universiteit hasselt

Faculteit Wetenschappen
School voor Informatietechnologie

Keeping up with Google: Searching in Text Databases

Masterproef voorgedragen tot het behalen van de graad van master in de informatica/ICT/kennistechnologie

Jan Vuurstaek

Promotor: Prof. dr. Marc Gyssens

Assessoren: Jonny Daenen & Jelle Hellings

Prof. dr. Frank Neven

2de Master Informatica - Databases
Academiejaar 2013 - 2014

INHOUDSOPGAVE

LIJST VAN FIGUREN

LIJST VAN ILLUSTRATIES

DANKWOORD

Deze masterproef is niet alleen door mij tot stand gekomen. Daarom wil ik graag van de gelegenheid gebruik maken om iedereen te bedanken die mij hiermee rechtstreeks of onrechtstreeks heeft geholpen.

Als eerste wil ik mijn promotor Prof. dr. Marc Gyssens en mijn begeleiders Jonny Daenen & Jelle Hellings bedanken. Zij stonden steeds klaar om al mijn vragen te beantwoorden en hebben mijn tekst ook altijd van feedback voorzien.

Verder wil ik mijn ouders bedanken, die mij altijd gesteund hebben tijdens mijn studies.

Als laatste wil ik mijn broer Gert bedanken voor het nalezen van mijn masterproef.

Jan Vuurstaek

Diepenbeek, augustus 2014

ONDERZOEKSVRAAG

Het World Wide Web (WWW) is een immens grote verzameling van webpagina's die met elkaar verbonden zijn. Sommige van deze webpagina's bevatten grote hoeveelheden informatie, gerepresenteerd door middel van HTML tables, HTML lists en geformatteerde (HTML) opmaak. De laatste jaren zijn gebruikers steeds meer geïnteresseerd in dergelijke gestructureerde informatie, zowel voor informatieve doeleinden als voor verdere verwerking. Google speelt in op deze vraag door gestructureerde informatie aan te bieden bij een zoekopdracht. Om dit te kunnen doen, moeten ze de gestructureerde informatie eerst uit de webpagina's zien te halen. In deze masterproef bekijken we daarom verschillende invullingen van een techniek genaamd *wrapper inductie*, waarmee we dit kunnen doen. Wij focussen ons echter op één bepaald type gestructureerde informatie, namelijk relationele informatie. Dit is ook de meest voorkomende vorm van gestructureerde informatie. Bij het bestuderen van de verschillende invullingen besteden we aandacht aan de situaties die door de invullingen verwerkt kunnen worden, alsook aan de situaties die onontvankelijk zijn voor deze invullingen. Het type masterproef is dan ook een *gevalsanalyse*. De vraag die in deze masterproef centraal staat is:

"Hoe kunnen we relationele informatie uit webpagina's halen?"

Relationele informatie, en gestructureerde informatie in het algemeen, uit webpagina's halen levert verscheidene voordelen op. Zo kunnen we het web op een verfijnde manier queryen, door gestructureerde data tussen de zoekresultaten te tonen, zoals hierboven beschreven werd voor Google. Daarnaast kunnen we de relationele informatie ook als echte relationele data behandelen, waardoor we reeds bestaande database technieken op de data kunnen toepassen. Een voorbeeld van zo'n database techniek is het combineren (joinen) van verschillende tabellen om tot een gedetailleerder geheel te komen. De analyse van grote hoeveelheden gestructureerde informatie maakt ook de ontwikkeling van applicaties zoals *schema auto-complete* mogelijk. Schema auto-complete helpt database designers bij het maken van een database schema. Wanneer een gebruiker een attribuut opgeeft, dan geeft de applicatie suggesties van gerelateerde attributen.

ABSTRACT

Op het internet kunnen we grote hoeveelheden informatie terugvinden. Deze informatie kan gaan over films van een producer, een album van een artiest, winkelproducten, etc. In de meeste gevallen zijn gebruikers slechts geïnteresseerd in een beperkt deel van de informatie die voorkomt op een webpagina, bijvoorbeeld de tracklist van een album van een artiest. Gebruikers willen dat deze informatie eenvoudig beschikbaar is op een gestructureerde manier. De gewenste informatie wordt echter vaak ingesloten door extra informatie. Denk bijvoorbeeld aan een Wikipedia webpagina van een album van een artiest. Enkel de tabel onderaan op de webpagina toont welke tracks op het album staan. De overige informatie op de webpagina heeft betrekking op de personen die aan het album hebben meegewerkt, hoe het album werd gepromoot, hoe het album werd ontvangen, etc. We kunnen stellen dat de gebruiker voornamelijk geïnteresseerd is in gestructureerde informatie die meestal voorkomt in tabellen, lijsten, infoboxen, etc.

Internetgigant Google speelt op twee manieren in op deze vraag naar gestructureerde informatie. Voor de eerste manier maken ze gebruik van hun zoekmachine. Wanneer een gebruiker bijvoorbeeld zoekt op de naam van een album en de albumartiest, dan verschijnt er rechts bovenaan een infobox die beknopt de belangrijkste metadata van het album weergeeft, gevolgd door de tracklist van het album. Een tweede manier waarmee Google inspeelt op deze vraag is door middel van Fusion Tables. Gebruikers willen namelijk niet alleen gestructureerde informatie zien, maar willen vaak ook met de informatie werken. Fusion Tables is een web applicatie die dit toelaat. Een gebruiker kan namelijk een tabel met gestructureerde informatie aanmaken en opslaan in de cloud. De gebruiker kan vervolgens kiezen of hij deze informatie wil delen met andere gebruikers. Indien hij dit doet kunnen andere gebruikers zijn data combineren met hun eigen data om gestructureerde informatie met een hogere kwaliteit te bekomen.

De gestructureerde informatie komt echter niet altijd uit een achterliggende database, in tegenstelling tot wat men misschien verwacht, maar is soms ingebed in de webpagina zelf. Vooraleer we de informatie op een eenvoudige en gestructureerde manier kunnen aanbieden aan gebruikers, moeten we deze eerst uit de webpagina's zien te halen. Dit kunnen we doen aan de hand van *wrappers*. Een wrapper is een procedure ontwikkeld om informatie uit een specifieke soort webpagina's te halen. Dit betekent dat we voor elke soort webpagina's een nieuwe wrapper moeten schrijven. Deze methode is echter niet schaalbaar vanwege de grote hoeveelheid verschillende soorten webpagina's die beschikbaar zijn op het web. Vanwege deze reden bestuderen we in deze masterproef

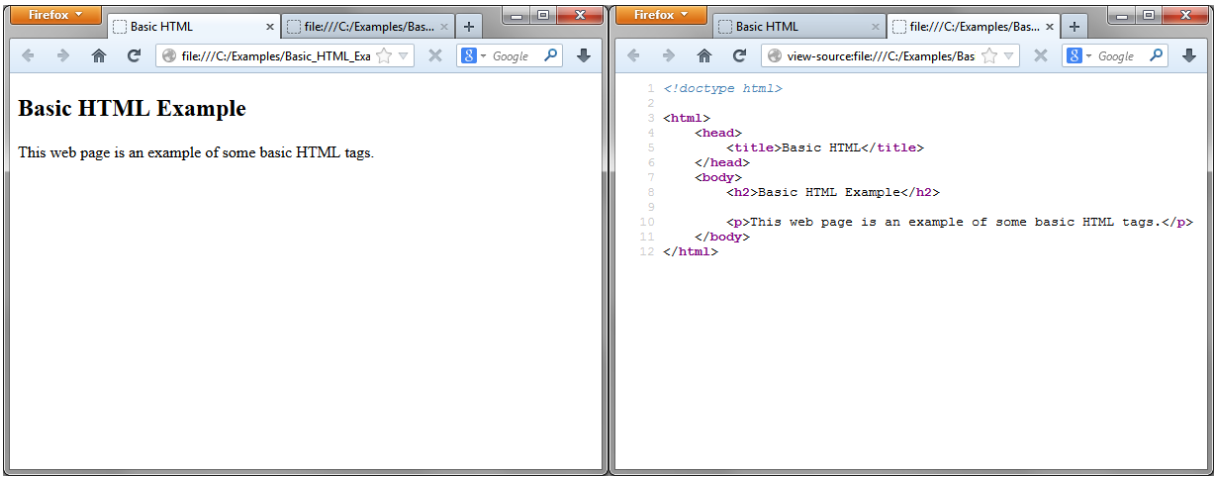
een techniek waarmee we de creatie van wrappers kunnen automatiseren. Deze techniek draagt de naam *wrapper inductie*.

We beginnen deze masterproef met het bestuderen van het informatie extractieproces, alsook hoe het wrapper inductieprobleem hierin past. Daarna bekijken we een aantal bestaande wrapper classes die dit probleem oplossen. Vervolgens bespreken we de beperkingen van deze wrapper classes, waarna we een eigen ontwikkelde wrapper class bespreken die met een aantal van deze beperkingen overweg kan. Daarnaast hebben we eveneens twee verbeteringen ontwikkeld die we kunnen doorvoeren aan de algoritmes van de wrapper classes.

We hebben de verschillende wrapper classes en de verbeteringen ook getest. Hieruit is gebleken dat de meeste wrappers gevonden worden wanneer onze gemeenschappelijke strings verbetering actief is. Indien deze verbetering niet actief is, worden enkel wrappers gevonden door de eenvoudigste wrapper class, genaamd LR. Wanneer men bijgevolg van de meer robuustere wrapper classes gebruik wil maken, is onze verbetering onmisbaar. Uit de experimenten is ook gebleken dat onze eigen ontwikkelde wrapper class, genaamd UOCLR, te strenge eisen heeft, waardoor deze niet binnen aanzienlijke tijd eindigt voor grote webpagina's.

1. INLEIDING

Het World Wide Web (WWW) is een immens grote verzameling van webpagina's die met elkaar verbonden zijn. Deze webpagina's kunnen opgevraagd worden door het ingeven van een URL in de adresbalk van een webbrowser. Om de webpagina's van structuur te voorzien, wordt gebruik gemaakt van een *markup language*. Markup languages worden met andere woorden gebruikt voor het annoteren van een document, waarbij de annotatie onderscheidbaar is van de standaard tekst. De meest gebruikte markup language voor webpagina's is HTML [1]. Deze markup language maakt gebruik van *HTML tags* voor het annoteren van een document. Dit zijn keywords die tussen angle brackets geplaatst worden. Een voorbeeld van een eenvoudige webpagina, samen met de bijhorende HTML code, wordt weergegeven in Figuur 1.1.



FIGUUR 1.1: EEN EENVOUDIGE WEBPAGINA.

Naast het toevoegen van structuur aan webpagina's, kunnen we webpagina's ook voorzien van opmaak. De opmaak wordt meestal beschreven in een *style sheet language*. Deze languages laten ons toe lettertypes te veranderen, achtergronden toe te voegen, de positie van elementen op de webpagina in te stellen, etc. Style sheet languages geven met andere woorden aan hoe de verschillende annotaties opgemaakt moeten worden. De style sheet language die gebruikt wordt voor webpagina's is CSS [2].

Tot slot kunnen websites interactiever gemaakt worden door gebruik te maken van *scripttalen*. Hiermee kunnen we bijvoorbeeld een klok weergeven op een webpagina. Veel gebruikte scripttalen zijn Javascript [3] [4] en PHP[1]. In Figuur 1.2 zien we een webpagina die HTML, CSS en JavaScript gebruikt.

[1] Voor meer informatie over PHP zie https://php.net/.

FIGUUR 1.2: EEN GEAVANCEERDERE WEBPAGINA.

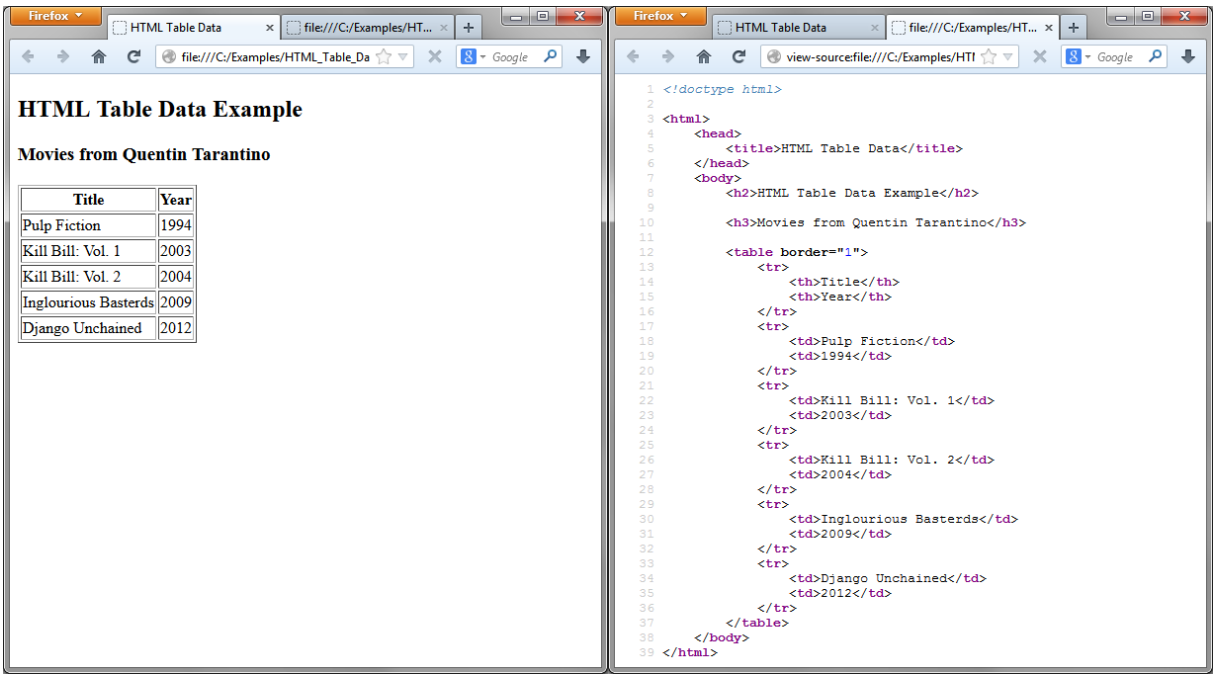
Sommige webpagina's geven grote hoeveelheden informatie weer. Deze informatie wordt echter niet altijd rechtstreeks tussen de HTML code geschreven, maar bevindt zich soms in een achterliggende database. Een voorbeeld hiervan is het resultaat van een zoekopdracht op Google. De resultaatpagina die je terugkrijgt is een lijst van sites die mogelijk aan je zoekopdracht voldoen. Google heeft echter geen webpagina's klaar staan voor elke mogelijke zoekopdracht, maar genereert deze dynamisch, door de webpagina's die aan je zoekopdracht voldoen uit de database te halen. Een database is dus een gestructureerde collectie van data. Het is mogelijk om een database te ondervragen met behulp van een *query language*, zoals SQL. Een voorbeeld van een query die we aan een filmdatabase kunnen stellen is: "Geef alle films van Quentin Tarantino die voor het jaar 2000 zijn uitgebracht." Als we deze query naar SQL vertalen krijgen we Illustratie 1.1.

Gestructureerde informatie vinden we op de eigenlijke webpagina altijd terug in de vorm van HTML tables (Figuur 1.3), in de vorm van HTML lists (Figuur 1.4), in de vorm van

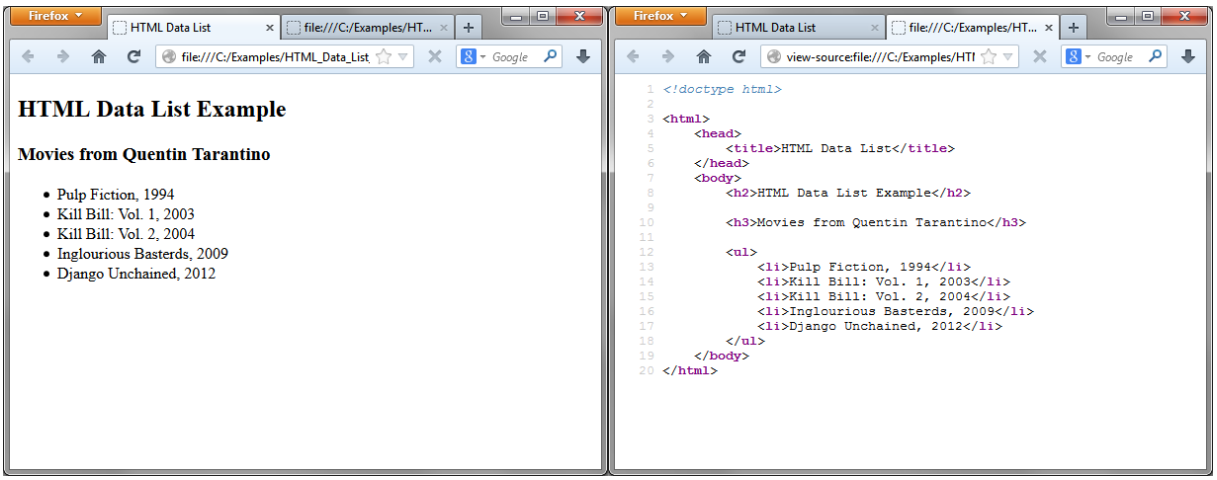
HTML formatting (Figuur 1.5), of in de vorm van tekst formatting (Figuur 1.6). Merk op dat HTML tables en HTML lists specifieke varianten zijn van HTML formatting. HTML formatting is op zijn beurt een specifiekere variant van tekst formatting.

```
1: SELECT   movieTitle, movieYear
2: FROM     Movies
3: WHERE    movieProducer = 'Quentin Tarantino'
4:          AND movieYear < '2000'
```

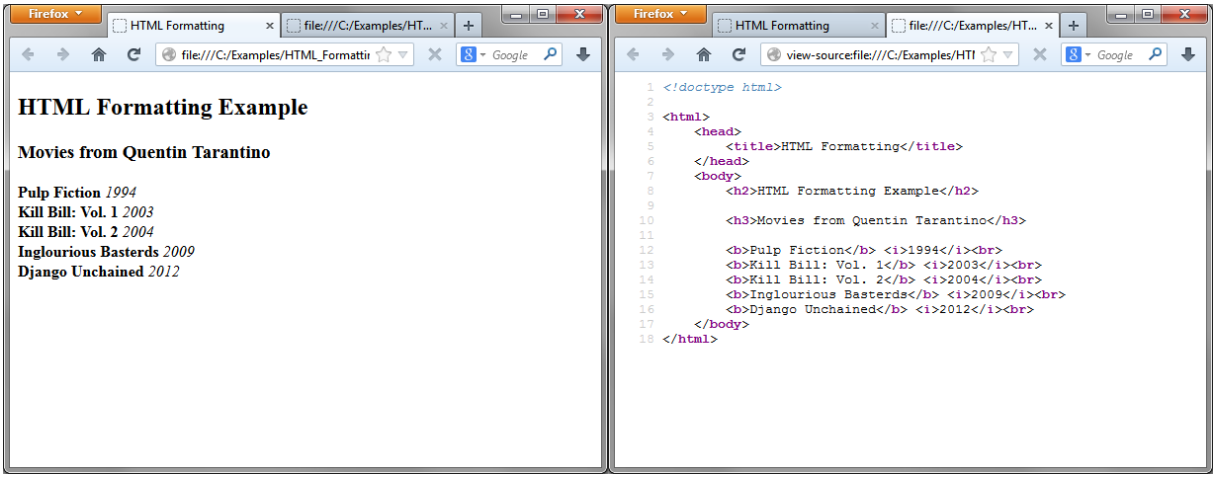
ILLUSTRATIE 1.1: QUERY DIE ALLE FILMS, GEPRODUCEERD DOOR QUENTIN TARANTINO EN UITGEBRACHT VOOR HET JAAR 2000, OPVRAAGT.



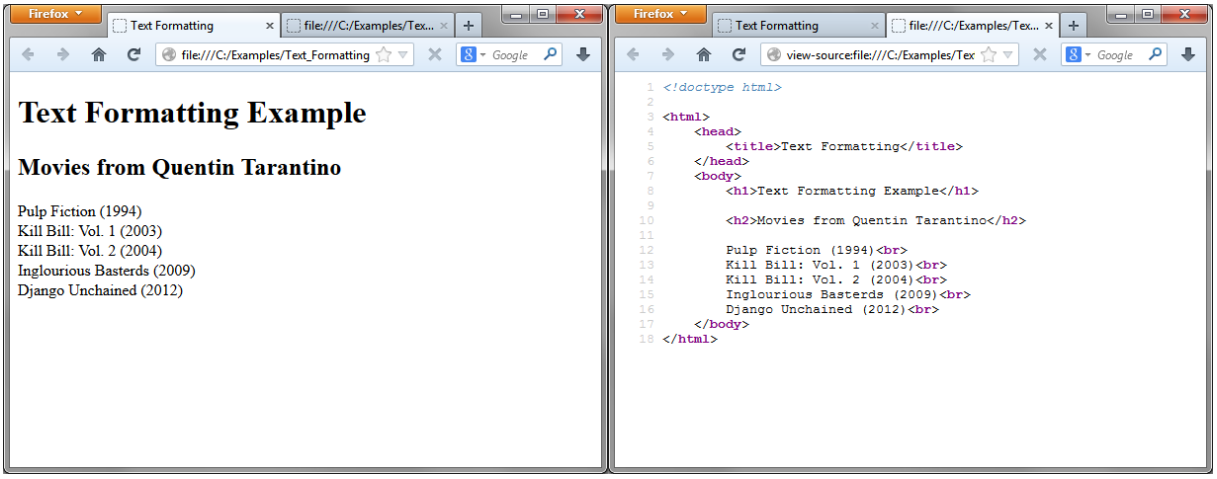
FIGUUR 1.3: EEN HTML TABLE DIE GESTRUCTUREERDE INFORMATIE BEVAT.



FIGUUR 1.4: EEN HTML LIST DIE GESTRUCTUREERDE INFORMATIE BEVAT.



FIGUUR 1.5: HTML FORMATTING GEBRUIKT VOOR HET WEERGEVEN VAN GESTRUCTUREERDE DATA.

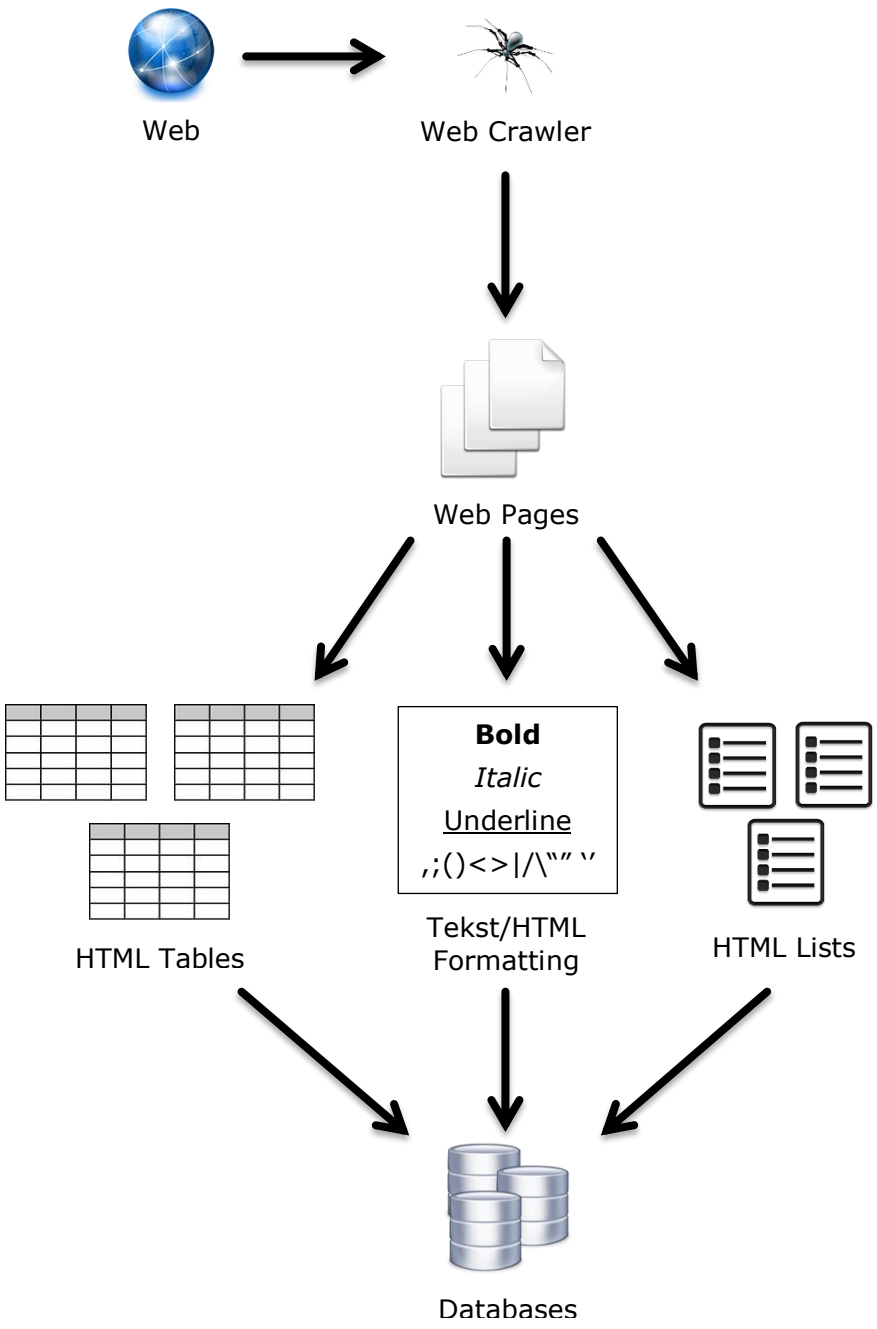


FIGUUR 1.6: TEKST FORMATTING GEBRUIKT VOOR HET WEERGEVEN VAN GESTRUCTUREERDE DATA.

De gestructureerde informatie die hardcoded in de HTML code voorkomt, is degene die ons interesseert. Dit type informatie komt namelijk veelvuldig voor op webpagina's, maar is tot op heden nog niet eenvoudig beschikbaar om te queryen. Dit komt omdat de standaard aanpak bij zoeksystemen, waarbij de relevantie van een webpagina bepaald wordt aan de hand van het aantal voorkomens van het zoekwoord, geen rekening houdt met de mogelijkheid dat een webpagina gestructureerde informatie bevat. Dit heeft als gevolg dat deze standaard aanpak een webpagina als niet relevant kan beschouwen, wanneer deze dat wel is. De reden waarom dit gebeurt wordt kort besproken op het einde van Sectie 1.1. In deze masterproef bekijken we daarom enkele methoden die de gestructureerde informatie uit een webpagina kunnen halen en vervolgens kunnen omzetten in databases. In het overige deel van deze inleiding geven we een algemeen overzicht van de stappen die in een dergelijk proces voorkomen.

1.1 INFORMATIE EXTRACTIE EN OMVORMING TOT DATABASES

Om van een webpagina met gestructureerde informatie tot een database te komen die deze informatie bevat, moeten we over een extractieproces beschikken. Dit proces bestaat uit vier stappen en wordt gevisualiseerd in Figuur 1.7. Hieronder wordt elke stap van dit proces in detail besproken.



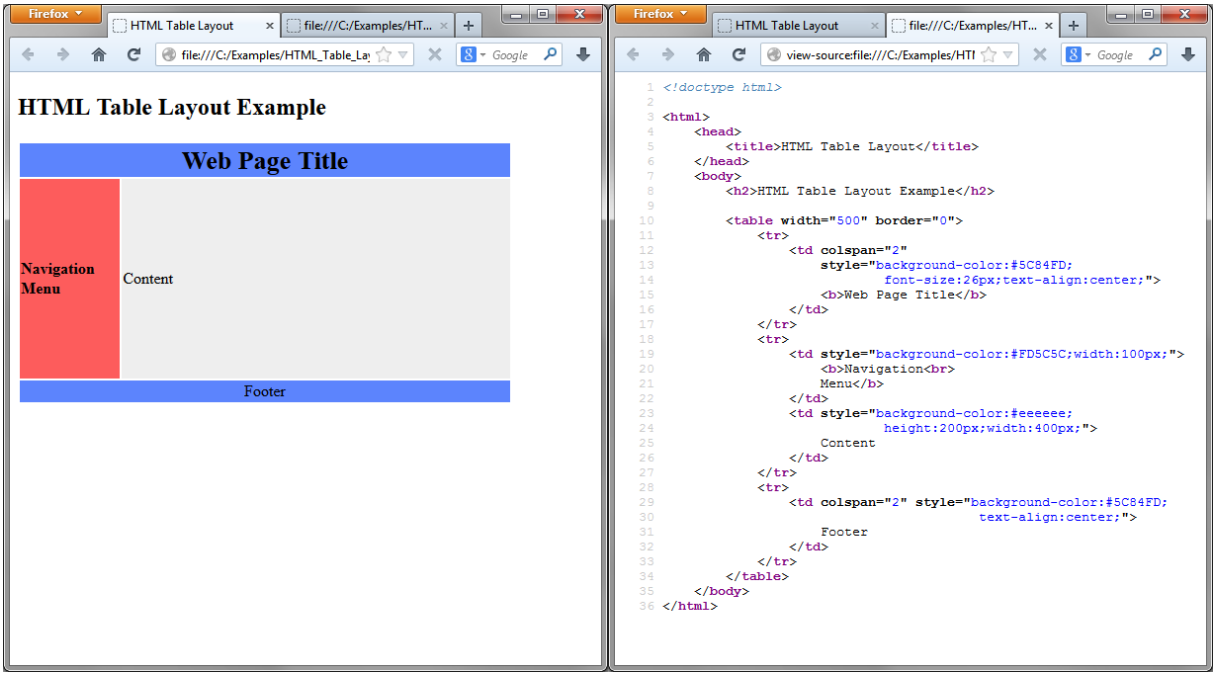
FIGUUR 1.7: HET INFORMATIE EXTRACTIEPROCES.

STAP 1: CRAWLING

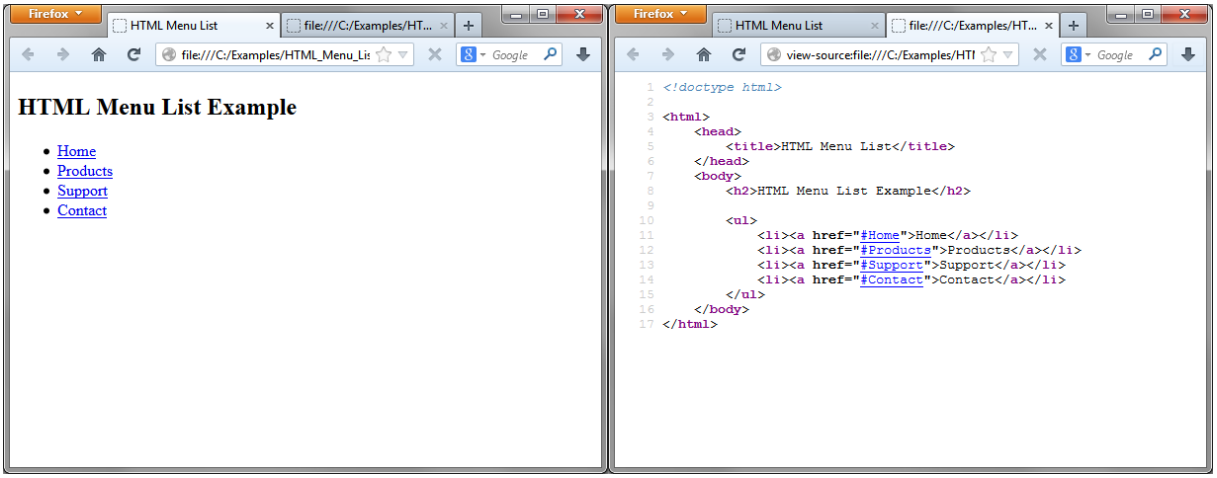
Het proces start met het zoeken naar webpagina's die gestructureerde informatie bevatten. Om deze webpagina's te vinden, kunnen we gebruik maken van web crawlers. Deze web crawlers, ook wel web spiders genoemd, scannen het web op een gestructureerde en geautomatiseerde manier, op zoek naar gestructureerde informatie. De web crawlers moeten met andere woorden de webpagina's die ze tegenkomen filteren, zodat we enkel webpagina's overhouden die gestructureerde informatie bevatten. Dit filteren kan gebeuren door te kijken welke webpagina's HTML tables of HTML lists bevatten, door op zoek te gaan naar hun bijhorende tags. HTML tables kunnen slechts met één tag aangemaakt worden, namelijk "`<table>`". Voor HTML lists zijn er verschillende tags voorhanden, namelijk "`<ul>`" (unordered list), "`<ol>`" (ordered list) en "`<dl>`" (descriptional list). Gestructureerde informatie kan, zoals we in de inleiding van dit hoofdstuk hebben gezien, ook verborgen zitten in geformatteerde HTML of tekst opmaak. We spreken van verborgen informatie, omdat het uit de HTML code niet triviaal duidelijk is wat de gestructureerde informatie is. Het detecteren van deze verborgen gestructureerde informatie vereist bijgevolg gesofisticeerde technieken. We kunnen bijvoorbeeld gebruik maken van machine learning [5], waarbij een verzameling van webpagina's wordt opgedeeld in een training set en test set. De training set wordt gebruikt om een patroon of structuur te herkennen, en de test set wordt gebruikt om het aangeleerd patroon te testen.

Wanneer we webpagina's hebben gevonden die HTML tables of HTML lists bevatten, moeten we nog een extra stap toepassen in het filterproces. Het is namelijk zo dat niet elke webpagina die een HTML table bevat ook effectief gestructureerde informatie bevat. HTML tables worden namelijk ook voor lay-out doeleinden gebruikt (zie Figuur 1.8). Dit geldt eveneens voor HTML lists, die soms gebruikt worden voor het visualiseren van een menu (zie Figuur 1.9). Men zou uit dergelijke HTML lists wel een database met één enkele kolom kunnen halen, maar dit is meestal niet erg interessant. Dergelijke lijsten zijn enkel interessant wanneer het gaat om waarden met eenzelfde semantiek.

We merken op dat het ook mogelijk is om een HTML table te simuleren met behulp van andere HTML tags, bijvoorbeeld met "`<div>`" (division) tags. Een voorbeeld hiervan is te zien in Figuur 1.10. De CSS code die bij dit voorbeeld hoort wordt weergegeven in Figuur 1.11. Ondanks het feit dat deze mogelijkheid bestaat, wordt dit niet als de standaardmethode beschouwd om een tabel aan te maken. Wanneer we naar de broncode kijken zien we ook dat er redelijk wat HTML en CSS code vereist is om de "`<div>`" tags op een tabel te laten lijken. De standaardmethode om een tabel te creëren gebeurt dan ook door gebruik te maken van de "`<table>`" tag. Omgekeerd dient de "`<table>`" tag niet om de webpagina van lay-out te voorzien. Hiervoor wordt de "`<div>`"



FIGUUR 1.8: EEN HTML TABLE DIE GEBRUIKT WORDT VOOR DE LAY-OUT VAN DE WEBPAGINA.

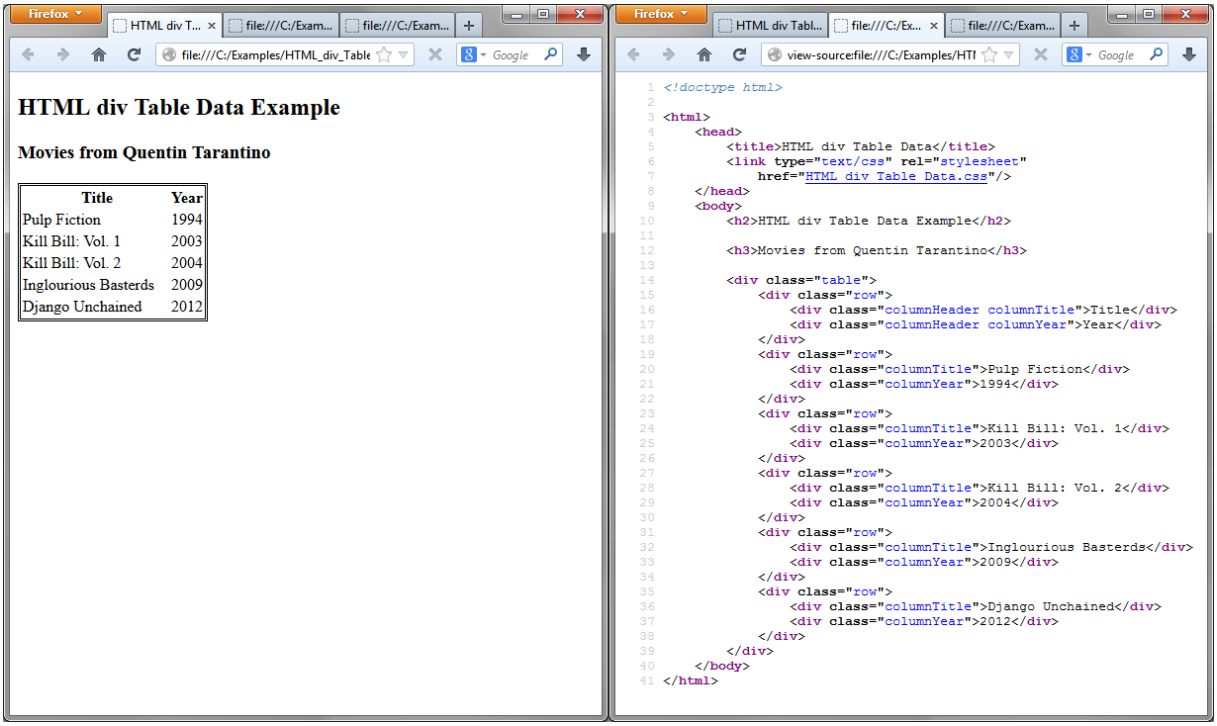


FIGUUR 1.9: EEN HTML LIST DIE GEBRUIKT WORDT VOOR EEN MENU.

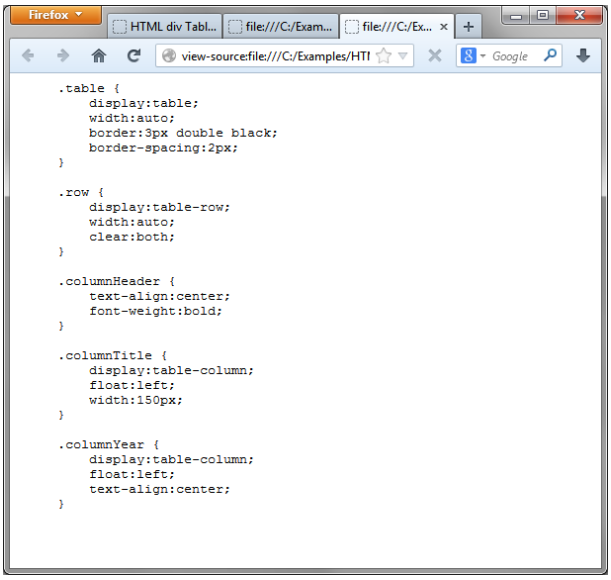
tag als de standaardmethode beschouwd, indien er geen semantische tag voor het gewenste doeleinde bestaat. Figuur 1.12 geeft de algemeen aanvaarde versie van Figuur 1.8 weer. Het feit dat de "`<table>`" tag en "`<div>`" tag als de standaardmethoden worden beschouwd voor respectievelijk tabellen en lay-out, impliceert echter niet dat iedereen deze methoden correct gebruikt. Dit heeft als gevolg dat het gebruik van de extra stap in het filterproces noodzakelijk is.

STAP 2: EXTRACTIE VAN INFORMATIE

Nu we enkel webpagina's overhouden die effectief gestructureerde informatie bevatten, kunnen we deze informatie uit de webpagina's gaan halen en in een database toevoegen. Dit proces is het eenvoudigst uit te voeren bij HTML tables. Deze data is immers

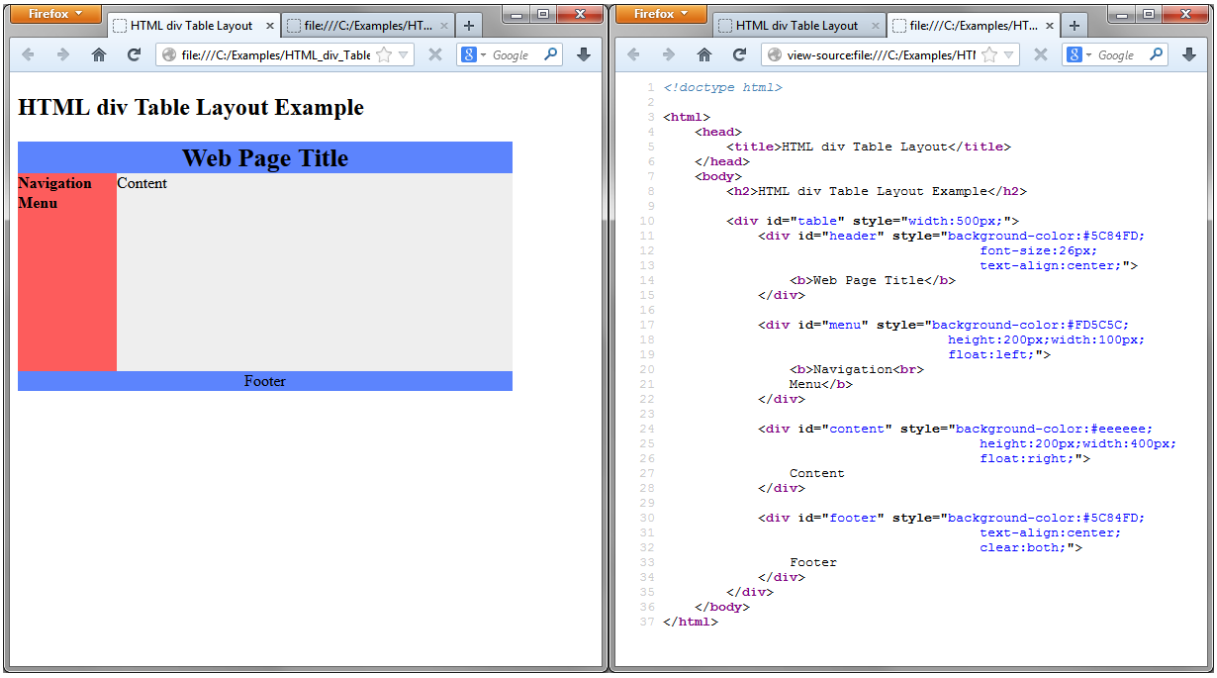


FIGUUR 1.10: EEN HTML TABLE GESIMULEERD MET BEHULP VAN "<DIV>" TAGS.



FIGUUR 1.11: DE CSS CODE VAN DE HTML TABLE IN FIGUUR 1.10.

voldoende gestructureerd om rechtstreeks uit een webpagina te extraheren. HTML lists zijn minder triviaal om te ontleden, vanwege de attribuutscheidingen die we moeten vinden. Deze attribuutscheidingen worden ook wel scheidingstekens genoemd. Attributen kunnen gescheiden worden door middel van opmaak, bijvoorbeeld een puntkomma, of door middel van geneste lijsten. Geneste lijsten worden echter zelden gebruikt voor relationele informatie, maar worden eerder aangewend voor hiërarchisch gestructureerde informatie, omdat er anders maar één bullet point aanwezig is per attribuut. Zo kan



FIGUUR 1.12: EEN WEBPAGINA WAARBIJ "<DIV>" TAGS GEBRUIKT WORDEN VOOR DE LAY-OUT.

bijvoorbeeld een persoon maar één telefoonnummer hebben voor het telefoonnummer attribuut bij relationele informatie, terwijl een persoon meerdere telefoonnummers kan hebben voor het telefoonnummer attribuut bij hiërarchische informatie. Merk op dat onder scheidingstekens ook tags worden verstaan, die bijvoorbeeld opmaak voorzien. Het herkennen van geformatteerde opmaak is zeer moeilijk, zoals we in stap één van dit extractieproces reeds hebben aangehaald. Attributen worden bij geformatteerde opmaak echter ook gescheiden door middel van scheidingstekens. We kunnen ons voor geformatteerde opmaak bijgevolg baseren op technieken die ook bij HTML lists gebruikt worden.

STAP 3: TOEVOEGING VAN SEMANTIEK

We hebben reeds de inhoud van onze database bemachtigd, en kunnen nu starten met de volgende stap in ons proces. Deze stap houdt in dat we onze database gaan voorzien van semantiek. Onder semantiek verstaan we database schema's, die onder andere kolomnamen, kolomtypes, edm. aanduiden. We kunnen onder andere proberen deze semantiek af te leiden van de data zelf. Dit gaat echter niet vanzelfsprekend, aangezien er niet altijd een tabelhoofding aanwezig is op de webpagina (dit is het geval voor de webpagina's in Figuur 1.4, Figuur 1.5 en Figuur 1.6). Andere mogelijkheden voor het vinden van semantiek zijn het kijken naar de omliggende tekst op de originele webpagina of het kijken naar reeds van semantiek voorziene databases die gelijkaardige informatie bevatten. [6] [7]

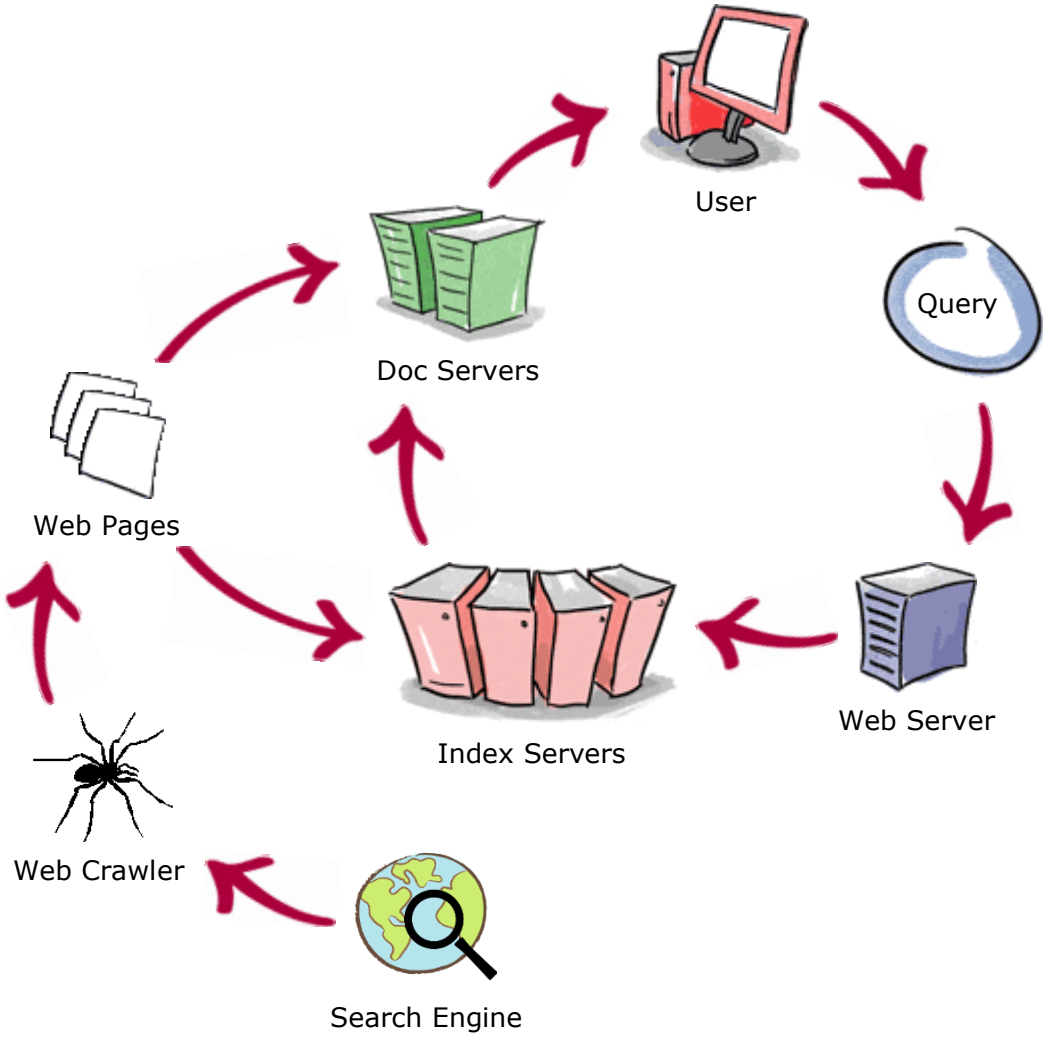
Kolomtypes duiden aan van welk type de waarden in een kolom zijn. Enkele voorbeelden van deze types zijn: integer, string, enum, set, list, array, etc. De moeilijkheidsgraad van het vinden van kolomtypes hangt af van de waarden die in een kolom voorkomen. Als een kolom enkel natuurlijke getallen bevat, dan is het kolomtype integer. Een minder triviale situatie ontstaat wanneer een kolom van honderd waarden enkel bestaat uit natuurlijke getallen, met uitzondering van tien strings. De moeilijkheid ligt hier in het feit dat we niet weten of de strings correcte waarden zijn voor deze kolom. Indien de strings geen correcte waarden blijken te zijn, dan is het kolomtype integer, maar als de strings wel correcte waarden zijn, dan is het kolomtype string. Om te achterhalen of de stringwaarden correct zijn, kunnen we gaan kijken naar de overige semantiek die al aan de database is toegekend. Stel dat de kolomnaam bijvoorbeeld "leeftijd" is, dan weten we dat de strings niet correct zijn, tenzij de strings voluit geschreven leeftijden zijn.

De reden waarom we onze database van semantiek willen voorzien, vloeit voort uit onze uitleg over kolomnamen en kolomtypes. De semantiek geeft namelijk een completer beeld van de informatie die zich in de database bevindt. De kolomnaam "leeftijd" duidt bijvoorbeeld aan wat de natuurlijke getallen in een kolom betekenen. Kolomtypes aan de andere kant helpen met het valideren van de waarden die in een kolom voorkomen. Semantiek beschrijft met andere woorden de informatie die in een database voorkomt en geeft ons de mogelijkheid om te controleren of de kolomwaarden toegestaan zijn in de kolom waarin ze zich bevinden.

Stap 4: Werken met de data

In de vierde en laatste stap gaan we aan de slag gaan met de verzamelde data. We kunnen deze nieuw gevormde databases bijvoorbeeld op verschillende traditionele manieren gebruiken, zoals databases queryen voor informatie, tabellen met elkaar joinen om een dieper inzicht in de data te verkrijgen, etc. Wanneer we over grote hoeveelheden gestructureerde informatie beschikken, kunnen we deze informatie ook gebruiken voor de ontwikkeling van nieuwe applicaties zoals *schema auto-complete* en *synonym discovery*. [8] Schema auto-complete helpt database designers bij het maken van een database schema. Wanneer een gebruiker een attribuut opgeeft, dan geeft de applicatie suggesties van gerelateerde attributen. Attribute synonym discovery zoekt attributen die synoniemen zijn van elkaar. De synoniemen die hieruit voortkomen zijn vollediger dan een thesaurus, aangezien ook niet natuurlijke taalstrings zoals "tel-#" aanwezig zijn tussen de attributen.

Een andere mogelijkheid bestaat erin de verkregen gestructureerde data te gebruiken om het web op een verfijndere manier te queryen, door gestructureerde data tussen de zoekresultaten te tonen. Hiervoor hebben we echter nieuwe zoekmethoden nodig. De standaard aanpak bij zoeksystemen [9] werkt namelijk niet voor databases. Om dit te



FIGUUR 1.13: STANDAARD AANPAK BIJ ZOEKSYSTEMEN.

verduidelijken maken we gebruik van een simplistische vorm van deze standaard aanpak, weergegeven in Figuur 1.13. De figuur bestaat uit twee grote delen: de gebruiker die een query stuurt naar een server en een zoekmachine die een web crawl uitvoert.

We beginnen met de zoekmachine die een web crawl uitvoert. Een zoekmachine, zoals Google, stuurt constant web crawlers uit. Deze web crawlers gaan op zoek naar zowel nieuwe als reeds bestaande webpagina's. Voor elke nieuwe webpagina wordt er een index gecreëerd op basis van de woorden die op een webpagina voorkomen. Zowel het aantal voorkomens als de posities van deze voorkomens worden per woord in rekening gebracht. De indexen worden opgeslagen op de index servers en de webpagina's worden opgeslagen op de document servers. Voor reeds bestande webpagina's worden de entries in de index en document servers geüpdatet.

Wanneer een gebruiker een query stuurt naar een zoekmachine, bijvoorbeeld "Films Quentin Tarantino", dan komt deze query aan bij de webserver van de zoekmachine.

FIGUUR 1.14: WIKIPEDIA PAGINA DIE EEN LIJST VAN LANDEN EN HUN BNP WEERGEEFT.

Deze webserver stuurt de query door naar de index servers die vervolgens gaan opzoeken welke webpagina's relevant zijn in verband met deze zoekopdracht. De relevantie van een webpagina wordt onder andere bepaald aan de hand van het aantal voorkomens van de ingevoerde zoektermen op de geïndexeerde webpagina's. Het resultaat van de index servers wordt door de document servers gebruikt om een presentatie te creëren die vervolgens wordt doorgestuurd naar de gebruiker. De presentatie omvat een link naar elke relevante webpagina samen met het stukje tekst waarin de zoekwoorden voorkomen.

