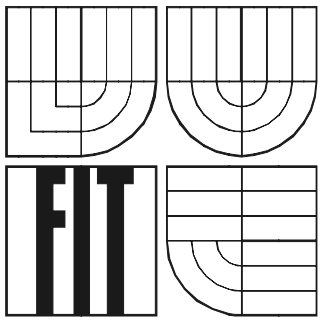
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ



Bakalářská práce

ECSS - Preprocesor jazyka CSS

2005 Jan Štefl

Abstrakt

Dle své definice nedisponuje jazyk CSS žádným preprocesorem. Důsledkem toho je, že je někdy potřeba se uchýlit k neefektivně psaným zdrojovým kódům. Máme-li například na různých místech souboru nadefinovánu konkrétní barvu a později se rozhodneme ji změnit, musíme tak učinit na všech místech ručně.

Tato práce si klade za cíl, předejít této nadbytečné činnosti a zavést do tohoto jazyka preprocesor zefektivňující psaní zdrojových kódů v něm.

Klíčová slova

preprocesor CSS, CSS, ECSS, efektivní zápis CSS

Poděkování

Na tomto místě bych rád poděkoval Prof. Ing. Tomášovi Hruškovi, Csc za výborné připomínky v době návrhu, dále pak Tomášovi Petříčkovi z diskusního fóra Builder.cz za jeho rady v době implementace.

Prohlášení

Prohlašuji, že jsem tento projekt vypracoval samostatně pod vedením Prof. Ing. Tomáše Hrušky, Csc a že jsem uvedl veškeré zdroje informací, ze kterých jsem v průběhu své práce čerpal.

V Brně 21.4. 2005 Jan Štefl

Abstract
According own definition language CSS not disposes of preprocessor. Sometimes we must depart to write code clumsiness. For example, we have at diffrent places defined concrete color and later we will decide to change this color we must do it at every place manually.

This project try to make codding CSS files more effective.

Keyword
preprocessor CSS, CSS, ECSS, effective writting CSS files

Obsah

Seznam obrázků

Seznam příkladů

1 Úvod

1.1 Vývoj značkovacích jazyků

1.1.1 SGML

S rozvojem informačních technologií vznikla potřeba předpisu, který by mohl poskytnout standardní architekturu pro vytváření, předávání, uchovávání a zpracování různorodých dokumentů v elektronické podobě. Požadavkem byla nezávislost, jak na softwarové, tak na hardwarové platformě, a to při dostatečné flexibilitě. Roku 1986, v rámci projektu ODA (Open Document Architecture) vnikl standard ISO 8879 -- SGML (Standard Generalized Markup Language),

Jedná se o první standardizovaný značkovací jazyk, resp. metajazyk. SGML umožňuje definovat vlastní značky pomocí tzv. DTD (Document Type Definition).

SGML je předně zcela otevřeným standardem nezávislým na platformách, výrobcích nebo aplikacích. Soubory SGML jsou ukládány jako text ASCII, což zajišťuje jejich použitelnost prakticky na libovolné počítačové platformě.

Jednou z nejznámější aplikací SGML jazyk HTML (Hypertext Markup Language).

1.1.2 HTML

První neoficiální verzi HTML vyvinuli Tim Berners-Lee a Robert Caillau v roce 1989. Tuto neoficiální verzi spojili s jednoduchým protokolem HTTP (HyperText Transfer Protocol). Prvním prohlížečem byl Mosaic vyrobený v NCSA (National Centre for Supercomputer Applications).

Oficiální HTML jazyk vznikl v první verzi v roce 1996, jako HTML 2.0. Vývoj na něm ustal po vydání verze HTML 4.01.

Velkou nevýhodou tohoto jazyka bylo, že od sebe neoddělovaly data a formátování. Vše bylo obsaženo v jednom dokumentu. Vznikla potřeba řešit tento fakt, protože práce na rozsáhlejší webové aplikaci se stala neefektivní a vlastní data byla obvykle zateměna množstvím formátovacích informací. Vzniká jazyk CSS (Cascading Style Sheet) a později XHTML (eXtensible HyperText Markup Language).

1.1.3 XML

SGML bylo velice flexibilní, aby vyhovělo různým požadavkům. To byla jeho velká přednost, ale zároveň i největší slabina. Vývoj aplikací, které by plně podporovaly SGML, byl velice nákladný.

Jazyk HTML byl zpočátku malou množinou elementů, které umožňovaly vytvářet hypertextové dokumenty. Výrobci prohlížečů postupně HTML rozšiřovali, aby se zavděčili uživatelům, kteří měli stále větší požadavky. Začaly tak vznikat problémy s kompatibilitou, které vyústily v nutnost vytvářet stránky v několika verzích pro různé verze prohlížečů.

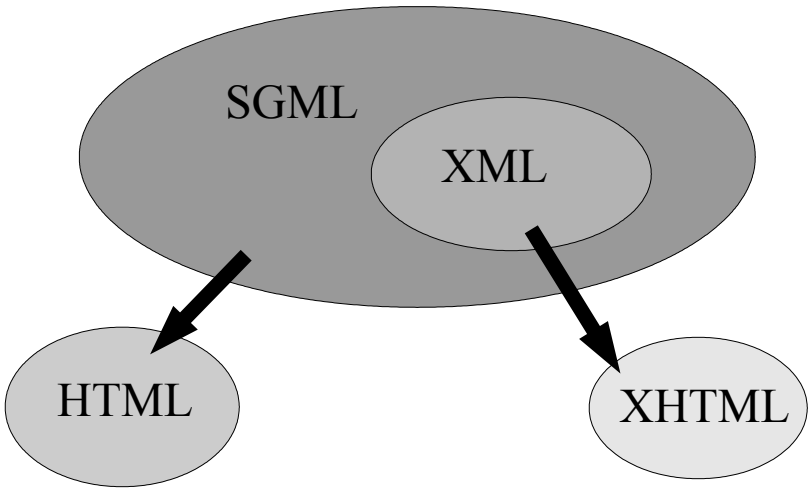
Bylo jasné, že současný stav je neudržitelný a že Web potřebuje novou technologii, která by byla mnohem flexibilnější než HTML. Výsledkem práce několika předních odborníků byl jazyk

XML, jehož finální verze byla publikována v únoru 1998. Jazyk XML je zjednodušenou podmnožinou SGML. Byl tak odstraněn největší problém SGML -- přílišná složitost. Flexibilita však zůstala zachována, a tak XML na rozdíl od HTML umožňuje tvorbu dokumentů s vlastními elementy a s možností validace díky přítomnosti DTD (resp. XSD Schema apod.).

V současné době se uvažuje o přebudování XML na XML 2.0. Důvodem je, že se definice asi ze 75% zabývá DTD, které je nahrazováno XSD Schematem případně Relaxem NG. Dále pak například externí entity jsou nahrazovány XInclude.

1.1.4 XHTML

V roce 2000 W3C definuje XHTML, zjednodušenou formu XML, šitou na míru pro web. U verze XHTML 1.0 je možno využívat veškeré atributy a elementy jazyka HTML 4.01. Zároveň je však třeba splňovat všechna základní pravidla XML. K nim například patří: Zákaz křížení elementů, nutnost uzavřít všechny (i prázdné) elementy, povinnost zapisovat atributy elementů do uvozovek resp. apostrofů apod.



Obr. 1-1: Vztahy mezi značkovacími jazyky

1.2 CSS

Základní myšlenka jazyka kaskádových stylů je oddělit formátování od vlastní struktury dokumentu. Dříve mohl HTML dokument obsahovat, dnes již zavrhnuté, značky jako <font>, <b> nebo <i>.

