Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Znalostní systémy a jejich aplikace

Luděk Letáček

Bakalářská práce
2021

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: 2020/2021

ZADÁNÍ BAKALÁŘSKÉ PRÁCE
(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Luděk Letáček**
Osobní číslo:	**I18158**
Studijní program:	**B2646 Informační technologie**
Studijní obor:	**Informační technologie**
Téma práce:	**Znalostní systémy a jejich aplikace**
Zadávající katedra:	**Katedra informačních technologií**

Zásady pro vypracování

Práce by měla v teoretické části popsat znalostní systémy včetně součástí a jejich obecné vlastnosti. Dále je možné rozšíření o neurčitost. Teoretické znalosti by měly být aplikovány na konkrétní případ, například z ekonomie. Znalostní systém by měl být implementován v některém z vyšších programovacích jazyků (například JAVA). Součástí implementace budou konkrétní výstupy pro aplikovaný příklad, včetně grafických, a uživatelská příručka.

Rozsah pracovní zprávy:
Rozsah grafických prací:
Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam doporučené literatury:

PROVAZNÍK, I.; KOZUMPLÍK, J. Expertní systémy – skriptum. Brno: VUT v Brně, 1999. p. 1-100. ISBN: 80-214-1486-3.
NAVARA, M., OLŠÁK. P. Základy fuzzy množin. 2. přeprac. vyd. Praha: Nakladatelství ČVUT, 2007, 150 s. ISBN 978-80-01-03668-6.
DVOŘÁK, J. Expertní systémy. Elektronický učební text. VUT v Brně, 2004

Vedoucí bakalářské práce: **Mgr. Alena Pozdílková, Ph.D.**
Katedra matematiky a fyziky

Datum zadání bakalářské práce: **31. října 2020**
Termín odevzdání bakalářské práce: **14. května 2021**

L.S.

Ing. Zdeněk Němec, Ph.D. v.r.
děkan

Ing. Jan Panuš, Ph.D. v.r.
vedoucí katedry

V Pardubicích dne 26. února 2021

Prohlašuji:

Tuto práci jsem vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Beru na vědomí, že v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů, a směrnicí Univerzity Pardubice č. 9/2012, bude práce zveřejněna v Univerzitní knihovně a prostřednictvím Digitální knihovny Univerzity Pardubice.

V Pardubicích dne 1. 5. 2021

Luděk Letáček v. r.

PODĚKOVÁNÍ

Rád bych poděkoval své vedoucí bakalářské práce Mgr. Aleně Pozdílkové, Ph.D. za odborné vedení, trpělivost, doporučení a cenné rady. Dále bych touto cestou poděkoval také své rodině a přátelům, kteří mě během studia podporovali a bez kterých bych tuto práci nemohl dokončit.

ANOTACE

Bakalářská práce se zaměřuje na popis znalostních/expertních systémů a podrobnou zprávu konkrétní aplikace.

Teoretická část se bude v prvních kapitolách zabývat charakteristikou, architekturou a typy znalostních/expertních systémů. Budou popsány moduly, ze kterých se systém skládá, jejich účel a propojení mezi nimi. Další kapitola se bude věnovat možnosti pro zpracování neurčitosti v těchto systémech.

V praktické části je popsána implementace grafické aplikace, napsané v programovacím jazyku Java, která řeší problematiku zařazení do evidence uchazečů o zaměstnání a podpory v nezaměstnanosti dle zákona č. 435/2004 Sb., o zaměstnanosti.

KLÍČOVÁ SLOVA

Znalostní systémy, expertní systémy, báze znalostí, inferenční mechanismus, vysvětlovací modul, neurčitost, nezaměstnanost, uchazeč o zaměstnání, evidence uchazečů o zaměstnání, podpora v nezaměstnanosti.

TITLE

Knowledge systems and their applications

ANNOTATION

This bachelor’s thesis focuses on descriptions of knowledge-based/expert systems and detailed documentation of a specific application.

The first chapters of the theoretical part cover characteristics, architecture, and types of knowledge-based/expert systems. There are also described purposes and relations between their components. The next chapter focuses on the possibilities of processing to handle uncertainties.

The practical part describes the implementation of a graphical application, written in the Java programming language, which addresses the issue of inclusion in the register of job seekers and unemployment benefits under Act No. 435/2004 Coll.

KEYWORDS

Knowledge-based systems, expert systems, knowledge base, inference engine, explanation module, uncertainties, unemployment, job applicant, records of work-seekers, unemployment benefits.

OBSAH

SEZNAM ILUSTRACÍ A TABULEK

SEZNAM ZKRATEK

ZS	Znalostní systém
UI	Umělá inteligence
BZ	Báze znalostí
IM	Inferenční mechanismus
ISA	is a (typ závislosti sémantických sítí)
AKO	a kind of (typ závislosti sémantických sítí)
ISPART	is a part of (typ závislosti sémantických sítí)

ÚVOD

Téma mé bakalářské práce jsem si vybral z důvodu, že se jedná o jednu z nejúspěšnějších aplikací umělé inteligence. Znalostní systém je podle staršího významu obecnější typ expertních systémů, ale v dnešní době jsou již brány oba pojmy jako synonyma. Od prvního předvedení v 80. letech minulého století zažily rozmach a nyní jsou používány v mnoha oblastech, kde je potřeba simulovat rozhodování složitých úloh podle podrobně a přímo vyjádřených znalostí či úsudků získaných od experta. [5]

Cílem je vytvořit znalostní systém, který bude schopen vkládat, odstraňovat a editovat jednotlivé znalosti a vybírat mezi různými znalostními bázemi. Znalosti tohoto systému budou aplikovány na problematiku týkající se rozhodování o zařazení do evidence uchazečů o zaměstnání. V případě vzniknuvšího nároku na podporu v nezaměstnanosti systém vypočítá její výši dle zákona o zaměstnanosti č. 435/2004 Sb. Systém bude rozhodovat na základě jím položených otázek, na které uživatel odpoví.

Práce je rozdělena na tři díly. První díl se zabývá teorií znalostních/expertních systémů. V této části se první kapitola věnuje charakteristice, druhá typy těchto systémů podle charakteristiky řešených úloh, třetí dává konkrétní příklad významných znalostních systémů, čtvrtá až šestá jejich strukturou a sedmá neurčitostí znalostních systémů. Druhý díl popisuje vytvořenou aplikaci. Zde se nejdříve zabývá analýzou a požadavky na systém, poté se zaobírá návrhem aplikace a poslední část je zaměřena na způsob její implementace. Třetí díl je zaměřen na popis řešené problematiky a ukázky vyhodnocení aplikace podle možných scénářů.

1 ZNALOSTNÍ SYSTÉMY

Umělá inteligence (UI) je oblast, která se používá v mnoha vědních oborech zahrnující například neurologii, filozofii, psychologii, počítačové vědy, robotiku a lingvistiku. Je zaměřena na napodobování lidského chování a myšlení. Před zájmem o komerční aplikace se používaly pouze v rámci výzkumných center. V dnešní době do populárních oblastí UI patří především umělé neuronové sítě, genetické algoritmy, znalostní/expertní systémy a mnoho dalšího. [1]

První použití znalostního/expertního systému bylo na přelomu 70. a 80. let minulého století. Patří mezi úspěšné a nejdéle nasazené nástroje UI. Bývá nazýván jako znalostní systém, expertní systém či expertní znalostní systém. Tato pojmenování se v tomto případě berou jako synonyma. Dále budeme používat pojmenování znalostní systém (ZS), byť je častější výraz expertní systém, který bývá označován také jako systém obsahující pouze obecně známé informace. ZS simuluje[1] rozhodování expertů při řešení úloh, které jsou úzce zaměřené na konkrétní problém. Znalosti jsou tvořeny z vědomostí nebo zkušeností expertů, ale také obecně z dostupných knih, časopisů apod. Expertem označujeme takového člověka, který je specializovaný v určité oblasti, která není známá či veřejně dostupná, a je v ní schopen řešit problémy efektivně. Oblasti řešených problémů bývají více či méně specializované. V praxi bývá běžné, že aplikace ZS řeší více typů problémů. [1], [4]

Na začátku 90. let 20. století, kdy se zájem o ZS výrazně zvýšil, došlo ke zvýšenému zájmu jejich používání v marketingu, účetnictví nebo financí. Posléze se začaly rozšiřovat i do plánování, výroby a dalších oblastí. Po tomto nárůstu se však obecně začaly omezovat na obecnou správu a provoz, což bylo způsobeno také hlavně tím, že zájem pocházel z oblastí se snadno strukturovatelnými problémy. ZS musí mít velmi dobře definovanou problémovou doménu, aby mohl správně fungovat. Nejoblíbenější odvětví pro aplikace znalostních systémů jsou taková průmyslová odvětví, která rychle prozkoumávají a přijímají nové technologie. Jedná se například o již zmíněné účetnictví, výrobu a finanční služby, ale i medicínu. [6]

Obecně lze říct, že *analytické* problémy zahrnují rozdělení na dílčí problémy. Díky tomu se jedná o lépe strukturované problémy. Jejich opakem jsou problémy typu *syntéza*, které jsou tvořeny pomocí různých kombinací zdrojů, čímž se stávají nestrukturovanějšími. Monitorování nebo predikce kombinují aspekty analytických i syntetických problémových domén. Podle [6] na základě zpráv vývojářů ZS vyplývá, že systémy, které řeší takový typ problémů, kvůli

[1] Termín simulovat je v tomto případě schopnost systému napodobit chování a rozhodování experta.

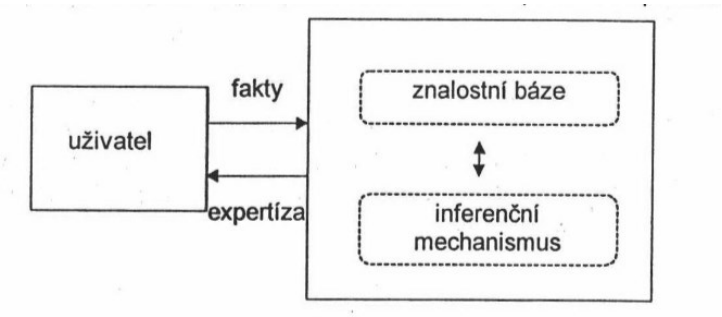
kterým byl jejich vývoj obtížnější, mají obvykle daleko větší vliv. Vzhledem k popularitě takových aplikací jako je Mycin (známý znalostní systém pro diagnostiku infekčních nemocí)[2], není překvapující, že i nadále dominují diagnostické aplikace a je značný zájem o navrhování znalostních systémů pro plánování či monitorování. [6]

1.1 Charakteristiky znalostních systémů

ZS pracují na principu, kdy uživatel poskytuje různá fakta a jako odezvu od systému dostane expertní radu. Současné ZS jsou schopné vysvětlit proces vlastního usuzování a tím i výsledný výrok. Je-li uživatelem expert, systém slouží jako opora, která může pomoci k efektivnějšímu nalezení řešení problému. K vyhodnocování se zahrnuje rozhodování, plánování, klasifikace atp. Pro definování problému v jazyce ZS není nutné mít vědomosti o tom, jak se programuje v nízko úrovňových jazycích[3]. Vytváří se pouze jednoduchá pravidla typu „co se má stát za jakých podmínek“. [1]

ZS je rozdělen na několik částí. Nejdůležitější části jsou *báze znalostí* obsahující znalosti a *inferenční mechanismus*, který se stará o jejich zpracování. Dále se člení na *vysvětlovací modul*, který popisuje a vysvětluje postup usuzování, *modul získávání znalostí*, skrze který se vkládají nové znalosti do systému, a *I/O rozhraní*, které je reprezentováno nejčastěji jako dialogový režim, kde na dotaz systému odpovídá uživatel. Tím, že jsou odděleny znalosti od mechanismu jejich využívání se odlišuje ZS od klasických programů. Charakteristikou ZS je možnost rozhodování za neurčitosti a schopnost vysvětlování. [2], [5]

Nejnovější ZS nepracují pouze s jednou bází znalostí, ale využívá jich současně pro vyhodnocování hned několik. Mluvíme pak o tzv. *zdrojích znalostí*. Závěry z jednotlivých bází se sdílí za pomocí datové struktury pro zbývající zdroje znalostí. [4]



Obrázek 1: Základní koncept znalostního systému [1]

[2] Viz kapitola 1.3.
[3] Například programovací jazyk C.

1.2 Typy znalostních systémů

Na rozdíl od klasických programů, kde řešení problémů jsou typicky algoritmické, tak ZS mají problémy s možným symbolickým řešením. Znalostní systémy můžeme dělit dle různých aspektů. Například je můžeme rozdělovat podle charakteristik řešených úloh. Mezi hlavní třídy, které se vyskytují ve známých, dnes již překonaných ZS (viz kapitola 1.3), patří: *diagnóza*, *interpretace*, *monitorování*, *plánování*, *návrh* a *predikce*. [1], [2]

Diagnóza má za cíl nalézt chyby v systémech živých i neživých. Jedná se o obyčejné odvození hypotéz na základě faktů. Problémy této třídy jsou v možné odchylné činnosti, nepřístupnosti některých dat, příznaky dysfunkcí překrývající jiné nebo že jeho chování nemusí být zcela známo. Interpretace je zaměřena na analýzu dat pro stanovení jejich významu. Zde se hlavně potýká s potížemi s daty tím, že například obsahují šum, jsou cizorodá, můžou také chybět či obsahovat chybu. Monitorování je příklad určování i vykládání dat a signálů v reálném čase. Následně se vyhodnocuje, kdy by mělo dojít k zásahu. Přijatá data se porovnávají s očekávanými. U tohoto typu úloh může docházet k poplašným alarmům. Další z výše vyjmenovaných tříd je plánování, které navrhuje posloupnost akcí pro řešení daného problému. Některé řešené problémy mohou být špatně strukturované nebo komplikované, což může vést k nepochopení možných důsledků nalezených akcí uživatelem. Návrh určuje, které konfigurace by vyhovovaly zadaným podmínkám. Stejně jako u plánování se zde vyskytují potíže se strukturou řešeného problému. Predikce předpovídá na základě minulosti a přítomnosti budoucí události. Tedy je zde vyžadováno pro vyhodnocení pozorování v čase. [1], [2]

Při prvotním dělení si vystačíme pouze se dvěma základními typy. Jsou to diagnostické ZS a generativní ZS. Diagnostické systémy obsahují z výše zmíněných skupin diagnózu, interpretaci a monitorování. Vybírá svá doporučení z pevné množiny závěrů (hypotéz). Jejich vývoj je poměrně jednodušší než u generických systémů a obsahují problémy řešené obvykle lidmi. To jsou také hlavní důvody, proč tento typ ZS převládá. Generativní systémy jsou skupinou, která je tvořena z plánování, návrhu a predikce. Své cíle si dynamicky generuje během odvozování. [2]

Diagnostické úlohy jsou určeny pro hledání, potvrzování či vyvracení přítomnosti závad v systému. Pro nalezení diagnózy se využívá empirický cyklus, který je složen z postupných na sebe navazujících kroků: formulování hypotézy, její testování a její následné přijetí a odmítnutí. Konceptuální model této úlohy využívají různé druhy znalostí. Tyto znalosti mohou mít povahu popisu chování normálního systému, abnormálního systému, seznamu příznaků určující normální situaci nebo seznamu závad a k nim přiřazeným výčet příznaků. V praxi se mohou

mezi sebou mísit. Při uvažování o konceptuálních modelech rozlišujeme dle způsobu diagnostikování: zjišťování odchylky od normálu, porovnáním abnormálního chování a klasifikováním abnormality. Diagnostikování odchylky od normálu využívá z výše zmíněných typů znalostí *popis normálního systému* a *seznam příznaků určující normální situaci*. Využívá se v případech, kdy nemáme dostatečné množství znalostí o abnormální činnosti systému, například k diagnostice neodzkoušeného stroje. Porovnává se normální (očekávaný) stav s pozorovaným. K porovnání abnormálního chování se využívá modelu abnormálního chování systému. Je-li v systému nějaká chyba, potom se dají předpovídat výskyty abnormalit, které se porovnávají se zpozorovanými. Diagnostiky zjišťování odchylky a porovnávání s abnormálním chováním jsou nazývané jako systémy založené na modelech (využívají chování a struktury systému). Klasifikování abnormality využívají *seznam příznaků pro jednotlivé závady* a někdy i *seznam příznaků pro normální situaci*. Tato diagnostika se od výše zmíněných liší tím, že se s modelem systému nepracuje a při řešení se využívá heuristických znalostí. Při hledání řešení se nalézají závady, které jsou spojeny s pozorovanými příznaky. Tento způsob je někdy nazýván jako heuristická klasifikace. [2]