Nu we de aanpak van traditionele zoeksystemen hebben besproken, kunnen we uitleggen waarom dit niet optimaal werkt voor databases. Beschouw bijvoorbeeld de Wikipedia webpagina in Figuur 1.14[2]. Deze webpagina bevat een tabel van alle landen samen met

[2] http://nl.wikipedia.org/wiki/Lijst_van_landen_naar_bnp opgeroepen op 16 februari 2014.

hun bruto nationaal product (bnp). Deze tabel bevat 183 rijen, inclusief tabelhoofding. De afkorting "bnp" komt echter maar 4 keer voor op de webpagina, net zoals de woordvolgorde "bruto nationaal product". Dit is een totaal van 8 keer dat er naar "bruto nationaal product" verwezen wordt. Het is niet moeilijk om voor te stellen dat er webpagina's bestaan die vaker verwijzen naar "bruto nationaal product", zonder dat zij een lijst van het bnp van elk land weergeven. Deze webpagina's zouden dan ook hoger scoren dan de Wikipedia pagina. Dit terwijl de Wikipedia webpagina in feite 182 keer een "bruto nationaal product" vermeld. Een zoeksysteem waarbij de semantiek van tabellen in rekening wordt gebracht, zoals het voorkomen van het gezochte woord in de tabelhoofding, zou bijgevolg betere zoekresultaten opleveren.

1.2 VERLOOP MASTERPROEF

In deze masterproef gaan we dieper in op stap 2 van het informatie extractieproces, namelijk het extraheren van gestructureerde informatie uit webpagina's. Meer bepaald focussen we ons op relationele informatie die verborgen zit in geformatteerde opmaak. We leggen ons toe op relationele informatie omdat dit de meest voorkomende vorm van gestructureerde informatie is. Onze toelegging op geformatteerde opmaak volgt uit een opmerking die we reeds in de inleiding van dit hoofdstuk hebben gemaakt, namelijk dat HTML tables en HTML lists specifieke varianten zijn van HTML formatting, en dat HTML formatting op zijn beurt een specifiekere variant is van tekst formatting. Indien we bijgevolg een oplossing voor tekst formatting kunnen vinden, dan hebben we eveneens een oplossing voor HTML tables, HTML lists en HTML formatting.

Omdat we in deze masterproef focussen op relationele informatie, hebben we ook enkel te maken met relationele databases. Elke toekomstige verwijzing naar een database is bijgevolg een verwijzing naar een relationele database, tenzij anders aangegeven. In een relationele database wordt de (gestructureerde) data bewaard in één of meerdere tabellen. Meer informatie over het relationeel database model kan men terugvinden in Hoofdstuk 2.

Handmatig de scheidingstekens instellen voor elke webpagina zou onbegonnen werk zijn vanwege de immens grote hoeveelheid webpagina's op het internet. We bestuderen in deze masterproef dan ook een techniek, *wrapper inductie* genaamd, die automatisch deze scheidingstekens kan vinden. In Hoofdstuk 3 wordt de theorie achter deze techniek uitgelegd.

In hoofdstuk 4 tot en met 7 worden reeds bestaande algoritmes besproken die op deze wrapper inductietechniek steunen. Vervolgens bespreken we in Hoofdstuk 8 een eigen variant op het algoritme van Hoofdstuk 6. Deze eigen variant kan tot op zekere hoogte

overweg met ontbrekende attributen en met inconsistente attribuutvolgorden. Hoofdstuk 9 beschrijft twee verbeteringen die ervoor zorgen dat de algoritmes van hoofdstuk 4 tot en met 8 efficiënter en vlotter werken.

We hebben alle besproken wrapper classes en verbeteringen ook uitgebreid getest. De opstelling, resultaten en conclusies van deze experimenten zijn terug te vinden in Hoofdstuk 10.

Tot slot eindigen we met een algemene conclusie voor deze masterproef (Hoofdstuk 11) en enkele gerelateerde onderzoeken naar het informatie extractieproces (Hoofdstuk 12).

2. Het Relationeel Model

Een database bestaat uit tabellen. Een webpagina kan op zijn beurt ook tabellen bevatten onder de vorm van HTML table tags. Om verwarring te vermijden wordt er naar tabellen op webpagina's steeds gerefereerd via de term *HTML tables.* In het geval van database tabellen maken we gebruik van de terminologie van het relationele model [10], die de term *relatie* gebruikt voor een tabel. Deze terminologie wordt in dit hoofdstuk kort uitgelegd, aangezien we in deze masterproef focussen op relationele data.

In Illustratie 2.1 wordt er een tabel over films weergegeven die geproduceerd zijn door Quentin Tarantino. In het relationeel model wordt naar een tabel verwezen door middel van de term relatie. De tabel in Illustratie 2.1 noemen we dan ook de Movies relatie. De kolomnamen van een relatie noemen we attributen en een bepaalde kolom bevat de waarden van een bepaald attribuut. Een attribuut beschrijft welke waarden er in een kolom kunnen voorkomen. De Movies relatie heeft twee attributen, namelijk "title" en "year". De string "Pulp Fiction" noemen we de attribuutwaarde van het attribuut "title" voor het eerste tupel.

title	year
Pulp Fiction	1994
Kill Bill: Vol. 1	2003
Kill Bill: Vol. 2	2004
Inglourious Basterds	2009
Django Unchained	2012

Illustratie 2.1: De Movies relatie.

De naam van een relatie, samen met de attributen van die relatie, noemen we het schema van een relatie. Zoals we al eerder hebben aangehaald kan een database uit meerdere tabellen, oftewel relaties, bestaan. De verzameling van de schema's van de relaties in een database wordt een database schema genoemd. Het schema van de Movies relatie is als volgt weergegeven:

Movies(title, year)

De rijen van een relatie worden tupels genoemd. Een tupel kan een waarde voor elk attribuut van de relatie hebben. Merk op dat de header rij geen tupel is in de relatie van Illustratie 2.1. Deze rij maakt deel uit van het schema en duidt enkel de attribuutnamen van de kolommen aan.

Het relationeel model vereist dat de kolomtypes elementaire types zijn, met andere woorden elke attribuutwaarde moet atomair zijn. Tot de elementaire types behoren onder andere integer, string, enum, etc. De types set, list en array zijn types die een record structuur hebben. Deze types zijn dan ook geen elementaire types. Een kolomtype wordt ook wel het domein van het attribuut genoemd. Attribuutdomeinen kunnen opgenomen worden in het database schema. Het schema van de Movies relatie wordt dan als volgt weergegeven:

Movies(title:string, year:integer)

Illustratie 2.1 toonde de standaard notatie voor een relatie in het relationeel model. Dit is echter niet de notatie die wij doorheen deze masterproef gaan gebruiken. De notatie die wij zullen gebruiken is weergegeven in Illustratie 2.2. Deze notatie sluit meer aan bij de achterliggende structuur die de algoritmes, die in deze masterproef aan bod komen, gebruiken voor het bewaren van een relatie.

$$R_{\text{Movies}}(\text{title, year}) = \left\{ \begin{array}{c} \langle \text{'Pulp Fiction'},1994\rangle, \\ \langle \text{'Kill Bill: Vol. 1'},2003\rangle, \\ \langle \text{'Kill Bill: Vol. 2'},2004\rangle, \\ \langle \text{'Inglourious Basterds'},2009\rangle, \\ \langle \text{'Django Unchained'},2012\rangle \end{array} \right\}$$

ILLUSTRATIE 2.2: ALTERNATIEVE NOTATIE VOOR DE MOVIES RELATIE.

3. WRAPPER INDUCTIE

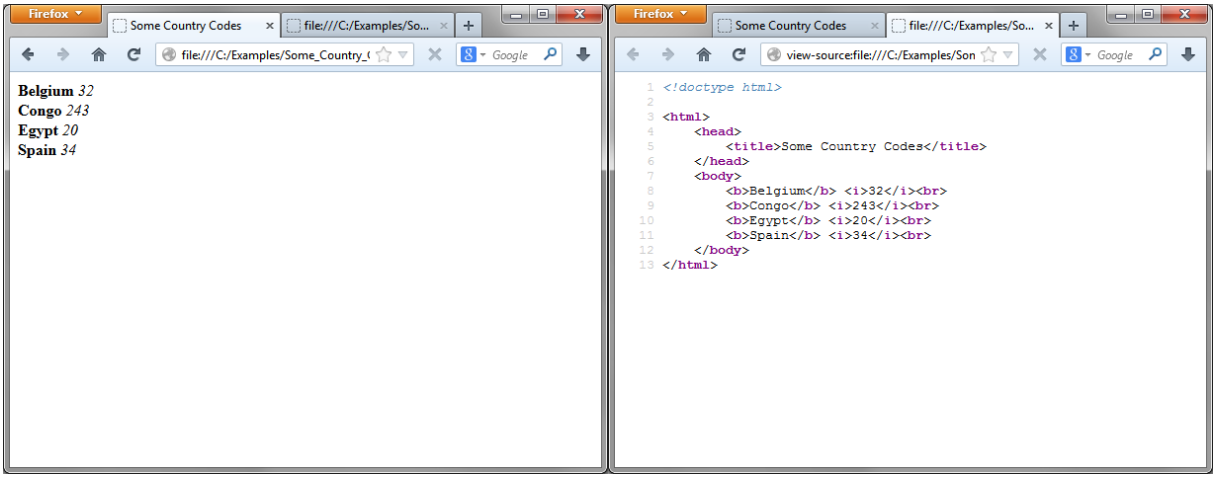
In Hoofdstuk 1 hebben we gezien dat verschillende webpagina's gestructureerde informatie bevatten, die verborgen zit tussen de HTML code. Deze informatie manueel uit de individuele webpagina's halen is onbegonnen werk. De hoeveelheid webpagina's op het web is namelijk immens groot en er komen elke dag nieuwe webpagina's bij. Om deze taak (gestructureerde informatie uit webpagina's halen) te vergemakkelijken, zijn er systemen ontwikkeld die gebruik maken van wrappers. Een wrapper is een procedure die ontwikkeld is om informatie uit een specifieke soort webpagina's te halen. Dit wil zeggen dat er voor elke soort webpagina's een nieuwe wrapper geschreven moet worden. Ondanks het feit dat dit al een verbetering is ten opzichte van het handmatig verzamelen van de informatie op een webpagina, blijft dit toch een moeizaam proces, dat gevoelig is voor fouten. Omwille van deze redenen bekijken we in dit hoofdstuk hoe we de creatie van wrapper procedures kunnen automatiseren. Deze techniek noemen we wrapper inductie.

Voor het schrijven van dit hoofdstuk is gebruik gemaakt van de paper "Wrapper Induction: Efficiency and expressiveness" van Kushmerick [11], alsook de paper "Wrapper Induction for Information Extraction" van Kushmerick et al. [12] en de doctoraatsthesis "Wrapper Induction for Information Extraction" van Kushmerick [13]. De basis van het country-code voorbeeld, dat doorheen deze masterproef gebruikt zal worden, is gebaseerd op het country-code voorbeeld dat doorheen de paper "Wrapper Induction: Efficiency and expressiveness" van Kushmerick [11] gebruikt wordt. Onze toevoegingen ten opzichte van deze bronnen zijn het uitgebreider en gedetailleerder beschrijven van het formele informatie extractieproces.

3.1 WRAPPERS

Om een beter inzicht te krijgen in de werking van wrappers, starten we deze sectie met een korte illustratie. In Figuur 3.1 zien we een fictieve internetsite die de telefooncodes van vier landen weergeeft. De broncode van deze website wordt aan de rechterkant in de figuur weergegeven. We gaan deze HTML code echter vereenvoudigen tot degene die is weergegeven in Illustratie 3.1. Dit heeft als doel de explicatie van de algoritmes die later aan bod komen te vereenvoudigen, daar de broncode niet langer over tabs beschikt en bij elkaar horende elementen voortaan op dezelfde regel staan.

De relatie (informatie) die we uit de webpagina willen halen is weergegeven in Illustratie 3.2. We kunnen dit doen met behulp van de specifieke wrapper procedure *ccwrap* (country-code wrapper) die in Illustratie 3.3 beschreven is. Deze procedure bevat twee



FIGUUR 3.1: WEBPAGINA DIE DE TELEFOONCODES VOOR EEN AANTAL LANDEN WEERGEEFT.

```
1: <html><head><title>Some Country Codes</title></head><body>
2: <b>Belgium</b> <i>32</i><br>
3: <b>Congo</b> <i>243</i><br>
4: <b>Egypt</b> <i>20</i><br>
5: <b>Spain</b> <i>34</i><br>
6: </body></html>
```

ILLUSTRATIE 3.1: VEREENVOUDIGDE BRONCODE VAN DE WEBPAGINA IN FIGUUR 3.1.

$$R_{\text{cc}} = \begin{Bmatrix} \langle\text{'Belgium','32'}\rangle \\ \langle\text{'Congo','243'}\rangle \\ \langle\text{'Egypt','20'}\rangle \\ \langle\text{'Spain','34'}\rangle \end{Bmatrix}$$

ILLUSTRATIE 3.2: DE RELATIE DIE WE UIT DE WEBPAGINA VAN FIGUUR 3.1 WILLEN HALEN.

```
1: procedure ccwrap(page P)
2:   while search('<b>',P) ≠ EOF
3:     for each ⟨l_k,r_k⟩ ∈ [⟨'<b>','</b>'⟩,⟨'<i>','</i>'⟩]
4:       search(l_k,P) + |l_k| indicates the beginning of the k^th attribute
           value, we also move to this position
5:       search_move(r_k,P)indicates the end of the k^th attribute value
6:   return relation {⟨country_1, code_1⟩,…,⟨country_n, code_n⟩}
```

ILLUSTRATIE 3.3: PROCEDURE CCWRAP, DIE DE RELATIE UIT DE WEBPAGINA VAN FIGUUR 3.1 HAALT.

herhalingslussen, die terug te vinden zijn op regel 2 en 3. De eerste lus gaat kijken of er nog nieuwe ⟨country, code⟩ *tupels* aanwezig zijn op de webpagina. Dit doet de lus door te zoeken naar "" tags. Deze duiden immers nieuwe tupels aan. De tweede lus gaat de eerst voorkomende waarde van elk attribuut uit de webpagina halen. Dit gebeurt volgens de volgorde van de attributen (zie lijst op regel 3). In ons country-code voorbeeld zijn de attributen het land en de telefooncode. Het land attribuut wordt omvat door de tags "" en "". Het telefooncode attribuut wordt omvat door de tags "" en "".

De hulpfuncties die in de ccwrap procedure van Illustratie 3.3 vermeld worden, maar niet beschreven zijn, zijn terug te vinden in Bijlage A.

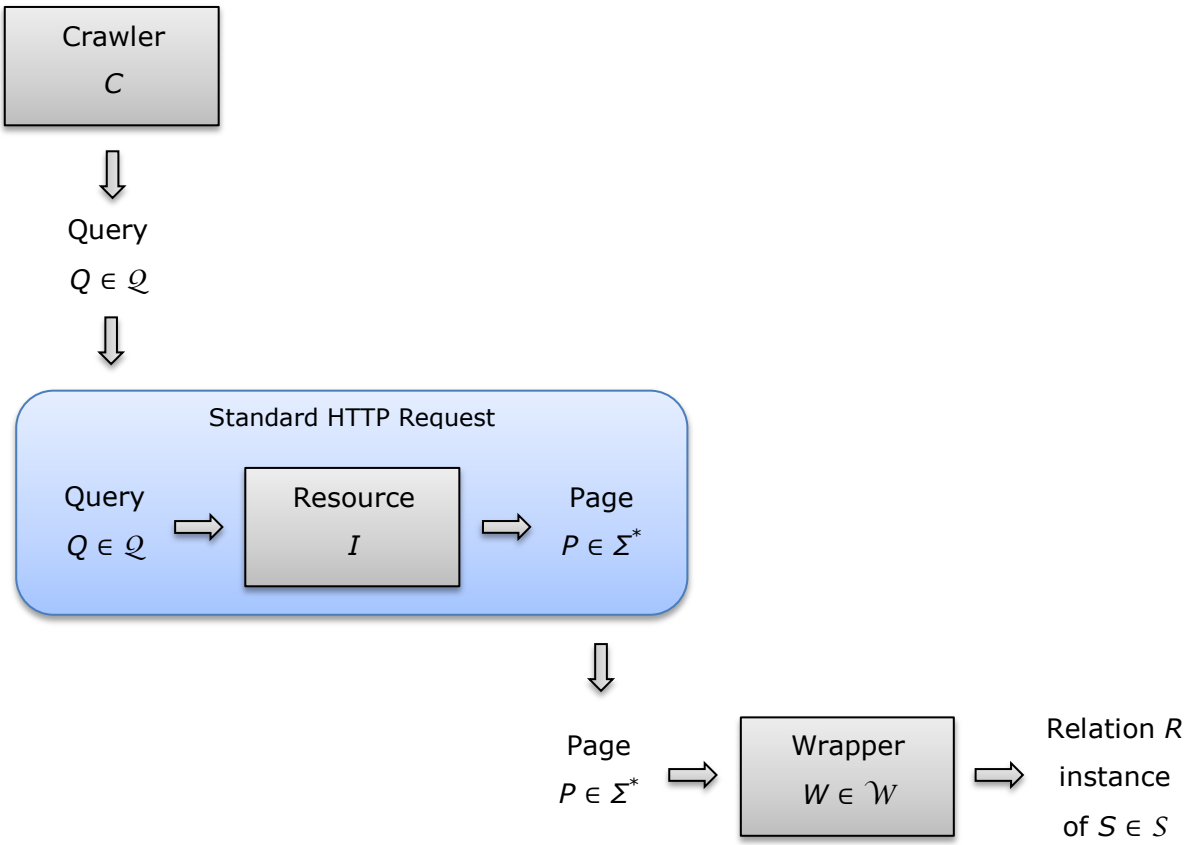
De procedure in Illustratie 3.3 werkt omdat de webpagina gebruik maakt van uniforme opmaakconventies. Met een uniforme opmaak bedoelen we dat alle attribuutwaarden van eenzelfde attribuut altijd omgeven zijn door dezelfde tekens. Zo wordt elk land altijd vetgedrukt weergegeven, terwijl elke telefooncode altijd cursief geschreven is. Daarnaast houdt uniforme opmaak ook in dat er in elk tupel scheidingstekens aanwezig zijn voor elk attribuut. In Hoofdstuk 8, Sectie 8.1, wordt besproken hoe een wrapper class reageert op een pagina met een non-uniforme opmaak. De procedure ccwrap zoekt bijgevolg in de webpagina naar de volgende strings: "", "", "" en "". Deze strings worden respectievelijk aangeduid als l_1, r_1, l_2 en r_2. De notatie l_k duidt de linkerkant van attribuut k aan, terwijl r_k de rechterkant van attribuut k aanduidt. We noemen deze strings scheidingstekens. Ze bakenen namelijk de attribuutwaarde af.

We hebben reeds aangehaald dat het schrijven van wrappers een moeizaam proces is, dat gevoelig is voor fouten. Daarnaast schaalt dit proces niet. Het is namelijk onbegonnen werk om een wrapper voor elke soort webpagina's te ontwikkelen. Dit proces wordt bovendien nog bemoeilijkt doordat bepaalde websites soms hun opmaakstijl veranderen. Wanneer dit gebeurd is de kans reëel dat de wrapper niet langer werkt.

3.2 HET INFORMATIE EXTRACTIEPROCES

Het automatisch leren van wrappers wordt het wrapper inductieprobleem genoemd. [11] Voor we een beschrijving kunnen geven van dit probleem en de procedures die automatisch wrappers kunnen leren, moeten we eerst wat terminologie overlopen. Het wrapper inductieprobleem is een onderdeel van het informatie extractieproces. Dit proces werd reeds informeel beschreven in Sectie 1.1. In deze sectie worden alle onderdelen van dit proces formeel beschreven, alsook de rol die het wrapper inductieprobleem hierin speelt. Het formele schema van dit proces wordt weergegeven in Figuur 3.2.

Het informatie extractieproces begint met een crawler C die een query Q opstelt. Deze query Q is geschreven in een query taal $\mathcal{Q}$ en beschrijft welke informatie we willen verkrijgen van een resource I. De query taal $\mathcal{Q}$ is in ons geval gebaseerd op URL's, aangezien we webpagina's gaan opvragen aan de hand van hun URL's. De informatie resource I komt in dit geval overeen met een webserver. Het antwoord van resource I op query Q, is een resultaatpagina P. Deze pagina is een string over een alfabet Σ, waarbij Σ meestal de Unicode tekenset is, of eventueel een subset hiervan. In Figuur 3.1 was de resultaatpagina een HTML document, maar dit is geen vereiste voor de wrappers die we in latere hoofdstukken nog gaan tegenkomen. De resultaatpagina's mogen evenzeer in



FIGUUR 3.2: FORMEEL SCHEMA VAN HET INFORMATIE EXTRACTIEPROCES.

XML, natuurlijke taal of nog een ander formaat geschreven zijn. Formeel stellen we dat de informatie resource I een functie is die een query Q omvormt tot een resultaatpagina P. De query Q, de informatie resource I en de resultaatpagina P vormen samen de onderdelen van een standaard HTTP request. Een dergelijke request treedt op wanneer een gebruiker bijvoorbeeld een webpagina opvraagt in zijn webbrowser.

Zoals we al eerder hebben aangehaald zijn we op zoek naar relationele data die ingebed zit in webpagina's. Deze kan door een webserver uit een database zijn gehaald, waarbij de data dynamisch is ingevuld in de webpagina of deze data kan hardcoded aanwezig zijn in de webpagina zelf. Beide methoden leiden tot eenzelfde resultaat, namelijk relationele data (relaties) ingebed in de webpagina P die we als resultaat terugkrijgen. Als een relatie van een pagina P bestaat uit K attributen, dan is een tupel een vector van K strings: $\langle A_1,\ldots,A_k,\ldots,A_K\rangle$, waarbij $A_k \in \Sigma^*$ voor elke $1 \leq k \leq K$. Het symbool Σ^* duidt alle string aan die we kunnen maken over het alfabet Σ. De string A_k is de attribuutwaarde van het tupel voor het k^{de} attribuut. In deze masterproef halen we telkens maar één relatie uit een webpagina. Indien er meerdere relaties op één webpagina aanwezig zijn, dan kan elke relatie uit de webpagina worden gehaald door de besproken methoden en algoritmes meerdere keren uit te voeren.

De relatie die we uit de webpagina willen halen noemen we *R*. Figuur 3.2 geeft aan dat deze relatie *R* een instantie is van een schema *S*. Elke relatie moet namelijk voldoen aan een relationeel schema. Dit komt overeen met onze beschrijving van het relationeel model (zie Hoofdstuk 2). De verzameling van alle mogelijke schema's beschrijft welke schema's er allemaal kunnen bestaan en wordt voorgesteld door de letter $\mathcal{S}$. Elk relationeel schema *S* is bijgevolg een element van $\mathcal{S}$.

Verder zien we in Figuur 3.2 dat een wrapper *W* een functie is die een relatie *R* uit een pagina *P* haalt. Dit wordt formeel voorgesteld als $W(P) = R$. In Sectie 3.1 hebben we gezien dat het belangrijkste onderdeel van een wrapper procedure de scheidingstekens zijn. Zo maakte de ccwrap procedure van Illustratie 3.3 gebruik van linker en rechter scheidingstekens voor het afbakenen van attribuutwaarden. Naast linker en rechter scheidingstekens bestaan er ook nog verschillende andere scheidingstekens waarvan een wrapper gebruik kan maken. We kunnen de wrappers die gebruik maken van dezelfde scheidingstekens groeperen in een zogenaamde wrapper class. Een wrapper class $\mathcal{W}$ is bijgevolg een verzameling van wrappers die een gemeenschappelijke eigenschap hebben, namelijk de scheidingstekens die ze elk gebruiken. De ccwrap procedure van Illustratie 3.3 behoort tot de LR wrapper class, net zoals alle andere wrappers die gebruik maken van linker en rechter scheidingstekens.

In deze masterproef komen volgende wrapper classes aan bod: LR, HLRT, OCLR en HOCLRT. Elk van deze classes is ontworpen voor het onttrekken van relationele data uit webpagina's met een bepaalde structuur. De karakteristieken van de wrapper classes worden hieronder kort beschreven.

- De LR wrapper class maakt gebruik van linker en rechter scheidingstekens voor het afbakenen van attribuutwaarden.
- De HLRT wrapper class maakt ook gebruik van linker en rechter scheidingstekens voor het afbakenen van attribuutwaarden en maakt daarbovenop gebruik van een head en tail scheidingsteken, die respectievelijk het einde van het hoofd en het begin van de voet van een webpagina aanduiden.
- De OCLR wrapper class maakt ook gebruik van linker en rechter scheidingstekens voor het afbakenen van attribuutwaarden en gaat ook het begin en einde van elk tupel afbakenen door middel van een open en close scheidingsteken.
- De HOCLRT wrapper class combineert de HLRT en OCLR wrapper class. Deze wrapper class maakt met andere woorden gebruik van een head, tail, open en close scheidingsteken, naast de linker en rechter scheidingstekens.

Daarnaast bekijken we ook nog de UOCLR wrapper class, een uitbreiding van de OCLR wrapper class, die met ontbrekende attributen en een inconsistente volgorde van attributen om kan gaan.

Nu alle elementen van het informatie extractieproces besproken zijn, kunnen we het wrapper inductieprobleem gaan beschrijven. Dit probleem luidt als volgt: we willen een wrapper W leren voor een informatie resource I, waarbij de wrapper W behoort tot een wrapper class $\mathcal{W}$. Het algoritme dat de wrappers gaat leren, noemen we het learn algoritme en elke wrapper class heeft hiervan zijn eigen variant. Een learn algoritme zal voor het eerst aan bod komen in Hoofdstuk 4. Formeel kunnen we het wrapper inductieprobleem als volgt definiëren.

Definitie 3.1: Het wrapper inductieprobleem

> Gegeven een verzameling $\mathcal{E} = \{\langle P_1,R_1\rangle,\ldots,\langle P_n,R_n\rangle\}$, waarbij elke P_i en R_i respectievelijk een pagina en de bijhorende relatie voorstellen. Het wrapper inductieprobleem bestaat er dan uit om een wrapper $W \in \mathcal{W}$ te vinden, zodat $W(P_i) = R_i$ voor elke $\langle P_i,R_i\rangle \in \mathcal{E}$.

3.3 Representatie van de relaties

In Illustratie 3.2 wordt de relatie van de webpagina van Figuur 3.1 weergegeven aan de hand van strings. De wrapper procedures gaan echter niet met strings werken, maar maken gebruik van index paren. Dit is een eenvoudige en beknopte representatie van de informatie, in tegenstelling tot strings die van een willekeurige lengte kunnen zijn. De index paren helpen ons ook een onderscheid te maken tussen een string, bijvoorbeeld "Egypt", die zowel in de relatie als in de tekst omheen de relatie kan voorkomen. De index paren duiden namelijk de exacte positie van de string aan, zodat er geen twijfel mogelijk kan zijn. Elke attribuutwaarde krijgt bijgevolg een set bestaande uit twee indices, die respectievelijk de beginpositie en eindpositie van de string in de pagina aanduiden. In Illustratie 3.4 is de relatie van Figuur 3.1 beschreven aan de hand van index paren. Als voorbeeld beschouwen we het paar ⟨112,117⟩ dat de string "Egypt" aanduidt (117 niet inclusief). We noemen deze indexweergave van de informatie, net zoals de standaardweergave van Illustratie 3.2, de relatie R van een pagina. We zien in Illustratie 3.4 dat deze relatie bestaat uit vier tupels, waarbij elk tupel twee attribuutwaarden bevat.

We kunnen de relatie R van een pagina P algemeen beschrijven aan de hand van de verzameling weergegeven in Illustratie 3.5. Een pagina P bevat $|R| > 0$ tupels, die op hun beurt elk $K > 0$ attributen bevatten. Elk paar $\langle b_{m,k},e_{m,k}\rangle$, met $1 \le k \le K$ en $1 \le m \le |R|$, beschrijft één attribuutwaarde, waarbij $b_{m,k}$ de index is die de beginpositie aanduidt van

het k^{de} attribuut in het m^{de} tupel en $e_{m,k}$ de index is die de eindpositie aanduidt van het k^{de} attribuut in het m^{de} tupel.

$$R_{cc} = \left\{ \begin{array}{c} \langle\langle 63,70\rangle,\langle 78,80\rangle\rangle, \\ \langle\langle 88,93\rangle,\langle 101,104\rangle\rangle, \\ \langle\langle 112,117\rangle,\langle 125,127\rangle\rangle, \\ \langle\langle 135,140\rangle,\langle 148,150\rangle\rangle \end{array} \right\}$$

ILLUSTRATIE 3.4: RELATIE VAN DE WEBPAGINA VAN FIGUUR 3.1, BESCHREVEN AAN DE HAND VAN INDEX PAREN.

$$R = \left\{ \begin{array}{c} \langle\langle b_{1,1}, e_{1,1}\rangle, \ldots, \langle b_{1,k}, e_{1,k}\rangle, \ldots, \langle b_{1,K}, e_{1,K}\rangle\rangle, \\ \vdots \\ \langle\langle b_{m,1}, e_{m,1}\rangle, \ldots, \langle b_{m,k}, e_{m,k}\rangle, \ldots, \langle b_{m,K}, e_{m,K}\rangle\rangle, \\ \vdots \\ \langle\langle b_{|R|,1}, e_{|R|,1}\rangle, \ldots, \langle b_{|R|,k}, e_{|R|,k}\rangle, \ldots, \langle b_{|R|,K}, e_{|R|,K}\rangle\rangle \end{array} \right\}$$

ILLUSTRATIE 3.5: ALGMENE BESHCRIJVEN VAN DE RELATIE *R* VAN EEN PAGINA *P*.

3.4 STRUCTUUR VOOR DE BESCHRIJVING VAN WRAPPER CLASSES

In de volgende vijf hoofdstukken worden de verschillende wrapper classes besproken (LR, HLRT, OCLRT, HOCLRT en UOCLR) individueel besproken. In elk van deze hoofdstukken beginnen we steeds met het beschrijven van een *exec* procedure die een wrapper, van de besproken wrapper class, kan uitvoeren om zo de relatie uit een pagina te halen. Vervolgens bespreken we telkens de relatie tussen de verschillende scheidingstekens, alsook de constraints waaraan deze moeten voldoen. We eindigen elk hoofdstuk met een beschrijving van een *learn* algoritme dat automatisch een wrapper kan leren uit een verzameling van voorbeeldpagina's en hun bijhorende relaties.

4. De LR Wrapper Class

In Sectie 3.1 hebben we de procedure ccwrap besproken (Illustratie 3.3). Deze procedure maakt gebruik van linker en rechter scheidingstekens om de attribuutwaarden van de tupels te vinden. Dit principe vormt de basis van de LR (Left-Right) wrapper class. De procedure ccwrap is dan ook een wrapper die tot de LR wrapper class behoort. In dit hoofdstuk wordt een algemene procedure besproken voor het uitvoeren van LR wrappers, genaamd exec_{LR}. Daarnaast wordt ook besproken hoe deze wrappers automatisch uit een verzameling van voorbeeldpagina's geleerd kunnen worden door middel van het learn_{LR} algoritme.

Voor het schrijven van dit hoofdstuk is gebruik gemaakt van de paper "Wrapper Induction: Efficiency and expressiveness" van Kushmerick [11], alsook de paper "Wrapper Induction for Information Extraction" van Kushmerick et al. [12] en de doctoraatsthesis "Wrapper Induction for Information Extraction" van Kushmerick [13]. De basis van het country-code voorbeeld, dat doorheen deze masterproef gebruikt zal worden, is gebaseerd op het country-code voorbeeld dat doorheen de paper "Wrapper Induction: Efficiency and expressiveness" van Kushmerick [11] gebruikt wordt. Onze toevoegingen ten opzichte van deze bronnen zijn het gedetailleerder beschrijven van de wrapper class, alsook het duidelijker definiëren van de constraints van deze wrapper class.

4.1 De exec_{LR} procedure

Het is mogelijk om de ccwrap procedure van Illustratie 3.3 te veralgemenen, zodat de scheidingstekens niet langer hardcoded opgenomen worden in het algoritme. Dit leidt tot de exec_{LR} procedure van Illustratie 4.1. Deze procedure krijgt als input een wrapper die de scheidingstekens bevat. Deze verandering laat toe dat scheidingstekens willekeurige strings kunnen zijn en dat er een willekeurig aantal attributen op een pagina aanwezig

```
1:  procedure execLR(wrapper ⟨⟨l1,r1⟩,…,⟨lK,rK⟩⟩, page P)
2:    m ≔ 0
3:    while search(l1,P) ≠ EOF
4:      m ≔ m + 1
5:      for each ⟨lk,rk⟩ ∈ [⟨l1,r1⟩,…,⟨lK,rK⟩]
6:        bm,k ≔ search(lk,P) + |lk|
7:        em,k ≔ search_after_move(rk,P,bm,k)
8:    return relation {⟨⟨b1,1,e1,1⟩,…,⟨b1,K,e1,K⟩⟩, … , ⟨⟨b|R|,1,e|R|,1⟩,…,⟨b|R|,K,e|R|,K⟩⟩}
```

Illustratie 4.1: Procedure exec_{LR}, die gebruik maakt van een LR wrapper om de relatie van een pagina te verkrijgen.

mogen zijn. De variabelen $l_1,\ldots,l_K$ duiden de linker scheidingstekens aan en de variabelen $r_1,\ldots,r_K$ duiden de rechter scheidingstekens aan, voor een pagina die K attributen bevat. De procedure maakt met andere woorden gebruik van een wrapper die bestaat uit $2K$ scheidingstekens. Deze wrapper noemen we een LR wrapper en wordt genoteerd aan de hand van de vector $\langle\langle l_1,r_1\rangle,\ldots,\langle l_K,r_K\rangle\rangle$.

De hulpfuncties die in de exec_{LR} procedure van Illustratie 4.1 vermeld worden, maar niet beschreven zijn, zijn terug te vinden in Bijlage A.

Als de procedure exec_{LR} de wrapper $\langle\langle$`'<b>'`,`'</b>'`$\rangle$, $\langle$`'<i>'`,`'</i>'`$\rangle\rangle$ als input krijgt, simuleert deze de ccwrap procedure, op het output formaat na. De procedure exec_{LR} maakt namelijk gebruik van index paren voor het aanduiden van attribuutwaarden, in tegenstelling tot de procedure ccwrap die van de attribuutstrings zelf gebruikt maakt. De redenen hiervoor werden besproken in Sectie 3.3.

Aangezien een LR wrapper een vector $\langle\langle l_1,r_1\rangle,\ldots,\langle l_K,r_K\rangle\rangle$ is, kunnen we het wrapper inductieprobleem herformuleren als het vinden van $2K$ scheidingstekens, op basis van een verzameling $\mathcal{E} = \{\langle P_1,R_1\rangle,\ldots,\langle P_n,R_n\rangle\}$ van pagina's en hun bijhorende relaties, zodat $W(P_i) = R_i$ voor elke $\langle P_i,R_i\rangle \in \mathcal{E}$.

4.2 Scheidingstekens

Vooraleer we de procedure voor het leren van wrappers kunnen uitleggen, moeten we eerst dieper ingaan op de scheidingstekens. Meer bepaald gaan we kijken welke strings in aanmerking komen voor welk scheidingsteken en welke van deze kandidaten effectief goede scheidingstekens zijn.

In het algemeen kunnen we stellen dat kandidaten voor een scheidingsteken worden gegenereerd uit een string die zich tussen twee opeenvolgende attribuutwaarden bevindt, ongeacht of deze string zich tussen twee attribuutwaarden van dezelfde of van verschillende tupels bevindt. Kandidaten kunnen ook gegenereerd worden uit de string die zich tussen het begin van de pagina en de eerste attribuutwaarde van het eerste tupel bevindt, of uit de string die zich tussen de laatste attribuutwaarde van het laatste tupel en het einde van de pagina bevindt.

Doorheen deze sectie wordt verder gebruik gemaakt van het country-code voorbeeld, dat geïntroduceerd werd in Hoofdstuk 3. We gaan er ook van uit de country-code pagina (Figuur 3.1) de enige pagina is in $\mathcal{E}$. Voor de eenvoud geven we de broncode van deze pagina opnieuw weer in Illustratie 4.2.

We merken op dat de LR wrapper class, net zoals alle andere wrapper classes die later nog aan bod komen, ervan uitgaat dat de pagina's in $\mathcal{E}$ een uniforme opmaak hebben.

```
1: <html><head><title>Some Country Codes</title></head><body>
2: <b>Belgium</b> <i>32</i><br>
3: <b>Congo</b> <i>243</i><br>
4: <b>Egypt</b> <i>20</i><br>
5: <b>Spain</b> <i>34</i><br>
6: </body></html>
```

ILLUSTRATIE 4.2: VEREENVOUDIGDE BRONCODE VAN HET COUNTRY-CODE VOORBEELD.

Met een uniforme opmaak bedoelen we dat alle attribuutwaarden van eenzelfde attribuut altijd omgeven zijn door dezelfde tekens. Zo wordt elk land altijd vetgedrukt weergegeven, terwijl elke telefooncode altijd cursief geschreven is in Illustratie 4.2. Daarnaast houdt uniforme opmaak ook in dat er in elk tupel scheidingstekens aanwezig zijn voor elk attribuut. In Hoofdstuk 8, Sectie 8.1, wordt besproken hoe een wrapper class reageert op een pagina met een non-uniforme opmaak.

4.2.1 KANDIDATEN VOOR L_K

Het country-code voorbeeld bevat twee attributen, namelijk "land" en "telefooncode". We moeten bijgevolg op zoek naar kandidaten voor l_1 en l_2. We beginnen met l_2, omdat l_1 een uitzondering vormt, zoals we later zullen zien. Scheidingsteken l_2 moet de linkerkant van het telefooncode attribuut afbakenen. Als we naar Illustratie 4.2 kijken, zien we dat elke telefooncode attribuutwaarde voorafgegaan wordt door "`</b> <i>`". Omdat l_2 de linkerkant van de attribuutwaarde moet afbakenen, zijn alle suffixen van deze string geldige kandidaten voor l_2.

We hebben hierboven aangehaald dat l_1 een uitzondering vormt. Dit komt doordat wanneer we bij l_1 dezelfde redeneringswijze toepassen als bij l_2, we op een probleem stuiten. De land attribuutwaarden op regels 3, 4 en 5 worden elk voorafgegaan door de korte string "`</i><br>⇓<b>`", maar de land attribuutwaarde op regel 2 wordt voorafgegaan door de lange string "`<html><head><title>Some Country Codes</title></head><body>⇓<b>`". Het newline karakter wordt in deze strings voorgesteld door het symbool ⇓. We hebben bijgevolg twee strings waaruit we kandidaten kunnen genereren. We zouden van beide strings alle suffixen kunnen genereren als kandidaten voor l_1, maar dit zouden teveel kandidaten zijn om te controleren. Dit zijn namelijk 74 kandidaten in tegenstelling tot de 12 kandidaten van de kortste string. Dit zijn meer dan zes keer zoveel kandidaten en bovendien werken we hier nog maar met één voorbeeldpagina, namelijk de country-code voorbeeldpagina. Wanneer we met tien voorbeeldpagina's zouden werken, krijgen we met zestig keer meer kandidaten te maken, oftewel ±720 kandidaten. We kunnen dit probleem echter eenvoudig oplossen door logisch te redeneren. We willen namelijk dat het scheidingsteken l_1 voor elke country attribuutwaarde voorkomt. Indien we echter een suffix van de langste string zouden

nemen, die nog steeds langer is dan de kortste string, dan gaat deze suffix niet aan elke country attribuutwaarde vooraf. We kunnen bijgevolg verdergaan met enkel de kortste string. Hiervan genereren we net zoals bij l_2 alle mogelijke suffixen als kandidaten voor het scheidingsteken l_1.

We kunnen bovenstaande ook veralgemenen. De kandidaten van een scheidingsteken l_k, gegeven een verzameling $\mathcal{E}$, van voorbeeldpagina's samen met hun bijhorende relaties, zijn alle mogelijke suffixen van de kortste string die zich links van een attribuutwaarde van attribuut k bevindt. We noteren dit als volgt: $\text{cands}_l(k,\mathcal{E})$. Alle kandidaten voor l_1 en l_2 voor het country-code voorbeeld worden gegeven in Illustratie 4.3.

$$\text{cands}_l(1,\mathcal{E}) = \left\{ \begin{array}{l} \text{'</i>
⇓'}, \\ \text{'/i>
⇓'}, \\ \text{'i>
⇓'}, \\ \text{'>
⇓'}, \\ \text{'
⇓'}, \\ \text{'br>⇓'}, \\ \text{'r>⇓'}, \\ \text{'>⇓'}, \\ \text{'⇓'}, \\ \text{''}, \\ \text{'b>'}, \\ \text{'>'} \end{array} \right\} \qquad \text{cands}_l(2,\mathcal{E}) = \left\{ \begin{array}{l} \text{' <i>'}, \\ \text{'/b> <i>'}, \\ \text{'b> <i>'}, \\ \text{'> <i>'}, \\ \text{' <i>'}, \\ \text{'<i>'}, \\ \text{'i>'}, \\ \text{'>'} \end{array} \right\}$$

ILLUSTRATIE 4.3: KANDIDATEN VOOR DE SCHEIDINGSTEKENS L_1 EN L_2.

4.2.2 KANDIDATEN VOOR R_K

Kandidaten voor rechter scheidingstekens kunnen op een gelijkaardige manier gevonden worden als kandidaten voor linker scheidingstekens. De methode verschilt echter op een aantal punten. Zo gaan we niet meer kijken naar wat er links van de attribuutwaarden staat, maar kijken we juist naar wat er rechts van de attribuutwaarden staat. Een rechter scheidingsteken moet namelijk de rechterkant van een attribuutwaarde afbakenen. Doordat we gaan kijken naar wat er achter de attribuutwaarden staat, en omdat we willen dat de scheidingstekens zo dicht mogelijk bij de attribuutwaarden zelf staan, gaan we met prefixen werken in de plaats van suffixen. Tot slot stellen we vast dat r_K en niet r_1 de uitzondering zal vormen. De reden hiertoe zal duidelijk worden tijdens de bespreking van het country-code voorbeeld.

We beginnen met het scheidingsteken r_1, dat geen uitzondering vormt. In Illustratie 4.2 zien we dat elke country attribuutwaarde gevolgd wordt door "`</b> <i>`". Alle mogelijke prefixen van deze string zijn kandidaten voor het scheidingsteken r_1.

De reden waarom r_K, of in het voorbeeld r_2, de uitzondering vormt, komt doordat we, zoals hierboven reeds aangehaald, naar de rechterkant van de attribuutwaarden kijken. Dit is geen probleem voor r_1, omdat dit scheidingsteken altijd gevolgd wordt door l_2 en

$$\text{cands}_r(1,\mathcal{E}) = \begin{Bmatrix} \text{' <i>'}, \\ \text{' <i'}, \\ \text{' <'}, \\ \text{' '}, \\ \text{''}, \\ \text{'</b'}, \\ \text{'</'}, \\ \text{'<'} \end{Bmatrix} \qquad \text{cands}_r(2,\mathcal{E}) = \begin{Bmatrix} \text{'</i>
⇓<b>'}, \\ \text{'</i>
⇓<b'}, \\ \text{'</i>
⇓<'}, \\ \text{'</i>
⇓'}, \\ \text{'</i>
'}, \\ \text{'</i><br'}, \\ \text{'</i><b'}, \\ \text{'</i><'}, \\ \text{'</i>'}, \\ \text{'</i'}, \\ \text{'</'}, \\ \text{'<'} \end{Bmatrix}$$

ILLUSTRATIE 4.4: KANDIDATEN VOOR DE SCHEIDINGSTEKENS R_1 EN R_2.

een attribuutwaarde voor het tweede attribuut. De attribuutwaarde voor het tweede attribuut op regel 5 daarentegen wordt gevolgd door de voet van de pagina (de tekst na het laatste tupel). In het voorbeeld hebben we bijgevolg twee strings voor r_2, namelijk "</i>
⇓<b>" en "</i>
⇓</body></html>". Net zoals bij l_1 gaan we ook hier weer enkel verder met de kortste string. Alle mogelijke prefixen van de string "</i>
⇓<b>" vormen bijgevolg de kandidaten voor r_1.

We kunnen bovenstaande ook veralgemenen. De kandidaten van een scheidingsteken r_k, gegeven een verzameling $\mathcal{E}$, van voorbeeldpagina's samen met hun bijhorende relaties, zijn alle mogelijke prefixen van de kortste string die zich rechts van een attribuutwaarde van attribuut k bevindt. We noteren dit als volgt: $\text{cands}_r(k,\mathcal{E})$. Alle kandidaten voor r_1 en r_2 voor het country-code voorbeeld worden gegeven in Illustratie 4.4.

4.2.3 RELATIE TUSSEN DE SCHEIDINGSTEKENS

In Sectie 4.2.1 en 4.2.2 werd besproken hoe kandidaten voor scheidingstekens l_k en r_k gevonden kunnen worden. Deze kandidaten moeten echter nog gevalideerd worden, om te weten te komen of ze geldige scheidingstekens zijn. Geldige scheidingstekens zorgen ervoor dat de exec_{LR} procedure de correcte relatie uit een pagina kan halen. Vooraleer we de constraints kunnen opstellen, waarmee we de kandidaten kunnen valideren, moeten we de relatie tussen de $2K$ scheidingstekens bespreken. Deze scheidingstekens zijn namelijk onafhankelijk van elkaar, met andere woorden de geldigheid van een scheidingsteken is onafhankelijk van de andere scheidingstekens.

We kunnen deze onafhankelijkheid op een aantal manieren aantonen. Een eerste manier is door te kijken naar de exec_{LR} procedure in Illustratie 4.1. In deze procedure wordt telkens maar naar één scheidingsteken gezocht (regel 3, 6 en 8). Als één van deze scheidingstekens niet geldig is, dan zal de output relatie van de exec_{LR} procedure niet correct zijn.

Een tweede manier om de onafhankelijkheid aan te tonen, is door logisch te redeneren. Stel dat we een ongeldige kandidaat als scheidingsteken kiezen. We kunnen deze ongeldigheid dan niet teniet doen door middel van onze keuze voor de andere scheidingstekens, hoe goed we deze ook kiezen. Deze situatie treedt op in ons country-code voorbeeld wanneer we bijvoorbeeld de string "/b> <i>" selecteren als scheidingsteken r_1. Deze string is geen prefix van de tekst die na een land attribuutwaarde voorkomt. De string bevat namelijk niet de angle bracket "<" die de "" tag opent. Dit heeft als gevolg dat deze angle bracket opgenomen zal worden in elke attribuutwaarde van het land attribuut. De kandidaten die we selecteren voor de overige scheidingsteken (l_1, l_2 en r_2) zullen dit probleem niet kunnen verhelpen.

De onafhankelijkheid tussen de 2*K* scheidingstekens zorgt ervoor dat we het probleem, namelijk het zoeken van 2*K* scheidingstekens, kunnen opdelen in 2*K* sub-problemen. Elke constraint van de LR wrapper class heeft bijgevolg altijd op exact één scheidingsteken betrekking. Indien dit niet het geval was en de scheidingstekens dus afhankelijk waren van elkaar, dan zouden we alle mogelijke combinaties van de linker en rechter scheidingstekens-kandidaten moeten maken. Het brute-force algoritme dat daaruit zou voortvloeien, zou een veel slechtere tijdsperformantie hebben.

Merk op dat scheidingstekens mogen overlappen, dit om de onafhankelijkheid van de scheidingstekens te bewaren en zo een betere tijdsperformantie te behouden.

4.2.4 Validatie van de kandidaten voor L_K

Nu we de relatie tussen de scheidingstekens hebben besproken kunnen we gaan kijken aan welke constraints de scheidingstekens onderworpen zijn. De constraints voor de l_k scheidingstekens kunnen afgeleid worden door naar de $exec_{LR}$ procedure te kijken. Op regel 3 van Illustratie 4.1 wordt gebruik gemaakt van het scheidingsteken l_1 om te zoeken naar nieuwe tupels. We willen bijgevolg niet dat dit scheidingsteken voorkomt in de voet van de pagina. Omdat deze regel enkel gebruik maakt van scheidingsteken l_1, geldt deze constraint ook enkel voor scheidingsteken l_1. Deze constraint wordt formeel beschreven in Illustratie 4.5 als constraint C^A. Wanneer deze constraint overtreden wordt, dan gaat de $exec_{LR}$ procedure proberen een extra tupel uit de pagina te halen.

Constraint A: u_{l_1} mag geen substring zijn van de voet van eender welke pagina

Constraint B: u_{l_k} moet een zuiver suffix zijn van de tekst die onmiddellijk voorkomt voor elke waarde van attribuut *k* in elk van de pagina's

Illustratie 4.5: Constraints waaraan de l_k scheidingstekens moeten voldoen.

De tweede plaats waar l_k scheidingstekens gebruikt worden in de $exec_{LR}$ procedure, is op regel 6 van Illustratie 4.1. Op deze regel wordt elk l_k scheidingsteken gebruikt om het begin van elke attribuutwaarde voor het k^{de} attribuut aan te duiden. Het scheidingsteken l_k moet bijgevolg een suffix zijn van de tekst die voor de attribuutwaarde verschijnt, meer bepaald moet het scheidingsteken zelfs een *zuiver* suffix zijn. Een string *s* is een zuiver suffix van een string *s'* als *s* een suffix is van *s'* en *s* enkel als een suffix voorkomt in *s'*. Zo is bijvoorbeeld "`>`" geen zuiver suffix van "`<br>⇓<b>`" omdat "`>`" ook het einde van de tag "`<br>`" aanduidt. De string "`<b>`" is wel een zuiver suffix. De reden waarom l_k een zuiver suffix moet zijn en niet een gewone suffix, is omdat we altijd naar de eerst voorkomende positie van een scheidingsteken zoeken in de $exec_{LR}$ procedure. Indien we enkel met gewone suffixen zouden werken, dan zouden er karakters in de attribuutwaarden opgenomen worden die niet tot de attribuutwaarde zelf behoren. Deze constraint geldt voor alle l_k scheidingstekens en is formeel beschreven in Illustratie 4.5 als constraint C^B. Wanneer deze constraint overtreden wordt, dan zal de attribuutwaarde te lang of te kort zijn, afhankelijk van de manier waarop l_k de constraint overtreedt.

We gebruiken de functie $valid_l(u_{l_k},k,\mathcal{E})$ om te controleren of een kandidaat u_{l_k} voldoet aan de constraints C^A en C^B voor het scheidingsteken l_k ten opzichte van de verzameling $\mathcal{E}$ van voorbeeldpagina's en hun bijhorende relaties.

Illustratie 4.6 toont welke waarde de functie $valid_l(u_{l_k},k,\mathcal{E})$ teruggeeft voor de kandidaten in ons lopende country-code voorbeeld. De kandidaten voor l_1 op regel 1 tot en met 7 zijn niet geschikt als scheidingsteken omdat deze kandidaten constraint C^B overtreden. Ze bevatten namelijk karakters in hun string die niet aanwezig zijn vóór de eerste attribuutwaarde. De kandidaat voor l_1 op regel 12 is in overtreding met constraints C^A en C^B. Deze kandidaat is namelijk geen zuiver suffix en komt voor in de voet van de pagina.

1:	$valid_l$(' ⇓',1,$\mathcal{E}$) = FALSE	$valid_l$(' ',2,$\mathcal{E}$) = TRUE
2:	$valid_l$('/i> ⇓',1,$\mathcal{E}$) = FALSE	$valid_l$('/b> ',2,$\mathcal{E}$) = TRUE
3:	$valid_l$('i> ⇓',1,$\mathcal{E}$) = FALSE	$valid_l$('b> ',2,$\mathcal{E}$) = TRUE
4:	$valid_l$('> ⇓',1,$\mathcal{E}$) = FALSE	$valid_l$('> ',2,$\mathcal{E}$) = TRUE
5:	$valid_l$(' ⇓',1,$\mathcal{E}$) = FALSE	$valid_l$(' ',2,$\mathcal{E}$) = TRUE
6:	$valid_l$('br>⇓',1,$\mathcal{E}$) = FALSE	$valid_l$('',2,$\mathcal{E}$) = TRUE
7:	$valid_l$('r>⇓',1,$\mathcal{E}$) = FALSE	$valid_l$('i>',2,$\mathcal{E}$) = TRUE
8:	$valid_l$('>⇓',1,$\mathcal{E}$) = TRUE	$valid_l$('>',2,$\mathcal{E}$) = FALSE
9:	$valid_l$('⇓',1,$\mathcal{E}$) = TRUE	
10:	$valid_l$('',1,$\mathcal{E}$) = TRUE	
11:	$valid_l$('b>',1,$\mathcal{E}$) = TRUE	
12:	$valid_l$('>',1,$\mathcal{E}$) = FALSE	

ILLUSTRATIE 4.6: VALIDATIE VAN DE KANDIDATEN VOOR L_k, VAN HET COUNTRY-CODE VOORBEELD.

De kandidaat voor l_2 op regel 8 is in overtreding met constraint C^B. Deze kandidaat is namelijk geen zuiver suffix van de tekst " ", die zich tussen elke land attribuutwaarde en telefooncode attribuutwaarde bevindt. De kandidaat is niet in overtreding met constraint C^A, in tegenstelling tot de kandidaat voor l_1 op regel 12, omdat constraint C^A enkel op scheidingsteken l_1 van toepassing is.

4.2.5 VALIDATIE VAN DE KANDIDATEN VOOR R_K

De constraints voor de scheidingstekens r_k kunnen ook afgeleid worden door naar de $exec_{LR}$ procedure te kijken. Enkel op regel 7 van Illustratie 4.1 wordt gebruik gemaakt van een r_k scheidingsteken, namelijk om het einde van een attribuutwaarde te vinden. Het scheidingsteken r_k moet bijgevolg een prefix zijn van de tekst die volgt achter de attribuutwaarde. Deze constraint wordt formeel beschreven in Illustratie 4.7 als constraint C^C. Wanneer deze constraint overtreden wordt, dan zal de attribuutwaarde te lang zijn.

