy nutné to při konverzi brát v úvahu. Poslední tabulkou, která je nutná pro konverzi mezi formáty událostí, je tabulka `attributes`. Ta obsahuje seznam jmen atributů (jména odpovídají pozicí v seznamu hodnotě na stejné pozici v seznamu uloženém v tabulce `event`).

Z takto získaných informací je možné vytvořit událost ve dříve popsaném formátu. Nejprve se vytvoří seznamy původců a cílů událostí (s ohledem na možnou výměnu rolí). Předpokládejme, že datový typ *IP* je definován pro stejný formát. Následně se časová známka (uložená ve formátu *RRRR-MM-DD hh:mm:ss*) převede do formátu definovaného datovým typem přiřazeným časové známce. V takovémto případě převedeme časovou známku do počtu sekund od

začátku éry.

Následně se ze zbývajících hodnot popisujících událost vytvoří dvojice atribut-hodnota. Seznam atributů je tak rozšířen o příslušná jména. Atribut obsahující identifikaci zdroje dat je pojmenován *nfcapd*, atribut obsahující kód metody, která událost vygenerovala potom *code*. Pro další zpracování není nutné předávat řetězec popisující událost použitý při zobrazení uživateli.

Takto modifikovaná událost již může být předána ke zpracování instancemi dotazů.

7 Závěr

Tato práce v různé míře splňuje jednotlivé požadavky dané zadáním. Navrhovaný dotazovací jazyk je popsán poměrně podrobně, avšak při implementaci příslušného překladače bude nutné některé myšlenky více rozvinout. Dotazovací jazyk je předkládán v samostatně nepoužitelné podobě, díky možnostem rozšíření však není omezen pouze na použití v prostředí síťové bezpečnosti, pro které byl původně navrhován. V textu práce je uveden také jednoduchý příklad dotazu aplikujícího jednoduché metody strojového učení (viz dotaz 4.7). Pro vytváření dotazů umožňujících využití složitějších metod (například shluková analýza a detekce odlehlých bodů) postačuje definovat příslušné funkce v rámci implementace daného datového typu.

Dalším cílem bude implementace překladače, správce dotazů a vybraných datových typů pro potřeby rozšíření aplikace ADS. Po dokončení implementace původního návrhu jazyka bude jazyk dále upravován podle zkušeností uživatelů. Dále je vhodné navrhnout optimalizaci dotazů. Jedním z dalších cílů může být také navržení uživatelského rozhraní, které by pomocí grafických prvků umožňovalo sestavení dotazu bez znalosti navrhovaného dotazovacího jazyka. Dalším možným rozšířením práce je návrh algoritmu strojového učení, který by dokázal pro několik zvolených posloupností událostí vytvořit dotaz, který by všechny tyto události popisoval.

Po přihlédnutí k rozsahu této práce lze konstatovat, že zadání bylo prací splněno a že práce poslouží jako dobrý základ pro samotný vývoj příslušného rozšíření.

Literatura

[1] EsperTech. *EsperHA 4.7.0*. [software]. Wayne: EsperTech, 2012.

[2] LUCKHAM, David. *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*. Boston: Addison-Wesley, 2001.

[3] KŘETÍNSKÝ, Mojmír. *IA006: Vybrané kapitoly z teorie automatů*. [cyklus přednášek]. Brno: Fakulta informatiky Masarykovy univerzity, 2010.

[4] KUČERA, Antonín. *IA011: Sémantiky programovacích jazyků*. [cyklus přednášek]. Brno: Fakulta informatiky Masarykovy univerzity, 2012.

[5] ČEŠKA, M., T. HRUŠKA a M. BENEŠ. *Překladače*. [učební text]. Brno: Vysoké učení technické, 1993. Dostupné také z: `http://www.fit.vutbr.cz/~meduna/fjp/skripta.pdf`.

[6] JIE, C., H. HE, G. WILLIAMS a H. JIN. *Temporal Sequence Associations for Rare Events*. In: PAKDD 2004. London: Springer, 2004.

[7] VILALTA, Ricardo a Ma SHENG. *Predicting Rare Events In Temporal Domains*. In: Data Mining 2002. Washington: IEEE, 2002.

[8] LAZAREVIČ, A., J. SRIVASTAVA a V. KUMAR. *Tutorial: Data Mining for Analysis of Rare Events: A case study in Security, Financial and Medical Applications*. In: PAKDD 2004. London: Springer, 2004.

[9] MENG, Y., M. H. DUNHAM, F. M. MARCHETTI a J. HUANG. *Rare Event Detection in a Spatiotemporal Environment*. In: Proceedings of the IEEE International Conference on Granular Computing. Washington: IEEE, 2006.

[10] BIFET, A., G. HOLMES, R. KIRKBY a B. PFAHRINGER. *Data Stream Mining: A Practical Approach*. Waikato: The University Of Waikato, 2011.

[11] DUNHAM, M. H., Y. MENG a J. HUANG. *Extensible Markov Model*. In: ICDM 'O4. Washington: IEEE, 2004.

[12] KALEKAR, Prajakta S. *Time series Forecasting using Holt-Winters Exponential Smoothing*. Bombaj: Kanwal Rekhi School of Information Technology, 2004.

[13] Microsoft Corporation. *Worm:Win32/Morto.A*. [online]. Redmond: Microsoft Corporation, 2011. Poslední úpravy 1.9.2011 [cit. 18.11.2012]. Dostupné z: `http://www.microsoft.com/security/portal/Threat/Encyclopedia/Entry.aspx?name=Worm%3AWin32%2FMorto.A`.

[14] JOE TOUCH et al. *Service Name and Transport Protocol Port Number Registry*. [online]. Los Angeles: Internet Assigned Numbers Authority, 2012. Poslední úpravy 14.11.2012 [cit. 18.11.2012]. Dostupné z: `http://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xml`.

[15] RAINER GERHARDS. *The Syslog Protocol*. [RFC 5424]. IETF, 2009. Request for Comments; 5424. Dostupné z: `http://www.ietf.org/rfc/rfc5424.txt`.

[16] JEFF CASE et al. *Introduction and Applicability Statements for Internet Standard Management Framework*. [RFC 3410]. IETF, 2002. Request for Comments; 3410. Dostupné z: `http://www.ietf.org/rfc/rfc3410.txt`.

[17] BENOIT CLAISE. *Cisco Systems NetFlow Services Export Version 9*. [RFC 3954]. IETF, 2004. Request for Comments; 3954. Dostupné z: `http://www.ietf.org/rfc/rfc3954.txt`.