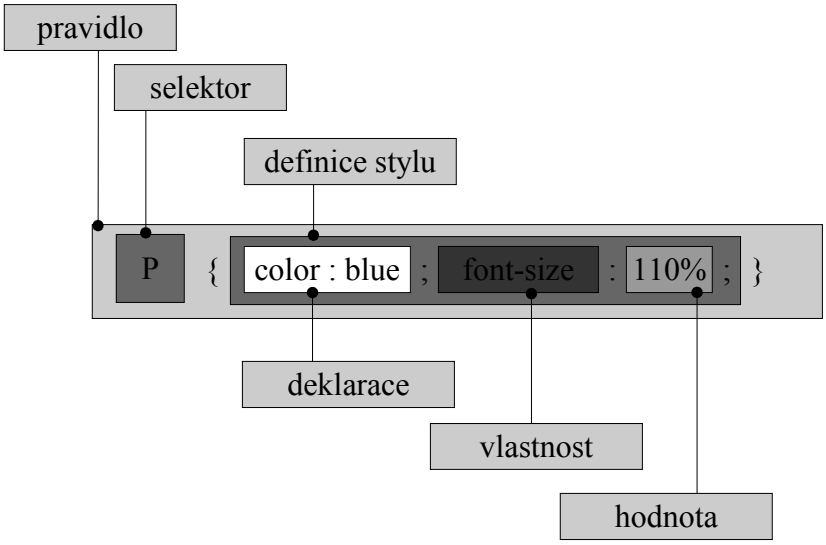
Při návrhu rozložení docházelo ke „zneužívání“ tabulek, které pro tento účel nebyly určeny. Jeden z hlavních problémů například byl, že se obsah stránky zobrazil až po načtení celé tabulky. Dalším neopomenutelným nepříjemným faktorem pak bylo, že zdrojový kód, je nepřehledný a těžko se provádějí pozdější úpravy.

S příchodem jazyka CSS (rok 1996) dochází k oddělení dat a jejich formátování. CSS umožňuje formátovat (X)HTML pro různá zařízení. Dále pak nabízí efektivní prostředky pro tvorbu rozložení.

V praxi je definice stylu umístěná v externím souboru a je připojena k jednotlivým (X)HTML, resp. XML dokumentům. Soubor se stylem bývá navíc ukládán do mezipaměti, čímž dochází ke zkracování doby nahrávání webových stánek.

1.3 Základní části dokumentu CSS

Celý styl ve kterém je definován vzhled nějakého dokumentu, je složen vždy alespoň z jednoho *pravidla*. To se následně skládá z dalších komponent jako *selektor* a *definice stylu.* Definice se pak skládá z *deklarace, vlastnosti a hodnoty.*



Obr. 1-2: Základní části dokumentu CSS

1.4 Vlastnosti jazyka CSS

1.4.1 Seskupování

V praxi je potřeba určit více deklarací pro jeden selektor. Jazyk CSS umožňuje tzv. seskupování těchto vlastností.

```
h1 {color: red}
h1 {font-variant: small-caps}
```

můžeme přepsat jako:

```
h1 {color: red; font-variant: small-caps}
```

Příklad 1-1: Seskupování v CSS

1.4.2 Selektory

Určují části dokumentu jejichž vzhled bude definován podle k nim přidruženého stylu. Selektory

mohou být již existující značky, například *p,* pro definici stylu odstavce. Dále pak můžeme vytvářet selektory vlastních jmen. Skládající se z tzv. *tříd* nebo *identifikátorů*.

V případě použití tříd se před její jméno zapíše tečka a v (X)HTML dokumentu se do patřičného tagu vloží atribut *class* s hodnotou požadované třídy.

Pokud chceme použít identifikátor, je zapotřebí jeho jméno začít symbolem „#“. V (X)HTML dokumentu pak v požadovaném tagu přidáme atribut *id* s hodnotou identifikátoru.

Konkrétní selektory mohou vypadat následně:

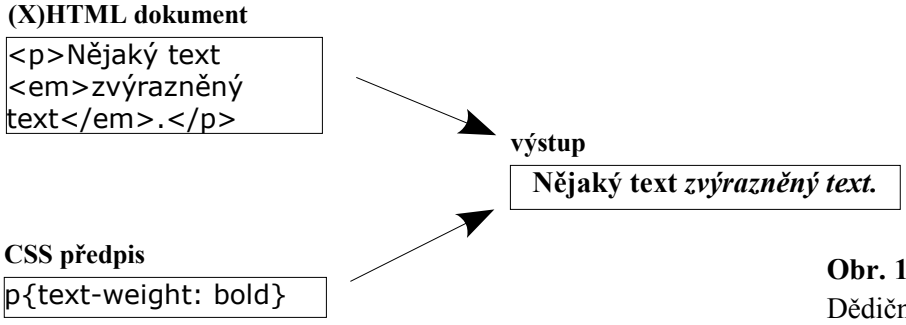
```
#hlavicka h1 { … }
#telo p em { … }
* { … }
a.bile em { … }
```

Příklad 1-2:
Selektory v CSS

V prvním selektoru definujeme styl pro všechny nadpisy první úrovně, které jsou součástí elementu, který je označen jako hlavicka. Druhý případ je obdobou prvního, s tím rozdílem, že je konkrétnější. Ve třetím případě určujeme vlastnost pro všechny elementy. V posledním případě sdělujeme, jak chceme nastylovat vše uvnitř elementu em, který je uvnitř odkazu třídy bile.

1.4.3 Dědičnost

Další charakteristickou vlastností, která je v tomto jazyce hojně využívána, je dědičnost. Umožňuje, pokud není řečeno jinak, aby element B který je obsažen v elementu A, dědil některé jeho vlastnosti. Můžeme tedy například snadno nadefinovat vlastnost společnou všem elementům tím, že ji nadefinujeme pro prvek *body*.



Obr. 1-3:
Dědičnost v CSS

1.4.4 Kaskáda

Kaskáda znamená stupňovitý vodopád. Podobným způsobem se chová i internetový prohlížeč, když v CSS předpisu vyhledává informace o výsledné podobě jednotlivých částí dokumentu. Prochází celým stylem a podle určitých pravidel vyhodnocuje, jak bude která část dokumentu zobrazena.

Pokud pravidla svými selektory oslovují jeden prvek, nastává konflikt. Platí zde pravidlo **přednosti konkrétnějších selektorů**. U rozsáhlejších selektorů je použito následující postup jak zjistit který předpis se má použít. Identifikátoru je přiřazena hodnota 100, třídám a pseudotřídám

10 a elementům 1. Selektor s nejvyšší hodnotou je nejkonkrétnější.

Pokud je po aplikování předchozího pravidla dosaženo shodných výsledků, přichází na řadu pravilo **přednost naposledy definovaného selektoru**.

Pokud předchozími postupy prohlížeč nezíská žádné informace o výsledném zobrazení určitého elementu, zjišťuje, zda prvek nezdědil hodnoty některých vlastností od nadřazených elementů. Pokud nezdědil, je použita výchozí hodnota definována standardem CSS.

