

Inhaltsverzeichnis

Danke

xiii

1. Einleitung	1
1.1. Modulares Constraint Checking im Überblick	3
1.1.1. Herausforderungen moderner Lehr-/Lernumgebungen	3
1.1.2. Was ist Modulares Constraint Checking?	4
1.1.3. Komponenten des MCC-Ansatzes	9
1.1.4. Wie das MCC-System verwendet wird	11
1.2. Konsequenzen des MCC-Ansatzes	13
1.2.1. MCC ist domänenunabhängig	13
1.2.2. Flache Wissensrepräsentation	15
1.2.3. Was für Feedback?	17
1.3. Wichtige technische Aspekte des MCC-Ansatzes	20
1.3.1. Vergleich von Zuständen mit Lösungen	20
1.3.2. Wenn der Lernende Fehler macht...	23
1.4. Didaktische Relevanz des MCC-Ansatzes	24
1.4.1. Ein einfaches Beispiel	24
1.4.2. Feedback für visuelle Sprachen	25
1.4.3. Ausweitung der Gruppe der Autoren	27
1.4.4. Schülerinnen und Schüler befähigen	27
1.4.5. Didaktische Einordnung des MCC	29
1.5. Erste Zusammenfassung und Ausblick	32
1.5.1. Vergleich mit wissensbasierten ITS	32
1.5.2. Aufbau der Arbeit	33
2. Grundlegungen	37
2.1. Überblick über Intelligente Tutor Systeme	38
2.1.1. Die vier Bestandteile eines ITS	38
2.1.2. Typen von ITS	40
2.1.3. Anforderungen an jedes gute ITS(-Autoren-System)	43

v

2.1.4.	Widersprüchliche Design-Anforderungen	44
2.1.5.	Einordnung von MCC in das Spektrum der ITS . .	45
2.2.	Visuelle Sprachen	46
2.2.1.	Visuelle Sprachen - Definition	46
2.2.2.	Forschungsbereiche auf dem Gebiet visueller Sprachen	48
2.2.3.	Klassifizierung allein auf Basis des Erscheinungsbilds	50
2.2.4.	Verhältnis von einer „Sache“ und ihrer visuellen Re- präsentation	53
2.2.5.	Visuelle Sprachen und MCC	56
2.3.	Interaktive und kooperative Modellierungsumgebungen . .	60
2.3.1.	Das Direct Manipulation Paradigma	60
2.3.2.	Cool Modes	65
2.4.	Programmieren mit Constraints	75
2.4.1.	Definition eines Constraint	75
2.4.2.	Lokale Konsistenz	76
2.4.3.	Definition von Kantenkonsistenz	76
2.4.4.	Algorithmen zur Herstellung lokaler Konsistenz . .	77
2.4.5.	Finden einer Lösung	78
2.4.6.	Heuristiken für die Lösung von CSPs	79
2.4.7.	Beste Annäherungen in überbestimmten Netzen . .	80
2.4.8.	MCC und Constraint-Netze	81
2.5.	Kategorisierung von Analyse-Methoden	83
2.5.1.	Klassifizierungs-Möglichkeiten	83
2.5.2.	Beispiele	86
2.6.	Zusammenfassung	90
3.	Relevante Arbeiten zum „Light Weight Tutoring“	93
3.1.	Der „Plug-In Tutor“	94
3.1.1.	Beschreibung der Idee	94
3.1.2.	Beispiel-Implementierungen	98
3.1.3.	Diskussion	100
3.2.	Behavior Recording	104
3.2.1.	Beschreibung des Konzepts	104
3.2.2.	Diskussion des Behavior Recording	106
3.3.	CTAT und Pseudo-Tutor	110
3.3.1.	Beschreibung der CTAT	110

