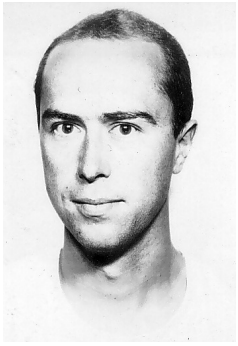
ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Institute of
Theoretical Computer Science
Prof. Dr. Jürg Nievergelt

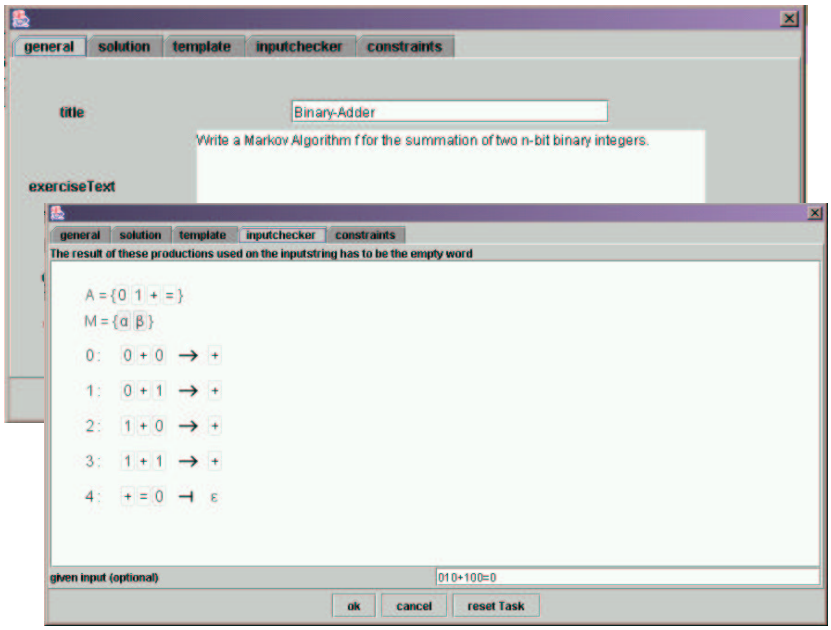
Semesterarbeit

25. März 2002
11. Juli 2002

Aufgabeneditor zu Markov Umgebung



Thomas Briner



Assistenz
Vincent Tscherter
Reto Lamprecht

www
tedu.ethz.ch/exorciser/

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Institute of
Theoretical Computer Science
Prof. Dr. Jürg Nievergelt

Semesterarbeit
25. März 2002
11. Juli 2002

Aufgabeneditor zu Markov Umgebung

Thomas Briner

Zusammenfassung

Einleitung Die Semesterarbeit ist ein Teil des Projekts „Exorciser - Automated generation and grading of exercises in an introductory course on the Theory of Computation". Mit dem Exorciser, einer Sammlung von Übungen für eine Einführungsvorlesung in die Berechenbarkeitstheorie, sollen die Teilnehmer selbstständig Übungen lösen, wobei ihr Können durch diese interaktive Übungsumgebung beurteilt wird.

Ziel Ziel der Semesterarbeit ist es, zu einer bestehenden Umgebung für Markov Algorithmen einen graphischen Aufgabeneditor zu entwickeln, mit dem neue Aufgaben erstellt oder bestehende modifiziert werden können, ohne Kenntnisse der Filestruktur vorauszusetzen. Darüber hinaus sollen Kommentare und Anregungen zum Markov Plugin aus Benutzersicht abgegeben werden.

Resultate Als Resultat dieser Semesterarbeit können nun Aufgaben zum Markov Plugin mittels eines graphischen Benutzerinterfaces editiert werden.

Ausblick Da die Modifikation des Fileformats mit allen dadurch erforderlichen Anpassungen einen erheblichen Teil der Zeit in Anspruch nahm und das Schwergewicht deshalb auf dem Erfüllen der Funktionalität lag, wären am Editor sicher noch optische Verbesserungen möglich.