1.5 Nevýhody jazyka CSS

Velkou nevýhodou tohoto jazyka je absence preprocesoru, tudíž pokud máme, například, na různých místech definovánu určitou barvu, a v jistém okamžiku, si ji přejeme změnit, jsme nuceni, tak učinit ve všech lokacích ručně.

```
#hlavicka{
 color: black;
}

#obsah{
 color: black;
}
```

Příklad 1-3:
CSS neumožňuje změnit obě barvy najednou jinak než manuálně

S předcházejícím problémem souvisí další problém. Nejsme schopni změnit odstín barvy se kterou pracujeme. Vždy bude dosazena tzv. bezpečná barva. Jenže může nastat situace, kdy je dostupný repertoár barev nedostačující a není umožněno zapsat následující.

```
green = #33cc33
h1{
 color: green
}
```

Příklad 1-4:
CSS nedisponuje proměnnými

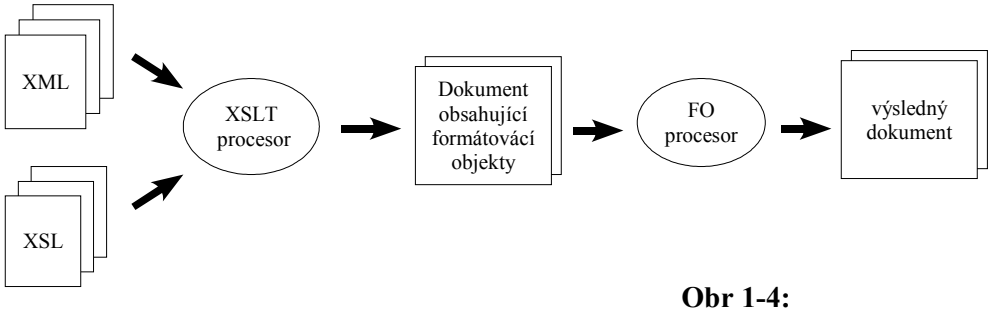
1.6 XSL

XSL (XML Stylesheet Language) je alternativa k jazyku CSS. XSL je jazyk, který transformuje XML do (X)HTML, filtruje a třídí XML data a umožňuje jejich formátování. XSL je popsáno jazykem XML. Skládá se ze třech částí XSLT, XPath a XSL-FO. XSLT transformuje XML dokumenty, XPath je užíván pro navigaci v nich a XSL-FO zajišťuje formátování. Pokud se mluví o XSL bývá obvykle myšleno XSL-FO.

Jazyk (X)HTML je složen ze značek, jejichž význam je jasně předem daný, tudíž stylovací jazyk CSS přesně ví, jakým způsobem k jednotlivým elementům přistupovat. V případě XML význam jednotlivých tagů jasný předem není a jsme nuceni dát nějakým způsobem najevo, jak se má s daným tagem naložit. Toho můžeme dosáhnout pomocí XSL.

XSL je komplexní stylovací jazyk, popsaný standardem XML. Proč je tedy v současnosti stále dominantní CSS? Jeden z důvodů je, že CSS je mnohem více čitelnější a momentálně poskytuje

dostatečné prostředky stylování (X)HTML dokumentů, které jsou v současnosti na webu nejrozšířenější. Dále je to také averze programátorů k novým technologiím.



Obr 1-4:
Formátování dokumentu pomocí XSL

1.7 ECSS – řešení nedostatků CSS

Následuje výčet kritérií, které jsem zohledňoval při návrhu jazyka ECSS.

1. V přítomnosti proměnné, která nebyla dříve definována, zpravit uživatele o její existenci a pokračovat dál ve zpracovávání.
2. Pokud dochází k překrytí již existující proměnné, informovat uživatele a pokračovat bez přerušení.
3. V případě, že se z nějakého důvodu nebudou příkazy pro preprocesor zpracovány, měly by zůstat pro webový prohlížeče skryty.
4. Snadno rozšířitelný pro eventuální validaci CSS souboru, případně na preprocesor obecného jazyka
5. Syntaxe jazyka by měla být intuitivní a zapadat do „filosofie“ syntaxe CSS.

1.8 Ukázka zdrojového kódu ECSS

```
/*--#
  define BORDERS    /*  komentář  */
  define KONTRAST

  ifdef BORDERS
     bord { border: 1px black solid; }
  else
     bord { border: 0 }
  endif

  ifdef KONTRAST
     black { #333333 }
     white { #FFFF99 }
     medium { 100% }
     large { 180% }
     normal { 500 }
     bold { 500 }
  else
     black { #FFFFFF }
     white { #355382 }
     medium { 105% }
     large { 180% }
     normal { 600 }
     bold { 700 }
  endif
*/

*{
  margin: 0;
  color: black;
  background-color: white;
}

p{
  font-family: Verdana, Helvetica, sans-serif;
  font-size: medium;
  font-weight: normal;
  bord
}
```

ECSS

CSS

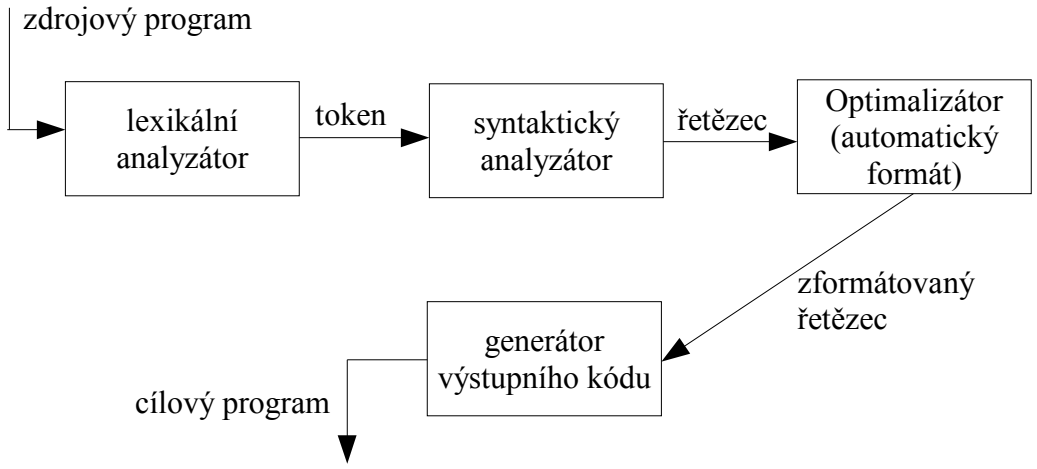
Příklad 1-5: Ukázka zdrojového kódu ECSS

2 Vnitřní struktura preprocesoru ECSS

Standardní součásti překladače jsou: Lexikální analyzátor, syntaktický analyzátor, sémantický analyzátor, generátor přechodného kódu, případný optimalizátor a generátor výstupního kódu.

Lexikální analyzátor zajišťuje zpracování vstupního souboru na lexikální elementy tzv. tokeny. Ty jsou předány syntaktickému analyzátoru, který na základě jejich typu posuzuje, zda je zdrojový kód zadán syntakticky správně, a buduje tzv. syntaktický strom. Syntaktický strom je dále předán generátoru přechodného kódu. Získaný přechodný kód putuje na vstup případného optimalizátoru a dále pak do generátoru výstupního kódu.