3.3.2.	Pseudo-Tutoren	112
3.3.3.	Diskussion	112
3.4.	Constraint Based Student Modeling	114
3.4.1.	Motivation	114
3.4.2.	Der Ansatz von Ohlsson	117
3.4.3.	Diskussion	120
3.5.	Design-Kritik-Systeme	122
3.5.1.	Unterstützung des Design-Prozesses	122
3.5.2.	Das SER-Modell	123
3.5.3.	Kontextualisierung	124
3.5.4.	Der JANUS Kitchen Designer	125
3.5.5.	Relevanz von Kritik-Systemen	129
3.6.	Der andere Weg: Hot Potatoes	132
3.6.1.	Die fünf „Kartoffeln“	132
3.6.2.	Diskussion	134
3.7.	Zusammenfassung	136
4.	Modulares Constraint Checking im Detail	139
4.1.	Überblick	139
4.1.1.	Blickwinkel von außen	140
4.1.2.	Blickwinkel von innen	141
4.1.3.	Aufbau der Systembeschreibung	143
4.2.	Anwendergruppen	145
4.2.1.	Die klassischen Rollen von Benutzer und System-Entwickler	145
4.2.2.	Rollen von Systementwickler und Nutzer bei Lehr-/Lernsystemen	146
4.2.3.	Zusätzliche Rolle bei Lehr-/Lernsystemen: Autoren	147
4.2.4.	Notwendige Qualifikationen der Rollen-Inhaber . .	148
4.2.5.	Unterstützung der klassischen Rollen	149
4.2.6.	Schüler als Anwender <i>und</i> Autoren	150
4.2.7.	Lehrer als Autoren <i>und</i> Systementwickler	152
4.3.	Wie werden Systemzustände repräsentiert?	153
4.3.1.	Domänenunabhängigkeit erreichen	153
4.3.2.	Wie es nicht geht	155

4.3.3.	Repräsentation von Applikationsobjekten im MCC-Tutor	156
4.3.4.	Erläuterung der Definitionen	157
4.3.5.	Autoren können neue Attribut-Typen ergänzen . .	161
4.3.6.	Spezifikation zur Laufzeit	162
4.3.7.	Vorzüge des MCC-Ansatzes zur Attribut-Erzeugung	163
4.4.	Definieren von Zielkonfigurationen	164
4.4.1.	Aufbau eines Bedingungsbaums zur Zielspezifikation	165
4.4.2.	Beschreibung einer Zielkonfiguration im Zusammen- hang	169
4.4.3.	Kapselung der Semantik	170
4.4.4.	Häufig nützliche Bedingungstypen	173
4.4.5.	Ein konkretes Beispiel für einen Bedingungsbaums .	178
4.4.6.	Überlegungen zur Komplexität	180
4.4.7.	Zusammenfassung	182
4.5.	Überprüfung von Prozessen	182
4.5.1.	Von der Zustandsüberprüfung zur Überprüfung von Abläufen	182
4.5.2.	Aufbau eines Prozessor-Graphen	183
4.5.3.	Unterschiedliche Traversierungs-Strategien	185
4.5.4.	Kantengewichte im Prozessor-Graphen	186
4.5.5.	Stopper	186
4.5.6.	Diskussion: Ist ein Prozessor-Graph ein Behavior- Graph?	187
4.6.	Automatische Zielspezifikation: „Aspekte“	189
4.6.1.	Verwendung von Aspekten: Snapshots	189
4.6.2.	Vordefinierte Aspekte	190
4.6.3.	Ein Aspekt ist mehr als die Summe seiner Attribute	191
4.7.	Der Checking-Algorithmus	193
4.7.1.	Der Algorithmus kann kaum optimiert werden . . .	193
4.7.2.	Der Checking-Algorithmus: Tiefensuche im Bedin- gungsbaum	195
4.7.3.	Worst-Case-Laufzeitverhalten	196
4.8.	Feedback	197
4.8.1.	Gestaffeltes Feedback	197
4.8.2.	Feedback-Arten	198

4.8.3. Output-Strategien	199
5. Anwendungsfunktionen eines MCC-Tutors	201
5.1. Semantische Illusion erzeugen	201
5.1.1. Was ist „Semantische Illusion“?	201
5.1.2. MCC und „Semantische Illusion“	202
5.1.3. Warum „Semantische Illusion“ verwenden?	203
5.1.4. Wann kann „Semantische Illusion“ verwendet werden? 203	
5.1.5. Wie kann man eine „Semantische Illusion“ erzeugen? 204	
5.1.6. „Semantische Illusion“ in der Praxis erzeugen	206
5.2. Integration eines MCC-Tutors in eine beliebige Applikation	209
5.2.1. Integration 1: Lernsoftware für Benutzer anpassen	210
5.2.2. Integration 2: Die Lernsoftware für Autoren anpassen	211
5.2.3. Anpassung von Komponenten des MCC-Tutors	213
5.3. Tutoring in Mehrbenutzer-Szenarien	215
5.3.1. Mit einem MCC-Tutor mögliche Unterstützung	216
5.3.2. MCC-Tutoren in Mehrbenutzer-Szenarien	217
5.4. Anwendungsbereiche für MCC-Tutoring	218
5.4.1. Wann lohnt sich der Einsatz von MCC?	218
5.4.2. MCC und visuelle Sprachen	219
5.4.3. Zusammenarbeit mit Cool Modes	219
5.4.4. Üben, nicht neuer Stoff	220
5.5. MCC reduziert die Kosten für Computer-Tutoring	220
5.5.1. Kein Vollständigkeit \Rightarrow Skalierbarkeit	221
5.5.2. Aspekte	222
5.5.3. Rapid Testing	222
5.6. Überlegungen zu Problemen	223
5.6.1. Vernünftiges Error-Handling durch View-orientierte Regeln	223
5.6.2. Verzicht auf automatische Feedback-Erzeugung ist auch ein Vorteil	224
5.6.3. Finden des „nächsten Schrittes“ durch den Computer	225
5.6.4. Unterschied zwischen Verständnis und gutem Raten	228
5.6.5. Beschränkung auf Üben, nicht auf neues Lernen	229
5.7. Grenzen des MCC-Ansatzes	231
5.7.1. Grenzen der Integration in andere Applikationen	231