[18] SHUTKO, Alexander. *UnOfficial OSCAR (ICQ v7/v8/v9) protocol documentation*. [online]. Dostupné z: `http://iserverd.khstu.ru/oscar/`.

[19] SAINT-ANDRE, Peter. *Extensible Messaging and Presence Protocol (XMPP): Core*. [RFC 6120]. IETF, 2011. Request for Comments; 6120. Dostupné z: `http://www.ietf.org/rfc/rfc6120.txt`.

[20] ADVAICT, a.s. *Anomaly Detection System 3.10.00*. [software]. Brno: AdvaICT, a.s., 2012.

[21] PETER HAAG et al. *NFDUMP – NetFlow processing tools*. [software]. Dostupné z: `http://sourceforge.net/projects/nfdump/`.

[22] INTERSYSTEMS CORPORATION. *Caché 2009.1.3*. [software]. Cambridge: InterSystems Corporation, 2009.

[23] INTERSYSTEMS CORPORATION. *Caché Technology Guide*. Cambridge: InterSystems Corporation, 2012. Dostupné z: `http://www.intersystems.com/cache/technology/techguide/CacheTechnologyGuide.pdf`.

[24] ČSN ISO/IEC 9075-1. *Informační technologie – Databázové jazyky – SQL – Část 1: Základní rámec (SQL/Základní rámec)*. Praha: Český normalizační institut, 2011.

[25] VAUGHN Randal a Gadi EVRON. *DNS Amplification Attacks*. [online]. ISOTF, 2006. Dostupné z: `http://www.isotf.org/news/DNS-Amplification-Attacks.pdf`.

Seznam dotazů

A Klíčová slova jazyka DAQL

AND	HAVING	SELECT
AS	HOURS	SEQUENCE
BETWEEN	HRS	SET
COUNT	ITERATIVE	SLIDING
DAYS	LIMITED BY	STRICT
EVENT	MINS	UNTIL
EVENTS	MINUTES	UP TO
FILTERED BY	NOT	WHERE
FOR ALL	OR	WHILE
FOR SOME	SAME	sources
FROM	SECONDS	targets
GAP	SECS	timestamp

B Funkce *FIRST*

$$
\begin{aligned}
FIRST(\langle S' \rangle) &= \{\text{SELECT}\} \\
FIRST(\langle S \rangle) &= \{\text{SELECT}\} \\
FIRST(\langle PROJECTION \rangle) &= \{\text{COUNT}, (, \#\text{idstring}, \#\text{num}\} \\
FIRST(\langle NEXTPROJ \rangle) &= \{,, \varepsilon\} \\
FIRST(\langle APPLICATION \rangle) &= \{\text{COUNT}, (, \#\text{idstring}, \#\text{num}\} \\
FIRST(\langle APPLICATION' \rangle) &= \{\text{COUNT}, \#\text{idstring}, \#\text{num}\} \\
FIRST(\langle APPLINF \rangle) &= \{+, \varepsilon\} \\
FIRST(\langle NEXT \rangle) &= \{,, \varepsilon\} \\
FIRST(\langle COMPLEXID \rangle) &= \{\#\text{idstring}\} \\
FIRST(\langle NEXTCOMP \rangle) &= \{., \varepsilon\} \\
FIRST(\langle APPLY \rangle) &= \{\text{COUNT}, \#\text{idstring}\} \\
FIRST(\langle SIMPLEID \rangle) &= \{\#\text{idstring}\} \\
FIRST(\langle NEXTID \rangle) &= \{., \varepsilon\} \\
FIRST(\langle INDEXED \rangle) &= \{(, \varepsilon\} \\
FIRST(\langle INDEX \rangle) &= \{\#\text{num}, * - \#\text{num}, *\} \\
FIRST(\langle LINDEX \rangle) &= \{,, \varepsilon\} \\
FIRST(\langle LINDEX' \rangle) &= \{\#\text{num}, * - \#\text{num}, *\} \\
FIRST(\langle CONSTANT \rangle) &= \{\#\text{num}\} \\
FIRST(\langle INFFUNCTOR \rangle) &= \{+\} \\
FIRST(\langle FUNCTOR \rangle) &= \{\text{COUNT}\} \\
FIRST(\langle STREAM \rangle) &= \{\text{ITERATIVE}, \text{SLIDING}, \text{SET}, \text{SEQUENCE LIMITED BY \#num}\} \\
FIRST(\langle STREAMX \rangle) &= \{\text{ITERATIVE}, \text{SLIDING}, \text{SET}, \text{SEQUENCE LIMITED BY \#num}\} \\
FIRST(\langle SET \rangle) &= \{\text{SET}\} \\
FIRST(\langle SETCONSTR \rangle) &= \{\text{UNTIL}, \text{WHILE}\} \\
FIRST(\langle CONSTR \rangle) &= \{\text{NOT}, (, \text{FOR ALL \#idstring}, \text{FOR SOME \#idstring}, \\
&\quad \text{SAME sources FOR ALL \#idstring}, \text{SAME targets FOR ALL \#idstring}, \\
&\quad \text{SAME}, \text{COUNT}, \#\text{idstring}, \#\text{num}, \text{match}\} \\
FIRST(\langle AFTERBR \rangle) &= \{\text{NOT}, (, \text{FOR ALL \#idstring}, \text{FOR SOME \#idstring}, \\
&\quad \text{SAME sources FOR ALL \#idstring}, \text{SAME targets FOR ALL \#idstring}, \\
&\quad \text{SAME}, \text{COUNT}, \#\text{idstring}, \#\text{num}, \text{match}\} \\
FIRST(\langle BRORNOT \rangle) &= \{), =, \text{BETWEEN}\} \\
FIRST(\langle ANDOR \rangle) &= \{\text{AND}, \text{OR}, \varepsilon\} \\
FIRST(\langle TAPPL \rangle) &= \{(, \text{timetamp}, \text{COUNT}\} \\
FIRST(\langle TAPPL' \rangle) &= \{+, \varepsilon\} \\
FIRST(\langle RELATION \rangle) &= \{\text{match}, \text{COUNT}, \\
&\quad \#\text{num}, \#\text{idstring}\} \\
FIRST(\langle PRREL \rangle) &= \{\text{match}\} \\
FIRST(\langle INFREL \rangle) &= \{=, \text{BETWEEN}\} \\
FIRST(\langle BINARY \rangle) &= \{=\}
\end{aligned}
$$