Generativní úlohy a celkově plánování se v UI postupně vyvíjelo a zároveň se stalo její oblastí zájmu. Cílem je vytvořit takové akce, které povedou k nalezení hledaného řešení. Dnešní pojem plánování představuje plnění různých restrikcí pro požadovaný cíl. Za předpokladu, že cíl podmínky nesplňuje, pak je nutné ho modifikovat. Takto jsou například řešeny úlohy pro konfiguraci nějakého systému. Jsou-li k dispozici potřebné znalosti pro určení činnosti, pak se postupuje stylem *navrhni a aplikuj*, kde se postupuje relativně přímočaře díky rozšiřování jednotlivých řešení. Pokud máme více suboptimálních řešení nebo jsou znalosti neúplné, postupuje se stylem *navrhni a reviduj*, díky kterému je možnost vrátit se zpět k původním variantám řešení. Řešení je složeno z jednotlivých složek, představující například akce při tvorbě plánu. [2]

1.3 Příklady znalostních systémů

Mycin je program, který byl vyvinut v 70. letech minulého století ve Stanfordu E. Shortliffem. Jedná se o ZS, který skrze dialog s uživatelem (lékařem) zjišťuje pomocí otázek, jaký je pacientův zdravotní stav, zdravotní historie a jaké jsou jeho výsledky z laboratorních testů. Pomocí těchto odpovědí se určí diagnóza. Následně podle pacientových proporcí, jako je jeho výška či váha, a případných alergií doporučí Mycin vhodný lék i s dávkováním. Znalosti jsou uloženy jako podmínky, ze kterých se usuzuje závěr s nějakou danou jistotou. Každá podmínka a závěr jsou tvořeny z atributu, objektu a hodnoty. Systém využíval neurčitostí vyjádřenou

faktory nejistoty. Vyhodnocoval směrem od diagnóz, které měly být potvrzeny, k dotazům o pacientovi. Jedná se o takzvané zpětné řetězení (viz kapitola 1.5.4). Tento ZS byl vzorem pro mnoho dalších systémů. Na jeho principu práce s neurčitostí navázaly i české systémy, například Equant nebo Sak. Později byly znalosti z tohoto systému odstraněny a vznikl tak prázdný znalostní systém s názvem EMycin, který byl použitý i pro jiné aplikace. [2]

Teiresias je systém, skrze který mohl expert upravovat bázi znalostí systému Mycin a to v případě, kdy systém odvodil špatnou diagnózu. Teiresias umožňuje trasování pravidel, která byla použita k odvození nesprávné diagnózy, a když bylo nalezeno závadné pravidlo nebo bylo potřeba o pravidlo systém rozšířit, tak se umožnilo změnit bázi. Systém nepracuje se znalostmi, ale s metaznalostmi, které uchovávají způsob chování systému Mycin a informace o její bázi znalostí. Díky tomu se expert například upozornil, chyběl-li mu u nově vytvářené znalosti v systému Mycin některý z nezbytných předpokladů dle její charakteristiky. [2]

Prospector je jeden z komerčně nejúspěšnějších systémů, byl vyvinut v 80. letech minulého století a spadá do kategorie diagnostických systémů. Oblast zájmu tohoto systému je rudná geologie. Z rozborů zkušebních vrtů a ze zkoumání terénu systém zjišťuje, jestli je na daném místě pravděpodobnost výskytu rudného ložiska. Podle typu naleziště se vybírá báze znalostí. Znalosti systému jsou reprezentovány pomocí pravidel. Prospektor využívá Bayesovský způsob pro jeho prací s neurčitostí v inferenčním mechanismu. Stejně jako Mycin tak i tento ZS byl předlohou pro různé systémy (například pro systém AL/X nebo český systém FEL-Expert). [2]

R1/Xcon, stejně jako Prospector patří mezi komerčně nejvýznamnější a nejúspěšnější systémy. Díky R1/Xcon vzrostl zájem u komerční sféry o UI. Na rozdíl od systému Prospector je typu generativního systému. Systém začal vyvíjet J. McDermott ke konci 70. let 20. století. Oblast systému byla zaměřena na konfiguraci sálových počítačů VAX firmy Digital Equipment. Podle různých specifikací na procesor, paměť, disky atp. se nakonfigurovala vhodná sestava včetně kabeláže. Nejprve byl (prototyp vytvořený v roce 1980) schopný nakonfigurovat jeden typ počítače a posléze byl rozšířen o desítky modelů, které byly tvořeny stovkou komponentů. Tento systém je výborným příkladem, že se musí po celou dobu existence systému aktualizovat jeho bázi znalostí. [2]

Další způsob využití znalostních systémů s jejich srozumitelnými a strukturovanými znalostmi je například ve výuce. Příklad takového systému je Internist, který pokrývá velkou část interní medicíny a jeho verze s názvem Caduceus je využívána pro výuku studentů medicíny. [2]

1.4 Báze znalostí

Proces rozhodování je tvořen informacemi, které jsou množinou vybraných nashromážděných dat, a znalostmi experta, které získal zkušeností a vzděláním. Znalosti a data jsou z hlediska ZS brány jako komplementární pojmy, pro které je typické, že se od sebe liší. Správnost dat je možno ověřit srovnáním s opakovaným pozorováním. Bývají získávány automatickým procesem nebo úředníkem, obsahují mnoho detailů a často se mění. Znalosti oproti tomu obsahují abstrakce, zobecnění a moc často se nemění. S novými daty v ZS může vzniknout rozpor se znalostmi, a to na základě chybných údajů nebo neúplností. Tato situace se může řešit buď revizí znalostí nebo faktory nejistoty. [2]

Giarratano a Riley vytvořili hierarchii pojmů znalostí. Na nejnižší pozici se nachází *šum*, představující nezajímavá data bez významu. Na vyšší úrovni jsou *data* s potencionálně významovou hodnotou. Nad nimi jsou *informace*, která jsou chápána jako zpracovaná data. Další úroveň obsahuje *znalosti*, které představují specializované informace na určitou oblast (získané např. generalizací informací) a nejvýše postavené jsou *metaznalosti* neboli znalosti o tom, jak znalosti využívat (znalosti o znalostech). [1], [2]

Báze znalostí (BZ) je tvořena znalostmi a metaznalostmi z konkrétní oblasti zkoumání (domény). Znalosti mohou být obecné i vysoce specifické nebo vědecky prokázané či nejisté heuristiky. Dále se klasifikují na mělké (*shallow knowledge*) a hluboké (*deep knowledge*). Mělké nalezneme ve většině ZS. Staví na heuristických a empirických znalostech. Jedná se o způsob vyhodnocování, kdy se z faktů něco usuzuje (např. když osoba nadměrně pije, z toho můžeme usuzovat, že má cukrovku). Hluboké znalosti mají příčinnou závislost, to znamená, že se určuje, k čemu fakta vedou (např. osoba má zvýšenou hladinu cukru v krvi, to vede k vylučování cukru pomocí močové soustavy, a to vede ke zvýšení příjmu tekutin). Znalosti jsou reprezentovány například pravidly, sémantickými sítěmi, objekty, matematickou logikou nebo rozhodovacími stromy. Mohou být díky dnešním nástrojům tvořeny i z více způsobů, pak hovoříme o tzv. hybridních schématech. Nejvíce se setkává s kombinací pravidel a rámců. Obecně je na komplikované systémy (jako například pro plánovací znalostní systémy) potřeba komplexnějších nástrojů. [2], [5]

1.4.1 Pravidla

Nejpoužívanějším způsobem z výše uvedených způsobů reprezentace jsou *pravidla*, která mají výhodu srozumitelnosti, jednoduchosti a snadné aktualizace BZ. Jejich nevýhodou je, že neumožňují vyjádřit strukturální znalost. Jedná se o strukturu IF-THEN (může být rozšířena na

IF-THEN-ELSE), která vychází z implikace ve výrokové logice. Pravidla mají dva principy zápisu. První ze způsobů je procedurální a druhý deklarativní. Procedurální princip využívá pravidla „pokud nastane situace pak systém provede akci“. Toho se využívá hlavně v generativních znalostních systémech. Deklarativní princip spočívá v tom, že klade důraz na řešení problému. Tedy jedná se o způsob typu „platí-li předpoklad, pak platí závěr“. Tento princip je používán v diagnostických znalostních systémech jako například v již dříve zmíněném systému Mycin. [2]

1.4.2 Rámce

Rámce si můžeme představit jako struktury u vyšších programovacích jazyků. Je to typ, který je zaměřen převážně na práci s instancemi. Jeho hlavní rys je ten, že se používá k popisu komplexnějších objektů se spoustou relací. Rámce můžeme rozdělit na dvě kategorie: generické a konkrétní. Generické objekty mají obecný vzor pro vytváření instancí, zatímco konkrétní reprezentují formu pro určitá data. Položky pro uchovávání dat se nazývají *sloty* a jejich hodnoty *náplně*. [1]

Rámce bývají rozšířeny o pravidla pro odvozování v bázi znalostí. Mají schopnost zapouzdřit své vlastnosti a dědit je od ostatních rámců. Dědičnost si můžeme představit jako vytváření hierarchizace ve struktuře objektů, kde rámec na nejnižší úrovni dědí vlastnosti a pravidla z vyšších úrovní. Další typ záznamu, který lze najít v objektech, je *metapoložka*, která umožňuje popsat procedury a definovat chování jednotlivých položek. Například *if-charge* provede při změně hodnoty určitého záznamu chtěnou akci nebo *order-of-sources* pro specifikování způsobu získání hodnoty položky. [2]

Výhody rámců jsou ve snazším usuzování, mohou uchovávat dynamické hodnoty, jsou více strukturované než například sémantické sítě, mají možnost stanovit své použití v konkrétní situaci a jsou vhodné pro simulaci, diagnostiku či plánování. Mezi jejich nevýhody patří špatné přizpůsobení novým situacím a složité definování heuristických znalostí. [5]

1.4.3 Sémantické sítě

Tento typ reprezentace vznikl v 60. let minulého století. Sémantické sítě představují orientované grafy s množinu objektů představující jejich uzly, které jsou mezi sebou propojeny. Tyto vztahy označujeme jako relace. Sémantická síť, na rozdíl od obecné sítě, popisuje jednotlivé závislosti objektů, které slouží jako způsob pro vyjadřování znalostí. Nejdůležitějšími typy závislostí jsou: *„is a“* (ISA), *„a kind of“* (AKO) a *„is a part of“* (ISPART). ISA představuje vztah, kdy lze objekt reprezentovat jako jiný, AKO určuje relaci,

kdy je objekt druhem jiného objektu (můžeme si představit například jako dědičnost) a ISPART vyjadřuje, kterého jiného objektu je daný objekt součástí. [1], [2]

Sémantické sítě jsou jasně prezentované a zrychlují dobu hledání. Do jejich nevýhod patří možnost chybné inference a neexistence standardů pro interpretaci či pojmenovávání vazeb. [5]

1.5 Inferenční mechanismus

Inference je definována jako odvozování určitých výroků z jiných. Inferenční mechanismus představuje ve ZS strojové usuzování. Na základě předaných faktů mechanismu se určuje, která pravidla v jeho prioritním seznamu jsou splněna. Podle toho se určuje výsledná znalost neboli expertíza. Inferenční mechanismus (IM) spolu s kvalitou báze znalostí má velký vliv na spolehlivost a efektivnost ZS. Pro sestavení IM se používá řada metod. Těmi nejdůležitějšími jsou dedukce, indukce, výroková a predikátová logika nebo dopředné a zpětné řetězení. Příklad jednoduché struktury pro IM je orientovaný strom. Orientovaný strom je hierarchický orientovaný acyklický souvislý graf, kde vstupním bodem je kořenový uzel. Každý uzel reprezentuje znalost či informaci. Konečný bod, který nemá své následníky se nazývá jako list a bod, který nemá předchůdce jako kořen. Jelikož se jedná o souvislý graf, tak bod, který je listem a zároveň kořenem musí být jediným uzlem v grafu. [1]

1.5.1 Deduktivní logika

Prvním typem IM, kterému se budeme věnovat, je *deduktivní logika*. Postupuje se od obecného tvrzení ke konkrétnímu. Tato metoda vychází z výrokové logiky, kde se z předpokladů dospívá k závěru či novému tvrzení. Předpokládáme-li pravdivost předpokladu a výsledku pravidla (implikaci), pak i závěr musí být pravdivý (*modus ponens*). Obdobně můžeme určit nepravdivost předpokladu pomocí platnosti pravidla a chybného závěru (*modus tollens*).

Tabulka 1: Dedukce [2]

modus ponens	modus tollens
$A \Rightarrow B$	$A \Rightarrow B$
A	$\neg B$
B	$\neg A$

Pro deduktivní úsudek se využívá hlavně sylogismu. Sylogismus má dva předpoklady (premisy), ze kterých se určuje závěr (výrok). Příklad sylogismu je například:

hlavní premisa: Každý člověk je smrtelný.

vedlejší premisa: Tchyně je člověk.

výrok: Tchyně je smrtelná.

V tomto příkladu je „člověk“ nazýván jako *střední termín*, „smrtelnost“ jako *vedlejší termín* a „tchyně“ jako *hlavní termín*. Sylogismy mohou být také přepsány do tvaru *„Když ..., potom ...“*. Premisy a výrok patří do jedné z kategorických tvrzení viz tabulka níže. [1], [2]

Tabulka 2: Kategorická tvrzení sylogismu [1]

forma	tvrzení	význam
A	všechna A jsou B	univerzální pozitivní
E	žádná A nejsou B	univerzální negativní
I	některé A jsou B	částečně pozitivní
O	některá A nejsou B	částečně negativní

1.5.2 Výroková logika

Výroková logika je symbolikou reprezentující logické operace nad výroky. Základem jsou atomy, které tvoří za pomocí běžných logických spojení výrokové formule. Logická spojení jsou: negace (zapisující se jako „¬“), konjunkce (představující spojku *A*, označována znakem „∧“), disjunkce (reprezentující spojku *NEBO*, znakem je „∨“), implikace (její význam by se dal přepsat do věty *„Z ... plyne ...“*, zapisuje se jako „⇒“) a ekvivalence (označuje stav, kdy platí-li jeden z atomů, pak musí zároveň i druhý a je označován jako „⇔“). [1]

Výroková logika nám poskytuje možnost vyjádřit více typů výroků než za pomocí sylogismů a deduktivní logiky, byť má také svá omezení. Schéma tohoto příkladu IM je modus ponens. Pokud využijeme příkladu sylogismu z kapitoly deduktivní logiky, pak se o modus ponens nejedná a nelze na něj využít osvědčení výrokovou logikou. Přepíšeme-li jednotlivé premisy a výrok do logických proměnných, dostáváme následující:

A = Každý člověk je smrtelný.

B = Tchyně je člověk.

C = Tchyně je smrtelná.

Jelikož na sebe jednotlivé proměnné nenavazují a není zde definováno pravidlo, tak není žádná možnost určit pravdivost výroku podle premis. Ale můžeme jej přepsat do tvaru, kdy už výroková logika použít lze, viz tabulka níže. [1]

Tabulka 3: Příklad výrokové logiky

Pokud je tchýně člověk, pak je smrtelná.	A ⇒ B
Tchýně je člověk.	A (je člověk)
Proto, tchýně je smrtelná.	B (je smrtelná)

1.5.3 Predikátová logika

Predikátová logika se může použít pro reprezentaci deduktivní logiky. Predikátová logika používá existenční (∃) a univerzální (∀) kvantifikátory. Kvantifikátor ∀ můžeme nahradit slovem *„každý“* a kvantifikátor ∃ jako *„existuje“*. Kvantifikátory nám umožňují potvrzovat či vyvracet tvrzení. Máme-li predikátovou reprezentaci $\neg(\forall\, x)\; P(x)$, pak ji můžeme ekvivalentně zapsat jako $(\exists\, x)\; \neg P(x)$ a pro negaci existenčního kvantifikátoru to funguje analogicky. Kategorická tvrzení, která jsou sepsaná výše (Tabulka 2), se dají přepsat predikátovou logikou viz tabulka níže. [1]

Tabulka 4: Kategorická tvrzení napsaná predikátovou logikou [1]

forma	Tvrzení	predikátová reprezentace
A	pro všechna *x* platí, že všechna *S* jsou *P*	$(\forall\, x)\; (S(x) \Rightarrow P(x))$
E	pro všechna *x* platí, že žádná *S* nejsou *P*	$(\forall\, x)\; (S(x) \Rightarrow \neg P(x))$
I	existuje *x*, kde *x* je *S* a *P*	$(\exists\, x)\; (S(x) \wedge P(x))$
O	existuje *x*, kde *x* je *S* a není *P*	$(\exists\, x)\;\; (S(x) \wedge \neg P(x))$

Využijeme-li příklad výše, pak při použití t = tchyně, C = člověk a S = smrtelná vypadá příklad následovně:

1. $(\forall\, x)\ (C(x) \Rightarrow S(x))$,
2. $C(t)$,
3. $C(t) \Rightarrow S(t)$,
4. $S(t)$.