De tweede constraint die van toepassing is op elk scheidingsteken r_k stelt dat het scheidingsteken r_k geen substring mag zijn van een attribuutwaarde van attribuut k. De reden voor deze constraint vloeit voort uit het feit dat we altijd naar de eerst voorkomende positie van een scheidingsteken zoeken in de $exec_{LR}$ procedure. Het voorkomen van het scheidingsteken r_k in de waarde voor attribuut k zou dan leiden tot het vroegtijdig afkappen van de attribuutwaarde. Deze constraint wordt formeel

Constraint C: u_{r_k} moet een prefix zijn van de tekst die onmiddellijk voorkomt na elke waarde van attribuut k in elk van de pagina's

Constraint D: u_{r_k} mag geen substring zijn van eender welke waarde van attribuut k in eender welke pagina

ILLUSTRATIE 4.7: CONSTRAINTS WAARAAN DE R_k SCHEIDINGSTEKENS MOETEN VOLDOEN.

1:	$valid_r$(' ',1,$\mathcal{E}$) = TRUE	$valid_r$(' ⇓',2,$\mathcal{E}$) = FALSE
2:	$valid_r$(' <i',1,$\mathcal{E}$) = TRUE	$valid_r$(' ⇓<b',2,$\mathcal{E}$) = FALSE
3:	$valid_r$(' <',1,$\mathcal{E}$) = TRUE	$valid_r$(' ⇓<',2,$\mathcal{E}$) = TRUE
4:	$valid_r$(' ',1,$\mathcal{E}$) = TRUE	$valid_r$(' ⇓',2,$\mathcal{E}$) = TRUE
5:	$valid_r$('',1,$\mathcal{E}$) = TRUE	$valid_r$(' ',2,$\mathcal{E}$) = TRUE
6:	$valid_r$('</b',1,$\mathcal{E}$) = TRUE	$valid_r$('<br',2,$\mathcal{E}$) = TRUE
7:	$valid_r$('</',1,$\mathcal{E}$) = TRUE	$valid_r$('<b',2,$\mathcal{E}$) = TRUE
8:	$valid_r$('<',1,$\mathcal{E}$) = TRUE	$valid_r$('<',2,$\mathcal{E}$) = TRUE
9:		$valid_r$('',2,$\mathcal{E}$) = TRUE
10:		$valid_r$('</i',2,$\mathcal{E}$) = TRUE
11:		$valid_r$('</',2,$\mathcal{E}$) = TRUE
12:		$valid_r$('<',2,$\mathcal{E}$) = TRUE

ILLUSTRATIE 4.8: VALIDATIE VAN DE KANDIDATEN VOOR R_k, VAN HET COUNTRY-CODE VOORBEELD.

beschreven in Illustratie 4.7 als constraint C^D. Wanneer deze constraint overtreden wordt, dan zal de attribuutwaarde te kort zijn.

We gebruiken de functie $\text{valid}_r(u_{r_k},k,\mathcal{E})$ om te controleren of een kandidaat u_{r_k} voldoet aan de constraints C^C en C^D voor het scheidingsteken r_k ten opzichte van de verzameling $\mathcal{E}$ van voorbeeldpagina's en hun bijhorende relaties.

Illustratie 4.8 toont welke waarde de functie $\text{valid}_r(u_{r_k},k,\mathcal{E})$ teruggeeft voor de kandidaten in ons lopende country-code voorbeeld. Enkel de kandidaten voor r_2 op regel 1 en 2 overtreden de constraint C^C. Dit komt omdat deze kandidaten karakters bevatten op het einde van hun string die zich niet voordoen achter het laatste tupel.

4.3 HET LEARN_{LR} ALGORITME

Nu we besproken hebben hoe de exec_{LR} procedure gebruik maakt van een wrapper en hoe we kandidaten voor de scheidingstekens kunnen vinden en valideren, kunnen we het algoritme bespreken dat verantwoordelijk is voor het automatisch leren van een wrapper. Dit algoritme, genaamd learn_{LR}, is weergegeven in Illustratie 4.9.

Het algoritme krijgt als input een verzameling $\mathcal{E}$, van voorbeeldpagina's en hun bijhorende relaties. Op regels 2 tot en met 5 gaat het algoritme opzoek naar de linker scheidingstekens. Hiervoor maakt het algoritme gebruik van de hulpfuncties cands_l (regel 3) en valid_l (regel 4). Deze worden hieronder nog in detail besproken. De volgorde waarin de kandidaten, teruggegeven door cands_l, worden doorlopen (regel 3), is niet van belang. We gaan de kandidaten met andere woorden niet sorteren. Zodra er een geldige kandidaat gevonden wordt voor l_k (regel 4), stopt het algoritme met zoeken naar een scheidingsteken voor l_k (regel 5) en gaat het algoritme op zoek naar een geldig scheidingsteken voor l_{k+1}. Het algoritme gaat op dezelfde wijze tewerk voor de rechter scheidingstekens (regels 6 tot en met 9). De output van het learn_{LR} algoritme is een wrapper $\langle\langle l_1,r_1\rangle,\ldots,\langle l_K,r_K\rangle\rangle$ van $2K$ scheidingstekens.

De hulpfuncties die in Illustratie 4.9 vermeld worden, maar niet beschreven zijn, zijn terug te vinden in Bijlage A.

De functie $\text{cands}_l(k,\mathcal{E})$ geeft alle kandidaten voor het scheidingsteken l_k terug. Hiervoor maakt de functie gebruik van de kortste string die teruggegeven wordt door de $\text{neighbors}_l(k,\mathcal{E})$ functie. Deze $\text{neighbors}_l(k,\mathcal{E})$ functie zoekt namelijk naar alle strings die een waarde van het k^{de} attribuut voorafgaan en waarvan de suffixen in aanmerking kunnen komen als kandidaten voor het scheidingsteken l_k. Het aantal kandidaten dat $\text{cands}_l(k,\mathcal{E})$ als output geeft, is gelijk aan de lengte van de kortste string teruggegeven door $\text{neighbors}_l(k,\mathcal{E})$.

```
1:  procedure learn_LR(examples E)
2:    for each 1 ≤ k ≤ K
3:      for each u_{l_k} ∈ cands_l(k, E)
4:        if valid_l(u_{l_k}, k, E) then
5:          l_k := u_{l_k} and break inner loop
6:    for each 1 ≤ k ≤ K
7:      for each u_{r_k} ∈ cands_r(k, E)
8:        if valid_r(u_{r_k}, k, E) then
9:          r_k := u_{r_k} and break inner loop
10:   return LR wrapper ⟨l_1, r_1, …, l_K, r_K⟩

11: procedure cands_l(index k, examples E)
12:   return set of all suffixes of the shortest string in neighbors_l(k, E)

13: procedure cands_r(index k, examples E)
14:   return set of all prefixes of the shortest string in neighbors_r(k, E)

15: procedure valid_l(candidate u_{l_k}, index k, examples E)
16:   if k = 1 then
17:     for each s ∈ tails(E)
18:       if u_{l_k} is a substring of s then
19:         return FALSE
20:   for each s ∈ neighbors_l(k, E)
21:     if u_{l_k} is not a proper suffix of s then
22:       return FALSE
23:   return TRUE

24: procedure valid_r(candidate u_{r_k}, index k, examples E)
25:   for each s ∈ neighbors_r(k, E)
26:     if u_{r_k} is not a prefix of s then
27:       return FALSE
28:   for each s ∈ attribs(k, E)
29:     if u_{r_k} is a substring of s then
30:       return FALSE
31:   return TRUE
```

ILLUSTRATIE 4.9: HET LEARN$_{LR}$ ALGORITME, DAT AUTOMATISCH EEN LR WRAPPER KAN LEREN UIT EEN VERZAMELING VAN VOORBEELDPAGINA'S EN HUN BIJHORENDE RELATIES.

De functie cands$_r(k,\mathcal{E})$ geeft alle kandidaten voor het scheidingsteken r_k terug. Hiervoor maakt de functie gebruik van de kortste string die teruggegeven wordt door de neighbors$_r(k,\mathcal{E})$ functie. Deze neighbors$_r(k,\mathcal{E})$ functie zoekt namelijk naar alle strings die een waarde van het k^{de} attribuut volgen en waarvan de prefixen in aanmerking kunnen komen als kandidaten voor het scheidingsteken r_k. Het aantal kandidaten dat cands$_r(k,\mathcal{E})$ als output geeft, is gelijk aan de lengte van de kortste string teruggegeven door neighbors$_r(k,\mathcal{E})$.

De functie $\text{valid}_l(u_{l_k},k,\mathcal{E})$ controleert of kandidaat u_{l_k} een geldige kandidaat is voor scheidingsteken l_k. De constraints waaraan u_{l_k} moet voldoen werden reeds besproken in Sectie 4.2.4. Constraint C_l^A vinden we in het algoritme terug op regel 16 en constraint C_l^B vinden we terug op regel 20.

De functie $\text{valid}_r(u_{r_k},k,\mathcal{E})$ controleert of kandidaat u_{r_k} een geldige kandidaat is voor scheidingsteken r_k. De constraints waaraan u_{r_k} moet voldoen werden al besproken in Sectie 4.2.5. Constraint C_r^A vinden we in het algoritme terug op regel 25 en constraint C_r^B vinden we terug op regel 28.

4.4 Conclusie

In dit hoofdstuk hebben we de LR wrapper class besproken. Deze wrapper class maakt gebruik van linker en rechter scheidingstekens voor het vinden van de attribuutwaarden van een relatie op een pagina. Een linker en bijhorend rechter scheidingsteken duiden respectievelijk het begin en einde aan van elke attribuutwaarde van een attribuut. De LR wrapper is bijgevolg een vector van $2K$ scheidingstekens, namelijk $\langle\langle l_1,r_1\rangle,\ldots,\langle l_K,r_K\rangle\rangle$.

Verder hebben we de exec_{LR} procedure besproken die aan de hand van een LR wrapper de relatie uit een pagina probeert te halen.

Tot slot hebben we het learn_{LR} algoritme besproken dat automatisch een LR wrapper kan leren uit een verzameling van voorbeeldpagina's en hun bijhorende relaties.

5. De HLRT Wrapper Class

In het vorige hoofdstuk hebben we de LR wrapper class besproken. In dit hoofdstuk bekijken we een variant op deze wrapper class, namelijk de HLRT (Head-Left-Right-Tail) wrapper class. De nood aan varianten op de LR wrapper class zal snel duidelijk worden. In Figuur 5.1 zien we een variant van het country-code voorbeeld waarmee we gewerkt hebben bij de LR wrapper class (Figuur 3.1). De webpagina in Figuur 5.1 bevat echter meer opmaak dan de webpagina in Figuur 3.1. Zo is er een hoofd-, "Some Country Codes", en voettekst, "End", aanwezig op deze pagina. De hoofd- en voettekst staan ook tussen "" tags, net zoals de attribuutwaarden van het eerste attribuut. De $exec_{LR}$ procedure, die we geïntroduceerd hebben in Hoofdstuk 4, gaat met de LR wrapper ⟨⟨'',''⟩,⟨'',''⟩⟩ bijgevolg teveel tupels uit deze webpagina proberen te halen.

In dit hoofdstuk bespreken we daarom de HLRT wrapper class, een wrapper class die toelaat dat webpagina's gelijkaardige opmaak gebruiken voor niet gestructureerde data. We volgen in dit hoofdstuk dezelfde aanpak als in het vorige hoofdstuk, waarbij we achtereenvolgens een exec procedure, de relatie tussen de scheidingstekens en een learn algoritme bespreken.

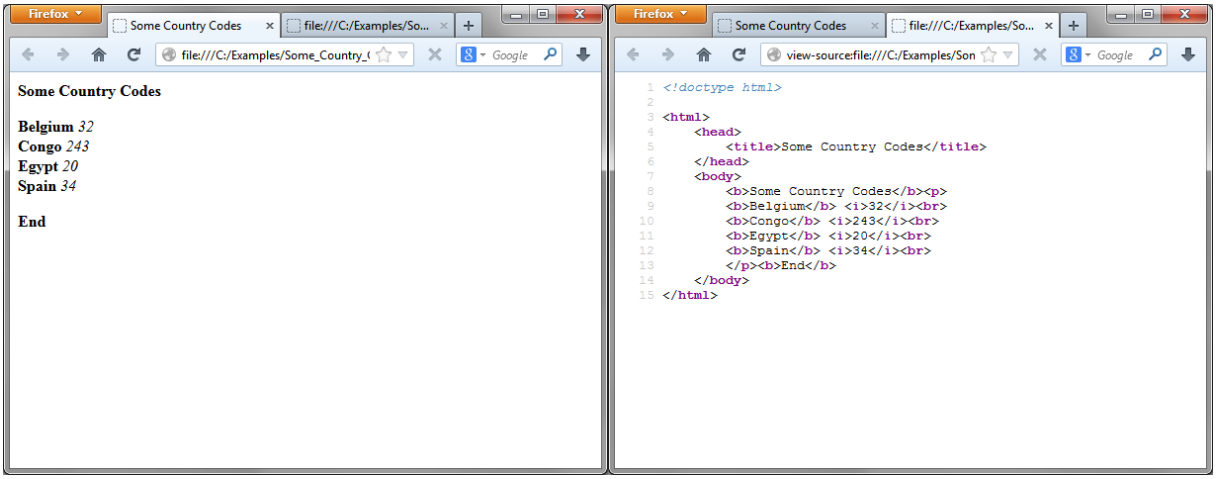
Voor het schrijven van dit hoofdstuk is gebruik gemaakt van de paper "Wrapper Induction: Efficiency and expressiveness" van Kushmerick [11], alsook de paper "Wrapper Induction for Information Extraction" van Kushmerick et al. [12] en de doctoraatsthesis "Wrapper Induction for Information Extraction" van Kushmerick [13]. De basis van het country-code voorbeeld, dat doorheen deze masterproef gebruikt zal worden, is gebaseerd op het country-code voorbeeld dat doorheen de paper "Wrapper Induction: Efficiency and expressiveness" van Kushmerick [11] gebruikt wordt. Onze toevoegingen ten opzichte van deze bronnen zijn het gedetailleerder beschrijven van de wrapper class, alsook het duidelijker definiëren van de constraints van deze wrapper class.

5.1 De $exec_{HLRT}$ procedure

In Figuur 5.1 zien we de webpagina die als voorbeeld zal gebruikt worden doorheen dit hoofdstuk. De vereenvoudigde broncode van deze webpagina is weergegeven in Illustratie 5.1. In de inleiding van dit hoofdstuk hebben we reeds besproken dat enkel gebruik maken van linker en rechter scheidingstekens niet voldoende zal zijn om enkel de gestructureerde informatie uit de pagina te halen. De naam van de wrapper class (HLRT) suggereert hoe we dit probleem kunnen oplossen. We gaan namelijk gebruik

maken van twee extra scheidingstekens, *head* en *tail*. Deze nieuwe scheidingstekens gaan gebruikt worden om het hoofd en de voet van een pagina af te bakenen, met andere woorden, ze duiden aan waartussen de gestructureerde informatie zich bevindt op een pagina. De $\text{exec}_{\text{HLRT}}$ procedure krijgt dus als input een vector van $2K+2$ scheidingstekens, $\langle h, t, [\langle l_1, r_1\rangle, \ldots, \langle l_K, r_K\rangle]\rangle$, naast een pagina waaruit de procedure een relatie moet halen.

We kunnen de exec_{LR} procedure aanpassen, zodat deze rekening houdt met deze nieuwe scheidingstekens. Het resultaat is de $\text{exec}_{\text{HLRT}}$ procedure, weergegeven in Illustratie 5.2.



FIGUUR 5.1: VARIANT VAN DE WEBPAGINA VAN HET COUNTRY-CODE VOORBEELD.

```
1: <html><head><title>Some Country Codes</title></head><body>
2: <b>Some Country Codes</b><p>
3: <b>Belgium</b> <i>32</i><br>
4: <b>Congo</b> <i>243</i><br>
5: <b>Egypt</b> <i>20</i><br>
6: <b>Spain</b> <i>34</i><br>
7: </p><b>End</b>
8: </body></html>
```

ILLUSTRATIE 5.1: VEREENVOUDIGDE BRONCODE VAN DE WEBPAGINA IN FIGUUR 5.1.

```
1:  procedure exec_HLRT(wrapper ⟨h, t, [⟨l_1, r_1⟩, …, ⟨l_K, r_K⟩]⟩, page P)
2:    m := 0
3:    search_move(h, P)
4:    while search(l_1, P) ≤ search(t, P)
5:      m := m + 1
6:      for each ⟨l_k, r_k⟩ ∈ [⟨l_1, r_1⟩, …, ⟨l_K, r_K⟩]
7:        b_m,k := search(l_k, P) + |l_k|
8:        e_m,k := search_after_move(r_k, P, b_m,k)
9:    return relation {⟨⟨b_1,1, e_1,1⟩, …, ⟨b_1,K, e_1,K⟩⟩, …, ⟨⟨b_|R|,1, e_|R|,1⟩, …, ⟨b_|R|,K, e_|R|,K⟩⟩}
```

ILLUSTRATIE 5.2: PROCEDURE $\text{EXEC}_{\text{HLRT}}$, DIE GEBRUIK MAAKT VAN EEN HLRT WRAPPER OM DE RELATIE VAN EEN PAGINA TE VERKRIJGEN.

Deze procedure verschilt slechts op twee punten van de $exec_{LR}$ procedure. Het eerste verschil vinden we terug op regel 3. Alle informatie die tot het hoofd van de pagina behoort, wordt overgeslagen. Het tweede verschil is dat in de while lus op regel 4 enkel naar l_1 scheidingstekens wordt gezocht die voorkomen voor de voet van de pagina.

De hulpfuncties die in de $exec_{HLRT}$ procedure van Illustratie 5.2 vermeld worden, maar niet beschreven zijn, zijn terug te vinden in Bijlage A.

5.2 SCHEIDINGSTEKENS

5.2.1 KANDIDATEN VOOR DE SCHEIDINGSTEKENS

De kandidaten voor de l_k en r_k scheidingstekens kunnen nog steeds op dezelfde manier gevonden worden als bij de LR wrapper class.

De kandidaten voor het head scheidingsteken kunnen gevonden worden door alle substrings te nemen van de kortste string die heads($\mathcal{E}$) teruggeeft. We nemen substrings en geen suffixen, omdat het head scheidingsteken niets exact moet afbakenen. In ons country-code voorbeeld, waarbij er maar één pagina met zijn relatie is in $\mathcal{E}$, is deze string "<html><head><title>Some Country Codes</title></head><body>⇓Some Country Codes ⇓". Deze string levert 4.186 substrings, oftewel kandidaten, op.

De kandidaten voor het tail scheidingsteken kunnen gevonden worden door alle substrings te nemen van de kortste string die tails($\mathcal{E}$) teruggeeft. We nemen substrings en geen prefixen, omdat het tail scheidingsteken niets exact moet afbakenen. In ons country-code voorbeeld is deze string "</i>
⇓ End⇓</body></html>". Deze string levert 741 substrings, oftewel kandidaten, op.

5.2.2 RELATIE TUSSEN DE SCHEIDINGSTEKENS

Net zoals bij de LR wrapper class, zijn ook hier alle l_k en r_k scheidingstekens onafhankelijk van elkaar, met uitzondering van het scheidingsteken l_1. Dit komt omdat de scheidingstekens h, t en l_1 interageren, met andere woorden de keuze van één van deze drie scheidingstekens hangt af van de keuze van de andere twee scheidingstekens. We illustreren dit hieronder aan de hand van het country-code voorbeeld.

Stel dat we voor l_1 de "" tag selecteren als scheidingsteken, net zoals dit het geval was bij het country-code voorbeeld van de LR wrapper class. Indien we dan als scheidingsteken voor h de "<html>" tag selecteren, dan haalt de $exec_{HLRT}$ procedure niet de correcte relatie uit de pagina. De $exec_{HLRT}$ procedure slaat namelijk niet de irrelevante "Some Country Codes" tekst over, die net zoals de attribuutwaarden van het eerste attribuut vetgedrukt staat. Wanneer we daarentegen de " " tag selecteren als scheidingsteken voor h, dan wordt deze irrelevante tekst wel overgeslagen.

De relatie tussen l_1 en t kunnen we op eenzelfde manier illustreren. Stel dat we voor l_1 nog steeds de "" tag gebruiken als scheidingsteken en dat we voor t de "</html>" tag als scheidingsteken selecteren. Doordat achter het laatste tupel en voor het tail scheidingsteken opnieuw irrelevante tekst vetgedrukt staat, namelijk "End", zal de $\text{exec}_{\text{HLRT}}$ procedure niet de correcte relatie uit de webpagina halen. De "</p>" tag daarentegen zal ervoor zorgen dat de $\text{exec}_{\text{HLRT}}$ procedure op tijd stopt met het zoeken naar tupels.

Omdat de scheidingstekens h, t en l_1 interageren, moeten we alle mogelijke combinaties van de kandidaten van elk van deze drie scheidingstekens beschouwen. We moeten met andere woorden het product nemen van de kandidaten van h, t en l_1. Dit levert 37.221.912 mogelijke kandidaat-combinaties op. Dit is het product van 4.186 kandidaten voor h (Sectie 5.2.1), 741 kandidaten voor t (Sectie 5.2.1) en 12 kandidaten voor l_1 (Illustratie 4.3).

Merk op dat scheidingstekens mogen overlappen, dit om de onafhankelijkheid van de scheidingstekens zo veel mogelijk te bewaren. Linker en rechter scheidingstekens zijn nu namelijk onafhankelijk van elkaar. Het verbieden van overlap zou ervoor zorgen dat alle scheidingstekens afhankelijk worden van elkaar en introduceert extra constraints. Dit zou een negatieve impact hebben op de tijdsperformantie.

5.2.3 Validatie van de kandidaten

Nu we de kandidaten voor onze scheidingstekens gevonden hebben, kunnen we kijken naar de constraints waaraan deze kandidaten moeten voldoen, opdat de $\text{exec}_{\text{HLRT}}$ procedure de correcte relaties uit de pagina's kan gaan halen. De constraints voor de l_k en r_k scheidingstekens zijn dezelfde als die van de LR wrapper class (Sectie 4.2.4 en 4.2.5). We bespreken in deze sectie daarom enkel de constraints die betrekking hebben op de interagerende scheidingstekens h, t en l_1. Deze constraints zijn weergegeven in Illustratie 5.3 en worden hieronder toegelicht.

De eerste constraint C^A heeft betrekking op het head scheidingsteken. Dit scheidingsteken moet namelijk in het hoofd van elke pagina voorkomen, zodat de $\text{exec}_{\text{HLRT}}$ procedure, op regel 3 in Illustratie 5.2, de irrelevante hoofdtekst van een pagina kan overslaan. Deze constraint vinden we terug op regel 17 in het $\text{learn}_{\text{HLRT}}$ algoritme (Illustratie 5.4).

De constraint C^B zorgt ervoor dat scheidingsteken l_1 voorkomt voor de attribuutwaarde van het eerste attribuut van het eerste tupel, maar na het head scheidingsteken, zodat de irrelevante hoofdtekst wordt overgeslagen. Indien een scheidingsteken in overtreding is met deze constraint, dan zal de $\text{exec}_{\text{HLRT}}$ procedure, op regel 4 in Illustratie 5.2, het

Constraint A: u_h moet een substring zijn van het hoofd van elke pagina

Constraint B: u_{l_1} moet een zuiver suffix zijn van het gedeelte van het hoofd van elke pagina dat voorkomt na het eerste voorkomen van u_h

Constraint C: u_t mag niet voorkomen tussen het eerste voorkomen van u_h en het daarop volgende voorkomen van u_{l_1} in het hoofd van eender welke pagina

Constraint D: u_t moet een substring zijn van de voet van elke pagina

Constraint E: u_{l_1} mag niet voorkomen voor u_t in de voet van eender welke pagina

Constraint F: u_{l_1} moet een zuiver suffix zijn van de tekst die zich tussen elke twee opeenvolgende tupels bevindt in elke pagina

Constraint G: u_t mag niet voorkomen voor u_{l_1} in de tekst die zich tussen eender welke twee opeenvolgende tupels bevindt in eender welke pagina

ILLUSTRATIE 5.3: CONSTRAINTS WAARAAN DE SCHEIDINGSTEKENS *H*, *T* EN L_1 MOETEN VOLDOEN.

eerste tupel van de pagina overslaan. Deze constraint vinden we terug op regel 19 in het $learn_{HLRT}$ algoritme (Illustratie 5.4).

De constraint C^C waakt erover dat de $exec_{HLRT}$ procedure ten minste één tupel uit de pagina kan halen. Dit doet deze constraint door te eisen dat het scheidingsteken *t* niet voorkomt tussen het scheidingsteken *h* en het eerst voorkomende scheidingsteken l_1. Indien deze constraint overtreden wordt, dan zal de $exec_{HLRT}$ procedure niet in de while lus gaan op regel 4 in Illustratie 5.2. Deze constraint vinden we terug op regel 21 in het $learn_{HLRT}$ algoritme (Illustratie 5.4).

De constraint C^D heeft betrekking op het tail scheidingsteken. Dit scheidingsteken moet namelijk in de voet van elke pagina voorkomen, zodat de while lus in de $exec_{HLRT}$ procedure, op regel 4 in Illustratie 5.2, ooit stopt met zoeken naar tupels. Deze constraint vinden we terug op regel 24 in het $learn_{HLRT}$ algoritme (Illustratie 5.4).

De vorige constraint C^D zorgde ervoor dat de $exec_{HLRT}$ procedure weet waar hij moet stoppen met zoeken naar tupels. Deze constraint voorkomt op zichzelf echter niet dat de $exec_{HLRT}$ procedure geen tupels meer probeert te halen uit de voet van de pagina. Indien er namelijk een scheidingsteken l_1 in de voet van de pagina voorkomt, voor het scheidingsteken *t*, dan zal de $exec_{HLRT}$ procedure proberen nog een tupel uit de pagina te halen. De constraint C^E voorkomt dit probleem door te stellen dat scheidingsteken l_1 niet in de voet van een pagina kan voorkomen voor scheidingsteken *t*. Indien deze constraint overtreden wordt, dan zal de while lus, op regel 4 in Illustratie 5.2, te veel itereren. Deze constraint vinden we terug op regel 26 in het $learn_{HLRT}$ algoritme (Illustratie 5.4).

Waar constraint C^B ervoor zorgt dat scheidingsteken l_1 voorkomt voor de attribuutwaarde van het eerste attribuut van het eerste tupel. Scheidingsteken l_1 moet echter ook voor de attribuutwaarde van het eerste attribuut van alle andere tupels voorkomen. Dit is exact wat constraint C^F eist. Indien deze constraint overtreden wordt, dan zal de $exec_{HLRT}$ procedure op regel 7 in Illustratie 5.2 niet correct werken wanneer $k = 1$. Deze constraint vinden we terug op regel 29 in het $learn_{HLRT}$ algoritme (Illustratie 5.4).

De laatste constraint C^G heeft veel weg van constraint C^C, maar daar waar de constraint C^C ervoor zorgt dat er minstens één tupel uit de pagina wordt gehaald, zorgt de constraint C^G ervoor dat alle tupels uit de pagina worden gehaald. Dit doet de constraint door te stellen dat het scheidingsteken t niet voor l_1 mag voorkomen tussen de tupels. Indien deze constraint overtreden wordt, dan zal de $exec_{HLRT}$ procedure te vroeg stoppen met de uitvoering van de while lus op regel 4 in Illustratie 5.2, en bijgevolg zullen hierdoor niet alle tupels uit de pagina gehaald worden. Deze constraint vinden we terug op regel 31 in het $learn_{HLRT}$ algoritme (Illustratie 5.4).

5.3 HET $LEARN_{HLRT}$ ALGORITME

Nu we besproken hebben hoe de $exec_{HLRT}$ procedure gebruik maakt van een wrapper en hoe we kandidaten voor de nieuwe scheidingstekens kunnen vinden en valideren, kunnen we het algoritme bespreken dat verantwoordelijk is voor het automatisch leren van een HLRT wrapper. Dit algoritme, genaamd $learn_{HLRT}$, is weergegeven in Illustratie 5.4.

Het algoritme krijgt als input een verzameling van voorbeeldpagina's en hun bijhorende relaties, genaamd $\mathcal{E}$. Op regel 2 maakt het $learn_{HLRT}$ algoritme gebruik van het $learn_{LR}$ algoritme om de l_k en r_k scheidingstekens te vinden, met uitzondering van l_1. De drie for lussen op regels 3 tot en met 5 hebben tot doel alle mogelijke combinaties van de kandidaten voor de drie scheidingstekens h, t en l_1 te verkrijgen. Hiervoor wordt respectievelijk gebruik gemaakt van de $cands_h$, $cands_t$ en de $cands_l$ functies. Op regel 6 wordt de $valid_{l_1,h,t}$ functie opgeroepen om de combinaties van de scheidingstekens h, t en l_1 te valideren. Zodra een geldige combinatie gevonden wordt stopt het algoritme met zoeken (regel 7). De output van het $learn_{HLRT}$ algoritme is een wrapper $\langle h, t, [\langle l_1, r_1\rangle, \ldots, \langle l_K, r_K\rangle]\rangle$ van $2K+2$ scheidingstekens.

De functie $cands_l(k,\mathcal{E})$ is dezelfde als degene die in het $learn_{LR}$ algoritme in Illustratie 4.9 werd gebruikt. Deze functie geeft alle kandidaten voor het scheidingsteken l_k terug. Hiervoor maakt de functie gebruik van de kortste string die wordt teruggegeven door de $neighbors_l(k,\mathcal{E})$ functie. Het aantal kandidaten dat $cands_l(k,\mathcal{E})$ als output geeft, is gelijk aan de lengte van de kortste string teruggegeven door $neighbors_l(k,\mathcal{E})$.

```
1:  procedure learn_HLRT(examples E)
2:    ⟨·, r_1, …, l_K, r_K⟩ := learn_LR(E)
3:    for each u_{l_1} ∈ cands_l(1, E)
4:      for each u_h ∈ cands_h(E)
5:        for each u_t ∈ cands_t(E)
6:          if valid_{l_1,h,t}(u_{l_1} u_h u_t, E) then
7:            l_1 := u_{l_1}, h := u_h, t := u_t and break all loops
8:    return HLRT wrapper ⟨h, t, [⟨l_1, r_1⟩, …, ⟨l_K, r_K⟩]⟩

9: procedure cands_l(index k, examples E)
10:   return set of all suffixes of the shortest string in neighbors_l(k, E)

11: procedure cands_h(examples E)
12:   return set of all substrings of the shortest string in heads(E)

13: procedure cands_t(examples E)
14:   return set of all substrings of the shortest string in tails(E)

15: procedure valid_{l_1,h,t}(candidates u_{l_1} u_h u_t, examples E)
16:   for each s ∈ heads(E)
17:     if u_h is not a substring of s then
18:       return FALSE
19:     if u_{l_1} is not a proper suffix of scan(s, u_h) then
20:       return FALSE
21:     if u_t occur before u_{l_1} in scan(s, u_h) then
22:       return FALSE
23:   for each s ∈ tails(E)
24:     if u_t is not a substring of s then
25:       return FALSE
26:     if u_{l_1} occurs before u_t in s then
27:       return FALSE
28:   for each s ∈ seps(K, E)
29:     if u_{l_1} is not a proper suffix of s then
30:       return FALSE
31:     if u_t occurs before u_{l_1} in s then
32:       return FALSE
33:   return TRUE
```

ILLUSTRATIE 5.4: HET LEARN$_{HLRT}$ ALGORITME, DAT AUTOMATISCH EEN HLRT WRAPPER KAN LEREN UIT EEN VERZAMELING VAN VOORBEELDPAGINA'S EN HUN BIJHORENDE RELATIES.

De functie cands$_h$($\mathcal{E}$) geeft alle kandidaten voor het scheidingsteken *h* terug. Hiervoor maakt de functie gebruik van de kortste string die wordt teruggegeven door de heads($\mathcal{E}$) functie. Het aantal kandidaten dat cands$_h$($\mathcal{E}$) als output geeft, is gelijk aan x∗(x+1)/2, waarbij x de lengte van de kortste string is die wordt teruggegeven door heads($\mathcal{E}$).

De functie cands$_t$($\mathcal{E}$) geeft alle kandidaten voor het scheidingsteken *t* terug. Hiervoor maakt de functie gebruik van de kortste string die wordt teruggegeven door de tails($\mathcal{E}$)

functie. Het aantal kandidaten dat $\text{cands}_t(\mathcal{E})$ als output geeft, is gelijk aan $x*(x+1)/2$, waarbij x de lengte van de kortste string is die wordt teruggegeven door $\text{tails}(\mathcal{E})$.

De functie $\text{valid}_{l_1,h,t}(u_{l_1}\ u_h\ u_t, \mathcal{E})$ controleert of de combinatie van de kandidaten u_{l_1}, u_h en u_t geldig is voor de scheidingstekens l_1, h en t. De constraints waaraan u_{l_1}, u_h en u_t moeten voldoen werden reeds besproken in Sectie 5.2.3.

De hulpfuncties die in Illustratie 5.4 vermeld worden, maar niet beschreven zijn, zijn terug te vinden in Bijlage A.

Een voorbeeld van een geldige HLRT wrapper voor het country-code voorbeeld van dit hoofdstuk (Figuur 5.1) is volgende vector van $2K+2$ scheidingstekens:

⟨'</u>
','
⇓
<p>',[⟨'',''⟩,⟨'',''⟩]⟩

5.4 Conclusie

In dit hoofdstuk hebben we de HLRT wrapper class besproken. Deze wrapper class maakt naast de linker en rechter scheidingstekens ook gebruik van een head (h) en tail (t) scheidingsteken voor het vinden van de attribuutwaarden van een relatie op een pagina. Dit head en tail scheidingsteken zorgen ervoor dat de $\text{exec}_{\text{HLRT}}$ procedure geen tupels uit het hoofd en de voet van de pagina probeert te halen. De HLRT wrapper is bijgevolg een vector van $2K+2$ scheidingstekens, namelijk $\langle h, t, [\langle l_1, r_1\rangle, \ldots, \langle l_K, r_K\rangle]\rangle$.

Verder hebben we gezien dat de scheidingstekens l_1, h en t afhankelijk zijn van elkaar, waardoor we alle combinaties van de kandidaten van deze drie scheidingstekens in beschouwing moeten nemen.

Tot slot hebben we het $\text{learn}_{\text{HLRT}}$ algoritme besproken dat automatisch een HLRT wrapper kan leren uit een verzameling van voorbeeldpagina's en hun bijhorende relaties.

6. De OCLR Wrapper Class

In dit hoofdstuk bespreken we een tweede variant op de LR wrapper class, namelijk de OCLR (Open-Close-Left-Right) wrapper class. Deze wrapper class maakt, zoals de naam al aangeeft, gebruik van een *open* en *close* scheidingsteken, naast de linker en rechter scheidingstekens. Het open en close scheidingsteken duiden respectievelijk het begin en het einde van elk tupel aan.

De reden waarom we tweede een variant willen hebben op de LR wrapper class wordt duidelijk in Figuur 6.1. In deze figuur wordt namelijk een variant van de country-code webpagina weergegeven waarin de landen en hun bijhorende telefooncodes gegroepeerd zijn per continent. De naam van elk continent is net zoals de naam van elk land in het vetgedrukt geschreven. Daarbovenop is elke continentnaam ook cursief geschreven en onderlijnd.

Wanneer we de $\text{exec}_{\text{HLRT}}$ procedure uitvoeren op de webpagina van Figuur 6.1, met de HLRT wrapper ⟨'</u>
','
⇓
<p>',[⟨'',''⟩,⟨'',''⟩]⟩ als input, dan zal de $\text{exec}_{\text{HLRT}}$ procedure in de fout gaan bij "Africa". Dit komt omdat deze continentnaam voorafgegaan wordt door een "" tag, wat leidt tot het tupel <"Africa", 243>. Dit probleem is te wijten aan het scheidingsteken l_1. Indien we dit scheidingsteken kunnen uitbreiden, met andere woorden langer maken, zodat dit niet meer voorkomt voor een continentnaam, dan kunnen we wel een geldige HLRT wrapper vinden. Een mogelijke uitbreiding voor dit scheidingsteken is "<li>", ware het niet voor de asterisk die voor "Vatican City" staat. Het is bijgevolg niet mogelijk om een geldige HLRT wrapper te vinden voor dit voorbeeld. Doordat de HLRT wrapper bovendien een uitbreiding is van de LR wrapper, is het ook niet mogelijk om een geldige LR wrapper te vinden. De OCLR wrapper class kan echter wel met deze country-code webpagina overweg.

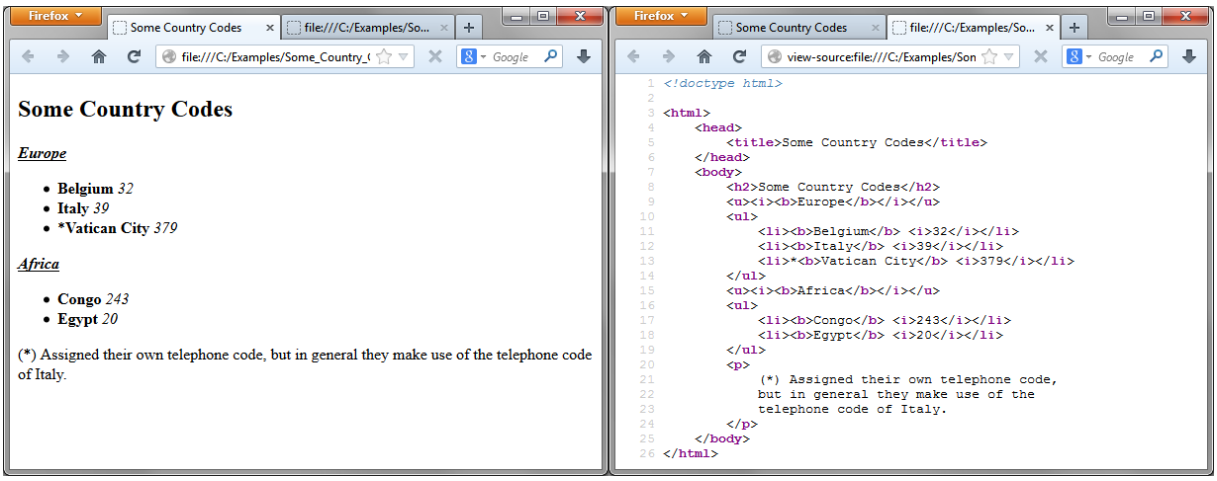
Het verloop van dit hoofdstuk is hetzelfde als in de voorgaande hoofdstukken. We beginnen met het bespreken van een exec procedure voor de OCLR wrapper. Vervolgens gaan we verder met de relatie tussen de scheidingstekens. Tot slot eindigen we met een learn algoritme dat automatisch een OCLR wrapper kan leren.

Voor het schrijven van dit hoofdstuk is gebruik gemaakt van de paper "Wrapper Induction: Efficiency and expressiveness" van Kushmerick [11], alsook de paper "Wrapper Induction for Information Extraction" van Kushmerick et al. [12] en de doctoraatsthesis "Wrapper Induction for Information Extraction" van Kushmerick [13]. De basis van het country-code voorbeeld, dat doorheen deze masterproef gebruikt zal

worden, is gebaseerd op het country-code voorbeeld dat doorheen de paper "Wrapper Induction: Efficiency and expressiveness" van Kushmerick [11] gebruikt wordt. Onze toevoegingen ten opzichte van deze bronnen zijn het gedetailleerder beschrijven van de wrapper class, het duidelijker definiëren van de constraints van deze wrapper class en het gebruiken van een voorbeeld dat duidelijk de nood aan een OCLR wrapper class aantoont.

6.1 De $\text{exec}_{\text{OCLR}}$ procedure

In Figuur 6.1 wordt de country-code webpagina weergegeven die in dit hoofdstuk als voorbeeld zal gebruikt worden. We hebben het country-code voorbeeld aangepast ten opzichte van de voorgaande hoofdstukken. Zo hebben we Spanje weggelaten en hebben we Italië en Vaticaanstad toegevoegd. Dit leidde tot de toevoeging van de opmerking, aangegeven door de asterisk. Deze opmerking zorgt ervoor dat we geen LR of HLRT wrapper class kunnen gebruiken, zoals reeds in de inleiding is besproken. Een tweede aanpassing die we hebben doorgevoerd is de groepering van landen en hun telefooncode per continent. Hierdoor bevat de webpagina nu twee HTML lists. Dit lijkt op het eerste zicht misschien vreemd, maar zoals we reeds in Sectie 1.1 besproken hebben, zijn alle wrapper methoden geschikt voor tekst formatting. We hebben toen ook besproken dat tekst formatting een veralgemening is van HTML tables, HTML lists en HTML formatting. We kunnen deze twee lijsten bijgevolg beschouwen als een vorm van tekst formatting.



Figuur 6.1: Variant van de webpagina van het country-code voorbeeld.

Illustratie 6.1 geeft de vereenvoudigde broncode weer van de country-code webpagina van Figuur 6.1. Om deze vereenvoudigde broncode overzichtelijk te houden, worden de attribuutwaarden die we uit deze webpagina willen halen in het vet weergegeven. De relatie die we uit de webpagina willen halen is nog steeds dezelfde, op de weglating van

```
1:  <html><head><title>Some Country Codes</title></head><body>
2:  <h2>Some Country Codes</h2>
3:  <u><i><b>Europe</b></i></u><ul>
4:  <li><b>Belgium</b> <i>32</i></li>
5:  <li><b>Italy</b> <i>39</i></li>
6:  <li>*<b>Vatican City</b> <i>379</i></li>
7:  </ul><u><i><b>Africa</b></i></u><ul>
8:  <li><b>Congo</b> <i>243</i></li>
9:  <li><b>Egypt</b> <i>20</i></li>
10: </ul><p>(*) Assigned their own telephone code, but in general they make
                                    use of the telephone code of Italy.</p>
11: </body></html>
```

ILLUSTRATIE 6.1: VEREENVOUDIGDE BRONCODE VAN DE WEBPAGINA IN FIGUUR 6.1.

$$R_{cc} = \left\{ \begin{array}{c} \langle \text{'Belgium','32'} \rangle \\ \langle \text{'Italy','39'} \rangle \\ \langle \text{'Vatican City','379'} \rangle \\ \langle \text{'Congo','243'} \rangle \\ \langle \text{'Egypt','20'} \rangle \end{array} \right\}$$

ILLUSTRATIE 6.2: DE RELATIE DIE WE UIT DE WEBPAGINA VAN FIGUUR 6.1 WILLEN HALEN.

```
1:  procedure exec_OCLR(wrapper ⟨o,c,[⟨l_1,r_1⟩,…,⟨l_K,r_K⟩]⟩, page P)
2:    m := 0
3:    while search(o,P) ≠ EOF
4:      m := m + 1
5:      search_move(o,P)
6:      for each ⟨l_k,r_k⟩ ∈ [⟨l_1,r_1⟩,…,⟨l_K,r_K⟩]
7:        b_m,k := search(l_k,P) + |l_k|
8:        e_m,k := search_after_move(r_k,P,b_m,k)
9:      search_move(c,P)
10:   return relation {⟨⟨b_1,1,e_1,1⟩,…,⟨b_1,K,e_1,K⟩⟩, … , ⟨⟨b_|R|,1,e_|R|,1⟩,…,⟨b_|R|,K,e_|R|,K⟩⟩}
```

ILLUSTRATIE 6.3: PROCEDURE $\text{EXEC}_{\text{OCLR}}$, DIE GEBRUIK MAAKT VAN EEN OCLR WRAPPER OM DE RELATIE VAN EEN PAGINA TE VERKRIJGEN.

Spanje en de toevoeging van Italië en Vaticaanstad na. Deze relatie is weergegeven in Illustratie 6.2.

De OCLR wrapper bevat, net zoals de HLRT wrapper, $2K+2$ scheidingstekens. Het enige verschil is dat het head en tail scheidingsteken zijn vervangen door een open en close scheidingsteken. De $\text{exec}_{\text{OCLR}}$ procedure krijgt bijgevolg, naast de pagina waaruit de procedure een relatie moet halen, een vector van $2K+2$ scheidingstekens als input, namelijk $\langle o, c, [\langle l_1, r_1 \rangle, \ldots, \langle l_K, r_K \rangle] \rangle$.

Wanneer we de exec_{LR} procedure aanpassen om rekening te houden met een open en close scheidingsteken, bekomen we de $\text{exec}_{\text{OCLR}}$ procedure die in Illustratie 6.3 wordt weergegeven. Deze procedure verschilt op twee punten van de exec_{LR} procedure. Het

eerste verschil bevindt zich in de while lus op regel 3. Hierin wordt telkens naar een open scheidingsteken gezocht in plaats van het scheidingsteken l_1. Op regel 5 wordt vervolgens ook telkens naar de positie van dit open scheidingsteken gegaan. Het tweede verschil is dat er na het extraheren van elke attribuutwaarde van het tupel er naar het eerstvolgende close scheidingsteken wordt gegaan.

De hulpfuncties die in de $\text{exec}_{\text{OCLR}}$ procedure van Illustratie 6.3 vermeld worden, maar niet beschreven zijn, zijn terug te vinden in Bijlage A.

6.2 SCHEIDINGSTEKENS

6.2.1 KANDIDATEN VOOR DE SCHEIDINGSTEKENS

De kandidaten voor de l_k en r_k scheidingstekens kunnen op dezelfde manier gevonden worden als bij de LR wrapper class.

Het open en close scheidingsteken moeten beiden voorkomen tussen elk tupel. De kandidaten voor deze twee scheidingstekens kunnen bijgevolg op dezelfde manier gevonden worden, namelijk door alle substrings te nemen van de kortste string die $\text{seps}(K,\mathcal{E})$ teruggeeft. We nemen substrings en geen suffixen of prefixen, omdat het open en close scheidingsteken niets exact moeten afbakenen. In ons country-code voorbeeld, is deze kortste string gelijk aan "</i>
⇓
<b>". Deze string levert 153 substrings, oftewel kandidaten, op.

De kandidaten voor het scheidingsteken l_1 worden eveneens gegenereerd uit de string "</i>
⇓
<b>". Dit is namelijk de kortste string die $\text{neighbors}_l(1,\mathcal{E})$ teruggeeft. Voor meer informatie over de werking van deze neighbors_l functie kan men Bijlage A raadplegen. De cands_l procedure genereert echter enkel suffixen uit deze string, in tegenstelling tot de $\text{cands}_{o,c}$ functie, die alle mogelijke substrings genereert. Dit zorgt voor een totaal van 17 kandidaten voor het scheidingsteken l_1.

6.2.2 RELATIE TUSSEN DE SCHEIDINGSTEKENS

Net zoals bij de LR wrapper class, zijn ook hier alle l_k en r_k scheidingstekens onafhankelijk van elkaar, met uitzonder van het scheidingsteken l_1. Dit komt omdat de scheidingstekens o, c en l_1 interageren, met andere woorden de keuze van één van deze drie scheidingstekens hangt af van de keuze van de andere twee scheidingstekens. We illustreren dit hieronder aan de hand van het country-code voorbeeld. Herinner dat bij de HLRT wrapper class de scheidingstekens h, t en l_1 ook interageren.

Stel dat we voor l_1 de "<b>" tag selecteren als scheidingsteken, net zoals bij het country-code voorbeeld van de LR wrapper class. Wanneer we dan als scheidingsteken voor o de

kandidaat "<" selecteren, dan haalt de $exec_{OCLR}$ procedure niet de correcte relatie uit de pagina. De $exec_{OCLR}$ procedure slaat namelijk niet de irrelevante "Europe" tekst over, die net zoals de attribuutwaarden van het eerste attribuut vetgedrukt staat. Dit komt omdat het scheidingsteken *o* onmiddellijk voorkomt in de "<html>" tag. Indien we daarentegen de "<li>" tag selecteren als scheidingsteken voor *o*, dan wordt deze irrelevante tekst wel overgeslagen.

De relatie tussen de scheidingstekens l_1 en *c* is gebaseerd op een gelijkaardige analogie. Ons country-code voorbeeld bevat echter geen illustratie van deze relatie. De keuze van het scheidingsteken *c* wordt in dit voorbeeld met andere woorden niet beïnvloed door de keuze van de scheidingstekens *o* en l_1. Ongeldige kandidaten voor het scheidingsteken *c*, zijn bijgevolg ongeldig omwille van het feit dat deze kandidaten niet na elk tupel in de webpagina voorkomen.

Omdat de scheidingstekens *o*, *c* en l_1 interageren, moeten we alle mogelijke combinaties van de kandidaten van elk van deze drie scheidingstekens in beschouwing nemen. We moeten met andere woorden het product nemen van de kandidaten van *o*, *c* en l_1. Dit levert 397.953 mogelijke kandidaat-combinaties op. Dit is het product van 153 kandidaten voor *o* (Sectie 6.2.1), 153 kandidaten voor *c* (Sectie 6.2.1) en 17 kandidaten voor l_1 (Sectie 6.2.1).

Merk op dat scheidingstekens mogen overlappen, dit om de onafhankelijkheid van de scheidingstekens zo veel mogelijk te bewaren. Linker en rechter scheidingstekens zijn nu namelijk onafhankelijk van elkaar. Het verbieden van overlap zou ervoor zorgen dat alle scheidingstekens afhankelijk worden van elkaar en introduceert extra constraints. Dit zou een negatieve impact hebben op de tijdsperformantie.

6.2.3 Validatie van de kandidaten

Nu we de kandidaten voor onze scheidingstekens gevonden hebben, kunnen we kijken naar de constraints waaraan deze kandidaten moeten voldoen, opdat de $exec_{OCLR}$ procedure de correcte relaties uit de pagina's kan gaan halen. De constraints voor de l_k en r_k scheidingstekens zijn dezelfde als bij de LR wrapper class (Sectie 4.2.4 en 4.2.5). We bespreken in deze sectie daarom enkel de constraints die betrekking hebben op de interagerende scheidingstekens *o*, *c* en l_1. Deze constraints zijn weergegeven in Illustratie 6.4 en worden hieronder toegelicht.

De eerste constraint C^A heeft betrekking op het open scheidingsteken. Dit scheidingsteken moet namelijk in het hoofd van elke pagina voorkomen, zodat de $exec_{OCLR}$ procedure, op regel 3 in Illustratie 6.3, het begin van het eerste tupel vindt en

Constraint A: u_o moet een substring zijn van het hoofd van elke pagina

Constraint B: u_{l_1} moet een zuiver suffix zijn van het gedeelte van het hoofd van elke pagina dat voorkomt na het eerste voorkomen van u_o

Constraint C: u_c moet een substring zijn van de voet van elke pagina

Constraint D: u_o mag niet voorkomen na u_c in de voet van eender welke pagina

Constraint E: u_o moet een substring zijn van de tekst die zich tussen elke twee opeenvolgende tupels bevindt in elke pagina

Constraint F: u_c moet een substring zijn van de tekst die zich tussen elke twee opeenvolgende tupels bevindt in elke pagina

Constraint G: u_{l_1} moet een zuiver suffix zijn van de tekst die voorkomt na u_o, in de tekst die voorkomt na u_c, in de tekst tussen elke twee opeenvolgende tupels, en in dit in elke pagina

ILLUSTRATIE 6.4: CONSTRAINTS WAARAAN DE SCHEIDINGSTEKENS *O*, *C* EN L_1 MOETEN VOLDOEN.

op regel 5 in Illustratie 6.3, de irrelevante hoofdtekst van een pagina kan overslaan. Deze constraint vinden we terug op regel 15 in het $\text{learn}_{\text{OCLR}}$ algoritme (Illustratie 6.5).

De constraint C^B zorgt ervoor dat scheidingsteken l_1 voorkomt voor de attribuutwaarde van het eerste attribuut van het eerste tupel, maar na het open scheidingsteken, zodat de irrelevante hoofdtekst overgeslagen wordt. Indien het scheidingsteken in overtreding is met deze constraint, dan zal de $\text{exec}_{\text{OCLR}}$ procedure, op regel 6 in Illustratie 6.3, een verkeerde attribuutwaarde uit de pagina halen. Deze verkeerde waarde kan afkomstig zijn van het tweede tupel of van een stuk tekst dat zich tussen twee tupels bevindt en begint met het scheidingsteken l_1. Deze constraint vinden we terug op regel 17 in het $\text{learn}_{\text{OCLR}}$ algoritme (Illustratie 6.5).

De constraint C^C heeft betrekking op het close scheidingsteken. Dit scheidingsteken moet namelijk in de voet van elke pagina voorkomen, zodat het laatste tupel van elke pagina wordt afgesloten in de $\text{exec}_{\text{OCLR}}$ procedure, op regel 9 in Illustratie 6.3. Deze constraint vinden we terug op regel 20 in het $\text{learn}_{\text{OCLR}}$ algoritme (Illustratie 6.5).

De volgende constraint is C^D. Deze constraint voorkomt dat de $\text{exec}_{\text{OCLR}}$ procedure tupels probeert te halen uit de voet van de pagina. Dit doet de constraint door te eisen dat er geen open scheidingsteken meer voorkomt na het laatste close scheidingsteken. Indien deze constraint overtreden wordt, dan zal de while lus, op regel 3 in Illustratie 6.3, te veel itereren. Deze constraint vinden we terug op regel 22 in het $\text{learn}_{\text{OCLR}}$ algoritme (Illustratie 6.5).

De constraint C^E vereist dat het open scheidingsteken, op elke pagina, tussen alle paren van twee opeenvolgende tupels voorkomt. Deze constraint zorgt, samen met constraint C^A, ervoor dat het open scheidingsteken voorkomt voor elk tupel op elke pagina. Zonder deze constraints zouden er tupels worden overgeslagen op regel 3 in de $\text{exec}_{\text{OCLR}}$ procedure (Illustratie 6.3). Deze constraint vinden we terug op regel 25 in het $\text{learn}_{\text{OCLR}}$ algoritme (Illustratie 6.5).

Wat constraint C^E doet voor het open scheidingsteken doet constraint C^F voor het close scheidingsteken. Constraint C^F vereist namelijk dat het close scheidingsteken, op elke pagina, tussen alle paren van twee opeenvolgende tupels voorkomt. Deze constraint zorgt, samen met constraint C^C, ervoor dat het close scheidingsteken voorkomt na elk tupel op elke pagina. Zonder deze constraints zouden er tupels worden overgeslagen op regel 9 in de $\text{exec}_{\text{OCLR}}$ procedure (Illustratie 6.3). Deze constraint vinden we terug op regel 27 in het $\text{learn}_{\text{OCLR}}$ algoritme (Illustratie 6.5).