$FIRST(\langle TERNARY1 \rangle) = \{$BETWEEN$\}$
$FIRST(\langle TERNARY2 \rangle) = \{AND\}$
$FIRST(\langle WHERE \rangle) = \{$WHERE, $\varepsilon\}$
$FIRST(\langle HAVING \rangle) = \{$HAVING, $\varepsilon\}$
$FIRST(\langle SETBODY \rangle) = \{$SELECT, ITERATIVE, SLIDING, SET, SEQUENCE LIMITED BY #num, EVENT$\}$
$FIRST(\langle NEXTSETB \rangle) = \{$SELECT, ITERATIVE, SLIDING, SET, SEQUENCE LIMITED BY #num, EVENT, $\varepsilon\}$
$FIRST(\langle ASEVENT \rangle) = \{$SELECT, ITERATIVE, SLIDING, SET, SEQUENCE LIMITED BY #num, EVENT$\}$
$FIRST(\langle EVENT \rangle) = \{$SELECT, ITERATIVE, SLIDING, SET, SEQUENCE LIMITED BY #num, EVENT$\}$
$FIRST(\langle FILTER \rangle) = \{$FILTERED BY, $\varepsilon\}$
$FIRST(\langle TSPEC \rangle) = \{$DAYS, HRS, HOURS, MINS, MINUTES, SECS, SECONDS$\}$
$FIRST(\langle SEQBODY \rangle) = \{$SELECT, ITERATIVE, SLIDING, SET, SEQUENCE LIMITED BY #num, EVENT$\}$
$FIRST(\langle NEXTSEQB \rangle) = \{$SELECT, ITERATIVE, SLIDING, SET, SEQUENCE LIMITED BY #num, EVENT, GAP, $\varepsilon\}$
$FIRST(\langle GAPSPEC \rangle) = \{$UP TO #num, STRICT BETWEEN #num, BETWEEN #num$\}$

C Funkce *FOLLOW*

$$
\begin{aligned}
FOLLOW(\langle S' \rangle) &= \{\$\} \\
FOLLOW(\langle S \rangle) &= \{\$, \text{AS}\} \\
FOLLOW(\langle PROJECTION \rangle) &= \{\text{FROM}\} \\
FOLLOW(\langle NEXTPROJ \rangle) &= \{\text{FROM}\} \\
FOLLOW(\langle APPLICATION \rangle) &= \{\text{AS}, ,,), =, \text{BETWEEN, AND, OR, WHERE, HAVING, SET,} \\
&\quad \text{ITERATIVE, SLIDING, SELECT, EVENT, \$,} \\
&\quad \text{SEQUENCE LIMITED BY \#num}\} \\
FOLLOW(\langle APPLICATION' \rangle) &= \{), =, \text{BETWEEN}\} \\
FOLLOW(\langle APPLINF \rangle) &= \{\text{AS}, ,,), =, \text{BETWEEN, AND, OR, WHERE, HAVING, SET,} \\
&\quad \text{ITERATIVE, SLIDING, SELECT, EVENT, \$,} \\
&\quad \text{SEQUENCE LIMITED BY \#num}\} \\
FOLLOW(\langle NEXT \rangle) &= \{)\} \\
FOLLOW(\langle COMPLEXID \rangle) &= \{+, \text{AS}, ,,), =, \text{BETWEEN, AND, OR, WHERE, HAVING, SET,} \\
&\quad \text{ITERATIVE, SLIDING, SELECT, EVENT, \$,} \\
&\quad \text{SEQUENCE LIMITED BY \#num}\} \\
FOLLOW(\langle NEXTCOMP \rangle) &= \{+, \text{AS}, ,,), =, \text{BETWEEN, AND, OR, WHERE, HAVING, SET,} \\
&\quad \text{ITERATIVE, SLIDING, SELECT, EVENT, \$,} \\
&\quad \text{SEQUENCE LIMITED BY \#num}\} \\
FOLLOW(\langle APPLY \rangle) &= \{+, \text{AS}, ,,), =, \text{BETWEEN, AND, OR, WHERE, HAVING, SET,} \\
&\quad \text{ITERATIVE, SLIDING, SELECT, EVENT, \$,} \\
&\quad \text{SEQUENCE LIMITED BY \#num}\} \\
FOLLOW(\langle SIMPLEID \rangle) &= \{)\} \\
FOLLOW(\langle NEXTID \rangle) &= \{)\} \\
FOLLOW(\langle INDEXED \rangle) &= \{+, \text{AS}, ,,), =, \text{BETWEEN, AND, OR, WHERE, HAVING, SET,} \\
&\quad \text{ITERATIVE, SLIDING, SELECT, EVENT, ., \$,} \\
&\quad \text{SEQUENCE LIMITED BY \#num}\} \\
FOLLOW(\langle INDEX \rangle) &= \{)\} \\
FOLLOW(\langle LINDEX \rangle) &= \{)\} \\
FOLLOW(\langle LINDEX' \rangle) &= \{)\} \\
FOLLOW(\langle CONSTANT \rangle) &= \{+, \text{AS}, ,,), =, \text{BETWEEN, AND, OR, WHERE, HAVING, SET,} \\
&\quad \text{ITERATIVE, SLIDING, SELECT, EVENT, \$,} \\
&\quad \text{SEQUENCE LIMITED BY \#num}\} \\
FOLLOW(\langle INFFUNCTOR \rangle) &= \{\text{COUNT}, (, \text{\#idstring, \#num}\} \\
FOLLOW(\langle FUNCTOR \rangle) &= \{(\} \\
FOLLOW(\langle STREAM \rangle) &= \{\$, \text{AS}\} \\
FOLLOW(\langle STREAMX \rangle) &= \{\$, \text{AS, WHERE, HAVING}\} \\
FOLLOW(\langle SET \rangle) &= \{\$, \text{AS, WHERE, HAVING}\} \\
FOLLOW(\langle SETCONSTR \rangle) &= \{\text{SELECT, SLIDING, ITERATIVE, SEQUENCE LIMITED BY \#num,} \\
&\quad \text{EVENT, SET}\}
\end{aligned}
$$