Ob und wie die Aenderungsvorschläge am Markov Plugin umgesetzt werden sollen, bleibt noch zu entscheiden.

Assistenz Vincent Tscherter – tscherter@inf.ethz.ch
Reto Lamprecht – lamprecht@inf.ethz.ch

www tedu.ethz.ch/exorciser/

Einleitung

Seit Frühling 2002 enthält das Exorciser Framework[1] ein weiteres Plugin zum Thema der Markov Algorithmen[2]. In diesem Plugin besteht einerseits die Möglichkeit, in einer Testumgebung erste Erfahrungen mit Markov Algorithmen zu sammeln und andererseits können vordefinierte Aufgaben gelöst werden.

Ziel der Semesterarbeit war es, zu einer bestehenden Umgebung für Markov Algorithmen einen graphischen Aufgabeneditor zu entwickeln, mit dem neue Aufgaben erstellt oder bestehende modifiziert werden können, ohne Kenntnisse der Filestruktur vorauszusetzen. Darüber hinaus sollen Kommentare und Anregungen zum Markov Plugin aus Benutzersicht abgegeben werden.

Struktur einer Aufgabe im Markov Plugin

Eine Aufgabe setzt sich aus den folgenden Teilen zusammen:

Aufgabenbeschrieb: Darin wird die Aufgabenstellung in textueller Form definiert und eventuell mit Hilfe eines Beispiels noch verdeutlicht.

Constraints: Hier werden die Rahmenbedingungen festgelegt, wie zum Beispiel die obere Schranke für die Anzahl Schritte für die Abarbeitung des Algorithmus'.

Benutzergrammatik: Eine Markov Grammatik besteht aus den Elementen des Alphabets, den Markern und den verschiedenen Produktionen. Das ist die Grammatik, auf welcher der Benutzer arbeitet.

Musterlösung: Anhand dieser Musterlösung kann das Ergebnis der Benutzergrammatik überprüft werden. Durch die Check-Funktion des E-Assistants wird diese Ueberprüfung gestartet. Die Musterlösung ist ebenfalls eine komplette Markov Grammatik.

Input: Es kann zur Aufgabe ein Input spezifiziert werden.

Inputüberprüfungsgrammatik: Mit Hilfe dieser Grammatik kann ein Input auf seine Gültigkeit überprüft werden. Die Produktionen werden dabei auf den Input angewendet. Aus einem gültigen Input muss der leere String resultieren.

Die Aufgaben im Markov Plugin werden im Gegensatz zu anderen Teilen des Exorciser Frameworks nicht zufällig generiert, sondern müssen von Hand erstellt werden. Sie sind in XML Files gespeichert, was sich aufgrund der stereotypen Strukturen, zum Beispiel bei den Produktionen, anbietet. Ein solches File ist für ein Programm einfach zu parsen, der Nachteil liegt aber in der Länge und der Unüberschaubarkeit des Files für den Author einer Aufgabe. Diese Problematik wird durch den Editor beseitigt, so dass zum Erstellen und Verändern von Aufgaben keinerlei Kenntnisse über die Struktur des gespeicherten Files benötigt werden.

Der Editor ist primär auf Dozierende und Mitarbeiter zugeschnitten, um neue Aufgaben zu gestalten und weniger als Hilfsmittel für Studierende konzipiert.

Die zugrundeliegende Idee ist, dass das Framework schon mit einigen Aufgaben ausgeliefert wird, auf einem Server aber noch weitere Aufgaben zum Download bereitstehen, wobei diese Sammlung je nach Bedarf noch erweitert werden kann.