De laatste constraint C^G heeft ook betrekking op de tekst tussen alle paren van twee opeenvolgende tupels, op elke pagina. Deze constraint stelt namelijk dat de volgorde van de scheidingstekens o, c en l_1 als volgt moet zijn: c, o en l_1. De constraint zorgt, in de $\text{exec}_{\text{OCLR}}$ procedure, voor de goede werking van regels 5, 6 en 9 in Illustratie 6.3. Indien deze constraint overtreden wordt, dan worden er tupels overgeslapen. Deze constraint vinden we terug op regel 29 in het $\text{learn}_{\text{OCLR}}$ algoritme (Illustratie 6.5).

6.3 HET $\text{LEARN}_{\text{OCLR}}$ ALGORITME

Nu we besproken hebben hoe de $\text{exec}_{\text{OCLR}}$ procedure gebruik maakt van een wrapper en hoe we kandidaten voor de nieuwe scheidingstekens kunnen vinden en valideren, kunnen we het algoritme bespreken dat verantwoordelijk is voor het automatisch leren van een OCLR wrapper. Dit algoritme, genaamd $\text{learn}_{\text{OCLR}}$, is weergegeven in Illustratie 6.5.

Het algoritme krijgt als input een verzameling van voorbeeldpagina's en hun bijhorende relaties, genaamd $\mathcal{E}$. Op regel 2 maakt het $\text{learn}_{\text{OCLR}}$ algoritme gebruik van het learn_{LR} algoritme om de l_k en r_k scheidingstekens te vinden, met uitzondering van l_1. De drie for lussen op regels 3 tot en met 5 dienen om alle mogelijke combinaties van de kandidaten voor de scheidingstekens o, c en l_1 te verkrijgen. Hiervoor wordt gebruik gemaakt van de $\text{cands}_{o,c}$ en de cands_l functies. Op regel 6 wordt de $\text{valid}_{l_1,o,c}$ functie opgeroepen om de combinaties van de scheidingstekens o, c en l_1 te valideren. Zodra een geldige combinatie gevonden wordt stopt het algoritme met zoeken (regel 7). De output van het $\text{learn}_{\text{OCLR}}$ algoritme is een wrapper $\langle o, c, [\langle l_1, r_1\rangle, \ldots, \langle l_K, r_K\rangle]\rangle$ van $2K+2$ scheidingstekens.

De functie $\text{cands}_l(k,\mathcal{E})$ is dezelfde als degene die in het learn_{LR} algoritme in Illustratie 4.9 werd gebruikt. Deze functie geeft alle kandidaten voor het scheidingsteken l_k terug.

```
1:  procedure learn_OCLR(examples ℰ)
2:    ⟨·, r_1, …, l_K, r_K⟩ ≔ learn_LR(ℰ)
3:    for each u_l1 ∈ cands_l(1, ℰ)
4:      for each u_o ∈ cands_o,c(ℰ)
5:        for each u_c ∈ cands_o,c(ℰ)
6:          if valid_l1,o,c(u_l1 u_o u_c, ℰ) then
7:            l_1 ≔ u_l1, o ≔ u_o, c ≔ u_c and break all loops
8:    return OCLR wrapper ⟨o, c, [⟨l_1, r_1⟩, …, ⟨l_K, r_K⟩]⟩

9:  procedure cands_l(index k, examples ℰ)
10:   return set of all suffixes of the shortest string in neighbors_l(k, ℰ)

11: procedure cands_o,c(examples ℰ)
12:   return set of all substrings of the shortest string in seps(K, ℰ)

13: procedure valid_l1,o,c(candidates u_l1 u_o u_c, examples ℰ)
14:   for each s ∈ heads(ℰ)
15:     if u_o is not a substring of s then
16:       return FALSE
17:     if u_l1 is not a proper suffix of scan(s, u_o) then
18:       return FALSE
19:   for each s ∈ tails(ℰ)
20:     if u_c is not a substring of s then
21:       return FALSE
22:     if u_o occurs after u_c in s then
23:       return FALSE
24:   for each s ∈ seps(K, ℰ)
25:     if u_o is not a substring of s then
26:       return FALSE
27:     if u_c is not a substring of s then
28:       return FALSE
29:     if u_l1 is not a proper suffix of scan(scan(s, u_c), u_o) then
30:       return FALSE
31:   return TRUE
```

ILLUSTRATIE 6.5: HET LEARN$_{OCLR}$ ALGORITME, DAT AUTOMATISCH EEN OCLR WRAPPER KAN LEREN UIT EEN VERZAMELING VAN VOORBEELDPAGINA'S EN HUN BIJHORENDE RELATIES.

Hiervoor maakt de functie gebruik van de kortste string teruggegeven door de neighbors$_l(k,\mathcal{E})$ functie. Het aantal kandidaten dat cands$_l(k,\mathcal{E})$ als output geeft, is gelijk aan de lengte van de kortste string teruggegeven door neighbors$_l(k,\mathcal{E})$.

De functie cands$_{o,c}(\mathcal{E})$ geeft alle kandidaten terug die zowel voor het scheidingsteken *o* als voor het scheidingsteken *c* gebruikt kunnen worden. Hiervoor maakt de functie gebruik van de kortste string die teruggegeven wordt door de seps$(K,\mathcal{E})$ functie. Het aantal kandidaten dat cands$_{o,c}(\mathcal{E})$ als output geeft, is gelijk aan $x*(x+1)/2$, waarbij x de lengte is van de kortste string teruggegeven wordt door seps$(K,\mathcal{E})$ is.

De functie $\text{valid}_{l_1,o,c}(u_{l_1}\ u_o\ u_c, \mathcal{E})$ controleert of de combinatie van de kandidaten u_{l_1}, u_o en u_c geldig is voor de scheidingstekens l_1, o en c. De constraints waaraan u_{l_1}, u_o en u_c moeten voldoen werden reeds besproken in Sectie 6.2.3.

De hulpfuncties die in Illustratie 6.5 vermeld worden, maar niet beschreven zijn, zijn terug te vinden in Bijlage A.

Een voorbeeld van een geldige OCLR wrapper voor het country-code voorbeeld van dit hoofdstuk (Figuur 6.1) is volgende vector van $2K+2$ scheidingstekens:

$$\langle\texttt{'
'},\texttt{'
'},[\langle\texttt{''},\texttt{''}\rangle,\langle\texttt{'<i>'},\texttt{'</i>'}\rangle]\rangle$$

6.4 Conclusie

In dit hoofdstuk hebben we de OCLR wrapper class besproken. Deze wrapper class maakt naast de linker en rechter scheidingstekens ook gebruik van een open (o) en close (c) scheidingsteken voor het vinden van de attribuutwaarden van een relatie op een pagina. Dit open en close scheidingsteken duiden respectievelijk het begin en het einde van elk tupel op de pagina aan, waardoor de $\text{exec}_{\text{OCLR}}$ procedure enkel attribuutwaarden van tupels uit de webpagina haalt. De OCLR wrapper is bijgevolg een vector van $2K+2$ scheidingstekens, namelijk $\langle o, c, [\langle l_1, r_1\rangle, \ldots, \langle l_K, r_K\rangle]\rangle$.

Verder hebben we gezien dat de scheidingstekens l_1, o en c afhankelijk zijn van elkaar, waardoor we alle combinaties van de kandidaten van deze drie scheidingstekens in beschouwing moeten nemen.

Tot slot hebben we het $\text{learn}_{\text{OCLR}}$ algoritme besproken dat automatisch een OCLR wrapper kan leren uit een verzameling van voorbeeldpagina's en hun bijhorende relaties.

7. De HOCLRT Wrapper Class

In dit hoofdstuk bespreken we de laatste reeds bestaande wrapper class, namelijk de HOCLRT (Head-Open-Close-Left-Right-Tail) wrapper class. De naam geeft aan dat deze wrapper class gebruik maakt van het head, tail, open en close scheidingsteken. Deze wrapper class combineert met andere woorden de functionaliteit van de HLRT en OCLRT wrapper class.

De reden waarom we een wrapper class willen hebben die de functionaliteit van de HLRT en OCLRT wrapper combineert is gelijkaardig aan de reden waarom we een HLRT wrapper class wilden hebben, ondanks het bestaan van de LR wrapper class. Dit was om overweg te kunnen met webpagina's waarin het scheidingsteken l_1 in het hoofd of in de voet van de pagina voorkwam. Deze keer draait het echter niet om het scheidingsteken l_1, maar om het scheidingsteken *o*.

In Figuur 7.1 wordt een nieuwe variant op de country-code webpagina van het vorige hoofdstuk weergegeven. Indien we de $\text{exec}_{\text{OCLR}}$ procedure op deze webpagina uitvoeren, met als input de OCLR wrapper ⟨'
','
',[⟨'',''⟩,⟨'',''⟩]⟩, dan zal de $\text{exec}_{\text{OCLR}}$ procedure denken dat de "<li>" tags na "Sources" nieuwe tupels aanduiden. Het scheidingsteken *o* komt in dit voorbeeld namelijk nog voor in de voet van de webpagina. De $\text{exec}_{\text{HLRT}}$ procedure is echter niet beter geschikt voor het automatisch leren van een wrapper. Zo hebben we in Hoofdstuk 6 reeds gezien dat de $\text{exec}_{\text{HLRT}}$ procedure in de fout gaat bij "Africa" wanneer de procedure de HLRT wrapper ⟨'</u>
','
⇓
<p>',[⟨'',''⟩,⟨'',''⟩]⟩ als input krijgt. Een combinatie van deze twee wrappers en een exec procedure die met deze nieuwe wrapper overweg kan, levert wel het gewenste resultaat op.

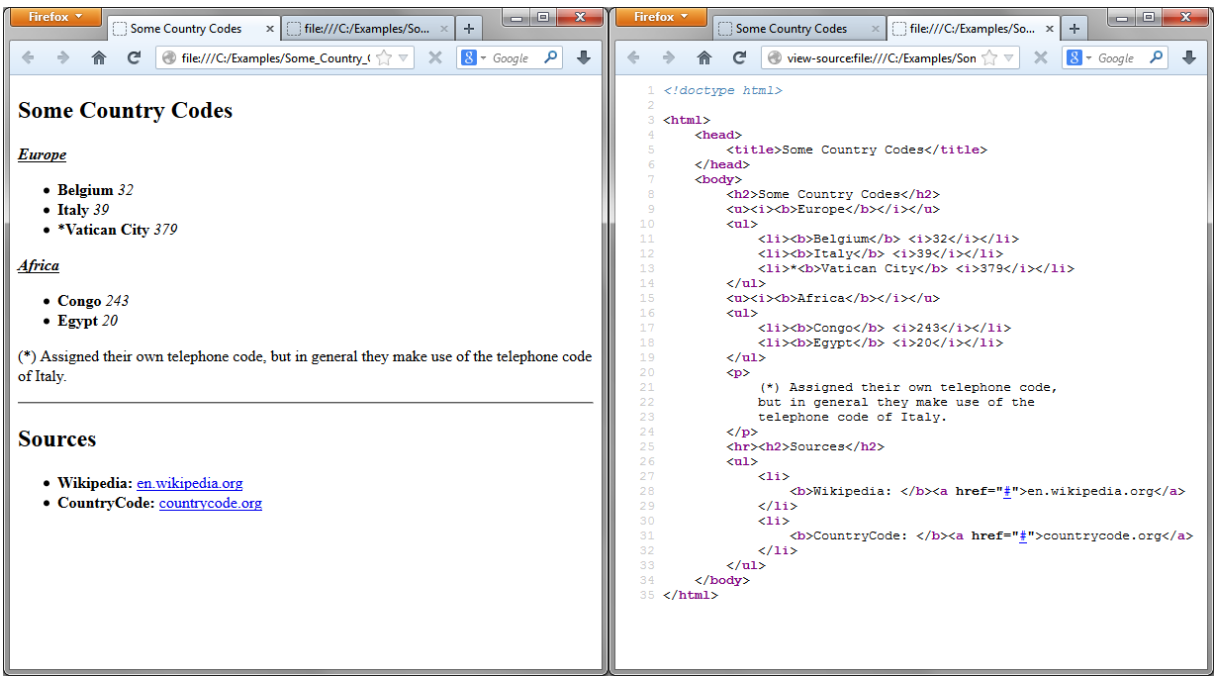
We beginnen dit hoofdstuk met het bespreken van een exec procedure voor de HOCLRT wrapper. Vervolgens gaan we verder met de relatie tussen de scheidingstekens. Daarna bespreken we een learn algoritme dat automatisch een HOCLRT wrapper kan leren. Tot slot eindigen we met de relatie tussen de HLRT en HOCLRT wrapper class.

Voor het schrijven van dit hoofdstuk is gebruik gemaakt van de paper "Wrapper Induction: Efficiency and expressiveness" van Kushmerick [11], alsook de paper "Wrapper Induction for Information Extraction" van Kushmerick et al. [12] en de doctoraatsthesis "Wrapper Induction for Information Extraction" van Kushmerick [13]. De basis van het country-code voorbeeld, dat doorheen deze masterproef gebruikt zal

worden, is gebaseerd op het country-code voorbeeld dat doorheen de paper "Wrapper Induction: Efficiency and expressiveness" van Kushmerick [11] gebruikt wordt. Onze toevoegingen ten opzichte van deze bronnen zijn het gedetailleerder beschrijven van de wrapper class, het duidelijker definiëren van de constraints van deze wrapper class en het gebruiken van een voorbeeld dat duidelijk de nood aan een HOCLRT wrapper class aantoont.

7.1 De exec$_{HOCLRT}$ procedure

De country-code webpagina die doorheen dit hoofdstuk als voorbeeld zal gebruikt worden, is in Figuur 7.1 weergegeven. Deze webpagina heeft, ten opzichte van de country-code webpagina van het vorige hoofdstuk (Figuur 6.1), een voet gekregen waarin de gebruikte bronnen voor de webpagina vermeld worden. Deze worden tussen "`<li>`" tags vermeld. De internetadressen van deze bronnen zijn voorafgegaan door de naam van de website, vermeld tussen "`<b>`" tags. Illustratie 7.1 geeft de vereenvoudigde broncode weer van de country-code webpagina van Figuur 7.1. Om deze vereenvoudigde broncode overzichtelijk te houden, worden de attribuutwaarden die we uit deze webpagina willen halen in het vet weergegeven. De relatie die we uit de webpagina willen halen is dezelfde als in het vorige hoofdstuk en wordt hier opnieuw weergegeven in Illustratie 7.2.



Figuur 7.1: Variant van de webpagina van het country-code voorbeeld.

In de inleiding hebben we reeds vermeld dat deze wrapper class zowel van het head en tail, als van het open en close scheidingsteken gebruik maakt. Dit naast de linker en

rechter scheidingstekens die reeds van het begin van de partij waren. Een HOCLRT wrapper bestaat met andere woorden uit $2K+4$ scheidingstekens. De $\text{exec}_{\text{HOCLRT}}$ procedure krijgt bijgevolg, naast de pagina waaruit de procedure een relatie moet halen, een vector van $2K+4$ scheidingstekens als input, namelijk $\langle h, t, o, c, [\langle l_1, r_1\rangle, \ldots, \langle l_K, r_K\rangle]\rangle$.

Net zoals de wrappers van de HLRT en OCLRT wrapper class samengetrokken worden om de HOCLRT wrapper te bekomen, worden de $\text{exec}_{\text{HLRT}}$ en $\text{exec}_{\text{OCLR}}$ procedures samengetrokken om de $\text{exec}_{\text{HOCLRT}}$ procedure te bekomen. Deze procedure is weergegeven in Illustratie 7.3.

```
1:  <html><head><title>Some Country Codes</title></head><body>
2:  <h2>Some Country Codes</h2>
3:  <u><i><b>Europe</b></i></u><ul>
4:  <li><b>Belgium</b> <i>32</i></li>
5:  <li><b>Italy</b> <i>39</i></li>
6:  <li>*<b>Vatican City</b> <i>379</i></li>
7:  </ul><u><i><b>Africa</b></i></u><ul>
8:  <li><b>Congo</b> <i>243</i></li>
9:  <li><b>Egypt</b> <i>20</i></li>
10: </ul><p>(*) Assigned their own telephone code, but in general they make
                                    use of the telephone code of Italy.</p>
11: <hr><h2>Sources</h2><ul>
12: <li><b>Wikipedia: </b><a href="#">en.wikipedia.org</a></li>
13: <li><b>CountryCode: </b><a href="#">countrycode.org</a></li>
14: </ul></body></html>
```

ILLUSTRATIE 7.1: VEREENVOUDIGDE BRONCODE VAN DE WEBPAGINA IN FIGUUR 7.1.

$$R_{cc} = \left\{ \begin{array}{c} \langle\text{'Belgium','32'}\rangle \\ \langle\text{'Italy','39'}\rangle \\ \langle\text{'Vatican City','379'}\rangle \\ \langle\text{'Congo','243'}\rangle \\ \langle\text{'Egypt','20'}\rangle \end{array} \right\}$$

ILLUSTRATIE 7.2: DE RELATIE DIE WE UIT DE WEBPAGINA VAN FIGUUR 7.1 WILLEN HALEN.

```
1:  procedure exec_HOCLRT(wrapper ⟨h,t,o,c,[⟨l_1,r_1⟩,…,⟨l_K,r_K⟩]⟩, page P)
2:    search_move(h,P)
3:    m := 0
4:    while search(o,P) ≤ search(t,P)
5:      m := m + 1
6:      search_move(o,P)
7:      for each ⟨l_k,r_k⟩ ∈ [⟨l_1,r_1⟩,…,⟨l_K,r_K⟩]
8:        b_m,k := search(l_k,P) + |l_k|
9:        e_m,k := search_after_move(r_k,P,b_m,k)
10:     search_move(c,P)
11:   return relation {⟨⟨b_1,1,e_1,1⟩,…,⟨b_1,K,e_1,K⟩⟩ , … , ⟨⟨b_|R|,1,e_|R|,1⟩,…,⟨b_|R|,K,e_|R|,K⟩⟩}
```

ILLUSTRATIE 7.3: PROCEDURE $\text{EXEC}_{\text{HOCLRT}}$, DIE GEBRUIK MAAKT VAN EEN HOCLRT WRAPPER OM DE RELATIE VAN EEN PAGINA TE VERKRIJGEN.

De hulpfuncties die in de $exec_{HOCLRT}$ procedure van Illustratie 7.3 vermeld worden, maar niet beschreven zijn, zijn terug te vinden in Bijlage A.

We eindigen deze sectie met de beschrijving van de werking van de $exec_{HOCLRT}$ procedure. Op regel 2 wordt alles wat tot het hoofd van de pagina behoort overgeslagen. De while lus op regel 4 zoekt naar nieuwe tupels die voorkomen voor het begin van de voet van de pagina. Deze tupels worden aangeduid door het open scheidingsteken. Vervolgens wordt op regel 6 naar de positie van dit open scheidingsteken gegaan om daarna op regel 7 te zoeken naar alle linker en rechter scheidingstekens die de attribuutwaarden aanduiden. Op regel 10 wordt telkens naar de eindpositie van het tupel gegaan. Deze wordt aangeduid door het close scheidingsteken.

7.2 SCHEIDINGSTEKENS

7.2.1 KANDIDATEN VOOR DE SCHEIDINGSTEKENS

De kandidaten voor de l_k en r_k scheidingstekens kunnen op dezelfde manier gevonden worden als bij de LR wrapper class. Specifiek zijn er voor het scheidingsteken l_1 17 kandidaten voorhanden in ons country-code voorbeeld.

De kandidaten voor het head en tail scheidingsteken kunnen op dezelfde manier gevonden worden als bij de HLRT wrapper class. Dit levert in ons country code voorbeeld 8.001 kandidaten op voor het head scheidingsteken en 41.328 kandidaten voor het tail scheidingsteken.

De kandidaten voor het open en close scheidingsteken kunnen op dezelfde manier gevonden worden als bij de OCLR wrapper class. Dit levert in ons country-code voorbeeld 153 kandidaten op voor zowel het open als het close scheidingsteken.

7.2.2 RELATIE TUSSEN DE SCHEIDINGSTEKENS

Net zoals bij de LR wrapper class, zijn ook hier alle l_k en r_k scheidingstekens onafhankelijk van elkaar, met uitzondering van het scheidingsteken l_1. Dit komt omdat de scheidingstekens h, t, o, c en l_1 interageren, met andere woorden de keuze van één van deze vijf scheidingstekens hangt af van de keuze van de andere vier scheidingstekens. Dit komt omdat deze wrapper class en een combinatie is van de HLRT wrapper class, waar de scheidingstekens h, t en l_1 interageren, en de OCLR wrapper class, waar de scheidingstekens o, c en l_1 interageren. Merk op dat er ook nog andere directe relaties zijn tussen de scheidingstekens h, t, o, c en l_1. Deze vloeien voort uit de samenvoeging van de constraints van de HLRT en OCLR wrapper class en worden bijgevolg ook duidelijk uit de constraints van de HOCLRT wrapper class, die in de volgende sectie besproken worden.

Omdat de scheidingstekens h, t, o, c en l_1 interageren, moeten we alle mogelijke combinaties van de kandidaten van elk van deze vijf scheidingstekens in beschouwing nemen. We moeten met andere woorden het product nemen van de kandidaten van h, t, o, c en l_1. Dit levert 131.589.259.273.584 mogelijke kandidaat-combinaties op. Dit is het product van 8.001 kandidaten voor h (Sectie 6.2.1), 41.328 kandidaten voor t (Sectie 6.2.1), 153 kandidaten voor o (Sectie 6.2.1), 153 kandidaten voor c (Sectie 6.2.1) en 17 kandidaten voor l_1 (Sectie 6.2.1). Dit aantal kandidaat-combinaties is zeer hoog, zelf voor ons kleine voorbeeld. Het doorlopen van alle kandidaat-combinaties, voor een webpagina die beter aansluit bij de werkelijkheid, is bijgevolg niet praktisch haalbaar binnen afzienbare tijd. Daarom bieden heuristieken of andere technieken misschien een betere oplossing voor webpagina's zoals Figuur 7.1. Dit vereist echter meer onderzoek.

Merk op dat scheidingstekens mogen overlappen, dit om de onafhankelijkheid van de scheidingstekens zo veel mogelijk te bewaren. Linker en rechter scheidingstekens zijn nu namelijk onafhankelijk van elkaar. Het verbieden van overlap zou ervoor zorgen dat alle scheidingstekens afhankelijk worden van elkaar en introduceert extra constraints. Dit zou een negatieve impact hebben op de tijdsperformantie.

7.2.3 Validatie van de kandidaten

Nu we de kandidaten voor onze scheidingstekens gevonden hebben, kunnen we kijken naar de constraints waaraan deze kandidaten moeten voldoen, opdat de $\text{exec}_{\text{HOCLRT}}$ procedure de correcte relaties uit de pagina's kan gaan halen. De constraints voor de l_k en r_k scheidingstekens zijn dezelfde als bij de LR wrapper class (Sectie 4.2.4 en 4.2.5). We bespreken in deze sectie daarom enkel de constraints die betrekking hebben op de interagerende scheidingstekens h, t, o, c en l_1. Deze constraints zijn weergegeven in Illustratie 7.4 en worden hieronder toegelicht.

Een aantal constraints in Illustratie 7.4 zijn onderlijnd. Dit hebben we gedaan om aan te duiden welke constraints we reeds in de vorige hoofdstukken besproken hebben. Zo komen een aantal constraints van de HLRT en OCLR wrapper class opnieuw voor. Dit vanwege het feit dat de HOCLRT class een samenvoeging is van de HLRT en OCLR class. De constraints C^A en C^E komen respectievelijk overeen met de constraints C^A en C^D van de HLRT wrapper class. Voor meer details over deze constraints raden we de lezer aan Sectie 5.2.3 te raadplegen. De constraints C^B, C^F, C^H, C^I en C^J komen respectievelijk overeen met de constraints C^A, C^C, C^E, C^F en C^G van de OCLR wrapper class. Meer details over deze constraints zijn terug te vinden in Sectie 6.2.3.

De eerste nieuwe constraint die we tegenkomen is constraint C^C. Deze constraint legt een volgorde op aan de scheidingstekens die in het hoofd van elke pagina moeten voorkomen. Deze scheidingstekens, in volgorde, zijn h, o en l_1. Verder meldt deze

Constraint A: u_h moet een substring zijn van het hoofd van elke pagina

Constraint B: u_o moet een substring zijn van het hoofd van elke pagina

Constraint C: u_{l_1} moet een zuiver suffix zijn van de tekst die voorkomt na u_o, in de tekst die voorkomt na u_h, en dit in het hoofd van elke pagina

Constraint D: u_t mag niet voorkomen voor u_o, in de tekst die voorkomt na u_h, en dit in het hoofd van eender welke pagina

Constraint E: u_t moet een substring zijn van de voet van elke pagina

Constraint F: u_c moet een substring zijn van de voet van elke pagina

Constraint G: u_t moet voorkomen voor u_o, in de tekst die voorkomt na u_c, en dit in de voet van elke pagina

Constraint H: u_o moet een substring zijn van de tekst die zich tussen elke twee opeenvolgende tupels bevindt in elke pagina

Constraint I: u_c moet een substring zijn van de tekst die zich tussen elke twee opeenvolgende tupels bevindt in elke pagina

Constraint J: u_{l_1} moet een zuiver suffix zijn van de tekst die voorkomt na u_o, in de tekst die voorkomt na u_c, in de tekst tussen elke twee opeenvolgende tupels, en in dit in elke pagina

Constraint K: u_t mag niet voorkomen voor u_o, in de tekst die voorkomt na u_c, in de tekst tussen eender welke twee opeenvolgende tupels, en dit in eender welke pagina

ILLUSTRATIE 7.4: CONSTRAINTS WAARAAN DE SCHEIDINGSTEKENS *H*, *T*, *O*, *C* EN L_1 MOETEN VOLDOEN.

constraint dat het scheidingsteken l_1 een zuiver suffix van elk paginahoofd moet zijn. De constraint zorgt, in de $exec_{HOCLRT}$ procedure, voor de goede werking van regels 2, 4 en 7 in Illustratie 7.3. Indien deze constraint overtreden wordt, dan worden er tupels worden overgeslagen. Deze constraint vinden we terug op regel 18 in het $learn_{HOCLRT}$ algoritme (Illustratie 7.5).

De constraint C^D vult constraint C^C aan door te stellen dat het tail scheidingsteken *t* niet mag voorkomen na het head scheidingsteken *h* en voor het open scheidingsteken *o*. De volgorde *h*, *t* en *o* is met andere woorden verboden. Hierdoor waakt deze constraint erover dat de $exec_{HOCLRT}$ procedure ten minste één tupel uit de pagina kan halen. Indien deze constraint overtreden wordt, dan zal de $exec_{HOCLRT}$ procedure niet in de while lus gaan op regel 4 in Illustratie 7.3. Deze constraint vinden we terug op regel 20 in het $learn_{HOCLRT}$ algoritme (Illustratie 7.5).

Om te voorkomen dat de $exec_{HOCLRT}$ procedure tupels uit de voet van de pagina probeert te halen, is er constraint C^G toegevoegd. Deze constraint stelt namelijk dat het tail scheidingsteken moet voorkomen na het close scheidingsteken en voor de mogelijke open scheidingstekens die kunnen voorkomen in de voet van de pagina. Indien deze constraint overtreden wordt, dan zal de while lus, op regel 4 in Illustratie 7.3, te veel itereren. Deze constraint vinden we terug op regel 27 in het $learn_{HLRT}$ algoritme (Illustratie 7.5).

Wat constraint C^D doet voor het hoofd van een pagina doet constraint C^K voor de tekst tussen tupels. Constraint C^K stelt namelijk dat het tail scheidingsteken *t* niet mag voorkomen tussen het close scheidingsteken *c* en het open scheidingsteken *o*, in de tekst tussen tupels. De volgorde *c*, *t* en *o* is met andere woorden verboden. Hierdoor waakt de constraint erover dat het tail scheidingsteken niet te vroeg voorkomt in de pagina, oftewel tussen de tupels. Indien deze constraint overtreden wordt dan zal de $exec_{HOCLRT}$ procedure te vroeg stoppen met de while lus op regel 4 in Illustratie 7.3, en bijgevolg niet alle tupels uit de pagina halen. Deze constraint vinden we terug op regel 36 in het $learn_{HOCLRT}$ algoritme (Illustratie 7.5).

In de paper Wrapper Induction for Information Extraction" van Kushmerick et al. [12] en de doctoraatsthesis "Wrapper Induction for Information Extraction" van Kushmerick [13] zijn de constraints C^D en C^K foutief. In dit hoofdstuk hebben we echter de correcte constraints gehanteerd. Lezers die geïnteresseerd zijn in de details van deze foutieve constraints, alsook waarom deze foutief zijn, kunnen hiervoor terecht in Bijlage B.

7.3 HET $LEARN_{HOCLRT}$ ALGORITME

Nu we besproken hebben hoe de $exec_{HOCLRT}$ procedure gebruik maakt van een wrapper en hoe we kandidaten voor de nieuwe scheidingstekens kunnen vinden en valideren, kunnen we het algoritme bespreken dat verantwoordelijk is voor het automatisch leren van een HOCLRT wrapper. Dit algoritme, genaamd $learn_{HOCLRT}$, is weergegeven in Illustratie 7.5.

Het $learn_{HOCLRT}$ algoritme krijgt als input een verzameling van voorbeeldpagina's en hun bijhorende relaties, genaamd $\mathcal{E}$. Op regel 2 maakt dit algoritme gebruik van het $learn_{LR}$ algoritme om de l_k en r_k scheidingstekens te vinden, met uitzondering van l_1. De vijf for lussen op regels 3 tot en met 7 dienen om alle mogelijke combinaties van de kandidaten voor de scheidingstekens *h*, *t*, *o*, *c* en l_1 te verkrijgen. Hiervoor wordt gebruik gemaakt van de $cands_h$ & $cands_t$, $cands_{o,c}$ en de $cands_l$ functies. Deze zijn respectievelijk in Sectie 5.3, Sectie 6.3 en Sectie 4.3 besproken en worden om het overzicht te bewaren hier niet opnieuw vermeld. Op regel 8 wordt de $valid_{l_1,h,t,o,c}$ functie opgeroepen om de combinatie

van de scheidingstekens h, t, o, c en l_1 te valideren. Zodra een geldige combinatie gevonden wordt stopt het algoritme met zoeken naar scheidingstekens (regel 10). De output van het $\text{learn}_{\text{HOCLRT}}$ algoritme is een wrapper $\langle h, t, o, c, [\langle l_1, r_1\rangle, \ldots, \langle l_K, r_K\rangle]\rangle$ van $2K+4$ scheidingstekens.

```
1:  procedure learn_HOCLRT(examples E)
2:    ⟨·, r_1, …, l_K, r_K⟩ := learn_LR(E)
3:    for each u_{l_1} ∈ cands_l(1, E)
4:      for each u_o ∈ cands_{o,c}(E)
5:        for each u_c ∈ cands_{o,c}(E)
6:          for each u_h ∈ cands_h(E)
7:            for each u_t ∈ cands_t(E)
8:              if valid_{l_1,h,t,o,c}(u_{l_1} u_h u_t u_o u_c, E) then
9:                l_1 := u_{l_1}, o := u_o, c := u_c, h := u_h, t := u_t
10:               break all loops
11:   return HOCLRT wrapper ⟨h, t, o, c, [⟨l_1, r_1⟩, …, ⟨l_K, r_K⟩]⟩

12: procedure valid_{l_1,h,t,o,c}(candidates u_{l_1} u_h u_t u_o u_c, examples E)
13:   for each s ∈ heads(E)
14:     if u_h is not a substring of s then
15:       return FALSE
16:     if u_o is not a substring of s then
17:       return FALSE
18:     if u_{l_1} is not a proper suffix of scan(scan(s, u_h), u_o) then
19:       return FALSE
20:     if u_t occur before u_o in scan(s, u_h) then
21:       return FALSE
22:   for each s ∈ tails(E)
23:     if u_t is not a substring of s then
24:       return FALSE
25:     if u_c is not a substring of s then
26:       return FALSE
27:     if u_o occur before u_t in scan(s, u_c) then
28:       return FALSE
29:   for each s ∈ seps(K, E)
30:     if u_o is not a substring of s then
31:       return FALSE
32:     if u_c is not a substring of s then
33:       return FALSE
34:     if u_{l_1} is not a proper suffix of scan(scan(s, u_c), u_o) then
35:       return FALSE
36:     if u_t occur before u_o in scan(s, u_c) then
37:       return FALSE
38:   return TRUE
```

ILLUSTRATIE 7.5: HET $\text{LEARN}_{\text{HOCLRT}}$ ALGORITME, DAT AUTOMATISCH EEN HOCLRT WRAPPER KAN LEREN UIT EEN VERZAMELING VAN VOORBEELDPAGINA'S EN HUN BIJHORENDE RELATIES.

De functie $\text{valid}_{l_1,h,t,o,c}(u_{l_1}\ u_h\ u_t\ u_o\ u_c,\mathcal{E})$ controleert of de combinatie van de kandidaten u_{l_1}, u_h, u_t, u_o en u_c geldig is voor de scheidingstekens l_1, h, t, o en c. De constraints waaraan u_{l_1}, u_h, u_t, u_o en u_c moeten voldoen werden reeds besproken in Sectie 7.2.3.

De hulpfuncties die in Illustratie 7.5 vermeld worden, maar niet beschreven zijn, zijn terug te vinden in Bijlage A.

Een voorbeeld van een geldige HOCLRT wrapper voor het country-code voorbeeld van dit hoofdstuk (Figuur 7.1) is volgende vector van $2K+4$ scheidingstekens:

⟨'</u>
','
⇓
<p>','
','
',[⟨'',''⟩,⟨'',''⟩]⟩

7.4 Relatie tussen de HLRT en HOCLRT wrapper class

Tussen de HLRT en HOCLRT wrapper class bestaat een interessante relatie die we kunnen uitdrukken als onderstaande stelling.

Stelling 7.1: HLRT wrapper impliceert HOCLRT wrapper

> Gegeven een geldige HLRT wrapper voor een verzameling van voorbeeldpagina's en hun bijhorende relaties, dan bestaat er eveneens een geldige HOCLRT wrapper voor dezelfde verzameling van voorbeeldpagina's en hun bijhorende relaties, bestaande uit enkel scheidingstekens van de HLRT wrapper.

Deze stelling is correct en dit zullen we aantonen aan de hand van de volgende abstracte HLRT wrapper $\langle h, t, [\langle l_1, r_1\rangle, \ldots, \langle l_K, r_K\rangle]\rangle$. De HOCLRT wrapper die we hieruit opbouwen is $\langle h, t, l_1, r_K, [\langle l_1, r_1\rangle, \ldots, \langle l_K, r_K\rangle]\rangle$. De HOCLRT wrapper neemt met andere woorden elk scheidingsteken van de HLRT wrapper over en gebruikt de scheidingstekens l_1 en r_K eveneens voor respectievelijk het open en close scheidingsteken. Deze HOCLRT wrapper is geldig omdat overlap tussen scheidingstekens is toegestaan.

In tegenstelling tot wat men misschien zou denken, bestaat deze relatie niet tussen de OCLR en HOCLRT wrapper class. We zullen dit illustreren aan de hand van de volgende abstracte OCLR wrapper $\langle o, c, [\langle l_1, r_1\rangle, \ldots, \langle l_K, r_K\rangle]\rangle$. We zijn hier verplicht om het open en close scheidingsteken als respectievelijk het head en tail scheidingsteken te gebruiken. We kunnen l_1 namelijk niet als head scheidingsteken gebruiken, omdat l_1 ook voor een open scheidingsteken kan voorkomen. We kunnen r_K niet als tail scheidingsteken gebruiken, omdat r_K ook in de tekst tussen tupels, tussen het close en open scheidingsteken kan voorkomen. De HOCLRT wrapper die we bijgevolg bekomen is $\langle o, c, o, c, [\langle l_1, r_1\rangle, \ldots, \langle l_K, r_K\rangle]\rangle$. Het probleem met deze wrapper is dat overlap is toegestaan. Na het vinden van een close scheidingsteken, zoeken we immers naar het eerst voorkomende open en tail scheidingsteken. Het tail scheidingsteken valt samen met het close scheidingsteken, terwijl de startpositie van het open scheidingsteken in het

beste geval gelijk is aan de startpositie van het close scheidingsteken. Dit is echter niet altijd het geval voor het open scheidingsteken, waardoor het tail scheidingsteken voor het open scheidingsteken voorkomt in de tekst tussen tupels en bijgevolg zal de $\text{exec}_{\text{HOCLRT}}$ procedure vroegtijdig stoppen. De HOCLRT wrapper $\langle o, c, o, c, [\langle l_1, r_1\rangle, \ldots, \langle l_K, r_K\rangle]\rangle$ is daarom niet geldig.

Stelling 7.1 van hierboven kan niet in omgekeerde richting gebruikt worden. Het niet bestaan van een geldige HLRT wrapper impliceert met andere woorden niet dat er geen geldige HOCLRT wrapper bestaat. Dit hebben we reeds besproken in de inleiding van dit hoofdstuk.

7.5 Conclusie

In dit hoofdstuk hebben we de HOCLRT wrapper class besproken. Deze wrapper class is een combinatie van de HLRT en OCLR wrapper class en maakt bijgevolg naast linker en rechter scheidingstekens ook gebruik van een head (*h*), tail (*t*), open (*o*) en close (*c*) scheidingsteken voor het vinden van de attribuutwaarden van een relatie op een pagina. Het head en tail scheidingsteken duiden respectievelijk het einde van het hoofd en het begin van de voet van een pagina aan. Het open en close scheidingsteken duiden respectievelijk het begin en het einde van elk tupel op de pagina aan. De $\text{exec}_{\text{HOCLRT}}$ procedure houdt met al deze scheidingstekens rekening wanneer hij de relatie uit een pagina probeert te halen. De HOCLRT wrapper is bijgevolg een vector van $2K+4$ scheidingstekens, namelijk $\langle h, t, o, c, [\langle l_1, r_1\rangle, \ldots, \langle l_K, r_K\rangle]\rangle$.

Verder hebben we gezien dat de scheidingstekens l_1, *h*, *t*, *o* en *c* afhankelijk zijn van elkaar, waardoor we alle combinaties van de kandidaten van deze vijf scheidingstekens in beschouwing moeten nemen. We hebben ook gezien dat dit aantal hoog kan oplopen, zelfs voor een klein voorbeeld.

Daarna hebben we het $\text{learn}_{\text{HOCLRT}}$ algoritme besproken dat automatisch een HOCLRT wrapper kan leren uit een verzameling van voorbeeldpagina's en hun bijhorende relaties.

Tot slot hebben we de relatie tussen de HLRT en HOCLRT wrapper class besproken. Het bestaan van een geldige HLRT wrapper betekent namelijk dat er ook een geldige HOCLRT wrapper class bestaat.

8. De UOCLR Wrapper Class

Voor we begonnen met het beschrijven van de verschillende wrapper classes hadden we de aanname gemaakt dat webpagina's over een uniforme opmaak beschikken. Dit is echter niet in volledige overeenstemming met de werkelijkheid. Het kan namelijk voorkomen dat attribuutwaarden ontbreken, scheidingstekens ontbreken of omgewisseld worden, etc. In dit hoofdstuk beschrijven we daarom een nieuwe wrapper class die met een aantal van deze problemen overweg kan.

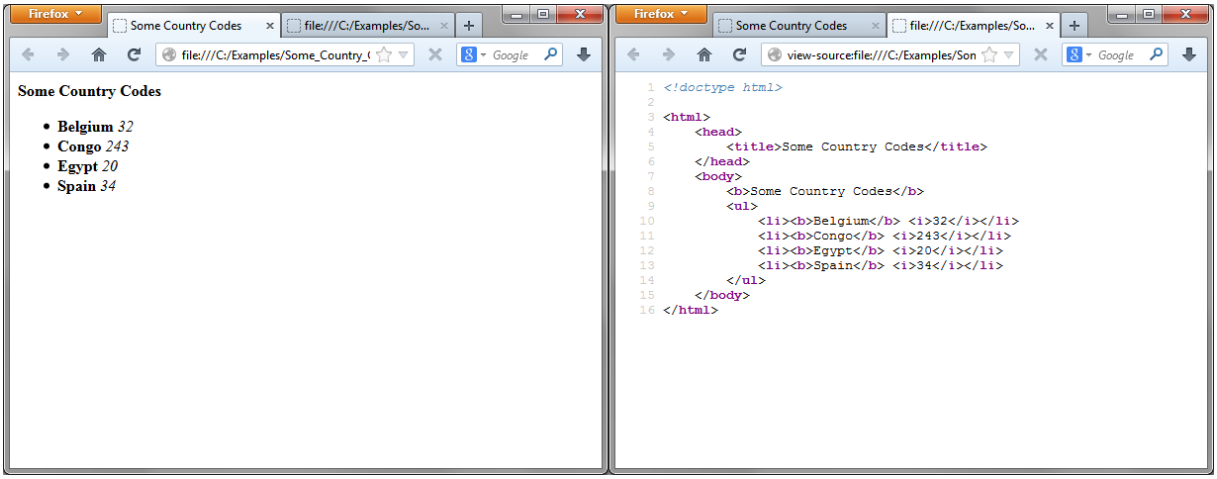
In de eerste sectie van dit hoofdstuk gaan we dieper in op een aantal problemen die een non-uniforme opmaak met zich meebrengt. Daarna gaan we verder met het bespreken van de nieuwe wrapper class, genaamd UOCLR. Voor de bespreking van deze wrapper class volgen we hetzelfde verloop als in de voorgaande hoofdstukken. We beginnen namelijk met het bespreken van een exec procedure voor de UOCLR wrapper. Vervolgens gaan we verder met de relatie tussen de scheidingstekens. Tot slot eindigen we met een learn algoritme dat automatisch een UOCLR wrapper kan leren.

8.1 Problemen

In deze sectie bekijken we een aantal problemen die kunnen voorkomen op webpagina's. Meer bepaald kijken we naar ontbrekende scheidingstekens, een inconsistente volgorde van attributen en inconsistent gebruik van scheidingstekens. We illustreren elk probleem aan de hand van een voorbeeld. We maken tijdens deze voorbeelden gebruik van de OCLR wrapper class, omdat onze nieuwe wrapper class, genaamd UOCLR, gebaseerd is op de OCLR wrapper class. De $\text{exec}_{\text{OCLR}}$ procedure wordt voor de eenvoud opnieuw weergegeven in Illustratie 8.1. De problemen die we in deze sectie aanhalen hebben echter dezelfde negatieve impact op de andere drie wrapper classes.

```
1:  procedure exec_OCLR(wrapper ⟨o,c,[⟨l_1,r_1⟩,…,⟨l_K,r_K⟩]⟩, page P)
2:    m := 0
3:    while search(o,P) ≠ EOF
4:      m := m + 1
5:      search_move(o,P)
6:      for each ⟨l_k,r_k⟩ ∈ [⟨l_1,r_1⟩,…,⟨l_K,r_K⟩]
7:        b_m,k := search(l_k,P) + |l_k|
8:        e_m,k := search_after_move(r_k,P,b_m,k)
9:      search_move(c,P)
10:   return relation {⟨⟨b_1,1,e_1,1⟩,…,⟨b_1,K,e_1,K⟩⟩ , … , ⟨⟨b_|R|,1,e_|R|,1⟩,…,⟨b_|R|,K,e_|R|,K⟩⟩}
```

Illustratie 8.1: Procedure $\text{exec}_{\text{OCLR}}$, die gebruik maakt van een OCLR wrapper om de relatie van een pagina te verkrijgen.



FIGUUR 8.1: WEBPAGINA MET UNIFORME OPMAAK.

```
1: <html><head><title>Some Country Codes</title></head><body>
2: <b>Some Country Codes</b><ul>
3: <li><b>Belgium</b> <i>32</i></li>
4: <li><b>Congo</b> <i>243</i></li>
5: <li><b>Egypt</b> <i>20</i></li>
6: <li><b>Spain</b> <i>34</i></li>
7: </ul></body></html>
```

ILLUSTRATIE 8.2: VEREENVOUDIGDE BRONCODE VAN DE WEBPAGINA IN FIGUUR 8.1.

$$R_{cc} = \left\{ \begin{array}{l} \langle \texttt{'Belgium','32'} \rangle \\ \langle \texttt{'Congo','243'} \rangle \\ \langle \texttt{'Egypt','20'} \rangle \\ \langle \texttt{'Spain','34'} \rangle \end{array} \right\}$$

ILLUSTRATIE 8.3: RELATIE HORENDE BIJ DE WEBPAGINA VAN FIGUUR 8.1.

In Figuur 8.1 zien we een webpagina met uniforme opmaak. De vereenvoudigde broncode van deze webpagina wordt weergegeven in Illustratie 8.2. De relatie die bij deze webpagina hoort, is weergegeven in Illustratie 8.3. Alle voorbeelden die we in dit hoofdstuk beschouwen zijn varianten, met non-uniforme opmaak, van deze webpagina.

8.1.1 ONTBREKENDE SCHEIDINGSTEKENS

Het eerste probleem dat we bespreken betreft ontbrekende scheidingstekens. In Illustratie 8.4 zien we de broncode van een webpagina waarin zowel de scheidingstekens als de attribuutwaarde ontbreken voor het telefooncode attribuut van het eerste tupel (regel 3) en het land attribuut van het derde tupel (regel 5). Wanneer we de $\text{exec}_{\text{OCLR}}$ procedure (Illustratie 8.1) op deze webpagina uitvoeren, met als input de OCLR wrapper $\langle$`'<li>','</li>',[`$\langle$`'<b>','</b>'`$\rangle$`,`$\langle$`'<i>','</i>'`$\rangle$`]`$\rangle$, dan bekomen we als output de relatie die wordt weergegeven Illustratie 8.5. In deze relatie ontbreken het tweede en derde tupel en is de telefooncode attribuutwaarde van het eerste tupel foutief. Dit komt doordat de $\text{exec}_{\text{OCLR}}$ procedure op regel 3 eerst op zoek gaat naar een open

```
1: <html><head><title>Some Country Codes</title></head><body>
2: <b>Some Country Codes</b><ul>
3: <li><b>Belgium</b></li>
4: <li><b>Congo</b> <i>243</i></li>
5: <li><i>20</i></li>
6: <li><b>Spain</b> <i>34</i></li>
7: </ul></body></html>
```

ILLUSTRATIE 8.4: BRONCODE VAN EEN WEBPAGINA WAARIN ATTRIBUUTWAARDEN EN HUN BIJHORENDE SCHEIDINGSTEKENS ONTBREKEN.

$$R_{cc} = \begin{Bmatrix} \langle \text{'Belgium','243'} \rangle \\ \langle \text{'Spain','34'} \rangle \end{Bmatrix}$$

ILLUSTRATIE 8.5: RELATIE DIE EEN OCLR WRAPPER UIT DE WEBPAGINA VAN ILLUSTRATIE 8.4 HAALT.

```
1: <html><head><title>Some Country Codes</title></head><body>
2: <b>Some Country Codes</b><ul>
3: <li><b>Belgium</b> <i></i></li>
4: <li><b>Congo</b> <i>243</i></li>
5: <li><b></b> <i>20</i></li>
6: <li><b>Spain</b> <i>34</i></li>
7: </ul></body></html>
```

ILLUSTRATIE 8.6: BRONCODE VAN EEN WEBPAGINA WAARIN ATTRIBUUTWAARDEN ONTBREKEN.

$$R_{cc} = \begin{Bmatrix} \langle \text{'Belgium',''} \rangle \\ \langle \text{'Congo','243'} \rangle \\ \langle \text{'','20'} \rangle \\ \langle \text{'Spain','34'} \rangle \end{Bmatrix}$$

ILLUSTRATIE 8.7: RELATIE DIE EEN OCLR WRAPPER UIT DE WEBPAGINA VAN ILLUSTRATIE 8.6 HAALT.

scheidingsteken, daarna op regel 6 alle linker- en rechterscheidingstekens in volgorde doorloopt, voor hij het tupel afsluit door op regel 9 naar het close scheidingsteken te zoeken. In de for lus op regel 6 wordt met andere woorden in volgorde gezocht naar de eerst voorkomende "" tag, de eerst voorkomende "" tag, de eerst voorkomende "" tag en de eerst voorkomende "" tag vooraleer naar het einde van een tupel wordt gezocht. Dit heeft tot gevolg dat de $\text{exec}_{\text{OCLR}}$ procedure bepaalde waarden uit verkeerde tupels gaat halen omdat daar het gewenste scheidingsteken wel aanwezig is. Merk op dat we een gelijkaardige redenering kunnen volgen wanneer enkel de scheidingstekens, en niet de attribuutwaarden, ontbreken of wanneer andere nieuwe scheidingstekens gebruikt worden voor bepaalde attribuutwaarden.

De situatie waarbij enkel de attribuutwaarden ontbreken en de scheidingstekens nog allemaal aanwezig zijn, vormt geen probleem. In Illustratie 8.6 wordt dezelfde broncode als in Illustratie 8.4 weergegeven, met als enige verschil dat de scheidingstekens van de ontbrekende attribuutwaarden ditmaal wel aanwezig zijn. Wanneer we de $\text{exec}_{\text{OCLR}}$

procedure (Illustratie 8.1) op deze webpagina uitvoeren, met als input de OCLR wrapper ⟨'
','
',[⟨'',''⟩,⟨'',''⟩]⟩, dan bekomen we als output de relatie die wordt weergegeven in Illustratie 8.7. In deze relatie zijn de ontbrekende attribuutwaarden leeg, net zoals de inhoud van de tags op de webpagina. De $exec_{OCLR}$ procedure moet op regel 6 met andere woorden niet in een volgend tupel gaan zoeken naar de gewenste scheidingstekens, maar vind deze telkens in het correcte tupel op de webpagina.

Indien we weten waar een tupel begint en eindigt op een pagina, dan kunnen we het probleem van ontbrekende scheidingstekens oplossen door enkel tussen dit tupel begin en tupel einde te zoeken naar de scheidingstekens van de attributen. Indien er dan scheidingstekens van een attribuut ontbreken, gaan we bijgevolg niet meer buiten de grenzen van het tupel op de pagina.

8.1.2 Inconsistente volgorde van attributen

Het tweede probleem dat we bespreken betreft de inconsistente volgorde van attributen. Illustratie 8.8 toont de broncode van een webpagina waarbij deze situatie zich voordoet. Het telefooncode attribuut wordt namelijk voor het land attribuut vermeld in het tweede en derde tupel van deze webpagina (regel 4 en 5). Wanneer we de $exec_{OCLR}$ procedure (Illustratie 8.1) op deze webpagina uitvoeren, met als input de OCLR wrapper ⟨'
','
',[⟨'',''⟩,⟨'',''⟩]⟩, dan bekomen we als output de relatie die wordt weergegeven in Illustratie 8.9. De reden voor dit resultaat is dezelfde als degene die we hebben gezien bij de ontbrekende scheidingstekens in de vorige sectie. De $exec_{OCLR}$ procedure zoekt namelijk in volgorde naar de eerst voorkomende waarde van elk scheidingsteken.

```
1: <html><head><title>Some Country Codes</title></head><body>
2: <b>Some Country Codes</b><ul>
3: <li><b>Belgium</b> <i>32</i></li>
4: <li><i>243</i> <b>Congo</b></li>
5: <li><i>20</i> <b>Egypt</b></li>
6: <li><b>Spain</b> <i>34</i></li>
7: </ul></body></html>
```

Illustratie 8.8: Broncode van een webpagina waarin de attributen een inconsistente volgorde hebben.

$$R_{cc} = \left\{ \begin{array}{l} \langle \text{'Belgium','32'} \rangle \\ \langle \text{'Congo','20'} \rangle \\ \langle \text{'Spain','34'} \rangle \end{array} \right\}$$

Illustratie 8.9: Relatie horende bij de webpagina van Illustratie 8.8.

Een mogelijke oplossing voor dit probleem houdt in dat we niet meer in een vaste volgorde gaan zoeken naar de scheidingstekens van de attributen. Dit houdt in dat we ook niet meer in volgorde gaan zoeken naar de attributen. We gaan met andere woorden kijken welke attribuut scheidingstekens eerst voorkomen op de webpagina. Doordat we geen volgorde meer opleggen aan de scheidingstekens van de attributen, moeten deze ook niet meer in een vaste volgorde voorkomen in elk tupel.

8.1.3 Inconsistent gebruik van scheidingstekens

Het derde en laatste probleem dat we bespreken heeft betrekking op het inconsistent gebruik van scheidingstekens. Een voorbeeld van dit probleem is terug te vinden in Illustratie 8.10. Deze illustratie toont de broncode van een webpagina waarin landen standaard vetgedrukt zijn en telefooncodes standaard cursief worden weergegeven. In het tweede tupel (regel 4) is deze conventie echter omgekeerd, de land attribuutwaarde is namelijk cursief terwijl de telefooncode attribuutwaarde vetgedrukt is. Wanneer we de $\text{exec}_{\text{OCLR}}$ procedure (Illustratie 8.1) op deze webpagina uitvoeren, met als input de OCLR wrapper ⟨'
','
',[⟨'',''⟩,⟨'',''⟩]⟩, dan bekomen we als output de relatie die wordt weergegeven in Illustratie 8.11. In deze relatie beschikking we slechts over drie tupels en bevat het tweede tupel twee telefooncodes. De reden voor dit resultaat is dezelfde als degene die we hebben gezien bij de ontbrekende scheidingstekens (Sectie 8.1.1). De $\text{exec}_{\text{OCLR}}$ procedure zoekt namelijk in volgorde naar de eerst voorkomende waarde van elk scheidingsteken.

```
1: <html><head><title>Some Country Codes</title></head><body>
2: <b>Some Country Codes</b><ul>
3: <li><b>Belgium</b> <i>32</i></li>
4: <li><i>Congo</i> <b>243</b></li>
5: <li><b>Egypt</b> <i>20</i></li>
6: <li><b>Spain</b> <i>34</i></li>
7: </ul></body></html>
```

Illustratie 8.10: Broncode van een webpagina waarin scheidingstekens inconsistent gebruikt worden.

$$R_{cc} = \left\{ \begin{array}{c} \langle\text{'Belgium','32'}\rangle \\ \langle\text{'243','20'}\rangle \\ \langle\text{'Spain','34'}\rangle \end{array} \right\}$$

Illustratie 8.11: Relatie horende bij de webpagina van Illustratie 8.10.