$FOLLOW(\langle CONSTR \rangle)$ = {SELECT, SLIDING, ITERATIVE, SEQUENCE LIMITED BY #num, EVENT, SET,), AND, OR, WHERE, HAVING, AS, $}
$FOLLOW(\langle AFTERBR \rangle)$ = {SELECT, SLIDING, ITERATIVE, SEQUENCE LIMITED BY #num, EVENT, SET,), AND, OR, WHERE, HAVING, AS, $}
$FOLLOW(\langle BRORNOT \rangle)$ = {SELECT, SLIDING, ITERATIVE, SEQUENCE LIMITED BY #num, EVENT, SET,), AND, OR, WHERE, HAVING, AS, $}
$FOLLOW(\langle ANDOR \rangle)$ = {SELECT, SLIDING, ITERATIVE, SEQUENCE LIMITED BY #num, EVENT, SET,), WHERE, HAVING, AS, $}
$FOLLOW(\langle TAPPL \rangle)$ = {FOR ALL #idstring,)}
$FOLLOW(\langle TAPPL' \rangle)$ = {FOR ALL #idstring,)}
$FOLLOW(\langle RELATION \rangle)$ = {SELECT, SLIDING, ITERATIVE, SEQUENCE LIMITED BY #num, EVENT, SET,), AND, OR, AS, WHERE, HAVING, $}
$FOLLOW(\langle PRREL \rangle)$ = {(}
$FOLLOW(\langle INFREL \rangle)$ = {SELECT, SLIDING, ITERATIVE, SEQUENCE LIMITED BY #num, EVENT, SET,), AND, OR, AS, WHERE, HAVING, $}
$FOLLOW(\langle BINARY \rangle)$ = {COUNT, #idstring, #num, (}
$FOLLOW(\langle TERNARY1 \rangle)$ = {COUNT, #idstring, #num, (}
$FOLLOW(\langle TERNARY2 \rangle)$ = {COUNT, #idstring, #num, (}
$FOLLOW(\langle WHERE \rangle)$ = {HAVING, AS, $}
$FOLLOW(\langle HAVING \rangle)$ = {AS, $}
$FOLLOW(\langle SETBODY \rangle)$ = {$, AS, HAVING, WHERE}
$FOLLOW(\langle NEXTSETB \rangle)$ = {$, AS, HAVING, WHERE}
$FOLLOW(\langle ASEVENT \rangle)$ = {SELECT, ITERATIVE, SLIDING, SEQUENCE LIMITED BY #num, SET, EVENT, GAP, $}
$FOLLOW(\langle EVENT \rangle)$ = {AS}
$FOLLOW(\langle FILTER \rangle)$ = {AS}
$FOLLOW(\langle TSPEC \rangle)$ = {AND, SELECT, SLIDING, SET, SEQUENCE LIMITED BY #num, ITERATIVE, EVENT}
$FOLLOW(\langle SEQBODY \rangle)$ = {$, AS, HAVING, WHERE}
$FOLLOW(\langle NEXTSEQB \rangle)$ = {$, AS, HAVING, WHERE}
$FOLLOW(\langle GAPSPEC \rangle)$ = {SELECT, SLIDING, ITERATIVE, SEQUENCE LIMITED BY #num, EVENT, SET}

D Rozkladová tabulka pro analýzu shora dolů

	SELECT	SET	ITERATIVE	SLIDING	SEQUENCE ...	EVENT
$\langle S' \rangle$	5.1					
$\langle S \rangle$	5.2					
$\langle APPLINF \rangle$	5.14	5.14	5.14	5.14	5.14	5.14
$\langle NEXTCOMP \rangle$	5.19	5.19	5.19	5.19	5.19	5.19
$\langle INDEXED \rangle$	5.26	5.26	5.26	5.26	5.26	5.26
$\langle STREAM \rangle$		5.38	5.38	5.38	5.38	
$\langle STREAMX \rangle$		5.41	5.39	5.40	5.42	
$\langle SET \rangle$		5.43				
$\langle ANDOR \rangle$	5.67	5.67	5.67	5.67	5.67	5.67
$\langle SETBODY \rangle$	5.85	5.85	5.85	5.85	5.85	5.85
$\langle NEXTSETB \rangle$	5.86	5.86	5.86	5.86	5.86	5.86
$\langle ASEVENT \rangle$	5.88	5.88	5.88	5.88	5.88	5.88
$\langle EVENT \rangle$	5.89	5.90	5.90	5.90	5.90	5.91
$\langle SEQBODY \rangle$	5.101	5.101	5.101	5.101	5.101	5.101
$\langle NEXTSEQB \rangle$	5.103	5.103	5.103	5.103	5.103	5.103

Tabulka D.1: Rozkladová tabulka 1.

	NOT	FOR ALL ...	FOR SOME ...	SAME s...	SAME t...	SAME
$\langle CONSTR \rangle$	5.46	5.48	5.49	5.50	5.51	5.52
$\langle AFTERBR \rangle$	5.54	5.56	5.57	5.58	5.59	5.60
$\langle TAPPL' \rangle$		5.71				

Tabulka D.2: Rozkladová tabulka 2.

	DAYS	HRS	MINS	SECS	HOURS	MINUTES	SECONDS
$\langle TSPEC \rangle$	5.94	5.95	5.96	5.97	5.98	5.99	5.100

Tabulka D.3: Rozkladová tabulka 3.

	COUNT	(	#idstring	#num	,	+	.	FROM
⟨*PROJECTION*⟩	5.3	5.3	5.3	5.3				
⟨*NEXTPROJ*⟩					5.4			5.5
⟨*APPLICATION*⟩	5.6	5.8	5.7	5.9				
⟨*APPLICATION′*⟩	5.10		5.11	5.12				
⟨*APPLINF*⟩					5.14	5.13		
⟨*NEXT*⟩					5.15			
⟨*COMPLEXID*⟩			5.17					
⟨*NEXTCOMP*⟩					5.19	5.19	5.18	
⟨*APPLY*⟩	5.20		5.21					
⟨*SIMPLEID*⟩			5.22					
⟨*NEXTID*⟩							5.23	
⟨*INDEXED*⟩		5.25			5.26	5.26	5.26	
⟨*INDEX*⟩				5.27				
⟨*LINDEX*⟩					5.31			
⟨*LINDEX′*⟩				5.32				
⟨*CONSTANT*⟩				5.35				
⟨*INFFUNCTOR*⟩						5.36		
⟨*FUNCTOR*⟩	5.37							
⟨*CONSTR*⟩	5.53	5.47	5.53	5.53				
⟨*AFTERBR*⟩	5.62	5.55	5.62	5.62				
⟨*TAPPL*⟩	5.70	5.69						
⟨*TAPPL′*⟩						5.72		
⟨*RELATION*⟩	5.74		5.74	5.74				

Tabulka D.4: Rozkladová tabulka 4.