1.5.4 Dopředné a zpětné řetězení

Množina násobných inferencí, které propojují zkoumaný problém se závěrem, je označována jako *řetězec*. Vyhodnocování inferencí od faktů k řešením takzvaná *usuzování zdola nahoru* se nazývá jako *dopředné řetězení*. Opakem dopředného řetězení je *zpětné řetězení*, kde se tedy postupuje směrem od domněnek k faktům (*usuzování shora dolů*). [1]

Dopředné řetězení se obvykle znázorňuje grafem s fakty dole a výsledky nahoře, od toho vyplývá označení zdola nahoru. Využívá se hlavně pro problémy monitorování, diagnostiku či problémy zahrnující syntézu. U problémů zahrnující syntézu (například plánování, konfigurace či navrhování) musí být znalosti obecnými vzory kvůli počtu možných řešení (přesné vztahy nemohou být určeny). V rámci řetězení se prohledávají možná řešení, která vyplývají z dat, pomocí metody prohledávání do šířky. V rámci této metody se cyklicky opakují tři kroky: porovnání, řešení konfliktu a provedení. Krok porovnání slouží k srovnání pravidel z BZ a fakty. Pomocí porovnání se určí, které předpoklady jsou splněny. Z množiny pravidel, která jsou splněna, je určeno pravidlo následující. Je-li splněno více pravidel, pak se vybírá to s největší prioritou. Nachází-li se více pravidel se stejnou prioritou, pak se určí na základě definované strategie. Příklady strategií jsou: upřednostňování nejaktuálnějších pravidel (data uložená v bázi znalostí nejkratší dobu), preferování starších pravidel, zvýhodňování pravidel založené na specifičnosti či složitosti (např. podle počtu podmínek) a preferování jednodušších pravidel. Následně se vykoná provedení. Tato poslední část algoritmu vykoná vybrané pravidlo z předchozího kroku. Při vykonání pravidla se může provést změna nad bází faktů, respektive nad bází znalostí, například odebráním či vložením faktu, respektive znalosti. Zpravidla se vyhodnocení provádí pouze jednou nad stejnou skupinou faktů. [1], [5]

U zpětného řetězení dochází k řízení na základě cíle nebo hypotézy. Stejně jako dopředné řetězení je zobrazováno grafem, kde jsou fakta dole a řešení nahoře. K prohledávání faktů, které předcházejí či podporují zvolenou hypotézu, je využívána metoda prohledávání do hloubky.

Zpětné řetězení se používá především k diagnostikám, tedy v aplikacích mající více vstupů než závěrů, či pro klasifikační problémy. Obsah algoritmu se skládá z následujících kroků.

1. Vytvoří se zásobník, který bude obsahovat všechny možné hypotézy.
2. Pokud je zásobník neprázdný, pak se k závěru na jeho vrcholu naleznou všechna pravidla, která dokážou splnit tento cíl, jinak se algoritmus ukončí.
3. V tomto kroku se postupně množina pravidel vytvořeného z předchozího kroku vyhodnocuje. Vybrané pravidlo se provede za předpokladu, že jsou všechny její podmínky splněny. Pokud je provedené pravidlo koncové, je odebráno ze zásobníku a pokračuje se 2. bodem. V opačném případě, kdy fakta podmínky nesplňují, je zkoumání pravidla ukončeno. Chybí-li některé podmínce v pravidlu fakta, tak se zjistí existence pravidla, ze kterého by hodnota šla odvodit. Existuje-li, pak se vyhodnocení na potřebnou dobu odloží a přejde se na bod 2, jinak se zjišťuje potřebná hodnota od uživatele.
4. Pokud žádné pravidlo nedokázalo odvodit důsledek, potom zůstává hypotéza neurčena a je odstraněna ze zásobníku. Pokračuje se krokem č. 2.

[1], [5]

1.5.5 Logické systémy a rezoluční metoda

Logický systém specifikuje vhodné formy tvrzení, validuje tvrzení podle inferenčních pravidel a vytváří nová inferenční pravidla pro rozšiřování nových tvrzení. Vhodnost formy je důležitým krokem z toho důvodu, když máme například dvě tvrzení obsahující stejné symboly, tak nemusí být obě validní. Špatné vyjádření formy může vést k chybě při prokazování platnosti tvrzení. [1]

Rezoluční metoda je využívána v aplikacích UI pro určování platnosti teorémů. Tato metoda vychází z konjunktivní normální formy. Testované tvrzení musí být v normální formě. Máme několik základních forem: *konjunktivní normální forma*, *klauzální forma* a *Hornova klauzální forma*. Normální formy jsou tvořeny základními logickými operátory konjunkce, disjunkce a negace. [1], [10]

Konjunktivní normální forma je zapisována ve tvaru konjunkce klauzulí. Klauzule je označována za konečnou disjunkci literálů. Literál představuje atomický výraz nebo negaci atomického výrazu (symbolu). Neobsahuje žádná logická spojení jako implikace, kvantifikátory apod.

$$(A_1 \vee A_2 \vee \ldots \vee A_n) \wedge (B_1 \vee B_2 \vee \ldots \vee B_n) \wedge \ldots \wedge (Z_2 \vee Z_2 \vee \ldots \vee Z_n).$$

Plná klauzální forma vyjadřuje formu, která je zapisována implikací. Tato forma jde za pomocí převedení implikace na disjunkci $((A \Rightarrow B) \Leftrightarrow (\neg A \vee B))$ a použití de Morganova zákona $(\neg(A \wedge B) \Leftrightarrow (\neg A \vee \neg B))$ přepsat na disjunktivní formu (klauzuli).

$$\big((A_1 \wedge A_2 \wedge \ldots \wedge A_n) \Rightarrow (B_1 \vee B_2 \vee \ldots \vee B_n)\big) \Leftrightarrow (\neg A_1 \vee \neg A_2 \vee \ldots \vee \neg A_n \vee B_1 \vee B_2 \vee \ldots \vee B_n)$$

Literály A_1 až A_n reprezentují požadované podcíle, které je nutné splnit ke splnění alespoň jednoho literálu z množiny klauzule B_1 až B_n. [1]

Hornova klauzální forma je typ omezené klauzální formy, která má na pravé straně implikace pouze jeden literál. Přepsáním z implikace na klauzuli pak dostáváme disjunkci literálů, kde se vyskytuje nejvýše jeden nenegovaný literál. Použití Hornovy klauzální formy lze najít například v programovacím jazyce Prolog. [1]

Rezoluční metoda odvozuje ze dvou klauzulí takzvané rezolventy (logické důsledky), které můžeme používat k vytváření dalších rezolventů. Rezolventa má strukturu klauzule. Za předpokladu, že je některá rezolventa kontradikce ($\bot$), pak tvrzení není splnitelné. Opak kontradikce je tautologie ($\top$), která označuje pravdu pro každé ohodnocení. Je-li A tautologií, pak to můžeme zapsat v následujícím tvaru: $\vDash A$. Výraz $A \vDash B$ platí právě tehdy, když je tautologie $A \Rightarrow B$. [10]

Pro odvozování rezolventů je základem následující věta[4]:

Nechť A, B, C jsou formule výrokové logiky. Potom

(i) $A \wedge \neg A \vDash \bot$,

(ii) $A \wedge (\neg A \vee B) \vDash B$,

(iii)$(A \vee B) \wedge (\neg A \vee C) \vDash (B \vee C)$.

Mějme příklad, který dokáže rezoluční metodou platnost *modus tollens.* Přepíšeme si toto tvrzení do následující podoby:

$$(A \Rightarrow B) \wedge \neg B \vDash \neg A.$$

[4] TRLIFAJOVÁ, Kateřina, Daniel VAŠATA a České vysoké učení technické v Praze. Matematická logika. Praha: České vysoké učení technické v Praze, 2013, s. 52. ISBN 978-80-01-05342-3.

Provedeme důkaz sporem. Tedy budeme předpokládat pro spor, že platí negace výše uvedeného tvrzení.

$$\neg\Big(\big((A \Rightarrow B) \wedge \neg B\big) \Rightarrow \neg A\Big)$$

Konjunktivní normální forma vypadá následovně. Nejdříve se zbavíme implikací, negací ($\neg\neg A \Leftrightarrow A$), použijeme asociativní zákon pro konjunkci ($(A \wedge B) \wedge C \Leftrightarrow A \wedge (B \wedge C)$) a de Morganův zákon.

$$\neg\big(\neg\big((\neg A \vee B) \wedge \neg B\big) \vee \neg A\big) \Leftrightarrow \neg\neg\big((\neg A \vee B) \wedge B\big) \wedge \mathrm{A} \Leftrightarrow (\neg A \vee B) \wedge B \wedge \mathrm{A}$$

Nalezneme rezolventy:

1. $(\neg A \vee B) \wedge A \vDash B$,
2. $B \wedge \neg B \vDash \bot$.

Druhá rezolventa je kontradikcí, tudíž vzniká spor s tím, že tvrzení platí. Platnost je tedy dokázána.

1.5.6 Indukce

Indukce je zobecnění případů neboli postupuje se od konkrétního výroku k obecnému závěru. Stejně jako dedukce tak i tato metoda vychází z výrokové logiky. Na indukci je například možné postavit automatické získávání znalostí z dat. Implikace se usuzuje na základě toho, že se z opakovaných pozorování nachází prvky, které se vyskytují současně. [1], [2]

1.5.7 Abdukce

Abdukce je třetím typem založeným na implikaci z výrokové logiky. Z pravdivého závěru dochází k usuzování, které předpoklady jej mohly předcházet. Tato metoda zachovává nepravdu, tedy postupuje se na základě toho, že se z neplatného závěru určují neplatné předpoklady, které skutečnému závěru nemohly předcházet. Označuje se také někdy za vyvozování nejvhodnějšího vysvětlení pro daná fakta. [2]

1.5.8 Analogie

Analogie je typ využívající se na odvozování závěru podle paralely s jiným případem. Tento typ je tvořen množinou již dříve vyřešených případů. Způsob takového vyhodnocování se dá připodobnit k angloamerickému právu, které je tvořeno soudci skrze precedenty. Pro poskytnutí závěru se hledá v množině případů takový případ, který je nové situaci co nejpodobnější.

Takový závěr, který se již osvědčil, se aplikuje znovu. Není tedy potřeba získávat znalosti od experta, stačí pouze mít dostatečný počet reprezentativních případů. [2]

1.5.9 Generování a testování

Při této metodě se využívá způsobu, kdy se cyklicky generuje možné řešení, které se následně testuje, zda vyhovuje veškerým požadavkům. Za předpokladu, že se najde vhodné řešení, pak se již nová řešení negenerují. Tento typ se využívá typicky v generativních znalostních systémech. Znalosti jsou tvořeny pravidly, která jsou reprezentována tvarem *„Když ..., potom ...“*. Běh jednoho opakování generativního systému se skládá ze tří částí. Nejprve se vytvoří rozhodovací množina obsahující pravidla, která jsou v dané chvíli aplikovatelná. Poté se vybere jedna dvojice (instance) složená z objektu uloženého v pracovní paměti, který vyhovuje podmínkám pravidla, a s ním i z daného vyhovujícího pravidla z rozhodovací množiny. V poslední části dochází k provádění akcí vybrané instance. Tyto akce slouží typicky pro operace nad objekty v pracovní paměti. [2]

1.5.10 Default

Za předpokladu, že IM nemá specifické znalosti, tak se usuzuje na základě obecných znalostí takzvaných defaultů. Pokud z faktů nedokážeme určit odpovídající závěr, použije se defaultní výrok. Například v době, kdy je pandemie, může doktor usuzovat, že se pacient pravděpodobně nakazil danou infekční nemocí, než že by se nakazil jinou. Pokud nám pacient blíže příznaky neurčí, bude se brát tento výchozí závěr. [2]

1.5.11 Nemonotónní usuzování a heuristiky

Nemonotónní IM používá usuzování, kde se mohou předchozí znalosti zneplatnit na základě nově získaných. [2]

Heuristiky jsou znalosti, které jsou založeny na zkušenostech experta či zobecnění situací ve kterých se již expert rozhodoval. Tyto znalosti tedy nemusejí být přesné a ani nemusejí být podložené. [2]

1.6 Vysvětlovací modul

Vysvětlovací modul slouží ve ZS pro vysvětlení postupů usuzování. Tento model je brán jako charakteristika znalostních systémů. Nepoužívá se k vyhodnocení závěrů či generování řešení. Důvodem proč mít tento modul v systému je ten, aby systém byl transparentní, důvěryhodný a uživatel měl jistotu, že je usuzování správné a řešení přijatelné. Modul může sloužit i pro experta k ověření, že implementované znalosti se vyhodnocují podle jeho představ. Ve ZS není

obvykle vysvětlování zcela promyšlené, poněvadž nemá celkový obraz nad celkovým chováním systému a vychází obvykle pouze z BZ. Při vysvětlování můžeme použít několik způsobů interpretace. Diagnostické znalostní systémy mají typicky následující způsoby: typ *Why*, *How* a *What if*. [2]

Vysvětlení typu *Why* je založeno na výkladu, proč systém položil svůj dotaz. Takový princip má smysl při požadování vysvětlení pouze v průběhu vyhodnocování. Zobrazuje se v tomto případě cesta od daného dotazu po aktuální cíl. Vysvětlování se může rozšířit o komentář k dotazům. [2]

Typ *How* je způsob, jakým se systém dostal k závěru. Na rozdíl od typu *Why* se může o vysvětlení požádat jak při odvozování, tak i po jeho skončení. Shodují se ale ve své možnosti doplnit vysvětlení do komentáře, v tomto případě k pravidlům. Pravidla, která mají kladný či záporný vliv na dosažení závěru, se zobrazí jako vysvětlení. [2]

What if je označován za typ, který vysvětluje, jak by při změně odpovědi na nějaký dotaz systém reagoval. Jedná se o takzvané hypotetické usuzování. Po tomto alternativním odvozování by měl následně vrátit výsledky dotazů do původního stavu. Stejně jako u typu *How* se o něj může požádat při odvozování i při již předloženém doporučení. [2]

Některé ZS mají schopnost zaznamenávat jednotlivé dotazy a pravidla, která se postupně odvozují. Jejich analýza se také může stát prostředkem pro pochopení vyhodnocení závěrů. Výhodou zaznamenávání trasy napříč dotazy je schopnost využití k ladění či úpravám BZ. [2]

1.7 Neurčitost ve znalostních systémech

Mnoho systémů je zpracováváno na základě jednoznačně daného usuzování, ale některé tuto možnost nemají a potřebují schopnost nepřesného usuzování. Charakteristikou složitých systémů je neurčitost, která řeší tento problém. Příčina neurčitosti ve ZS je nedostatečnost informací pro jednoznačnou inferenci. To může být způsobeno například nedostupnými, nespolehlivými, nepřesnými či chybějícími daty nebo znalostmi, které mohou obsahovat nejasné pojmy. Takové znalosti a data nastávají dvojznačností, lidskou chybou, chybou měření, sémantickou chybou, nekompletností nebo náhodnou chybou. Pravděpodobnost je reprezentována číselnými parametry, které nejčastěji nabývají hodnot z uzavřeného intervalu od −1 do 1, popřípadě od 0 do 1. Nejčastěji se setkáme s jednočíselnou hodnotou, ale začínají se vyskytovat i dvojice. Tyto parametry se mohou na jednotlivých systémech nazývat různě. Například jsou pojmenovávány jako *váhy*, *faktory nejistoty*, *míry* nebo jako *stupně důvěry*.

