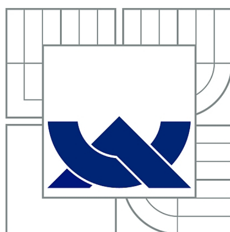
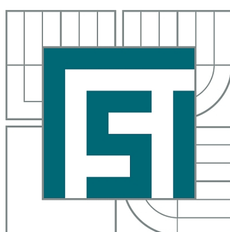


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA STROJNÍHO INŽENÝRSTVÍ
ÚSTAV AUTOMATIZACE A INFORMATIKY

FACULTY OF MECHANICAL ENGINEERING
INSTITUTE OF AUTOMATION AND COMPUTER SCIENCE

POUŽITÍ STRUKTURÁLNÍ METODY PRO ROZPOZNÁVÁNÍ OBJEKTŮ

USING STRUCTURAL METHOD FOR OBJECTS RECOGNITION

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE Bc. VÍT VALSA
AUTHOR

VEDOUCÍ PRÁCE prof. RNDr. Ing. JIŘÍ ŠŤASTNÝ, CSc.
SUPERVISOR

BRNO 2015

Abstrakt
Tato diplomová práce se zabývá možností využití strukturálních metod pro rozpoznávání objektů v obraze. Nejprve jsou popsány způsoby pro přípravu obrazu před samotným zpracováním. Vlastní jádro celé práce spočívá v kapitole 3, kde je podrobně rozebrán problém tvorby deformačních gramatik pro syntaktickou analýzu a jejich použití. Dále je věnován prostor syntaktickému analyzátoru interpretujícího deformační gramatiku. Závěr práce je zaměřen na testování navržených metod a jejich výsledky.

Summary
This diploma thesis deals with posibilities of using structural methods for recognition objects in a picture. The first part of this thesis describes methods for preparing the picture before processing. The core of the whole thesis is in chapter 3, where is analyzed in details the problem of the formation of deformation grammars for parsing and their using. In the next part is space for syntactic parser describing the deformation grammar. The conclusion is focused on testing the suggested methods and their results.

Klíčová slova
syntaktická analýza, bezkontextové gramatiky, deformační gramatika, klasifikace obrazu, detekce hran

Keywords
syntax analysis, context-free grammar, grammar deformation, image classification, edge detection

VALSA, V.*Použití strukturální metody pro rozpoznávání objektů*. Brno: Vysoké učení technické v Brně, Fakulta strojního inženýrství, 2015. 56 s. Vedoucí prof. RNDr. Ing. Jiří Šťastný, CSc.

Prohlašuji, že jsem diplomovou práci na téma Použití strukturální metody pro rozpoznávání objektu vypracoval samostatně s použitím odborné literatury a pramenů, uvedených na seznamu, který tvoří přílohu této práce.

Bc. Vít Valsa

Děkuji tímto prof. RNDr. Ing. Jiřímu Šťastnému, CSc. za cenné připomínky a rady při vypracování diplomové práce.

Bc. Vít Valsa

Obsah

1 Úvod 3

2 Postup zpracování obrazu při rozpoznávání objektů 5
2.1 Snímání a digitalizace obrazu 5
2.1.1 Snímání 5
2.1.2 Digitalizace 5
2.2 Předzpracování 5
2.3 Segmentace 6
2.3.1 Prahování 7
2.3.2 Detekce hran 8
2.3.3 Cannyho hranový detektor 9
2.4 Popis 10
2.4.1 Gramatiky 10
2.4.2 Chomského klasifikace gramatik 10
2.4.3 Návrh primitiv 12
2.5 Kvalifikace 12
2.5.1 Syntaktická analýza 12
2.5.2 LL(k) syntaktický analyzátor 15

3 Deformační gramatiky 17
3.1 Konstrukce deformační gramatiky 18
3.1.1 Výchozí komponenty 18
3.1.2 Deformace komponenty 19
3.1.3 Pravidla deformačních gramatik 19
3.2 Generátor gramatik 21
3.3 Deformační gramatika 22
3.4 Překladač 23
3.4.1 Lexikální analýza 25
3.4.2 Syntaktická analýza 26
3.4.3 Konstrukce syntaktického analyzátoru 29

4 Návrh systému pro rozpoznávání objektů 31
4.1 Aplikace Recognizer 31
4.2 Předzpracování scény 31
4.2.1 Prahování 32
4.3 Nalezení komponent 32
4.3.1 Detekce hran 33
4.3.2 Vyhledání komponent 34
4.3.3 Parsování komponent 38
4.3.4 Zajištění invariancí vůči rotaci 38
4.3.5 Rozpoznání komponent 39

5 Možnosti automatické tvorby algoritmů pro popis a rozpoznání objektů 41
5.1 Generátor řídícího kódu 41
5.2 Teoretická aplikace 42

6 Popis testovacího prostředí **43**

7 Zhodnocení testovacích výstupů **47**
7.1 Vyhodnocení výsledků 51
7.2 Možná vylepšení 52

8 Závěr **53**

A Obsah přiloženého CD **56**

1. Úvod

Rozpoznávání objektů v obraze je v dnešní době velmi využívaným nástrojem. Nachází uplatnění v nejrůznějších odvětvích. Problémem však může být náročnost vyhledání konkrétní struktury v obraze. Tato práce je zaměřena na oblast vyhodnocování obrazů pomocí strukturálních metod.

Začátek práce je věnován obecným postupům pro přípravu scény k dalšímu zpracování. Jsou zde zmíněný základní metody pro rozdělení obrazu na relevantní části. Také je zde rozebrána problematika popisu objektů v obraze za použití syntaktických metod a z toho vyplývající obecná syntaktická analýza.

Hlavní částí práce je kapitola následující. Pojednává o základním cíli práce, tvorbě deformační gramatiky. Je v ní detailně popsáno navržené řešení a jeho algoritmické zpracování, postup vytvoření syntaktického analyzátoru a jeho činnosti.

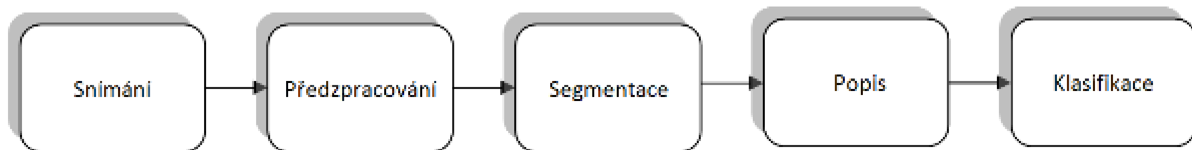
Čtvrtá a pátá kapitola pojednávají o realizaci testovací aplikace a jejím teoretickém využití.

Závěrečné kapitoly jsou věnovány popisu testovacího prostředí a vyhodnocení výsledků navržené metody. Při posuzování výsledků byl brán hlavní ohled na časovou náročnost zpracování a na úspěšnost rozpoznávání komponent.

2. Postup zpracování obrazu při rozpoznávání objektů

Detekce různých objektů a tvarů v digitálním obraze je nedílnou součástí operací zahrnujících práci s obrazem. Uplatňuje se v nejrůznějších odvětvích od lékařství (zpracování struktur v tomografech) přes bezpečnost (rozpoznávání registračních značek) po osobní sektor, kde u fotoaparátu a kamer slouží například k rozpoznávání obličejů a úsměvu.

Vlastní průběh zpracování a rozpoznávání obrazu reálného světa obvykle rozdělujeme do několika základních kroků [3]. Posloupnost těchto kroků je znázorněna na obrázku 2.1.



Obrázek 2.1: Postup zpracování obrazu

2.1. Snímání a digitalizace obrazu

Základním krokem pro zpracování a rozpoznávání obrazu je samotné získání obrazu reálného světa, jeho převod do digitální formy vhodné pro další zpracování počítačem či jiným systémem [3].

2.1.1. Snímání

Snímání obrazu je převod optické veličiny na elektrický signál. Tento signál je spojitý v čase i úrovni. Proces snímání lze také chápat jako radiometrické měření. Na kvalitu výsledného obrazu má vliv mnoho různých faktorů, jako například vlastnosti snímaného objektu a jeho ozáření [3].

2.1.2. Digitalizace

Digitalizace je převod analogového (spojitého) signálu do nespojité posloupnosti digitálních údajů, obvykle v binární formě. Digitalizace obrazu probíhá ve dvou krocích: kvantování a vzorkování. Podrobněji se digitalizací obrazu zabývá [5].

2.2. Předzpracování

Po úspěšném snímání a digitalizaci máme k dispozici obraz pozorované scény. Ten však může být zkreslen zejména díky nevhodným podmínkám v průběhu snímání.

Metody předzpracování obrazu slouží ke zlepšení obrazu z hlediska dalšího zpracování. Cílem předzpracování je potlačit šum vzniklý při digitalizaci a přenosu obrazu, odstranit zkreslení dané vlastnostmi snímacího zařízení, případně potlačit nebo zvýraznit jiné rysy důležité z hlediska dalšího zpracování [16].

Mezi základní operace patří převedení na stupně šedi, úprava jasu a kontrastu, ostření obrazu, ekvalizace histogramu a filtrace. Tyto operace mohou být aplikovány na celý obraz nebo jen na jeho část.

- Převedení na stupně šedi - operace při které dochází k transformaci hodnoty jasu pixelu na jinou bez ohledu na jeho pozici v obraze. [16]. Pixely jsou převedeny do rozmezí bílá - stupně šedi - černá.
- Úprava jasu a kontrastu - transformace sloužící pro korekci jasové funkce, která může obsahovat zkreslení způsobené např. nerovnoměrným osvětlením scény [16].
- Ostření obrazu - cílem ostření obrazu je upravit obraz tak, aby v něm byly strmější hrany. Pro ostření obrazu se obvykle používají gradientní operátory [16].
- Ekvalizace histogramu - metoda patřící mezi nejpoužívanější pro vylepšení kontrastu obrazu. Jejím hlavním cílem je získání jednotného histogramu. Po ekvalizaci je obraz kontrastnější s vystouplými detaily.
- Filtrace - převádí hodnoty jasu vstupního obrazu na jiné jasové hodnoty s cílem potlačení některých charakteristik obrazu. Jednou z nejčastěji prováděných operací je vyhlazování šumu obrazu. Nejčastěji používané typy filtrů pro danou aplikační oblast jsou [15]:
  - filtrování průměrováním
  - medián filtry
  - filtrování posuvným průměrováním
  - filtrování Gaussovým filtrem
  - filtrování pomocí Fourierovy transformace

Předzpracování obrazu obecně není nutné a má negativní vliv na rychlost zpracování. Může však značně vylepšit výsledky následujících operací.

2.3. Segmentace

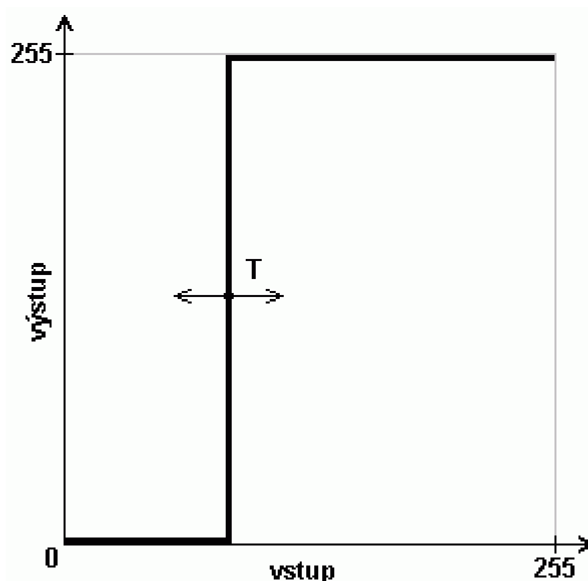
Jedním z nejdůležitějších kroků při zpracování obrazu je segmentace. Jedná se o operaci, při které se provádí analýza vedoucí k nalezení objektů v obraze. Za objekty se považují ty části program, které jsou bodem zájmu v dalším průběhu zpracování [3]. Cílem segmentace je nalezení částí obrazu, které odpovídají položkám reálného světa. V optimálním případě by po segmentaci měly být nalezeny takové prvky, které by z obrazu vyčlenil lidský mozek. Tedy takové, které přesně odpovídají objektům obsaženým v obraze [1]. Pro různé oblasti zpracování obrazu jsou vhodné různé přístupy segmentace. Rozdílné metody dosahují pro vstupní obraz odlišných výsledků. Metody, které se hodí například pro zpracování obyčejných fotografií, nemusí být vhodné pro zpracování medicínských obrazů [1].

2.3.1. Prahování

Prahování je nejjednodušší segmentační metoda. Vychází ze skutečnosti, že mnoho objektů nebo oblastí obrazu má konstantní odrazivost nebo pohltivost povrchu. Díky tomu je možné využití určené jasové konstanty *(prahu)* k oddělení objektů od pozadí. Vzhledem k jednoduchosti výpočtu je prahování nejrychlejší segmentační metodou, kterou lze provádět v reálném čase a je tedy hojně využívána [16]. Prahování je transformace vstupního obrazu f na výstupní binární obraz g daná vztahem [15]:

$$\begin{aligned} g(i,j) &= 1 \text{ pro } f(i,j) > T \\ g(i,j) &= 0 \text{ pro } f(i,j) \leq T \end{aligned} \tag{2.1}$$

kde: T je předem určená konstanta *(práh)*, $g(i,j) = 1$ pro pixely náležející po segmentaci popředí a $g(i,j) = 0$ pro pixely pozadí. Vzorec 2.1 je znázorněn na obrázku 2.2.