	)	=	BETWEEN	AND	OR	WHERE	HAVING	AS
⟨*APPLINF*⟩	5.14	5.14	5.14	5.14	5.14	5.14	5.14	5.14
⟨*NEXT*⟩	5.16							
⟨*NEXTCOMP*⟩	5.19	5.19	5.19	5.19	5.19	5.19	5.19	5.19
⟨*NEXTID*⟩	5.24							
⟨*INDEXED*⟩	5.26	5.26	5.26	5.26	5.26	5.26	5.26	5.26
⟨*LINDEX*⟩	5.30							
⟨*BRORNOT*⟩	5.63	5.64	5.64					
⟨*ANDOR*⟩	5.67			5.65	5.66	5.67	5.67	5.67
⟨*TAPPL′*⟩	5.71							
⟨*INFREL*⟩		5.76	5.77					
⟨*BINARY*⟩		5.78						
⟨*TERNARY*1⟩			5.79					
⟨*TERNARY*2⟩				5.80				
⟨*WHERE*⟩						5.81	5.82	5.82
⟨*HAVING*⟩							5.83	5.84
⟨*NEXTSETB*⟩						5.87	5.87	5.87
⟨*FILTER*⟩								5.93
⟨*NEXTSEQB*⟩						5.104	5.104	5.104
⟨*GAPSPEC*⟩			5.107					

Tabulka D.5: Rozkladová tabulka 5.

	timestamp	UP TO	STRICT BETWEEN	GAP
⟨*TAPPL*⟩	5.68			
⟨*NEXTSEQB*⟩				5.102
⟨*TAPPL′*⟩		5.105	5.106	

Tabulka D.6: Rozkladová tabulka 6.

	$	* − #num	*	UNTIL	WHILE	match	FILTERED BY
⟨*APPLINF*⟩	5.14						
⟨*NEXTCOMP*⟩	5.19						
⟨*INDEXED*⟩	5.26						
⟨*INDEX*⟩		5.28	5.29				
⟨*LINDEX′*⟩		5.33	5.34				
⟨*SETCONSTR*⟩				5.45	5.44		
⟨*CONSTR*⟩						5.53	
⟨*AFTERBR*⟩						5.61	
⟨*ANDOR*⟩	5.67						
⟨*RELATION*⟩						5.73	
⟨*PRREL*⟩						5.75	
⟨*WHERE*⟩	5.82						
⟨*HAVING*⟩	5.84						
⟨*NEXTSETB*⟩	5.87						
⟨*FILTER*⟩							5.92
⟨*NEXTSEQB*⟩	5.104						
⟨#⟩	pop						

Tabulka D.7: Rozkladová tabulka 7.

E Transformační pravidla

Níže jsou popsána transformační pravidla pro převod derivačního stromu gramatiky do podoby, která může být dále použita při překladu. Transformační pravidla je nutné aplikovat od listových uzlů stromu. Po aplikaci všech pravidel je nutné aplikovat dodatečná pravidla pro odstranění nepotřebných uzlů stromu – pokud má uzel obsahující neterminál pouze jednoho potomka, je tímto potomkem nahrazen (je odstraněn a jeho potomku je jako rodič přiřazen rodič právě odstraněného uzlu). Pokud nemá uzel obsahující neterminál žádného potomka, je odstraněn.

Při zápisu transformačních pravidel je využita následující syntaxe:

- Podstromy daného uzlu jsou uváděny za tímto uzlem, uzavřeny do hranatých závorek.
- Jednotlivé podstromy jsou odděleny svislou čárou.

Zápis $\langle * \rangle$ odpovídá proměnné *libovolný neterminál*. Stromy s neterminálem $\langle APPLICATION' \rangle$ v kořeni jsou transformovány podle stejných pravidel jako stromy, které mají v kořeni neterminální symbol $\langle APPLICATION \rangle$.

$[\langle * \rangle[\varepsilon]]$
$\longrightarrow [\langle * \rangle]$
$[\langle S' \rangle[\langle S \rangle]]$
$\longrightarrow [\langle S \rangle]$
$[\langle S \rangle$[SELECT$|\langle PROJECTION \rangle[\ldots]|$FROM$|\langle STREAM \rangle]]$
$\longrightarrow$ [FROM[SELECT$[\ldots]|\langle STREAM \rangle]]$
$[\langle PROJECTION \rangle[\langle APPLICATION \rangle|$AS #idstring$|\langle NEXTPROJ \rangle]]$
$\longrightarrow [\langle PROJECTION \rangle$[AS #idstring$[\langle APPLICATION \rangle]|\langle NEXTPROJ \rangle]]$
$[\langle NEXTPROJ \rangle[,|\langle APPLICATION \rangle|$AS #idstring$|\langle NEXTPROJ \rangle_1]]$
$\longrightarrow$ [AS #idstring$[\langle APPLICATION \rangle]|\langle NEXTPROJ \rangle_1]$
$[\langle APPLICATION \rangle[\langle FUNCTOR \rangle|(|\langle APPLICATION \rangle_1|\langle NEXT \rangle|)|\langle APPLINF \rangle]]$
$\longrightarrow [\langle APPLINF \rangle[\langle FUNCTOR \rangle$[()$[\langle APPLICATION \rangle_1|\langle NEXT \rangle]]]]$
$[\langle APPLICATION \rangle[\langle COMPLEXID \rangle|\langle APPLINF \rangle]]$
$\longrightarrow [\langle APPLINF \rangle[\langle COMPLEXID \rangle]]$
$[\langle APPLICATION \rangle[\langle CONSTANT \rangle|\langle APPLINF \rangle]]$
$\longrightarrow [\langle APPLINF \rangle[\langle CONSTANT \rangle]]$