Neurčitosti jsou zpracovávány různými nástroji, kterých je celá řada. Jedním takovým z nástrojů je například fuzzy logika. [1], [5]

1.7.1 Fuzzy logika

Fuzzy logika někdy bývá nesprávně považována za pravděpodobnost a dochází tak často k nedorozumění. Oba přístupy se liší především ve zdroji neurčitosti. Vysvětleme si nejprve rozdíl mezi pravděpodobností a fuzzy logikou na příkladu. Řekněme, že se rozhodneme jít do kina. V tomto případě se jedná o pravděpodobnost, že film uvidíme (kino může být například zavřené). Přijdeme-li do kina, pak už se o neurčitost nejedná a dostáváme jasný výsledek. Ve fuzzy logice neurčitost přetrvává i po provedení daného pokusu. Příklad pro fuzzy logiku v tomto případě může představovat hodnocení daného filmu. Na zodpovězení této otázky bychom potřebovali více možných odpovědí než jen dvě (ano/ne), například je rozšířit o odpovědi „spíše ano“ a „spíše ne“. Jejich reprezentací může být jak jazyková, tak i numerická hodnota. O pravděpodobnost se v tomto případě jednat nemůže, protože neurčujeme, jestli se náhodně vybranému člověku bude či nebude film líbit. Závisí to na každé hodnotě daného jevu: na konkrétním člověku a konkrétním filmu v daném kině. [3]

U Fuzzy množin se omezíme na podmnožiny univerzální množiny, která je neprázdná a pevně daná (dále budeme značit jako *U*). Fuzzy množinu A v univerzu U představuje dvojice definovaná jako $A = (U, \mu_A)$, kde μ_A je funkce označovaná jako *funkce příslušnosti*. U ostrých množin, které nejsou fuzzy, je prvkům z univerza přiřazována příslušnost 1 za předpokladu, že do této množiny patří, v opačném případě je příslušnost 0. Jak jsme si již řekli, potřebujeme mít větší škálu možností, proto funkce příslušnosti pro fuzzy množinu je definovaná následovně $\mu_A: U \rightarrow \langle 0,1 \rangle$. Tato funkce pro každý prvek x z univerza U přiřazuje hodnotu z uzavřeného intervalu od 0 do 1. Stupeň příslušnosti je určen předpisem $\mu_A(x)$. Pro prázdnou množinu *A* je stupeň příslušnosti rovna 0 pro všechny prvky z univerza *U*. Podmnožina fuzzy množiny, kde každý prvek má stupeň příslušnosti 1, je nazývána jádrem ($ker(A) = \{x \in U: \mu_A(x) = 1\}$). Nosičem je taková podmnožina, která obsahuje prvky mající stupeň příslušnosti větší než 0 ($supp(A) = \{x \in U: \mu_A(x) > 0\}$). Dle výšky ($h(A) = sup\{\alpha \in \langle 0,1 \rangle: (\exists x \in U: \mu_A(x) = \alpha)\}$) můžeme určit, zda je fuzzy množina normální ($h(A) = 1$) či subnormální (v ostatních případech). Pro univerzum *U*, která je konečnou množinou, můžeme operátor *sup* zaměnit s operátorem *max*. [3], [5]

Pro vyjádření nepřesností se využívají *fuzzy čísla*, která představují fuzzy množinu definovanou na množině reálných čísel. Fuzzy čísla jsou reprezentována spojitou funkcí a čtveřicí bodů (a_1, a_2, a_3, a_4), pro které platí následující:

(i) $a_1 \leq a_2 \leq a_3 \leq a_4$,

(ii) pro $x \leq a_1, x \geq a_4$ platí $\mu_A(x) = 0$,

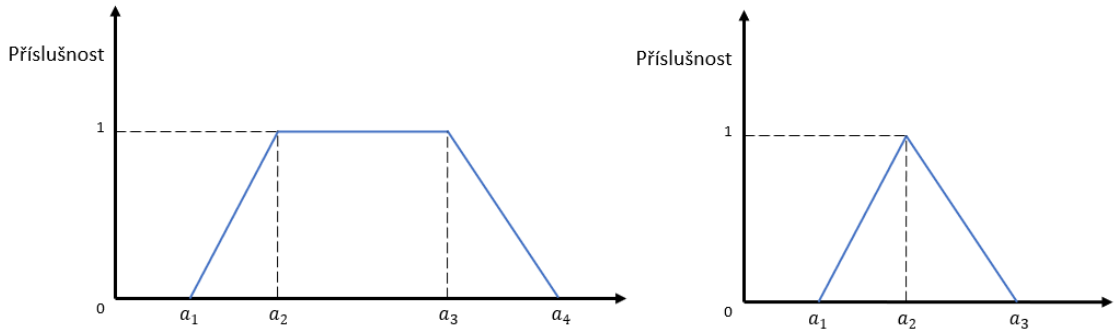
(iii) pro $a_1 \leq x \leq a_2$ platí $\mu_A(x) = 1$,

(iv) funkce $\mu_A(x)$ je na intervalu mezi a_1 a a_2 rostoucí a na intervalu mezi body a_2 a a_3 klesající.

V praxi se používají lichoběžníková nebo trojúhelníková fuzzy čísla, protože složitější průběh funkce je obtížné určit. Trojúhelníková fuzzy čísla jsou specifikována za pomocí trojicí bodů. Můžeme říct, že se jedná o speciální případ lichoběžníkových fuzzy čísel, kde pro čtveřici čísel platí $a_2 = a_3$. Funkce příslušnosti je definována následovně [5]:

$$\mu_A(x) = \max\left\{min\left\{\frac{x-a_1}{a_2-a_1}, \frac{a_4-x}{a_4-a_3}, 1\right\}, 0\right\}.$$

Příklad těchto čísel je znázorněn na obrázku níže. [3], [5]



Obrázek 2: Příklad lichoběžníkového a trojúhelníkového fuzzy čísla, upraveno dle [5]

Mějme pravidlo IF $X = A$ THEN $Y = B$, kde A je fuzzy množina na univerzu U a B je fuzzy množina na univerzu V. Toto pravidlo můžeme chápat jako *fuzzy relaci* $R = (U \times V, \mu_R)$, kde její funkce příslušnosti je definována jako $\mu_R(x, y) = \min\{\mu_A(x), \mu_B(y)\}$. Příkladem takového pravidla může být např. IF $nadmořská\ výška = velká$ THEN $atmosferický\ tlak = malý$. Fuzzy systémy používají zobecněná pravidla fuzzy modus ponens. Určíme-li například pro pravidlo předpoklad $X = A'$, ze kterého plyne $Y = B'$, potom fuzzy množina $B' = (V, \mu_{B'})$ může být definována následovně: $\mu_{B'}(y) = \sup_{x \in U} \min\{\mu_{A'}(x), \mu_R(x, y)\}$. [5]

Pokud potřebujeme pracovat s ostrými hodnotami, potřebujeme fuzzy množině přiřadit takovou ostrou hodnotu, která ji co nejlépe bude vystihovat. Pro tento účel slouží *defuzzifikace*. Pro určení ostré hodnoty slouží řada metod. Nejpoužívanějšími metodami je *metoda těžiště*, která u funkce příslušnosti vybere těžiště plochy pod grafem. Dále se používá *metoda maxima*, jejíž výsledkem je absolutní maximum funkce příslušnosti. Je-li takových bodů více, pak můžeme vybrat první bod či vypočítat průměr z těchto hodnot. Metody se vybírají podle předchozích zkušeností z konceptů v dané oblasti. [3], [5]

Fuzzy znalostní systémy dokážou zpracovávat vágní a nepřesná data za pomocí fuzzy logiky. Tyto systémy nejprve přijímají *ostrá* data, která následně předají fuzzyfikátoru. Fuzzyfikátor převede tyto vstupní data na pravdivostní hodnoty. Báze znalostí je zde využívána jako soubor pravidel hledající jednotlivé stupně příslušnosti. V inferenčním mechanismu se pravdivostní výstupy jednotlivých funkcí mezi sebou v rámci vyhodnocování předávají. Konečným výstupem těchto pravidel je fuzzy množina. Defuzzifikátor převádí pravdivostní výstupy těchto funkcí na ostré hodnoty za pomocí fuzzy množiny a jednotlivých stupňů příslušnosti. Systém jako konečný výsledek vrací převedenou ostrou hodnotu. Podle [7], který analyzoval sedm jednotlivých znalostních systémů z různých oblastí, jako je zdravotnictví, vzdělávání, finance a výběr povolání se v těchto systémech častěji vyskytoval fuzzyfikátor, který využíval především lichoběžníková než trojúhelníková fuzzy čísla. [7]

2 POPIS VYTVOŘENÉ APLIKACE

Tato kapitola má za cíl popsat vývoj vlastního znalostního systému, který je realizován v podobě grafické aplikace. Pro tuto aplikaci byla vybrána problematika rozhodování o zařazení do evidence uchazečů o zaměstnání a v případě vzniknuvšího nároku na podporu v nezaměstnanosti jejího výpočtu. Tento případ zaměření vychází ze zákona č. 435/2004 Sb., o zaměstnanosti v platném znění.

V této bakalářské práci je při návrhu využito modelování pomocí jazyka UML (Unified Modeling Language). Jedná se o jazyk, který vznikl v 90. letech minulého století za účelem popisu vývoje softwaru. K popisu vlastní aplikace byly použity následující diagramy: diagram případu užití (use-case diagram) a diagram tříd. [9]

2.1 Analýza

Výpočet výše podpory v nezaměstnanosti nebo ověření, zda má osoba právo být zařazena do evidence uchazečů o zaměstnání, je bez znalosti zákona nemožné. Tato aplikace tedy poskytuje expertní rozhodnutí na základě odpovědí případného uchazeče na položené otázky. S každou otázkou se uchovává odpověď uživatele a zároveň paragraf, ke kterému náleží. Jako výsledek není tedy pouze rozhodnutí, ale také seznam pracovněprávních předpisů pro snadnější orientaci či jako podpůrná data pro další uživatelův postup. Tato aplikace tedy slouží každému nezaměstnanému, který si není jist předpisy a chce si ověřit, jestli v jeho případě vzniká možnost nároku na podporu nebo alespoň splňuje podmínky pro zařazení do evidence uchazečů o zaměstnání a s ním i hrazení zdravotního pojištění. Na druhou stranu slouží i pro osoby, které se v tomto odvětví velmi dobře pohybují či se vyznají v zákonech o zaměstnanosti a všech prováděcích předpisech. O takových osobách budeme dále mluvit jako o expertech. Expert má možnost editovat otázky a konstanty pro výpočet v bázi znalostí. Jelikož se taková čísla každoročně mění a nemusela by aplikace být nadále aktuální, je zde implementován režim pro editaci.

Navrhovaný znalostní systém je navržen na konkrétní příklad s možností snadného rozšíření a postaven na výpočtu podpory. Změní-li se legislativa, která změní způsob kalkulace podpory, musí dojít k úpravám v aplikaci. Ostatní změny v legislativě nemají na aplikaci vliv, pouze na její znalostní bázi, kterou bude muset expert poupravit. Způsob, jak je systém vytvořen, umožňuje vytvořit i zcela jiný příklad rozhodování. Využití výpočtu není tedy v aplikaci nutný, proto můžeme říct, že má znaky obecného znalostního systému, který je schopný vytvořit rozhodování pro jakékoli úlohy.

Aplikace využívá Java Development Kit ve verzi 1.8.0 od společnosti Oracle.

2.1.1 Požadavky

V rámci analýzy si musíme určit podmínky, respektive požadavky, jak bude znalostní systém vypadat a jak se bude chovat. Všechny definované požadavky musejí být v konečné fázi tvorby systému splněny. Tyto podmínky neřeší, jak dosáhnout požadované podoby či jak je přesně implementovat, nýbrž tvoří množinu nezbytností vytvářeného systému. Analytické požadavky se dělí na dva typy: funkční a nefunkční.

Nejprve si stanovíme funkční požadavky, které slouží k bližšímu určení chování systému. Takové požadavky jsou definovány následovně.

1. Znalostní systém bude schopen vypočítat podporu v nezaměstnanosti.
2. Znalostní systém bude uživateli zobrazovat relevantní výsledek dle jeho odpovědí.
3. Znalostní systém bude umožňovat vybrat bázi znalostí.
4. Znalostní systém bude umožňovat správu báze znalostí.

První až třetí bod z výše uvedených požadavků se týká přímo chování systému. Čtvrtý bod nám říká, že systém bude schopen upravit načtenou bázi. V tomto případě tento požadavek také nepřímo naznačuje podmínku na znalostní bázi, která určuje potřebu ji mít editovatelnou.

Nefunkční požadavky na rozdíl od funkčních popisují pouze jeho vlastnost či omezení. Pro vytvářený systém jsou popsány níže.

1. Znalostní systém bude odolný proti chybám.
2. Znalostní systém bude v podobě počítačové aplikace.
3. Znalostní systém bude zajišťovat integritu báze znalostí.
4. Znalostní systém bude mít grafické uživatelské rozhraní.

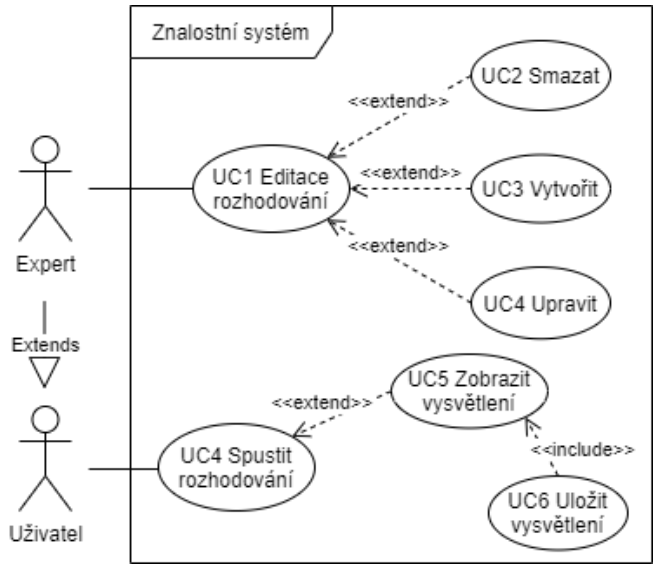
Bod 1. až 3. nám popisuje vlastnosti systému. Zbylý požadavek nám určuje omezení rozhraní, aby práce s tímto systémem byla pro uživatele co nejpřívětivější.

2.2 Návrh

Tato kapitola se zaobírá návrhem systému a jeho popisem. V první řadě se bude zabývat možným užitím systému všech jeho uživatelů, rozčleněných podle rolí. K tomu je využito diagramu případu užití. Další podkapitola řeší návrh jednotlivých tříd, jejich vzájemné propojení a vztahy. Tento návrh je znázorněn pomocí diagramu tříd.

2.2.1 Případy užití

Aplikace je využívaná dvěma uživatelskými rolemi. Jsou to běžný uživatel a expert. Běžný uživatel je takový uživatel, který nemá vědomosti o dané problematice znalostního systému. V tomto případě o zákonu o zaměstnanosti. Pouze tuto aplikaci využívá ke zjištění jeho konkrétní situace. Expert je takový uživatel, který má znalosti o dané problematice a který je schopen na jejich základě upravit, aktualizovat či vytvořit rozhodování v této aplikaci.



Obrázek 3: Případy užití

Expert má v expertním režimu aplikace možnost editovat rozhodování systému. Jsou povoleny základní operace vkládání, úprava a odstraňování nad daty znalostí či proměnných, které jsou ukládány do báze znalostí. Chce-li expert otestovat své vlastní vytvořené rozhodování, provede to v uživatelském režimu, kde má stejné funkce jako uživatel.

V uživatelském režimu je povoleno pouze spustit rozhodování, kde se odpovídá na zobrazované otázky, dokud nedojde k vyhodnocení na základě odpovědí uživatele. V případě dosáhnutí konečného výsledku má uživatel možnost zobrazit vysvětlení, jak se tohoto výsledku docílilo. Tento výpis vysvětlení může uložit v textovém souboru na svém zařízení.