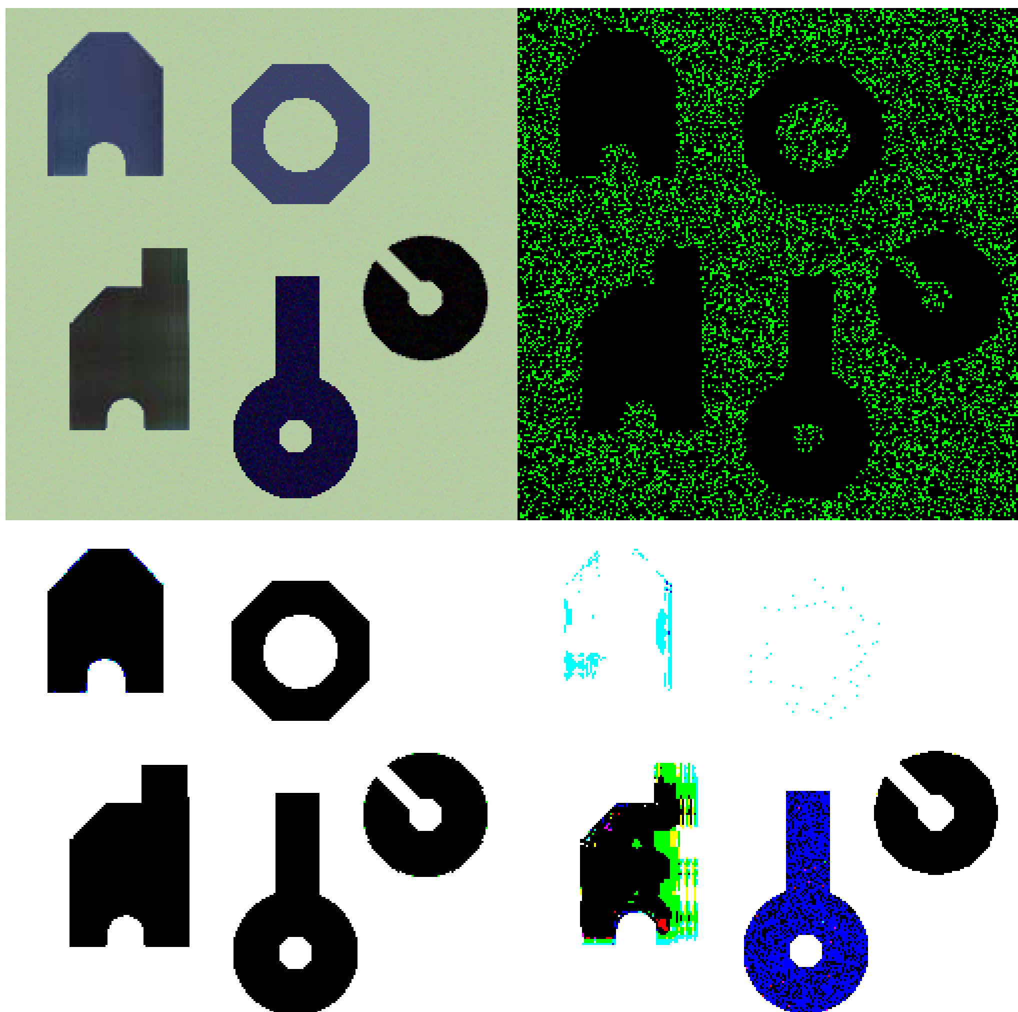


Obrázek 2.2: Funkce prahování

Správné určení hodnoty prahu je důležité pro výsledek operace. Na obrázku 2.3 je znázorněna volba správné hodnoty prahu.

Prahování celé scény jedním zvoleným prahem, může vykazovat nedostatečné segmentační výsledky. To může být způsobeno nerovnoměrným osvětlením scény a dalšími vlivy. V těchto případech je žádoucí použít prahování s proměnným prahem, kdy je hodnota prahu určována podle lokálních vlastností obrazu [15].

Mechanismus prahování lze postupně zdokonalovat zvyšováním počtu prahů (prahování do několika tříd) nebo/a dynamickým stanovováním hodnot prahů podle lokálních poměrů v obrazu. Takto stanovený práh se nazývá jasově adaptivní a může být vypočten např. jako průměr indexů dvou prostorově oddělených maxim histogramu, kdy se předpokládá, že jedno maximum odpovídá jasovým složkám pozadí a druhé jasovým složkám objektů. Někdy se také pro stanovení hodnoty prahu používají složitější výpočty vázané jen na určitou část obrazu (prostorově adaptivní práh) [7].



Obrázek 2.3: Výchozí scéna, prahování vysokým prahem, vhodným prahem a nízkým prahem

2.3.2. Detekce hran

Segmentace na základě detekce hran je postup, sloužící k nalezení oblastí pixelů, ve kterých se podstatně mění jas. Hrana v objektu se nemusí krýt s hranicí mezi objekty ve scéně a mohou vznikat a zanikat v závislosti na úhlu pohledu. Můžeme je najít na hranici objektů nebo rozhraní světla a stínu, nebo v místech trojrozměrných hran objektů. Reálná hrana na rozdíl od teoretické bývá zašuměná [18].

Hrana v obraze je dána vlastnostmi obrazového elementu a jeho okolí. Je určena tím, jak náhle se mění hodnota obrazové funkce $f(x, y)$. Matematickým nástrojem pro studium změn funkce dvou proměnných jsou parciální derivace. Změnu funkce udává její *gradient*. Gradient je směr největšího růstu obrazové funkce (od černé po bílou). Hrany jsou kolmice na směr gradientu. Gradientní operátory udávající strmost obrazové funkce lze rozdělit do čtyř kategorií [9][16]:

- Operátory aproximující derivace pomocí diferencí. Některé operátory jsou invariantní vůči rotaci (např. Laplacián) a mohou být počítány konvoluce s jedinou maskou.

- Operátory aproximující první derivaci. Využívají několik masek odpovídajících příslušné orientaci. Z nich se vybere ta, která nejlépe lokálně aproximuje obrazovou funkci. Výběrem jedné z masek je určen i směr gradientu (orientace).

- Operátory založené na hledání hrany v místech, kde druhá derivace obrazové funkce prochází nulou (angl. zero-crossing). Příkladem je Marrův-Hildrethové operátor a Cannyho hranový detektor.

- Operátory snažící se lokálně aproximovat obrazovou funkci poměrně jednouchým parametrickým modelem. Příkladem může být polynom dvou proměnných.

Je důležité zvolit metodu, která pro zvolený problém vykazuje dostatečné výsledky v co nejkratší možné době. Výsledky hranové detekce je možné vylepšit použitím vhodných metod předzpracování zejména filtrací.

2.3.3. Cannyho hranový detektor

V jistém smyslu završení období hledání „nejlepšího“ hranového detektoru. Je používán pro většinu aplikací. Pro tento detektor existuje velké množství dostupných implementací [9].

Základní myšlenka vychází z představy, že skokovou hranu lze hledat filtrem. Návrh tohoto filtru je formulován jako úloha variačního počtu za podmínky, že budou splněny jisté požadavky na chování filtru. Detektor je optimální pro skokové hrany vzhledem ke třem kritériím [6]:

1. Detekční kritérium požaduje, aby významné hrany nebyly přehlédnuty a aby na jednu hranu nebyly vícenásobné odezvy.

2. Lokalizační kritérium požaduje, aby rozdíl mezi skutečnou a nalezenou polohou hrany byl minimální.

3. Požadavek jedné odezvy zajišťuje, aby detektor nereagoval na jednu hranu v obraze vícenásobně. Toto očekávání je již částečně zajištěno prvním kritériem. Tento požadavek je zaměřen zejména na zašuměné a nehladké hrany, což první požadavek nezajistí.

Algoritmus Cannyho hranového detektoru lze popsat následovně [6]:

1. Najdi přibližné směry gradientu.

2. Pro každý pixel najdi 1D derivaci ve směru gradientu pomocí „optimální“ masky spojující vyhlazení a derivaci.

3. Najdi lokální maxima těchto derivací.

4. Hranové body získej prahováním s hysterezí.

5. Proveď syntézu hran získaných pro různě velká vyhlazení (málokdy se používá).

2.4. Popis

Popis obrazu je nejdůležitější součástí rozpoznávání obrazu. Jedná se o popis nalezených objektů z předešlé segmentace. Ve většině případů je popis objektů vstupní informací pro kvalifikaci objektů. Nalezení popisu, který by zajistil invariance vůči posunu, rotaci a změně měřítka není jednoduché.

Při použití strukturálních metod pro rozpoznávání objektů, je objekt tvořen řetězcem primitiv. Primitiva, reprezentují elementární část hrany objektu, jako jsou úsečky a oblouky. Řetězec primitiv definuje tvar objektu. Za předpokladu, že každá primitiva je reprezentována určitým symbolem (terminálem), můžeme říci, že popis objektu lze popsat za pomoci formálních jazyků a gramatik.

2.4.1. Gramatiky

Gramatika, jako nejznámější prostředek pro reprezentaci jazyků [2]. Každá gramatika je definována konečnou množinou pravidel, pomocí nichž lze generovat daný jazyk. Jednotlivá slova tohoto jazyka jsou tvořena postupným přepisováním řetězců podle těchto pravidel [10]. Gramatika používá dvou konečných disjunktních abeced:

1. množiny N nonterminálních symbolů
2. množiny Σ terminálních symbolů

Nonterminální symboly, krátce *nonterminály*, mají roli pomocných proměnných označujících určité syntaktické celky – syntaktické kategorie.

Množina *terminálních symbolů*, krátce *terminálů*, je identická s abecedou, nad níž je definován jazyk. Sjednocení obou množin, tj. $N \cup \Sigma$, nazýváme slovníkem gramatiky [2].

Gramatika G je čtveřice $G = (N, \Sigma, P, S)$, kde

- N je konečná množina nonterminálních symbolů
- Σ je konečná množina terminálních symbolů, $N \cap \Sigma = \emptyset$
- P je konečná podmnožina kartézského součinu $(N \cup \Sigma)^* N (N \cup \Sigma)^* \times (N \cup \Sigma)^*$,
- $S \in N$ je výchozí (také počáteční) symbol gramatiky

Prvek (α, β) množiny P nazýváme přepisovacím pravidlem (krátce pravidlem) a budeme jej zapisovat ve tvaru $\alpha \rightarrow \beta$. Řetězec α resp. β nazýváme levou resp. pravou stranou přepisovacího pravidla [2].

2.4.2. Chomského klasifikace gramatik

Chomského klasifikace gramatik (a jazyků), známá také pod názvem Chomského hierarchie jazyků, vymezuje čtyři typy gramatik, podle tvaru přepisovacích pravidel, jež obsahuje množina přepisovacích pravidel P. Tyto typy se označují jako typ 0, typ 1, typ 2 a typ 3 [2].

Typ 0

Gramatika typu 0 obsahuje pravidla v nejobecnějším tvaru, shodným s pravidly definice gramatik.

$$\alpha \rightarrow \beta, \quad \alpha \in (N \cup \Sigma)^* N (N \cup \Sigma)^* \times (N \cup \Sigma)^*, \quad \beta \in (N \cup \Sigma)^* \tag{2.2}$$

Z tohoto důvodu se gramatiky typu 0 nazývají také gramatikami *neomezenými* [2].

Typ 1

Gramatika typu 1 obsahuje pravidla tvaru:

$$\alpha A \beta \rightarrow \alpha \gamma \beta, \quad A \in N, \quad \alpha, \beta \in (N \cup \Sigma)^*), \quad \gamma \in (N \cup \Sigma)^+ \, nebo \, S \rightarrow \epsilon \tag{2.3}$$

Gramatikám typu 1 se také nazývá *kontextové gramatiky*, jelikož pravidla této gramatiky říkají, že nonterminál A může být nahrazen řetězcem γ pouze tehdy, je-li jeho pravým kontextem řetězec β a levým kontextem řetězec α.

Kontextové gramatiky neobsahují pravidla ve tvaru $\alpha A \beta \rightarrow \alpha \beta$, tj. nepřipouštějí, aby nonterminál byl nahrazen prázdným řetězcem. Jedinou přípustnou výjimkou je pravidlo $S \rightarrow \epsilon$ (kde S je výchozí symbol), které dovoluje popsat příslušnost prázdného řetězce k jazyku, který je danou gramatikou generován [2].

Typ 2

Gramatika typu 2 obsahuje pravidla tvaru:

$$A \rightarrow \gamma, \quad A \in N, \quad \gamma \in (N \cup \Sigma)^* \tag{2.4}$$

Gramatiky typu 2 jsou také nazývány *bezkontextové gramatiky*, jelikož nahrazení pravé strany γ pravidla za nonterminál A lze vykonat bez ohledu na kontext, kterým je nonterminál A obklopen. Na rozdíl od gramatik kontextových, gramatiky bezkontextové mohou obsahovat pravidla pro generování prázdných symbolů $A \rightarrow \epsilon$. Každou bezkontextovou gramatiku lze transformovat takovým způsobem, že obsahuje pouze jedno pravidlo s prázdným řetězcem na pravé straně $S \rightarrow \epsilon$, aniž by se změnil jazyk generovaný touto gramatikou [2].

Typ 3

Gramatika typu 3 obsahuje pravidla tvaru:

$$A \rightarrow xB, \quad nebo \quad A \rightarrow x; \quad A, B \in N, \quad x \in \Sigma^* \tag{2.5}$$

Gramatika s tímto tvarem pravidel se nazývá *pravá lineární gramatika* (jediný možný nonterminál pravé strany pravidla stojí úplně napravo). K uvedené gramatice lze sestrojit ekvivalentní speciální pravou lineární gramatiku s pravidly tvaru

$$A \rightarrow aB \quad nebo \quad A \rightarrow x; \quad A, B \in N, \quad a \in \Sigma \quad nebo \quad S \rightarrow \epsilon \tag{2.6}$$

Tato gramatika je nazývána *regulární* gramatikou, přesněji *pravou regulární gramatikou.* Gramatiky typu 3 se proto také nazývají *regulárními gramatikami* [2].

Pomocí regulárních gramatik je možné řešit veškerou problematiku popisu objektů z dané aplikační oblasti.

2.4.3. Návrh primitiv

Primitiva jsou základním stavebním prvkem obrazu u strukturálních metod rozpoznávání. Jejich návrh je přímo závislí na řešené aplikaci, kde snadné rozpoznání primitiv by měl být primární cíl. Pro scény, na kterých jsou objekty charakterizovány svou hranicí, jsou vhodnými primitivy části čar. Např. úsečka může být charakterizována svým začátkem a koncem [16]. Obecně platí:

- Primitiva musí být snadno rozpoznatelná.
- Primitiva musí poskytovat kompaktní a dostatečný popis obrazů pomocí specifikovaných vztahů.

Pokud jsou zvoleny jednodušší primitivy pro popis obrazu, výsledkem je složitý strukturální popis objektu. Ten má za důsledek nutnost použití složitější gramatiky pro popis objektů, avšak tyto primitivy se v obraze snadno hledají. Na druhou stranu složitější primitiva vede k jednodušší gramatice, ale její identifikace v obraze je podstatně náročnější.

Pro danou aplikaci byly zvoleny primitivy úseček a oblouku. Úsečky se dělí do osmi okolí, kterým lze dostatečně popsat objekty na scéně. Kombinací těchto primitiv je získán popis tvaru objektu (*řetězcový kód*) vhodný pro strukturální metody rozpoznávání. Primitivy s řetězcovým kódem jsou zobrazeny na obrázku 2.4. Popis objektu pomocí daných primitiv je znázorněn na obrázku 2.5.

2.5. Kvalifikace

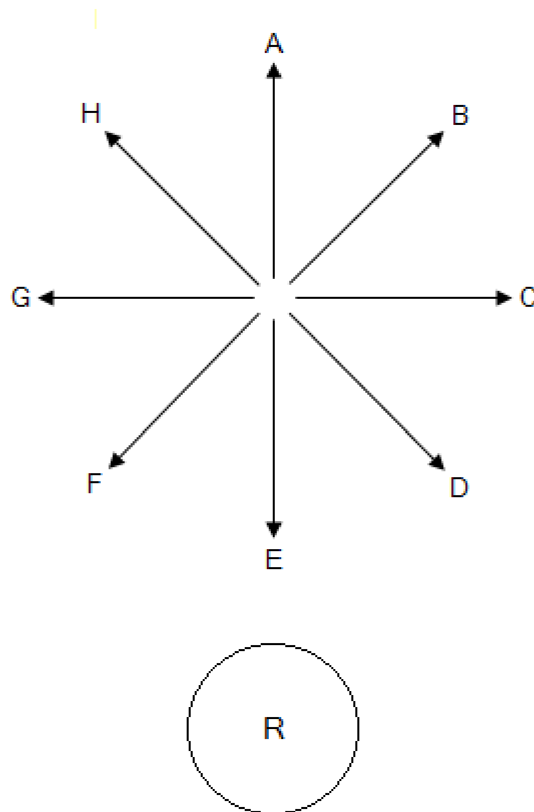
Finálním krokem při zpracování obrazu je klasifikace (rozpoznání obrazu). Objekt v počítačovém vidění nejčastěji představuje části segmentovaného obrazu. Objekty nalezené ve scéně se ve většině případů zařazují do skupin předem známých tříd. Metody klasifikace objektů se dělí do dvou základních skupin, které jsou úzce spjaty se způsobem popisu objektů. Jedná se o příznakové rozpoznávání a strukturální rozpoznávání. Příznakové metody jsou založeny na principu využití příznaků, což je skupina číselných charakteristik objektu. Učení vlastního klasifikátoru zde může být s příkladovou množinou i bez ní, na principu shlukové analýzy. Strukturální rozpoznávání využívá jako vstupu kvalitativní popis objektů. Objekty jsou zde popsány primitivy. Dále je definována abeceda, jazyk popisu a gramatiky jednotlivých tříd. Vlastní rozpoznávání je pak založeno na principu rozboru slova a kontroly správnosti syntaxe pro všechny třídy [3].