[⟨APPLICATION⟩[(|⟨APPLICATION⟩$_1$|)|⟨APPLINF⟩]]
⟶ [⟨APPLINF⟩[()[⟨APPLICATION⟩$_1$]]]
[⟨APPLINF⟩[⟨INFFUNCTOR⟩|⟨APPLICATION⟩]]
⟶ [⟨INFFUNCTOR⟩[⟨APPLICATION⟩]]
[⟨NEXT⟩[,|⟨APPLICATION⟩|⟨NEXT⟩]]
⟶ [⟨APPLICATION⟩|⟨NEXT⟩]
[⟨COMPLEXID⟩[#idstring|⟨INDEXED⟩|⟨NEXTCOMP⟩]]
⟶ [⟨NEXTCOMP⟩[⟨INDEXED⟩[#idstring]]]
[⟨NEXTCOMP⟩[.|⟨APPLY⟩]]
⟶ [.[⟨APPLY⟩]]
[⟨APPLY⟩[⟨FUNCTOR⟩|(|⟨SIMPLEID⟩|)]]
⟶ [⟨FUNCTOR⟩[()[⟨SIMPLEID⟩]]]
[⟨APPLY⟩[#idstring|⟨INDEXED⟩|⟨NEXTCOMP⟩]]
⟶ [⟨NEXTCOMP⟩[⟨INDEXED⟩[#idstring]]]
[⟨SIMPLEID⟩[#idstring|⟨INDEXED⟩|⟨NEXTID⟩]]
⟶ [⟨NEXTID⟩[⟨INDEXED⟩[#idstring]]]
[⟨NEXTID⟩[.|#idstring|⟨INDEXED⟩|⟨NEXTCOMP⟩]]
⟶ [.[⟨NEXTID⟩[⟨INDEXED⟩[#idstring]]]]
[⟨INDEXED⟩[(|⟨INDEX⟩|)]]
⟶ [⟨INDEXED⟩[⟨INDEX⟩]]
[⟨INDEX⟩[#num|⟨LINDEX⟩]]
⟶ [⟨LINDEX⟩[#num]]
[⟨INDEX⟩[* - #num|⟨LINDEX⟩]]
⟶ [⟨LINDEX⟩[* - #num]]
[⟨LINDEX⟩[,|⟨LINDEX′⟩]]
⟶ [,[⟨LINDEX⟩]]
[⟨STREAMX⟩[ITERATIVE|⟨SET⟩]]
⟶ [ITERATIVE[⟨SET⟩]]
[⟨STREAMX⟩[SLIDING|⟨SET⟩]]
⟶ [SLIDING[⟨SET⟩]]
[⟨STREAMX⟩[SEQUENCE LIMITED BY #num|⟨TSPEC⟩|⟨SEQBODY⟩]]
⟶ [SEQUENCE[#num|⟨TSPEC⟩|⟨SEQBODY⟩]]
[⟨SET⟩[SET|⟨SETCONSTR⟩|⟨SETBODY⟩]]
⟶ [SET[⟨SETCONSTR⟩|⟨SETBODY⟩]]
[⟨SETCONSTR⟩[WHILE|⟨CONSTR⟩]]
⟶ [WHILE[⟨CONSTR⟩]]
[⟨SETCONSTR⟩[UNTIL|⟨CONSTR⟩]]
⟶ [UNTIL[⟨CONSTR⟩]]
[⟨CONSTR⟩[NOT|⟨CONSTR⟩$_1$]]
⟶ [NOT[⟨CONSTR⟩$_1$]]
[⟨CONSTR⟩[(|⟨AFTERBR⟩|⟨ANDOR⟩]]
⟶ [⟨ANDOR⟩[⟨AFTERBR⟩]]

[$\langle CONSTR \rangle$[FOR ALL #idstring|$\langle INDEXED \rangle$|$\langle CONSTR \rangle_1$]]
$\longrightarrow$ [ALL[$\langle INDEXED \rangle$[#idstring]|$\langle CONSTR \rangle_1$]]
[$\langle CONSTR \rangle$[FOR SOME #idstring|$\langle INDEXED \rangle$|$\langle CONSTR \rangle_1$]]
$\longrightarrow$ [SOME[$\langle INDEXED \rangle$[#idstring]|$\langle CONSTR \rangle_1$]]
[$\langle CONSTR \rangle$[SAME sources FOR ALL #idstring|$\langle INDEXED \rangle$|$\langle ANDOR \rangle$]]
$\longrightarrow$ [$\langle ANDOR \rangle$[SAME sources[$\langle INDEXED \rangle$[#idstring]]]]
[$\langle CONSTR \rangle$[SAME targets FOR ALL #idstring|$\langle INDEXED \rangle$|$\langle ANDOR \rangle$]]
$\longrightarrow$ [$\langle ANDOR \rangle$[SAME targets[$\langle INDEXED \rangle$[#idstring]]]]
[$\langle CONSTR \rangle$[SAME|$\langle TAPPL \rangle$|FOR ALL #idstring|$\langle INDEXED \rangle$|$\langle ANDOR \rangle$]]
$\longrightarrow$ [$\langle ANDOR \rangle$[SAME[$\langle TAPPL \rangle$|$\langle INDEXED \rangle$[#idstring]]]]
[$\langle CONSTR \rangle$[$\langle RELATION \rangle$|$\langle ANDOR \rangle$]]
$\longrightarrow$ [$\langle ANDOR \rangle$[$\langle RELATION \rangle$]]
[$\langle AFTERBR \rangle$[NOT|$\langle CONSTR \rangle$|)]]
$\longrightarrow$ [()[NOT[$\langle CONSTR \rangle$]]]
[$\langle AFTERBR \rangle$[(|$\langle AFTERBR \rangle$|$\langle ANDOR \rangle$|)]]
$\longrightarrow$ [()[$\langle ANDOR \rangle$[$\langle AFTERBR \rangle$]]]
[$\langle AFTERBR \rangle$[FOR ALL #idstring|$\langle INDEXED \rangle$|$\langle CONSTR \rangle$|)]]
$\longrightarrow$ [()[ALL[$\langle INDEXED \rangle$[#idstring]|$\langle CONSTR \rangle$]]]
[$\langle AFTERBR \rangle$[FOR SOME #idstring|$\langle INDEXED \rangle$|$\langle CONSTR \rangle$|)]]
$\longrightarrow$ [()[SOME[$\langle INDEXED \rangle$[#idstring]|$\langle CONSTR \rangle$]]]
[$\langle AFTREBR \rangle$[SAME sources FOR ALL #idstring|$\langle INDEXED \rangle$|$\langle ANDOR \rangle$|)]]
$\longrightarrow$ [()[$\langle ANDOR \rangle$[SAME sources[$\langle INDEXED \rangle$[#idstring]]]]]
[$\langle AFTERBR \rangle$[SAME targets FOR ALL #idstring|$\langle INDEXED \rangle$|$\langle ANDOR \rangle$|)]]
$\longrightarrow$ [()[$\langle ANDOR \rangle$[SAME targets[$\langle INDEXED \rangle$[#idstring]]]]]
[$\langle AFTERBR \rangle$[SAME|$\langle TAPPL \rangle$|FOR ALL #idstring|$\langle INDEXED \rangle$|$\langle ANDOR \rangle$|)]]
$\longrightarrow$ [()[$\langle ANDOR \rangle$[SAME[$\langle TAPPL \rangle$|$\langle INDEXED \rangle$[#idstring]]]]]
[$\langle AFTERBR \rangle$[$\langle PRREL \rangle$|(|$\langle APPLICATION \rangle$|$\langle NEXT \rangle$|)|$\langle ANDOR \rangle$]|)]
$\longrightarrow$ [()[$\langle ANDOR \rangle$[$\langle PRREL \rangle$[()[$\langle APPLICATION \rangle$|$\langle NEXT \rangle$]]]]]
[$\langle AFTERBR \rangle$[$\langle APPLICATION' \rangle$|$\langle BRORNOT \rangle$[)|$\langle INFREL \rangle$]|$\langle ANDOR \rangle$]]
$\longrightarrow$ [$\langle ANDOR \rangle$[$\langle INFREL \rangle$[()[$\langle APPLICATION' \rangle$]]]]
[$\langle AFTERBR \rangle$[$\langle APPLICATION' \rangle$|$\langle BRORNOT \rangle$[$\langle INFREL \rangle$|$\langle ANDOR \rangle_2$|)]|$\langle ANDOR \rangle_1$]]
$\longrightarrow$ [$\langle ANDOR \rangle_1$[()[$\langle ANDOR \rangle_2$[$\langle INFREL \rangle$[$\langle APPLICATION' \rangle$]]]]]
[$\langle ANDOR \rangle$[AND|$\langle CONSTR \rangle$]]
$\longrightarrow$ [AND[$\langle CONSTR \rangle$]]
[$\langle ANDOR \rangle$[OR|$\langle CONSTR \rangle$]]
$\longrightarrow$ [OR[$\langle CONSTR \rangle$]]
[$\langle TAPPL \rangle$[timestamp|$\langle TAPPL' \rangle$]]
$\longrightarrow$ [$\langle CONSTR \rangle$[timestamp]]
[$\langle TAPPL \rangle$[(|$\langle TAPPL \rangle_1$|)|$\langle TAPPL' \rangle$]]
$\longrightarrow$ [$\langle TAPPL' \rangle$[()[$\langle TAPPL \rangle_1$]]]