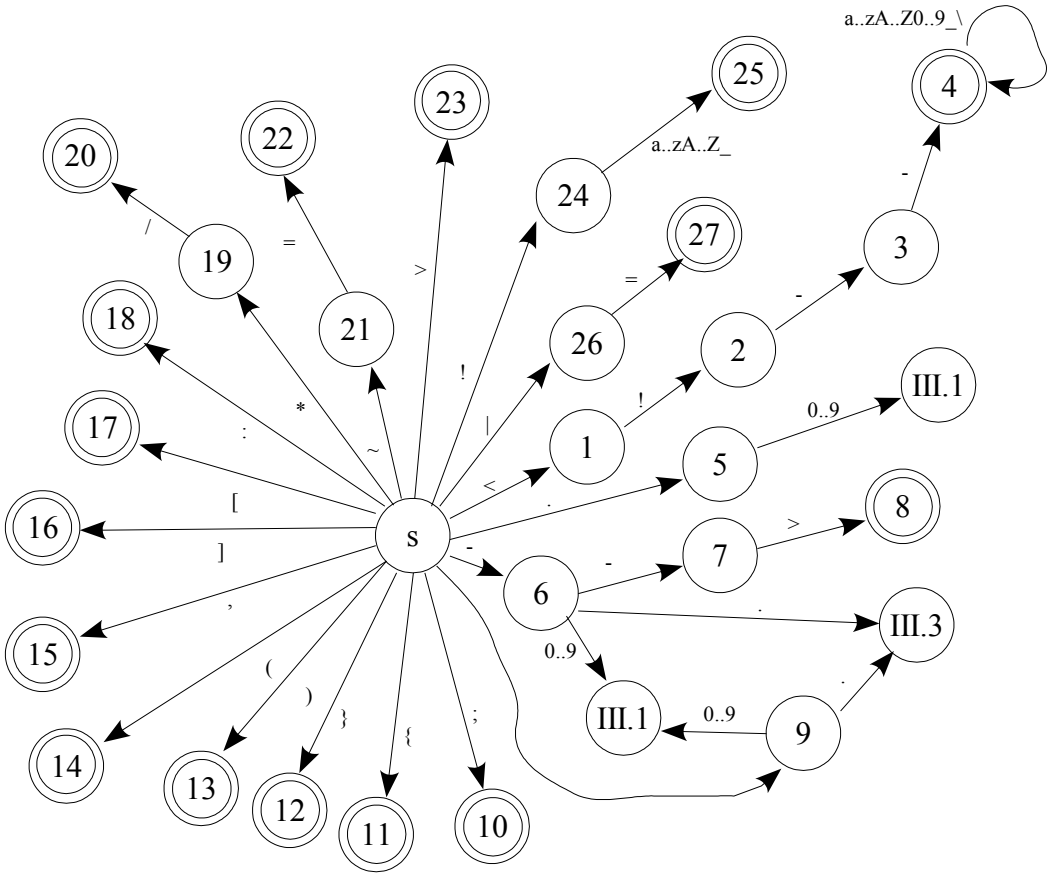
Součástí preprocesoru CSS jsou následující součásti: Lexikální analyzátor, syntaktický analyzátor, optimalizátor, plnící funkci automatického formátu a generátoru výstupního kódu zároveň. Sémantický analyzátor není zapotřebí z důvodu absence typů.



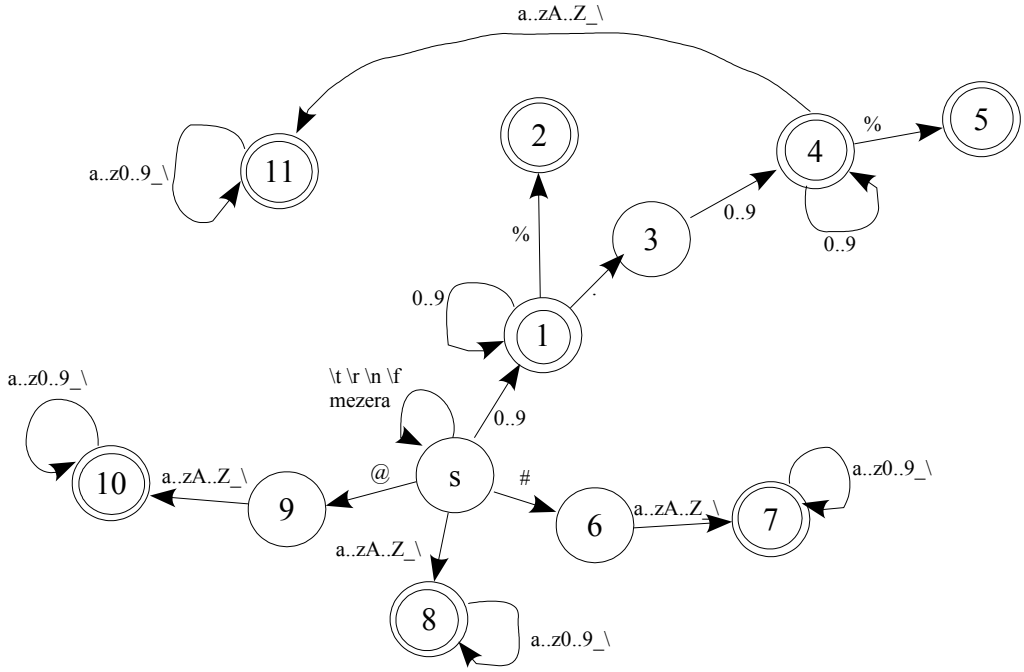
Obr. 2-1: Vnitřní struktura preprocesoru

2.1 Lexikální analýza

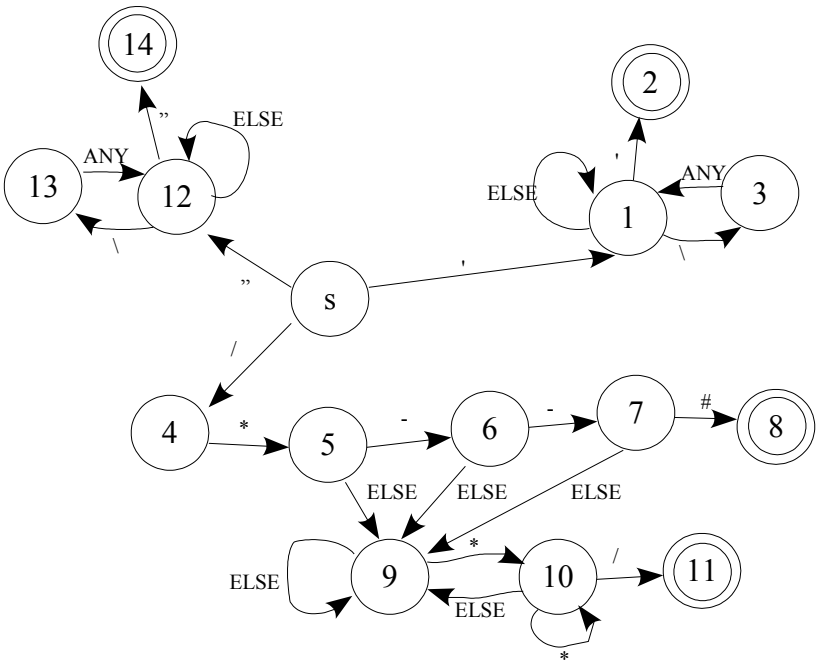
Lexikální analýzu zajišťuje lexikální analyzátor, tzv. scanner. Zpracovává vstupní symboly a sestavuje z nich tzv. lexémy a následně pak tokeny. Token je struktura, skládající se z položek *nalezená lexéma* a její *typ*. Výkonné jádro je tvořeno konečným automatem. Následují obrázky, které společně reprezentují jeden konečný automat popisující jádro scanneru.