Samotnou činnost klasifikace vykonává $klasifiktor$. Klasifikátor nerozhoduje o třídě objektu podle objektu skutečného, nýbrž podle jeho obrazu [16].

V případě, že je pro rozpoznání objektu potřebný úplný popis obrazu, je nutná syntaktická analýza.

2.5.1. Syntaktická analýza

Úkolem syntaktického rozpoznávání obrazu je určit, zda analyzovaný obraz reprezentovaný slovem odpovídá slovu dané gramatiky, tedy zda gramatika může tento popis obrazu generovat. Obraz je reprezentován řetězcem jazyka, který je generován danou gramatikou [15].



Obrázek 2.4: Primitivy s řetězcovým kódem

Syntaktická analýza je proces sestrojení derivačního stromu pro danou větu. Program, který provádí rozklad vět určitého jazyka, se nazývá *syntaktický analyzátor* [2].

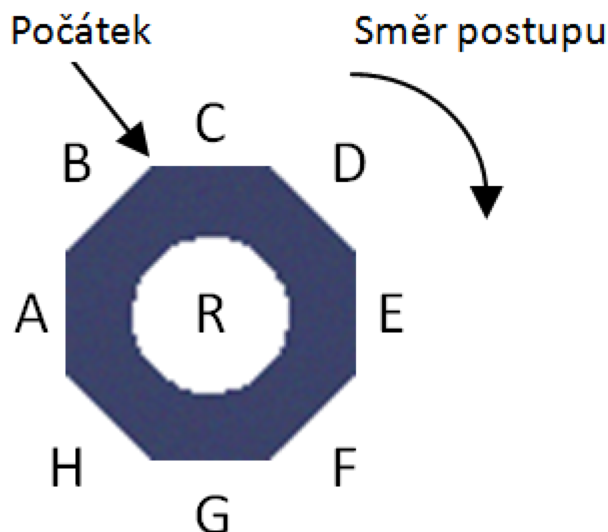
Algoritmy syntaktické analýzy je možné rozdělit dle způsobu konstrukce derivace věty (vytvoření derivačního stromu). Dělí se do dvou základních skupin:

- syntaktická analýza shora dolů
- syntaktická analýza zdola nahoru

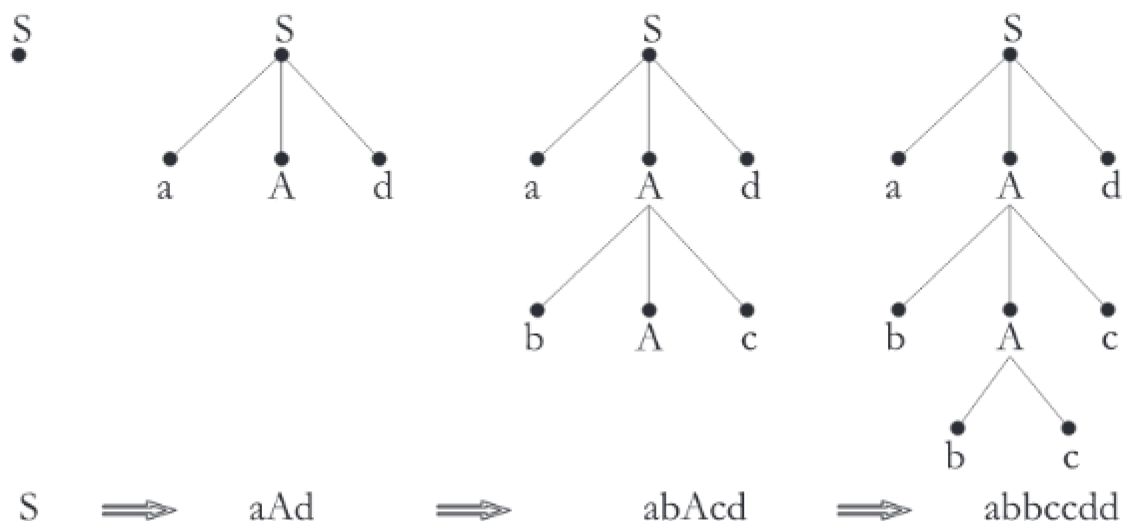
Syntaktická analýza shora dolů

Při syntaktické analýze shora dolů je derivační strom stavěn od výchozího symbolu a postupnými přímými derivacemi dojde k terminálním symbolům, které tvoří analyzovanou větu. Problém této metody je ve správnosti volby přímých derivací (pořadí použití přepisovacích pravidel) [2].

Na obrázku 2.6 je zobrazena konstrukce derivačního stromu řetězec *abbccdd* metodou shora dolů (převzato z [2]).



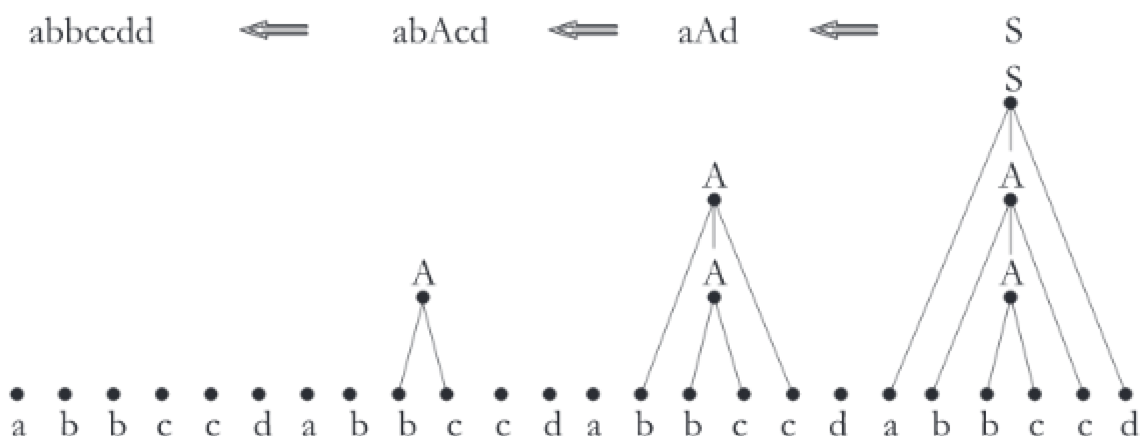
Obrázek 2.5: Popis objektu pomocí řetězcového kódu



Obrázek 2.6: Syntaktická analýza shora dolů

Syntaktická analýza zdola nahoru

Při syntaktické analýze zdola nahoru je derivační strom stavěn od koncových uzlů a postupnými přímými redukcemi dojde ke kořenu stromu (výchozí symbol gramatiky). Problémem této metody je nalezení prvního podřetězce věty, který může být redukován ke kořenu podstromu derivačního stromu [2]. Na obrázku 2.7 je zobrazena konstrukce derivačního stromu řetězec *abbccdd* metodou zdola nahoru (převzato z [2]).



Obrázek 2.7: Syntaktická analýza zdola nahoru

2.5.2. LL(k) syntaktický analyzátor

LL syntaktický analyzátor (neboli parser nebo překladový automat) využívá metody zdola nahoru pro bezkontextové jazyky. Analyzuje vstup zleva doprava, a konstruuje nejlevější derivaci věty. Gramatiky, které mohou být takto zpracovány, se nazývají LL gramatiky.

$LL(k)$ gramatika generuje jazyk typu $LL(k)$. LL parser se nazývá $LL(k)$, jestliže pro deterministickou analýzu věty je potřeba znát maximálně k následujících symbolů a není nutné použít zpětného vyhledávání. Nejpoužívanější gramatikou je $LL(1)$ gramatika, protože i přes jistá omezení této gramatiky stačí k deterministické analýze znát maximálně jeden následující symbol, což významně zjednodušuje konstrukci parseru [17].

Parser pro rozhodování používá LL tabulku. Tabulka se skládá ze tří částí.

- Množina $Empty$
- Množina $Follow$
- Množina $Predict$

Více o výpočtu jednotlivých množin k nalezení v [10][11].

Samotný algoritmus pro syntaktickou analýzu používající LL-tabulky probíhá takovýmto způsobem:

- Na zásobník vlož symboly $\$S$, kde S je počáteční nonterminál.
- Hlavní cyklus:
  - Nechť a je aktuální vstupní symbol, X symbol na vrcholu zásobníku
    - $X = \$$
      - Pokud $a = \$$, syntaktická analýza byla úspěšná.
      - Jinak nastala chyba.
    - $X \in T$
      - Pokud $X = a$, načti symbol a ze vstupu a odstraň tento symbol ze zásobníku.

- · Jinak nastala chyba.
- * $X \in N$
  - · Pokud $LL-tabulka$ na souřadnicích $[X, a]$ obsahuje pravidlo $r.X \rightarrow x$, odstraň symbol x ze zásobníku a vlož za něj převrácený řetězec x.
  - · Jinak nastala chyba.

- Proveď další smyčku cyklu

3. Deformační gramatiky

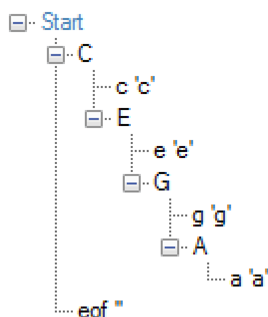
Jak již bylo zmíněno, gramatika je prostředek pro reprezentaci jazyka. Pokud jako jazyk bereme řetězcové reprezentace objektů, pak nad tímto jazykem je možné postavit gramatiku. Jako příklad uveďme obdélník znázorněný na obrázku 3.1. Řetězcový kód

Obrázek 3.1: Obdélník s řetězcovým kódem CEGA

tohoto objektu je $CEGA$. Gramatika popisující tento objekt pak vypadá následovně:

$$\begin{gathered} G = (\{C, E, G, A\}, \{c, e, g, a\}, P, C), \\ kde\, P\, obsahuje\, pravidla \\ C \rightarrow cE \\ E \rightarrow eG \\ G \rightarrow gA \\ A \rightarrow a \end{gathered}$$

Pokud na vstup syntaktického analyzátoru (blíže v kapitole 3.4) přijímajícího tuto gramatiky přivedeme řetězec $CEGA$, bude vytvořen derivační strom zobrazen na obrázku 3.2.

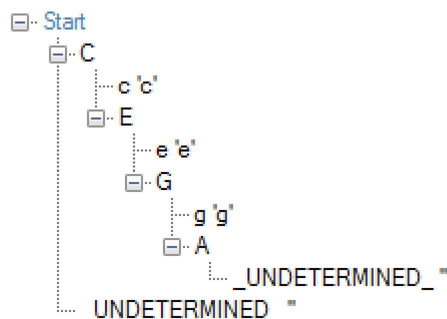


Obrázek 3.2: Derivační strom komponenty CEGA

Používat syntaktickou analýzu pro porovnávání řetězců by ovšem bylo zcela zbytečné a existuje velká řada jiných způsobů jak tuto operaci uskutečnit. Hlavní výhoda této metody vychází z předpokladu, že objekty na testovací scéně mohou být náhodně deformovány. Příklad náhodně deformovaného obdélníku je na obrázku 3.3, jehož řetězcový kód je $CEGHA$.

Rozklad řetězcového kódu deformovaného obdélníku, vypadá obdobným způsobem jak je vidět z obrázku 3.4, s tím rozdílem, že v pravidle A je očekáván jiný znak, než je na vstupu. Dalším problémem je jiná délka řetězcového kódu.

Obrázek 3.3: Deformovaný obdélník s řetězcovým kódem CEGHA



Obrázek 3.4: Derivační strom deformované komponenty CEGHA, bez použití deformační gramatiky

Pokud je cílem rozpoznávání obrazu, zařazení i deformovaných objektů do příslušných tříd, je nutné upravit gramatiku tak, aby přijímala i slova popisující deformovaný objekt. Neboli vytvořit *deformační gramatiku*.

3.1. Konstrukce deformační gramatiky

Základními pilíři při konstrukci deformační gramatiky jsou:

- Volba výchozí množiny komponent, takzvaných *etalonů*. Při konstrukci deformační gramatiky, je vždy nutné brát v potaz množinu prvků, nad kterými bude syntaktický analyzátor operovat. V případě množiny velmi podobných prvků, je nutné vytvářet jednotlivá přepisovací pravidla s velkou obezřetností. V opačném případě by mohla klasifikace dvou odlišných prvků skončit stejným výsledkem. V případě rozdílných komponent, je možnost volby deformačních pravidel širší, jelikož pravděpodobnost chybné klasifikace se značně snižuje.

- Určení co je možno považovat za akceptovatelnou deformaci komponenty. Pokud by deformační gramatika byla zkonstruována příliš obecně, s největší pravděpodobností by docházelo k situacím, kdy gramatika příjme i řetězec, který nespadá do požadované třídy.

3.1.1. Výchozí komponenty

Výchozí komponenty (*etalony*) určují jednotlivé třídy pro klasifikaci objektů ve scéně. Etalony zvoleny pro tuto práci byly převzaty nebo inspirovány z [16]. Vybraná skupina komponent vykazuje dostatečnou rozmanitost pro zjištění kvality zvoleného řešení. Byly požadovány následující vlastnosti testovací množiny:

- Množina obsahuje alespoň dva navzájem podobné prvky.
- Vyskytují se prvky, které mají otvory. Pro možnost klasifikace podřízených komponent.
- Existují nesymetrické prvky pro zajištění invariance vůči rotaci.

Vybraná skupina etalonů je znázorněna na obrázku 3.5.

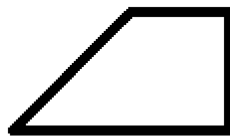
Obrázek 3.5: Množina zvolených etalonů

3.1.2. Deformace komponenty

Deformace komponenty je definována odlišností komponenty od etalonu. Pro zvolenou aplikační oblast byly uvažovány tyto kategorie chyb řetězcového kódu:

1. Před požadovaným znakem se nachází znak chybný (obrázek 3.3).
2. Namísto požadovaného znaku se nachází znak jiný (obrázek 3.6).
3. Požadovaný znak chybí (obrázek 3.7).

Pokud se některá z chyb vyskytne vícekrát za sebou, řetězec nebude gramatikou akceptován. Takovéto řetězce by reprezentovaly příliš deformovaný objekty.