2.2.2 Návrh tříd

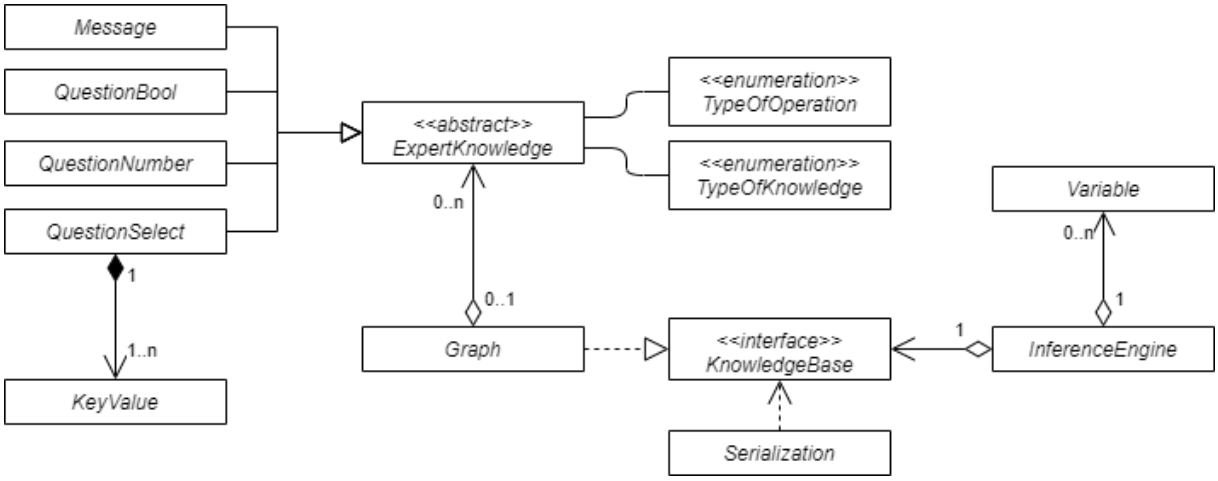
Při návrhu tříd se drželo základního členění znalostních systémů. Základními třídy, na kterých se při návrhu stavělo, byly třídy *InferenceEngine*, *KnowledgeBase* a *ExpertKnowledge.*

Programovací jazyk Java, stejně jako spousta dalších jazyků, nedovoluje vytvořit instanci takovéto třídy. Tuto třídu lze použít pouze jako typ ukazatele, který zastupuje odvozené třídy. Z tohoto důvodu je využíván jako prvek v inferenčním mechanismu. Každému z jednotlivých

druhů znalostí je proto přiřazena konkrétní odvozená třída, ze které už je možné instanci vytvořit a konkrétněji s nimi pracovat dle jejich vlastností. Abstraktní třída ExpertKnowledge reprezentuje typy znalostí báze, a to třídy *Message*, *QuestionBool*, *QuestionNumber* a *QuestionSelect*. QuestionSelect oproti ostatním uchovává seznam prvků klíč-hodnota, ze kterých uživatel vybírá odpověď. Abstraktní třída si jako informaci o daném typu znalosti uchovává v proměnné obsahující element výčtové třídy *TypeOfKnowledge*, který se využívá pro uložení znalosti do báze, jelikož pro více skutečných typů znalostí může být vytvořena stejná instance ze skupiny rozšiřujících tříd. Pokud bychom si takto informaci neuchovávali, je zde možnost o skutečnou informaci přijít. Každá třída má možnost rozhodování, na jakou další znalost přejít. Rozhodování se provádí pomocí lambda funkcí. Aby se při ukládání uložila informace o funkci, je uchována v proměnné, která tento problém zaštiťuje výčtovým prvkem z *TypeOfOperation*. [9]

Nejdůležitější třídou je zde ale InferenceEngine. Ta se stará s třídou KnowledgeBase o přechod mezi jednotlivými znalostmi a celkové vyhodnocení. Tyto třídy vyhodnocují podle skupin znalostí, do které patří. V rámci sekce je několik podgrafů mající určité vlastnosti, mezi kterými postupně mechanismus vybírá. Pokud v rámci sekce nedosáhne jednoznačného výsledku, přechází se na další. Třída KnowledgeBase byla navrhnuta jako interfejs, který je rozšiřován třídou *Graph* uchovávající jednotlivé znalosti.

Pro ukládání, respektive načítání z báze znalostí je navrhnuta třída *Serialization*, která využívá právě KnowledgeBase pro průchod nad znalostmi, respektive přímé ukládání vytvořených instancí skrze referenci.



Obrázek 4: Diagram tříd

Na obrázku je možnost vidět návrhy a propojení jednotlivých tříd popsaných výše.

2.3 Implementace

Při výběru způsobů jak znalostní systém konkrétně implementovat, bylo vycházeno z toho, že bude určen pro řešení problematiky zařazení do evidence uchazečů o zaměstnání a přiznání či nepřiznání podpory v nezaměstnanosti. Schopnost programu být snadno škálovatelný, zahrnuje možnost být využit i pro zcela jiný znalostní systém. Program byl postaven na platném výpočtu z editovatelných proměnných, kde poslední legislativní změna pro výpočet podpory je platná od 1. 1. 2011[5]. Pro implementaci byl zvolen objektový programovací jazyk Java[6].

2.3.1 Báze znalostí

Na uchování otázek, znalostí a proměnných, které se používají aplikací pro výpočet podpory nebo jsou použity expertem pro určité rozhodování v rámci vyhodnocení, slouží báze znalostí. Každé pravidlo je uchováváno v inferenčním mechanismu systému. Pro uchování těchto dat byla vybrána specificky organizovaná reprezentace dat v *csv* souborech. Bázi znalostí tvoří dva rozdělené soubory na proměnné a znalosti. Každý soubor má své vlastní uspořádání dat pro konkrétní zpracování aplikací.

Struktura souboru pro proměnné je rozdělena do tří položek. První sloupec uchovává jméno, sloužící jako klíč pro hledání uložené hodnoty, druhý inicializační hodnotu a poslední sloupec expertovu poznámku k jakému použití či využití slouží. Aplikace používá jednu hlavní proměnnou uchovávající identifikátor znalosti, uloženého ve druhém souboru, kterou sekcí začne inferenční mechanismus procházet znalosti až do konečného vyhodnocení. Je-li v aplikaci vybraná možnost využití vnitřního výpočtu podpory v nezaměstnanosti, je k této proměnné přiřazeno devět dalších. Jsou to dále popsané proměnné. Proměnná identifikátoru takového bodu v inferenčním mechanismu, kde má program veškeré získané informace od uživatele, podle kterých dojde k výpočtu výše podpory. Průměrná mzda v národním hospodářství za první až třetí čtvrtletí předcházejícího kalendářního roku, na které je postaven výpočet podpory. Pro uchování výsledku slouží tři proměnné, které mají za cíl reprezentovat výsledek výše podpory pro jednotlivé části podpůrčí doby. Dále proměnné, které budou naplněny skrze zadané informace uživatelem. Jedná se o délku podpůrčí doby branou skrze věk uživatele, výsluhový příspěvek a s výpočtem související také množina násobků průměrné mzdy pro jednotlivé podpůrčí doby. Je-li předán aplikaci prázdný soubor, sama se postará o vytvoření

[5] Často kladené otázky. *Úřad práce ČR* [online]. Poslední aktualizace 19. 11. 2019 [cit. 2021-03-27]. Dostupné z: https://www.uradprace.cz/casto-kladene-otazky-2

[6] „Java je čistě objektový jazyk, program se tedy skládá pouze z deklarací objektových typů (tříd). Alespoň jedna z tříd musí mít metodu, která se jmenuje main, a běh programu začne prvním příkazem této metody.“ [9]

výše uvedených proměnných s inicializačními hodnotami aktuální k roku 2021. Tím je ošetřeno to, že nenastane tedy taková situace, aby vzorec výpočtu uvnitř aplikace nebyl provázán s žádnou proměnnou naplněnou expertem či uživatelem.

Soubor pro ukládání otázek nebo znalostí je složen z identifikátoru, typu, interpretace, vysvětlení, podmínky, proměnné pro uložení odpovědi (případně pro využití v podmínce), identifikátor následující znalosti při splněné, resp. prázdné podmínce, identifikátor následující znalosti při nesplněné podmínce a poslední položkou je identifikátor sekce do které patří[7]. Máme osm druhů znalostí, které jsou reprezentovány v souboru vlastním jedinečným identifikátorem. Je-li konkrétní otázka nepovinná, identifikátor má příponu *„_CH“*. Typy máme:

- BOOL – Otázka ANO/NE,
- NUM – otázka s možností zadání čísla (nemusí být požadována odpověď),
- SEL – otázka s možností výběru (nemusí být požadována odpověď),
- MES – typ informativního okna (u podmínky se využívá hodnota z proměnné),
- SEC – položka sekce sloužící jako vstupní bod pro vyhodnocení množiny znalostí do ní spadající,
- CLR – nastavení proměnné na inicializační hodnotu,
- INC – inkrementaci proměnné o hodnotu 1,
- END – konečný bod, sloužící ke konečnému vyhodnocení.

Druh *„otázka s možností výběru“* má oproti ostatním další rozdělení. V souboru je dělena na tři části znakem „|“. První část je zmíněný identifikátor, druhá odpovídá seznamu výběru odpovědí a třetí seznam číselných hodnot, obě části oddělované jednotlivé položky čárkou. Oba seznamy se párují podle pozice, z toho vyplývá, že mohutnost těchto množin musí být stejná. Pro podmínku se využívá porovnání vstupu, resp. proměnné (pouze u typu *„informativního okna“*) s číslem či proměnnou. Typy porovnání jsou: NULL (podmínka není nastavena a vždy bude znalost brána jako pravdivá), EQ (rovná se), NE (nerovná se), LT (je menší než), LE (je menší rovno než), GT (je větší než) a GE (je větší rovno než). Další část s vlastním rozdělením je využívaná proměnná. Má pouze dvě části. Část vyhrazenou pro operaci, která definuje, co se má stát se vstupní hodnotou. Tato položka je prázdná pro druh otázek či znalostí, které

[7] Poslední údaj se nevztahuje na koncovou znalost a znalost představující sekci, které do žádné sekce nespadají.

nepřijímají vstup od uživatele jako číslo (platí tedy i pro výběrovou otázku). Operace jsou: ADD (přičíst číslo), SUB_VAR (odečíst vstup od proměnné), SUB_IN (odečíst proměnnou od vstupu a přiřadit výsledek do této proměnné), INS (přepsat proměnnou vstupem), DIV_VAR (vydělit proměnnou vstupem), DIV_IN (vydělit vstup proměnnou a přiřadit výsledek do této proměnné) a MUL (vynásobit proměnnou vstupem). A druhou částí, která uchovává název proměnné ze druhého souboru.

2.3.2 Inferenční mechanismus

Inferenční mechanismus je implementován za pomocí orientovaného grafu. Orientovaný graf je tvořen konečnou množinou uzlů V a množinou uspořádaných párů uzlů z množiny V, které tvoří orientované hrany E. U orientovaných hran záleží na pořadí tohoto páru uzlů. Pořadí určuje směr návaznosti. Mějme následující příklad. Nechť máme vrcholy v_1 a v_2 z množiny V, pro které je dána orientovaná hrana. Potom hrana, která jde z uzlu v_1 do v_2, je znázorněna pomocí předpisu $e = \langle v_1, v_2 \rangle$. To znamená, že můžeme jít z uzlu v_1 na uzel v_2, ale opačně už nikoliv. Graf je zapisován jako $G = (V, E)$. Množina vrcholů, v našem případě se jedná o množinu znalostí, uchovávají odkazy, které určují mezi sebou možnost návaznosti neboli hran. Odkaz je tvořen identifikátorem buď na další znalost či označením představující zakončení, není-li dána pro něj další hrana. [9]

Jelikož je znalostní systém určen spíše pro čtení a průchod než ukládání či editování znalostí, byl graf časově zaměřen na průchod. Implementace inferenčního mechanismu byla vytvořena za pomocí setříděného seznamu podle identifikátoru znalostí. Ukládání znalostí máme proto, co se procesorového času týče, pomalejší ve prospěch rychlosti prohledávání a nacházení dalších znalostí, které má v tomto ohledu větší prioritu. Pro zápis asymptotické složitosti se používá Landauovův symbol O.

Nechť existují dvě funkce $f(n)$ a $g(n)$ definované na množině přirozených čísel. Existuje-li konstanta $K \in \mathbb{R}$ a $n_0 \in \mathbb{N}$ takové, že všechna $n \in \mathbb{N}$ jsou větší rovno n_0, potom platí

$$f(n) \leq K|g(n)|$$

právě tehdy když

$$f(n) = O\big(g(n)\big).$$

Pro seznam byla vybrána datová struktura *ArrayList*, která pro řazení využívá algoritmus *quicksort*, jehož složitost dosahuje v průměru $O(n \log n)$, kde n reprezentuje počet prvků.

Vyhledávání znalostí bylo implementováno jako binární prohledávání nad seznamem. Časová průměrná složitost tohoto algoritmu je v tomto případě rovna $O(\log n)$. [9]

Procházení grafem zodpovídá právě tato část znalostního systému. Pomocí vyhodnocování jednotlivých odpovědí od uživatele, na základě znalosti/otázky, se hledá následující, která se uživateli zobrazí. Začíná se vždy znalostí reprezentující začátek sekce. Každá sekce může odkazovat na konečný výrok (default) nebo další sekci, která se začne procházet po jejím úplném vyhodnocení nebo odkazuje-li na ní znalost z této množiny. Před průchodem se vytvoří seznam identifikátorů znalostí, pro které v rámci této skupiny uzlů neexistují znalosti na ně odkazující. Tento seznam tedy obsahuje množinu vrcholů podgrafů pro jednotlivé vyhodnocení. Pokud se při vyhodnocování jednoho z podgrafu nedosáhne koncové znalosti, je vybrán další klíč znalosti z tohoto seznamu. Podle tohoto klíče se hledá v množině všech znalostí výše zmíněným algoritmem. Prochází se sekcemi do té doby, dokud se nedospěje ke koncové znalosti.

Inferenční mechanismus si uchovává seznam jednotlivých odpovědí s vysvětlením. Můžeme mluvit o zabudovaném vysvětlovacím mechanismu. Schopnost systému mít vysvětlení u jednotlivých zodpovězených znalostí má účel takový, aby se neinformovaný uživatel dozvěděl, jak se přesně přišlo na jeho výsledek a z čeho tak vyplynulo. Takové výklady jsou nesmírně důležité, aby osoba využívající aplikaci pronikla do dané problematiky bez jakýkoliv předpokladů, že má vědomosti o tom, jakým způsobem systém vyhodnocuje dané úlohy a proč. Vysvětlovací mechanismus je zaveden v řetězcové podobě. U každé znalosti, na kterou se odpoví a má vysvětlení pro uživatele, se eviduje interpretace znalosti, odpověď na ni a vyložení vyhodnocení nebo účelu položení.

2.3.3 Uživatelské rozhraní

Uživatelské rozhraní je vytvořeno pomocí *JavaFX FXML*, značkovacího jazyku odvozeného z XML. Jedná se o platformu postavenou na bázi platformy Javy, která vytváří grafické rozhraní. Pro tvorbu designu znalostního systému bylo využito open source aplikace *JavaFX Scene Builder 8.5.0*. Tato aplikace umožňuje vytvářet grafické rozhraní pomocí operace *„Drag and drop“* neboli přetahováním komponent na zobrazované okno, bez nutnosti psaní ve výše zmíněném značkovacím jazyce.

Systém je rozdělen na dvě rozhraní. Na část přizpůsobenou pro experta a část pro běžného uživatele. Přechod mezi těmito dvěma oddělenými režimy se uskutečňuje skrze nabídku v menu. Menu dále obsahuje položky jako je načíst jinou bázi znalostí, či nápovědu, která

obsahuje položku pro krátký popis aplikace a zobrazení příručky. Režim pro experta je zaměřena na operace pro znalostní bázi jako například úprava, smazání či přidání znalosti nebo proměnné. Expert vidí výpis celé báze znalostí a pro detail si vybere vždy jednu znalost nebo proměnnou, se kterou může provádět další úkony. Běžný uživatel má prostředí jednodušší. Spouští jen vyhodnocování. Při vyhodnocování má uživatel pouze tyto možnosti: Ukončit vyhodnocování, odpovídat na otázky, dokud nedojde k závěrečnému výsledku a dojde-li k poslední znalosti, má možnost si zobrazit v okně výpis vysvětlovacího modulu, který si může uložit na dané zařízení v textovém souboru. Oba režimy byly vytvářeny s cílem udělat aplikaci co nejintuitivněji pro všechny z možných rolí uživatelů.