Obr. 2-2: Konečný automat lexikální analýzy Část I / III



Obr. 2-3: Konečný automat lexikální analýzy Část II / III



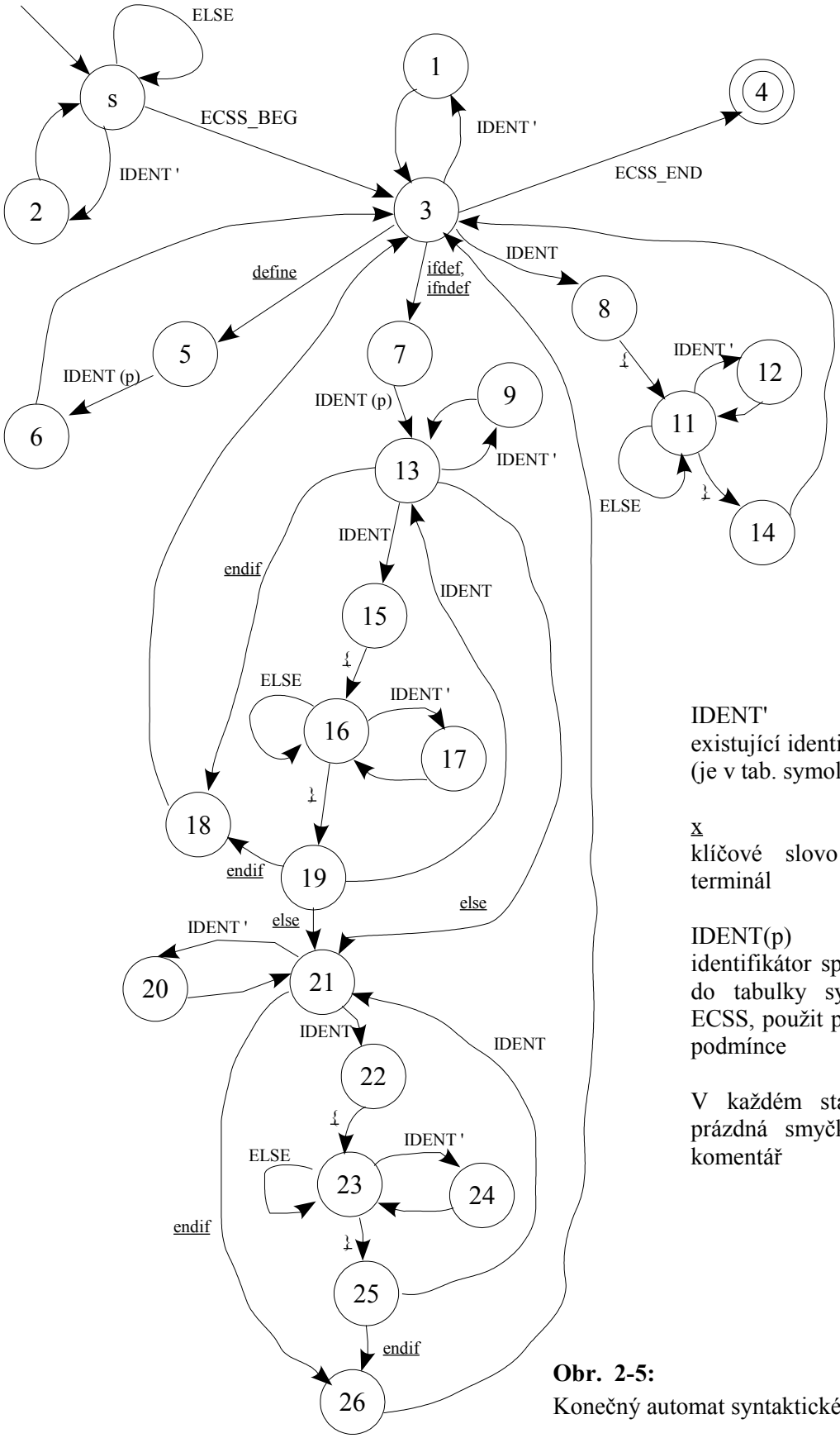
Obr. 2-4: Konečný automat lexikální analýzy Část III / III.

Poznámka: Název stavu „římská číslice.číslice“ identifikuje odkaz na část automatu a číslo stavu v něm.

2.2 Syntaktická analýza

Syntaktickou analýzu, tzv. parsing, zajišťuje syntaktický analyzátor. Jeho úkolem je sestavit lexikální jednotky, zdrojového programu, do gramatických frází. Gramatická fráze bývá zpravidla reprezentována syntaktickým stromem. Hierarchická struktura programu se obvykle vyjadřuje pomocí rekurzivních pravidel, zapsaných ve formě bezkontextové gramatiky.

Jazyk ECSS je regulární a proto pro jeho popis stačí regulární výrazy, resp. konečné automaty. Dále z toho také vyplývá, že všechny jazykové konstrukce jsou atomické, tzn. není umožněno příkazy zanořovat. Následující konečný automat reprezentuje gramatiku jazyka ECSS.



IDENT'
existující identifikátor (je v tab. symolů)

x
klíčové slovo nebo terminál

IDENT(p)
identifikátor spadající do tabulky symbolů ECSS, použit pouze v podmínce

V každém stavu je prázdná smyčka pro komentář

Obr. 2-5:
Konečný automat syntaktické analýzy

Poznámka: Do stavu 7 se dostaneme s terminály „ifdef“ a „ifndef“, je proto zapotřebí určit zda se bude následující blok programu vykonávat nebo zda se bude ignorovat. V programu je použito následující makro, které daný problém řeší, jedná se vlastně o ekvivalenci.

```
#define CAN_SAVE(pSymbolExist, condType) \
 ((((!(pSymbolExist)) || (condType))&&((pSymbolExist) || (!(condType))))
```

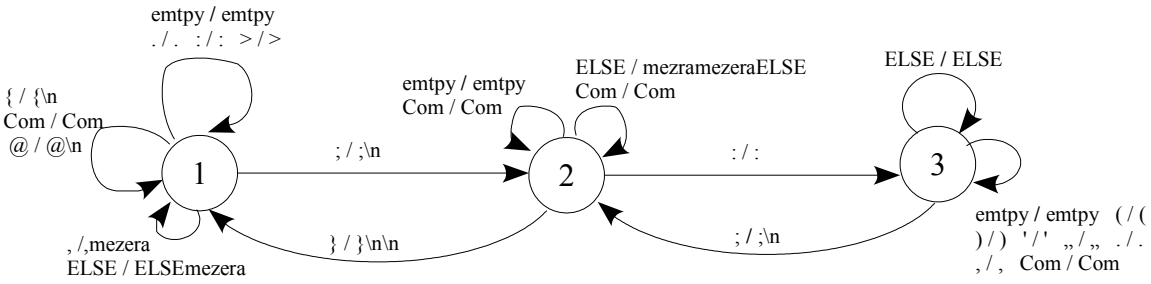
Příklad. 2-1: Makro CAN_SAVE

Parametr pSymbolExist sděluje zda je konstanta, následující po klíčovém slově ifdef, resp. ifndef, definována. Parametr condType určuje typ podmínky, hodnota false je určena ifndef a true pro ifdef. Při přejití do části else je hodnota vygenerována tímto makrem negována.

2.3 Optimalizátor

Typický optimalizátor plní funkci vylepšování mezikódu, který je následně předán generátoru výstupního kódu. Výsledkem jeho činnosti je generovaný optimálnější strojový kód.