Obrázek 3.6: Deformovaný obdélník s řetězcovým kódem CEGB

3.1.3. Pravidla deformačních gramatik

Po stanovení množiny etalonů a přípustných deformací je možné přistoupit ke konstrukci deformační gramatiky. Deformační gramatika vychází z původní gramatiky. Její počáteční symbol a množina terminálů je identická. Avšak množina nonterminálů a přechodových pravidel je značně odlišná. Tyto množiny musí být rozšířeny o stavy pro detekci chyb.

Obrázek 3.7: Deformovaný obdélník s řetězcovým kódem CEH

Obecný tvar přechodového pravidla

Přechodové pravidlo deformační gramatiky musí pokrývat výše zmíněné případy. Jeho obecný tvar pak vypadá:

$$N \rightarrow \textit{korektní znak}$$
$$N \rightarrow \textit{znak chybí}$$
$$N \rightarrow \textit{neočekávaný znak}$$

Je nutné, aby pravidla byla v uvedeném pořadí, jinak by derivace probíhala nekorektním způsobem.
Jako příklad uveďme vytvoření pravidla pro nonterminál C pro komponentu z obrázku 3.1 s řetězcovým kódem $CEGA$.
Korektní znak je zpracován stejným způsobem jako v původní gramatice:

$$C \rightarrow c\ E$$

Detekce chybějícího znaku je uskutečněna přeskočením hledaného nonterminálu. Tudíž je pro znak přidáno pravidlo korektního znaku jeho následovníka.

$$C \rightarrow e\ G$$

Pravidlo pro případ neočekávaného znaku na vstupu přichází v úvahu pouze ve chvíli, kdy syntaktický analyzátor neuspěl s předchozími možnostmi. Pro jeho uskutečnění je nutné na zásobník načíst další znak. Z toho vyplívá, že zpracování probíhá ve dvou krocích a je tedy nutné vytvoření dvou přechodových pravidel.
Nejprve je na vstupu nalezen libovolný terminální symbol x, který je vyhodnocen jako nový nonterminální symbol X.

$$C \rightarrow x\ X$$

Jelikož navržená gramatika nemá akceptovat více jak jednu chybu v řadě, je nutné, aby za nalezenou chybou následoval korektní symbol. Ten se odvíjí od situace, zda se neočekávaný znak nachází před nebo namísto požadovaného symbolu.

$$X \rightarrow c\ E$$
$$X \rightarrow e\ G$$

Přechodová pravidla pro jeden znak ve zkrácené formě mají podobu

$$C \rightarrow c\ E \mid e\ G \mid x\ X$$
$$X \rightarrow e\ G \mid c\ E$$

Z důvodu nutnosti načtení dalšího prvku v chybovém stavu X, vyplívá, že pravidla pro poslední znak musí být odlišná.
Jelikož u posledního znaku může nastat několik odlišných situací, je nutné vytvořit speciální pravidla.

1. Poslední znak neexistuje
2. Místo posledního znaku se nachází jiný znak
3. Za posledním znakem je další znak

Tyto situace lze pokrýt následující sadou pravidel, kde eof je značkou pro načtení celého vstupního řetězce.

$$
\begin{gathered}
A \rightarrow a\ X1 \mid x\ X2 \\
X1 \rightarrow eof \mid a \\
X2 \rightarrow eof \mid x
\end{gathered}
$$

Také může nastat situace, kdy se poslední znak ve vstupu nenachází vůbec. Díky tomu je potřebné upravit pravidla pro předposlední znak.

$$
\begin{gathered}
G \rightarrow g\ A \mid a\ X2 \mid x\ X \\
X \rightarrow b \mid g\ A
\end{gathered}
$$

3.2. Generátor gramatik

Pravidla pro vytváření deformační gramatiky jsou jasně daná. Jejich vytvoření je poměrně náročný postup a při ručním zpracování náchylný k chybám. Z toho důvodu byl do testovací aplikace implementován nástroj pro automatizované generování gramatik. Pomocí tohoto nástroje byla rovněž vytvořena gramatika pro syntaktický analyzátor aplikace.

Generátor terminálních symbolů

Jak je napsáno v kapitole 3.4.1 jako terminální symboly jsou přijímány regulární výrazy. Pro každou primitivu všech vstupních komponent je vytvořen terminální symbol ve tvaru

$$tx_y \rightarrow @"z+";$$

kde x je pořadové číslo komponenty a y pořadové číslo primitivy. Na pravé straně je pak regulární výraz reprezentující 1 až n výskytů požadované primitivy.
Na závěr je na začátek tohoto výčtu přidán terminální symbol pro přečtení celého vstupu a na konec symbol pro chybný vstup.

Generátor přechodových pravidel

Implementovaný syntaktický analyzátor očekává, že počáteční pravidlo gramatiky bude pojmenováno $Start$. Pravidlo slouží jako zásadní rozhodovací most mezi jednotlivými etalony. Pro gramatiky vzniklou z množiny pěti objektů má počáteční pravidlo tvar:

$$Start \rightarrow (\ Get1 \mid Get2 \mid Get3 \mid Get4 \mid Get5\ )?\ eof;$$

Gramatika přijímá i prázdné řetězce.
Dále je pro každou primitivu vytvořena sada přechodových pravidel, dle zásad popsaných výše ve tvaru:

$$Nx_y \rightarrow (\ tx_y\ Nx_y+1\ )\ |\ (\ tx_y+1\ N1_y+2\ )\ |\ (\ xx\ Nx_yX\ );$$
$$Nx_yX \rightarrow (\ tx_y\ Nx_y+2\ )\ |\ (\ tx_y\ N1_y+1\ );$$

kde

- x je pořadové číslo komponenty
- y pořadové číslo primitivy
- symboly začínající N značí nonterminály
- symboly začínající t značí terminály
- X reprezentuje chybové přechodové pravidlo
- xx chybný vstup s pořadovým číslem komponenty

3.3. Deformační gramatika

V kapitole 3.1 jsou uvedeny jednotlivé postupy při tvorbě navržené deformační gramatiky. Obecně je možné říci, že pomocí generátoru gramatik lze sestavit gramatiku pro jazyk přijímající řetězce takové, které na sousedních dvou pozicích nemají více jak jednu odlišnost od vzoru.

Pokud budeme uvažovat jako vstupní komponentu objekt z obrázku 3.1, pak výsledná deformační gramatika $G = (N, T, P, Start)$ vytvořená generátorem bude obsahovat množinu nonterminálních symbolů N

$$N = \{Start, Get1, N1_1, N1_1X, N1_2, N1_2X, N1_3, N1_3X, N1_4, N1_4X1, N1_4X2\}$$

množinu terminálních symbolů T reprezentovaných pomocí regulárních výrazů zobrazenou v tabulce 3.1 a množina přechodových pravidel P zobrazených v tabulce 3.2

Tabulka 3.1: Terminální symboly gramatiky G

eof	→	@"$";
t1	→	@"1";
t1_1	→	@"C+";
t1_2	→	@"E+";
t1_3	→	@"G+";
t1_4	→	@"A+";
x1	→	@".?";

Ve výsledných přechodových pravidlech stojí za povšimnutí pravidlo $Get1$. Jedná se o takzvané identifikační pravidlo komponenty. Syntaktická analýza probíhá pro každý etalon zvlášť a toto pravidlo slouží k identifikaci, který etalon je právě zkoumán.

Aplikujeme-li tuto gramatiku na vstupní komponentu, derivace bude probíhat obdobnými kroky jako je tomu na obrázku 3.2. Avšak základním požadavkem bylo, aby

Tabulka 3.2: Přechodová pravidla gramatiky G

Start	→	(Get1)? eof;
Get1	→	(t1 N1_1);
N1_1	→	(t1_1 N1_2) | (t1_2 N1_3) | (x1 N1_1X);
N1_1X	→	(t1_2 N1_3) | (t1_1 N1_2);
N1_2	→	(t1_2 N1_3) | (t1_3 N1_4) | (x1 N1_2X);
N1_2X	→	(t1_3 N1_4) | (t1_2 N1_3);
N1_3	→	(t1_3 N1_4) | (t1_4 N1_4X2) | (x1 N1_3X);
N1_3X	→	t1_4 | (t1_3 N1_4);
N1_4	→	(t1_4 N1_4X2) | (x1 N1_4X1);
N1_4X1	→	eof | t1_4;
N1_4X2	→	eof | x1;

gramatika přijímala i deformované objekty, jako jsou komponenty zobrazené na obrazcích 3.3, 3.6 a 3.7.

Na obrázku 3.8 jsou zobrazeny čtyři derivační stromy popisující průběh derivace původní komponenty a jejích deformovaných variant pomocí vytvořené deformační gramatiky.

1. $CEGA$ derivace popisu etalonu. Průběh zpracování je stejný jako při použití původní gramatiky, s rozdílem v posledním kroku derivace. Při použití deformační gramatiky syntaktický analyzátor navíc zjišťuje možnou přítomnost dalšího prvku.

2. $CEGHA$ derivace popisu se znakem navíc. Do zpracování vstupního znaku G probíhá proces standardní cestou. Při načtení znaku H dojde k rozpoznání chyby, načtení dalšího vstupu, který je určeného jako znak požadovaný v předchozím kroku, díky čemuž je pokračováno standardní cestou.

3. $CEGB$ derivace popisu se zaměněným znakem. Při načtení posledního znaku je zjištěna chyba. Analyzátor přejde do větve, kde očekává korektní znak, nebo informaci o načtení celého vstupu. Na zásobníku již další znak nemáme, tudíž je řetězec rozpoznán.

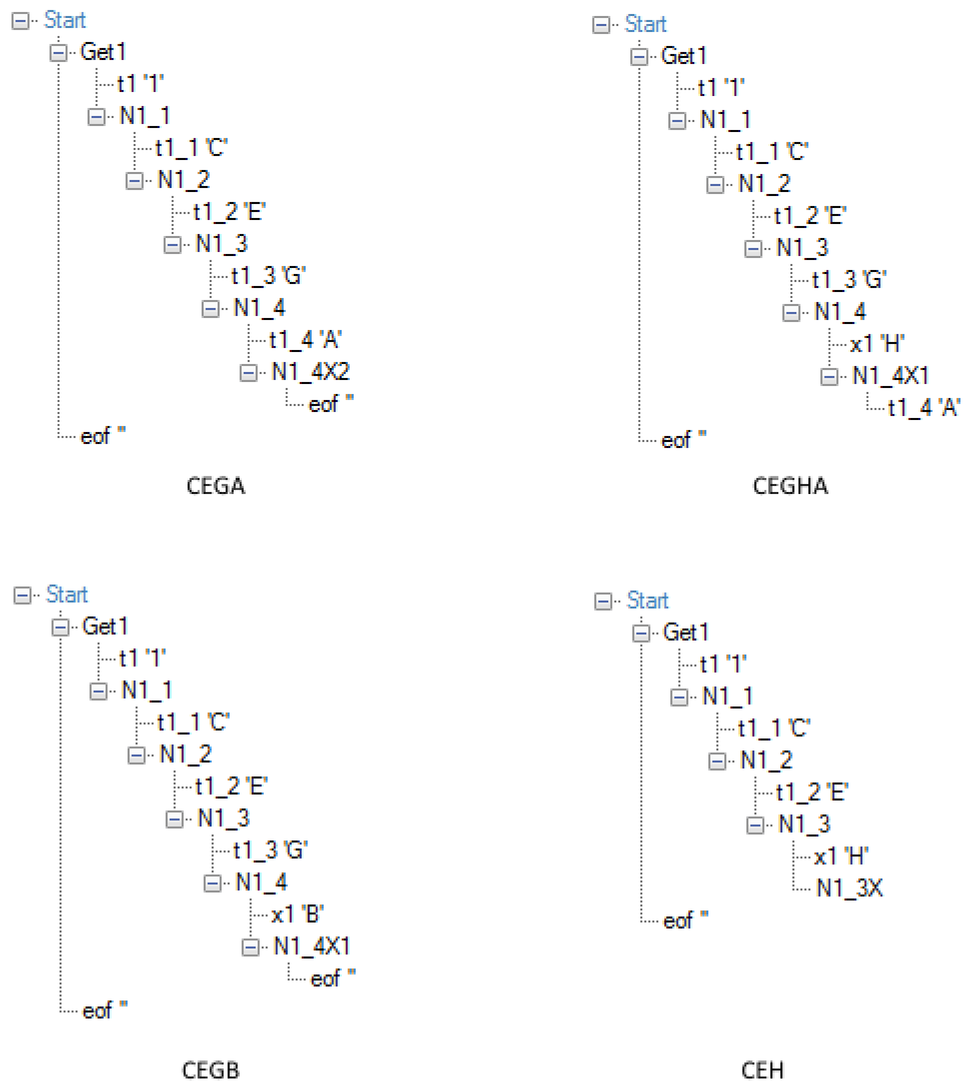
4. CEH derivace zkráceného popisu. Na posledním znaku je detekována chyba, ke které je přistupováno stejným způsobem jako v případě vstupu $CEGB$.

Pokud by v objektu byla složitější deformace, gramatika je navržena tak, aby tento objekt neklasifikovala. Kupříkladu, mějme objekt s řetězcovým kódem $CEGHFGA$, který se od původního vzorového objektu značně liší. Derivace takto deformovaného objektu probíhá do určité chvíle standardním způsobem, ale ve chvíli, kdy syntaktický analyzátor narazí na dvojitou chybu, prohlásí objekt za neznámý. Situace je znázorněna na obrázku 3.9, kde je vykreslen zkoumaný objekt společně s derivačním stromem.

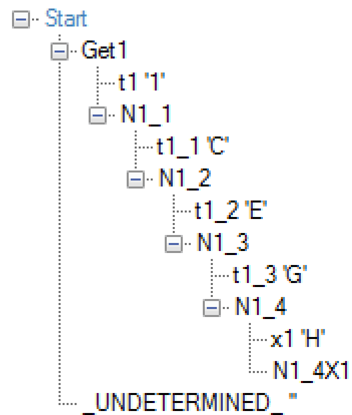
V tabulce 3.3 jsou vypsány příklady slov přijímaných tetovací gramatikou G.

3.4. Překladač

Strukturální metody rozpoznávání objektů jsou založeny na reprezentaci komponent pomocí formálního jazyku. Z toho důvody bylo nutné do testovací aplikace implementovat



Obrázek 3.8: Derivační stromy obdélníku a jeho deformovaných variant



Obrázek 3.9: Derivace značně deformovaného objektu

Tabulka 3.3: Přijímaná slova gramatikou G

CEGA
XCEGA
XEGA
XCEXGA
CEGAX

překladač, který by tento jazyk uměl interpretovat. Interpretace probíhá ve dvou hlavních fázích:

- Lexikální analýza
- Syntaktická analýza

3.4.1. Lexikální analýza

Lexikální analýza je zajištěna lexikálním analyzátorem. Jeho hlavní činností je číst vstupní řetězec a překládat ho na řetězec lexikálních symbolů (*lexémů*), které jsou dále předloženy syntaktickému analyzátoru.