$[\langle TAPPL\rangle[\langle FUNCTOR\rangle|(|\langle TAPPL\rangle_1|)|\langle TAPPL'\rangle]]$
$\longrightarrow [\langle TAPPL'\rangle[\langle FUNCTOR\rangle[0[\langle TAPPL\rangle_1]]]]$
$[\langle TAPPL'\rangle[\langle INFFUNCTOR\rangle|\langle TAPPL\rangle]]$
$\longrightarrow [\langle INFFUNCTOR\rangle[\langle TAPPL\rangle]]$
$[\langle RELATION\rangle[\langle PRREL\rangle|(|\langle APPLICATION\rangle|\langle NEXT\rangle|)]]$
$\longrightarrow [\langle PRREL\rangle[0[\langle APPLICATION\rangle|\langle NEXT\rangle]]]$
$[\langle RELATION\rangle[\langle APPLICATION'\rangle|\langle INFREL\rangle]]$
$\longrightarrow [\langle INFREL\rangle[\langle APPLICATION'\rangle]]$
$[\langle INFREL\rangle[\langle BINARY\rangle|\langle APPLICATION\rangle]]$
$\longrightarrow [\langle BINARY\rangle[\langle APPLICATION\rangle]]$
$[\langle INFREL\rangle[\langle TERNARY1\rangle|\langle APPLICATION\rangle_1|\langle TERNARY2\rangle|\langle APPLICATION\rangle_2]]$
$\longrightarrow [\langle TERNARY1\rangle.\langle TERNARY2\rangle[\langle APPLICATION\rangle_1|\langle APPLICATION\rangle_2]]$
$[\langle WHERE\rangle$[WHERE|$\langle CONSTR\rangle$]]
$\longrightarrow$ [WHERE[$\langle CONSTR\rangle$]]
$[\langle HAVING\rangle$[HAVING|$\langle CONSTR\rangle$]]
$\longrightarrow$ [HAVING[$\langle CONSTR\rangle$]]
$[\langle ASEVENT\rangle[\langle EVENT\rangle|$AS #idstring]]
$\longrightarrow$ [AS #idstring[$\langle EVENT\rangle$]]
$[\langle NEXTSETB\rangle[\langle ASEVENT\rangle|\langle NEXTSETB\rangle]]$
$\longrightarrow [\langle ASEVENT\rangle|\langle NEXTSETB\rangle]$
$[\langle EVENT\rangle$[EVENT|$\langle FILTER\rangle$]]
$\longrightarrow$ [EVENT[$\langle FILTER\rangle$]]
$[\langle FILTER\rangle$[FILTERED BY|$\langle CONSTR\rangle$]]
$\longrightarrow$ [FILTERED BY[$\langle CONSTR\rangle$]]
$[\langle NEXTSEQB\rangle[\langle ASEVENT\rangle|\langle NEXTSEQB\rangle]]$
$\longrightarrow [\langle ASEVENT\rangle|\langle NEXTSEQB\rangle]$
$[\langle NEXTSEQB\rangle$[GAP|$\langle GAPSPEC\rangle|\langle ASEVENT\rangle|\langle NEXTSEQB\rangle$]]
$\longrightarrow$ [GAP[$\langle GAPSPEC\rangle$]|$\langle ASEVENT\rangle|\langle NEXTSEQB\rangle$]
$[\langle GAPSPEC\rangle$[UP TO #num|$\langle TSPEC\rangle$]]
$\longrightarrow$ [UP TO[#num|$\langle TSPEC\rangle$]]
$[\langle GAPSPEC\rangle$[BETWEEN #num|$\langle TSPEC\rangle_1$|AND #num|$\langle TSPEC\rangle_2$]]
$\longrightarrow$ [BETWEEN[#num|$\langle TSPEC\rangle_1$|#num|$\langle TSPEC\rangle_2$]]
$[\langle GAPSPEC\rangle$[STRICT BETWEEN #num|$\langle TSPEC\rangle_1$|AND #num|$\langle TSPEC\rangle_2$]]
$\longrightarrow$ [STRICT[#num|$\langle TSPEC\rangle_1$|#num|$\langle TSPEC\rangle_2$]]

F Možnosti rozšíření dotazovacího jazyka

V této části textu jsou ve stručnosti popsány náležitosti, které musí jednotlivé části rozšíření splňovat. V některých odstavcích mohou být obsaženy duplicitní informace vzhledem k předchozímu textu.

F.1 Datové typy

Pro definici datového typu je nutné vytvořit třídu, která tento datový typ implementuje (nejlépe v balíčku *DAQL.DataTypes*). Aby byla třída datového typu rozpoznána, je nutné provést jeho registraci s využitím metody *Register* třídy *Manager* balíčku tříd *DAQL.DataTypes*. Tato metoda očekává jako první argument celé jméno třídy (tj. včetně jména balíčku), dále jméno datového typu a jako nepovinný argument jméno datového typu, který je tímto rozšiřován (například datový typ popisující celá čísla může být považován za rozšíření datového typu popisujícího přirozená čísla).

Požadované rozhraní:

Metoda *Validate* Metoda, která na vstupu přijímá libovolný řetězec a vrací pravdivostní hodnotu podle příslušnosti zadaného řetězce k danému datovému typu.