Problemen in verband met inconsistent gebruik van scheidingstekens kunnen niet eenvoudig opgelost worden. Een manier om dit probleem op te lossen is door aan elk attribuut een type toe te kennen, bijvoorbeeld elke land attribuutwaarde moet een string zijn en elke telefooncode attribuutwaarde moet een getal zijn. We hebben met andere

woorden semantiek nodig. Het gebruiken van semantiek bij informatie extractie valt echter buiten het bereik van deze masterproef, zoals reeds aangehaald in Hoofdstuk 1.

8.2 De $exec_{UOCLR}$ procedure

In de vorige sectie hebben we een aantal problemen besproken die kunnen voorkomen op een webpagina. Daarnaast hebben we voor een sommige van deze problemen kort een oplossing aangehaald. In deze sectie bespreken we een nieuwe wrapper class die met twee van deze problemen kan omgaan, namelijk het ontbreken van scheidingstekens en een inconsistente volgorde van attributen. Deze wrapper class draagt de naam UOCLR (Unique-Open-Close-Left-Right) en is gebaseerd op de OCLR wrapper class. De reden voor deze naam wordt duidelijk wanneer we de relatie tussen de scheidingstekens bespreken (Sectie 8.3.2).

In Illustratie 8.12 zien we de broncode van een webpagina met een non-uniforme opmaak. In deze webpagina ontbreken attribuutwaarden (inclusief scheidingstekens) en zijn attributen van plaats verwisseld. Deze webpagina kan tot stand zijn gekomen doordat de ontwikkelaar bijvoorbeeld minder oplettend was tijdens het maken van de webpagina en omdat sommige data misschien niet voorhanden was. Wanneer we de $exec_{OCLR}$ procedure (Illustratie 8.1) op deze webpagina uitvoeren, met als input de OCLR wrapper ⟨'
','
',[⟨'',''⟩,⟨'',''⟩]⟩, dan krijgen we als output de relatie die wordt weergegeven in Illustratie 8.13. Deze relatie bevat slechts één correct tupel, namelijk het tupel met "Spain" als attribuutwaarde voor het land attribuut. De correcte relatie, degene die we uit deze webpagina willen halen, wordt weergegeven in Illustratie 8.14.

Om de relatie van Illustratie 8.14 uit de webpagina van Illustratie 8.12 te halen moeten we de $exec_{OCLR}$ procedure aanpassen. Het resultaat is de $exec_{UOCLR}$ procedure van Illustratie 8.15. Deze procedure krijgt, naast de pagina waaruit een relatie gehaald moet

```
1: <html><head><title>Some Country Codes</title></head><body>
2: <b>Some Country Codes</b><ul>
3: <li><b>Belgium</b></li>
4: <li><i>243</i> <b>Congo</b></li>
5: <li><i>20</i></li>
6: <li><b>Spain</b> <i>34</i></li>
7: </ul></body></html>
```

Illustratie 8.12: Vereenvoudigde broncode van een webpagina met non-uniforma opmaak.

$$R_{cc} = \begin{Bmatrix} \langle \text{'Belgium','243'} \rangle \\ \langle \text{Spain,'34'} \rangle \end{Bmatrix}$$

Illustratie 8.13: Relatie die een OCLR wrapper uit de webpagina van Illustratie 8.12 haalt.

$$R_{cc} = \begin{Bmatrix} \langle \text{'Belgium','')} \\ \langle \text{'Congo','243'} \rangle \\ \langle \text{'','20'} \rangle \\ \langle \text{'Spain','34'} \rangle \end{Bmatrix}$$

ILLUSTRATIE 8.14: RELATIE DIE WE UIT DE WEBPGINA VAN ILLUSTRATIE 8.12 WILLEN HALEN.

```
1:  procedure exec_UOCLR(wrapper ⟨o,c,[⟨l_1,r_1⟩,…,⟨l_K,r_K⟩]⟩, page P)
2:    m ≔ 0
3:    while search(o,P) ≠ EOF
4:      m ≔ m + 1
5:      file pointer position ≔ search(o,P) + |o|
6:      pos_c ≔ search(c,P)
7:      for each ⟨l_k,r_k⟩ ∈ {⟨l_1,r_1⟩,…,⟨l_K,r_K⟩}
8:        b_m,k ≔ search_before(l_k,P,pos_c) + |l_k|
9:        e_m,k ≔ search_after_before(r_k,P,b_m,k,pos_c)
10:     search_move(c,P)
11:   return relation {⟨⟨b_1,1,e_1,1⟩,…,⟨b_1,K,e_1,K⟩⟩ , … , ⟨⟨b_|R|,1,e_|R|,1⟩,…,⟨b_|R|,K,e_|R|,K⟩⟩}
```

ILLUSTRATIE 8.15: PROCEDURE EXEC$_{UOCLR}$, DIE GEBRUIKT MAAKT VAN EEN UOCLR WRAPPER OM DE RELATIE VAN EEN PAGINA TE VERKRIJGEN.

worden, een UOCLR wrapper mee. Deze UOCLR wrapper is net zoals de OCLR wrapper een vector van $2K+2$ scheidingstekens, namelijk $\langle o, c, [\langle l_1, r_1 \rangle, \ldots, \langle l_K, r_K \rangle] \rangle$. De exec$_{UOCLR}$ procedure verschilt op drie punten van de exec$_{OCLR}$ procedure, die hieronder toegelicht worden.

De hulpfuncties die in de exec$_{UOCLR}$ procedure van Illustratie 8.15 vermeld worden, maar niet beschreven zijn, zijn terug te vinden in Bijlage A.

Het eerste verschil vinden we terug op regel 6. Op deze regel zoeken we de positie van het eerst voorkomende close scheidingsteken. Deze positie zal gebruikt worden als grens voor het zoeken naar een linker scheidingsteken op regel 8 en een rechter scheidingsteken op regel 9.

Het tweede verschil bestaat erin dat we op regel 7 niet meer in volgorde door de paren van scheidingstekens van de attributen lopen, maar dat we kijken welke attribuut scheidingstekens we eerst tegenkomen. Op deze manier kunnen attributen in een tupel in eender welke volgorde voorkomen.

Het derde en laatste verschil komt tot stand doordat we op regel 8 en 9 niet meer naar de positie van het linker/rechter scheidingsteken gaan. We blijven met andere woorden staan op de positie van het open scheidingsteken. We gebruiken de positie van het linker scheidingsteken, dat we bekomen in regel 8, wel nog als startpositie voor het zoeken naar het rechter scheidingsteken op regel 9.

Indien we de $exec_{UOCLR}$ procedure op deze webpagina uitvoeren met als input de OCLR wrapper ⟨'
','
',[⟨'',''⟩,⟨'',''⟩]⟩ die we hierboven reeds gebruikt hebben, dan krijgen we als output de gewenste relatie, namelijk Illustratie 8.14. Dit komt omdat deze OCLR wrapper ook een geldige UOCLR wrapper is.

8.3 SCHEIDINGSTEKENS

8.3.1 KANDIDATEN VOOR DE SCHEIDINGSTEKENS

Kandidaten voor de l_k en r_k scheidingstekens kunnen op dezelfde manier gevonden worden als bij alle voorgaande wrapper classes, namelijk door gebruik te maken van de $cands_l$ en $cands_r$ functies.

Kandidaten voor het open en close scheidingsteken kunnen op dezelfde manier gevonden worden als bij de OCLR wrapper class, namelijk door gebruik te maken van de $cands_{o,c}$ functie.

```
1: <html><head><title>Some Country Codes</title></head><body>
2: <b>Some Country Codes</b><ul>
3: <li><b>Belgium</b> <i>32</i></li>
4: <li><b>Congo</b> <i>243</i></li>
5: <li><b>Egypt</b> <i>20</i></li>
6: <li><b>Spain</b> <i>34</i></li>
7: </ul></body></html>
```

ILLUSTRATIE 8.16: VEREENVOUDIGDE BRONCODE VAN DE WEBPAGINA IN FIGUUR 8.1.

Scheidingsteken	Aantal kandidaten
o	153
c	153
l_1	17
r_1	8
l_2	8
r_2	17

ILLUSTRATIE 8.17: AANTAL KANDIDATEN VOOR ELK SCHEIDINGSTEKEN VAN DE UOCLR WRAPPER.

De UOCLR wrapper class mag dan wel met ontbrekende scheidingstekens en een inconsistente volgorde van attributen kunnen omgaan, voor het leren van de scheidingstekens heeft de wrapper class nog altijd een uniforme webpagina nodig. De wrapper class moet namelijk leren wat de "correcte" situatie is vooraleer hij kan omgaan met "incorrecte" situaties. Als voorbeeld van een uniforme webpagina maken we bijgevolg gebruik van de webpagina van Figuur 8.1. De broncode van deze webpagina

wordt voor de eenvoud opnieuw weergegeven in Illustratie 8.16. Het aantal kandidaten voor elk scheidingstekens voor deze webpagina wordt weergegeven in Illustratie 8.17.

8.3.2 Relatie tussen de scheidingstekens

In tegenstelling tot bij de OCLR wrapper class interageren in de UOCLR wrapper class niet enkel de scheidingstekens *o*, *c* en l_1, maar interageren alle scheidingstekens met elkaar (ofwel direct, ofwel indirect). Dit komt vanwege de oplossing voor het probleem van een inconsistente volgorde van attributen (Sectie 8.1.2) die verwerkt is in de UOCLR wrapper class.

De scheidingstekens *o*, *c* en l_1 interageren nog steeds met elkaar vanwege dezelfde reden als bij de OCLR wrapper class (Sectie 6.2.2).

In Sectie 8.1.2 hebben we een oplossing aangehaald voor het probleem van een inconsistente volgorde van attributen. Deze oplossing legt geen vaste volgorde meer op aan de attributen in een tupel. Eender welk attribuut mag met andere woorden als eerste voorkomen in een tupel. Opdat deze oplossing ook effectief werkt, moeten de linker scheidingstekens uniek zijn. Hiermee bedoelen we dat een scheidingsteken l_k enkel mag voorkomen voor het begin van de attribuutwaarde van attribuut *k*, in een tupel. Indien dit niet het geval is, scheidingsteken l_k komt bijvoorbeeld ook in het begin van het tupel voor, dan zal de $\text{exec}_{\text{UOCLR}}$ procedure (Illustratie 8.15) op regel 8 een foutieve positie kiezen als begin van de attribuutwaarde van attribuut *k*, aangezien deze foute positie de eerst voorkomende positie van het scheidingsteken l_k is. Deze "uniek" eigenschap leidt tot het interageren van alle linker scheidingstekens. De eerste letter van de naam van deze wrapper class (UOCLR, Unique-Open-Close-Left-Right) is ook te danken aan deze "uniek" eigenschap. Voor de rechter scheidingstekens is deze eigenschap niet van toepassing. Dit komt doordat enkel naar een rechter scheidingsteken gezocht wordt, nadat de positie van het bijhorende linker scheidingsteken is gevonden.

Verder mogen rechter en linker scheidingstekens niet overlappen. r_k mag met andere woorden niet overlappen met l_{k+1}. Indien dit wel gebeurt, dan zal de wrapper class falen wanneer de volgorde van de attributen inconsistent is. Stel dat tijdens het leren van een wrapper voor de webpagina van Illustratie 8.16 de scheidingstekens r_1 en l_2 gelijkgesteld worden aan "`</b> <i>`". Indien we de geleerde wrapper vervolgens gaan gebruiken om de relatie uit de webpagina van Illustratie 8.12 te halen, dan zal de $\text{exec}_{\text{UOCLR}}$ procedure (Illustratie 8.15) geen telefooncode attribuutwaarde uit het tweede tupel van de webpagina halen. Op regel 4 van Illustratie 8.12 is de tekst "`</b> <i>`" namelijk niet aanwezig. Deze "geen-overlap" eigenschap leidt tot het interageren van alle linker en rechter scheidingstekens.

De reden waarom linker en rechter scheidingstekens niet mogen overlappen is ook van toepassing op het close en r_K scheidingsteken. Dit betekent dat er niet alleen een indirecte, maar ook een directe relatie is tussen deze twee scheidingstekens.

Omdat alle scheidingstekens met elkaar interageren (ofwel direct, ofwel indirect), moeten we alle mogelijke combinaties van de kandidaten van elk scheidingsteken in beschouwing nemen. We moeten met andere woorden het product nemen van de kandidaten van o, c, l_1, …, l_K en r_1, …, r_K. Voor ons voorbeeld van Illustratie 8.16 levert dit 432.972.864 mogelijke kandidaat-combinaties op. Dit is het product van 153 kandidaten voor o, 153 kandidaten voor c, 17 kandidaten voor l_1, 8 kandidaten voor r_1, 8 kandidaten voor r_1 en 17 kandidaten voor r_2. (Illustratie 8.17)

Merk op dat we enkel de belangrijkste directe relaties hebben aangehaald. Deze directe relaties zorgen er tevens voor dat er indirecte relaties zijn tussen alle scheidingstekens. Overige directe relaties worden duidelijk uit de constraints in de volgende sectie en staan in nauw verband met de directe relaties die we in deze sectie hebben besproken.

Verder merken we ook nog op dat we de kandidaten voor elk scheidingsteken van deze wrapper class sorteren van klein naar groot. Kortere scheidingstekens overlappen namelijk minder snel, waardoor we sneller een geldige wrapper kunnen vinden.

8.3.3 Validatie van de kandidaten

Nu we de kandidaten voor onze scheidingstekens gevonden hebben, kunnen we kijken naar de constraints waaraan de kandidaten moeten voldoen, opdat de $\text{exec}_{\text{UOCLR}}$ procedure de correcte relaties uit de pagina's kan gaan halen.

Daar de UOCLR wrapper class gebaseerd is op de OCLR wrapper class, kunnen we alle constraints van de OCLR wrapper class overnemen. Deze worden voor de eenvoud opnieuw weergegeven in Illustratie 8.18. Ter vervollediging worden al deze constraints ook opnieuw weergegeven in het $\text{learn}_{\text{UOCLR}}$algoritme (Illustratie 8.20 - Illustratie 8.22). Zo vinden we C^A op regel 76, C^B op regel 107, C^C regel 110, C^D op regel 45, C^E op regel 48, C^F op regel 55, C^G op regel 57, C^H op regel 60, C^I op regel 62 en C^J op regel 64 terug. Meer details over deze constraints zijn terug te vinden in Sectie 6.2.3. De nieuwe constraints worden weergegeven in Illustratie 8.19 en worden hieronder toegelicht.

De constraint C^K heeft betrekking op het close scheidingsteken. Dit scheidingsteken mag namelijk niet voorkomen achter het open scheidingsteken in het hoofd van een pagina. Op deze manier waakt de constraint erover dat het close scheidingsteken niet te vroeg voorkomt. Dit leidt tot de goede werking van regels 7, 8 en 9 in de $\text{exec}_{\text{UOCLR}}$ procedure (Illustratie 8.15). Op deze regels wordt namelijk gezocht naar linker en rechter scheidingstekens, die voorkomen na het open scheidingsteken en voor het close

Constraint A: u_{l_k} moet een zuiver suffix zijn van de tekst die onmiddellijk voorkomt voor elke waarde van attribuut k in elk van de pagina's

Constraint B: u_{r_k} moet een prefix zijn van de tekst die onmiddellijk voorkomt na elke waarde van attribuut k in elk van de pagina's

Constraint C: u_{r_k} mag geen substring zijn van eender welke waarde van attribuut k in eender welke pagina

Constraint D: u_o moet een substring zijn van het hoofd van elke pagina

Constraint E: u_{l_1} moet een zuiver suffix zijn van het gedeelte van het hoofd van elke pagina dat voorkomt na het eerste voorkomen van u_o

Constraint F: u_c moet een substring zijn van de voet van elke pagina

Constraint G: u_o mag niet voorkomen na u_c in de voet van eender welke pagina

Constraint H: u_o moet een substring zijn van de tekst die zich tussen elke twee opeenvolgende tupels bevindt in elke pagina

Constraint I: u_c moet een substring zijn van de tekst die zich tussen elke twee opeenvolgende tupels bevindt in elke pagina

Constraint J: u_{l_1} moet een zuiver suffix zijn van de tekst die voorkomt na u_o, in de tekst die voorkomt na u_c, in de tekst tussen elke twee opeenvolgende tupels, en in dit in elke pagina

ILLUSTRATIE 8.18: CONSTRAINTS VAN DE OCLR WRAPPER CLASS, DIE DOOR DE UOCLR WRAPPER CLASS OVERGENOMEN WORDEN.

scheidingsteken. Wanneer het eerste open scheidingsteken onmiddellijk gevolgd wordt door een close scheidingsteken, dan worden er geen attribuutwaarden van het eerste tupel van de pagina opgenomen in de relatie. We vinden deze constraint terug op regel 50 in het $\text{learn}_{\text{UOCLR}}$ algoritme (Illustratie 8.20 - Illustratie 8.22).

De constraint C^L stelt dat het scheidingsteken l_1 niet mag overlappen met het open scheidingsteken in het hoofd van een pagina. Indien deze constraint overtreden wordt en we een pagina hebben waarop de volgorde van de attributen inconsistent is in het eerste tupel (attribuut 1 komt niet als eerste voor in het tupel), dan zal de $\text{exec}_{\text{UOCLR}}$ procedure, op regel 8 in Illustratie 8.15, het scheidingsteken l_1 niet vinden. Dit heeft tot gevolg dat de attribuutwaarde van attribuut 1 niet in de relatie opgenomen zal worden. Deze constraint vinden we terug op regel 52 in het $\text{learn}_{\text{UOCLR}}$ algoritme (Illustratie 8.20 - Illustratie 8.22).

Constraint K: u_c mag niet voorkomen na u_o in het hoofd van eender welke pagina

Constraint L: u_{l_1} mag niet overlappen met u_o, in de tekst die voorkomt na u_o, in het hoofd van eender welke pagina

Constraint M: u_{l_1} mag niet overlappen met u_o, in de tekst die voorkomt na u_o, in de tekst die voorkomt na u_c, in de tekst tussen eender welke twee opeenvolgende tupels, en dit in eender welke pagina

Constraint N: u_c mag niet voorkomen in de tekst die voorkomt na u_o, in de tekst die voorkomt na u_c, in de tekst tussen eender welke twee opeenvolgende tupels, en dit in eender welke pagina

Constraint O: u_c mag niet voorkomen in de tekst die zich tussen eender welke twee opeenvolgende attribuutwaarden van hetzelfde tupel bevindt in eender welke pagina

Constraint P: u_{l_k} mag niet strikt na u_o voorkomen, in het hoofd van eender welke pagina, tenzij $k = 1$

Constraint Q: u_{l_k} mag niet strikt voor u_c voorkomen, in de tekst tussen eender welke twee opeenvolgende tupels, en dit in eender welke pagina

Constraint R: u_{l_k} mag niet strikt na u_o voorkomen, in de tekst die voorkomt na u_c, in de tekst tussen eender welke twee opeenvolgende tupels, en dit in eender welke pagina, tenzij k = 1

Constraint S: u_{l_k} mag niet voorkomen in de tekst die zich tussen eender welke twee opeenvolgende attribuutwaarden bevindt, tenzij deze tekst voorkomt tussen waarden voor de attributen k-1 en k

Constraint T: u_{l_k} mag niet strikt voor u_c voorkomen, in de voet van eender welke pagina

Constraint U: u_{r_K} mag niet overlappen met u_c, in de tekst die voorkomt na de laatste attribuutwaarde van eender welk tupel, en dit in eender welke pagina

Constraint V: $u_{l_{k+1}}$ mag niet overlappen met u_{r_k}, in de tekst die voorkomt tussen eender welke waarden voor de attributen k en k+1, en dit in eender welke pagina

ILLUSTRATIE 8.19: NIEUWE CONSTRAINTS WAARAAN DE SCHEIDINGSTEKENS VAN DE UOCLR WRAPPER CLASS MOETEN VOLDOEN.

Wat constraint C^L doet voor het hoofd van een pagina, doet constraint C^M voor de tekst tussen tupels. Constraint C^M vereist namelijk dat het scheidingsteken l_1 niet overlapt met het open scheidingsteken in de tekst tussen tupels. Voor deze constraint moeten we eerst naar de startpositie van het close scheidingsteken gaan en daarna naar de

startpositie van het open scheidingsteken, alvorens naar het scheidingsteken l_1 te zoeken. Indien deze constraint overtreden wordt en we een pagina hebben waarop de volgorde van de scheidingstekens inconsistent is in een tupel (attribuut 1 komt niet als eerste voor in een tupel), dan zal de $exec_{UOCLR}$ procedure, op regel 8 in Illustratie 8.15, het scheidingsteken l_1 niet vinden voor dit tupel. Dit heeft tot gevolg dat de attribuutwaarde van attribuut 1 van dit tupel niet in de relatie wordt opgenomen. Deze constraint vinden we terug op regel 66 in het $learn_{UOCLR}$ algoritme (Illustratie 8.20 - Illustratie 8.22).

Wat constraint C^K doet voor het hoofd van een pagina, doet constraint C^N voor de tekst tussen tupels. Constraint C^N stelt namelijk dat de situatie *c*, *o*, *c* niet mag voorkomen in de tekst tussen tupels. Op deze manier waakt de constraint erover dat het close scheidingsteken niet te vroeg voorkomt. Dit leidt tot de goede werking van regels 7, 8 en 9 in de $exec_{UOCLR}$ procedure (Illustratie 8.15). Op deze regels wordt namelijk gezocht naar linker en rechter scheidingstekens, die voorkomen na het open scheidingsteken en voor het close scheidingsteken. Wanneer beide scheidingstekens (*o* en *c*) in de tekst tussen tupels voorkomen, en dit na de afsluiting van een voorgaande tupel, dan wordt in dit nieuw tupel geen enkel linker en bijgevolg ook geen enkel rechter scheidingsteken gevonden. Bij een overtreding van deze constraint worden er geen attribuutwaarden voor de tupels opgenomen in de relatie. Deze constraint vinden we terug op regel 68 in het $learn_{UOCLR}$ algoritme (Illustratie 8.20 - Illustratie 8.22).

We hebben reeds gezien dat de constraints C^K en C^N erover waken dat het close scheidingsteken niet te vroeg voorkomt. Constraint C^K zorgt hiervoor in het hoofd van een pagina en constraint C^N zorgt hiervoor in de tekst tussen tupels. Het close scheidingsteken mag echter ook niet tussen de attribuutwaarden van een tupel voorkomen. Dit is exact wat constraint C^O eist. Indien deze constraint overtreden wordt, kunnen bepaalde linker scheidingstekens niet gevonden worden op regel 8 in de $exec_{UOCLR}$ procedure (Illustratie 8.15). Dit komt doordat deze scheidingstekens buiten de tupel grenzen vallen. Een overtreding van deze constraint leidt bijgevolg tot het niet opgenomen worden van attribuutwaarden van tupels. Deze constraint is terug te vinden op regel 38 in het $learn_{UOCLR}$ algoritme (Illustratie 8.20 - Illustratie 8.22).

De constraint C^P heeft betrekking op de "uniek" eigenschap. Deze constraint stelt namelijk dat het scheidingsteken l_k niet strikt achter het open scheidingsteken mag voorkomen in het hoofd van een pagina, tenzij k=1. We zeggen strikt achter, omdat overlap niet is toegestaan. Indien deze constraint overtreden wordt, denkt de $exec_{UOCLR}$ procedure, op regel 8 in Illustratie 8.15, dat de attribuutwaarde van attribuut *k* te vroeg begint. Een overtreding van deze constraint leidt bijgevolg tot het opnemen van foutieve

attribuutwaarden. Deze constraint is terug te vinden op regel 84 in het $learn_{UOCLR}$ algoritme (Illustratie 8.20 - Illustratie 8.22).

Constraint C^Q heeft ook betrekking op de "uniek" eigenschap. Deze constraint stelt namelijk dat het scheidingsteken l_k niet strikt voor het close scheidingsteken mag voorkomen in de tekst tussen tupels. We zeggen strikt voor, omdat overlap niet is toegestaan. Wanneer deze constraint overtreden wordt en we een pagina hebben waarop het correcte scheidingsteken l_k plots achter het foutieve scheidingsteken l_k op het einde van een tupel voorkomt (vanwege een inconsistente volgorde van attributen), dan zal de $exec_{UOCLR}$ procedure, op regel 7 in Illustratie 8.15, een foutieve attribuutwaarde voor attribuut k uit dit tupel halen. Deze constraint is terug te vinden op regel 90 in het $learn_{UOCLR}$ algoritme (Illustratie 8.20 - Illustratie 8.22).

In de tekst tussen tupels, wordt een tupel afgesloten door een close scheidingsteken, en wordt een nieuw tupel geopend door een open scheidingsteken. Waar constraint C^Q nagaat of het scheidingsteken l_k niet strikt voor het close scheidingsteken voorkomt, gaat constraint C^R nagaan of het scheidingsteken l_k niet strikt achter het open scheidingsteken voorkomt, tenzij k=1. We zeggen strikt achter, omdat overlap niet is toegestaan. Deze constraint komt overeen met wat constraint C^P doet voor het hoofd van een pagina en heeft bijgevolg ook betrekking op de "uniek" eigenschap. Indien deze constraint overtreden wordt, denkt de $exec_{UOCLR}$ procedure, op regel 8 in Illustratie 8.15, dat de attribuutwaarde van attribuut k te vroeg begint. Een overtreding van deze constraint leidt bijgevolg tot het opnemen van foutieve attribuutwaarden. Deze constraint is terug te vinden op regel 92 in het $learn_{UOCLR}$ algoritme (Illustratie 8.20 - Illustratie 8.22).

Constraint C^S heeft ook betrekking op de "uniek" eigenschap. Deze constraint controleert of het scheidingsteken l_k niet voorkomt in de tekst tussen de attribuutwaarden, tenzij het de tekst betreft tussen kolom k-1 en k. Tussen deze kolommen moet het scheidingsteken l_k namelijk wel voorkomen, anders kunnen we de attribuutwaarde voor attribuut k nooit uit een tupel halen. Wanneer deze constraint overtreden wordt en we een pagina hebben waarop het correcte scheidingsteken l_k plots achter het foutieve scheidingsteken l_k voorkomt (vanwege een inconsistente volgorde van attributen), dan zal de $exec_{UOCLR}$ procedure, op regel 7 in Illustratie 8.15, een foutieve attribuutwaarde voor attribuut k uit dit tupel halen. Deze constraint is terug te vinden op regel 95 in het $learn_{UOCLR}$ algoritme (Illustratie 8.20 - Illustratie 8.22).

Wat constraint C^Q doet voor de tekst tussen tupels, doet Constraint C^T voor de voet van een pagina. Constraint C^T stelt namelijk dat het scheidingsteken l_k niet strikt voor het close scheidingsteken mag voorkomen in de voet van een pagina. We zeggen strikt voor, omdat overlap niet is toegestaan. Constraint C^T is tevens de laatste constraint die

betrekking heeft op de "uniek" eigenschap. Wanneer deze constraint overtreden wordt en we een pagina hebben waarop het correcte scheidingsteken l_k plots achter het foutieve scheidingsteken l_k op het einde van het laatste tupel van een pagina voorkomt (vanwege een inconsistente volgorde van attributen), dan zal de $\text{exec}_{\text{UOCLR}}$ procedure, op regel 7 in Illustratie 8.15, een foutieve attribuutwaarde voor attribuut k uit dit tupel halen. Deze constraint is terug te vinden op regel 97 in het $\text{learn}_{\text{UOCLR}}$ algoritme (Illustratie 8.20 - Illustratie 8.22).

Constraint C^U stelt dat het scheidingsteken r_K niet mag overlappen met het close scheidingsteken in de tekst die voorkomt na een tupel. Indien deze constraint overtreden wordt en we een pagina hebben waarop de volgorde van de attributen inconsistent is (attribuut K komt niet als laatste voor in een tupel), dan zal de $\text{exec}_{\text{UOCLR}}$ procedure, op regel 9 in Illustratie 8.15, het scheidingsteken r_K niet vinden voor een bepaald tupel. Dit heeft tot gevolg dat de attribuutwaarde van het laatste attribuut van dit tupel niet in de relatie opgenomen wordt. Deze constraint vinden we terug op regel 106 in het $\text{learn}_{\text{UOCLR}}$ algoritme (Illustratie 8.20 - Illustratie 8.22).

De laatste constraint C^V houdt er toezicht op dat de scheidingstekens r_k en l_{k+1} niet overlappen in de tekst tussen de attributen k en k+1. Indien deze constraint overtreden wordt en we een pagina hebben waarop attribuut k niet gevolgd wordt door attribuut k+1 in een tupel (vanwege een inconsistente volgorde van attributen), dan zal de $\text{exec}_{\text{UOCLR}}$ procedure, op regel 8 en 9 in Illustratie 8.15, problemen hebben met het vinden van de scheidingstekens l_{k+1} en r_K voor dit tupel. Hierdoor zullen bepaalde attribuutwaarden van dit tupel niet in de relatie opgenomen worden. Deze constraint vinden we terug op regel 119 in het $\text{learn}_{\text{UOCLR}}$ algoritme (Illustratie 8.20 - Illustratie 8.22).

8.4 Het $\text{learn}_{\text{UOCLR}}$ algoritme

Nu we besproken hebben hoe de $\text{exec}_{\text{UOCLR}}$ procedure gebruik maakt van een wrapper en hoe we kandidaten voor de scheidingstekens kunnen vinden en valideren, kunnen we het algoritme bespreken dat verantwoordelijk is voor het automatisch leren van een UOCLR wrapper. Dit algoritme, genaamd $\text{learn}_{\text{UOCLR}}$, is weergegeven in Illustratie 8.20, Illustratie 8.21 en Illustratie 8.22.

Het $\text{learn}_{\text{UOCLR}}$ algoritme krijgt als input een verzameling van voorbeeldpagina's en hun bijhorende relaties, genaamd $\mathcal{E}$. De for lussen op regels 2 tot en met 9 dienen om alle mogelijke combinaties van de kandidaten van alle scheidingstekens te verkrijgen. Hiervoor wordt gebruik gemaakt van de cands_l & cands_r en de $\text{cands}_{o,c}$ functies. Deze zijn respectievelijk in Sectie 4.3 en Sectie 6.3 besproken en worden voor de volledigheid opnieuw vermeld op respectievelijk regel 16 & 18 en regel 20. Op regel 10 wordt de

$valid_{l_1,\ldots,l_K,r_1,\ldots,r_K,o,c}$ functie opgeroepen om de combinatie van alle scheidingstekens te valideren. Zodra een geldige combinatie gevonden wordt stopt het algoritme met zoeken naar scheidingstekens (regel 14). De output van het $learn_{UOCLR}$ algoritme is een wrapper $\langle o, c, [\langle l_1, r_1\rangle, \ldots, \langle l_K, r_K\rangle]\rangle$ van $2K+2$ scheidingstekens.

De functie $valid_{l_1,\ldots,l_K,r_1,\ldots,r_K,o,c}$ controleert of de combinatie van de kandidaten van alle scheidingstekens geldig is. Deze functie is opgedeeld in drie hulpfuncties, namelijk $valid_{l_1,\ldots,l_K,o,c}$, $valid_{r_1,\ldots,r_K,c}$ en $valid_{l_1,\ldots,l_K,r_1,\ldots,r_K}$. Elke hulpfunctie controleert de constraints die betrekking hebben op de scheidingstekens die in subscript in de naam van de functie zijn vermeld. De constraints waaraan de kandidaten moeten voldoen werden reeds besproken in Sectie 8.3.3.

De functies $valid_{l_1,o,c}$, $valid_l$ en $valid_r$ zijn overgenomen van het $learn_{OCLR}$ algoritme en worden hier weergegeven voor de volledigheid. De functie $valid_{l_1,o,c}$ is echter wel uitgebreid met enkele van de nieuwe constraints die toebehoren aan de deze UOCLR wrapper class.

De hulpfuncties die in Illustratie 8.20 - Illustratie 8.22 vermeld worden, maar niet beschreven zijn, zijn terug te vinden in Bijlage A.

Een voorbeeld van een geldige UOCLR wrapper voor het country-code voorbeeld van dit hoofdstuk (Figuur 8.1) is volgende vector van $2K+2$ scheidingstekens:

⟨'
','
',[⟨'',''⟩,⟨'',''⟩]⟩

```
1:   procedure learn_UOCLR(examples E)
2:     for each u_o ∈ cands_{o,c}(E)
3:       for each u_c ∈ cands_{o,c}(E)
4:         for each u_{l_1} ∈ cands_l(1, E)
5:           ...
6:             for each u_{l_K} ∈ cands_l(K, E)
7:               for each u_{r_1} ∈ cands_r(1, E)
8:                 ...
9:                   for each u_{r_K} ∈ cands_r(K, E)
10:                    if valid_{l_1,…,l_K,r_1,…,r_K,o,c}(u_{l_1}…u_{l_K} u_{r_1}…u_{r_K} u_o u_c, E) then
11:                      l_1 := u_{l_1}, …, l_K := u_{l_K},
12:                      r_1 := u_{r_1}, …, r_K := u_{r_K},
13:                      o := u_o, c := u_c,
14:                      break all loops
15:    return UOCLR wrapper ⟨o, c, [⟨l_1, r_1⟩, …, ⟨l_K, r_K⟩]⟩

16:  procedure cands_l(index k, examples E)
17:    return set of all suffixes of the shortest string in neighbors_l(k, E),
               Sorted in ascending order

18:  procedure cands_r(index k, examples E)
19:    return set of all prefixes of the shortest string in neighbors_r(k, E),
               Sorted in ascending order

20:  procedure cands_{o,c}(examples E)
21:    return set of all substrings of the shortest string in seps(K, E),
               sorted in ascending order

22:  procedure valid_{l_1,..,l_K,r_1,..,r_K,o,c}(candidates u_{l_1}…u_{l_K} u_{r_1}…u_{r_K} u_o u_c, examples E)
23:    if not valid_{l_1,…,l_K,o,c}(u_{l_1}…u_{l_K} u_o u_c, E) then
24:      return FALSE
25:    if not valid_{r_1,…,r_K,c}(u_{r_1}…u_{r_K} u_c, E) then
26:      return FALSE
27:    if not valid_{l_1,…,l_K,r_1,…,r_K}(u_{l_1}…u_{l_K} u_{r_1}…u_{r_K}, E) then
28:      return FALSE
29:    return TRUE

30:  procedure valid_{l_1,…,l_K,o,c}(candidates u_{l_1}…u_{l_K} u_o u_c, examples E)
31:    for each u_{l_k} ∈ [u_{l_1}, …, u_{l_K}]
32:      if k = 1 then
33:        if not valid_{l_1,o,c}(u_{l_1} u_o u_c, E) then
34:          return FALSE
35:      else
36:        if not valid_l(u_{l_k}, k, E) then
37:          return FALSE
```

ILLUSTRATIE 8.20: HET LEARN$_{UOCLR}$ ALGORITME, DAT AUTOMATISCH EEN UOCLR WRAPPER KAN LEREN UIT EEN VERZAMELING VAN VOORBEELDPAGINA'S EN HUN BIJHORENDE RELATIES. (DEEL 1)

```
38:         for each s ∈ seps(k-1, ℰ)
39:           if u_c is a substring of s then
40:             return FALSE
41:       if not valid_{l_k,o,c}(u_{l_k} u_o u_c, ℰ) then
42:         return FALSE
43:     return TRUE

44:   procedure valid_{l_1,o,c}(candidates u_{l_1} u_o u_c, examples ℰ)
45:     for each s ∈ heads(ℰ)
46:       if u_o is not a substring of s then
47:         return FALSE
48:       if u_{l_1} is not a proper suffix of scan(s, u_o) then
49:         return FALSE
50:       if u_c is a substring of scan(s, u_o) then
51:         return FALSE
52:       if u_{l_1} overlaps with u_o in scan(s, u_o) then
53:         return FALSE
54:     for each s ∈ tails(ℰ)
55:       if u_c is not a substring of s then
56:         return FALSE
57:       if u_o occurs after u_c in s then
58:         return FALSE
59:     for each s ∈ seps(K, ℰ)
60:       if u_o is not a substring of s then
61:         return FALSE
62:       if u_c is not a substring of s then
63:         return FALSE
64:       if u_{l_1} is not a proper suffix of scan(scan(s, u_c), u_o) then
65:         return FALSE
66:       if u_{l_1} overlaps with u_o in scan(scan(s, u_c), u_o) then
67:         return FALSE
68:       if u_c is a substring of scan(scan(s, u_c), u_o) then
69:         return FALSE
70:     return TRUE

71:   procedure valid_1(candidate u_{l_k}, index k, examples ℰ)
72:     if k = 1 then
73:       for each s ∈ tails(ℰ)
74:         if u_{l_k} is a substring of s then
75:           return FALSE
76:     for each s ∈ neighbors_1(k, ℰ)
77:       if u_{l_k} is not a proper suffix of s then
78:         return FALSE
79:     return TRUE
```

ILLUSTRATIE 8.21: HET LEARN$_{UOCLR}$ ALGORITME, DAT AUTOMATISCH EEN UOCLR WRAPPER KAN LEREN UIT EEN VERZAMELING VAN VOORBEELDPAGINA'S EN HUN BIJHORENDE RELATIES. (DEEL 2)

```
80:  procedure valid_{l_k,o,c}(candidates u_{l_k} u_o u_c, examples ℰ)
81:    for each a ∈ [1,…,K]
82:      if a = k then
83:        continue
84:      if a = 1 then
85:        for each s ∈ heads(ℰ)
86:          if u_{l_k} is a substring of scanAfter(s,u_o) then
87:            return FALSE
88:      for each s ∈ seps(a-1,ℰ)
89:        if a-1 = K then
90:          if u_{l_k} is a substring of scanBefore(s,u_c) then
91:            return FALSE
92:          if u_{l_k} is a substring of scanAfter(scan(s,u_c),u_o) then
93:            return FALSE
94:        else
95:          if u_{l_k} is a substring of s then
96:            return FALSE
97:      if a = K then
98:        for each s ∈ tails(ℰ)
99:          if u_{l_k} is a substring of scanBefore(s,u_c) then
100:           return FALSE
101:   return TRUE

102: procedure valid_{r_1,…,r_K,c}(candidates u_{r_1}…u_{r_K} u_c, examples ℰ)
103:   for each u_{r_k} ∈ [u_{r_1},…,u_{r_K}]
104:     if not valid_r(u_{r_k}, k, ℰ) then
105:       return FALSE
106:   for each s ∈ neighbors_r(K,ℰ)
107:     if u_c overlaps with u_{r_K} in s then
108:       return FALSE
109:   return TRUE

110: procedure valid_r(candidate u_{r_k}, index k, examples ℰ)
111:   for each s ∈ neighbors_r(k,ℰ)
112:     if u_{r_k} is not a prefix of s then
113:       return FALSE
114:   for each s ∈ attribs(k,ℰ)
115:     if u_{r_k} is a substring of s then
116:       return FALSE
117:   return TRUE

118: procedure valid_{l_1,…,l_K,r_1,…,r_K}(candidates u_{l_1}…u_{l_K} u_{r_1}…u_{r_K}, examples ℰ)
119:   for each a ∈ [1,…,K-1]
120:     for each s ∈ seps(a,ℰ)
121:       if u_{l_{a+1}} overlaps with u_{r_a} in s then
122:       return FALSE
123:   return TRUE
```

ILLUSTRATIE 8.22: HET LEARN$_{UOCLR}$ ALGORITME, DAT AUTOMATISCH EEN UOCLR WRAPPER KAN LEREN UIT EEN VERZAMELING VAN VOORBEELDPAGINA'S EN HUN BIJHORENDE RELATIES. (DEEL 3)

8.5 Conclusie

We zijn dit hoofdstuk begonnen met het bespreken van enkele problemen in verband met een non-uniforme opmaak op webpagina's. Voor twee van deze problemen, ontbrekende scheidingstekens en een inconsistente volgorde van attributen, hebben we een oplossing gegeven. Deze oplossing hebben we later ook opgenomen in de UOCLR wrapper class. We hebben daarnaast ook een derde probleem in verband met een non-uniforme opmaak aangehaald, namelijk inconsistent gebruik van scheidingstekens. Dit probleem is echter niet eenvoudig oplosbaar en vereist kennis van de semantiek van de data.

Na het bestuderen van de problemen hebben we de UOCLR wrapper class besproken. Deze wrapper class bouwt verder op de OCLR wrapper class en kan overweg met ontbrekende scheidingstekens en een inconsistente volgorde van attributen. Dit is mogelijk door de toevoeging van twee eigenschappen aan de wrapper class, namelijk de "uniek" en "geen-overlap" eigenschap. De "uniek" eigenschap stelt dat elk linker scheidingsteken uniek moet zijn binnen een tupel, het scheidingsteken l_k mag met andere woorden enkel juist voor attribuut k voorkomen. De "geen-overlap" eigenschap stelt dat scheidingstekens niet mogen overlappen, omdat deze anders ontbreken wanneer de volgorde van de attributen niet meer consistent is. Deze twee eigenschappen hebben echter tot gevolg dat alle scheidingstekens interageren, wat een totaal van 22 constraints oplevert voor deze wrapper class. Doordat alle scheidingstekens interageren, kan het aantal kandidaat-combinaties snel hoog oplopen voor relaties met veel attributen.

We hebben de $\text{exec}_{\text{UOCLR}}$ procedure besproken die gebruikt maakt van een UOCLR wrapper om de relatie uit een webpagina te halen. Deze webpagina mag ontbrekende scheidingstekens hebben en/of een inconsistente volgorde van attributen. De UOCLR wrapper is, net zoals de OCLR wrapper, een vector van $2K+2$ scheidingstekens, namelijk $\langle o, c, [\langle l_1, r_1\rangle, \ldots, \langle l_K, r_K\rangle]\rangle$.

Als laatste hebben we het $\text{learn}_{\text{UOCLR}}$ algoritme besproken dat automatisch een UOCLR wrapper kan leren uit een verzameling van voorbeeldpagina's en hun bijhorende relaties.

Tot slot merken we op dat we de UOCLR wrapper class kunnen combineren met de HLRT wrapper class om zo de UHOCLRT wrapper class te verkrijgen, net zoals we de OCLR en HLRT wrapper class gecombineerd hebben om de HOCLRT wrapper class te verkrijgen.

9. VERBETERINGEN

In hoofdstuk 4 tot en met 8 hebben we learn algoritmes besproken voor respectievelijk de LR, HLRT, OCLR, HOCLRT en UOCLR wrapper class. Uit de voorbeelden die bij deze algoritmes gebruikt werden, bleek al snel dat het aantal kandidaat-combinaties zeer hoog kan oplopen. Bij het voorbeeld van de HOCLRT wrapper class waren er iets meer dan $1{,}31\times10^{14}$ kandidaat-combinaties en dit betrof nog maar een klein voorbeeld. Dergelijke kandidaat-aantallen kosten veel tijd om te controleren. In dit hoofdstuk bespreken we daarom twee verbeteringen/uitbreidingen die dit aantal omlaag kunnen brengen. Een eerste verbetering maakt gebruik van gemeenschappelijke strings waaruit de kandidaten gegenereerd moeten worden. Deze verbetering is beschreven in Sectie 9.1. De tweede verbetering, beschreven in Sectie 9.2, gaat gebruik maken van de syntax van HTML om het aantal kandidaat-combinaties te verlagen.

9.1 GEMEENSCHAPPELIJKE STRINGS

In Sectie 4.2.1 hebben we reeds besproken waarom het voldoende is om enkel uit de kortste string kandidaten te genereren. Dit komt omdat een langere string zal leiden tot kandidaten die langer zijn dan de kortste string. Deze lange kandidaten kunnen onmogelijk voorkomen in de kortste string en zijn bijgevolg al op voorhand ongeldig. Deze techniek hebben we zowel voor leren van de linker en rechter scheidingstekens, als voor het leren van het head, tail, open en close scheidingsteken toegepast. Bij linker scheidingstekens worden suffixen van de kortste string gegenereerd, bij rechter scheidingstekens worden prefixen van de kortste string gegenereerd en bij het head, tail, open en close scheidingsteken worden alle mogelijke substrings van de kortste string gegenereerd.

Als we de twee strings, van Sectie 4.2.1 paragraaf 2, die in aanmerking kwamen voor het genereren van kandidaten voor het scheidingsteken l_1 even hernemen (“</i>
⇓<b>” en “<html><head><title>Some Country Codes</title></head><body>⇓<b>”), zien we dat “>⇓<b>” de langste gemeenschappelijke suffix is van deze twee strings. Alle suffix-kandidaten die meer dan deze 5 karakters bevatten, zullen bijgevolg als ongeldig aangeduid worden door een validatie test. Het is met andere woorden voldoende om alle suffix-kandidaten te genereren uit deze gemeenschappelijke suffix. Dit lijdt tot 5 kandidaten voor scheidingsteken l_1, in tegenstelling tot de 12 kandidaten die uit de kortste string tot stand komen.

De techniek die hierboven beschreven wordt, is niet alleen toepasbaar bij linker scheidingstekens, maar ook bij alle andere scheidingstekens die we hebben gezien. De

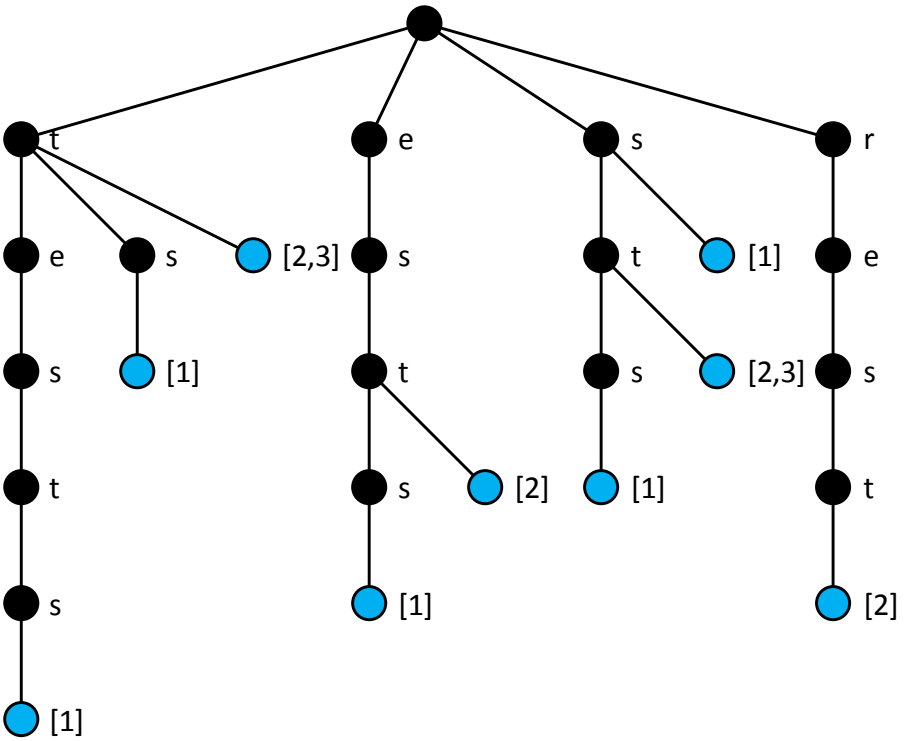
reductie in het aantal kandidaten voor een scheidingsteken hangt af van de inkorting van de lengte van de string waaruit de kandidaten worden gegenereerd. Anders gesteld, hoe korter de gemeenschappelijke string is, ten opzichte van de kortste string die in aanmerking komt voor het genereren van kandidaten, hoe groter de reductie in het aantal kandidaten zal zijn.

Bij linker en rechter scheidingstekens worden respectievelijk suffixen en prefixen uit de gekozen string gegenereerd. Voor linker scheidingstekens gaan we bijgevolg op zoek naar de langste gemeenschappelijke suffix en voor rechter scheidingstekens gaan we op zoek naar de langste gemeenschappelijke prefix.

In het geval van het head, tail, open en close scheidingsteken worden substrings uit de gekozen string gegenereerd. Substrings hebben echter de eigenschap dat deze eender waar kunnen beginnen in de oorspronkelijke string. Dit in tegenstelling tot suffixen en prefixen die respectievelijk op het einde en het begin van een string moeten voorkomen. Deze eigenschap van substrings moet worden weerspiegeld in het vinden van de gemeenschappelijke substrings, waaruit de kandidaten gegenereerd worden. Indien deze eigenschap niet weerspiegeld wordt, kunnen we gemeenschappelijke substrings mislopen. Merk op dat we hier spreken over meerdere gemeenschappelijke substrings waaruit we kandidaten gaan genereren, dit in tegenstelling tot de standaard methode waarbij we alle kandidaten uit één gekozen string generen. De reden hiervoor wordt duidelijk uit volgend voorbeeld. Stel dat we de twee fictieve strings "bbb" en "ddd" hebben. We zien dat beide strings een "" en "" tag hebben. De inhoud tussen deze tags is echter verschillend in beide strings. We krijgen bijgevolg twee losstaande gemeenschappelijke substrings, waaruit we alle mogelijke substrings (kandidaten) gaan genereren. Deze methode kan, net zoals de standaard aanpak, leiden tot duplicaten. Zo zullen beiden gemeenschappelijke substrings de substring "i>" genereren. We moeten deze duplicaten bijgevolg uit onze lijst met kandidaten filteren.

Om alle mogelijke gemeenschappelijke substrings te vinden, maken we gebruik van een eigen variant op de *Generalized suffix tree* [14]. We zullen dit begrip, samen met onze aanpassingen, stapsgewijs uitleggen aan de hand van volgende drie strings: "test", "rest", "st". Deze drie strings krijgen respectievelijk de id's 1, 2 en 3.

Een generalized suffix tree is een boom die wordt opgebouwd door elke suffix van een set van strings toe te voegen. In Figuur 9.1 zien we de generalized suffix tree horende bij onze drie strings. Elke knoop van deze boom heeft een karakter als label, met uitzondering van de wortel en de blauwe knopen. De wortel van de boom heeft geen label, omdat deze knoop het begin van elke suffix aanduidt. De blauwe knopen duiden op hun beurt het einde van een suffix aan. Deze blauwe knopen bevatten bijgevolg geen



FIGUUR 9.1: STANDAARD GENERALIZED SUFFIX TREE.

karakter van de suffix, maar krijgen als label het id van de string waaruit deze suffix gegenereerd is, geschreven tussen square brackets. Voor het vinden van gemeenschappelijke substrings gaan we op zoek naar gemeenschappelijke paden. De formele definitie van een gemeenschappelijk pad wordt hieronder gegeven. De laatste voorwaarde van deze definitie stelt dat we het gemeenschappelijke pad zo lang mogelijk moeten maken. In onze boom hebben we twee gemeenschappelijke paden, wat betekent dat we voor onze drie strings twee gemeenschappelijke substrings hebben, namelijk "t" en "st".

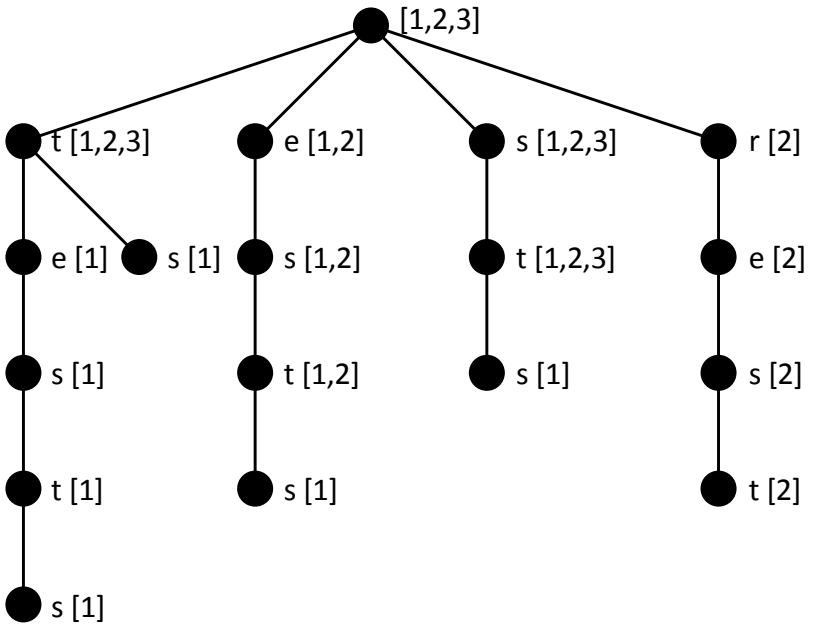
Definitie 9.1: Gemeenschappelijk pad in een generalized suffix tree.