Lexikální analyzátor je koncipován jako podprogram syntaktického analyzátoru, který ho volá v případě, že je nutný další vstupní znak. Pro potřeby rozpoznávání objektů v obraze byl zvolen lexikální analyzátor, který jako jednotlivé lexémy přijímá části vstupu popsané regulárním výrazem. Díky této vlastnosti jsme schopní za pomoci jednoduché reprezentace docílit zajištění invariance vůči změně měřítka objektu.

Zajištění invariance vůči změně měřítka

Mějme obdélník popsaný řetězcový kódem $CEGA$. Pokud bychom tento objekt třikrát zvětšili, dostaneme $CCCEEEGGGAAA$. Existují dvě možnosti jak zvětšený objekt klasifikovat do stejné třídy jako původní.

1. Do přepisovacích pravidel gramatiky přidat pravidlo pro každý terminální symbol zajišťující redukci řetězcového kódu. Pravidla by měla tvar $C- > CC$. Hlavní nevýhodou je zvýšení složitosti syntaktické analýzy a díky tomu horší porozumění jejímu průběhu.

2. Reprezentace terminálních symbolů pomocí regulárních výrazů. Tento přístup nám zajistí, že v případě vstupního řetězce $CCCEEEGGGAAA$ je syntaktickému analyzátoru jako vstup v jednotlivých krocích syntaktické analýzy předkládán redukovaný popis primitivy.

Regulární výrazy

Regulární výrazy jsou nástroj umožňující efektivně hledat v textu pomocí vzoru. Ke konstrukci regulárních výrazů se používají tzv. *meta znaky.* Při hledání v textu pomocí regulárních výrazů se testuje shoda meta znaků s prohledávaným textem [19]. V tabulce 3.4 je ukázka použitých regulárních výrazů v deformační gramatice programu. Více o regulárních výrazech lze nalézt v [12].

Tabulka 3.4: Regulární výrazy

Výraz	**Význam**	**Příklady**
@"$"	Dosáhnutí konce vstupu	
@"1"	Znak 1 pouze jednou	1
@"$C+$"	Výskyt znaku C 1 až n-*krát*	C, CC
@".?"	Libovolný znak pouze jednou	X, D

3.4.2. Syntaktická analýza

Druhou fází překladu je syntaktická analýza. Úkolem syntaktické analýzy je určit syntaktickou strukturu vstupního řetězce. Jinak řečeno syntaktický analyzátor se snaží sestrojit pro vstupní slovo lexikálních symbolů derivační strom.

Pro potřeby vyhodnocování byl do aplikace implementován syntaktický analyzátor typu $LL(1)$. Jak je blíže popsáno v kapitole 2.5.1, pro LL syntaktickou analýzu jsou nutné množiny $Empty$, $Follow$ a $Predict$. Pro navrženou testovací gramatiku popsanou v kapitole 3.3, jsou výsledné množiny popsány tabulkou 3.6 pro množinu $Empty$, tabulkou 3.5 pro množinu $Follow$ a tabulkou 3.7 pro množinu $Predict$.

Tabulka 3.5: Množina $Follow$ gramatiky G

Nonterminál	**Follow**
eof	$
t1	@"C+";, @".?";, @"E+";
t1_1	@"E+";, @".?";, @"G+";
t1_2	@"G+";, @".?";, @"A+";
t1_3	@"A+";, @".?";
t1_4	@$";
x1	@"$";, @".?";, @"A+";, @"G+";, @"E+";, @"C+";
Get1	∅
N1_1	∅
N1_1X	∅
N1_2	∅
N1_2X	∅
N1_3	∅
N1_3X	∅
N1_4	∅
N1_4X1	∅
N1_4X2	∅

Tabulka 3.6: Množina *Empty* gramatiky G

Nonterminál	**First**
@”$”;	@”$”;
@”1”;	@”1”;
@”C+”;	@”C+”;
@”E+”;	@”E+”;
@”G+”;	@”G+”;
@”A+”;	@”A+”;
@”.?”;	@”.?”;
eof	@”$”;
t1	@”1”;
t1_1	@”C+”;
t1_2	@”E+”;
t1_3	@”G+”;
t1_4	@”A+”;
x1	@”.?”;
Get1	@”1”;
N1_1	@”C+”;, @”.?”;, @”E+”;
N1_1X	@”E+”;, @”C+”;
N1_2	@”E+”;, @”.?”;, @”G+”;
N1_2X	@”G+”;, @”E+”;
N1_3	@”G+”;, @”.?”;, @”A+”;
N1_3X	@”A+”;, @”G+”;
N1_4	@”A+”;, @”.?”;
N1_4X1	@”$”;, @”.?”;, @”A+”;
N1_4X2	@”$”;

Tabulka 3.7: Množina *Predict* gramatiky *G*

#	**Pravidlo**	**Predict**
1	eof → @”$”;	@\$\;
2	t1 → @”1”;	@\1\;
3	t1_1 → @”C+”;	@\C+\;
4	t1_2 → @”E+”;	@\E+\;
5	t1_3 → @”G+”;	@\G+\;
6	t1_4 → @”A+”;	@\A+\;
7	x1 → @”.?”;	@\.?\;
8	Get1 → t1 N1_1	@\1\;
9	N1_1 → t1_1 N1_2	@\C+\;
10	N1_1 → t1_2 N1_3	@\E+\;
11	N1_1 → x1 N1_1X	@\.?\;
12	N1_1X → t1_2 N1_3	@\E+\;
13	N1_1X → t1_1 N1_2	@\C+\;
14	N1_2 → t1_2 N1_3	@\E+\;
15	N1_2 → t1_3 N1_4	@\G+\;
16	N1_2 → x1 N1_2X	@\.?\;
17	N1_2X → t1_3 N1_4	@\G+\;
18	N1_2X → t1_2 N1_3	@\E+\;
19	N1_3 → t1_3 N1_4	@\G+\;
20	N1_3 → t1_4 N1_4X2	@\A+\;
21	N1_3 → x1 N1_3X	@\.?\;
22	N1_3X → t1_4	@\A+\;
23	N1_3X → t1_3 N1_4	@\G+\;
24	N1_4 → t1_4 N1_4X2	@\A+\;
25	N1_4 → x1 N1_4X1	@\.?\;
26	N1_4X1 → eof	@\$\;
27	N1_4X1 → t1_4	@\A+\;
28	N1_4X1 → x1	@\.?\;
29	N1_4X2 → eof	@\$\;

3.4.3. Konstrukce syntaktického analyzátoru

Samotný syntaktický analyzátor interpretující jazyk vytvořené gramatiky, je poměrně složitý nástroj. Při jeho konstrukci se většinou využívá principu, kdy pro každý nonterminální symbol je vytvořena procedura, která zpracovává jednotlivá pravidla. Tato procedura v případě potřeby volá lexikální analyzátor pro načtení dalšího vstupního symbolu (*lexmu*, *tokenu*). Příklad této procedury, je zobrazen v následujícím bloku kódu převzatého z testovací aplikace.

```
private void ParseN5_2(ParseNode parent)
{
  Token tok;
  ParseNode n;
  ParseNode node = parent.CreateNode(
    scanner.GetToken(TokenType.N5_2), "N5_2");
  parent.Nodes.Add(node);

  tok = scanner.LookAhead(TokenType.t5_2,
    TokenType.t5_3, TokenType.x5);
  switch (tok.Type)
  {
    case TokenType.t5_2:
      tok = scanner.Scan(TokenType.t5_2);
      n = node.CreateNode(tok, tok.ToString() );
      node.Token.UpdateRange(tok);
      node.Nodes.Add(n);
      if (tok.Type != TokenType.t5_2) {
        return;
      }
      ParseN5_3(node);
      break;
    case TokenType.t5_3:
      tok = scanner.Scan(TokenType.t5_3);
      n = node.CreateNode(tok, tok.ToString() );
      node.Token.UpdateRange(tok);
      node.Nodes.Add(n);
      if (tok.Type != TokenType.t5_3) {
        return;
      }
      ParseN5_4(node);
      break;
    case TokenType.x5:
      tok = scanner.Scan(TokenType.x5);
      n = node.CreateNode(tok, tok.ToString() );
      node.Token.UpdateRange(tok);
      node.Nodes.Add(n);
      if (tok.Type != TokenType.x5) {
          return;
```

```
        }
        ParseN5_2X(node);
        break;
    default:
        break;
}
```

Podrobnější se problematice postupu vytvoření syntaktického analyzátoru věnuje například [10] nebo [11]. Syntaktický analyzátor testovací aplikace byl vytvořen pomocí nástroje *TinyPG*, který je volně dostupný na [8].

4. Návrh systému pro rozpoznávání objektů

Hlavním cílem této práce bylo navržení aplikace pro rozpoznávání objektů v obraze pomocí strukturálních metod s ohledem na rychlost a přesnost. V následujících kapitolách je popsán, způsob řešení a implementace jednotlivých kroků nutných pro rozpoznání objektů v obraze ve výsledné aplikaci.

4.1. Aplikace Recognizer

Aplikace Recognizer byla vyvinuta jako testovací prostředí pro zhodnocení navržených strukturálních metod rozpoznávání obrazu. Využívá syntaktický analyzátor typu $LL(1)$ pracující nad bezkontextovou gramatikou popisující vybrané testovací komponenty. Hlavním vstupem aplikace je obraz testovací scény ve formátu *bmp*.

Tento typ syntaktického analyzátoru byl zvolen s ohledem na jeho jednoduchost, která je výhodou pro použití ve vybrané aplikační oblasti, kde není nutné analyzovat složité řetězce. Díky nízkému stupni zanoření je vyhodnocení chybné větve analýzy dřívější. Další výhodou může být snadno pochopitelný průběh syntaktické analýzy.

Implementována byla uskutečněna za použití objektově orientovaného programování, psána v jazyce *Visual Basic .NET* (zkráceně $VB.NET$). Více o této problematice je k nalezení na [13][14].

Při rozpoznávání testovací scény je postupováno v několika krocích, kde u každého je sledována výpočetní náročnost. Jednotlivé kroky jsou:

- Předzpracování scény
- Nalezení komponent
- Syntaktická analýza komponent
- Rozpoznání komponent

Výsledek rozpoznávání je zanesen přímo do původní testovací scény.

4.2. Předzpracování scény

Předzpracování obrazu je důležitou součástí identifikace objektů ve scéně a usnadňuje další zpracování, jak bylo napsáno v 2.2. Časová náročnost této operace se pohybuje v řádu milisekund.

Pro zvolenou aplikační oblast byly uvažovány následující postupy předzpracování vstupní scény:

- Převod na stupně šedi
- Prahování

Z důvodu kladení velkého důrazu na rychlost výpočtu bylo převedení na stupně šedi vynecháno. Tato operace měla v důsledku pozitivní vliv na výsledek operace prahování. Avšak rozdíl výsledků při jejím nevyužití, byl zanedbatelný s porovnáním časové náročnosti výpočtu.

4.2.1. Prahování

Operace prahování je provedena nad celou scénou za pomoci konstantního prahu. Hodnota tohoto prahu byla určena experimentálně, aby výsledný obraz odpovídal oddělení komponent od podkladu.

Jazyk VB.NET obsahuje řadu nativních knihoven pro práci s obrázky, které jsou výpočetně velmi rychlé. Níže je popsán způsob, jakým je realizováno prahování vstupní scény za pomoci knihovny *ImageAttributes*.

```
Public Shared Sub ImageTreshold(
    oImg As Bitmap,
    Optional iTrashold As Single = 0.45F)
  Dim oImgattr As New Imaging.ImageAttributes()
  oImgattr.SetThreshold(iTrashold)
  ImageApplyAttribute(oImg, oImgattr)
End Sub
```

Původní pixely vstupní scény jsou překresleny novými, na nichž je provedena operace prahování.

```
Private Shared Sub ImageApplyAttribute(
    oImg As Bitmap,
    oImgattr As Imaging.ImageAttributes)
  Using g As Graphics = Graphics.FromImage(oImg)
    g.DrawImage(oImg,
                New Rectangle(0, 0, oImg.Width, oImg.Height),
                0, 0, oImg.Width, oImg.Height, _
                GraphicsUnit.Pixel, oImgattr)
  End Using
End Sub
```

4.3. Nalezení komponent

Vstupem pro nalezení komponent je testovací scéna, ve které by měly být jasně odděleny objekty od pozadí. Na tuto scénu je aplikována řada operací, jejichž výsledkem je seznam řetězcových kódů, reprezentujících jednotlivé komponenty ve scéně. Některé operace jsou pro celkový výsledek kritické, a proto je výpočetní náročnost tohoto kroku vyšší. Pohybuje se v řádu jednotek až desítek milisekund, podle složitosti vstupní scény. Jednotlivé prováděné operace jsou v tomto pořadí:

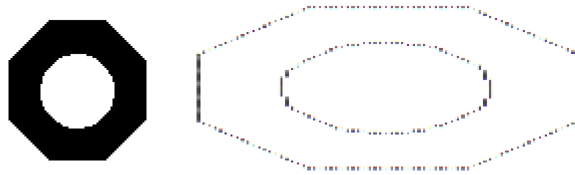
- Detekce hran
- Vyhledání komponent

– Vyhledání bodů objektu
– Vytvoření řetězcového kódu
– Zjištění souřadnic komponenty
– Určení hierarchie komponent

4.3.1. Detekce hran

Problém detekce hran je úzce spjatý s popisem objektů v obraze. Pro syntaktické metody rozpoznávání je vhodná reprezentace objektů scény a jejich primitiv pomocí řetězcových kódů, jak je znázorněno na obrázku 2.5. Primitivy vycházejí z hrany objektu. Díky této skutečnosti je možné aplikovat operaci detekce hran a rozpoznávání primitiv do jisté míry současně.

Pro potřeby vyvinuté aplikace byla zvolena velmi jednoduchá metoda pro zjišťování hran objektu. Výstupem této metody je bitové pole reprezentující hrany objektů vstupního obrazu. Ukázka převodu vstupní scény na bitové pole je znázorněna na obrázku 4.1, kde bitové pole je reprezentováno pomocí znaku X, tam kde se hrana nachází a *prázdného znaku*, tam kde se hrana nenachází. Vstupem této metody je scéna, u které se předpo-



Obrázek 4.1: Objekt a reprezentace jeho hran pomocí bitového pole

kládá, že na ni byla aplikována operace prahování. Kroky vedoucí k získání bitového pole jsou následující:

1. Vytvoř dvojrozměrné bitové pole stejné velikosti jako vstupní scéna.
2. Pro každý pixel vstupní scény urči hodnotu bitového pole na souřadnicích stejných jako zkoumaný pixel. Pokud má pixel černou barvu vlož hodnotu 1 jinak hodnotu 0.
3. Procházej bitové pole po řádcích od prvního (horního) řádku.
4. Procházej jednotlivé bity daného řádku.
5. Pokud bit obsahuje hodnotu 0 pokračuj na další.
6. Jinak urči hodnoty jeho čtyř okolí (horní, pravý, spodní, levý).
7. Pokud alespoň v jednom bodu okolí existuje jiná hodnota, než na současném bitu jedná se o hranu.

Tímto způsobem jsou projity všechny položky pole a jednotlivým bodům je určeno, zda jsou hrany či nikoliv.