Metoda *FunctionList* Tato metoda má jeden nepovinný argument. V případě, že je volána bez argumentu, vrátí seznam všech funkcí, která lze na tento datový typ aplikovat. Funkce jsou zapsány jako řetězec `jméno##I/P##DT##DT1##...##DTn`, ve kterém `jméno` představuje zápis funkce, `I/P` určuje způsob zápisu funkce (je možné `P`, `I`, `PI` i `IP`), `DT` představuje datový typ návratové hodnoty funkce a `DT1` až `DTn` představují datové typy jednotlivých argumentů. V zápisu jména funkce není povoleno použití diakritiky. Není možné, aby měla funkce infixový zápis v případě, že není binární.

Jako argument je možné metodě zadat jméno funkce. Pokud je tato funkce povolena, vrátí metoda jednoprvkový seznam obsahující popis funkce. Jinak vrátí prázdný seznam.

Metoda *AggregateList* Tato metoda má podobnou funkcionalitu jako předchozí, pouze s tím rozdílem, že vypisuje pouze agregační funkce. Agregační funkce smí mít definován pouze jeden datový typ argumentů a jméno může být tvořeno pouze písmeny.

Metoda *Apply* Metoda přijímá jako argumenty jméno funkce a seznam atributů, návratovou hodnotou je výsledek aplikace dané funkce.

Metoda *RelationList* Tato metoda má podobné chování jako metoda *FunctionList*, pouze s tím rozdílem, že místo popisu funkcí vrací popis povolených relací. Zápis popisující relaci je podobný jako v případě funkcí, musí ale zohlednit povolení infixového zápisu ternárních relací. Řetězec, kterým je relace popsána, vypadá podobně jako řetězec popisující funkci a to následovně: `jméno##P/B/T##jméno2##DT1##...##DTn`, řetězec `jméno2` je povinný pouze v případě definice ternární relace zapisované infixově (volba `T`). Nejsou povoleny kombinace voleb `P`, `B` a `T`.

Metoda *Decide* Metoda poskytuje podobnou funkcionalitu, jako metoda *Apply*, pro jméno relace seznam argumentů vrací pravdivostní hodnotu podle příslušnosti do relace.

Metoda *Attributes* Metoda nevyžaduje žádný argument, pouze vypisuje jména všech atributů, které mohou nabývat hodnot z daného datového typu. Implementace metody není povinná.

F.2 Přiřazení datových typů

Pro správnou kontrolu zpracovávaných událostí je nutné definovat přiřazení datových typů použitých pro identifikaci původců, zdrojů a pro zápis časové známky. K přiřazení je možné využít metodu *Assign* třídy *Manager* balíčku *DAQL.DataTypes*. Metoda přijímá dva argumenty, prvním je jméno datového pole popisujícího událost (například *sources*), druhým je jméno přiřazeného datového typu. Není možné přiřazení více datových typů jednomu poli, při opakovaném pokusu o přiřazení je vždy předchozí záznam smazán.

Pomocí tohoto způsobu je možné definovat také povinné atributy, které musí mít také přiřazen datový typ.

F.3 Makro příkazy pro filtry

Pojmenované filtry použité v dotazech lze předat několika způsoby. Prvním ze způsobů je vložení souboru, který obsahuje záznamy `X: EVENT FILTERED BY code = 'X'`. Tyto záznamy musí být ukončeny symbolem tečka. K předání pomocí souboru je nutné použít metodu *ImportFile* třídy *Macro* balíčku *DAQL.Extension*. Tato metoda přijímá jako argument plnou cestu k danému souboru. Při nahrání ze souboru jsou přepsány změněné pojmenované filtry, ale filtry, které se v souboru nevyskytují nejsou smazány.

Dalším způsobem je volání metody *Add* též třídy. Metoda jako první argument požaduje zápis jména pojmenovaného filtru (např. `X`), jako druhý argument příslušnou podmínku (zde `code = 'X'`).

Jména pojmenovaného filtru musí být v obou případech tvořena pouze písmeny.

Třída *Macro* dále poskytuje metodu *DeleteAll*, která smaže všechny pojmenované filtry, a metodu *Delete*, která smaže filtr, jehož jméno je jí předáno jako argument.

F.4 Rozhraní pro přístup k událostem

Jako rozhraní pro přístup k událostem je požadována třída *Connector* balíčku *DAQL.Manager*. Tato třída musí umožňovat převod mezi formátem událostí použitém v daném systému a formátem událostí použitém při zpracování dotazů. Pro odesílání událostí do daného systému musí poskytovat metodu *Output*, která jako argument přijímá objekt události.

G Doplňující dotazy jazyka DAQL

Dotaz G.1 Použití funkce *COUNT(*)* v rámci podmínky ukončení (2.)

```
1: SET WHILE
2:     COUNT(*) = 5
3:     AND
4:     COUNT(udalost1) > 0
5:     AND
6:     COUNT(udalost2) > 0
7:     AND
8:     COUNT(udalost3) > 0
9:     EVENT
10:        FILTERED BY
11:            code = 'X'
12:    AS udalost1
13:    EVENT
14:        FILTERED BY
15:            code = 'Y'
16:    AS udalost2
17:    EVENT
18:        FILTERED BY
19:            code = 'Z'
20:    AS udalost3
```

Dotaz G.2 Zjištění infekce virem Morto (bez použití *FILTERED BY*)

```
 1: SELECT
 2:     MIN(timestamp) AS timestamp,
 3:     first_scan.sources AS sources,
 4:     dns_set.sources AS targets,
 5:     'MORTO' AS code,
 6:     'The target device was infected by MORTO malware.' AS template
 7: FROM
 8:     SEQUENCE LIMITED BY 8 HOURS
 9:         EVENT
10:         AS first_scan
11:         GAP UP TO 6 HOURS
12:         SET UNTIL
13:             (COUNT(EVENTS) > 12
14:             AND
15:             COUNT(DISTINCT(targets)) > 5)
16:             EVENT
17:             AS dns_anomaly
18:         WHERE
19:             dns_anomaly.code = 'DNSANOMALY'
20:             AND
21:             type = 'unexpected'
22:             AND
23:             COUNT(sources) = 1
24:             AND
25:             FOR ALL EVENTS SAME sources
26:         AS dns_set
27:         GAP UP TO 30 MINUTES
28:         EVENT
29:         AS second_scan
30:     WHERE
31:         first_scan.code = 'SCANS'
32:         AND
33:         port = 3389
34:         AND
35:         second_scan.code = 'SCANS'
36:         AND
37:         port = 3389
38:         AND
39:         dns_set.sources IN first_scan.targets
40:         AND
41:         dns_set.sources = second_scan.sources
```