[1] http://www.tedu.ethz.ch/exorciser

[2] Daniel Müller, Diplomarbeit, "Interaktive Lernkomponente für Markov Algorithmen"

Die XML-Struktur, in der die Files gespeichert waren, wies aber im Hinblick auf den Aufgabeneditor verschiedene Probleme auf. Deshalb musste die Struktur angepasst werden. Folgende Punkte wurden hierbei verändert:

- Die drei oben erwähnten Grammatiken arbeiteten alle auf der gleichen Menge von Markern, da im File nur ein tag für die Marker vorgesehen war. Es besteht aber keine Notwendigkeit, dass der Benutzer die gleichen Marker benützt wie beispielsweise die Musterlösung. So soll jede Grammatik ihr eigenes Set von Markern haben, welches im Editor dann auch separat modifiziert werden kann. Deshalb wurde die XML-Struktur dahingehend geändert, dass die Marker nun zu Childnodes der verschiedenen Grammatiken wurden.
- Es hatte tags, die nicht einheitlich gehandhabt, beziehungsweise nur in einem Teil der Aufgaben überhaupt benutzt wurden. Das waren hauptsächlich die tags "renderer", mit welchem die graphische Darstellung festgelegt werden konnte, und "exercisetype", mit welchem zwischen den drei Modi "write from scratch", "reorder productions" und "look at solution" differenziert werden konnte. Diese Unterscheidung war aber bereits im Filenamen enthalten und deshalb auch nicht konsequent ausgefüllt. Diese beiden Tags sind in der neuen Filestruktur ganz weggefallen.
- In der alten XML-Struktur wurde das tag "el" sowohl für die Symbole des Alphabets und die Marker verwendet, als auch für die ganzen Produktionen. Um die Verständlichkeit zu erhöhen, wurden deshalb in der neuen Struktur zwei separate tags für die beiden Anwendungen benutzt.

Die Document Type Definition (DTD) der neuen Struktur sieht nun folgendermassen aus:

```
<!ELEMENT exercise (title, exercisetext, definition, example, alphabetElem,
                    answer, solution, input?, inputchecker, constraints?)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT exercisetext (#PCDATA)>
<!ELEMENT definition (#PCDATA)>
<!ELEMENT example (#PCDATA)>
<!ELEMENT alphabetElem (symbol*)>
<!ELEMENT symbol (#PCDATA)>
<!ELEMENT answer (production*, marker*)>
<!ELEMENT production (leftside?, rightside?)>
<!ELEMENT leftside (left*)>
<!ELEMENT left (#PCDATA)>
<!ELEMENT rightside (right*)>
<!ELEMENT right (#PCDATA)>
<!ELEMENT marker (symbol*)>
<!ELEMENT solution (production+, marker*)>
<!ELEMENT input (#PCDATA)>
<!ELEMENT inputchecker (production*, marker*)>
<!ELEMENT constraints (runtime?, stringsize?, max_len_of_production?,
                       max_input_length?, max_nof_productions?,
                       max_nof_terms?, max_nof_vars?)>
<!ELEMENT runtime (#PCDATA)>
<!ELEMENT stringsize (#PCDATA)>
<!ELEMENT max_len_of_production (#PCDATA)>
<!ELEMENT max_input_length (#PCDATA)>
<!ELEMENT max_nof_productions (#PCDATA)>
<!ELEMENT max_nof_terms (#PCDATA)>
```

```
<!ELEMENT max_nof_vars (#PCDATA)>
<!ATTLIST production
        stop (true | false) #REQUIRED
>
<!ENTITY false "false">
<!ENTITY true "true">
```

Durch diese Aenderungen an der Filestruktur wurden nun aber auch Modifikationen an sämtlichen Punkten im Markov Plugin notwendig, an denen Daten aus den Files gelesen und in die Files geschrieben werden.

Zusätzlich mussten alle bestehenden Files in das neue Format übertragen werden, was mittels eines in Java geschriebenen FileConverters geschah.

Konkrete Implementierung des Aufgabeneditors

Der Aufgabeneditor arbeitet auf einem Klon der Aufgabe, der beim Start des Editors erzeugt wird. Beim Klick auf den “ok”-Button werden die Aenderungen übernommen. Mittels des "reset task"-Buttons werden alle Einstellungen zurückgesetzt. Das bedeutet, dass der Inhalt der Textfelder und sämtliche Produktionen gelöscht werden, die Mengen der Marker und das Alphabet werden auf die leere Menge gesetzt und die Constraints werden wieder mit den Defaultwerten initialisiert.