4.3.2. Vyhledání komponent

Vyhledání komponent je finální fázi předzpracování vstupní scény. Po jejím dokončení máme vstupní scénu rozdělenou na jednotlivé komponenty a ty popsány pomocí řetězcového kódu. Komponenty jsou následně předány syntaktickému analyzátoru na klasifikaci.

Komponenty jsou v testovací aplikaci popsány pomocí třídy *Component*, která obsahuje následující pro klasifikaci podstatné položky:

- Řetězcový kód komponenty reprezentující její tvar.
- Kolekci podřazených komponent popsaných pomocí třídy *Component*.
- Souřadnice komponenty ve vstupní scéně.

Třída obsahuje i další položky, ale ty jsou používány pro reprezentaci na uživatelské úrovni.

Vyhledání komponent následně probíhá v bitovém poli, které je výstupem detekce hran tímto způsobem:

1. Procházej jednotlivé položky bitového pole a to po řádcích od levého bodu.
2. Pokud narazíš na hranu, pokus se vyhledat body objektu.
3. Pokud byly nalezeny nějaké body objektu, vytvoř z těchto bodů řetězcovou reprezentaci objektu (řetězcový kód). Jinak pokračuj bodem 1.
4. Zjisti souřadnice nalezeného objektu.
5. Vytvoř novou instanci třídy *Component* a ulož ji do kolekce.

Vyhledání bodů objektu

Pro vyhledání bodů program na vstupu dostane bod bitového pole reprezentujícího hrany objektů a pokusí se nalézt všechny body hran objektu. Jak je popsáno ve zdrojovém kódu metody, vyhledávání probíhá od počátečního bodu po směru hodinových ručiček. Metoda probíhá tak dlouho, dokud není opětovně naraženo na vstupní bod, nebo na chybu. Z toho je patrné, že jsou brány v potaz pouze spojité hrany. Nespojitá hrana může vzniknout například při špatně zvolené hodnotě prahu, jak je znázorněno na obrázku 2.3.

```
Private Function GetComponentPoints(
    oStart As Point) As List(Of Point)
  Dim oResult As New List(Of Point)
  Dim oPoint As Point = oStart
  Do
    If GetNewPoint(oPoint) Then Return Nothing
    oResult.Add(oPoint)
  Loop Until oStart = oPoint
  Return oResult
End Function

Private Function GetNewPoint(
```

```
    ByRef oPoint As Point) As Boolean
  With oPoint
    Dim x As Integer = 0, y As Integer = 0
    If mbMatrix(.X + 1, .Y) Then
      x = .X + 1 : y = .Y
    ElseIf mbMatrix(.X + 1, .Y + 1) Then
      x = .X + 1 : y = .Y + 1
    ElseIf mbMatrix(.X, .Y + 1) Then
      x = .X : y = .Y + 1
    ElseIf mbMatrix(.X - 1, .Y + 1) Then
      x = .X - 1 : y = .Y + 1
    ElseIf mbMatrix(.X - 1, .Y) Then
      x = .X - 1 : y = .Y
    ElseIf mbMatrix(.X - 1, .Y - 1) Then
      x = .X - 1 : y = .Y - 1
    ElseIf mbMatrix(.X, .Y - 1) Then
      x = .X : y = .Y - 1
    ElseIf mbMatrix(.X + 1, .Y - 1) Then
      x = .X + 1 : y = .Y - 1
    End If
    If x > 0 And y > 0 Then
      mbMatrix(x, y) = False
      oPoint = New Point(x, y)
      Return False
    End If
    Return True
  End With
End Function
```

Výstupem je kolekce všech bodů hran objektu, které jsou předány pro překódování na řetězcový kód.

Vytvoření řetězcového kódu

Vytvoření řetězcového kódu objektu neboli popis objektu v obraze je nejkritičtější část celého rozpoznávání.

Pro vybranou aplikační oblast byly zvoleny primitivy s řetězcovým popisem znázorněným na obrázku 2.4. Algoritmus rozpoznávání pracuje na principu postupného vyhledání jednotlivých úseček. Úsečka je definována pomocí sledu jednotlivých pixelů a odchylky od ideálního tvaru křivky. Následuje krok rozpoznání oblouků, kde se vychází z předpokladu, co není úsečka, je oblouk. Každá z těchto primitiv musí mít minimální předepsanou délku. Pokud v objektu existuje sled bodů, který neodpovídá žádné primitivě, aplikace nabízí možnost substituce těchto bodů za náhradní symbol. Tato možnost slouží k identifikaci drobných nepřesností. Avšak v praktickém využití byla shledána nadbytečnou.

Nejdůležitější součástí tohoto kroku je identifikace úseček objektu, od které se odvíjí oblouky a případné chyby. Identifikace je v programu implementována do následující funkce:

```
Private Sub CheckPrimitive(
```

```
    oPoints As List(Of Point), fCondition As Condition,
    cCode As Char, oPrimitives As List(Of Primitive))
  Dim iIndex As Integer
  'pocita se s minimalni velikosti primky
  For i = 0 To oPoints.Count - 1 - MinLength
    Dim l As Integer = 0
    'zacatek prohledavani
    Dim f = oPoints(i) 'next
    For j = i + 1 To oPoints.Count - 1
      iIndex = oPoints.Count - 1
      Dim c = oPoints(j) 'current
      'pokud bod neodpovida parametrum primitivy
      If Not fCondition(f, c, l) Then
        iIndex = j - 1
        Exit For
      End If
      l += 1
    Next
    'pocet souhlasnych bodu vetsi nez minimalni delka
    'jedna se o primitivu
    If iIndex - i + 1 >= MinLength Then
      oPrimitives.Add(New Primitive(cCode, i, iIndex))
      i = iIndex + 1
    End If
  Next
End Sub
```

kde na vstupu jsou seznam bodů objektu, delegát na funkci podmínky příslušnosti, znak reprezentující hledanou primitivu a kolekce již nalezených primitiv. Hlavním místem metody je rozhodování, zda následující bod odpovídá podmínce hledané primitivy. To je realizováno pomocí funkce předané delegátem $fCondition$. Každá z osmi úseček má vlastní rozhodovací funkci. V následujícím kódu je příklad rozhodovací funkce pro primitivu D z obrázku 2.4. Vstupem jsou výchozí bod přímky, zkoumaný bod a možná odchylka od ideálního tvaru přímky.

```
Private Function CheckD(
    f As Point, c As Point, l As Integer) As Boolean
  Return (c.X >= f.X + l - t And c.X <= f.X + l + t _
          And c.Y >= f.Y + l - t And c.Y <= f.Y + l + t)
End Function
```

Tímto způsobem jsou projity všechny body jednotlivých komponent. Komponenta může být tvořena *jednou* (kruh) až n primitivy.

Zjištění souřadnic komponenty

Souřadnice komponenty slouží k identifikaci podřízených prvků komponenty, jak je popsáno dále. Metoda zvolená pro definování umístění komponenty v obraze vychází z předpokladu, že vybraná aplikační oblast nepokrývá složité útvary. Za tohoto předpokladu

je možné říci, že pro dostatečnou identifikaci pozice objektu postačí čtyři hraniční body. Jedná se o maxima a minima v jednotlivých osách. Výběr hraničních bodů je popsán v následujícím kódu, kde funkce *Find* je nativní metoda vracející první prvek odpovídající vstupní podmínce.

```
Private Function GetBorders(
    oComponentPoints As List(Of Point)) As Point()
  'horni, pravy, dolni, levy
  Dim aoBorders(3) As Point
  With oComponentPoints
    aoBorders(0) = .Find(Function(f) f.Y = .Max(Function(m) m.Y))
    aoBorders(1) = .Find(Function(f) f.X = .Max(Function(m) m.X))
    aoBorders(2) = .Find(Function(f) f.Y = .Min(Function(m) m.Y))
    aoBorders(3) = .Find(Function(f) f.X = .Min(Function(m) m.X))
  End With
  Return aoBorders
End Function
```

Určení hierarchie komponent

Posledním krokem vyhledání komponent je určení jejich hierarchie. Jak je patrné z obrázku 2.5, některé komponenty mohou obsahovat otvory. Tyto otvory se pro algoritmus jeví jako běžné komponenty, díky čemuž se k nim přistupuje zcela běžným způsobem. Pro správnou klasifikaci objektu je nutné uchovat informaci o existenci těchto otvorů v komponentě. Pokud by tento krok nebyl proveden, mohlo by se stát, že komponenta s otvorem a stejná komponenta pouze bez otvoru by byly klasifikovány do stejné třídy.

Určení hierarchie je v aplikaci implementováno následujícím způsobem:

```
Private Sub ProcessChilds()
  'usporadani hierarchie
  For Each oComponent In maComponents
    oComponent.Childs = FindChilds(oComponent)
  Next
  'nastaveni main polozek
  SetMain()
End Sub

Private Function FindChilds(
    oComponent As Component) As List(Of Component)
  With oComponent
    Return maComponents.Where(
      Function(f) f.Borders(0).Y < .Borders(0).Y _
        And f.Borders(1).X < .Borders(1).X _
        And f.Borders(2).Y > .Borders(2).Y _
        And f.Borders(3).X > .Borders(3).X _
        ).ToList
  End With
End Function
```

Z kódu je patrné, že jsou projity všechny komponenty z kolekce (metoda *ProcessChilds*), kterým je přiřazena množina jejich potomků (metoda *FindChilds*). Jako potomek jsou určeny ty komponenty, které mají všechny hraniční body uvnitř hraničních bodů zkoumané komponenty. Dále je použita metoda *SetMain*, která pro mateřské položky nastaví příznak, který se používá při výpisu komponent uživateli.

4.3.3. Parsování komponent

Parsování komponent je krokem, ve kterém se rozhoduje, zda řetězcové kódy komponent přísluší do jazyka generovaného gramatikou syntaktického analyzátoru popsanou v kapitole 3. Jak již bylo napsáno, je použit typ syntaktického analyzátory $LL(1)$. V tomto kroku je také zajištěna invariance vůči natočení komponenty oproti výchozí komponentě (etalonu). Tyto akce jsou zajištěny poměrně jednoduchou metodou:

```
Public Function ParseWithRotation(
    sChainCode As String, ByRef sResult As String,
    ByRef oResTree() As TreeNode) As Boolean
  For i = 1 To 8
    'pro jednotlive zname kody
    For c = 1 To Config.ComponentCount
      If Parse(c & sChainCode, sResult, oResTree) Then
        Return True
      End If
    Next
    'natoceni
    sChainCode = ChainCodeFinder.RotateChainCode(sChainCode)
  Next
  Return False
End Function
```

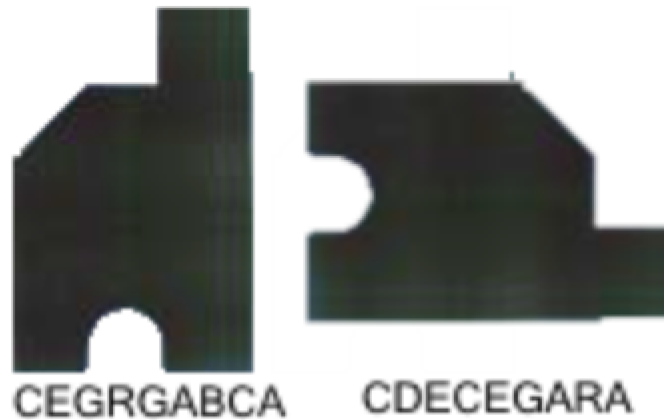
Na vstupu je zkoumaný řetězcový kód a proměnné pro zaznamenání výsledku syntaktické analýzy. Průběh samotné syntaktické analýzy je dobře popsán v kapitole 3.

4.3.4. Zajištění invariancí vůči rotaci

Jedná se o velmi důležitou část rozpoznávání komponent v obraze. Jak je vidět na obrázku 4.2, tak řetězcové kódu stejné komponenty pouze jinak natočené jsou diametrálně rozdílné. Tato odlišnost je způsobena tvorbou řetězcového kódu vždy od levého horního bodu komponenty, jak je popsáno v kapitole 4.3.2.

Zajištění invariance vůči natočení je možné zprostředkovat na základě celkem jednoduchého principu. Objekt rotujeme po směru hodinových ručiček kolem své osy tak dlouho, než nalezneme odpovídající pozici, nebo dosáhneme pozice původní (rotace o 360°). Jednotlivé rotace (v případě testovací aplikace o 45°) jsou prováděny překódováním všech primitiv na jejich ekvivalent překlopený o 45°.

```
Public Shared Function RotateChainCode(
    sChainCode As String) As String
  Dim sResult As String = ""
  If sChainCode.Length < 2 Then Return sChainCode
```



Obrázek 4.2: Řetězcový kód komponenty a jejího natočení

```
    For Each c In sChainCode.ToCharArray
      Dim iAsc As Integer = 0
      Select Case Asc(c)
        Case Codes.R, Codes.X : sResult &= c
        Case Codes.H : sResult &= Chr(Codes.A)
        Case Else : sResult &= Chr(Asc(c) + 1)
      End Select
    Next
    Return sResult
End Function
```

Je zřejmé, že oblouk a substituční symbol nemá význam rotovat, proto zůstávají nezměněny. V tabulce 4.1 je ukázka rotace komponenty z obrázku 4.2 o 360°.

Tabulka 4.1: Ukázka rotace komponenty

0°	CEGRGABCA
45°	DFHRHBCDB
90°	EGARACDEC
135°	FHBRBDEFD
180°	GACRCEFGE
225°	HBDRDFGHF
270°	ACEREGHAG
315°	BDFRFHABH
360°	CEGRGABCA

4.3.5. Rozpoznání komponent

Posledním krokem je samotné rozpoznání komponent neboli klasifikace. Aby byla komponenta rozpoznána, musí projít všechny řetězcové kódy komponenty (hlavní i podřízené) syntaktickou analýzou. To však není dostatečná podmínka prohlášení komponenty za známou. Jelikož se komponenta může skládat z více řetězcových kódů, které syntaktický

analyzátor vnímá jako samostatné oddíly, je nutné výslednou kolekci komponent porovnat se seznamem známých komponent (etalonů). Tato operace je uskutečněna metodou *FindComponent*. Metoda je volána pro každou položku z výsledné kolekce nalezených komponent, a pokud je výsledek kladný, přiřadí komponentě název třídy etalonu.

```
Public Shared Function FindComponent(
    oComponent As Component) As String
  Dim sNotFound As String = "Nenalezeno"
  Dim aoCandidates As List(Of Component) = _
    moData.Components.Where(Function(x)
      x.ChainCode = oComponent.Result And
      x.Childs.Count = oComponent.Childs.Count).ToList
  If aoCandidates.Count = 0 Then Return sNotFound

  For Each oCandidate In aoCandidates
    Dim bResult As Boolean = True
    For Each oChild In oComponent.Childs
      If oCandidate.Childs.Where(Function(x)
        x.ChainCode = oChild.ChainCode).Count = 0 Then
        bResult = False
        Exit For
      End If
    Next
    If bResult Then Return oCandidate.Name
  Next
  Return sNotFound
End Function
```

Základním předpokladem je, že každá komponenta může obsahovat pouze jednu úroveň podřízených komponent. To je dáno skutečností, že podřízené komponenty jsou otvory, které již nemohou obsahovat další otvory. Časová náročnost této operace je v řádu desetin milisekund.