Grafika prostředí aplikace byla cílená tak, aby odpovídala k bázi znalostí zaměřené na problematiku zařazení do evidence uchazečů o zaměstnání a výpočtu podpory v nezaměstnanosti.

3 APLIKACE A VÝSLEDKY

3.1 Zařazení do evidence uchazečů o zaměstnání

Tato a následující podkapitola se bude věnovat tématu, na které je zaměřena báze znalostí vytvářeného systému. Tedy o zařazení do evidence uchazečů o zaměstnání (dále jen „evidence“) a výpočtu podpory v nezaměstnanosti (dále jen „podpory“). Kapitoly se budou věnovat primárně nejčastěji se vyskytujícím situacím. Popíšeme si, co uživatel musí splňovat pro každý z možných výsledků a dále definujeme výpočet podpory. Pro vyhodnocování možnosti zařazení do evidence máme jen tři možnosti. Nezařadit, zařadit bez možnosti nároku na podporu a zařadit s možným nárokem na podporu v nezaměstnanosti.

Uživatel, který má nárok na zařazení do evidence, musí splňovat určité podmínky. Není osobou samostatně výdělečně činnou a není ve služebním poměru nebo v pracovně právním vztahu. Tato podmínka se nevztahuje na takový služební poměr, pracovně právní vztah nebo dohodu o pracovní činnosti, kde tato činnost není ohodnocena měsíční odměnou vyšší, než je polovina minimální mzdy (hovoříme o takzvaném nekolidujícím zaměstnání). Za předpokladu, že je výdělek nižší, tak má nárok být zařazen, ale nemá oproti ostatním dále nárok na podporu v nezaměstnanosti (to platí i u nulového výdělku). Pro rok 2021 je tato částka určena na 7 600 Kč. Dále nesmí vykonávat činnost nuceného správce, prokuristy nebo likvidátora. Není členem zastupitelstva územního samosprávného celku, který pobírá odměny jako uvolněný člen, pěstounem, kterému je vyplácena odměna pěstouna, soudcem, poslancem nebo senátorem. Pokud je uživatel student denního studia, musí splňovat podmínku, že v posledních dvou letech v délce alespoň 12 měsíců byl zaměstnán nebo měl jinou výdělečnou činnost s účastí na důchodovém pojištění. Zařazena do evidence nemůže být fyzická osoba, která je uznána dočasně neschopnou práce, pobírá peněžitou pomoc v mateřství, je 6 týdnů po porodu, je plně invalidní ve třetím stupni, pokud není schopna pracovat za zcela mimořádných podmínek, či je ve výkonu trestu odnětí svobody, zabezpečovací detenci nebo je ve vazbě. [8]

3.2 Podpora v nezaměstnanosti

Předpoklady, které uživatel musí splňovat, má-li mít nárok na podporu v nezaměstnanosti, navazují na kapitolu viz výše. Nutná podmínka pro každého uchazeče, který tento nárok splňuje, je být v evidenci uchazečů o zaměstnání. Podmínky už přímo týkající se nároku na podporu jsou popsány níže.

Uživatel pro splnění nároku musí mít v rozhodném období, jehož délka je stanovena na dva roky, být v zaměstnání nebo mít jinou výdělečnou činnost s účastí na době důchodového

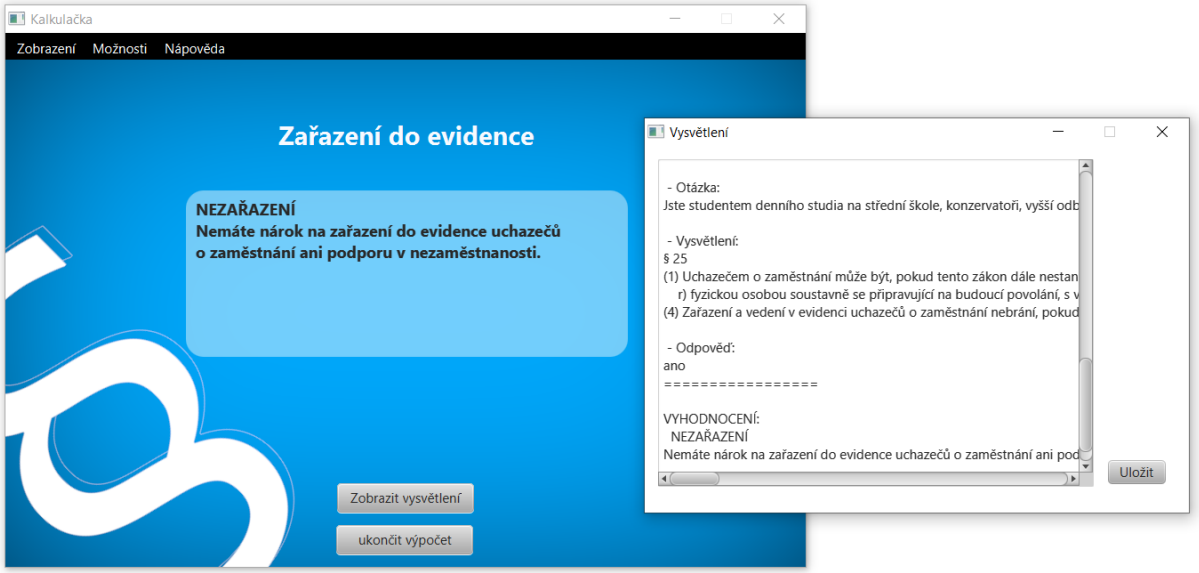
pojištění. Do této doby se započítává i doba náhradní. Za náhradní dobu se považuje taková doba, ve které uživatel pečoval o dítě ve věku do 4 let, pečoval o fyzickou osobu závislou na pomoci ostatních alespoň stupně II (středně těžká závislost) anebo pobíral plný invalidní důchod pro invaliditu třetího stupně. Zároveň musí platit, že uživatel neskončil v posledním zaměstnání z důvodu porušení povinnosti z pracovněprávních předpisů, které se vztahují na jeho práci, nebo porušením jiné povinnosti zvláště hrubým způsobem. [8]

Pokud uživatel již podporu v minulosti pobíral a zároveň splňuje požadavky viz výše, potom má nárok na podporu za následujících předpokladů. Byl po uplynutí podpůrčí doby v zaměstnání nebo měl jinou výdělečnou činnost s účastí na důchodovém pojištění po období alespoň 6 měsíců. Neuplynula-li mu celá podpůrčí doba a měl po skončení činnost s účastí na důchodovém pojištění po časovém úseku alespoň 3 měsíců, má opět nárok na celou podpůrčí dobu, jinak má nárok na zbývající část podpůrčí doby, kterou nevyčerpal v minulé evidenci, popřípadě minulých evidencí. [8]

Maximální výše podpory se odvozuje od průměrné mzdy v národním hospodářství za první až třetí čtvrtletí předcházejícího kalendářního roku (dále jen „průměrná mzda“). Pro rok 2021 tato částka odpovídá 34 611 Kč. Maximální výše podpory činí její 0,58násobek. Doba poskytování podpory, tzv. podpůrčí doba, se určuje dle věku uživatele. Do věku 50 let je stanovena na 5 měsíců, pro věk mezi 50 a 55 lety je to 8 měsíců a nad 55 let je určena 11 měsíci. V prvních dvou měsících z této doby se pobírá 65 %, další dva měsíce 50 % a zbytek podpůrčí doby 45 % z průměrného měsíčního čistého výdělku posledního zaměstnání. Je zde výjimka, kdy uživatel ukončil zaměstnání sám nebo dohodou se zaměstnavatelem, pak je to 45 % po celou podpůrčí dobu. Má-li nárok na výsluhový příspěvek, náleží mu rozdíl tohoto příspěvku a podpory, která by mu náležela. Je-li tento rozdíl nekladný, na podporu nemá nárok. V případě splnění podmínky doby předchozího zaměstnání započítáním náhradní doby a tato doba byla poslední vykonávaná činnost, podpora činí v období prvních dvou měsíců 0,15násobek, pro další dva měsíce 0,12násobek a po zbývající podpůrčí dobu 0,11násobek průměrné mzdy. Začátek vyplácení podpory se posouvá o počet násobku průměrného měsíčního výdělku uživatele z vyplaceného odstupného, odbytného či odchodného. [8]

3.3 Výsledky

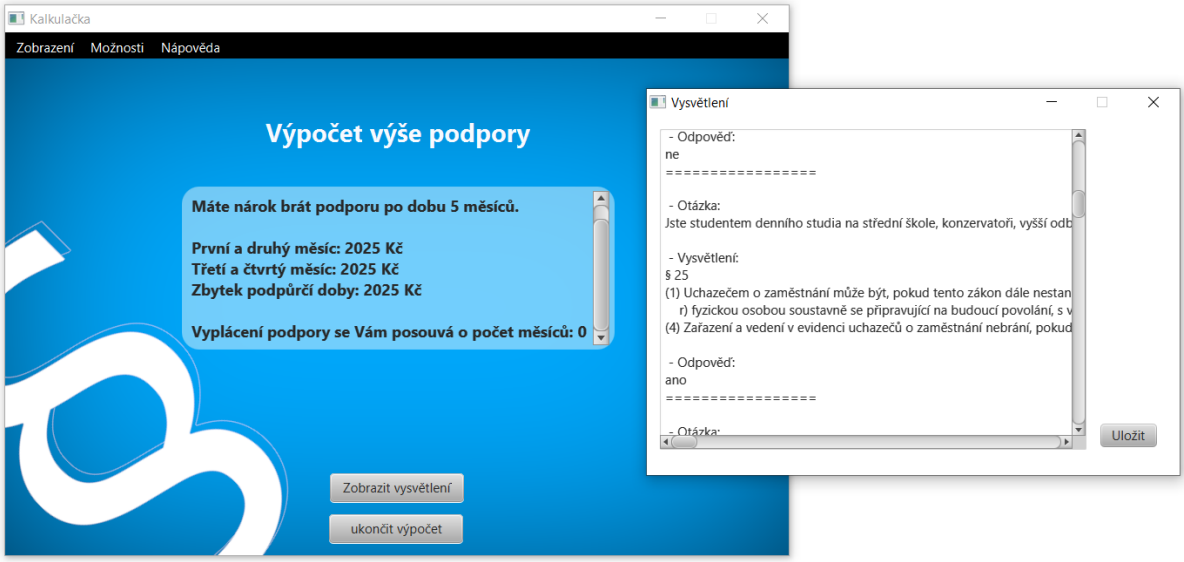
V této kapitole si na modelových příkladech ukážeme výsledky aplikace. Pro první příklad máme muže ve věku 24 let, který je studentem vysoké školy denního studia.



Obrázek 5: Výsledek prvního příkladu

Aplikace pro daný příklad vyhodnotila, že student nemá nárok na zařazení do evidence uchazečů o zaměstnání dle § 25 odst. 1, písm. r) zákona č. 435/2004 Sb., o zaměstnanosti.

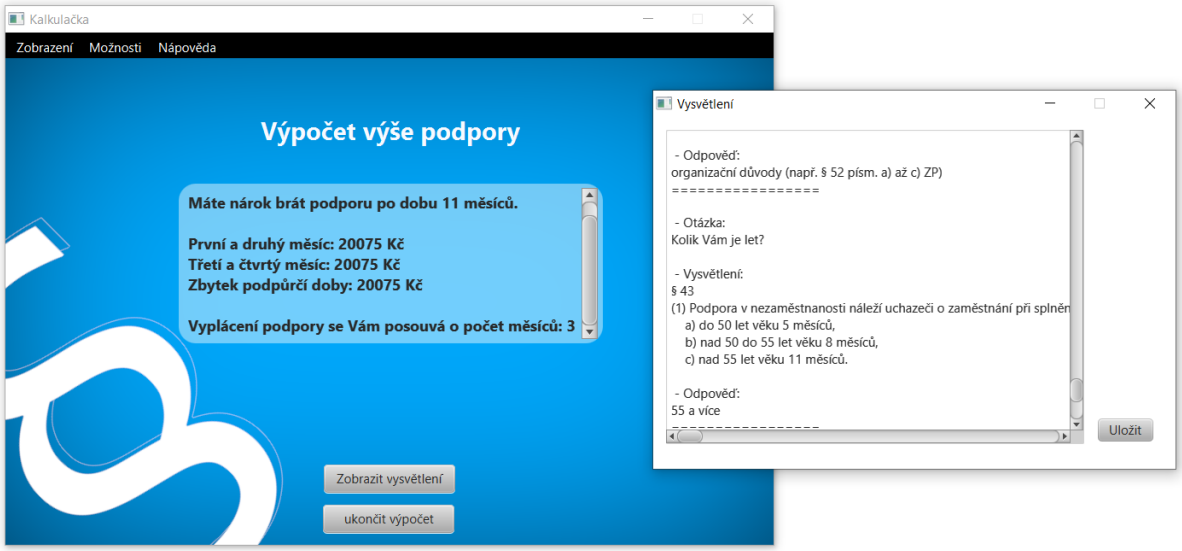
Ve druhém případě máme ženu ve věku 26 let, studentku vysoké školy denního studia s ukončenou dohodou o pracovní činnosti s účastí na době důchodového pojištění v posledních dvou letech 13 měsíců, kde měla průměrný měsíční čistý výdělek 4 500 Kč. Tuto činnost ukončila dohodou.



Obrázek 6: Výsledek druhého příkladu

V tomto případě tato studentka má nárok dle § 25 odst. 4 zákona č. 435/2004 Sb., o zaměstnanosti.

Pro třetí příklad máme muže, kterému je 56 let. Ve firmě odpracoval 15 let a skončil z organizačních důvodů (jeho místo bylo zrušeno). Jeho průměrný měsíční čistý výdělek byl 65 450 Kč. Po ukončení zaměstnání obdržel podle délky pracovního poměru tří měsíční odstupné.



Obrázek 7: Výsledek třetího příkladu

Tento muž bude pobírat maximální možnou podporu po dobu 11 měsíců, která se mu posouvá o 3 měsíce z důvodu nároku na odstupné.

ZÁVĚR

Hlavním cílem této bakalářské práce bylo popsat a vytvořit znalostní systém aplikovaný na konkrétní příklad, v tomto případě jde o problematiku zařazení do evidence uchazečů o zaměstnání a podpory v nezaměstnanosti dle zákona č. 435/2004 Sb., o zaměstnanosti.

První díl této práce se zaměřil na teorii znalostních systémů. Mým cílem bylo nastínit problematiku těchto systémů a jejich charakteristiky. První část byla věnována úvodu do znalostních systémů a architektury. Dále se znalostní systémy rozdělily na jednotlivé typy podle rysu řešených úloh. Uvedl jsem zde šest hlavních tříd. A to diagnózu, interpretaci, monitorování, plánování, návrh a predikci. Pro příklad reálného využití v praxi byly popsány příklady nejznámějších znalostních systémů, jimiž jsou Mycin, Teiresias, Prospector a R1/Xcon. Následné části se týkaly již zmíněné architektury. Každá část byla popsána v samostatné kapitole, kde se blíže specifikovala. Nejprve se popisoval způsob reprezentace znalostí. Zde jsem dal příklad tří nejpoužívanějších způsobů, jako jsou pravidla, rámce či sémantické sítě. Poté jsem se zabýval metodami vyhodnocování závěrů pomocí inferenčního mechanismu. Poslední popisovaná část znalostního systému byla věnována vysvětlovacímu modulu, který slouží pro objasnění postupů usuzování. Konec teoretické části byl zaměřen na neurčitost, konkrétně na nástroj fuzzy logiky.

V praktické části bakalářské práce jsem řešil tvorbu konkrétní aplikace. Výsledkem byl znalostní systém, který jsem vytvořil v prostředí NetBeans, kde jsem jej naprogramoval za pomocí programovacího jazyka Java. Po vytvoření jsem začal tento prázdný systém plnit znalostmi ve spolupráci s expertkou na pracovněprávní předpisy. Vzniklo tak 33 otázek, které se týkají zařazení do evidence uchazečů o zaměstnání a výpočtu podpory v nezaměstnanosti. Všechny řešené možnosti ve znalostní bázi byly popsány ve třetím dílu mé bakalářské práce, kde následně byly otestovány konkrétními příklady.