5.7.2.	Fehlendes Gedächtnis	232
5.7.3.	Grenzen der Unterstützung von Kooperation	232
5.7.4.	Grenzen der Benutzerfreundlichkeit	233
6.	Szenarien zur praktischen Verwendung des MCC-Tutors	235
6.1.	Der MCC-Tutor als Kontexthilfe-System	236
6.1.1.	Das <i>Stochastic Experimenting</i> Plug-in für Cool Modes	236
6.1.2.	Bedarf für die Integration eines MCC-Tutors	239
6.1.3.	Spezifikation der Lösung	240
6.1.4.	Arbeitsteilung	242
6.1.5.	Konfiguration der Hilfe für den <i>Coin Node</i>	244
6.1.6.	Komplexer: Simple Drawing Node	249
6.1.7.	Einsatz des Feedback-Systems	253
6.2.	Verkehrserziehung	253
6.2.1.	Einführung	253
6.2.2.	Beschreibung des Szenarios	254
6.2.3.	Man stelle sich vor, das mit einem <i>intelligenten</i> Tutor umzusetzen...	256
6.2.4.	Modellierung mit einem MCC-Tutor	258
6.2.5.	Verwendung von Aspekten	260
6.2.6.	Aus Zuständen werden Regeln	261
6.3.	Zusammenfassung	264
7.	Zusammenfassung und Ausblick	267
7.1.	Mögliche Weiterentwicklungen von MCC und <i>ImpleCC</i> . .	267
7.2.	Zusammenfassung	270
A.	Anhang: Implementierung eines MCC-Tutors	273
A.1.	Aufbau des Systems im Überblick	273
A.1.1.	Einführung: Checker und Visuelle Entwicklungsum- gebung	273
A.1.2.	Systemkomponenten	273
A.1.3.	Die Beispiel-Applikation	275
A.2.	Die Benutzerschnittstelle von <i>ImpleCC</i>	276
A.2.1.	Checker.class : Die Schnittstelle zu anderen Appli- kationen	276
A.2.2.	Die Datenstruktur CheckerConfiguration	278

A.2.3. Integration des Checkers in die Beispiel-Applikation	281
A.3. Input – Gewinnen von Informationen	283
A.3.1. Das Ziel: Jedes Programm überprüfbar machen	283
A.3.2. Server für Kommunikation über „größere Distanz“	285
A.3.3. Repräsentation von Daten innerhalb des MCC	286
A.3.4. Die Schnittstelle PropertyCalculator	289
A.3.5. Beispiel: Informationsgewinnung in Anwendungssituation	298
A.4. Spezifikation von Zielkonstellationen	303
A.4.1. Allgemeines zu Bedingungen	303
A.4.2. Die Standard-Bedingungen	305
A.4.3. Die Standard-Verknüpfungen	312
A.4.4. Implementierung von Bedingungsäumen	314
A.4.5. Snapshots	316
A.5. Der Checker-Kern	318
A.5.1. Überprüfung einer Zielkonfiguration als Lösung eines CSP	318
A.5.2. Der Überprüfungs-Algorithmus und sein Ergebnis	318
A.6. Rückmeldungen an den Benutzer erzeugen	324
A.6.1. Trennung von Überprüfung und Feedback-Generierung	324
A.6.2. Verschiedene Arten von Feedback	326
A.6.3. Komposition von Feedback mit OutputStrategy	328
A.7. Die Implementierung des Prozessor-Konzeptes	329
A.8. Die User Interfaces von <i>ImpleCC</i>	334
A.8.1. Editable als Wegbereiter einfacher Visualisierung	335
A.8.2. CheckerMainFrame	338
A.8.3. Integration in andere Applikationen am Beispiel Cool Modes	342