5. Možnosti automatické tvorby algoritmů pro popis a rozpoznání objektů

Jedním z cílů této práce bylo analyzovat možnosti automatické tvorby algoritmů pro popis a rozpoznávání objektů. Tedy navrhnout algoritmický postup, který by bez vnějšího zásahu (lidského faktoru), byl schopný, přijmou množinu objektů a tu následně rozpoznávat.

Představme si scénář, kdy máme zvolenou množinu výchozích komponent (etalonů), například jako na obrázku 3.5. Výchozí komponenty vložíme do aplikace, která vytvoří řídící program pro rozpoznávání etalonů a jejich deformovaných variant. Z potřeby rozpoznávat deformované objekty vychází optimální využití strukturálních metod pro rozpoznávání objektů (deformačních gramatik). Aplikace pracující na tomto principu by musela obsahovat tyto součásti:

- Převodník vstupního obrazu.
- Generátor gramatik.
- Generátor řídícího kódu.

Převodník vstupního obrazu

Nástroj, který vytvoří vhodnou reprezentaci komponent vstupního obrazu pro zpracování pomocí strukturálních metod. Pro dosažení nejlepších výsledků by měl zajišťovat následující množinu operací:

- Převod hran objektů na řetězcové kódy.
- Zajištění jedinečnosti výskytu řetězcového kódu.
- Vytvoření reprezentace jednotlivých komponent, aby bylo možné určit hierarchickou příslušnost.
- Optimalizovat pořadí řetězcových kódů pro dosažení optimálních výsledků syntaktické analýzy.

Tato problematika je podrobně popsána v kapitole 4.

Generátor gramatik

Pro uvažovanou aplikaci je ideálním řešením generátor gramatik popsaný v kapitole 3.

5.1. Generátor řídícího kódu

Zatím neřešenou problematikou zůstává generování řídícího kódu. Jeho úkolem je přijmout zformovanou gramatiku a vytvořit syntaktický analyzátor, který by tuto gramatiku interpretoval. Problémem vytvoření syntaktického analyzátoru (parseru) z gramatiky jazyka se zabývají takzvané parser generátory (compiler-compiler).

Úkolem parser generátoru je načíst popis jazyka a z tohoto popisu vygenerovat překladač vloženého jazyka. Klasický parser generátor vytváří asociovaný spustitelný blok kódu pro každé z pravidel gramatiky. Těmto blokům se říká *akční rutiny*. V závislosti na typu syntaktického analyzátoru, který by měl být vygenerován, konstruují tyto rutiny derivační strom nebo abstraktní syntaktický strom. Více o problematice parser generátorů například v [4]. Mezi známé parser generátory patří:

- ANTLR
- Bison
- Yacc
- Java CC

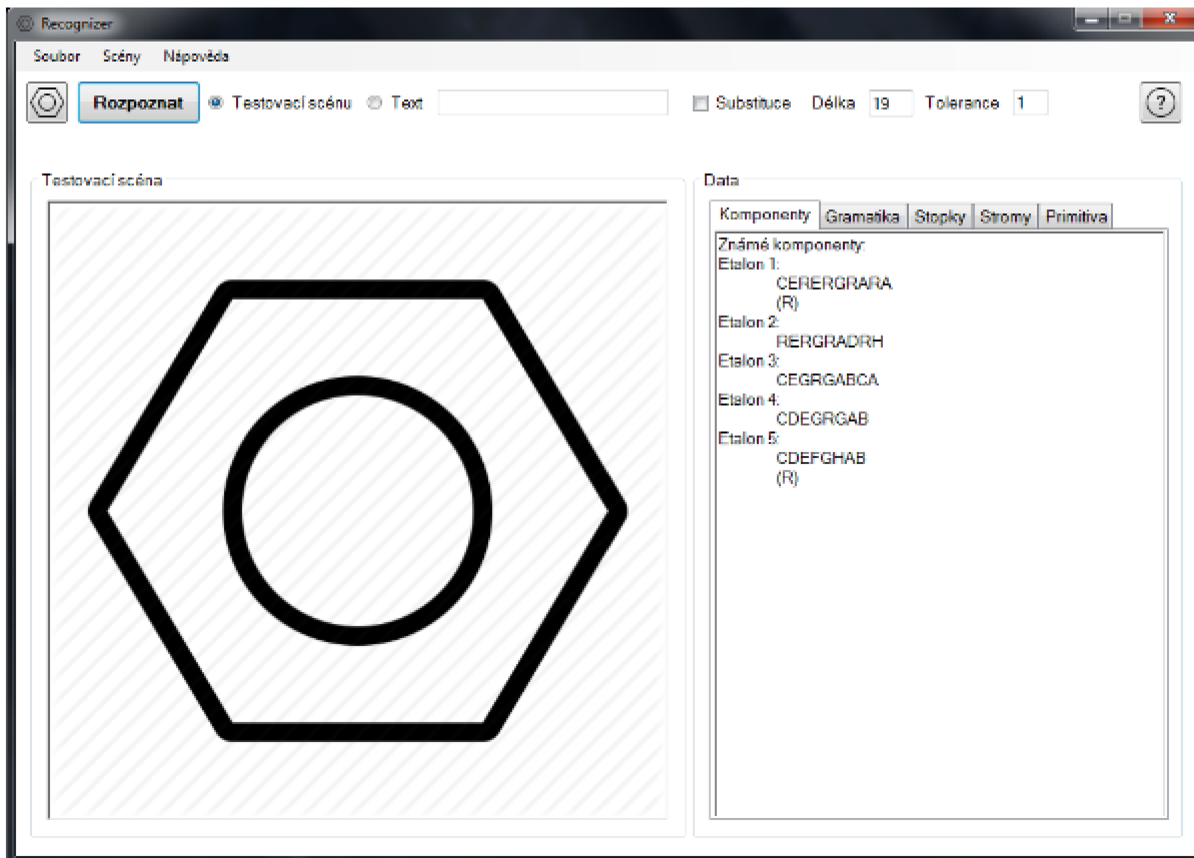
5.2. Teoretická aplikace

Výsledná aplikace by mohla mít využití například ve strojírenství pro kontrolu vyrobených součástek. Kde jako vstup aplikace by sloužil snímač (například CCD kamera), který by v daném časovém intervalu pořizoval snímky výrobního pásu. Pokud by se na pásu vyskytla součástka, která neodpovídá gramatice řídícího programu, došlo by k okamžitým krokům pro odstranění součástky.

V testovací aplikaci je syntaktický analyzátor pevně začleněn do aplikace. To přináší úskalí nemožnosti změny množiny etalonů. Implementací generátoru řídícího kódu do aplikace by bylo možné tento problém eliminovat. V takovém případě by aplikace vyhovovala nastíněnému scénáři.

6. Popis testovacího prostředí

Jak již bylo zmíněno, pro zhodnocení testovacích výsledků navržené metody, byla vyvinuta aplikace Recognizer. Tato kapitola zprostředkovává seznámení s aplikací Recognizer a možnostmi jejího použití. Aplikace je znázorněna na obrázku 6.1. Pro co nejsnadnější



Obrázek 6.1: Aplikace Recognizer

vyhodnocení testovacích scén byly do aplikace implementovány prvky, shromažďující veškeré relevantní informace o průběhu zpracování.

Vložení komponent pro rozpoznání

Aplikace umožňuje rozpoznávání z dvou různých vstupů.

1. Testovací scéna vložená pomocí tlačítka (*matice*), nebo zvolením předdefinované scény z menu *Scény*.

2. Řetězcový kód (pole *Text*) reprezentující komponentu. Slouží pro možnost zjištěná, zda vybrané slovo je generováno gramatikou aplikace.

Nastavení aplikace

Program umožňuje nastavit několik parametrů ovlivňujících výsledek rozpoznávání. Hodnoty, které lze měnit v uživatelském rozhraní mají vliv na výsledek kroku vyhledání komponent popsaném v kapitole 4.3.

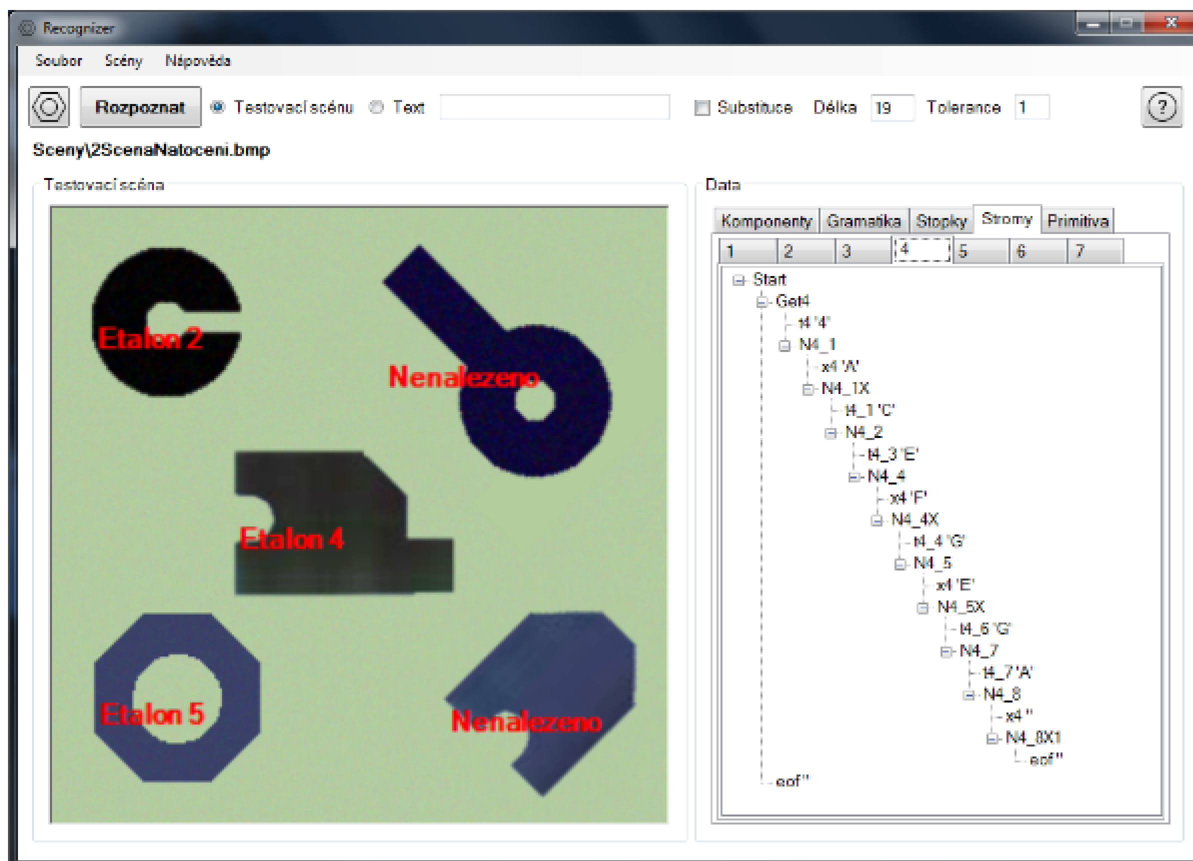
- *Substituce* - určuje zda nerozpoznané primitivy budou nahrazeny za substituční symbol.
- *Délka* - minimální délka primitivy v pixelech.
- *Tolerance* - odchylka od ideálního tvaru křivky primitivy.
- *Práh* - pro operaci prahování. Měnit tuto hodnotu není umožněno v uživatelském rozhraní, jelikož má kritický vliv na výsledek operace vyhledání komponent. Pokud by však bylo zjištěno, že nastavení je nevyhovující, lze práh změnit v konfiguračním souboru aplikace.

Výstup aplikace

Výstupem aplikace jsou:

- Zpracovaná testovací scéna.
- Údaje informující o průběhu zpracování.

Výsledek rozpoznávání je zanesen do původní vstupní scény a zobrazen v panelu scény. Ke každému objektu, který byl ve scéně nalezen, je vložen výsledek. Pokud objekt spadá do množiny etalonů, je zde zobrazen přímo jeho název. V opačném případě informace o nenalezení. Výstup aplikace je znázorněn na obrázku 6.2.



Obrázek 6.2: Aplikace Recognizer se zobrazenými výstupy

Záložky sekce *Data*:

- *Komponenty* - po spuštění programu zobrazení výchozích komponent (etalonů) a jejich řetězcových kódů. Pokud etalon obsahuje podřízený prvek, je tento prvek zobrazen v závorce. Po dokončení rozpoznávání jsou přidány komponenty nalezené v testovací scéně a výsledek jejich klasifikace.
- *Gramatika* - zobrazení deformační gramatiky přijímané syntaktickým analyzátorem aplikace.
- *Stopky* - informace o době trvání jednotlivých částí programu.
- *Stromy* - derivační stromy průběhu syntaktické analýzy nalezených komponent.
- *Primitiva* - znázornění základních stavebních bloků komponent. Slouží jako pomůcka při zadávání řetězcových kódů do textového pole.

Generátor gramatik

Generátor gramatik je podpůrný nástroj sloužící pro vytvoření bezkontextové gramatiky. Princip generátoru je popsán v kapitole 3.2. Nástroj lze spustit z menu *Soubor*.

- Vstupem jsou řetězce výchozích komponent. Každá komponenta se zadává na samostatný řádek.
- Výstupem je jednoznačná pravá lineární gramatika ve standardní formě generující všechna slova vstupních řetězců a jejich deformace.

7. Zhodnocení testovacích výstupů

Pro otestování navržené metody, byly sestaveno pět testovacích scén z množiny etalonů zobrazených na obrázku 3.5. Při sestavování scén byl brán ohled na vyzkoušení požadovaných parametrů rozpoznávání, kterými jsou:

- Klasifikace mírně deformovaných objektů.
- Označení více deformovaných objektů za neznáme.
- Zajištění invariance vůči natočení.
- Klasifikace objektu s vnitřní hranou.

Hlavním požadavkem byla rychlost zpracování. Zpracování scény musí probíhat v reálném čase, kde maximální doba výpočtu je jedna sekunda. Doba zpracování se odvíjí od zařízení, na kterém jsou testy prováděny. V tabulce 7.1 jsou uvedeny parametry testovacího zařízení mající vliv na rychlost zpracování.

Tabulka 7.1: Parametry testovacího zařízení

Operační systém	Windows 7 Professional Service Pack 1
Procesor	Intel Core2 Duo 2,00 Ghz
RAM	3GB
Typ systému	32bitový

Ke každé testovací scéně je uveden rozbor nalezených komponent. Jednotlivé sloupce rozboru uvádění tyto parametry:

- *Komponenta* - řetězcový kód nalezené komponenty
- *Klasifikace* - výsledek klasifikace
- *Výsledek* - zda byl objekt klasifikován správně

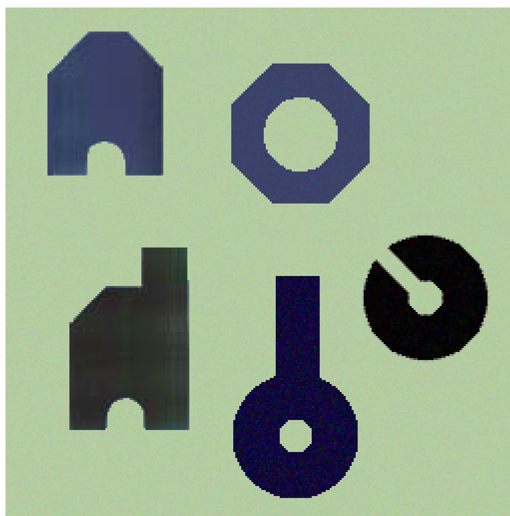
Nastavení

Pro všechny testovací scény bylo zvoleno stejné nastavení:

- Substituce = vypnuto
- Délka = 19
- Tolerance = 1

Testovací scéna 1

Komponenty na scéně tvoří etalony v základní poloze. Scéna slouží k určení výchozích hodnot rozpoznávání scény.