POUŽITÁ LITERATURA

[1] PROVAZNÍK, Ivo a Jiří KOZUMPLÍK. *Expertní systémy*. Brno: Vysoké učení technické, 1999. Učební texty vysokých škol. ISBN 80-214-1486-3.

[2] BERKA, Petr, Petr JIRKŮ a Jiřina VEJNAROVÁ. *Expertní systémy*. Skripta. Praha: Vysoká škola ekonomická, 1998. ISBN 80-707-9873-4. Dostupné také z: https://sorry.vse.cz/~berka/4IZ229/.

[3] NAVARA, Mirko a Petr OLŠÁK. *Základy fuzzy množin*. Vyd. 2., přeprac. Praha: Nakladatelství ČVUT, 2007. ISBN 978-80-01-03668-6.

[4] CELBOVÁ, Iva. Úvod do problematiky expertních systémů. *Ikaros* [online]. 1999, ročník 3, číslo 8 [cit. 2020-11-13]. urn:nbn:cz:ik-10378. ISSN 1212-5075. Dostupné z: http://ikaros.cz/node/10378.

[5] DVOŘÁK, Jiří. *Expertní systémy* [online]. Brno: Vysoké učení technické v Brně, 2004 [cit. 2020-11-13]. Dostupné z: http://www.uai.fme.vutbr.cz/~jdvorak/Opory/ExpertniSystemy.pdf.

[6] WAGNER, William P. Trends in expert system development: A longitudinal content analysis of over thirty years of expert system case studies. *Expert Systems with Applications* [online]. 2017, 76, 85-96 [cit. 2020-11-13]. ISSN 09574174. Dostupné z: doi: 10.1016/j.eswa.2017.01.028.

[7] THAKER, Shaily a Viral NAGORI. Analysis of Fuzzification Process in Fuzzy Expert System. *Procedia Computer Science* [online]. 2018, 132, 1308-1316 [cit. 2020-11-13]. ISSN 18770509. Dostupné z: doi: 10.1016/j.procs.2018.05.047.

[8] *Pracovněprávní předpisy: Zaměstnanost: zákon o zaměstnanosti; Úřady práce, odbory, kolektivní vyjednávání, ochrana zaměstnanců při platební neschopnosti zaměstnavatele, inspekce práce.* Sagit, a. s., podle stavu k 15. 7. 2019, 288 stran. Ostrava: Sagit, 2019. ÚZ. ISBN 978-80-7488-359-0.

[9] VIRIUS, Miroslav. *Základy algoritmizace*. Vyd. 2., přeprac. Praha: Česká technika – nakladatelství ČVUT, 2008. ISBN 978-80-01-04003-4.

[10] TRLIFAJOVÁ, Kateřina, Daniel VAŠATA a České vysoké učení technické v Praze. Matematická logika. Praha: České vysoké učení technické v Praze, 2013. ISBN 978-80-01-05342-3.

PŘÍLOHY

PŘÍLOHA A – CD

Tato příloha obsahuje tuto práci v PDF formátu a zdrojové kódy aplikace.

PŘÍLOHA B – PŘÍRUČKA

Tato příloha obsahuje uživatelskou příručku pro vytvořený znalostní systém.

Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Uživatelská příručka znalostního systému pro výpočet podpory v nezaměstnanosti

Luděk Letáček

2021

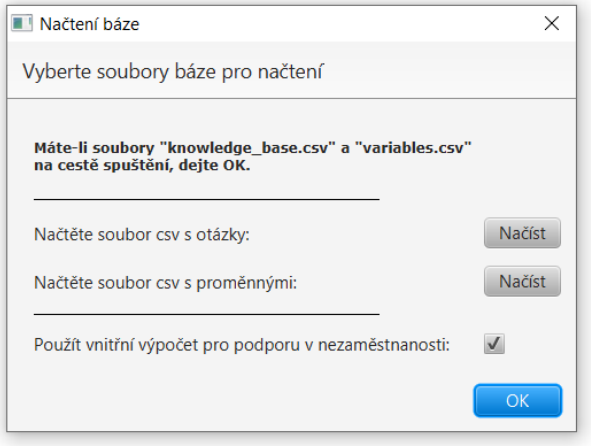
OBSAH

SEZNAM OBRÁZKŮ

1 SPUŠTĚNÍ APLIKACE

1.1 Načtení báze

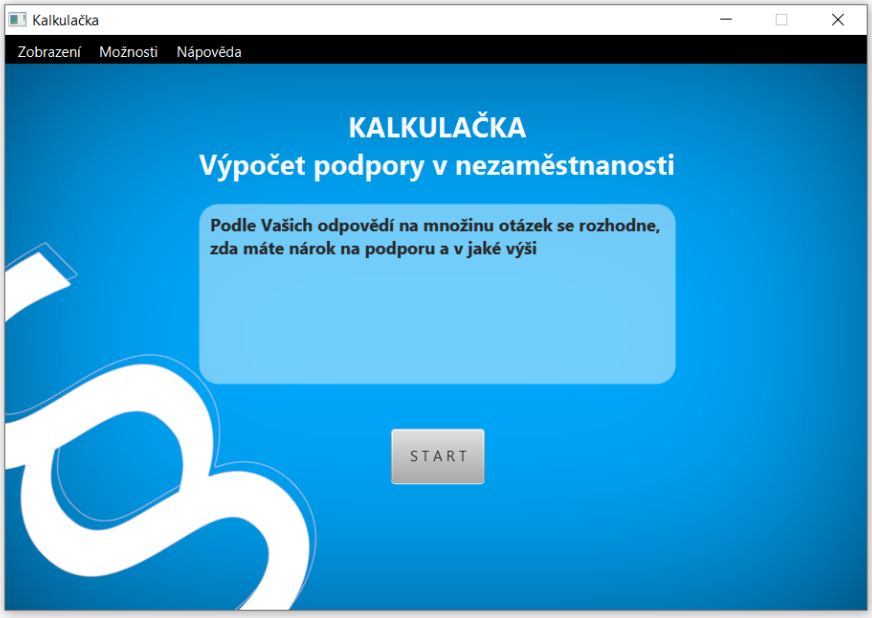
Po spuštění aplikace se otevře nabídka pro načtení proměnných a znalostí ze souboru. Soubor s proměnnými a znalostmi jsou uloženy v souborech typu *„csv“* s pevnou, sobě vlastní, datovou strukturou. V případě, že na cestě, kde jsme soubor spustili, existují soubory *„knowledge_base.csv“* a *„variables.csv“*, nemusíme je načítat ručně a stačí pouze kliknout na tlačítko *„OK“*. Pokud báze znalostí nevyužívá vnitřní výpočet podpory v nezaměstnanosti, odškrtneme zaškrtávací pole, aby se nevytvořily chybějící proměnné spojené právě s tímto výpočtem. Chceme-li načíst vlastní soubor znalostí a proměnných, vybereme je prostřednictvím tlačítek *„Načíst“*. V případě, že nenačtete žádnou bázi, či dojde k chybě (např. neplatný formát, nenalezen výchozí soubor), pokračuje se na úvodní obrazovku s tím, že operace jako je editace či spuštění kalkulačky nebude povolena. Další způsob načtení existující báze je možný viz kapitola 1.3.



Obrázek 1: Načtení báze znalostí

1.2 Úvodní obrazovka

Úvodní obrazovka poskytuje pohled na uživatelský režim viz kapitola 3.



Obrázek 2: Uživatelský režim

1.3 Menu

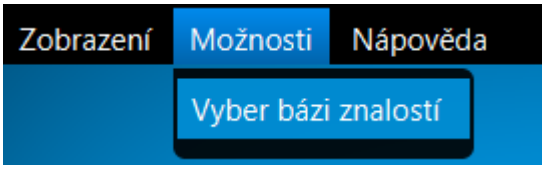
1.3.1 Zobrazení

Položka nabídky zobrazení se mění na základě režimu, ve kterém se nacházíme. Slouží k přepínání mezi *„Expertním“* a *„Uživatelským“* režimem.

Obrázek 3: Menu zobrazení

1.3.2 Možnosti

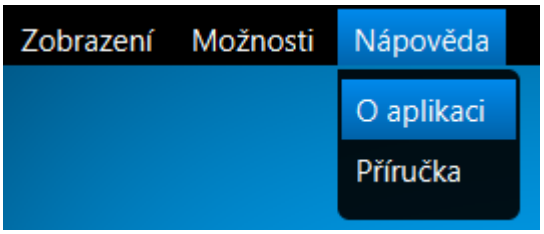
Chceme-li změnit bázi, načteme ji pomocí *„Vyber bázi znalostí“* skrze nabídku *„Možnosti“*.



Obrázek 4: Menu načtení báze

1.3.3 Nápověda

Nápověda má dvě části. Část *„O aplikaci“* a *„Příručka“*. První možnost zobrazí informaci o aplikaci, která blíže specifikuje uživateli její účel. Druhá část zobrazí tento dokument.



Obrázek 5: Menu nápovědy

2 EXPERTNÍ REŽIM

V expertním režimu máme možnost editovat načtenou bázi. Můžeme odebírat, přidávat, upravovat či zobrazovat znalosti, ale i proměnné. Na některých proměnných je postavená funkčnost celé aplikace, takže takové odebírat nelze.

2.1 Editace znalostí

V listu pro editaci znalostí můžeme nejen měnit, vytvářet a zobrazovat znalosti, ale také některé z nich nastavovat jako vstupní body. Vstupní bod slouží pro nastavení první množiny otázek. Takovými body mohou být pouze typy znalostí, představující reprezentanta množiny otázek, tzv. sekce.

Jelikož aplikace může sloužit jako kalkulačka pro výpočet podpory, potřebujeme znát bod, kdy máme zadané veškeré nutné informace pro výpočet od uživatele. Takovým bodem může být informativní okno, konec či znalost definující sekci.

2.1.1 Druhy znalostí

Máme 8 druhů znalostí.

- Otázka ANO/NE,
- otázka s možností zadání čísla (nemusí být požadována odpověď),
- otázka s možností výběru (nemusí být požadována odpověď), kde každý výběr reprezentuje číslo,
- typ informativního okna (možnost větvení na základě hodnoty v proměnné),
- položka sekce sloužící jako vstupní bod pro vyhodnocení množiny znalostí do ní spadající,
- nastavení proměnné na inicializační hodnotu,
- inkrementaci proměnné o hodnotu 1,
- konečný bod, sloužící jako vyhodnocení.

Druh znalostí nejde měnit u již vytvořených.

2.1.2 Vkládání nové znalosti

Máme-li vybranou nějakou z otázek, ukončíme její editaci tlačítkem *„Zrušit výběr“*. Zde nejprve vybereme typ znalosti a poté nastavíme. Pro lokální přidání použijeme *„Přidej“* a do souboru uložíme pomocí *„Ulož do csv“*.

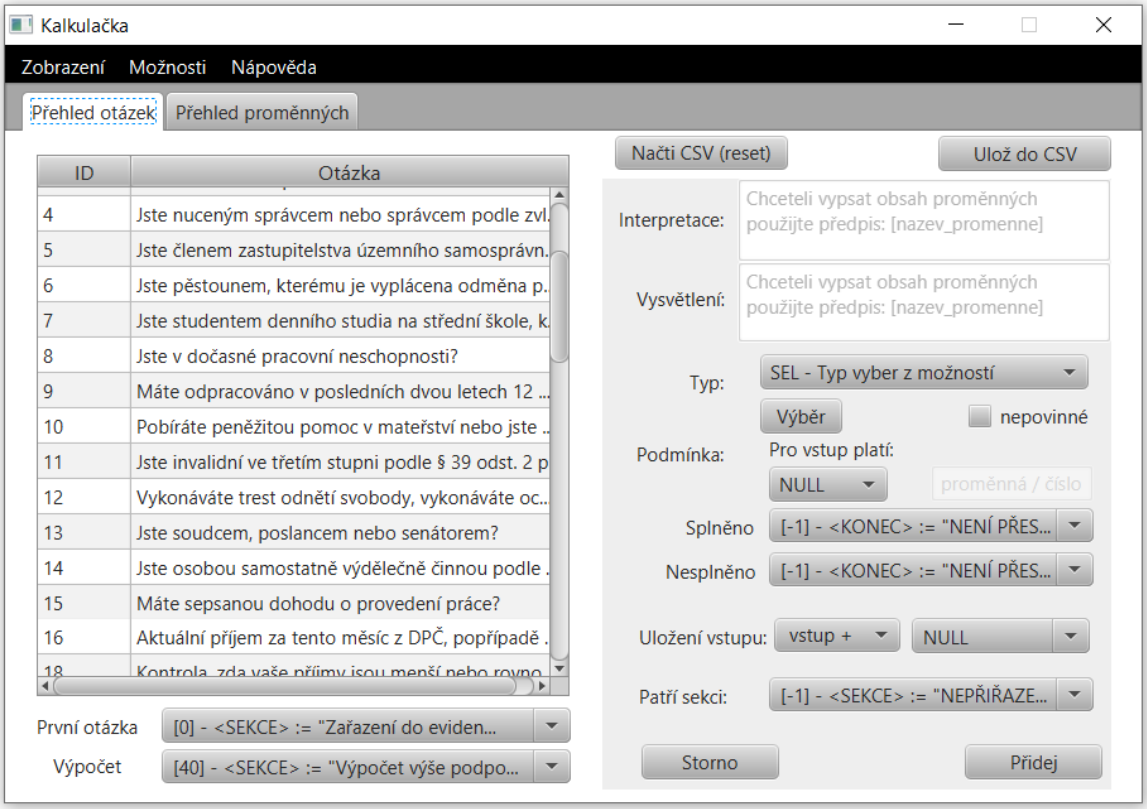
U znalosti se nastavuje *„Interpretace“*, která slouží pro viditelný text (např. pro otázku). Pole *„Vysvětlení“* má dvě možnosti použití, pokud se jedná o konečný bod, pak její chování odpovídá interpretaci znalosti, která se v tomto případě nenastavuje. Pro ostatní případy slouží jako vysvětlení, proč se tato znalost vyskytuje a zároveň slouží uživateli pochopit výsledek, kterého docílil. Chceme-li zobrazit již nastavenou proměnnou, dáme ji do hranatých závorek (např. *[proměnná]*). Pro jiné důvody než zobrazení proměnných, se hranaté závorky nesmí použít.

Typ otázky by se měl vybírat vždy jako první, jelikož se od ní vyvozují různá nastavení. Pokud to typ dovoluje, vybírá se, zda je povinné odpovídat. Není-li povinná a zároveň odpověď není získána, je vyhodnocena znalost jako splněná. Typ výčtové znalosti má oproti ostatním tlačítko *„Výběr“*. Zde se nastaví možnosti, kterým se přiřadí číslo pro další zpracování aplikací.

Každá znalost, která má být použita, musí patřit pod nějakou sekci, kromě definice sekce a konečného výroku. Do navazujících znalostí definující její vlastnost, se vybírají otázky, které jsou v rámci stejné sekce, dále vyhodnocení nebo reprezentant další sekce. Pokud není v rámci podmnožiny otázek jednoznačné, jak určit vyhodnocení, vybere se předpřipravená hodnota představující nejednoznačného konce. Systém vybere sám další podmnožinu otázek ze stejné sekce. Pokud v dané sekci nenajde řešení, pokračuje se navazující sekcí, která je u definice této sekce zvolena jako následující. Poslední sekce odkazuje na jeden z verdiktů nebo neodkazuje na nic, je-li si expert jist, že i přesto se ke konci vždy dospěje.

Podmínka slouží pro rozhodování, na kterou další znalost se přistoupí. Jsou zde možnosti porovnání pro hodnotu/podmínku v textovém okně a pro vstup, resp. proměnnou (pouze u typu „informativního okna“): NULL (podmínka není nastavena a vždy bude znalost brána jako pravdivá), EQ (rovná se), NE (nerovná se), LT (je menší než), LE (je menší rovno než), GT (je větší než), GE (je větší rovno než). Pod touto definicí, se nachází výběr ze znalostí, na které se má přejít na základě pravdivosti této podmínky.