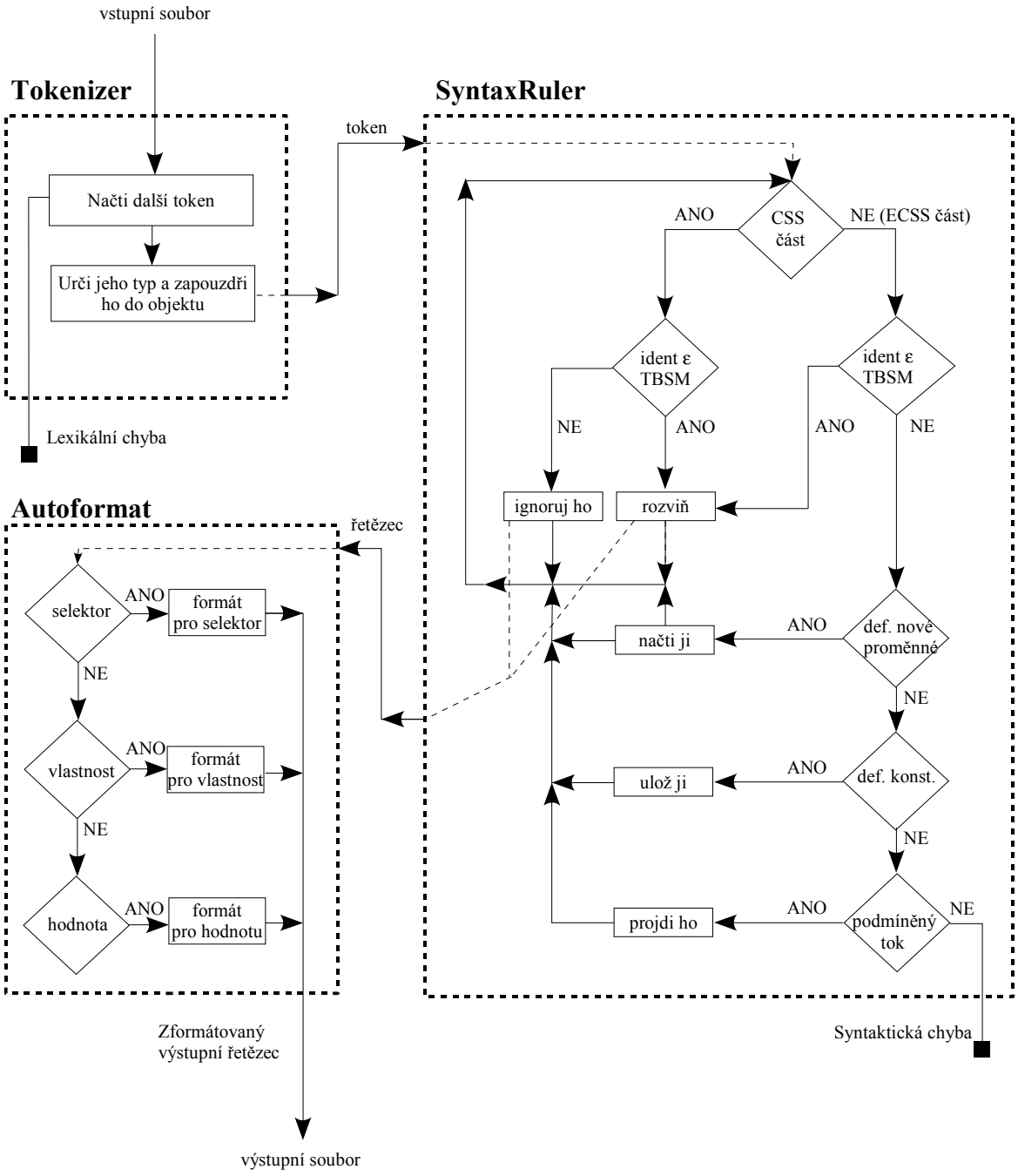
V případě jazyka ECSS zastává funkci automatického formátu, který způsobuje, že výsledný kód je z estetického hlediska optimální pro člověka. Na vstupu je vložena lexéma. Na výstupu ji obdržíme zformátovanou. Výstup je pak dále přesměrován do výstupního proudu. Jádro je tvořeno konečným automatem, viz následující obrázek.



Obr. 2-6: Konečný automat automatického formátu

3 Implementace

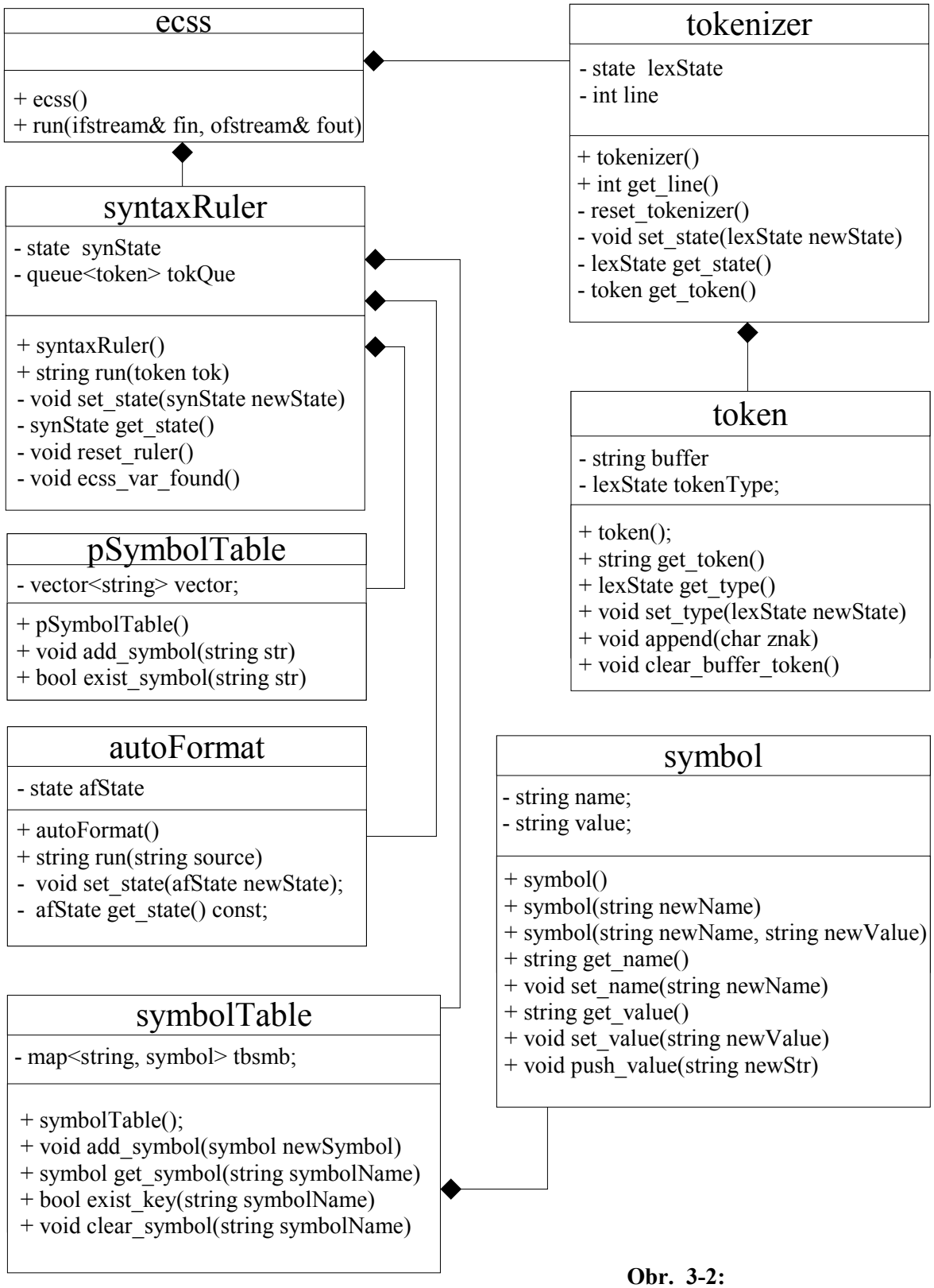
3.1 Vývojové schéma



Obr. 3-1:
Vývojové schéma preprocesoru CSS

Poznámka: TBSM je tabulka symbolů.

3.2 Diagram tříd



Obr. 3-2:
Diagram tříd preprocesoru CSS

4 Testování

Při testování projektu jsem se držel obecných zásad softwarového inženýrství. S přihlédnutím na rozsáhlost celého projektu jsem zvolil metodu černé skříňky tzv. black-box.