Der Editor ist in Form eines JTabbedPane (Karteikarten) gestaltet, das aus den fünf Karten "general", "solution", "template", "inputchecker" und "constraints" besteht.

Die Karten "solution", "template" und "inputchecker" bestehen jeweils aus dem bekannten Drag&Drop Produktionseditor, in welchem die jeweilige Grammatik modifiziert werden kann. Die Modifikationen werden dann aber ohne irgendwelche weiteren Kontrollen übernommen. Das heisst, dass vorausgesetzt wird, dass, wer Aufgaben modifiziert, mit den Markov Algorithmen vertraut sein muss. Ansonsten können beispielsweise bei Endlosschleifen in der Musterlösung unerwünschte Effekte auftreten.

Im Weiteren musste darauf geachtet werden, dass die Alphabete, die ja in allen drei Grammatiken verändert werden können, immer konsistent sind.

Auf der Karte "general" wird die Aufgabenstellung definiert. Unter "constraints" können die Schranken für Laufzeit, Stringlänge und diverse andere Parameter gesetzt werden. Leere Felder auf der "constraints"-Karte bedeuten, dass diese Parameter nicht spezifisch für diese Aufgabe gesetzt sind, sondern die Defaultwerte Gültigkeit haben. Dieselbe Konvention wird auch bei der Darstellung der Aufgabenstellung im Markov Plugin verwendet, um den Benutzer nicht unnötig durch das Anzeigen aller Constraints zu verwirren, die zum grossen Teil gar nicht explizit für diese Aufgabe gesetzt wurden.

Kommentare zur Benutzung des gesamten Markov Plugins aus Sicht der Studierenden

Im Gesamten schätze ich das gesamte Markov Plugin für das Verständnis der Studierenden für Markov Algorithmen und im Speziellen für eine gewisse Uebung im Umgang damit als sehr wertvoll ein. Besonders die Visualisierung der Abarbeitung einer Grammatik, die ja von Hand sehr aufwändig und fehleranfällig ist, lässt sich hier hilfreich einsetzen.

Um möglichst effizient mit der Uebungsumgebung zu arbeiten, empfiehlt es sich, den Algorithmus zuerst vollständig von Hand zu entwerfen und die eigene Lösung dann erst mit dem Plugin zu testen. Für ein eigentliches Entwerfen und Ausprobieren von einzelnen Produktionen innerhalb der Umgebung ist die Hilfestellung, die der Static Checker bietet, der die Produktionen zum Beispiel auf Endlosschleifen überprüft, wohl schon zu weit fortgeschritten.

Der Ueberprüfungsalgorithmus, der das Resultat der Benutzergrammatik mit dem Resultat der Musterlösung vergleicht, müsste noch genau getestet werden, um die Spezialfälle, in denen Studierende vom E-Assistant falsche Anworten erhalten haben, zu eliminieren.

Die Idee mit den verschiedenen Modi müsste noch einmal überdacht werden. Der Modus "reorder productions" macht für das Verständnis der Markov Algorithmen nur bedingt Sinn, muss doch beim Erstellen einer Grammatik von Grund auf diese Frage ebenfalls beantwortet werden. Es besteht hier die Gefahr, dass die Regeln einfach mittels Versuch und Irrtum hinauf- und hinuntergeschoben werden, um so das korrekte Resultat zu erhalten. Mein Vorschlag wäre, dass es nur noch den Modus "write from scratch" gäbe, da auch in diesem Modus über den E-Assistant die Musterlösung angeschaut werden kann. So ist auch der Modus "look at solution" ebenfalls überflüssig.

In der Aufgabensammlung hat es Aufgaben, die mehrfach vorhanden sind ("multiplication", "double" und "reverse"). Diese Aufgaben differieren nur in der Musterlösung. Für den Benutzer, der die Aufgaben aber "from scratch" löst, ist es völlig uneinsichtig, wieso die gleiche Aufgabenstellung mehrfach vorkommt. Es würde genügen, jede Aufgabe nur einmal in der Sammlung zu haben mit einer der möglichen Lösungen.