Obrázek 7.1: Testovací scéna 1

Tabulka 7.2: Vyhodnocení testovací scény 1

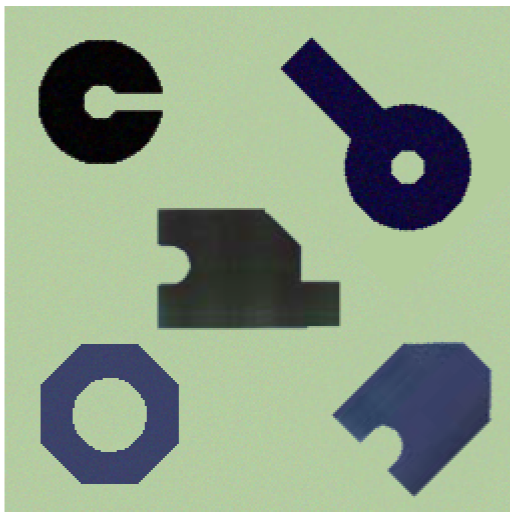
Komponenta	**Klasifikace**	**Výsledek**
CDEGRGAB	Etalon 4	✔
CDEFGHAB (R)	Etalon 5	✔
RERGRADRH	Etalon 2	✔
CEGRGABCA	Etalon 3	✔
CERERGRARA (R)	Etalon 1	✔

Testovací scéna 2

Scéna obsahuje natočené etalony.

Tabulka 7.3: Vyhodnocení testovací scény 2

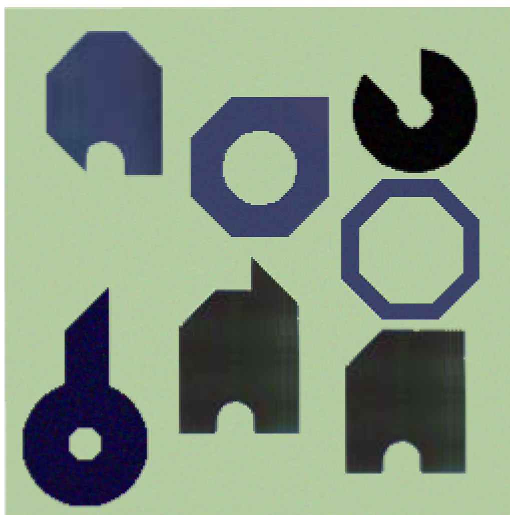
Komponenta	**Klasifikace**	**Výsledek**
DCRERAH (R)	Nenalezeno	⛔
RGRCRGRA	Etalon 2	✔
CDECEGGA	Etalon 3	✔
CDEFGHAB (R)	Etalon 5	✔
CEFFRB	Etalon 4	✔



Obrázek 7.2: Testovací scéna 2

Testovací scéna 3

Scéna obsahuje deformované komponenty.



Obrázek 7.3: Testovací scéna 3

Jak je možno v aplikaci z jednotlivých derivačních stromů vyčíst, tak syntaktická analýza pro všechny komponenty proběhla úspěšně. Avšak, komponenta *CDEFGHAB (CDEFGHA)* nebyla klasifikována, jelikož nebyl nalezen odpovídající vzor z množiny etalonů.

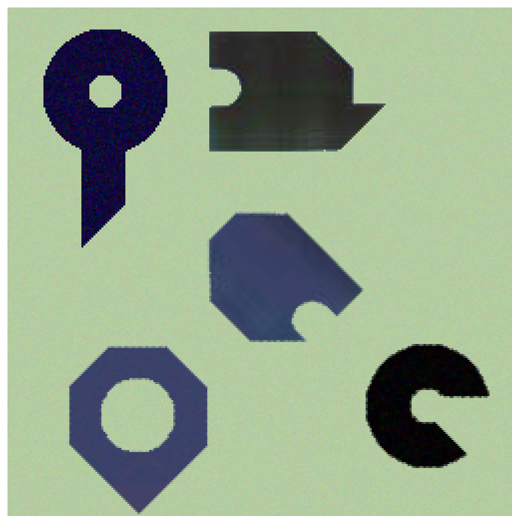
Komponenta nacházející se v testovací scéně vpravo dole s řetězcovým kódem *CEGRGAB* je původem deformovaný *Etalon 3*. Aplikace komponentu ovšem klasifikovala jako *Etalon 4*. To je způsobeno tvarem deformace, který komponentu přiblížil k rozpoznanému etalonu. Tudíž je klasifikaci možno považovat za správnou.

Testovací scéna 4

Scéna obsahuje natočené deformované komponenty.

Tabulka 7.4: Vyhodnocení testovací scény 3

Komponenta	**Klasifikace**	**Výsledek**
DEGRHAB	Etalon 4	✔
RERGRADRA	Etalon 2	✔
CEFGHAB (R)	Etalon 5	✔
CDEFGHAB (CDEFGHA)	Nenalezeno	✔
DEGRGABC	Etalon 3	✔
ERERGRARAB (R)	Etalon 1	✔
CEGRGAB	Etalon 4	✔



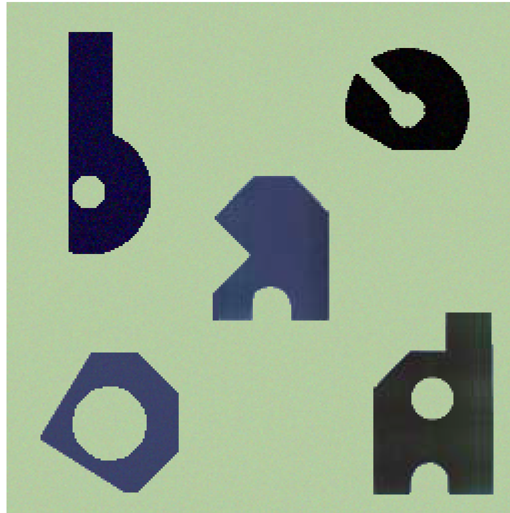
Obrázek 7.4: Testovací scéna 4

Tabulka 7.5: Vyhodnocení testovací scény 4

Komponenta	**Klasifikace**	**Výsledek**
REREFARA (R)	Nenalezeno	⛔
CDEFGA	Nenalezeno	⛔
CDRGHA	Etalon 4	✔
RGRDGRA	Nenalezeno	⛔
DEFHAB (R)	Etalon 5	✔

Testovací scéna 5

Scéna obsahuje značně deformované komponenty.



Obrázek 7.5: Testovací scéna 5

Tabulka 7.6: Vyhodnocení testovací scény 5

Komponenta	**Klasifikace**	**Výsledek**
CERERGARA	Nenalezeno	✔
REGA	Nenalezeno	✔
CDEGRGBHB	Nenalezeno	✔
CEGRGABCA (R)	Nenalezeno	✔
CDEF (R)	Nenalezeno	✔

7.1. Vyhodnocení výsledků

V tabulce 7.7 jsou uvedeny doby trvání jednotlivých kroků vyhodnocení pro testovací scény. Každá hodnota je průměrem z deseti provedených rozpoznání jednotlivých scén. Všechny hodnoty jsou uvedeny v milisekundách. Z tabulky je patrné, že požadavek na zpracování v reálném čase byl splněn.

Z celkového počtu dvaceti sedmi komponent bylo:

- dvacet tři klasifikováno správně
- čtyři klasifikovány chybně

Kritickým bodem rozpoznávání scény se ukázalo být vyhledání komponent. Jednotlivé primitivy vykazují při různých natočeních odlišné vlastnosti. Nastávají situace, kdy stejná komponenta, pouze jinak natočená, je reprezentována řetězcovým kódem, který je rozdílný. Tento problém lze kompenzovat zvětšením testovací scény.

Tabulka 7.7: Časové průběhy rozpoznání testovacích scén

Testovací scéna	Předzpracování scény	Nalezení komponent	Parsování komponent	Rozpoznání komponent	Σ
1	31	48	2	0	81
2	16	55	9	0	80
3	9	62	3	0	74
4	16	45	9	0	70
5	8	37	10	0	55
$\oslash$	16	49	7	0	72

Naopak navržená deformační gramatika vykazuje velmi dobré vlastnosti. Ve všech testovacích scénách nenastal případ, kdy by objekt byl klasifikován chybně z důvodu špatného průchodu derivačním stromem.

Aplikace se chová deterministicky a výpočetní doba je stálá.

7.2. Možná vylepšení

Pro zlepšení funkčnosti testovací aplikace je možné vykonat řadu věcí.

1. Zvýšení výpočetní rychlosti, je možné dosáhnout optimalizací aplikace pro více procesorové zpracování.

2. Pro možnost samostatného učení aplikace novým etalonům, implementovat generátor syntaktického analyzátoru. Takováto součást programu, by měla za důsledek možnost přidávat nové etalony v podstatě za běhu programu.

3. Vylepšení algoritmu rozpoznávání primitiv. Navržené schéma pro identifikaci jednotlivých primitiv, se ukázalo jako problematické. Pro lepší funkčnost programu by významnou měrou pomohlo tuto operaci upravit. To by však znamenalo celkovou změnu konceptu popisu obrazu.

8. Závěr

Tato práce pojednávala o možnosti využití syntaktických metod pro rozpoznávání obrazu. Základním cílem bylo analyzovat možnosti použití těchto metod a vytvořit optimalizovaný návrh pro dosažení maximální efektivity rozpoznávání. Především byl kladen důraz na rychlost zpracování v reálném čase. Tohoto cíle bylo bezezbytku dosaženo. Cílem bylo rozpoznávat testovací scény do jedné sekundy. Výsledný čas zpracování se přitom pohybuje v řádu desítek milisekund.

Hlavním přínosem této práce je vytvoření generátoru deformačních gramatik. Tento generátor je schopen z popisu objektů pomocí primitiv vytvořit gramatiku, která popisuje vstupní objekty a jejich do jisté míry deformované varianty.

Byly analyzovány možnosti automatické tvorby algoritmu pro popis a rozpoznání obrazu pomocí syntaktických metod a navrženo teoretické řešení s návrhem reálné aplikace.

Podařilo se zajistit invariance objekt na vstupní scéně vůči změně měřítka a natočení.

Testovací scénáře vykazují nedostatky v určování popisu objektu scény, avšak navržení řešení deformačních gramatik se jeví jako zcela funkční. Při rozpoznávání objektů nenastal případ, kdy by aplikace klasifikovala komponentu do špatné třídy. Nastaly případy, kdy nebyl objekt klasifikován vůbec, což je přisuzováno problému s vytvořením popisu.

Navržené testovací prostředí poskytuje dostatečnou míru informace pro zhodnocení postupů.

Literatura

[1] Dvořák, P.: *Popis objektů v obraze*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2011, 62 s., vedoucí diplomové práce Ing. Martin Zukal.

[2] Češka, M.; Vojnar, T.; Smrčka, A.: *Teoretická informatika*. FIT VUT, Brno, 2008.

[3] Fiřt, J.; Holota, R.: Digitalizace a zpracování obrazu. [online], poslední aktualizace 30. 9. 2002 9:04 [cit.2014-05-05].
URL http://home.zcu.cz/~holota5/publ/DigZpr0.pdf

[4] Freunhofer: Lexer and Parser Generators. [online], [cit.2015-05-10].
URL http://catalog.compilertools.net/lexparse.html

[5] Hlaváč, V.: Digitální obraz, základní pojmy. [online], [cit.2014-05-05].
URL http://cmp.felk.cvut.cz/~hlavac/TeachPresCz/11DigZprObr/014DigitalImageCz.pdf

[6] Hlaváč, V.: Hledání hran. [online], [cit.2014-05-05].
URL http://cmp.felk.cvut.cz/~hlavac/TeachPresCz/11DigZprObr/22EdgeDetectionCz.pdf

[7] Horák, K.: Computer vision. [online], [cit.2014-05-05].
URL http://midas.uamt.feec.vutbr.cz/ZVS/Exercise05/content_cz.php

[8] Kuijpers, H.: a Tiny Parser Generator v1.2. [online], [cit.2015-05-10].
URL
http://www.codeproject.com/Articles/28294/a-Tiny-Parser-Generator-v

[9] Linka, A.; Volf, P.; Košek, M.: Zpracování obrazu a jeho statistická analýza. [online], [cit.2014-05-05].
URL http://e-learning.tul.cz/cgi-bin/elearning/elearning.fcgi?ID_tema=67&stranka=publ_tema

[10] Meduna, A.; Luká, R.: *Formální jazyky a překladače IFJ Studijní opora*. FIT VUT, Brno, 2006.

[11] Meduna, A.; Lukáš, R.: *Formální jazyky a překladače IFJ Studijní opora řešené příklady s ilustrací*. FIT VUT, Brno, 2006.

[12] Microsoft: .NET Framework Regular Expressions. [online], [cit.2015-05-10].
URL
https://msdn.microsoft.com/en-us/library/hs600312%28v=vs.110%29.aspx

[13] Microsoft: Objektově orientované programování. [online], [cit.2015-05-10].
URL https://msdn.microsoft.com/cs-cz/library/dd460654.aspx

[14] Microsoft: Visual Basic Language. [online], [cit.2015-05-10].
URL
https://msdn.microsoft.com/en-us/library/aa903378%28v=vs.71%29.aspx

[15] Minařík, M.: *Strukturální metody identifikace objektů pro řízení průmyslového robotu*. Brno: Vysoké učení technické v Brně, Fakulta strojního inženýrství, 2009, 83 s., vedoucí prof. RNDr. Ing. Jiří Šťastný, CSc.

[16] Šťastný, J.: *Netradiční metody a algoritmy pro rozpoznávání objektů technologické scény, zkrácená verze habilitační práce*. FSI VUT, Brno, 2005, ISBN 80-214-3117-2.

[17] Wikipedia: LL syntaktický analyzátor. [online], poslední aktualizace 22. 2. 2015 19:56 [cit.2015-05-10].
URL http://cs.wikipedia.org/wiki/LL_syntaktick%C3%BD_analyz%C3%A1tor

[18] Wikipedie: Detekce hran. [online], poslední aktualizace 9. 11. 2013 21:56 [cit.2014-05-05].
URL http://cs.wikipedia.org/wiki/Detekce_hran

[19] Zezula, L.: Regulární výrazy. [online], poslední aktualizace 30. 6. 2004 19:56 [cit.2015-05-10].
URL http://www.zezula.net/cz/prog/regularni_vyrazy.html

A. Obsah přiloženého CD

Název	**Popis**
Recognizer	Složka s testovací aplikací. Aplikace se spouští souborem *Recognizer.exe*
DP_Valsa.pdf	Textová část práce
SourceCode.zip	Archiv obsahující zdrojové soubory testovací aplikace