Testovány byli vždy jednotlivé celky a to vždy po jejich implementaci. Každý zdrojový soubor, uchovávající implementaci objektu obsahuje i zakomentovanou funkci main, tudíž je možné se kdykoliv vrátit k testování libovolné části programu.

Pro testování lexikálního analyzátoru jsem použil již hotové stylové předpisy a to ze zpravodajského serveru www.idnes.cz a ze stránek zabývajících se nezávislou grafickou tvorbou www.deviantart.com.

Dále jsem použil hlavičkový soubor „debug.hpp“, který obsahuje konstanty řídící úroveň ladění. Lze pomocí nich nastavit různě výpisy, nebo výpis pozastavit.

Zjištěné chyby byly podle uvážení, v závislosti na závažnosti a složitosti, opraveny a v případě, že ne, byly zaneseny do dokumentace.

Testovací vstupní data, pro jednotlivé součásti a pro celkový projekt, jsou umístěna na CD.

5 Známé chyby

5.1 Chybný převod hodnoty url

Tato chyba se projevuje jen v případě, že bylo použito zápisu typu url(odkaz.nekam) (bez uvozovek nebo apostrofů). Takto zapsané url se ve fázi automatického formátování převede na url (odkaz.nekam).

S přihlédnutím na závažnost této chyby jsem se rozhodl ji neopravovat. Zápis bez uvozovek, resp. apostrofu není totiž mezi všemi webovými prohlížeči interpretován správně.

Zápis s použitím apostrofů, resp. uvozovek, je automatickým formátem převeden správně a v současné době je plně podporován všemi prohlížeči.

6 Alternativy k jazyku ECSS

Jazyk ECSS není jedinou možností jak přistupovat k preprocessingu jazyka CSS. Následují dvě varianty, které jsem shledal jako nejzajímavější.

6.1 SSI na webserveru Apache

Jednou z možností jak do CSS zavést preprocesor je využití SSI (Server Side Includes) webového serveru Apache. Pro názornost uvedu fragment kódu.

```
<!--#set var="background" value="white" -->
<!--#set var="foreground" value="black" -->
body {
  background-color: <!--#echo var="background" -->;
  color: <!--#echo var="foreground" -->;
}
```

Příklad. 6-1: Předzpracování CSS pomocí SSI na webovém serveru Apache

Je potřeba následně upravit konfiguraci webserveru.

```
Options +Includes
AddType text/css .scss
AddHandler server-parsed .scss
```

Příklad. 6-2: Nastavení webového serveru Apache pro preprocessing CSS

Výhodou je, že není zapotřebí žádné další aplikace, vše probíhá na serveru. Nevýhodou je, že nemusíme mít možnost změnit nastavení serveru. Vlastní syntaxe by se též dala považovat za nevýhodu s ohledem na to, že je dost nepřehledná.

6.2 Využití serverového skriptovacího jazyka

Další, o něco zajímavější možností, je použít nějaký serverový skriptovací jazyk. Pro demonstraci jsem zvolil jazyk PHP. Opět následuje ukázka kódu.

```
<?php
  header('Content-Type: text/css');
  $background='white';
  $foreground='black';
  echo <<<_CSS_
    body {background-color: $background; color: $foreground}
  _CSS_;
?>
```

Příklad. 6-3: Využití PHP pro preprocessing CSS - soubor se stylem

```
<link rel="stylesheet" href="mystyle.css.php">
```

Příklad. 6-4: součást hlavičky souboru, ve kterém se má styl používat

Opět jako v předchozím případě, není zapotřebí žádné další aplikace, bude pouze stačit, aby na serveru, kde chceme stránky provozovat, byl spuštěný příslušný modul. V tomto případě modul jazyka PHP.

Nevýhoda této varianty spočívá v tom, že na serveru nemusí být zmiňovaný modul přístupný a dále pak neumožňuje předzpracování zdrojového programu „off-line“.

7 Závěr

Tato práce měla za úkol vyřešit problémy jazyka CSS. Hlavní problém spočíval v absenci preprocesoru, který by umožňoval definování symbolických konstant a podmíněné zpracování zdrojového programu. V následujících podkapitolách jsem učinil shrnutí toho co bylo vypracováno a nástin možných rozšíření aktuální verze programu.

7.1 Zhodnocení výsledků

V kapitole 1.5 jsem si vytyčil cíle, které jsem považoval za vhodné, aby je finální program splňoval. Nyní shrnu, do jaké míry se mi tyto cíle podařilo naplnit.

- **V přítomnosti proměnné, která nebyla dříve definována, zpravit uživatele o její existenci a pokračovat dál ve zpracovávání.**
  Bylo splněno, použita jsou varování, zmiňovaná v kapitole 3.4.

- **Pokud dochází k překrytí již existující proměnné, informovat uživatele a pokračovat bez přerušení.**
  Bylo splněno, použita jsou varování, zmiňovaná v kapitole 3.4.

- **V případě, že z nějakého důvodu nebudou příkazy pro preprocesor zpracovány, měli by zůstat pro webový prohlížeče skryty.**
  Bylo splněno, veškerý zápis jazyka ECSS je uschován v komentářích jazyka CSS.

- **Snadno rozšířitelný pro eventuální validaci CSS souboru, případně na preprocesor obecného jazyka**
  Program je napsán v jazyce C++, tudíž využívá objektově orientovaného přístupu, který jsem se snažil využít v co největší míře. Zdrojový kód je bohatě komentovaný a obsahuje programovou dokumentaci, neměl by tedy být problém stávající verzi dále rozšiřovat.

- **Syntaxe jazyka by měla být intuitivní a zapadat do „filosofie“ syntaxe CSS.**
  Zde se jedná spíše o subjektivní dojem každého uživatele tohoto jazyka.

7.2 Přínosy práce

Přínos jazyka ECSS spočívá v tom, že umožňuje efektivní práci se stylovacím jazykem CSS. Jeho výhodou dále je, že se dá snadno začlenit do libovolného vývojového prostředí a poskytuje okamžitý výstup, který můžeme použít kdekoliv. U předešlých alternativních řešení jsou jistá omezení, která jsou u nich uvedena.

7.3 Možnosti dalšího rozšiřování

Jazyk ECSS by do budoucna mohl umět vyhodnocování výrazů. Ať už by se jednalo o logické vyhodnocování v podmínkách či aritmetické operace.

Jazyk by se také mohl rozšířit o možnost hierarchie. V současné době neumožňuje definice konstant v podmínkách, či vnořené podmínky.

Dále by se jazyk ECSS dal, celkem snadno, rozšířit na preprocesor libovolného jazyka. Jediné co je potřeba znát, jak jsou v daném jazyce symbolizovány komentáře.

Zajisté by také nebylo špatné, kdyby program nabízel validaci výstupního souboru, tedy jazyka CSS.

Literatura

[1] Prokop Marek: CSS kaskádové styly pro webdesignéry. Brno, iDnes, 2003

[2] Grusová Marie: CSS kaskádové styly pro úplné začátečníky. Brno, Computer Press, 2003

[3] Doc. RNDr. Milan Češka, Csc., Doc. Ing. Tomáš Hruška, Csc., Ing. Miroslav Beneš: skriptum Překladače. Brno, 1999

[4] CSS2 syntax and basic data types, 1998
http://www.w3.org/TR/1998/REC-CSS2-19980512/syndata.html

[5] XSL-FO Tutorial,
http://www.w3schools.com/xslfo/default.asp

[6] Kompletní průvodce XSLT - úvod do problematiky, 2004,
http://interval.cz/clanek.asp?article=3301

[7] XML -- staronový formát, 1999
http://www.kosek.cz/clanky/xml/xml-historie.html

[8] SGML: Standard Generalized Markup Language, 1999
http://www.kosek.cz/clanky/cw/sgml.html

[9] Dočkáme se XML 2.0?, 2005
http://www.root.cz/clanky/akta-x-0502/

Přílohy

1 Stručný popis tříd

Tato část zevrubně popisuje význam jednotlivých tříd. Veškeré podrobnosti jsou obsaženy v programové dokumentaci.