Gegeven een set van strings $\{s_1,\ldots,s_m\}$, waaruit een generalized suffix tree T=(V,E,L) is opgebouwd, met V de verzameling van knopen van de boom, E de verzameling van bogen van de boom en L de verzameling van bladeren (blauwe knopen) van de boom. Een gemeenschappelijk pad in deze boom is een reeks van knopen $v_1 \ldots v_n$ waarvoor geldt dat:

- $v_i \in V$ en $v_i \notin L$ voor elke $1 \leq i \leq n$
- v_1 is de wortel van boom T
- $(v_i,v_{i+1}) \in E$ voor elke $1 \leq i < n$
- v_i is, voor elke $1 \leq i \leq n$, bezocht door minimum één suffix van elke string $s_k \in \{s_1,\ldots,s_m\}$

- elke $v_{n+1} \in V$, met $(v_n,v_{n+1}) \in E$, is ook een blad ($v_{n+1} \in L$) of is niet bezocht door minimum één suffix van elke string $s_k \in \{s_1,\ldots,s_m\}$

Voor onze toepassing kunnen we echter twee verbeteringen doorvoeren die de constructie van de generalized suffix tree optimaliseren. De eerste verbetering houdt in dat we niet enkel op het einde van een tak bijhouden welke strings deze suffix hebben, maar dat we voor elke knoop bijhouden welke strings een suffix hebben die langs de knoop komt. We gaan met andere woorden onze blauwe knopen verwijderen en hun labels integreren in de labels van de zwarte knopen. Deze verbetering heeft tot gevolg dat we niet meer alle takken tot het einde moeten doorlopen om te weten te komen of een knoop bezocht is geweest door een suffix van elke string. Wanneer we bijgevolg zoeken naar gemeenschappelijke paden, kunnen we onmiddellijk stoppen met een pad zodra we een knoop tegen komen die niet meer bezocht is door een suffix van elke string. In Figuur 9.2 zien we de nieuwe generalized suffix tree voor onze drie strings. Elke knoop in deze boom heeft nu een id sectie in zijn label, geschreven tussen square brackets. Omdat we de blauwe knopen hebben verwijderd, moet de definitie van een gemeenschappelijk pad eveneens aangepast worden. Deze nieuwe definitie wordt hieronder gegeven.



FIGUUR 9.2: GENERALIZED SUFFIX TREE WAARBIJ WE STRING ID'S BIJHOUDEN IN ELKE KNOOP.

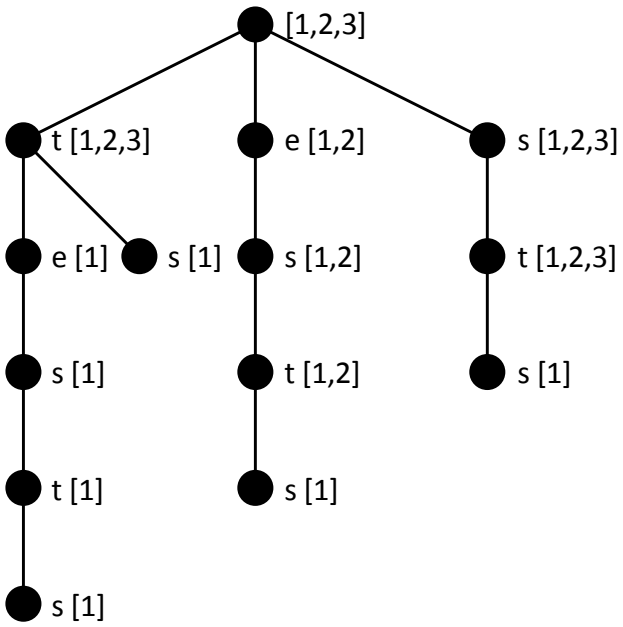
Definitie 9.2: Gemeenschappelijk pad in een generalized suffix tree.

Gegeven een set van strings $\{s_1,\ldots,s_m\}$, waaruit een generalized suffix tree T=(V,E) is opgebouwd, met V de verzameling van knopen van de boom en E de verzameling

van bogen van de boom. Een gemeenschappelijk pad in deze boom is een reeks van knopen $v_1 \ldots v_n$ waarvoor geldt dat:

- $v_i \in V$ voor elke $1 \leq i \leq n$
- v_1 is de wortel van boom T
- $(v_i, v_{i+1}) \in E$ voor elke $1 \leq i < n$
- v_i is, voor elke $1 \leq i \leq n$, bezocht door minimum één suffix van elke string $s_k \in \{s_1, \ldots, s_m\}$
- een knoop $v_{n+1} \in V$, met $(v_n, v_{n+1}) \in E$, bestaat niet of is niet bezocht door minimum één suffix van elke string $s_k \in \{s_1, \ldots, s_m\}$

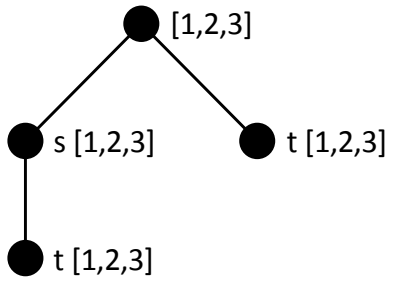
Onze tweede verbetering zorgt ervoor dat het aantal vertakkingen wordt verminderd. Dit doet deze verbetering door te stellen dat er enkel knopen worden aangemaakt voor het toevoegen van suffixen van de eerste string. Alle andere strings die volgen kunnen enkel melden dat zij een suffix hebben die langs een bestaande knoop komt. Zij mogen met andere woorden geen nieuwe vertakkingen aanmaken. Deze nieuwe vertakkingen zijn namelijk overbodig, aangezien deze nooit gedeeld zullen zijn met de suffixen van de eerste string. Dit betekent dat deze nieuwe vertakkingen niet tot substrings zullen leiden die gemeenschappelijk zijn voor alle strings. Figuur 9.3 toont de nieuwe generalized suffix tree voor onze drie strings.



FIGUUR 9.3: GENERALIZED SUFFIX TREE WAARBIJ WE STRING ID'S BIJHOUDEN IN ELKE KNOOP EN WAARBIJ WE ENKEL KNOPEN AANMAKEN BIJ HET TOEVOEGEN VAN DE EERSTE STRING.

De boom in Figuur 9.3 bevat één vertakking minder ten opzichte van de boom in Figuur 9.2, namelijk de meest rechtse tak. Dit is maar een kleine vermindering in het aantal

knopen/vertakkingen. Stel dat we de strings niet meer volgens hun id toevoegen aan de boom, maar dat we beginnen met de kortste string ("st", id 3). Dit levert ons de generalized suffix tree van Figuur 9.4 op. Deze boom bevat slechts 4 knopen, dit in tegenstelling tot de boom van Figuur 9.3 die 18 knopen bevat. Deze kortste string kunnen we eenvoudig vinden door één keer door de set van strings te gaan. Een verbetering zoals in Figuur 9.4 is echter enkel mogelijk wanneer het verschil in lengte, tussen de kortste string en de eerste string in de set, aanzienlijk is.



FIGUUR 9.4: GENERALIZED SUFFIX TREE WAARBIJ WE STRING ID'S BIJHOUDEN IN ELKE KNOOP EN WE ENKEL KNOPEN AANMAKEN BIJ HET TOEVOEGEN VAN DE KORTSTE STRING.

Als laatste beschrijven we twee interessante stellingen die de gemeenschappelijke strings verbetering met zich meebrengt. De eerste stelling wordt hieronder gegeven.

Stelling 9.1:

> Indien een learn algoritme een geldige wrapper vindt wanneer hij gebruik maakt van de gemeenschappelijke strings verbetering, dan zal het learn algoritme eveneens dezelfde wrapper vinden wanneer hij geen gebruik maakt van deze verbetering.

Deze stelling is correct aangezien de gemeenschappelijke strings verbetering enkel stringdelen weggooit die niet gemeenschappelijk zijn voor alle voorbeeldpagina's. Kandidaten die deze stringdelen bevatten zullen namelijk altijd falen bij één of andere constraint van de wrapper class, waartoe het learn algoritme behoort. Het omgekeerde van deze stelling is ook waar en wordt hieronder in een tweede stelling gegeven.

Stelling 9.2:

> Indien een learn algoritme geen geldige wrapper vindt wanneer hij gebruik maakt van de gemeenschappelijke strings verbetering, dan zal het learn algoritme eveneens geen wrapper vinden wanneer hij geen gebruik maakt van deze verbetering.

9.2 TOKENSTRINGS

Door gebruik te maken van de syntax van HTML, meer bepaald tags, beschikken we over een tweede mogelijkheid om het aantal kandidaten te reduceren. Stel dat we volgend stukje tekst op een webpagina hebben staan: "Title goes here". Deze string heeft een lengte van 42 en levert bijgevolg 903 substrings op. In deze string zien we een open tag en een close tag, waarbij de open tag een attribuut bevat. (In HTML bestaat elk attribuut uit een ⟨naam, waarde⟩ paar.) Indien we elke tag als één token zouden beschouwen, dan krijgen we een string van lengte 17 en bijgevolg 153 substrings. Dit is een enorme verbetering. We weten dat de kortst mogelijke tag ook altijd uit minimum drie karakters bestaat, namelijk de angle brackets ("<" en ">") en de naam van de tag die tussen de angle brackets vermeld wordt. Een voorbeeld van een dergelijke korte tag is "<i>". Elke tag die we in een string vervangen door een token levert bijgevolg een reductie, van minimum 2, op in de lengte van de string. We noemen deze nieuwe string, waarin elke tag vervangen is door één token, een tokenstring.

Karakters die geen deel uitmaken van een tag worden niet door deze techniek aangepast. Deze karakters worden met andere woorden rechtstreeks in de tokenstring gekopieerd. Hierdoor blijft het nog steeds mogelijk om attribuutwaarden te scheiden door middel van bijvoorbeeld een puntkomma.

In Illustratie 9.1 zien we de broncode van een eenvoudige webpagina. In deze webpagina worden landen tussen "" tags geplaatst. Het "id" van een "" tag komt overeen met de telefooncode van het land dat de tag omvat. In de eerste "" tag wordt eerst het "id" attribuut vermeld en daarna het "class" attribuut. In de tweede "" tag wordt het "class" attribuut echter eerst vermeld en daarna pas het "id" attribuut. Deze webpagina hanteert met andere woorden geen consistente volgorde voor de tag attributen. Dit is ook niet vereist door de HTML standaard, wat betekent dat dit een geldige webpagina is. In deze webpagina is de gemeenschappelijke suffix voor kandidaat l_1 gelijk aan "'>". (Merk op dat we in deze paragraaf nog geen gebruik maken van een tokenstring.) De "<title>" tag in deze webpagina heeft echter eenzelfde suffix, waardoor het onmogelijk is om een LR wrapper voor deze webpagina te leren. Indien de attributen in de tags gesorteerd waren, met als volgorde "id" en "class", dan hadden we "class='country'>" als gemeenschappelijke suffix voor kandidaat l_1, waardoor het wel mogelijk was om een LR wrapper te leren. Dit voorbeeld illustreert dat het leren van een wrapper bemoeilijkt kan worden door de volgorde van de attributen in tags.

```
1: <html><head><title class='title'>Some Countries</title></head><body>
2: <span id='32' class='country'>Belgium</span><br>
3: <span class='country' id='243'>Congo</span><br>
4: </body></html>
```

ILLUSTRATIE 9.1: BRONCODE VAN EEN WEBPAGINA, WAARBIJ ER GEEN CONSISTENTIE IS IN DE VOLGORDE VAN DE TAG ATTRIBUTEN.

De moeilijkheden die de tag attributen veroorzaken worden alleen maar erger wanneer we gebruik gaan maken van tokenstrings. Dit komt omdat we tijdens het vergelijken van twee tokens zowel met de naam, als met de attributen van de tag rekening moeten houden. Enkel wanneer de naam en de attributen van beide tags overeenkomen, zijn de tokens hetzelfde. Voor Illustratie 9.1 betekent dit dat er geen gemeenschappelijke token-suffix is voor kandidaat l_1. Omwille van deze reden gaan we de tag attributen negeren, zodat we een gemeenschappelijke token-suffix voor kandidaat l_1 behouden, namelijk "⇓". Deze gemeenschappelijke token-suffix zal leiden tot twee kandidaten voor het scheidingsteken l_1, namelijk "" en "⇓". De kortste kandidaat voor l_1 bestaat dus uit één token, oftewel 6 karakters. Dit in tegenstelling tot de oorspronkelijke kortste kandidaat die één karakter lang was, namelijk ">". Tokenstrings leiden, naast minder kandidaten, bijgevolg ook tot "langere" korte kandidaten. Deze "langere" korte kandidaten hebben bovendien meer kans om de zuiver suffix validatie te doorstaan, aangezien deze minder snel als een normale suffix zullen voorkomen in de tekst voor een attribuutwaarde. Dit betekent dat tokenstrings tot minder foutieve kandidaten leiden.

De keuze om tag attributen te negeren brengt een eerste aanname met zich mee, namelijk dat de relationele data die we uit de webpagina willen halen niet mag voorkomen in een tag. Een tweede aanname die we maken, stelt dat de tags gestructureerd zijn. Dit betekent dat wanneer één attribuutwaarde van attribuut *k* tussen "" tags staat, alle attribuutwaarden voor attribuut *k* tussen "" tags moeten staan.

Wanneer we de relatie van Illustratie 9.2 uit de webpagina van Illustratie 9.1 willen halen, dan voldoet Illustratie 9.1 aan beiden aannames. In Illustratie 9.3 wordt een variant op de broncode van Illustratie 9.1 gegeven. In deze variant is het eerste land tussen "" tags vermeld, terwijl het tweede land tussen " " tags is vermeld. Hierdoor wordt onze tweede aanname niet gerespecteerd. Indien we uit deze broncode de relatie van Illustratie 9.4 willen halen, dan wordt ook de eerste aanname niet gerespecteerd, aangezien de telefooncodes enkel in de attributen van de tags voorkomen.

$$R_c = \begin{Bmatrix} \langle \text{'Belgium'} \rangle \\ \langle \text{'Congo'} \rangle \end{Bmatrix}$$

ILLUSTRATIE 9.2: DE RELATIE DIE WE UIT DE WEBPAGINA VAN ILLUSTRATIE 9.1 WILLEN HALEN.

```
1: <html><head><title class='title'>Some Countries</title></head><body>
2: <span id='32' class='country'>Belgium</span><br>
3: <div class='country' id='243'>Congo</div><br>
4: </body></html>
```

ILLUSTRATIE 9.3: BRONCODE VAN EEN WEBPAGINA, DIE NIET AAN ONZE AANNAMES IN VERBAND MET TOKENSTRINGS VOLDOET.

$$R_{cc} = \begin{Bmatrix} \langle \text{'32','Belgium'} \rangle \\ \langle \text{'243','Congo'} \rangle \end{Bmatrix}$$

ILLUSTRATIE 9.4: DE RELATIE DIE WE UIT DE WEBPAGINA VAN ILLUSTRATIE 9.3 WILLEN HALEN.

Tot slot merken we op dat de attributen van een tag kunnen bijdragen tot het vinden van de correcte attribuutwaarden voor een attribuut van een relatie. In Illustratie 9.1 zien we bijvoorbeeld dat in beide "`<span>`" tags het attribuut "`class='country'`" wordt vermeld. Aan de hand van deze informatie zouden we alle landen eenvoudig kunnen terugvinden op een webpagina. Dit "`class`" attribuut wordt dan echter als metadata gebruikt en dit valt, net zoals semantiek, buiten de focus van deze masterproef.

10. EXPERIMENTEN

In dit hoofdstuk onderzoeken we de bruikbaarheid van de wrapper classes in de praktijk. Dit doen we door een aantal experimenten uit te voeren. Tijdens deze experimenten gaan we een aantal parameters bestuderen, waaronder de uitvoeringstijd, de correctheid van de gevonden wrappers, etc. Deze parameters helpen ons de bruikbaarheid van de wrapper classes en bijgevolg ook de bruikbaarheid van de wrapper inductietechniek te bepalen.

We beginnen dit hoofdstuk met een beschrijving van de opstelling van de experimenten. Vervolgens bespreken we de resultaten. Tot slot eindigen we met de conclusies die we uit de experimenten kunnen trekken.

Meer informatie over de ontwikkelde programma's, waarmee we de wrapper classes kunnen testen, is terug te vinden in Bijlage C.

10.1 OPSTELLING

Vooraleer we de experimenten daadwerkelijk kunnen uitvoeren, moeten we nadenken over de opstelling van deze experimenten. We moeten immers weten wat onze test cases zijn. Verder hebben we verschillende verzamelingen van webpagina's nodig die de dataset vormen van onze experimenten. We moeten eveneens beslissen welke parameters we tijdens de testen willen bestuderen. Deze moeten zo gekozen worden dat we achteraf zoveel mogelijk informatie kunnen verzamelen die nuttig kan zijn voor de vorming van conclusies. We dienen ook rekening te houden met het systeem waarop we de experimenten uitvoeren, aangezien het systeem de experimenten kan beïnvloeden.

10.1.1 TEST CASES

De wrapper classes die in de experimenten aan bod komen zijn:

- LR
- HLRT
- OCLR
- HOCLRT
- UOCLR

De condities waaronder de wrapper classes getest worden zijn:

- Geen verbeteringen actief (G)
- Enkel de "gemeenschappelijke strings" verbetering actief (GS)
- Enkel de "tokenstrings" verbetering actief (TS)
- "Gemeenschappelijke strings" en "tokenstrings" verbetering actief (GS-TS)

De verzamelingen van webpagina's waarop de wrapper classes onder de verschillende condities getest zullen worden, komen van de volgende drie websites:

- dblp
- IMDb
- Wikipedia

Het learn algoritme van een wrapper class wordt onder elke conditie op elke website getest. Dit levert een totaal van 60 test cases op (5 wrapper classes x 4 condities x 3 websites) voor de verschillende learn algoritmes.

De exec procedure van een wrapper class wordt ook onder elke conditie op elke website getest. Dit levert een maximum van 60 test cases op (5 wrapper classes x 4 condities x 3 websites) voor de verschillende exec procedures. We spreken hier van een maximum omdat sommige learn test cases misschien geen geldige wrapper vinden, waardoor we de bijhorende exec test cases niet kunnen uitvoeren.

10.1.2 Dataset / Websites

De dataset voor onze experimenten bestaat uit drie verzamelingen van webpagina's, met elke verzameling komende van een andere website. Twee van deze websites, IMDb en Wikipedia, worden veelvuldig bezocht door het grote publiek. De derde website, dblp, wordt door een specifiekere groep personen veelvuldig bezocht. We hebben voor deze websites gekozen omdat ze grote hoeveelheden informatie bevatten waarin veel mensen geïnteresseerd zijn. De technieken die we in deze experimenten testen hebben tot doel juist deze informatie uit webpagina's te halen. Elke website wordt hieronder in detail besproken.

De website dblp[3] biedt bibliografische informatie aan over informatica bronnen, waaronder conferenties, papers, tijdschriften, etc. De notatie van informatie op deze website is niet alleen verschillend tussen de verscheidene soorten bronnen, maar ook binnen een bron kunnen soms verschillende notaties gehanteerd worden. Vanwege deze reden selecteren wij enkel webpagina's van één enkele tijdschriftenreeks, namelijk "ACM Transactions on Database Systems (TODS)"[4]. Deze reeks bestaat tot op heden reeds uit 39 volumes, waardoor we bijgevolg 39 webpagina's hebben voor onze verzameling webpagina's van de website dblp. In Figuur 10.1 (pag. 97) wordt een webpagina van deze tijdschriftenreeks weergegeven. In Illustratie 10.1 (pag. 100) worden de eerste drie tupels weergegeven die we uit de relatie van deze webpagina willen halen.

[3] http://www.informatik.uni-trier.de/~ley/db/
[4] http://www.informatik.uni-trier.de/~ley/db/journals/tods/

De website IMDb[5] (Internet Movie Database) biedt informatie aan over mediaobjecten of gerelateerd aan mediaobjecten. Enkele voorbeelden van mediaobjecten zijn films, tv series, games, etc. Tot de gerelateerde informatie behoort onder andere acteurs/actrices, producers, reviews, ratings, etc. De webpagina van een mediaobject heeft een URL van de vorm "http://www.imdb.com/title/tt1154120/", waarbij "tt1154120" het specifieke mediaobject aanduidt. Dankzij de consistente vorm van de URL's, is het eenvoudig om een web crawl uit te voeren op deze website. We hebben bijgevolg 10.000 willekeurige URL's gegenereerd, waarna we geprobeerd hebben om de bijhorende webpagina's op te vragen. Voor 123 URL's gaf de web crawl geen resultaat terug. Deze webpagina's konden met andere woorden niet opgevraagd worden, waardoor we een totaal van 9.877 effectieve webpagina's bekomen voor onze verzameling. In Figuur 10.2 (pag. 98) wordt een webpagina weergegeven die zich in de verzameling bevindt. Illustratie 10.2 (pag. 100) toont de volledige relatie die we uit deze webpagina willen halen.

De derde en laatste website is Wikipedia[6], een vrije meertalige internetencyclopedie die door meerdere auteurs op vrijwillige basis wordt geschreven. Voor deze website focussen we op de tracklistings van muziekalbums. Figuur 10.3 (pag. 99) toont de webpagina die de tracklist van het album "Teenage Dream" van de artiest "Katy Perry" bevat. In de tracklist tabel zien we 12 songs gevolgd door vier uitklapbare boxen met bonus tracks. Wij zijn enkel geïnteresseerd in "main" songs. Illustratie 10.3 (pag. 100) toont de drie eerste tupels die we uit de relatie van de webpagina van Figuur 10.3 willen halen. Voor onze verzameling van webpagina's van de website Wikipedia hebben we 15 willekeurige webpagina's van populaire artiesten gekozen.

We merken op dat de relaties van Illustratie 10.1, Illustratie 10.2 en Illustratie 10.3 geen meta-informatie bevatten, bijvoorbeeld een link naar de webpagina van een artiest op Wikipedia. Deze informatie wordt in sommige gevallen echter wel door onze algoritmes mee opgenomen, aangezien dit nuttige informatie is die tot de attribuutwaarden behoort. Dergelijke meta-informatie laat de gebruiker immers toe gedetailleerde informatie van een attribuutwaarde op te vragen, weliswaar niet meer relationeel, maar indien gewenst interessant.

Nu we besproken hebben hoeveel en welke soort webpagina's horen bij de betreffende websites, moeten we enkel nog bepalen hoeveel webpagina's we per website gebruiken als invoer voor een learn algoritme. We gaan elke verzameling van webpagina's als het ware in twee opdelen, namelijk een leer-set en een uitvoer-set. De leer-set wordt gebruikt voor het leren van de wrappers. De uitvoer-set wordt gebruikt om te controleren hoe doeltreffend deze wrappers zijn. We hebben deze opdeling manueel uitgevoerd, daar

[5] http://www.imdb.com/
[6] http://en.wikipedia.org/

we willen dat de webpagina's van de leer-set zo verschillend mogelijk zijn. Op deze manier zorgen we ervoor dat een wrapper de correcte relatie uit zoveel mogelijk webpagina's van de uitvoer-set kan halen. Voor het selecteren van de pagina's voor de leer-set kijken we naar de visuele structuur van de webpagina's van een website. We kijken met andere woorden naar de gerenderde webpagina's in de browser en niet naar de broncode. Naar de broncode van een webpagina kijken zou namelijk te omslachtig zijn en hierdoor zouden we bovendien al een aantal stappen van het learn algoritme manueel aan het uitvoeren zijn. Enkele visuele indicatoren waarop we letten zijn: de plaats van de relatie op de webpagina, de opmaak tussen attribuutwaarden, de opmaak tussen tupels, de tekst die voor en achter de relatie voorkomt, de aanwezigheid van tussentitels in de relatie, etc. We selecteren altijd minimum drie webpagina's om te vermijden dat pagina specifieke tekst opgenomen wordt in een scheidingsteken.

De webpagina's van dblp lijken allemaal dezelfde visuele structuur te hebben, waardoor we 3 willekeurige webpagina's selecteren voor de leer-set van dblp.

Bij IMDb hebben niet alle webpagina's dezelfde structuur. Sommige webpagina's hebben namelijk een andere titel voor de tabel waarin we geïnteresseerd zijn. Daarnaast hebben sommige webpagina's tussentitels in de tabel (zie Figuur 10.2, pag. 98), terwijl andere dit niet hebben. We hebben 5 verschillende structuurvormen gevonden over een steekproef van 25 willekeurige webpagina's. We selecteren bijgevolg 5 willekeurige webpagina's, met elk een andere structuurvorm, voor de leer-set van IMDb.

Bij Wikipedia hebben we 3 verschillende structuurvormen gevonden, namelijk uitklapbare boxen met bonus tracks aanwezig, geen uitklapbare boxen met bonus tracks aanwezig, en bonus tracks die niet in een uitklapbare box staan. We selecteren bijgevolg 3 willekeurige webpagina's, met elk een andere structuurvorm, voor de leer-set van Wikipedia.

10.1.3 Parameters

We willen conclusies trekken op het vlak van twee gebieden, namelijk expressiviteit en efficiëntie. Om dit te kunnen doen moeten we parameters bestuderen die relevant zijn voor deze twee gebieden. Deze parameters worden hieronder per gebied besproken.

De expressiviteit van een wrapper class kunnen we afleiden uit het aantal benodigde webpagina's voor het leren van een wrapper voor een website, en uit het aantal webpagina's waaruit de wrapper class de correcte relatie kan halen met de gevonden wrapper voor de website waartoe de webpagina's behoren. Beide parameters zijn met elkaar gerelateerd, aangezien het aantal webpagina's waaruit de correcte relatie gehaald wordt, beïnvloed wordt door de webpagina's waaruit de wrapper geleerd wordt. Het

uitvoeren van de exec procedure op elke webpagina in de uitvoer-set en het verifiëren of de exec procedure wel degelijk de correcte relatie uit de webpagina heeft gehaald, zal ons bijgevolg de informatie opleveren die we nodig hebben om onze conclusies te kunnen trekken op het vlak van expressiviteit. Indien we beschikken over de correcte relatie van elke webpagina, dan kunnen we de verificatie automatiseren. Deze relaties zijn echter niet beschikbaar, wat betekent dat we ze zelf moeten genereren. Dit zou echter te veel werk zijn binnen de tijdslimieten van deze masterproef, aangezien we beschikken over ±10.000 webpagina's. Omwille van deze reden gaan we de output van elke exec test case voor slechts vijf pagina's manueel verifiëren. Naast het manueel verifiëren gaan we de resultaten van de exec procedures van de verschillende wrapper classes voor een bepaalde website met elkaar vergelijken. Deze vergelijking kunnen we wel op een geautomatiseerde manier uitvoeren, door gebruik te maken van een meetcoëfficiënt. De meetcoëfficiënt die wij hanteren is de Jaccard Index, waarmee we de gelijkheid van sets kunnen meten. Een set komt overeen met een webpagina en de waarden van een set zijn de tupels die een exec procedure uit de webpagina heeft gehaald.

De efficiënte van een wrapper class gaan we bepalen aan de hand van drie parameters, namelijk de uitvoeringstijd van het learn algoritme, de uitvoeringstijd van de exec procedure en het aantal kandidaten voor elk scheidingsteken van de wrapper behorende tot de wrapper class.

10.1.4 Systeemspecificaties

Voor het uitvoeren van de experimenten werd gebruik gemaakt van een Sony Vaio VPCF12M1E laptop. Deze laptop bezit een 64 bit Intel(R) Core(TM) i5-520M processor met een kloksnelheid van 2,40 GHz en een turbo kloksnelheid van 2,933 GHz. Deze processor beschikt over twee cores met elk twee threads. Verder beschikt de laptop over 4GB RAM. De harde schijf van 500GB draait met 7200 toeren per minuut.

De experimenten zijn uitgevoerd onder een 32bit Ubuntu besturingssysteem (versie 13.10), dat beschikt over de Linux kernel 3.11.0-24-generic. Dit besturingssysteem heeft 32GB van de totale schijfruimte ter beschikking. Naast de standaard 4GB RAM is er in Ubuntu ook nog 256MB swap voorzien.

De code is gecompileerd door middel van de g++ compiler (versie 4.8.1). Daarnaast is ook gebruik gemaakt van de libxml2 library (versie 2.9.1).

10.1.5 Bemerkingen

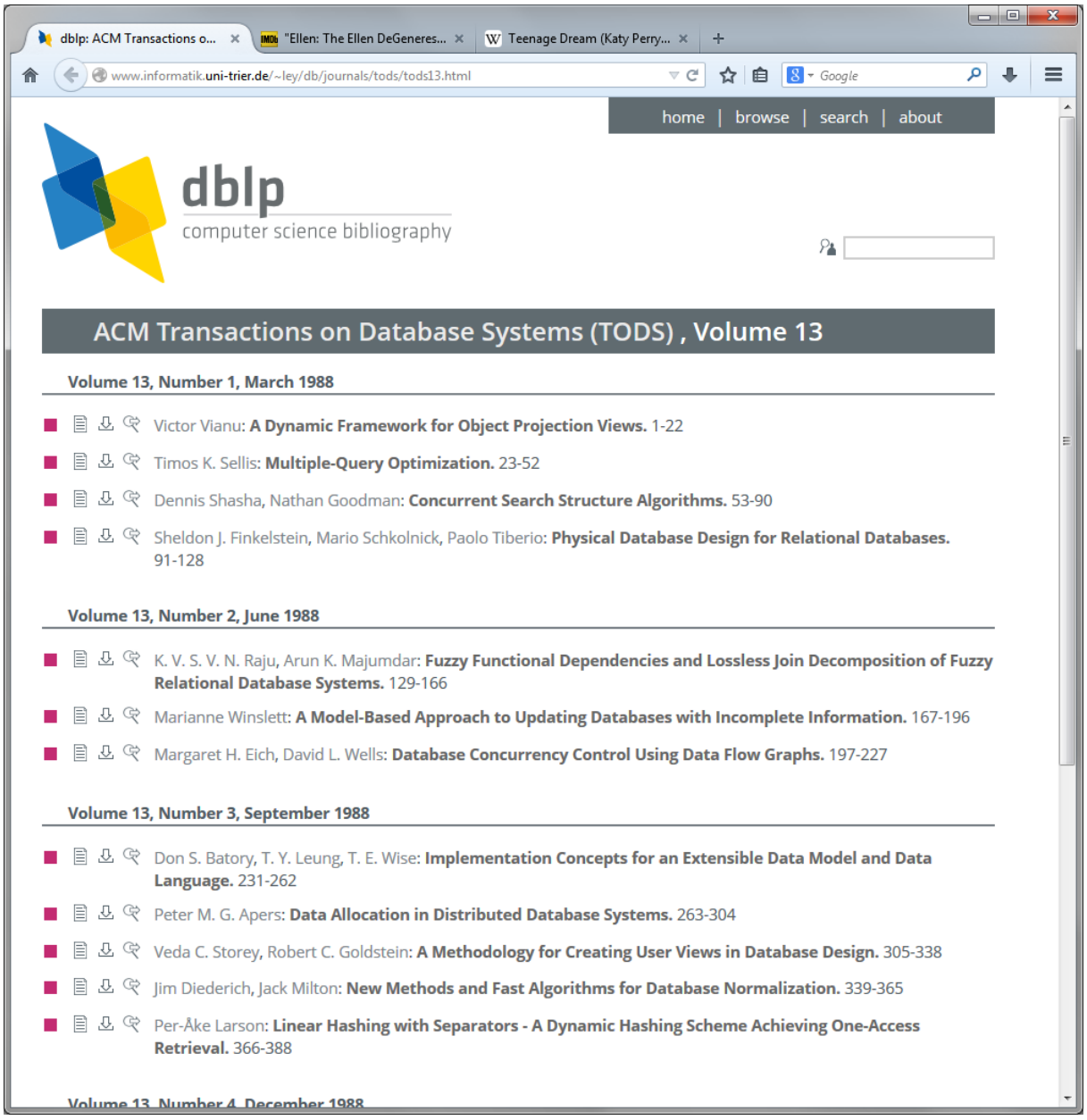
Experimenten die korter dan tien minuten duren zijn altijd in isolatie uitgevoerd, terwijl experimenten die langer dan tien minuten duren hoogstens per twee uitgevoerd zijn. De testmachine werd tijdens de experimenten nooit gebruikt voor andere doeleinden. Het in

paren uitvoeren van experimenten vormde geen probleem, aangezien de processor beschikt over twee cores. Invloeden van I/O waren ook niet aanwezig, aangezien alle nodige data telkens aan het begin van het programma werd ingelezen. Tijdens het uitvoeren van de learn test cases werd wel aan voortgang I/O gedaan. Elke keer dat een learn algoritme 50.000 kandidaat-combinaties had bekeken, werd dit gemeld.

Vanwege de beperkte hoeveelheid RAM geheugen, was het niet altijd mogelijk om de generalized suffix tree (Sectie 9.1) op te bouwen. Dit was het geval bij de set van paginahoofden en paginavoeten, vanwege onze grote webpagina's. De kleinste leer-pagina (komende van dblp) is namelijk 31.059 tekens lang en de grootste leer-pagina (komende van Wikipedia) is zelfs 335.786 tekens lang. We hebben dit probleem opgelost door een restrictie te leggen op de lengte van elke string/tokenstring, die zich in de set waaruit de generalized suffix tree wordt opgebouwd bevind. De maximum lengte bedraagt 4.000 tekens/tokens. Op deze manier blijft het geheugengebruik van het learn algoritme altijd onder de 1,5GB, waardoor we twee learn algoritmes naast elkaar kunnen uitvoeren en nog steeds voldoende RAM over houden voor de noodzakelijke taken van het OS. Bij een paginahoofd behouden we telkens de laatste 4.000 tekens/tokens. Bij een paginavoet behouden we telkens de eerste 4.000 tekens/tokens.

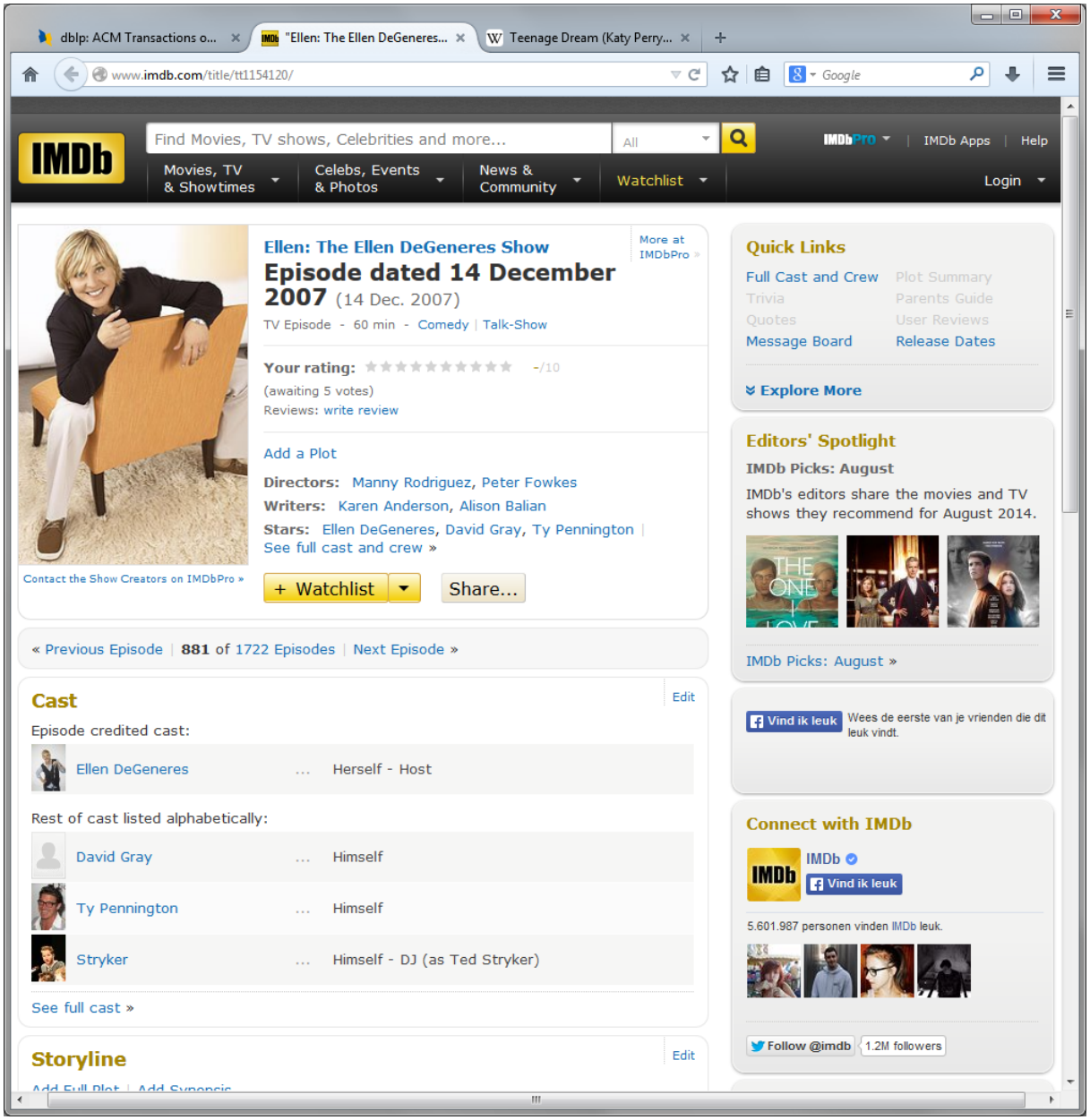
Indien we alle kandidaten voor een scheidingsteken op voorhand genereren, dan kunnen we alle duplicaten uit de lijst met kandidaten filteren. Dubbele kandidaten komen voor bij de scheidingstekens *h*, *t*, *o* en *c*. Bij deze scheidingstekens worden kandidaten namelijk gegenereerd door alle substrings van de kortste string (of van de gemeenschappelijke substrings, zie de "Gemeenschappelijke strings" verbetering in Sectie 9.1) te nemen. De string "`<i>32</i>`" levert in dergelijk geval twee keer de substring "`i>`" op. De lijst met kandidaten voor een scheidingsteken kan echter zo lang worden dat deze niet meer in het geheugen past. Hierdoor zijn we genoodzaakt alle kandidaten dynamisch één voor één te genereren. Deze techniek heeft echter als nadeel dat we duplicaten niet meer uit de lijst kunnen filteren.

De laatste bemerking heeft betrekking op de uitvoeringstijd van learn algoritmes. Deze uitvoeringstijd hangt af van het aantal kandidaat-combinaties dat doorlopen moet worden. Als er geen geldige wrapper bestaat, moet het algoritme zelfs alle kandidaat-combinaties doorlopen. Het aantal kandidaat-combinaties kan echter zeer snel hoog oplopen, zoals we reeds gezien hebben doorheen de hoofdstukken van de verschillende wrapper classes. (Herinner dat het aantal kandidaat-combinaties van een wrapper class is gelijk aan het product van het aantal kandidaten van elk van de interagerende scheidingstekens.) Omwille van deze reden laten we een learn algoritme maximum ±9 uur uitvoeren. We zeggen ± omdat we het algoritme handmatig stoppen.



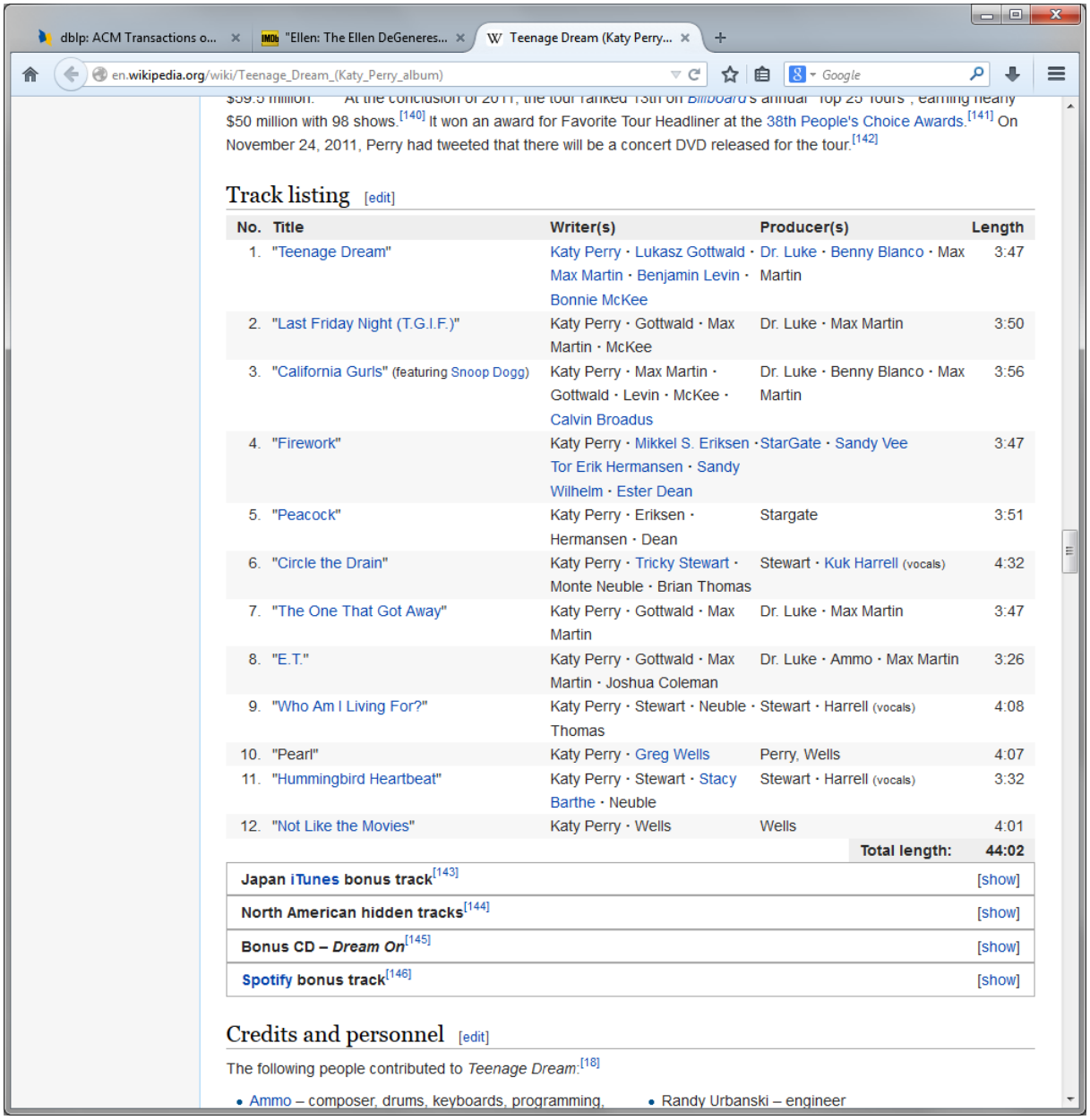
FIGUUR 10.1: WEBPAGINA DIE ZICH IN DE VERZAMELING VAN WEBPAGINA'S VAN DE WEBSITE DBLP BEVINDT. DE RELATIE OP DEZE WEBPAGINA BESTAAT UIT 3 ATTRIBUTEN EN BEVAT 17 TUPELS, WAARVAN 5 TUPELS NIET ZICHTBAAR ZIJN IN DEZE SCREENSHOT. ILLUSTRATIE 10.1 GEEFT DE EERSTE DRIE TUPELS VAN DE RELATIE IN TABELVORM WEER.

HTTP://WWW.INFORMATIK.UNI-TRIER.DE/~LEY/DB/JOURNALS/TODS/TODS13.HTML



FIGUUR 10.2: WEBPAGINA DIE ZICH IN DE VERZAMELING VAN WEBPAGINA'S VAN DE WEBSITE IMDB BEVINDT. DE RELATIE OP DEZE WEBPAGINA BESTAAT UIT 2 ATTRIBUTEN EN BEVAT 4 TUPELS. ILLUSTRATIE 10.2 GEEFT DE VOLLEDIGE RELATIE IN TABELVORM WEER.

HTTP://WWW.IMDB.COM/TITLE/TT1154120/



FIGUUR 10.3: WEBPAGINA DIE ZICH IN DE VERZAMELING VAN WEBPAGINA'S VAN DE WEBSITE WIKIPEDIA BEVINDT. DE RELATIE OP DEZE WEBPAGINA BESTAAT UIT 5 ATTRIBUTEN EN BEVAT 12 TUPELS. ILLUSTRATIE 10.3 GEEFT DE EERSTE DRIE TUPELS VAN DEZE RELATIE IN TABELVORM WEER.

HTTP://EN.WIKIPEDIA.ORG/WIKI/TEENAGE_DREAM_(KATY_PERRY_ALBUM)

Authors	Article name	Pages
Victor Vianu	A Dynamic Framework for Object Projection Views.	1-22
Timos K. Sellis	Multiple-Query Optimization.	23-52
Dennis Shasha, Nathan Goodman	Concurrent Search Structure Algorithms.	53-90

ILLUSTRATIE 10.1: DE EERSTE DRIE TUPELS VAN DE RELATIE DIE WE UIT DE WEBPAGINA VAN FIGUUR 10.1 WILLEN HALEN. DE TABELHOOFDING WORDT NIET UIT DE WEBPAGINA GEHAALD, MAAR WORDT HIER WEERGEGEVEN OM DE ATTRIBUUTNAMEN AAN TE DUIDEN.

Actor/Actress	Role
Ellen DeGeneres	Herself - Host
David Gray	Himself
Ty Pennington	Himself
Stryker	Himself – DJ (as Ted Stryker)

ILLUSTRATIE 10.2: DE VOLLEDIGE RELATIE DIE WE UIT DE WEBPAGINA VAN FIGUUR 10.2 WILLEN HALEN. DE TABELHOOFDING WORDT NIET UIT DE WEBPAGINA GEHAALD, MAAR WORDT HIER WEERGEGEVEN OM DE ATTRIBUUTNAMEN AAN TE DUIDEN.

No.	Title	Writer(s)	Producer(s)	Length
1.	"Teenage Dream"	Katy Perry, Lukasz Gottwald, Max Martin, Benjamin Levin, Bonnie McKee	Dr. Luke, Benny Blanco, Max Martin	3:47
2.	"Last Friday Night (T.G.I.F.)"	Katy Perry, Gottwald, Max Martin, McKee	Dr. Luke, Max Martin	3:50
3.	"California Gurls" (featuring Snoop Dogg)	Katy Perry, Max Martin, Gottwald, Levin, McKee, Calvin Broadus	Dr. Luke, Benny Blanco, Max Martin	3:56

ILLUSTRATIE 10.3: DE EERSTE DRIE TUPELS VAN DE RELATIE DIE WE UIT DE WEBPAGINA VAN FIGUUR 10.3 WILLEN HALEN. DE TABELHOOFDING WORDT NIET UIT DE WEBPAGINA GEHAALD, MAAR WORDT HIER WEERGEGEVEN OM DE ATTRIBUUTNAMEN AAN TE DUIDEN.

10.2 RESULTATEN

Nu we de opstelling van onze experimenten besproken hebben, kunnen we verder gaan met de analyse van de resultaten. We beginnen met het bespreken van het aantal kandidaten voor elk scheidingsteken. Daarna bespreken we de resultaten van de learn test cases, gevolgd door de resultaten van de exec test cases. Tot slot vergelijken we onze resultaten met die van Kushmerick [11].

10.2.1 AANTAL KANDIDATEN VOOR ELK SCHEIDINGSTEKEN

De tabel in Illustratie 10.5 (pag. 112) geeft een overzicht van het aantal kandidaten voor elk scheidingsteken onder elke conditie ten opzichte van de leer-pagina's van elke website. Merk op dat deze waarden afhankelijk zijn van de hoeveelheid leer-pagina's en de gekozen leer-pagina's (Sectie 10.1.2). Een andere hoeveelheid en/of andere leer-pagina's resulteert bijgevolg in andere waarden. In deze sectie bespreken we enkele interessante relaties tussen de verschillende scheidingsteken die we uit de resultaten van de tabel kunnen afleiden.

In deze tabel zien we dat het aantal kandidaten voor de scheidingstekens l_1, r_K, *h*, *t*, *o* en *c* kleiner is onder de "GS" conditie dan onder de "G" conditie. Dit betekent dat de kortste string waaruit de kandidaten voor een scheidingsteken gegenereerd worden, voor elke van deze scheidingstekens, pagina specifieke karakters bevat. (Het scheidingsteken *t* van dblp vormt hier een uitzondering op, omdat de voet van een webpagina van dblp altijd dezelfde is.) Hetzelfde kan niet gezegd worden van de tekst tussen de attribuutwaarden. De scheidingstekens r_1 tot en met l_K ondergaan namelijk geen reductie op vlak van het aantal kandidaten. De tekst tussen de attribuutwaarden van een tupel is met andere woorden dezelfde voor elk tupel van een website.

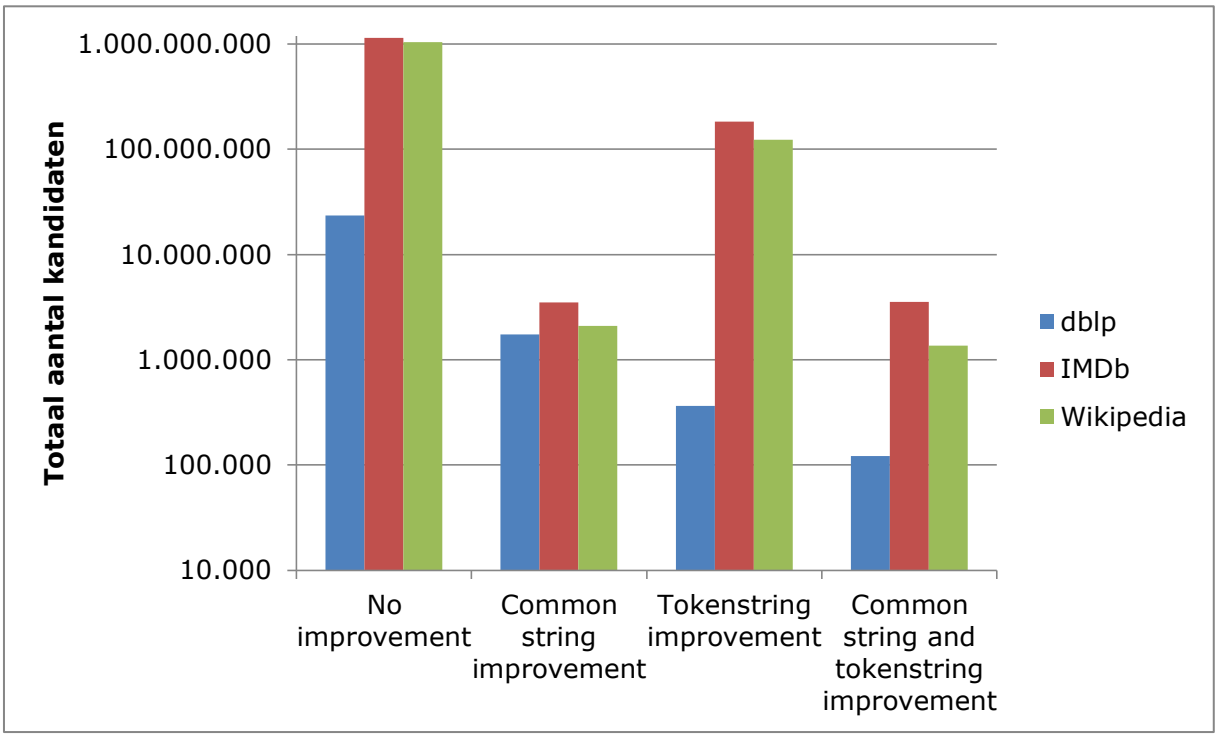
De "TS" conditie zorgt, ten opzichte van de "G" conditie, voor een reductie in het aantal kandidaten voor elk scheidingsteken. Voor de *h*, *t*, *o* en *c* scheidingstekens is deze reductie zelfs aanzienlijk. Dit wijst er op dat er veel tag attributen aanwezig zijn in de tags.

Het bovenstaande besproken fenomeen dat zich voordoet tussen de "G" conditie en de "GS" conditie, doet zich ook voor tussen de scheidingstekens van de "TS" conditie en de "GS-TS" conditie. De kandidaat *t* van dblp, en de kandidaten *o* & *c* van Wikipedia vormen echter een uitzondering. Het aantal kandidaten voor deze drie scheidingstekens blijft namelijk hetzelfde. Bij de *t* van dblp werd dit veroorzaakt door de voet die op elke webpagina van dblp hetzelfde is, zoals hierboven reeds werd aangehaald. Voor de twee scheidingstekens van Wikipedia heeft dit te maken de tag attributen die genegeerd worden bij tokenstrings. Het zijn net deze tag attributen die niet gemeenschappelijk waren bij de "G" en "GS" condities.

Tussen de "GS" en "GS-TS" conditie zien we ook een reductie in alle kandidaten, met uitzondering van kandidaat *t* van IMDb. Deze kandidaat ondergaat namelijk een stijging in het aantal kandidaten. Dit lijkt op het eerste zicht misschien een onlogische situatie, aangezien de lengte van een tokenstring kleiner of gelijk is aan de lengte van een gewone string. Toch is deze situatie correct. Het negeren van tag attributen kan namelijk tot meer gemeenschappelijke substrings leiden, omdat tags die voorheen verschillend waren (vanwege verschillende tag attributen/attribuutwaarden) nu hetzelfde zijn.

Aangezien de "GS" en de "TS" conditie voor een daling in het aantal kandidaten zorgt ten opzichte van de "G" conditie, zal ook de combinatie van deze twee verbeteringen (de "GS-TS" conditie) voor een daling in het aantal kandidaten zorgen ten opzichte van de "G" conditie.

Tussen de "GS" conditie en de "TS" conditie is er geen relatie. Dit is vanzelfsprekend, aangezien dit twee verbeteringen zijn die los staan van elkaar.



FIGUUR 10.4: TOTAAL AANTAL KANDIDATEN VOOR DE LEER-PAGINA'S VAN ELKE WEBSITE ONDER ELKE CONDITIE. DE LEER-SET VAN DBLP BEVAT 3 LEER-PAGINA'S. DE LEER-SET VAN IMDB BEVAT 5 LEER-PAGINA'S. DE LEER-SET VAN WIKIPEDIA BEVAT 3 LEER-PAGINA'S.

In Figuur 10.4 zien we het totaal aantal kandidaten voor de leer-pagina's van elke website onder elke conditie weergegeven in een grafiek (groepering per wrapper class). Het totaal wordt berekend door de sommatie van alle kandidaten voor de leer-pagina's van een website onder één van de vier conditie te nemen. De grafiek laat duidelijk zien dat elke verbetering een vooruitgang is, waarbij de gemeenschappelijke strings

verbetering de grootste verbetering laat optekenen. Daarnaast zien we ook dat de combinatie van beide verbeteringen een grotere impact heeft dan één enkele verbetering.