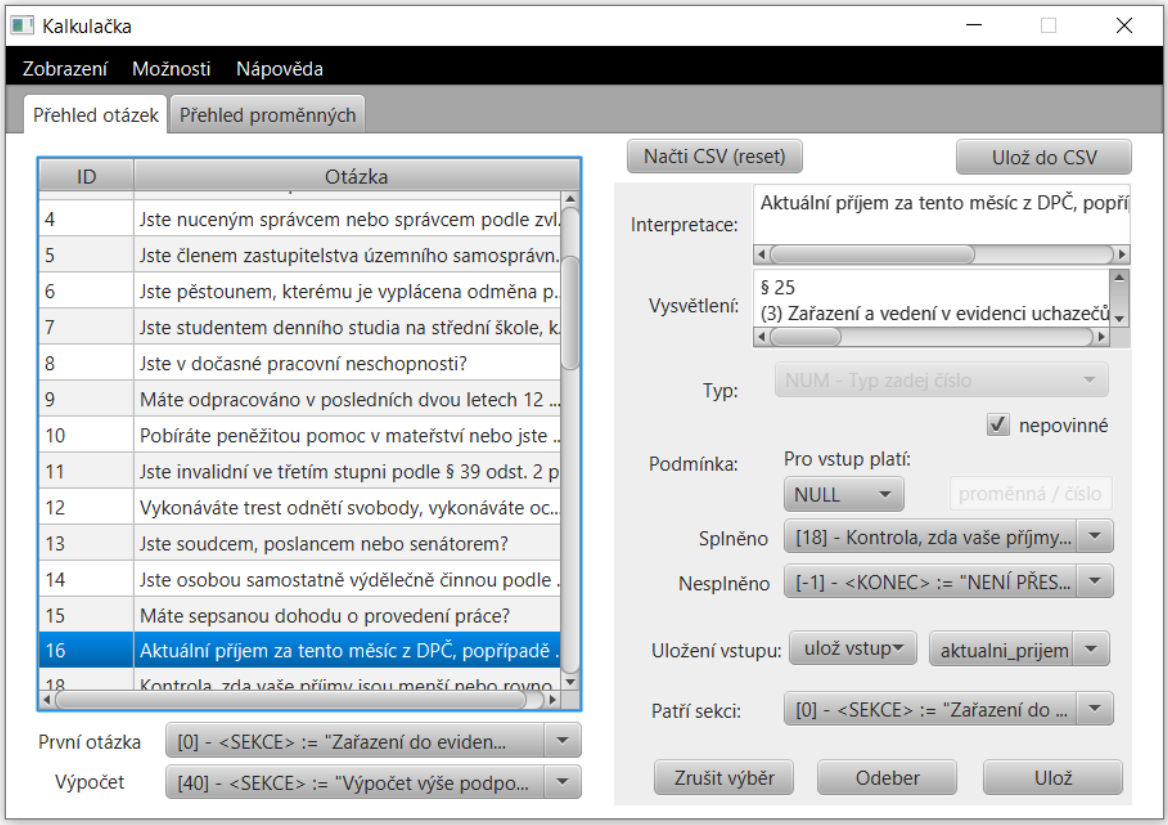
Vstup můžeme ukládat či jej nad proměnou zpracovat pomocí jedné ze základních operací, jako je sčítání, odčítání, násobení nebo dělení.



Obrázek 6: Vytvoření znalosti

2.1.3 Úprava existující znalosti

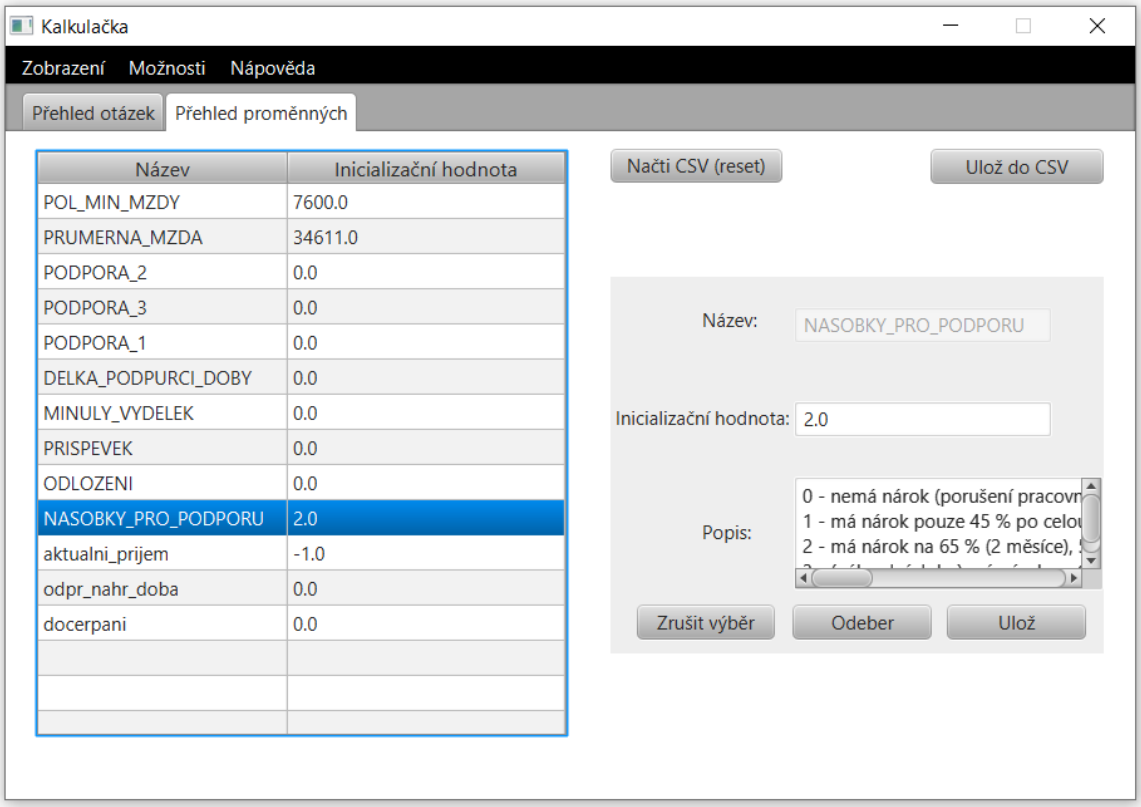
Pro úpravu konkrétní znalosti je ji třeba nejprve v tabulce vybrat. Na základě povolených položek k editaci změníme chování či znění znalosti. Chceme-li uložit úpravy, uložíme ji nejprve lokálně skrze tlačítko *„Ulož“* a poté propíšeme změny do souboru báze tlačítkem *„Ulož do csv“*. Pro odstranění použijeme *„Odstranit“* a ekvivalentně změníme soubor *csv* viz úprava.



Obrázek 7: Editace znalosti

2.2 Editace proměnných

U editace proměnných se používá stejný způsob jako u editace znalostí. Jediný rozdíl je, že pokud se edituje proměnná, na které je závislý program, bude expert upozorněn. Taková proměnná nemá povolení k odstranění. Název proměnné je brán jako identifikátor, proto nemá povoleno být změněn.



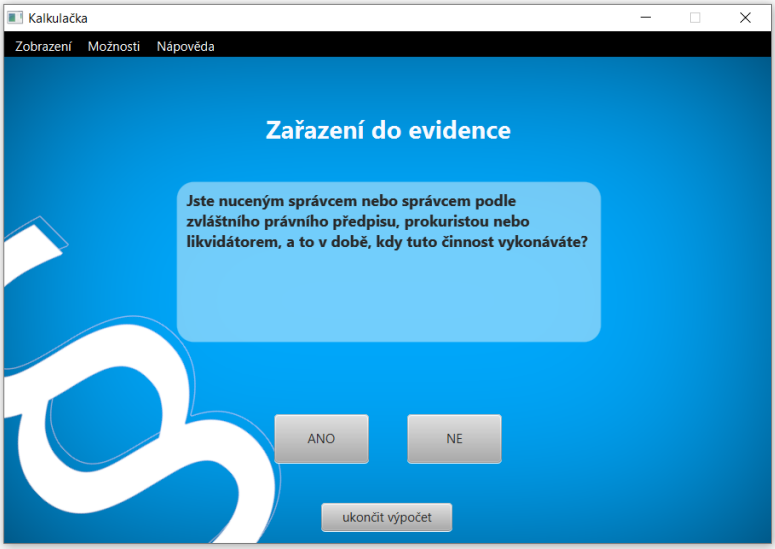
Obrázek 8: Editace proměnné

3 UŽIVATELSKÝ REŽIM

3.1 Spuštění kalkulačky

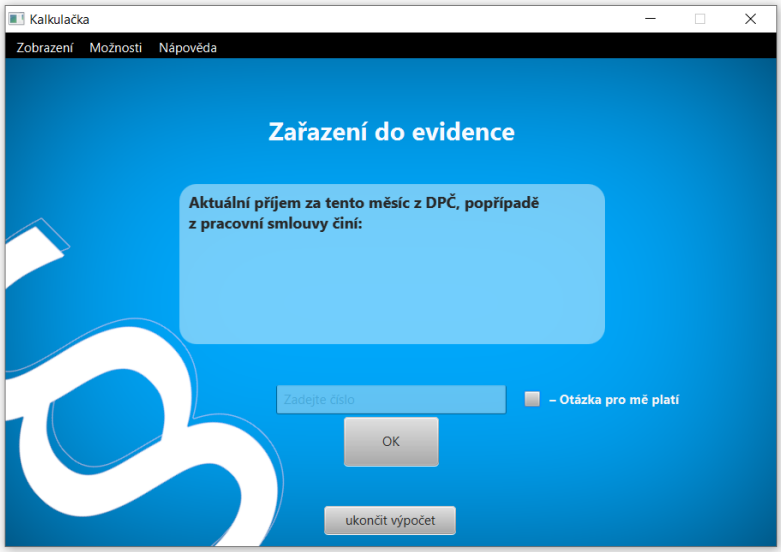
Pro spuštění kalkulačky stiskne uživatel tlačítko *„START“*. Pro ukončení je určeno tlačítko *„ukončit výpočet“*.

Na dichotomické otázky se odpoví *„ANO“* či *„NE“*.



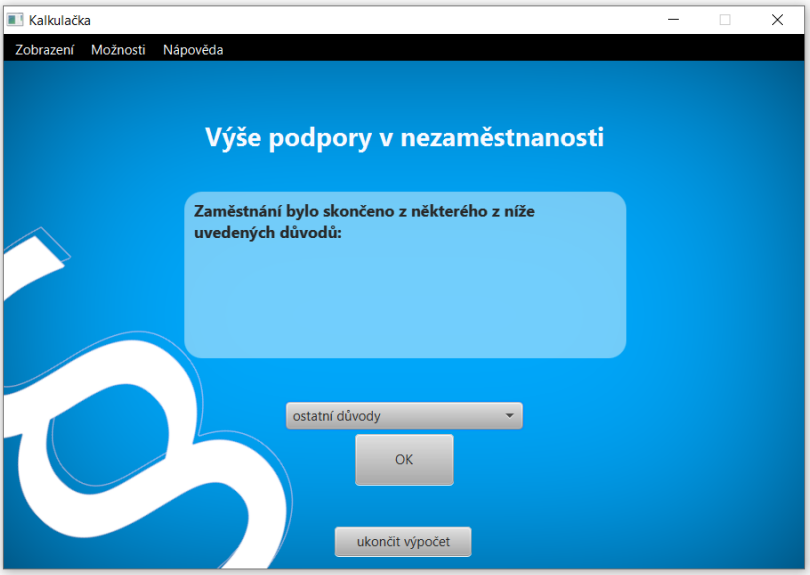
Obrázek 9: Dichotomická otázka

Na otevřenou otázku, ve které se vyžaduje pouze kladné číslo, může být nevyplněno pouze v případě, že není odpověď expertem vyžadována. Odpověď se přeskočí odškrtnutím *„Otázka pro mě platí“* (platí i u ostatních typů znalostí).



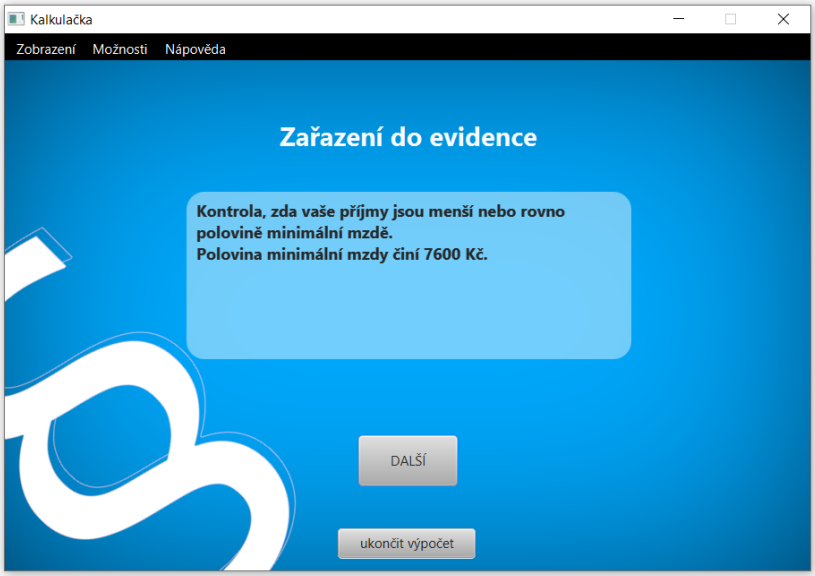
Obrázek 10: Otevřená otázka

U výčtové znalosti se vybere odpověď a pokračuje se na další pomocí *„OK“*.



Obrázek 11: Výčtová znalost

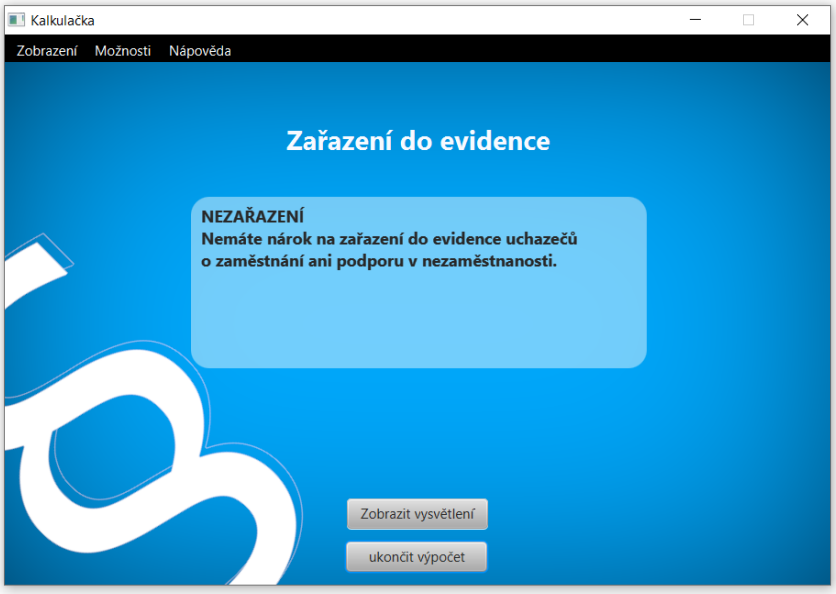
Pokud se zobrazí informativní okno, přejdete na následující znalost pomocí tlačítka *„DALŠÍ“*.



Obrázek 12: Informativní znalost

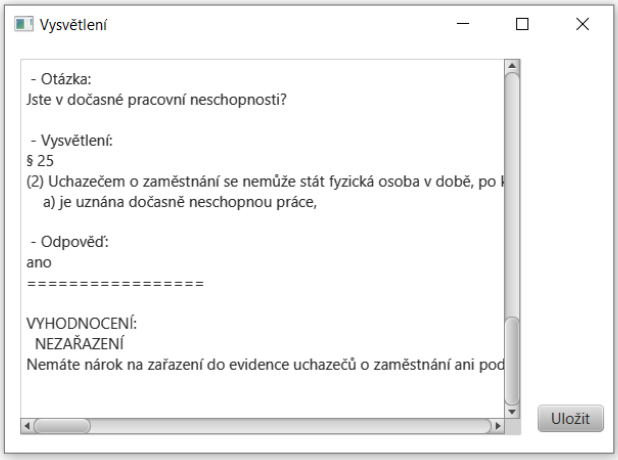
3.2 Vyhodnocení

U vyhodnocení má uživatel možnost ukončit výpočet nebo si zobrazit vysvětlení svého závěru.



Obrázek 13: Vyhodnocení

V případě zobrazení vysvětlení je zde možnost uložení v textovém souboru.



Obrázek 14: Vysvětlení