1.1 Ecss

Třída nejvyšší abstrakce. Obsahuje jednu metodu „run“, která jako vstupní parametry přebírá referenci na vstupní a výstupní proud. Popisuje propojení mezi jednotlivými třídami. Jejím úkolem je též zachycení a ošetření výjimek.

1.2 Tokenizer

Zastává funkci lexikálního analyzátoru. Metoda „get_next_token“ je hlavní výkonná metoda, vracející nalezenou lexému zapouzdřenou do tokenu. Jako vstupní parametr přebírá referenci na vstupní proud. Metoda „get_line“ vrací aktuálně zpracovávaný řádek.

1.3 Token

Objekt reprezentující token. Členské proměnné udržují informaci o lexémě a jejím typu. Přidružené metody „append“ a „clear_token_buffer“ pracují se znakovým bufferem, do kterého se postupně ukládá lexéma ze vstupního proudu.

1.4 SyntaxRuler

Zajišťuje syntaktickou analýzu vstupního programu. Metoda „Run“ je hlavní výkonná. Jako vstupní parametr přebírá objekt typu token. V případě, že bylo nalezeno pravidlo odpovídající syntaxi a zároveň jsou splněna určitá kritéria (znázorněná na obr. 3-1), vrací tato metoda nezformátovaný řetězec určený pro finální výstup.

1.5 PsymbolTable

Uchovává informaci o tom, které konstanty byly v rámci ECSS bloků deklarovány. Na jejich základě pak probíhá podmíněný tok programu.

1.6 SymbolTable

Tabulka symbolů. Jsou do ní zaznamenávány všechny proměnné, které jsou v programu nadefinovány. Je zde uplatňována aktualizační sémantika. Z hlediska implementace se jedná o mapu symbolů. Klíčovou roli, pro získání hodnoty proměnné, zde sehrává její jméno. Hlavními metodami zde jsou „add_symbol“, vkládající nový symbol do tabulky na základě předávaného parametru typu symbol a „get_symbol“, která vrací požadovanou proměnnou na základě jejího jména, jenž je zde předáno jako vstupní parametr.

1.7 Symbol

Základní stavební jednotka tabulky symbolů. Členské proměnné uchovávají informaci o jménu proměnné a její hodnotě

1.8 AutoFormat

Zastává funkci automatického formátování. V případě nepřítomnosti tohoto objektu by byl výstupem jeden řádek výstupní posloupnosti. Hlavní výkonná metoda je „run“. Jako parametr přebírá výstupní řetězec z objektu syntaxRuler a na výstupu poskytuje tento řetězec upravený do uživatelem stravitelné podoby.

2 Výjimky a varování

Objekty tokenizer a syntaxRuler mohou v průběhu zpracování vyvolat výjimky či varování.

2.1 Výjimka xLexical

Původce: objekt tokenizer

Příčiny: zdrojový program obsahuje lexikální chybu

Řešení: zahlášena chyba a skončení programu

2.2 Výjimka xSyntax

Původce: objekt syntaxRuler

Příčiny: zdrojový program obsahuje syntaktickou chybu

Řešení: zahlášena chyba a skončení programu

2.3 Výjimka xBadCondConstruction

Původce: objekt syntaxRuler

Příčiny: chybně zapsaná podmíněná konstrukce, například je else bez předchozího ifdef

Řešení: zahlášena chyba a skončení programu

2.4 Výjimka xMissEndif

Původce: objekt syntaxRuler

Příčiny: chybí zapsané uzavírací endif na konci podmíněné sekce

Řešení: zahlášena chyba a skončení programu

2.5 Varování „Redefined variable“

Původce: objekt syntaxRuler

Příčiny: dochází k definici již definované proměnné

Řešení: zahlášeno varování, program pokračuje dál

2.6 Varování „Symbolic constant has been already defined.“

Původce: objekt syntaxRuler

Příčiny: definice konstanty která je již definována

Řešení: zahlášeno varování, program pokračuje dál, neznámý identifikátor není poslán dál

2.7 Varování „Unknown identifier“

Původce: objekt syntaxRuler

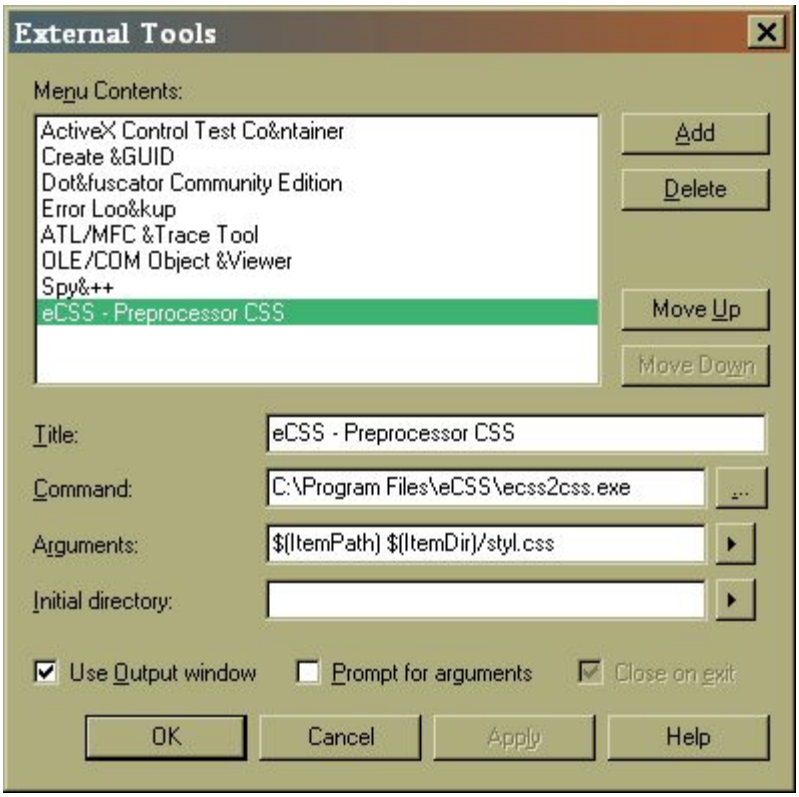
Příčiny: nalezen neznámý identifikátor, pravděpodobně by byl známý při splnění nějké konkrétní podmínky

Řešení: zahlášeno varování, program pokračuje dál, neznámý identifikátor není poslán dál

3 Začlenění do MS Visual Studia .NET

Pro efektivnější práci s ECSS je výhodné tuto aplikaci začlenit do nějakého většího vývojového prostředí. Součástí zadání byl požadavek spolupráce s MS Visual Studiem .NET.

V menu *Tools* vybereme *External Tools*. Obdržíme následující tabulku. Klikneme na tlačítko *Add.* Vhodně zvolíme titulek, nastavíme cestu a zadáme argumenty tak, jak je uvedeno na obrázku. Je též vhodné zaškrtnout volbu „Use Output window“, která zapříčiní, že všechen výstup externí aplikace je přesměrován do standardního výstupního okna vývojového prostředí.



Obr. 6-1:
Začlenění do
MS Visual Studia .NET