10.2.2 Resultaten learn test cases

In Illustratie 10.6 (pag. 113) wordt de uitvoeringstijd weergegeven van elke test case die betrekking heeft op een learn algoritme. Illustratie 10.7 (pag. 114) toont voor dezelfde test cases het aantal doorlopen kandidaat-combinaties ten opzichte van het totaal aantal kandidaat-combinaties. (Herinner dat het aantal kandidaat-combinaties van een wrapper class is gelijk aan het product van het aantal kandidaten van alle interagerende scheidingstekens.) De betekenis van de celkleuren in beide tabellen is als volgt: het learn algoritme heeft een wrapper gevonden (groen), het learn algoritme heeft geen wrapper gevonden (rood), en het learn algoritme is afgebroken na een looptijd van ±9 uur (oranje). Een aantal oranje cellen zijn groen gearceerd. Dit betekent dat deze test cases ooit een wrapper zullen vinden, maar hier wordt later in deze sectie dieper op ingegaan. Merk op dat er geen waarden voor de LR wrapper class in Illustratie 10.7 aanwezig zijn. Dit vanwege het feit dat er geen scheidingstekens interageren bij de LR wrapper class, waardoor er bijgevolg ook geen kandidaat-combinaties zijn voor de LR wrapper class.

Illustratie 10.8 (pag. 115) toont het totaal aantal wrapper mogelijkheden voor elke learn test case die betrekking heeft op de LR wrapper class. Dit is de som van het aantal kandidaten van alle linker en rechter scheidingstekens horende bij de test case. Om het totaal aantal wrapper mogelijkheden van de HLRT, OCLR of HOCLRT wrapper class te achterhalen voor een learn test case, dient men het totaal aantal wrapper mogelijkheden voor de LR wrapper class onder dezelfde conditie voor dezelfde website, bij het totaal aantal kandidaat-combinaties van de HLRT, OCLR of HOCLRT wrapper class op te tellen. Voor de UOCLR wrapper class is het totaal aantal kandidaat-combinaties gelijk aan het totaal aantal wrapper mogelijkheden, aangezien alle scheidingstekens met elkaar interageren bij de UOCLR wrapper class. We geven geen tabel met het totaal aantal wrapper mogelijkheden, omdat het verschil tussen het totaal aantal kandidaat-combinaties en het totaal aantal wrapper mogelijkheden verwaarloosbaar is.

Het valt onmiddellijk op dat veel test cases niet eindigen binnen de voorziene tijdslimiet. Dit is te wijten aan het grote aantal kandidaat-combinaties voor elke test case. Bij de test cases die eindigen zijn maar weinig kandidaat-combinaties doorlopen alvorens een geldige wrapper werd gevonden. In totaal zijn 23 van de 60 learn test cases geëindigd, waarvan 17 succesvol. Het valt ook op dat de meeste test cases eindigen onder een conditie die de gemeenschappelijke strings verbetering bevat (conditie “GS” en “GS-TS”).

Wanneer een test case niet binnen de voorziene tijdslimiet eindigt, betekent dit niet dat er geen geldige wrapper bestaat voor die test case. Voor sommige niet geëindigde test cases kunnen we afleiden of ze na een langere looptijd wel succesvol zullen eindigen. Hiervoor kunnen we gebruik maken van Stelling 7.1 van Sectie 7.4 en van Stelling 9.1 van Sectie 9.1. De eerste stelling stelt dat wanneer een test case onder de "GS" conditie een geldige wrapper vindt, de "G" conditie uiteindelijk dezelfde wrapper zal vinden. Dezelfde redenering geldt ook tussen de "GS-TS" conditie en "TS" conditie. De tweede stelling stelt dat wanneer een test case voor de HLRT wrapper class, onder eender welke conditie voor eender welke website, een geldige wrapper vindt, er ook een geldige wrapper zal gevonden worden voor de test case onder de HOCLRT wrapper class, onder dezelfde conditie en voor dezelfde website. Als we beide stellingen toepassen op onze resultaten, dan kunnen we afleiden dat onderstaande test cases ooit succesvol zullen eindigen, mits ze voldoende tijd ter beschikking krijgen. De cellen in Illustratie 10.6 (pag. 113) en Illustratie 10.7 (pag. 114) horende bij deze test cases zijn groen gearceerd.

- (dblp, G, HLRT)
- (dblp, G, OCLR)
- (dblp, G, HOCLRT)
- (dblp, CS, HOCLRT)
- (dblp, TS, HLRT)
- (dblp, TS, OCLR)
- (dblp, TS, HOCLRT)
- (IMDb, G, HLRT)
- (IMDb, G, OCLR)
- (IMDb, G, HOCLRT)
- (IMDb, CS, HOCLRT)
- (IMDb, TS, HLRT)
- (IMDb, TS, HOCLRT)
- (IMDb, CS-TS, HOCLRT)
- (Wikipedia, G, HLRT),
- (Wikipedia, G, HOCLRT)
- (Wikipedia, CS, HOCLRT)

Dankzij Stelling 9.2 van Sectie 9.1 kunnen we afleiden welke niet geëindigde test cases uiteindelijk niet succesvol zullen eindigen. De stelling stelt namelijk dat indien een test case onder de "GS" conditie geen wrapper vindt, de "G" conditie ook geen wrapper zal vinden. Dezelfde redenering geldt ook tussen de "GS-TS" en "TS" conditie. Deze methode geeft voor onze resultaten echter geen bijkomend inzicht.

We merken op dat er nog verschillende andere afleidingen bestaan, maar deze zijn minder bruikbaar. Dit vanwege het feit dat deze afleidingen vertrekken van een wrapper class die een langere tijd nodig heeft om te eindigen, dan de wrapper class waarvoor we het resultaat kunnen afleiden. Bijvoorbeeld: een niet succesvol resultaat voor de HOCLRT wrapper class impliceert dat er ook geen succesvol resultaat is voor de HLRT en voor de OCLR wrapper class.

Het learn algoritme van de LR wrapper class is voor elke website onder elke conditie geëindigd. Voor dblp en IMDb was dit iedere keer succesvol, maar voor Wikipedia was dit nooit het geval. Wanneer we de broncode van alle voorbeeldpagina's van Wikipedia bestuderen, wordt al snel duidelijk dat de andere tabellen op de pagina's de boosdoener zijn. Deze andere tabellen hebben namelijk dezelfde structuur. Een geldige wrapper zal bijgevolg een head en tail scheidingsteken nodig hebben om de juiste tabel te kunnen isoleren. Dit verklaart waarom de HLRT wrapper class wel een geldige wrapper voor Wikipedia kan vinden. De nood aan een head en tail scheidingsteken impliceert dat er ook geen geldige OCLR wrapper bestaat voor Wikipedia. De HLRT wrapper class vindt echter alleen een HLRT wrapper onder de "GS" conditie. Dit betekent dat er ook een geldige HLRT wrapper bestaat onder de "G" conditie, zoals we hierboven reeds besproken hebben. Of er een geldige HLRT wrapper bestaat onder een conditie die de tokenstrings verbetering bevat, kan niet afgeleid worden uit onze experimenten. Het is mogelijk dat de "GS" conditie uitgebreid gebruik maakt van tag attributen die niet meer aanwezig zijn in een conditie die gebruik maakt van de tokenstrings verbetering. Het bestuderen van de broncode van Wikipedia leidt echter niet tot meer inzicht in deze situatie.

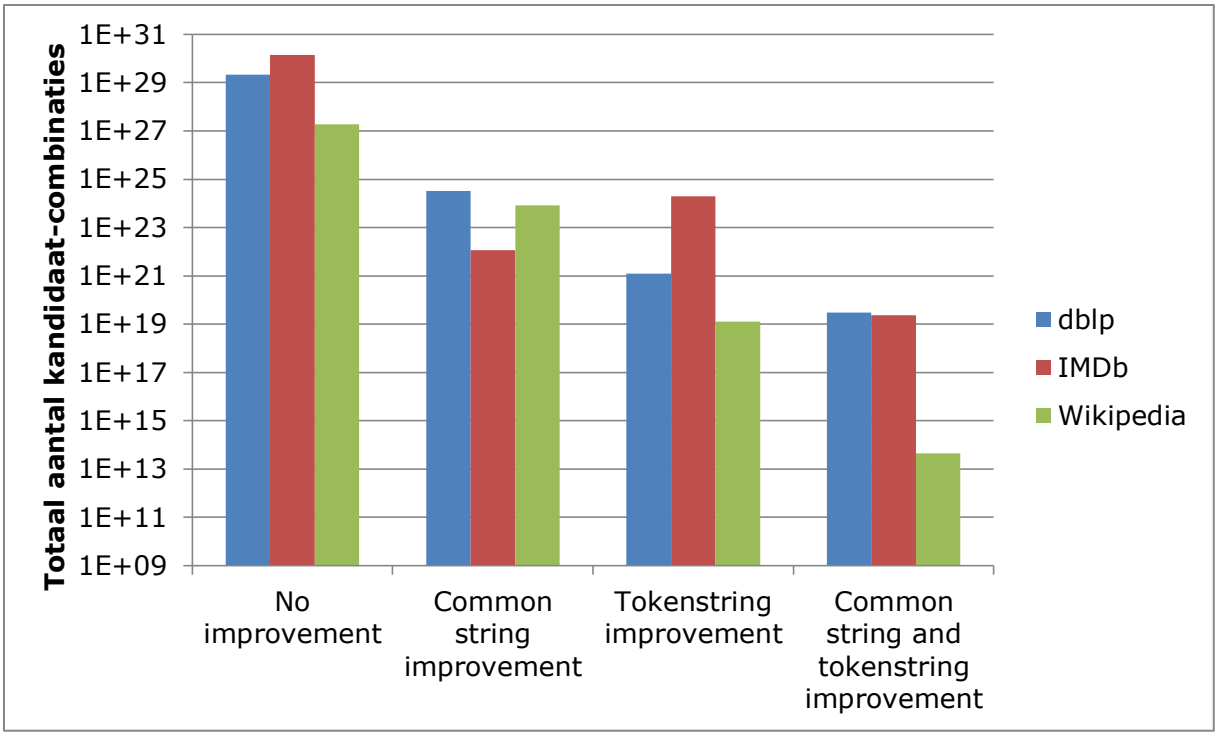
Geen enkele test case die betrekking heeft op de UOCLR wrapper class is geëindigd. De strenge constraints waaraan de scheidingstekens moeten voldoen, spelen hierin een rol. Deze hebben namelijk tot gevolg dat er minder geldige wrappers zijn, waardoor er langer gezocht moet worden alvorens een geldige wrapper gevonden wordt. Het bestuderen van de broncode van de webpagina's in de leer-set van dblp en IMDb geeft geen uitsluitsel over het al dan niet bestaan van een UOCLR wrapper. De broncode van de webpagina's in de leer-set van de website Wikipedia levert wel meer inzicht op. De verschillende attribuutwaarden worden namelijk omgeven door dezelfde tags, waardoor aan de "uniek"-constraints van de UOCLR wrapper class nooit voldaan kan worden.

Over het aantal doorlopen kandidaat-combinaties kunnen we weinig zeggen, aangezien dit aantal afhangt van de hoeveelheid kandidaten en de kwaliteit van de kandidaten. Kandidaat-combinaties die aan de meerderheid van de constraints voldoen, maar bij een latere constraint toch falen, nemen namelijk veel nutteloze tijd in beslag. De hoeveelheid kandidaten en de kwaliteit hebben dus ook een directe invloed op de uitvoeringstijd. Daarnaast is de hoeveelheid kandidaten en de kwaliteit van de kandidaten ook nog eens

afhankelijk van de webpagina's in de leer-set. We kunnen echter wel concluderen dat het aantal doorlopen kandidaat-combinaties enorm weinig is ten opzichte van het totaal aantal kandidaat-combinaties. Bij de test cases die niet binnen de voorziene tijdslimiet eindigden wordt gemiddeld 0,16% van het totaal aantal kandidaat-combinaties doorlopen. Dit gemiddelde wordt echter sterk beïnvloed door de niet eindigende OCLR wrapper class test cases van Wikipedia. De mediaan ligt namelijk veel lager, op $3{,}65\times10^{-10}$% om precies te zijn. Bij de test cases die wel binnen de voorziene tijdslimiet eindigen wordt gemiddeld $3{,}28\times10^{-5}$% van het totaal aantal kandidaat-combinaties doorlopen. De test cases die niet binnen de voorziene tijdslimiet eindigen, zitten ver boven dit percentage.

Uit de uitvoeringstijden van de test cases die betrekking hebben op de LR wrapper class, blijkt dat de gemeenschappelijke strings verbetering de uitvoeringstijd verlaagt en de tokenstrings verbetering de uitvoeringstijd verhoogt. Dit is te wijten aan het feit dat de gemeenschappelijke strings verbetering zorgt voor minder foutieve kandidaten en dat de tokenstrings verbetering een HTML parsing stap bevat. Voor de andere wrapper classes is de uitvoeringstijd soms sneller onder de "GS-TS" conditie, dan onder de "GS" conditie. De HLRT en OCLR wrapper class bij de website dblp zijn hier twee voorbeelden van. Onder de "GS-TS" conditie bij de website dblp wordt tevens een HOCLRT wrapper gevonden binnen de tijdslimiet. Dankzij Stelling 7.1 van Sectie 7.4 weten we dat er ook een geldige HOCLRT wrapper bestaat onder de "GS" conditie voor de website dblp, zoals we hierboven reeds besproken hebben. Deze wrapper werd echter niet gevonden binnen de tijdslimiet. Het omgekeerde doet zich ook voor, namelijk dat de uitvoeringstijd voor een wrapper class soms sneller is onder de "GS" conditie, dan onder de "GS-TS" conditie. De test case die betrekking heeft op de HLRT wrapper class bij de website IMDb is hier een voorbeeld van. Daarbovenop zijn er twee wrappers gevonden onder de "GS" conditie, die niet gevonden zijn onder de "GS-TS" conditie. Dit is het geval bij de OCLR wrapper class van IMDb en bij de HLRT wrapper class van Wikipedia. We kunnen het resultaat onder de "GS-TS" conditie ook niet voorspellen in deze gevallen. Tag attributen kunnen bijgevolg zowel een negatief als positief effect hebben op het sneller vinden van geldige wrappers bij alle wrapper classes met uitzondering van LR.

Figuur 10.5 toont een grafiek waarin het totaal aantal kandidaat-combinaties voor elke website gegroepeerd is per conditie. De groepering gebeurt door de sommatie van alle totaal aantal kandidaat-combinaties van elke wrapper class van een website onder een bepaalde conditie. De grafiek toont een dalend patroon, maar zoals we hierboven reeds besproken hebben kan de tokenstrings verbetering zowel een positief als negatief effect hebben op het vinden van een geldige wrapper.

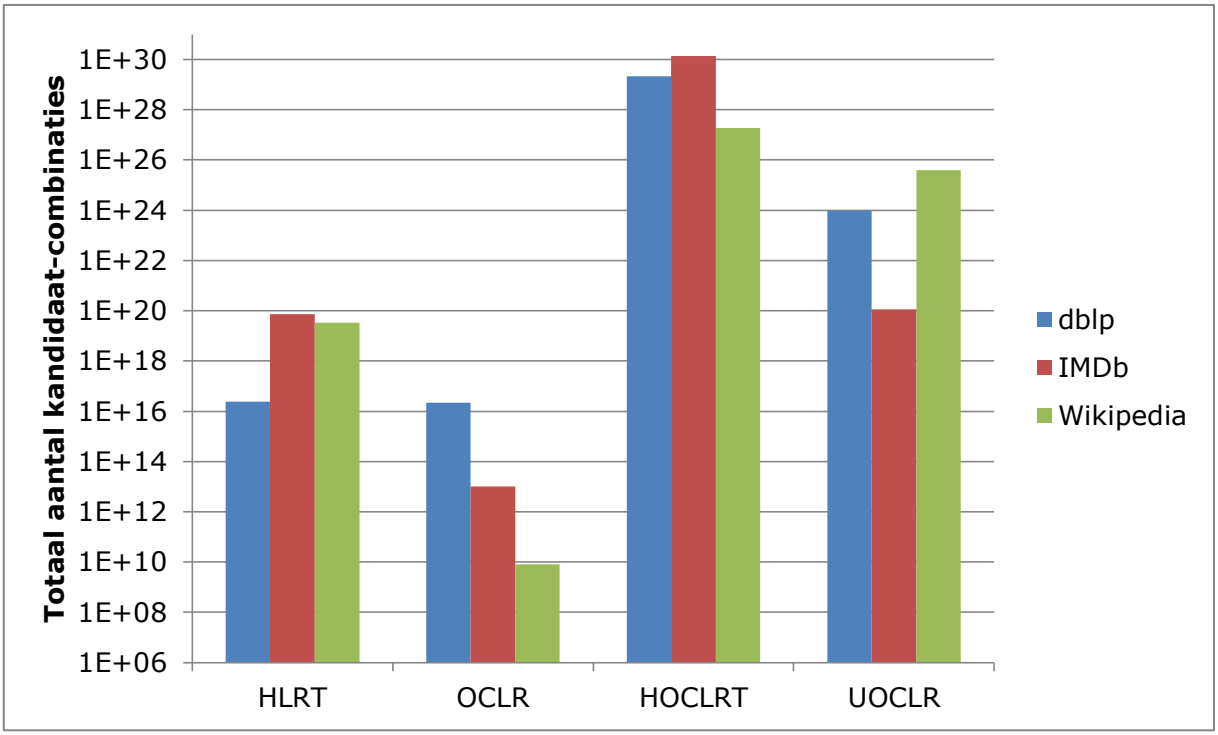


FIGUUR 10.5: TOTAAL AANTAL KANDIDAAT-COMBINATIES VOOR ELKE WEBSITE GEGROEPEERD PER CONDITIE. KANDIDAAT-COMBINATIES WORDEN BEREKEND DOOR GEBRUIK TE MAKEN VAN ALLE WEBPAGINA'S IN DE LEER-SET VAN EEN WEBSITE. DE LEER-SET VAN DBLP BEVAT 3 LEER-PAGINA'S. DE LEER-SET VAN IMDB BEVAT 5 LEER-PAGINA'S. DE LEER-SET VAN WIKIPEDIA BEVAT 3 LEER-PAGINA'S.

Figuur 10.6 toont een grafiek waarin het totaal aantal kandidaat-combinaties van een website gegroepeerd is per wrapper class. De groepering gebeurt door de sommatie van alle totaal aantal kandidaat-combinaties van een wrapper class van een website onder elke conditie nemen. Deze grafiek geeft een mooi overzicht van de wrapper classes en hun aantal kandidaat-combinaties, waarbij de HOCLRT wrappers class als duidelijke leider naar voren komt.

10.2.3 RESULTATEN EXEC TEST CASES

In Sectie 10.1.3 hebben we vermeld dat we de resultaten van de exec procedures van de verschillende wrapper classes onder een conditie voor een website onderling met elkaar gaan vergelijken. Hiervoor hebben we minimum twee wrappers nodig onder een conditie voor een website. Dit is het geval bij de "GS" conditie voor de website dblp, de "GS-TS" conditie voor de website dblp, de "GS" conditie voor de website IMDb en de "GS-TS" conditie voor de website IMDb. De vergelijking werd telkens gedaan door middel van een Jaccard Index. De resultaten voor deze vier situaties zijn respectievelijk terug te vinden in Illustratie 10.9, Illustratie 10.10, Illustratie 10.11 en Illustratie 10.12 (pag. 116 en 117). In elke illustratie zijn enkel de waarden boven de diagonaal ingevuld, omdat de waarden onder de diagonaal het spiegelbeeld hiervan zijn. De laatste kolom geeft de



FIGUUR 10.6: TOTAAL AANTAL KANDIDAAT-COMBINATIES VOOR ELKE WEBSITE, VOOR ELKE WRAPPER CLASS. KANDIDAAT-COMBINATIES WORDEN BEREKEND DOOR GEBRUIK TE MAKEN VAN ALLE WEBPAGINA'S IN DE LEER-SET VAN EEN WEBSITE. DE LEER-SET VAN DBLP BEVAT 3 LEER-PAGINA'S. DE LEER-SET VAN IMDB BEVAT 5 LEER-PAGINA'S. DE LEER-SET VAN WIKIPEDIA BEVAT 3 LEER-PAGINA'S.

uitvoeringstijd weer die nodig was om de exec procedure van de wrapper class die vermeld is op de regel, uit te voeren op alle webpagina's in de uitvoer-set.

Voor we beginnen met het beschrijven van de resultaten in bovenvermelde illustraties, merken we op dat een LR wrapper onder de "G" conditie voor een website dezelfde is als de LR wrapper onder de "GS" conditie voor dezelfde website. Dit volgt uit Stelling 9.1 van Sectie 9.1 en geldt eveneens tussen de "TS" conditie en "GS-TS" conditie. Omwille van deze reden worden LR wrappers, die alleen zijn onder een "G" conditie of "TS" conditie voor een website, niet apart besproken.

Illustratie 10.9 toont de Jaccard Indexen tussen de wrappers LR, HLRT en OCLR onder de "GS" conditie voor de website dblp. De Jaccard Index is altijd 1, wat betekent dat elke wrapper dezelfde relatie uit elke webpagina van de uitvoer-set haalt. Merk op dat dit niet wil zeggen dat ze de correcte relatie uit elke webpagina halen. Ze kunnen namelijk allemaal eenzelfde fout maken. Voor de vijf willekeurige webpagina's van de uitvoer-set die we manueel verifiëren, halen ze de correcte relatie uit elke webpagina. Dit is een goede indicatie dat ze uit elke webpagina van de uitvoer-set de correcte relatie halen.

Illustratie 10.10 toont de Jaccard Indexen tussen de wrappers LR, HLRT, OCLR en HOCLRT onder de "GS-TS" conditie voor de website dblp. De Jaccard Index is ook hier

altijd 1, wat betekent dat elke wrapper dezelfde relatie uit elke webpagina van de uitvoer-set haalt. Merk op dat dit niet wil zeggen dat ze de correcte relatie uit elk webpagina halen. Ze kunnen namelijk allemaal eenzelfde fout maken. Voor de vijf willekeurige webpagina's van de uitvoer-set die we manueel verifiëren, wordt ook hier telkens de correcte relatie uit elke webpagina gehaald. Dit is een goede indicatie dat ze uit elke webpagina van de uitvoer-set de correcte relatie halen.

Wanneer we de uitvoeringstijden van Illustratie 10.9 en Illustratie 10.10 met elkaar vergelijken, dan zien we dat deze langer zijn onder de "GS-TS" conditie, dan onder de "G" conditie. Dit is te wijten aan de HTML parsing stap die wordt uitgevoerd door de tokenstrings verbetering.

Illustratie 10.11 toont de Jaccard Indexen tussen de wrappers LR, HLRT en OCLR onder de "GS" conditie voor de website IMDb. Dit keer is de Jaccard Index enkel 1 tussen de LR en HLRT wrapper. Tussen de LR/HLRT en OCLR is de Jaccard index 0,8679. Dit betekent dat er pagina's in de uitvoer-set aanwezig zijn waarvoor de LR en HLRT wrapper een andere relatie halen uit de webpagina halen dan de OCLR wrapper. Dit impliceert ook dat er minstens één wrapper is die niet de correcte relatie uit elke webpagina haalt. Twee van de vijf willekeurige webpagina's van de uitvoer-set die we manueel verifiëren, bevatten ontbrekende attribuutwaarden. Voor deze twee webpagina's gaan alle drie de wrappers in de fout. Voor de eerste webpagina met ontbrekende attribuutwaarden halen de LR en de HLRT wrapper dezelfde foutieve relatie uit de webpagina. De OCLR wrapper haalt eveneens dezelfde foutieve relatie uit de webpagina en probeert hierbij nog één extra tupel uit de pagina te halen waarin hij desondanks toch faalt. Voor de tweede webpagina met ontbrekende attribuutwaarden halen alle drie de wrappers dezelfde foutieve relatie uit de pagina. Wanneer we de broncode van de twee webpagina's met ontbrekende attribuutwaarden bestuderen, zien we dat een deel van het linker scheidingsteken van de ontbrekende attribuutwaarden ontbreekt. Het grootste deel van het scheidingsteken is echter nog intact. Indien we een webpagina met enkele ontbrekende attribuutwaarden opnemen in onze leer-set, dan zouden de wrappers hiermee overweg kunnen. Voor de andere drie van de vijf willekeurige webpagina's van de uitvoer-set, die we manueel verifiëren halen, de LR, HLRT en OCLR de correcte relatie uit de pagina.

Illustratie 10.12 toont de Jaccard Indexen tussen de LR en HLRT wrapper onder de "GS-TS" conditie voor de website IMDb. De Jaccard Index benadert de 1. Dit betekent dat er een paar pagina's in de uitvoer-set aanwezig zijn waaruit de LR en HLRT wrapper class een andere relatie halen. Voor de vijf willekeurige webpagina's van de uitvoer-set die we manueel verifiëren, halen ze uit drie webpagina's de correcte relatie (zoals in de vorige alinea), maar voor de twee pagina's met de ontbrekende attributen, gaan ze allebei op

dezelfde manier als hierboven in de fout. Omdat we geen voorbeeld hebben van een webpagina waaruit de LR en HLRT wrapper elk een andere relatie halen, hebben we bijgevolg ook geen idee hoe dit komt.

Wanneer we de uitvoeringstijden van Illustratie 10.11 en Illustratie 10.12 met elkaar vergelijken, zien we dat deze langer zijn onder de "GS-TS" conditie, dan onder de "G" conditie. Dit is, net zoals bij de vorige twee illustraties, opnieuw te wijten aan de HTML parsing stap die wordt uitgevoerd door de tokenstrings verbetering.

Voor de website Wikipedia hebben we maar één geldige wrapper. Omdat onze uitvoer-set voor deze website van beperkte grootte is, namelijk 12 pagina's, hebben we besloten een manuele verificatie uit te voeren voor elke webpagina in de uitvoer-set. Het resultaat is echter zeer slecht. Uit 10 van de 12 webpagina's werd namelijk geen relatie gehaald door de HLRT wrapper. Het bestuderen van de broncode leert ons dat het tail scheidingsteken na het head scheidingsteken en voor het eerste linker scheidingsteken voorkomt, waardoor de exec procedure voor deze 10 webpagina's te vroeg eindigt. Uit een andere webpagina haalt de exec procedure naast de correcte relatie ook veel ongewenste data op, waaronder stukken van bonus tracks. De broncode vertelt ons dat het scheidingsteken r_K ontbreekt achter het laatste tupel van de "main" tracks. Uit de laatste webpagina van de uitvoer-set wordt wel de correcte relatie gehaald. Deze situatie illustreert hoe belangrijk het is om goede webpagina's te selecteren voor de leer-set. Zelfs al selecteren we webpagina's die een verschillende visuele structuur hebben, zie Sectie 10.1.2, dan nog kunnen de webpagina's dezelfde broncode structuur hebben. In ons geval hadden we ook veel tegenslag gehad, aangezien sommige van de 10 webpagina's waaruit geen relatie werd gehaald, dezelfde visuele structuur hadden als de webpagina's in onze leer-set. We hadden met andere woorden evengoed één van die pagina's kunnen selecteren en dan had dit probleem zich niet voorgedaan.

10.2.4 Vergelijking met resultaten van Kushmerick

Voor we conclusies uit onze eigen experimenten trekken, gaan we onze resultaten vergelijken met die van Kushmerick [11]. We hebben de standaard LR, HLRT, OCLR en HOCLRT wrapper class namelijk overgenomen van Kushmerick. We hebben deze wrapper classes echter zelf moeten implementeren, aangezien zijn code niet beschikbaar is. Een vergelijking tussen zijn resultaten en die van ons is bijgevolg op zijn plaats.

Wanneer we de resultaten van Kushmerick bekijken valt het onmiddellijk op dat wanneer hij een geldige LR wrapper vindt voor de leer-set van een website, hij ook een geldige HLRT, OCLR en HOCLRT wrapper vindt voor dezelfde leer-set. Een geldige wrapper wordt altijd in minder dan 400 seconden gevonden, met uitzondering van twee HOCLRT wrappers waarbij de uitvoeringstijd van het learn algoritme langer dan 900 seconden

duurde. (Exacte tijden zijn niet bekend wanneer de uitvoeringstijd langer dan 900 seconden bedraagt.) De conditie die in onze experimenten overeenkomt met de omstandigheden waaronder Kushmerick zijn experimenten heeft uitgevoerd, is "G". In deze conditie worden echter alleen maar LR wrappers gevonden. De verklaring hiervoor vinden we terug in de complexiteit van de wrapper classes. De complexiteit van elke wrapper class die we van Kushmerick [11] hebben overgenomen, wordt weergegeven in Illustratie 10.4. Hierin staat *K* voor het aantal attributen dat elk tupel van eender welke pagina in de leer-set bevat, *M* voor het totaal aantal tupels van alle pagina's in de leer-set, $|\mathcal{E}|$ voor het aantal pagina's in de leer-set en *V* voor de langste pagina in de leer-set. De variabele *V* heeft de grootste invloed op de complexiteit van elke wrapper class, met uitzondering voor de LR wrapper class. Bij de experimenten van Kushmerick ligt de waarde van *V* altijd tussen 899 en 57.116 tekens. Voor onze experimenten ligt de waarde van *V* tussen 54.084 en 335.786 tekens. Dit verklaart waarom wij voor de LR wrapper class wel een geldige wrapper kunnen vinden, terwijl wij binnen afzienbare tijd (±9 uur) geen geldige wrapper vinden voor de HLRT, OCLR en HOCLRT wrapper class (onder de "G" conditie weliswaar). Dit terwijl Kushmerick in dergelijk geval wel een geldige wrapper voor de HLRT, OCLR en HOCLRT wrapper class kon vinden binnen een beperkte tijd. De grootte van de pagina's speelt namelijk geen grote rol bij de LR wrapper class, maar voor de overige wrapper classes wordt dit echter wel de dominerende factor, en dit met een macht van minimum 6.

Wrapper Class	Complexity
LR	$O(K \times M^2 \times \|\mathcal{E}\|^2 \times V^2)$
HLRT	$O(K \times M^2 \times \|\mathcal{E}\|^4 \times V^6)$
OCLR	$O(K \times M^4 \times \|\mathcal{E}\|^2 \times V^6)$
HOCLRT	$O(K \times M^4 \times \|\mathcal{E}\|^4 \times V^{10})$

ILLUSTRATIE 10.4: COMPLEXITEIT VAN ELKE WRAPPER CLASS DIE WE VAN KUSHMERICK HEBBEN OVERGENOMEN. [10]

Door dit verschil in grootte van de grootste pagina in de leer-set kunnen weinig andere resultaten van Kushmerick vergeleken worden met onze resultaten. Daarnaast is er bijvoorbeeld weinig tot geen gedetailleerde informatie over de hoeveelheid kandidaten voor de scheidingstekens, waardoor het nog moeilijker wordt om te kunnen vergelijken.

		No improvement	Common string improvement	Tokenstring improvement	Common string and tokenstring improvement
dblp	l_1	2.502	164	261	107
	r_1	22	22	3	3
	l_2	22	22	3	3
	r_2	9	9	3	3
	l_3	9	9	3	3
	r_3	1.019	13	170	3
	h	16.799.706	776.846	282.376	74.076
	t	519.690	519.690	14.535	14.535
	o	3.131.253	224.171	34.191	16.369
	c	3.131.253	224.171	34.191	16.369
IMDb	l_1	526	57	73	53
	r_1	146	146	84	84
	l_2	146	146	84	84
	r_2	526	37	73	24
	h	1.004.214.520	149.836	161.307.741	41.416
	t	136.827.153	3.323.354	22.811.635	3.499.054
	o	138.601	19.988	2.701	1.731
	c	138.601	19.988	2.701	1.731
Wikipedia	l_1	122	77	7	7
	r_1	57	57	3	3
	l_2	57	57	3	3
	r_2	39	39	3	3
	l_3	39	39	3	3
	r_3	39	39	3	3
	l_4	39	39	3	3
	r_4	79	79	3	3
	l_5	79	79	3	3
	r_5	122	15	7	7
	h	471.106.860	301.509	99.991.011	6.123
	t	575.503.701	1.781.696	23.048.655	1.359.903
	o	7.503	3.949	28	28
	c	7.503	3.949	28	28

ILLUSTRATIE 10.5: OVERZICHT VAN HET AANTAL KANDIDATEN VOOR ELK SCHEIDINGSTEKEN ONDER ELKE CONDITIE TEN OPZICHTE VAN DE LEER-PAGINA'S VAN ELKE WEBSITE. VOOR DE WEBSITE DBLP ZIJN ER 3 LEER-PAGINA'S EN BESTAAT ELKE RELATIE UIT 3 ATTRIBUTEN. VOOR DE WEBSITE IMDB ZIJN ER 5 LEER-PAGINA'S EN BESTAAT ELKE RELATIE UIT 2 ATTRIBUTEN. VOOR DE WEBSITE WIKIPEDIA ZIJN ER 3 LEER–PAGINA'S EN BESTAAT ELKE RELATIE UIT 5 ATTRIBUTEN.

		No improvement	Common string improvement	Tokenstring improvement	Common string and tokenstring improvement
dblp	*LR*	0,051 sec	0,021 sec	0,062 sec	0,057 sec
	HLRT		15,040 sec		1,047 sec
	OCLR		22,898 sec		9,666 sec
	HOCLRT				83 min 45,995 sec
	UOCLR				
IMDb	*LR*	0,247 sec	0,046 sec	2,056 sec	0,122 sec
	HLRT		41,446 sec		3 min 7,801 sec
	OCLR		1 min 24,083 sec		
	HOCLRT				
	UOCLR				
Wikipedia	*LR*	0,070 sec	0,053 sec	0,111 sec	0,113 sec
	HLRT		70 min 18,543 sec		
	OCLR			2,572 sec	2,602 sec
	HOCLRT				
	UOCLR				

ILLUSTRATIE 10.6: UITVOERINGSTIJDEN VAN ELKE LEARN TEST CASE. EEN LEARN TEST CASE HOUDT IN DAT WE HET LEARN ALGORITME VAN EEN WRAPPER CLASS UITVOEREN OP ALLE PAGINA'S IN DE LEER-SET VAN EEN WEBSITE ONDER EEN BEPAALDE CONDITIE. ELKE CEL KOMT BIJGEVOLG OVEREEN MET EEN LEARN TEST CASE. DE LEER-SET VAN DBLP BEVAT 3 LEER-PAGINA'S. DE LEER-SET VAN IMDB BEVAT 5 LEER-PAGINA'S. DE LEER-SET VAN WIKIPEDIA BEVAT 3 LEER-PAGINA'S.

		No improvement	Common string improvement	Tokenstring improvement	Common string and tokenstring improvement
dblp	*LR*				
	HLRT	$3,69\times10^{9}$ / $2,18\times10^{16}$	19 / $6,62\times10^{13}$	$6,47\times10^{8}$ / $1,07\times10^{12}$	6 / $1,15\times10^{11}$
	OCLR	$2,57\times10^{9}$ / $2,45\times10^{16}$	23.131 / $8,24\times10^{12}$	$1,31\times10^{8}$ / $3,05\times10^{11}$	16.373 / $2,86\times10^{10}$
	HOCLRT	$2,66\times10^{9}$ / $2,14\times10^{29}$	$1,21\times10^{9}$ / $3,32\times10^{24}$	$1,30\times10^{8}$ / $1,25\times10^{21}$	10.444.057 / $3,08\times10^{19}$
	UOCLR	$2,49\times10^{9}$ / $9,80\times10^{23}$	$2,69\times10^{9}$ / $4,20\times10^{18}$	$2,40\times10^{8}$ / $4,20\times10^{15}$	$2,60\times10^{8}$ / $6,96\times10^{12}$
IMDb	*LR*				
	HLRT	$1,28\times10^{8}$ / $7,22\times10^{19}$	3 / $2,83\times10^{13}$	$1,22\times10^{7}$ / $2,68\times10^{17}$	1 / $7,68\times10^{12}$
	OCLR	$1,33\times10^{8}$ / $1,01\times10^{13}$	43.416 / $2,27\times10^{10}$	$7,00\times10^{5}$ / $5,32\times10^{8}$	$4,50\times10^{5}$ / $1,58\times10^{8}$
	HOCLRT	$1,32\times10^{8}$ / $1,38\times10^{30}$	$2,83\times10^{7}$ / $1,13\times10^{22}$	$7,00\times10^{5}$ / $1,95\times10^{24}$	$3,00\times10^{5}$ / $2,30\times10^{19}$
	UOCLR	$4,88\times10^{8}$ / $1,13\times10^{20}$	$2,75\times10^{8}$ / $1,79\times10^{16}$	$3,15\times10^{6}$ / $2,74\times10^{14}$	$1,35\times10^{6}$ / $2,68\times10^{13}$
Wikipedia	*LR*				
	HLRT	$2,29\times10^{7}$ / $3,30\times10^{19}$	2.419.317 / $4,13\times10^{13}$	$9,24\times10^{7}$ / $1,61\times10^{16}$	$6,26\times10^{7}$ / $5,82\times10^{10}$
	OCLR	$8,64\times10^{7}$ / $6,86\times10^{9}$	$6,93\times10^{7}$ / $1,20\times10^{9}$	5.488 / 5.488	5.488 / 5.488
	HOCLRT	$9,71\times10^{7}$ / $1,86\times10^{27}$	$7,33\times10^{7}$ / $6,45\times10^{20}$	$4,79\times10^{7}$ / $1,26\times10^{19}$	$5,79\times10^{7}$ / $4,56\times10^{13}$
	UOCLR	$5,24\times10^{9}$ / $3,93\times10^{25}$	$4,14\times10^{9}$ / $8,44\times10^{23}$	$1,14\times10^{8}$ / $2,52\times10^{8}$	$1,13\times10^{8}$ / $2,52\times10^{8}$

ILLUSTRATIE 10.7: HET AANTAL DOORLOPEN KANDIDAAT-COMBINATIES TEN OPZICHTE VAN HET TOTAAL AANTAL KANDIDAAT-COMBINATIES VOOR ELKE LEARN TEST CASE. DEZE WAARDEN ZIJN AFHANKELIJK VAN DE HOEVEELHEID LEER-PAGINA'S EN DE GESELECTEERDE LEER-PAGINA'S DIE WE HEBBEN GEKOZEN VOOR ELKE LEARN TEST CASE (SECTIE 10.1.2).

		No improvement	Common string improvement	Tokenstring improvement	Common string and tokenstring improvement
dblp	*LR*	3.583	239	443	122
IMDb	*LR*	1.344	386	314	245
Wikipedia	*LR*	672	520	38	38

ILLUSTRATIE 10.8: AANTAL MOGELIJKE LR WRAPPERS VOOR ELKE LEARN TEST CASE DIE BETREKKING HEEFT OP DE LR WRAPPER CLASS. DEZE WAARDEN ZIJN AFHANKELIJK VAN DE HOEVEELHEID LEER-PAGINA'S EN DE GESELECTEERDE LEER-PAGINA'S DIE WE HEBBEN GEKOZEN VOOR ELKE LEARN TEST CASE (SECTIE 10.1.2).

dblp *Common string improvement*

	LR	HLRT	OCLR	HOCLRT	UOCLR	Time
LR	x					0,22 sec
HLRT	1	x				0,38 sec
OCLR	1	1	x			0,26 sec
HOCLRT	-	-	-	x		-
UOCLR	-	-	-	-	x	-
						Total: 0,86 sec

ILLUSTRATIE 10.9: JACCARD INDEXEN TUSSEN DE WRAPPERS VOOR DE WEBSITE DBLP ONDER DE "GEMEENSCHAPPELIJKE STRINGS VERBETERING" CONDITIE. EEN JACCARD INDEX GEEFT AAN HOEVEEL GELIJKENIS ER IS TUSSEN DE OUTPUT VAN DE EXEC PROCEDURE VAN TWEE WRAPPERS OVER ALLE WEBPAGINA'S IN DE UITVOER-SET. DE LAATSTE KOLOM VAN DEZE TABEL GEEFT VOOR ELKE WRAPPER CLASS DE UITVOERINGSTIJD WEER DIE NODIG WAS OM DE EXEC PROCEDURE UIT TE VOEREN OP ALLE WEBPAGINA'S IN DE UITVOER-SET VAN DBLP.

dblp *Common string and tokenstring improvement*

	LR	HLRT	OCLR	HOCLRT	UOCLR	Time
LR	x					1,36 sec
HLRT	1	x				1,44 sec
OCLR	1	1	x			1,33 sec
HOCLRT	1	1	1	x		1,33 sec
UOCLR	-	-	-	-	x	-
						Total: 5,46 sec

ILLUSTRATIE 10.10: JACCARD INDEXEN TUSSEN DE WRAPPERS VOOR DE WEBSITE DBLP ONDER DE "GEMEENSCHAPPELIJKE STRINGS EN TOKENSTRINGS VERBETERING" CONDITIE. EEN JACCARD INDEX GEEFT AAN HOEVEEL GELIJKENIS ER IS TUSSEN DE OUPUT VAN DE EXEC PROCEDURE VAN TWEE WRAPPERS OVER ALLE WEBPAGINA'S IN DE UITVOER-SET. DE LAATSTE KOLOM VAN DEZE TABEL GEEFT VOOR ELKE WRAPPER CLASS DE UITVOERINGSTIJD WEER DIE NODIG WAS OM DE EXEC PROCEDURES UIT TE VOEREN OP ALLE WEBPAGINA'S IN DE UITVOER-SET VAN DBLP.

IMDb Common string improvement

	LR	HLRT	OCLR	HOCLRT	UOCLR	Time
LR	x					46,54 sec
HLRT	1	x				49,10 sec
OCLR	0,8679	0,8679	x			46,21 sec
HOCLRT	-	-	-	x		-
UOCLR	-	-	-	-	x	-
						Total: 141,85 sec

ILLUSTRATIE 10.11: JACCARD INDEXEN TUSSEN DE WRAPPERS VOOR DE WEBSITE IMDB ONDER DE "GEMEENSCHAPPELIJKE STRINGS VERBETERING" CONDITIE. EEN JACCARD INDEX GEEFT AAN HOEVEEL GELIJKENIS ER IS TUSSEN DE OUTPUT VAN DE EXEC PROCEDURE VAN TWEE WRAPPERS OVER ALLE WEBPAGINA'S IN DE UITVOER-SET. DE LAATSTE KOLOM VAN DEZE TABEL GEEFT VOOR ELKE WRAPPER CLASS DE UITVOERINGSTIJD WEER DIE NODIG WAS OM DE EXEC PROCEDURE UIT TE VOEREN OP ALLE WEBPAGINA'S IN DE UITVOER-SET VAN IMDB.

IMDb Common string and tokenstring improvement

	LR	HLRT	OCLR	HOCLRT	UOCLR	Time
LR	x					189,64 sec
HLRT	0,9998	x				189,09 sec
OCLR	-	-	x			-
HOCLRT	-	-	-	x		-
UOCLR	-	-	-	-	x	-
						Total: 378,73 sec

ILLUSTRATIE 10.12: JACCARD INDEXEN TUSSEN DE WRAPPERS VOOR DE WEBSITE IMDB ONDER DE "GEMEENSCHAPPELIJKE STRINGS EN TOKENSTRINGS VERBETERING" CONDITIE. EEN JACCARD INDEX GEEFT AAN HOEVEEL GELIJKENIS ER IS TUSSEN DE OUTPUT VAN DE EXEC PROCEDURE VAN TWEE WRAPPERS OVER ALLE WEBPAGINA'S IN DE UITVOER-SET. DE LAATSTE KOLOM VAN DEZE TABEL GEEFT VOOR ELKE WRAPPER CLASS DE UITVOERINGSTIJD WEER DIE NODIG WAS OM DE EXEC PROCEDURE UIT TE VOEREN OP ALLE WEBPAGINA'S IN DE UITVOER-SET VAN IMDB.

10.3 Conclusies

De experimenten laten ons toe enkele belangrijke conclusies te trekken. Dit mede dankzij de parameters die we hebben gekozen in Sectie 10.1.3.

Onze eerste en belangrijkste conclusie heeft betrekking op de gemeenschappelijke strings verbetering. Uit de resultaten blijkt namelijk dat een test case die deze verbetering bevat, een beter resultaat geeft dan dezelfde test case zonder de deze verbetering. Dit komt doordat deze verbetering het aantal kandidaten en daarmee ook het aantal kandidaat-combinaties verlaagt. De kandidaten zijn eveneens van een betere kwaliteit, aangezien zij nu gemeenschappelijk zijn voor alle voorbeeldpagina's. Wrappers voor de wrapper classes HLRT, OCLR en HOCLRT worden in onze resultaten ook alleen maar onder condities gevonden die deze verbetering bevatten. Deze verbetering is bijgevolg onmisbaar.

Uit de uitvoeringstijden van de learn test cases blijkt dat de LR wrapper class, al dan niet succesvol, het snelst tot resultaten leidt. Alle test cases die betrekking hebben op de LR wrapper class zijn namelijk geëindigd. Dit is voor geen enkele andere wrapper class het geval. Daarnaast blijkt uit de resultaten van de exec procedures dat de wrappers van de LR wrapper class in de meeste gevallen hetzelfde presteren als wrappers van andere wrapper classes. Een LR wrapper voor een website onder een conditie maakt namelijk eenzelfde fout als de HLRT wrapper voor dezelfde website onder dezelfde conditie, indien aanwezig. De conclusie die we hieruit kunnen trekken luidt daardoor als volgt: zoek eerst naar een geldige LR wrapper en indien geen geldige LR wrapper wordt gevonden, ga dan pas op zoek naar een wrapper van een andere wrapper class.

De tokenstrings verbetering zorgt voor een reductie in het aantal kandidaten en het aantal kandidaat-combinaties. Uit de resultaten van de learn test cases blijkt dat de tokenstrings verbetering in sommige gevallen ook sneller tot een geldige wrapper leidt. We hebben echter gezien dat de tokenstrings verbetering in sommige test cases geen geldige wrapper heeft gevonden wanneer dezelfde test case zonder de tokenstrings verbetering wel al een geldige wrapper heeft gevonden. De tokenstrings verbetering heeft met andere woorden potentie, maar meer experimenten zijn nodig vooraleer we kunnen concluderen dat deze verbetering van toegevoegde waarde is.

Tijdens de experimenten is er geen enkele learn test case, die betrekking heeft op de UOCLR wrapper class, geëindigd. Dit is een spijtig resultaat aangezien deze wrapper class om kan gaan met ontbrekende scheidingstekens en een inconsistente volgorde van scheidingstekens. De oorzaken hiervan zijn de grote hoeveelheid kandidaat-combinaties voor deze wrapper class en de strenge constraints waaraan voldaan moet worden. Dit

leidt tot de conclusie dat de UOCLR wrapper class in zijn huidige vorm niet praktisch is voor grote pagina's.

In Sectie 10.1.5 hebben we een restrictie gelegd op de lengte van de strings/tokenstrings die zich bevinden in de set waaruit de generalized suffix tree wordt opgebouwd. De maximum lengte bedroeg 4.000 tekens/tokens. Deze restrictie gaf geen problemen voor het leren van een wrapper. Dit is een interessant fenomeen omdat we de maximum lengte misschien nog kunnen verkleinen, waardoor het aantal kandidaten en aantal kandidaat-combinaties afneemt, wat op zijn beurt kan leiden tot het sneller leren van een geldige wrapper.

De laatste conclusie die we kunnen trekken is dat het leren van een wrapper de meeste tijd in beslag neemt. Zodra een wrapper gevonden is, kan de exec procedure met behulp van deze wrapper de relaties uit $\pm$10.000 webpagina's en dit slechts enkele minuten tijd.

11. CONCLUSIE MASTERPROEF

Wij zijn deze masterproef begonnen (Hoofdstuk 1) met een inleiding over relationele data op het web, gevolgd door een informele beschrijving van het informatie extractieproces. Dit proces bestaat uit de volgende 4 stappen: crawling, de eigenlijke extractie van informatie, het toevoegen van semantiek, en werken met de verkregen data.

Daarna hebben we in Hoofdstuk 2 het relationeel model besproken. We hebben het in deze masterproef voortdurend gehad over relationele data, waardoor een basiskennis van het relationeel model noodzakelijk is.

Hoofdstuk 3 opende met een bespreking van wrapper procedures. Een wrapper procedure is een procedure die ontwikkeld is om informatie uit een specifieke soort webpagina's te halen. Vervolgens hebben we het formele informatie extractieproces besproken, alsook de rol die het wrapper inductieprobleem hierin speelt. Het wrapper inductieprobleem is het automatisch leren van een wrapper uit een verzameling van webpagina's en hun bijhorende relaties, zodat de wrapper later gebruikt kan worden om relaties uit andere gelijkaardige webpagina's te halen.

In de volgende vijf hoofdstukken hebben we de verschillende wrapper classes besproken. Deze hebben elk hun eigen learn algoritme voor het leren van een wrapper en een eigen exec procedure om aan de hand van de geleerde wrapper relaties uit webpagina's te halen.

In Hoofdstuk 4 hebben we de LR wrapper class besproken. Deze wrapper class maakt gebruik van linker en rechter scheidingstekens voor het vinden van de attribuutwaarden van een relatie op een webpagina. De LR wrapper is een vector van $2K$ scheidingstekens, namelijk $\langle\langle l_1,r_1\rangle,\ldots,\langle l_K,r_K\rangle\rangle$.

In Hoofdstuk 5 hebben we de HLRT wrapper class besproken. Deze wrapper class maakt naast de linker en rechter scheidingstekens van de LR wrapper class ook gebruik van een head en tail scheidingsteken voor het vinden van de attribuutwaarden van een relatie op een webpagina. De HLRT wrapper is een vector van $2K+2$ scheidingstekens, namelijk $\langle h, t, [\langle l_1, r_1\rangle, \ldots, \langle l_K, r_K\rangle]\rangle$.

In Hoofdstuk 6 hebben we de OCLR wrapper class besproken. Deze wrapper class maakt naast de linker en rechter scheidingstekens van de LR wrapper class ook gebruik van een open en close scheidingsteken voor het vinden van de attribuutwaarden van een relatie op een webpagina. De OCLR wrapper is een vector van $2K+2$ scheidingstekens, namelijk $\langle o, c, [\langle l_1, r_1\rangle, \ldots, \langle l_K, r_K\rangle]\rangle$.

In Hoofdstuk 7 hebben we de HOCLRT wrapper class besproken. Deze wrapper class is een combinatie van de HLRT en OCLR wrapper class en maakt naast linker en rechter scheidingstekens ook gebruik van een head, tail, open en close scheidingsteken voor het vinden van de attribuutwaarden van een relatie op een webpagina. De HOCLRT wrapper is een vector van $2K+4$ scheidingstekens, namelijk $\langle h, t, o, c, [\langle l_1, r_1\rangle, \ldots, \langle l_K, r_K\rangle]\rangle$.

In Hoofdstuk 8 hebben we de UOCLR wrapper class besproken. Deze wrapper class is gebaseerd op de OCLR wrapper class, maar kan tijdens de exec procedure omgaan met ontbrekende attributen en een inconsistente volgorde van attributen. Dit doordat tijdens het learn algoritme aan strengere constraints voldaan moet worden. De UOCLR wrapper is, net zoals de OCLR wrapper, een vector van $2K+2$ scheidingstekens, namelijk $\langle o, c, [\langle l_1, r_1\rangle, \ldots, \langle l_K, r_K\rangle]\rangle$.

Hoofdstuk 9 beschrijft twee verbeteringen die we hebben doorgevoerd aan de learn algoritmes van de wrapper classes. De eerste verbetering, genaamd gemeenschappelijke strings, gaat enkel kandidaten uit gemeenschappelijke strings genereren, zodat een kandidaat voor een scheidingsteken altijd op elke leer-pagina voorkomt. Dit zorgt voor een verlaging van het aantal kandidaten. De tweede verbetering, genaamd tokenstrings, maakt gebruik van de syntax van HTML om eveneens het aantal kandidaten te verlagen.

We hebben de verschillende wrapper classes en verbeteringen ook allemaal in de praktijk getest. Deze experimenten staan beschreven in Hoofdstuk 10 en tonen aan dat het leren van een wrapper, zonder gebruik te maken van een verbetering, veel tijd in beslag kan nemen. De gemeenschappelijke strings verbetering verlaagt deze leertijd aanzienlijk en is bijgevolg een noodzaak. De tokenstrings verbetering zorgt niet altijd voor een verlaging van de leertijd. Uit de experimenten is niet duidelijk gebleken of de tokenstrings verbetering een toegevoegde waarde is. Verder hebben de experimenten aangetoond dat de LR wrapper, indien deze bestaat, snel gevonden kan worden en net zo goed is als een HLRT, OCLR of HOCLRT wrapper om de relatie uit een webpagina te halen. De UOCLR wrapper class bleek te strenge constraints te hebben om binnen een afzienbare tijd een geldige wrapper te vinden.

De experimenten hebben ook aangetoond waar er in de toekomst nog gewerkt kan worden aan de wrapper classes. Learn algoritmes vereisen namelijk teveel tijd vanwege de grote hoeveelheden kandidaat-combinaties. Als we het aantal kandidaat-combinaties kunnen verlagen, helpt dit om het praktisch nut van de wrapper classes te verhogen. Onderzoek naar de constraints zou ook kunnen helpen de algemene bruikbaarheid te verhogen. Minder strenge constraints betekent namelijk sneller een geldige wrapper. Zeker voor de UOCLR wrapper class zijn de constraints te streng om praktisch nut te hebben. Bovenstaande problemen geven ook aanleiding om andere technieken dan

wrapper inductie te bestuderen, waarbij de complexiteit van de algoritmes aanzienlijk lager ligt.

In het volgende en laatste hoofdstuk van deze masterproef wordt gerelateerd werk besproken waarin andere technieken dan wrapper inductie onderzocht werden.

12. RELATED WORK

In dit laatste hoofdstuk bespreken we een aantal onderzoeken naar andere technieken die betrekking hebben op één of meerdere stappen van het informatie extractieproces.

In de paper "Uncovering the Relational Web" van Cafarella et al. [7] wordt onderzoek gedaan naar de eerste drie stappen van het informatie extractieproces. Ze gaan op zoek naar webpagina's met relationele informatie door te kijken welke webpagina's de HTML table tag bevatten. De HTML table tag wordt echter ook voor andere doeleinden gebruikt, zoals de webpagina van lay-out voorzien, het weergeven van een kalender, etc. Omwille van deze reden hebben ze een filter stap toegevoegd, die alle webpagina's die geen relationele data bevatten wegfiltert. Hun filter maakt zowel gebruik van parsers als van classifiers voor het detecteren van de relationele informatie. Naast het extraheren van relationele informatie besteden ze ook aandacht aan het vinden van semantiek. Meer bepaald gaan ze op zoek naar labels voor de attributen van de relatie. Dit doen ze door te controleren of er een header rij aanwezig is in de relatie, of door zelf attribuutnamen te genereren op basis van relaties die reeds attribuutnamen hebben.

De paper "WebTables: Exploring the Power of Tables on the Web" van Cafarella et al. [8] gaat verder op het onderzoek van Cafarella et al. [7]. Dit onderzoek hebben we reeds hierboven aangehaald en richtte zich op de eerste drie stappen van het informatie extractieproces. In het nieuwe onderzoek focussen ze zich daarentegen op de laatste stap van het informatie extractieproces, namelijk "werken met de data".
Enerzijds hebben ze een nieuw zoeksysteem ontwikkeld dat geschikt is voor relationele data. Dit zoeksysteem maakt gebruik van relational ranking en relational indexing om zo de meest relevante resultaten eerst terug te geven, en dit op basis van een keyword zoekopdracht. Relational ranking gaat relaties rangschikken op grond van hun belangrijkste eigenschappen. Enkele van deze eigenschappen zijn: aantal rijen, aantal kolommen, aanwezigheid van een header rij, aantal lege waardes in de relatie, etc. Sommige van deze eigenschappen maken gebruik van de rij en kolom posities in de relatie. De inverted index die traditionele zoekmachines gebruiken, ondersteunt deze eigenschappen niet. De inverted index houdt namelijk enkel een offset bij vanaf de top van een pagina. Omwille van deze reden hebben Cafarella et al. de relational index ontwikkeld. Deze index maakt gebruik van een rij en kolom offset voor elke attribuutwaarde van een relatie, waardoor de relational ranking eigenschappen wel ondersteund worden.
Anderzijds hebben ze ook drie applicaties ontwikkeld die de grote hoeveelheden relaties, die Cafarella et al. [7] verkregen hebben, uitbuiten. De eerste applicatie, genaamd

schema auto-complete, helpt database designers bij het maken van een database schema. Wanneer een gebruiker een attribuut opgeeft, bijvoorbeeld “album”, dan geeft de applicatie suggesties van gerelateerde attributen, bijvoorbeeld “artiest”, “lengte”, “jaar”, edm. die de gebruiker eventueel ook kan opnemen in het schema. De tweede applicatie, genaamd attribute synonym discovery, zoekt attributen die synoniemen zijn van elkaar. De synoniemen die hieruit voortkomen zijn vollediger dan een thesaurus, aangezien ook niet natuurlijke taalstrings zoals “tel-#” aanwezig zijn tussen de attributen. De derde en laatste applicatie, genaamd join-graph traversal, is een schema explorer. De applicatie laat met andere woorden een gebruiker toe te navigeren door alle schema’s die Cafarella et al. [7] verkregen hebben. In deze graaf zijn twee schema’s met elkaar geconnecteerd wanneer ze een gemeenschappelijk attribuut hebben.

Venetis et al. [6] hebben onderzoek gedaan naar stap 3 (toevoegen van semantiek) en stap 4 (werken met de data) van het informatie extractieproces. Dit onderzoek is beschreven in de paper “Recovering Semantics of Tables on the Web” [6]. Semantiek wordt toegevoegd in twee vormen, namelijk kolomnamen en relaties tussen paren van kolommen. Hiervoor wordt gebruik gemaakt van twee databases. De eerste database, genaamd isA, bestaat uit paren van de vorm (instance, class). De tweede database, genaamd relations, bestaat uit triplets van de vorm (argument1, predicate, argument2). Wanneer de isA database voldoende paren bevat waarbij instance een waarde is van een kolom A en waarbij class gelijk is aan C, dan wordt kolom A met class C gelabeld. Wanneer de relations database voldoende triplets bevat waarbij argument1 een waarde is uit kolom A en argument2 een waarde is uit kolom B, telkens met predicate gelijk aan R, dan wordt de relatie tussen de kolommen A en B gelabeld met R. Zowel de isA database als de relations database wordt geconstrueerd door feiten te extraheren uit tekst op webpagina’s. Een voorbeeld van een feit waaruit tupels voor de isA database worden gehaald is: “cities such as Brussels and Amsterdam”. Dit feit resulteert in twee tupels voor de isA database, namelijk (Brussels, cities) en (Amsterdam, cities). Een voorbeeld van een feit waaruit een tupel voor de relations database wordt gehaald is: “Brussels is the capital of Belgium”. Dit feit resulteert in het triplet (Brussels, capital of, Belgium) voor de relations database. Stap 4 van het informatie extractieproces komt in het onderzoek van Venetis et al. aan bod onder de vorm van een nieuw zoeksysteem voor tabellen. Dit zoeksysteem maakt uitvoerig gebruik van de verkregen semantiek voor het vinden van tabellen die gerelateerd zijn aan de zoek query.

Informatie extractietechnieken die gebruik maken van wrappers veronderstellen meestal dat de pagina’s een uniforme opmaak hebben. Deze veronderstelling heeft betrekking op zowel de leer-pagina’s, als de uitvoer-pagina’s. Dit is echter niet in overeenstemming met de werkelijkheid, waar pagina’s een non-uniforme opmaak kunnen hebben. Omwille

van deze reden hebben wij de UOCLR wrapper class ontwikkeld. Deze wrapper class kan tot op zekere hoogte overweg met uitvoer-pagina's die een minder uniforme opmaak hebben. Elmeleegy et al. [15] vinden dat deze veronderstelling toepasselijk is voor tabellen, maar niet voor lijsten. Lijsten worden namelijk handmatig gegenereerd en bevatten vaak menselijke fouten waardoor ze zeer ongestructureerd kunnen zijn. De scheidingstekens waarop wrappers zich baseren, kunnen bijgevolg zeer inconsistent zijn of zelf helemaal niet aanwezig zijn. Dit wordt allemaal nog eens bemoeilijkt door het eventueel ontbreken van informatie in bepaalde tupels. Elmeleegy et al. hebben daarom een techniek ontwikkeld specifiek voor HTML lijsten die met deze moeilijkheden kan omgaan. Deze techniek is beschreven in de paper "Harvesting Relational Tables from Lists on the Web" [15] en werkt volledig automatisch, zonder tussenkomst van een gebruiker. De techniek maakt gebruik van lokale beslissingen binnen een lijn van een lijst en globale tabel beslissingen over alle lijnen van die lijst om zo de best mogelijke tabel uit die lijst te halen. Binnen een lijn wordt gebruik gemaakt van data types, scheidingstekens, een taalmodel en een verzameling van tabellen om te achterhalen wat de attribuutwaarden van een tupel zijn. Het taalmodel wordt gebruikt voor het identificeren van tekstdelen die bij elkaar horen. De verzameling van tabellen wordt gebruikt om te kijken of een tekstdeel als een geheel voorkomt in een cel van één of meerdere andere tabellen. De globale tabel beslissingen houden in dat we controleren of alle waarden in een kolom van de tabel logisch samen horen. Hiervoor wordt opnieuw gebruik gemaakt van de verzameling van tabellen.

In de paper "Towards an Ecosystem of Structured Data on the Web" van Alon Halevy [16] wordt er gesproken over een ecosysteem voor data op het web. Dit ecosysteem houdt in dat gebruikers data moeten kunnen vinden, hergebruiken, visualiseren en publiceren door middel van eenvoudige tools. Het lijkt alsof dit ecosysteem enkel op de laatste stap ("werken met data") van het informatie extractieproces focust, maar dat is niet het geval. Wanneer gebruikers data uploaden en verwerken bij een service die een dergelijk ecosysteem hanteert, dan wordt aan deze service steeds nieuwe data toegevoegd in het gewenste formaat. De service moet met andere woorden geen webpagina's crawlen om deze data te vinden of technieken gebruiken om deze data uit webpagina's te halen. De data is onmiddellijk gereed om opnieuw gebruikt te worden. Enkel stap 3 ("toevoegen van semantiek") van het informatie extractieproces moet nog uitgevoerd worden, indien gebruikers zelf niet voldoende semantiek aan hun data toevoegen. Een voorbeeld van een service die dergelijk ecosysteem hanteert is Google Fusion Tables. Deze service is beschreven in "Google Fusion Tables: Web-Centered Data Management and Collaboration" van Gonzalez et al. [17] en "Google Fusion Tables: Data management, Integration and Collaboration in the Cloud" van Gonzalez et al. [18]. Google Fusion Tables laat gebruikers toe om data in verschillende formaten te uploaden

(spreadsheets, CSV, KML, etc). Deze data kan de gebruiker vervolgens uitbreiden met data die reeds door andere personen is geüpload. Tot slot kan hij zijn data visualiseren (kaarten, grafieken, tijdlijnen, etc.) en publiceren, zodat andere gebruikers hierop verder kunnen bouwen. Een voorbeeld hiervan is een gebruiker die over een dataset beschikt over de koffieproductie per land. Hij upload deze dataset naar Google Fusion Tables en zoekt vervolgens naar een dataset over de koffieconsumptie per land en voegt deze twee datasets samen. Tot slot maakt hij een kaart visualisatie die hij op zijn eigen website plaatst en publiceert hij zijn data op Fusion Tables zodat andere gebruikers zijn data kunnen gebruiken om bijvoorbeeld de koffieproductie en –consumptie van elk land te vergelijken met de suikerproductie en –consumptie van elk land. Merk op dat gebruikers kunnen kiezen met wie ze hun data kunnen delen (iedereen, specifieke personen, of niemand), zodat Fusion Tables ook voor meer gevoelige data gebruikt kan worden.

BIBLIOGRAFIE

[1] Raggett, D., Le Hors, A., & Jacobs, I. (1999, December 24). *HTML 4.01 Specification W3C Recommendation*. Opgeroepen op november 29, 2013, van W3C: http://www.w3.org/TR/1999/REC-html401-19991224/

[2] Etemad, E. (2011, mei 12). *Cascading Style Sheets (CSS) Snapshot 2010*. Opgeroepen op november 29, 2013, van W3C: http://www.w3.org/TR/CSS/

[3] *Standard ECMA-262 - ECMAScript Language Specification - Edition 5.1*. (2011, juni). Opgeroepen op december 9, 2013, van Ecma International: http://www.ecma-international.org/publications/standards/Ecma-262.htm

[4] W3C. (2014, januari 29). *JavaScript Web APIs*. Opgeroepen op februari 6, 2014, van W3C: http://www.w3.org/standards/webdesign/script

[5] Leskovec, J., Rajaraman, A., & Ullman, J. (2011). *Mining of Massive Datasets.* Cambridge University Press.

[6] Venetis, P., Halevy, A., Madhavan, J., Pasca, M., Shen, W., Wu, F., et al. (2011). VLDB Endowment, Vol. 4, No. 9. *Recovering Semantics of Tables on the Web*, (pp. 528-538). Seattle, Washington.

[7] Cafarella, M., Halevy, A., Wang, D., Wu, E., & Zhang, Y. (2008). 11th International Workshop on Web and Databases (WebDB). *Uncovering the Relational Web.* Vancouver, Canada.

[8] Cafarella, M. J., Halevy, A., Wang, D. Z., Wu, E., & Zhang, Y. (2008). PVLDB. *WebTables: Exploring the Power of Tables on the Web* (pp. 538-549). Auckland, New-Zealand: ACM.

[9] Page, L., Brin, S., Motwani, R., & Winograd, T. (1998). *The PageRank Citation Ranking: Bringing Order to the Web.*

[10] Garcia-Molina, H., Ullman, J., & Widom, J. (2009). *Database Systems: The Complete Book (Second Edition).* Upper Saddle River: Prentice Hall.

[11] Kushmerick, N. (2000). Wrapper Induction: Efficiency and Expressiveness. *Artificial Intelligence* (118), 15-68.

[12] Kushmerick, N., Weld, D. S., & Doorenbos, R. (1997). Wrapper Induction for Information Extraction. *Proceedings of the 15th International Joint Conference on Artificial Intelligence* (pp. 729-737). Nagoya, Japan: University of Washington.

[13] Kushmerick, N. (1997). *Wrapper Induction for Information Extraction.* Washington: University of Washington.

[14] Mehta, D. P., & Sahni, S. (2004). *Handbook Of Data Structures And Applications (Chapman & Hall/CRC Computer and Information Science Series.).* Chapman & Hall/CRC.

[15] Elmeleegy, H., Madhavan, J., & Halevy, A. (2009). VLDB. *Harvesting Relational Tables from Lists on the Web* (pp. 1078-1089). Lyon, France: ACM.

[16] Halevy, A. (2012). EDBT. *Towards an Ecosystem of Structured Data on the Web* (pp. 1-2). Berlin, Germany: ACM.

[17] Gonzalez, H., Halevy, A., Jensen, C., Langen, A., Madhavan, J., Shapley, R., et al. (2010). SIGMOD. *Google Fusion Tables: Web-Centered Data Management and Collaboration* (pp. 1061-1066). Indianapolis, Indiana, USA: ACM.

[18] Gonzalez, H., Halevy, A., Jensen, C., Langen, A., Madhavan, J., Shapley, R., et al. (2010). SoCC. *Google Fusion Tables: Data Management, Integration and Collaboration in the Cloud* (pp. 175-180). Indianapolis, Indiana, USA: ACM.

Bijlage A: HULPFUNCTIES

In deze bijlage worden de hulpfuncties van de exec procedures en de learn algoritmes die doorheen deze masterproef aan bod komen besproken.

A.1 HULPFUNCTIES EXEC PROCEDURES

In deze sectie worden de hulpfuncties van de exec procedures besproken. Voor elke functie geven we een korte beschrijving van de werking. Merk op dat wanneer in een functie hieronder naar een scheidingsteken op een pagina wordt gezocht, er altijd vertrokken wordt vanaf de huidige positie van de *file pointer*.

A.1.1 SEARCH

De functie search(delimiter *d*, page *P*) gaat op zoek naar het eerste voorkomen van scheidingsteken *d* in pagina *P* en geeft de startpositie van dit scheidingsteken terug. We definiëren deze functie als volgt:

> search(delimiter *d*, page *P*) =
>
> find the start position of the first occurrence of *d* in *P* and return this start position

Als scheidingsteken *d* niet voorkomt in pagina *P*, dan geeft deze functie het EOF symbool terug.

A.1.2 SEARCH AND MOVE

De functie search_move(delimiter *d*, page *P*) gaat op zoek naar het eerste voorkomen van scheidingsteken *d* in pagina *P*, verplaatst de file pointer naar de startpositie van dit scheidingsteken en geeft deze startpositie terug. We definiëren deze functie als volgt:

> search_move(delimiter *d*, page *P*) =
>
> find the start position of the first occurrence of *d* in *P*, move the file pointer to this start position and return this start position

Als scheidingsteken *d* niet voorkomt in pagina *P*, dan geeft deze functie het EOF symbool terug.

A.1.3 SEARCH AFTER

De functie search_after(delimiter *d*, page *P*, position *x*) gaat op zoek naar het eerste voorkomen van scheidingsteken *d* in pagina *P* na positie *x* en geeft de startpositie van het scheidingsteken *d* terug. We definiëren deze functie als volgt:

search_after(delimiter *d*, page *P*, position *x*) =
find the start position of the first occurrence of *d* in *P* after position *x* and return this start position

Als scheidingsteken *d* niet voorkomt in pagina *P* na positie *x*, dan geeft deze functie het EOF symbool terug.

A.1.4 Search after and move

De functie search_after_move(delimiter *d*, page *P*, position *x*) gaat op zoek naar het eerste voorkomen van scheidingsteken *d* in pagina *P* na positie *x*, verplaatst de file pointer naar de startpositie van dit scheidingsteken en geeft deze startpositie terug. We definiëren deze functie als volgt:

search_after_move(delimiter *d*, page *P*, position *x*) =
find the start position of the first occurrence of *d* in *P* after position *x*, move the file pointer to this start position and return this start position

Als scheidingsteken *d* niet voorkomt in pagina *P* na positie *x*, dan geeft deze functie het EOF symbool terug.

A.1.5 Search before

De functie search_before(delimiter *d*, page *P*, position *x*) gaat op zoek naar het eerste voorkomen van scheidingsteken *d* in pagina *P* voor positie *x* en geeft de startpositie van het scheidingsteken *d* terug. We definiëren deze functie als volgt:

search_before(delimiter *d*, page *P*, position *x*) =
find the start position of the first occurrence of *d* in *P* before position *x* and return this start position

Als scheidingsteken *d* niet voorkomt in pagina *P* voor positie *x*, dan geeft deze functie het EOF symbool terug.

A.1.6 Search after and before

De functie search_after_before(delimiter *d*, page *P*, position *x*, position *y*) gaat op zoek naar het eerste voorkomen van scheidingsteken *d* in pagina *P* na positie *x* & voor positie *y* en geeft de startpositie van het scheidingsteken *d* terug. We definiëren deze functie als volgt:

search_after_before(delimiter *d*, page *P*, position *x*, position *y*) =
find the start position of the first occurrence of *d* in *P* after position *x* & before positon *y* and return this start position

Als scheidingsteken *d* niet voorkomt in pagina *P* na positie *x* & voor positie *y*, dan geeft deze functie het EOF symbool terug.

A.2 HULPFUNCTIES LEARN ALGORITMES

In deze sectie worden de hulpfuncties van de learn algoritmes besproken. Voor elke hulpfunctie wordt het doel besproken, wordt een mathematische voorstelling van de werking gegeven, wordt het aantal output waarden aan de hand van een formule uitgedrukt, en wordt er tot slot een voorbeeld gegeven. Dit voorbeeld wordt telkens toegepast op het country-code voorbeeld van Figuur 3.1. De vereenvoudigde broncode van deze webpagina wordt voor de eenvoud opnieuw weergegeven in Illustratie A.1.

```
1: <html><head><title>Some Country Codes</title></head><body>
2: <b>Belgium</b> <i>32</i><br>
3: <b>Congo</b> <i>243</i><br>
4: <b>Egypt</b> <i>20</i><br>
5: <b>Spain</b> <i>34</i><br>
6: </body></html>
```

ILLUSTRATIE A.1: VOORBEELD WAAROP DE HULPFUNCTIES VAN DE LEARN ALGORITMES WORDEN GEDEMONSTREERD.

Doorheen deze bijlage komt de notatie $P_n[x,y]$ veelvuldig voor. In deze notatie duidt het symbool P_n een pagina aan. De symbolen x en y, met $x \leq y$, duiden posities in de pagina P_n aan. Het geheel zegt dat we uit pagina P_n een substring nemen die begint op positie x en eindigt op positie y (niet inclusief).

A.2.1 ATTRIBS

De functie attribs(k,$\mathcal{E}$) geeft, voor elke relatie in $\mathcal{E}$, alle attribuutwaarden voor het attribuut k terug. De mathematische voorstelling van deze functie is:

$$\text{attribs}(k,\mathcal{E}) = \{P_n[b_{m,k},e_{m,k}] \mid \langle\ldots,\langle b_{m,k},e_{m,k}\rangle,\ldots\rangle \in R_n,\ \langle P_n,R_n\rangle \in \mathcal{E}\}$$

Het aantal waarden dat deze functie teruggeeft kunnen we uitdrukken met behulp van volgende formule:

$$|\text{attribs}(k,\mathcal{E})| = \sum_{\langle P_n,R_n\rangle \in \mathcal{E}} |R_n|$$

De output die deze functie teruggeeft voor het country-code voorbeeld van Illustratie A.1 is te zien in Illustratie A.2.

```
attribs(1,ℰ) = {'Belgium', 'Congo', 'Egypt', 'Spain'}
attribs(2,ℰ) = {'32', '243', '20', '34'}
```

ILLUSTRATIE A.2: OUTPUT VAN DE FUNCTIE ATTRIBS VOOR HET COUNTRY-CODE VOORBEELD.

A.2.2 HEADS

De functie heads($\mathcal{E}$) geeft het hoofd van elke pagina in $\mathcal{E}$ terug. Het hoofd van een pagina is de tekst die voor het eerste tupel voorkomt. De mathematische voorstelling van deze functie is:

$$\text{heads}(\mathcal{E}) = \{P_n[1,b_{1,1}] \mid \langle P_n,\{\langle\langle b_{1,1},e_{1,1}\rangle,\ldots\rangle,\ldots\}\rangle \in \mathcal{E}\}$$

Het aantal waarden dat deze functie teruggeeft kunnen we uitdrukken met behulp van volgende formule:

$$|\text{heads}(\mathcal{E})| = |\mathcal{E}|$$

De output die deze functie teruggeeft voor het country-code voorbeeld van Illustratie A.1 is te zien in Illustratie A.3.

```
heads(ℰ) =
     {'<html><head><title>Some Country Codes</title></head><body>⇓<b>'}
```

ILLUSTRATIE A.3: OUTPUT VAN DE FUNCTIE HEADS VOOR HET COUNTRY-CODE VOORBEELD.

A.2.3 TAILS

De functie tails($\mathcal{E}$) geeft de voet van elke pagina in $\mathcal{E}$ terug. De voet van een pagina is de tekst die achter het laatste tupel voorkomt. De mathematische voorstelling van deze functie is:

$$\text{tails}(\mathcal{E}) = \{P_n[e_{|R|,K},|P_n|] \mid \langle P_n,\{\ldots,\langle\ldots,\langle b_{|R|,K},e_{|R|,K}\rangle\rangle\}\rangle \in \mathcal{E}\}$$

Het aantal waarden dat deze functie teruggeeft kunnen we uitdrukken met behulp van volgende formule:

$$|\text{tails}(\mathcal{E})| = |\mathcal{E}|$$

De output die deze functie teruggeeft voor het country-code voorbeeld van Illustratie A.1 is te zien in Illustratie A.4.

```
tails(ℰ) = {'</i><br>⇓</body></html>'}
```

ILLUSTRATIE A.4: OUTPUT VAN DE FUNCTIE TAILS VOOR HET COUNTRY-CODE VOORBEELD.

A.2.4 SEPS

De functie seps(k,$\mathcal{E}$) geeft voor elke pagina de tekst tussen het k^{de} en het $((k \bmod K)+1)^{de}$ attribuut terug. De modulo functie is nodig voor het geval dat $k = K$. De seps functie moet dan teruggeven wat er tussen het laatste attribuut van het huidige tupel en het eerste attribuut van het volgende tupel staat. De mathematische voorstelling van deze functie is:

$$\text{seps}(k,\mathcal{E}) = \begin{cases} \bigcup_{\langle P_n,R_n\rangle\in\mathcal{E}} \{P_n[e_{m,K},b_{m+1,1}] \mid \langle\ldots,\langle b_{m,K},e_{m,K}\rangle\rangle \in R_n \wedge m < |R_n|\} & \text{if } k = K \\ \bigcup_{\langle P_n,R_n\rangle\in\mathcal{E}} \{P_n[e_{m,k},b_{m,k+1}] \mid \langle\ldots,\langle b_{m,k},e_{m,k}\rangle,\ldots\rangle \in R_n\} & \text{otherwise} \end{cases}$$

Het aantal waarden dat deze functie teruggeeft hangt af van de waarde van k. Wanneer $k = K$ krijgen we namelijk één separator minder per pagina, dan wanneer $k \neq K$. Dit komt omdat de voet van een pagina geen twee attributen scheidt. De voet wordt bijgevolg niet opgenomen in de lijst van separatoren. Als $k \neq K$ kunnen we het aantal waarden dat deze functie teruggeeft uitdrukken aan de hand van volgende formule:

$$|\text{seps}(k,\mathcal{E})| = \sum_{\langle P_n,R_n\rangle \in \mathcal{E}} |R_n|$$

Als $k = K$ kunnen we het aantal teruggegeven waarden uitdrukken met behulp van de volgende formule:

$$|\text{seps}(K,\mathcal{E})| = \sum_{\langle P_n,R_n\rangle \in \mathcal{E}} (|R_n|-1)$$

De output die deze functie teruggeeft voor het country-code voorbeeld van Illustratie A.1 is te zien in Illustratie A.5. We zien dat seps(2,$\mathcal{E}$) niet de voet van de pagina bevat, omdat $K = 2$ in dit voorbeeld.

```
seps(1,ℰ) = {'</b> <i>','</b> <i>','</b> <i>','</b> <i>'}
seps(2,ℰ) = {'</i><br>⇓<b>','</i><br>⇓<b>','</i><br>⇓<b>'}
```

ILLUSTRATIE A.5: OUTPUT VAN DE FUNCTIE SEPS VOOR HET COUNTRY-CODE VOORBEELD.

A.2.5 NEIGHBORS

De functie neighbors$_l$(k,$\mathcal{E}$) geeft alle strings terug die links van de k^{de} attribuutwaarde voorkomen. Hiervoor maakt de neighbors$_l$ functie gebruik van de seps functie en wanneer $k = 1$ zal er ook gebruik gemaakt worden van de heads functie. De mathematische voorstelling van de neighbors$_l$ functie is:

$$\text{neighbors}_l(k,\mathcal{E}) = \begin{cases} \text{seps}(K,\mathcal{E}) \cup \text{heads}(\mathcal{E}) & \text{if } k = 1 \\ \text{seps}(k\text{-}1,\mathcal{E}) & \text{otherwise} \end{cases}$$

Het aantal waarden dat deze functie teruggeeft kunnen we uitdrukken met behulp van volgende formule:

$$|\text{neighbors}_l(k,\mathcal{E})| = \sum_{\langle P_n,R_n\rangle \in \mathcal{E}} |R_n|$$

De functie $\text{neighbors}_r(k,\mathcal{E})$ geeft alle strings terug die rechts van de k^{de} attribuutwaarde voorkomen. Hiervoor maakt de neighbors_r functie gebruik van de seps functie en wanneer $k = K$ zal er ook gebruik gemaakt worden van de tails functie. De mathematische voorstelling van de neighbors_r functie is:

$$\text{neighbors}_r(k,\mathcal{E}) = \begin{cases} \text{seps}(K,\mathcal{E}) \cup \text{tails}(\mathcal{E}) & \text{if } k = K \\ \text{seps}(k,\mathcal{E}) & \text{otherwise} \end{cases}$$

Het aantal waarden dat deze functie teruggeeft kunnen we uitdrukken met behulp van volgende formule:

$$|\text{neighbors}_r(k,\mathcal{E})| = \sum_{\langle P_n,R_n\rangle \in \mathcal{E}} |R_n|$$

De output die de functies neighbors_l en neighbors_r teruggeven voor het country-code voorbeeld van Illustratie A.1 is te zien in Illustratie A.6.

$\text{neighbors}_l(1,\mathcal{E}) =$
$\{$ '
⇓', '
⇓', '
⇓', $\}$
$\{$'<html><head><title>Some Country Codes</title></head><body>⇓'$\}$
$\text{neighbors}_l(2,\mathcal{E}) =$ {' ',' ',' ',' '}
$\text{neighbors}_r(1,\mathcal{E}) =$ {' ',' ',' ',' '}
$\text{neighbors}_r(2,\mathcal{E}) =$ $\{$'
⇓', '
⇓', '
⇓',$\}$
'
⇓</body></html>'

ILLUSTRATIE A.6: OUTPUT VAN DE FUNCTIES NEIGHBORS_L EN NEIGHBORS_R VOOR HET COUNTRY-CODE VOORBEELD.

A.2.6 SCAN

De functie scan(string s_1, string s_2) geeft een suffix van s_1 terug die begint bij het eerste voorkomen van s_2 in s_1. We definiëren deze functie als volgt:

$$\text{scan}(s_1, s_2) = \text{the suffix of } s_1 \text{ starting at the first occurrence of } s_2$$

Als s_2 niet in s_1 voorkomt, dan geeft deze functie de lege string terug. In Illustratie A.7 zien we een voorbeeld van de scan functie.

s_1 = 'Some Country Codes<p>⇓'
s_2 = '<p>'

scan(s_1, s_2) = {'<p>⇓'}

ILLUSTRATIE A.7: VOORBEELD VAN DE SCAN FUNCTIE.

A.2.7 SCAN BEFORE

De functie scanBefore(string s_1, string s_2) geeft een prefix van s_1 terug die eindigt bij het eerste voorkomen van s_2 in s_1. We definiëren deze functie als volgt:

$$\text{scanBefore}(s_1, s_2) = \text{the prefix of } s_1 \text{ ending at the first occurrence of } s_2$$

Als s_2 niet in s_1 voorkomt, dan geeft deze functie de lege string terug. In Illustratie A.8 zien we een voorbeeld van de scanBefore functie.

```
s1 = '<b>Some Country Codes</b><p>⇓<b>'
s2 = '<p>'

scanBefore(s1, s2) = {'<b>Some Country Codes</b>'}
```

ILLUSTRATIE A.8: VOORBEELD VAN DE SCANBEFORE FUNCTIE.

A.2.8 SCAN AFTER

De functie scanAfter(string s_1, string s_2) geeft een suffix van s_1 terug die begint bij het einde van het eerste voorkomen van s_2 in s_1. We definiëren deze functie als volgt:

$$\text{scanAfter}(s_1, s_2) = \text{the suffix of } s_1 \text{ starting at the end of the first occurrence of } s_2$$

Als s_2 niet in s_1 voorkomt, dan geeft deze functie de lege string terug. In Illustratie A.9 zien we een voorbeeld van de scanAfter functie.

```
s1 = '<b>Some Country Codes</b><p>⇓<b>'
s2 = '<p>'

scanAfter(s1, s2) = {'⇓<b>'}
```

ILLUSTRATIE A.9: VOORBEELD VAN DE SCANAFTER FUNCTIE.

Bijlage B: Correctie HOCLRT Wrapper Class Constraints

In de paper "Wrapper Induction for Information Extraction" van Kushmerick et al. [12] en de doctoraatsthesis "Wrapper Induction for Information Extraction" van Kushmerick [13] zijn twee constraints van de HOCLRT wrapper class foutief. In deze bijlage bespreken we waarom deze constraints fout zijn, wat de correcte constraints zijn en een voorbeeld dat het verschil tussen beide constraints illustreert.

B.1 De foutieve constraints en hun correctie

In Illustratie B.1 worden de twee foutieve constraints van de HOCLRT wrapper class gegeven, namelijk C^D en C^K. We kunnen aantonen dat deze constraints foutief zijn door de werking van de $exec_{HOCLRT}$ procedure te bestuderen. Deze procedure wordt voor de eenvoud opnieuw weergegeven in Illustratie B.2. De hulpfuncties die in de $exec_{HOCLRT}$ procedure van Illustratie B.2 vermeld worden, maar niet beschreven zijn, zijn terug te vinden in Bijlage A.

Constraint D: u_t mag niet voorkomen voor u_{l_1}, in de tekst die voorkomt na u_o, in de tekst die voorkomt na u_h, en dit in het hoofd van eender welke pagina

Constraint K: u_t mag niet voorkomen voor u_{l_1}, in de tekst die voorkomt na u_o, in de tekst die voorkomt na u_c, in de tekst tussen eender welke twee opeenvolgende tupels, en dit in eender welke pagina

Illustratie B.1: Foutieve constraints van de HOCLRT wrapper class.

```
1:  procedure exec_HOCLRT(wrapper ⟨h,t,o,c,[⟨l_1,r_1⟩,…,⟨l_K,r_K⟩]⟩, page P)
2:    search_move(h,P)
3:    m ≔ 0
4:    while search(o,P) ≤ search(t,P)
5:      m ≔ m + 1
6:      search_move(o,P)
7:      for each ⟨l_k,r_k⟩ ∈ [⟨l_1,r_1⟩,…,⟨l_K,r_K⟩]
8:        b_m,k ≔ search(l_k,P) + |l_k|
9:        e_m,k ≔ search_after_move(r_k,P,b_m,k)
10:     search_move(c,P)
11:   return relation {⟨⟨b_1,1,e_1,1⟩,…,⟨b_1,K,e_1,K⟩⟩, … , ⟨⟨b_|R|,1,e_|R|,1⟩,…,⟨b_|R|,K,e_|R|,K⟩⟩}
```

Illustratie B.2: Procedure $exec_{HOCLRT}$, die gebruik maakt van een HOCLRT wrapper om de relatie van een pagina te verkrijgen.

De foutieve constraint C^D stelt dat het tail scheidingsteken niet mag voorkomen na het open scheidingsteken en voor het scheidingsteken l_1, in de tekst die volgt na het head scheidingsteken in het hoofd van een pagina. De constraint stelt met andere woorden dat de volgorde *h*, *o*, *t* en l_1 verboden is. Wanneer we naar de $exec_{HOCLRT}$ procedure kijken, zien we dat er enkel op regel 4 gesproken wordt over het tail scheidingsteken. Op deze regel kijken we telkens of het eerstvolgende open scheidingsteken voorkomt voor het eerstvolgende tail scheidingsteken. Deze regel gaat met andere woorden alle tupels uit de pagina halen en heeft niets te maken met het scheidingsteken l_1. Een constraint die bij deze regel hoort, in verband met het hoofd van een pagina, moet bijgevolg erover waken dat de $exec_{HOCLRT}$ procedure ten minste één tupel uit de pagina kan halen. Dit kunnen we doen door een constraint te definiëren die stelt dat het tail scheidingsteken niet mag voorkomen voor het open scheidingsteken, in de tekst die volgt na het head scheidingsteken in het hoofd van een pagina. De volgorde *h*, *t* en *o* moet met andere woorden verboden zijn. Deze nieuwe en correcte constraint is formeel gedefinieerd in Illustratie B.3.

Constraint D: u_t mag niet voorkomen voor u_o, in de tekst die voorkomt na u_h, en dit in het hoofd van eender welke pagina

Constraint K: u_t mag niet voorkomen voor u_o, in de tekst die voorkomt na u_c, in de tekst tussen eender welke twee opeenvolgende tupels, en dit in eender welke pagina

ILLUSTRATIE B.3: CORRECTIE VAN DE FOUTIEVE CONSTRAINTS VAN DE HOCLRT WRAPPER CLASS.

De foutieve constraint C^K stelt hetzelfde over het tail scheidingstekens als constraint C^D, maar dan voor de tekst tussen tupels. De constraint stelt namelijk dat de volgorde *c*, *o*, *t* en l_1 verboden is. Deze constraint heeft net zoals constraint C^D enkel betrekking op regel 4 van de $exec_{HOCLRT}$ procedure. De correcte constraint die bij deze regel hoort, in verband met de tekst tussen tupels, moet erover waken dat het tail scheidingsteken niet te vroeg voorkomt in de pagina, met andere woorden tussen de tupels. De volgorde *c*, *t* en *o* moet bijgevolg verboden zijn. Deze nieuwe en correcte constraint is formeel gedefinieerd in Illustratie B.3.

B.2 VOORBEELD

Illustratie B.4 toont de vereenvoudigde broncode van de country-code webpagina die we in Hoofdstuk 3 als voorbeeld hebben gebruikt. In Hoofdstuk 4 hebben we gezien dat we een LR wrapper kunnen leren voor deze webpagina en aangezien de HOCLRT wrapper class een uitbreiding is van de LR wrapper class, moeten we ook een HOCLRT wrapper uit deze webpagina kunnen leren.

Wanneer we de $learn_{HOCLRT}$ procedure, die gebruik maakt van de foutieve constraints, uitvoeren op de broncode van Illustratie B.4, dan krijgen we als output de foutieve wrapper ⟨'<html>','','','',[⟨'',''⟩,⟨'',''⟩]⟩. Indien we de $exec_{HOCLRT}$ procedure uitvoeren met als input dezelfde broncode en de zojuist verkregen wrapper, dan krijgen we als output de relatie van Illustratie B.5. Dit komt omdat het tail scheidingsteken "" voorkomt voor het open scheidingsteken "" en na het close scheidingsteken "" in de tekst tussen tupels. (Herinner dat overlap tussen scheidingstekens is toegestaan.) Hierdoor stopt de while lus op regel 4 in de $exec_{HOCLRT}$ procedure na het extraheren van het eerste tupel.

```
1: <html><head><title>Some Country Codes</title></head><body>
2: <b>Belgium</b> <i>32</i><br>
3: <b>Congo</b> <i>243</i><br>
4: <b>Egypt</b> <i>20</i><br>
5: <b>Spain</b> <i>34</i><br>
6: </body></html>
```

ILLUSTRATIE B.4: VEREENVOUDIGDE BRONCODE VAN DE WEBPAGINA IN FIGUUR 3.1.

$$R = \{\langle \text{'Belgium','32'} \rangle\}$$

ILLUSTRATIE B.5: DE RELATIE DIE DOOR DE FOUTIEVE WRAPPER UIT DE BRONCODE VAN ILLUSTRATIE B.4 WORDT GEHAALD.

$$R = \begin{Bmatrix} \langle \text{'Belgium','32'} \rangle \\ \langle \text{'Congo','243'} \rangle \\ \langle \text{'Egypt','20'} \rangle \\ \langle \text{'Spain','34'} \rangle \end{Bmatrix}$$

ILLUSTRATIE B.6: DE RELATIE DIE WE UIT DE BRONCODE VAN ILLUSTRATIE B.4 WILLEN HALEN.

De $learn_{HOCLRT}$ procedure, die gebruik maakt van de correcte constraints, uitvoeren op de de broncode van Illustratie B.4, leidt tot een bijna identieke wrapper als output, namelijk ⟨'<html>','</html>','','',[⟨'',''⟩,⟨'',''⟩]⟩. Het enige verschil is dat het tail scheidingstekens is veranderd van een "" tag naar een "</html>" tag. Dit heeft tot gevolg dat het tail scheidingsteken "</html>" niet meer voorkomt voor het open scheidingsteken "" en na het close scheidingsteken "" in de tekst tussen tupels. De $exec_{HOCLRT}$ procedure haalt bijgevolg de correcte relatie uit de broncode van Illustratie B.4, wanneer we de nieuw verkregen wrapper als input meegeven. Deze relatie wordt weergegeven in Illustratie B.6.

Dit voorbeeld heeft betrekking op de scheidingstekens in de tekst tussen tupels, oftewel op constraint C^K. De lezer kan zelf een voorbeeld nagaan dat betrekking heeft op de scheidingstekens in het hoofd van een pagina, oftewel op constraint C^D.

Bijlage C: IMPLEMENTATIE

In deze bijlage worden de belangrijkste elementen van de implementatie besproken. We starten met de vereisten die nodig zijn om de programma's uit te kunnen voeren. Vervolgens gaan we verder met de mappenstructuur. Tot slot bespreken we hoe de ontwikkelde programma's opgestart moeten worden.

C.1 VEREISTEN

Om de verschillende programma's uit te kunnen voeren, zijn een g++ compiler en de HTML module[7] van de Libxml2[8] parser nodig.

We hebben gekozen voor de Libxml2 parser, omdat deze een HTML module bevat die geschikt is voor het parsen van webpagina's. Dit wil zeggen dat de parser overweg kan met tags die ontbreken, speciale HTML karakters, etc. De parser probeert de pagina namelijk zo goed mogelijk om te vormen tot een deftige webpagina. Hiermee bedoelen we dat de parser onder andere zelf tags sluit, die in de originele webpagina niet gesloten waren.

C.2 MAPPENSTRUCTUUR

De hoofdmap draagt de naam "Implementatie" en bevat de volgende vier submappen: "data", "doc", "obj" en "src". Deze worden hieronder elk kort besproken. Naast deze submappen bevat de hoofdmap ook een *makefile* waarmee de gebruiker de uitvoerbare bestanden (programma's) kan genereren.

De "data" map dient voor zowel de input als de output van elk programma. Merk op dat wanneer een gebruiker naar een bestand in deze map wil verwijzen, als input voor een programma, de bestandsnaam niet voorafgegaan moet worden door de naam van deze map. Het programma gaat namelijk altijd zelf in deze map opzoek naar bestanden.

De "src" map bevat alle broncode van de verschillende programma's.

In de "obj" map komen alle objectbestanden te staan die gegenereerd worden tijdens het compileren van de verschillende programma's.

De "doc" map bevat door doxygen gegenereerde documentatie van de broncode. De doxyfile waarmee deze documentatie gegenereerd werd, is ook aanwezig in deze map.

[7] Module HTMLparser from libxml2: http://xmlsoft.org/html/libxml-HTMLparser.html
[8] The XML C parser and toolkit of Gnome: http://xmlsoft.org/

C.3 PROGRAMMA'S

C.3.1 WRAPPERS

Het "Wrappers" programma laat een gebruiker toe om de vijf wrapper classes (LR, HLRT, OCLR, HOCLRT en UOCLR) te gebruiken. De gebruiker kan zowel de exec procedure als het learn algoritme van elke wrapper class uitvoeren.

Illustratie C.1 toont de opdrachtregel waarmee het "Wrappers" programma opgestart moet worden. Merk op dat het meegeven van parameters bij het opstarten van het programma optioneel is, maar indien dit niet gebeurt, dan zal het programma deze één voor één vragen tijdens de uitvoering. Deze parameters duiden namelijk alle informatie aan die het programma nodig heeft om een exec procedure of learn algoritme van een wrapper class uit te kunnen voeren.

```
./Wrappers [wrapper_class_number wrapper_class_function
                                          [wrapper_filename paga_filename]]
```

ILLUSTRATIE C.1: OPDRACHTREGEL WAARMEE HET "WRAPPERS" PROGRAMMA OPGESTART MOET WORDEN.

Het "Wrappers" programma heeft minimum twee en maximum vier parameters nodig. De eerste twee parameters geven aan welke wrapper class en welke functie van deze wrapper class de gebruiker wil uitvoeren. Dit kan de gebruiker aanduiden door het overeenkomstige wrapper class nummer en functie nummer op te geven. De wrapper classes zijn als volgt genummerd:

1) LR wrapper class
2) HLRT wrapper class
3) OCLR wrapper class
4) HOCLRT wrapper class
5) UOCLR wrapper class

De functies van een wrapper class zijn als volgt genummerd:

1) Exec
2) Learn

Wanneer de gebruiker het learn algoritme van een wrapper class wil uitvoeren, vereist het programma geen verdere input van de gebruiker. Het programma verwacht echter wel dat een bestand genaamd "ExamplePagesAndRelations.csv" aanwezig is in de "data" map. Dit bestand moet de bestandsnamen van alle voorbeeldpagina's en hun bijhorende relaties bevatten, waaruit het learn algoritme een wrapper moet leren voor de gekozen wrapper class. Illustratie C.2 toont een mogelijke inhoud voor dit bestand. Hierin zien we dat elke voorbeeldpagina en zijn bijhorende relatie vermeld wordt als een tupel van de vorm "page_filename;relation_filename". Elke bestandsnaam die in dit bestand vermeld

```
CountryCode1.html;CountryCode1_Relation.csv
CountryCode2.html;CountryCode2_Relation.csv
```

ILLUSTRATIE C.2: VOORBEELD VAN EEN INHOUD VAN HET BESTAND "EXAMPLEPAGESANDRELATIONS.CSV".

```
use common strings: yes
use tokenized strings: yes
sort candidates: asc
large inputs: no
```

ILLUSTRATIE C.3: VOORBEELD VAN EEN INHOUD VAN HET CONFIGURATIEBESTAND "CONFIGURATIONS.CONFIG".

wordt, moet eveneens aanwezig zijn in de "data" map. Naast de bestanden gerelateerd aan de voorbeeldpagina's en hun relaties, verwacht het "Wrappers" programma, voor de uitvoering van een learn algoritme, ook de aanwezigheid van een configuratiebestand in de "data" map. Dit bestand moet de naam "Configurations.config" dragen. In dit bestand kunnen de volgende vier configuraties ingesteld worden:

- use common strings
- use tokenized strings
- sort candidates
- large inputs

Illustratie C.3 toont een voorbeeld van de inhoud van het configuratiebestand. Hieronder worden de vier configuraties één voor één besproken.

De "use common strings" instelling accepteert twee waarden: "yes" of "no". Hiermee kan de gebruiker instellen of de gemeenschappelijke strings verbetering gebruikt moet worden tijdens het uitvoeren van een learn algoritme.

De "use tokenized strings" instelling accepteert twee waarden: "yes" of "no". Hiermee kan de gebruiker instellen of de tokenstrings verbetering gebruikt moet worden tijdens het uitvoeren van een learn algoritme.

De "sort candidates" instelling accepteert drie waarden: "asc", "desc" of "no". Hiermee kan de gebruiker instellen of de kandidaten voor elk scheidingsteken gesorteerd moeten worden van klein naar groot ("asc"), van groot naar klein ("desc"), of niet gesorteerd moeten worden ("no"). Deze instelling was alleen nodig om kandidaten van klein naar groot te kunnen sorteren bij de UOCLR wrapper class (Hoofdstuk 8). Voor alle andere wrapper classes stond deze instelling steeds op "no".

De "large input" instelling accepteert twee waarden: "yes" of "no". Hiermee kan de gebruiker instellen of een restrictie gelegd moet worden op de lengte van de

strings/tokenstrings, die zich bevinden in de set waaruit de generalized suffix tree wordt opgebouwd. Voor meer informatie over deze restrictie zie Sectie 10.1.5.

De output van een learn algoritme van een wrapper class, wordt weggeschreven naar het bestand "Extracted_Wrapper.csv".

Wanneer de gebruiker de exec procedure van een wrapper class wil uitvoeren, vereist het programma twee extra parameters, namelijk de naam van het bestand waarin zich een wrapper bevindt, die hoort bij de wrapper class die de gebruiker heeft gekozen, en de naam van het bestand dat een pagina bevat, waaruit de exec procedure een relatie moet halen. Deze bestanden moeten beide aanwezig zijn in de "data" map. Naast deze extra input parameters vereist het "Wrappers" programma voor de uitvoering van de exec procedure, net zoals het learn algoritme, de aanwezigheid van het configuratiebestand "Configurations.config". De exec procedure moet namelijk weten of de tokenstrings verbetering actief was tijdens het leren. De reden hiervoor is als volgt: wanneer de tokenstring verbetering actief is, worden tag attributen genegeerd. Tag attributen worden bijgevolg ook niet opgenomen in een scheidingsteken van de geleerde wrapper. De echte pagina's bevatten echter wel tags met tag attributen. Een tag met tag attributen (pagina's) komt echter niet overeen met een tag zonder tag attributen (scheidingstekens). Indien de exec procedure tag attributen negeert, wanneer tag attributen genegeerd werden voor het leren van een wrapper, dan doet dit probleem zich niet voor. De tokenstrings verbetering is de enige instelling die de exec procedure nodig heeft uit het configuratiebestand.

De output van de exec procedure van een wrapper class wordt weggeschreven naar het bestand "Extracted_Relation.csv".

C.3.2 Relation extractor

Het tweede programma is genaamd "RelationExtractor" en helpt de gebruiker bij het maken van een verzameling van voorbeeldpagina's en hun bijhorende relaties. De relatie die bij een voorbeeldpagina hoort, moet namelijk een index relatie zijn (zie Sectie 3.3). Het handmatig zoeken van de start- en eindpositie van elke attribuutwaarde is echter onbegonnen werk. Het programma helpt hierbij door automatisch de index relatie uit een pagina te halen, waarin de attribuutwaarden gemarkeerd zijn volgens een aantal regels. De markeerregels luiden als volgt:

- De eerste attribuutwaarde van elk tupel wordt voorafgegaan door "***".
- De eerste attribuutwaarde van elk tupel wordt gevolgd door "###".
- Alle andere attribuutwaarden van elk tupel worden voorafgegaan door "###".
- Alle andere attribuutwaarden van elk tupel worden gevolgd door "###".

```
1: <html><head><title>Some Country Codes</title></head><body>
2: <b>***Belgium###</b> <i>###32###</i><br>
3: <b>***Congo###</b> <i>###243###</i><br>
4: <b>***Egypt###</b> <i>###20###</i><br>
5: <b>***Spain###</b> <i>###34###</i><br>
6: </body></html>
```

ILLUSTRATIE C.4: GEMARKEERDE BRONCODE VAN EEN WEBPAGINA.

```
63,70;78,80
88,93;101,104
112,117;125,127
135,140;148,150
```

ILLUSTRATIE C.5: INDEX RELATIE DIE HET "RELATIONEXTRACTOR" PROGRAMMA UIT DE GEMARKEERDE BRONCODE VAN ILLUSTRATIE C.4 HAALT.

Illustratie C.4 toont de gemarkeerde broncode van een webpagina en Illustratie C.5 toont de index relatie die het "RelationExtractor" programma hieruit haalt.

Illustratie C.6 toont de opdrachtregel waarmee het "RelationExtractor" programma opgestart moet worden. Merk op dat het meegeven van parameters bij het opstarten van het programma optioneel is, maar indien dit niet gebeurt, dan zal het programma deze één voor één vragen tijdens de uitvoering. Deze parameters duiden namelijk alle informatie aan die het programma nodig heeft. De eerste parameter is de bestandsnaam van een pagina die gemarkeerd is volgens bovenstaande regels. Dit bestand moet zich in de "data" map bevinden. De tweede parameter duidt het aantal attributen aan dat elk tupel van de relatie op de pagina heeft.

```
./RelationExtractor [marked_page_filename number_of_attributes]
```

ILLUSTRATIE C.6: OPDRACHTREGEL WAARMEE HET "RELATIONEXTRACTOR" PROGRAMMA OPGESTART MOET WORDEN.

Het "RelationExtractor" programma vereist de aanwezigheid van het configuratiebestand "Configurations.config", in de "data" map. Wanneer een learn algoritme de tag attributen negeert, moet ook de index relatie die hierbij hoort de tag attributen genegeerd hebben. Indien dit niet gebeurd is komen de start- en eindposities van de attribuutwaarden in de index relatie niet overeen met de start- en eindposities van de attribuutwaarden in de geparste webpagina.

Wanneer de bestandsnaam van een gemarkeerde pagina bijvoorbeeld "CountryCode _Marked.html" is, dan wordt de output van het "RelationExtractor" programma weggeschreven naar een bestand met de naam "CountryCode_Relation.csv". Wanneer de suffix "_Marked.html" niet aanwezig is in de naam van het inputbestand, dan wordt de

naam van het outputbestand hetzelfde als de naam van het input bestand, maar dan wordt "_Relation" op het einde van de bestandsnaam toegevoegd.

C.3.3 TEST WRAPPERS

Het laatste programma, genaamd "TestWrappers", dient om de output van de exec procedure van verschillende wrapper classes met elkaar te vergelijken, en dit voor een verzameling van pagina's. Illustratie C.7 toont de opdrachtregel waarmee dit programma opgestart moet worden. Merk op dat er geen parameters meegegeven kunnen worden tijdens het opstarten. Desondanks zijn er toch parameters, zoals de hoeveelheid pagina's die zich in de verzameling bevinden, de wrapper classes wiens exec procedure uitgevoerd moet worden, de namen van de bestanden die de wrappers bevatten, etc. Deze parameters kunnen enkel in de broncode worden aangepast, meer bepaald in het begin van de "executeTest" functie.

```
./TestWrappers
```

ILLUSTRATIE C.7: OPDRACHTREGEL WAARMEE HET "TESTWRAPPERS" PROGRAMMA OPGESTART MOET WORDEN.

Net zoals de twee voorgaande programma's vereist het "TestWrappers" programma de aanwezigheid van enkele files. Allereerst vereist het programma dat de verzameling van pagina's aanwezig is een map genaamd "pages", in de "data" map. De bestandsnamen van de pagina's moeten van de vorm "index.html.5" zijn, waarbij het nummer op het einde van de naam, het paginanummer voorstelt, beginnende bij 0. Verder wordt ook de aanwezigheid geëist van een wrapper voor elke wrapper class wiens exec procedure uitgevoerd moet worden. Deze bestanden moeten aanwezig zijn in de "data" map. Als laatste wordt de aanwezigheid van het configuratiebestand "Configurations.config", in de "data" map, vereist. Dit omwille van dezelfde reden als het "Wrappers" programma de aanwezigheid van dit bestand eiste bij de uitvoering van een exec procedure (zie Sectie C.3.1).

De output van het "TestWrappers" programma wordt weggeschreven naar het bestand "TestWrappers.txt".

Auteursrechtelijke overeenkomst

Ik/wij verlenen het wereldwijde auteursrecht voor de ingediende eindverhandeling:
Keeping up with Google: Searching in Text Databases

Richting: **master in de informatica-databases**
Jaar: **2014**

in alle mogelijke mediaformaten, - bestaande en in de toekomst te ontwikkelen - , aan de Universiteit Hasselt.

Niet tegenstaand deze toekenning van het auteursrecht aan de Universiteit Hasselt behoud ik als auteur het recht om de eindverhandeling, - in zijn geheel of gedeeltelijk -, vrij te reproduceren, (her)publiceren of distribueren zonder de toelating te moeten verkrijgen van de Universiteit Hasselt.

Ik bevestig dat de eindverhandeling mijn origineel werk is, en dat ik het recht heb om de rechten te verlenen die in deze overeenkomst worden beschreven. Ik verklaar tevens dat de eindverhandeling, naar mijn weten, het auteursrecht van anderen niet overtreedt.

Ik verklaar tevens dat ik voor het materiaal in de eindverhandeling dat beschermd wordt door het auteursrecht, de nodige toelatingen heb verkregen zodat ik deze ook aan de Universiteit Hasselt kan overdragen en dat dit duidelijk in de tekst en inhoud van de eindverhandeling werd genotificeerd.

Universiteit Hasselt zal mij als auteur(s) van de eindverhandeling identificeren en zal geen wijzigingen aanbrengen aan de eindverhandeling, uitgezonderd deze toegelaten door deze overeenkomst.

Voor akkoord,

Vuurstaek, Jan

Datum: **22/08/2014**