ALLEN'SCHE ZEITLOGIK

Funktionserweiterung der Allen'schen Logik

Ephraim Petry &
Patrick Robinson

Einleitung

Im vorliegenden Projekt ging es um die Umsetzung einer Software zum Einlesen von Ereignissen und anschließenden zeitlichen Prüfen der zeitlichen Relationen zwischen diesen Ereignissen. Die Eingabe sollte dabei auch einen Existenzquantor verarbeiten und Aussagen mit diesem auf Korrektheit prüfen können. Dabei sollte z.B. angegeben werden können, dass ein Ereignis A mit mindestens einem der Ereignisse {B, C, D, E} in einer Beziehung steht.

Über diese gegebene Aufgabenstellung hinaus, sollte aus persönlichem Ehrgeiz die Verwendung mehrerer hintereinander gereihter Existenzquantoren ermöglicht werden.

Zur Umsetzung und insbesondere um die in der Vorlesung erworbenen Kenntnisse der Sprache LISP zu vertiefen, wurde zur Implementierung die Sprache LISP gewählt. Ein weiterer Grund für die Umsetzung in LISP liegt in der Tatsache begründet, dass sich das gegebene Problem sehr gut in einer funktionalen Programmiersprache umsetzen lässt.

Umsetzung

Existenzquantorkonzept

Die Existenzbedingung ist erfüllt sobald mindestens eine Lösung gefunden wird, die valide ist. Daher ist der Grundgedanke unserer Umsetzung die Ergebnisse der Tests jeder möglichen Lösung durch ein logisches oder zu verknüpfen. Ist mehr als ein Existenzquantor angegeben, wird jede mögliche Kombination der möglichen Lösungen getestet (inklusive der Fälle mit multiplen Anwendungen, da der Existenzquantor nur eine Mindestanforderung beschreibt). Definiert z.B.

ein Existenzquantor eine Verbindung von A nach B oder C und ein weiterer eine andere Verbindung von C oder A nach B, so werden die Kombinationen (AB CA), (AB CB), (AB CA CB), (AC CA), (AC CB), (AC CA CB), (AB AC CA), (AB AC CB) und (AB AC CA CB) getestet. Funktioniert mindestens eine, so ist die Existenzbedingung für alle Quantoren erfüllt.

Um diese Menge aller möglichen Kombinationen zu generieren (Funktion allexquantcombinations) wird die Potenzmenge (ausgeschlossen die leere Menge, da ja mindestens eine Möglichkeit erhalten bleiben muss, damit die Existenzbedingung erfüllt ist) des letzten Existenzquantors rekursiv mit der Potenzmenge des jeweils vorherigen Existenzquantors vollständig verknüpft (kartesisches Produkt).

Test einer einzelnen Dreiecksbeziehung

Jede einzelne Möglichkeit oder die einzige Möglichkeit, wenn kein Existenzquantor angegeben ist, wird getestet (Funktion test), indem alle drei Einzeltests (Funktion testsingle) durchgeführt werden.

Dafür wird jeweils die Funktion combine aufgerufen und die Schnittmenge gebildet, genau wie es in Allen'sche Zeitlogik auf dem Papier auch passiert.

Util

mapsingle & mapcomb

Zwei sehr häufig genutzte Funktionen, die nicht direkt mit Allen'sche Zeitlogik in Verbindung stehen, sind mapsingle & mapcomb. Mapsingle führt eine Funktion auf jedem Element einer List mit ein bis 3 fixen Parametern aus. Mapcomb baut auf mapsingle auf und führt die gegebene Funktion auf jeder Kombination zweier Listen aus (mit 2 optionalen, fixen Parametern). Beide Funktionen geben

als Resultat eine Liste mit den Rückgabewerten der einzelnen Funktionsaufrufe zurück.

keyvalue/-s/-tester & hashmap

Um z.B. die P-Matrix zu implementieren, wurde auf das Grundprinzip eines Dictionary/Hashmap/Key-Value-Store/... zurückgegriffen, indem eine Liste von 2-Tupeln der Form (key (values)) angelegt und durchsucht wird.

Aufteilung

Zur besseren Übersicht in einer Programmiersprache ohne Klassen ist die Implementierung auf mehrere separate Dateien verteilt:

- allen.lsp: Basisimplementierung Allen'sche Zeitlogik
- static.lsp: Statische Definitionen (z.B. p-Matrix)
- file.lsp: Auslesen und Auswerten von Input
- main.lsp: Zusammenführung durch load, Hauptfunktion
- quant.lsp: Erweiterung Existenzquantor
- util.lsp: Nicht direkt mit Allen'scher Zeitlogik zusammenhängende Hilfsfunktionen

Test

Für das Testen des LISP-Programms wurde eine separate Ordnerstruktur erstellt: Im Ordner test/ liegen die Unterordner success/ und fail/ erstellt. Da hoher Automatisierungsgrad beim Testen Entdeckungswahrscheinlichkeit von Fehlern drastisch erhöht und manuellen Fehlern beim Testen vorbeugt, sollte eine eigene Testsuite für die Tests erstellt werden. Hierzu wurden alle Fälle, die erfolgreich sein sollten im Unterordner success/ in einzelne Dateien abgelegt, alle Fälle die

fehlschlagen sollten im Ordner fail/. Direkt im Ordner test/ wurde ein makefile (makefile.lsp) erstellt, das sämtliche Tests per Load-Befehl einbindet.

Zum Testen wurde folgendermaßen Vorgegangen: Mangels Automatisierungs-Möglichkeit wurden als erstes alle 169 Möglichkeiten der P-Matrix nach dem Vier-Augen-Prinzip überprüft. Anschließend wurden folgende Randbedingungen aufgestellt:

- Eine Dreiecksbeziehung sollte logisch korrekt als wahr oder falsch erkannt werden
- Ein Existenzquantor, der nur eine Beziehung beinhaltet, sollte logisch korrekt je nach Eingabewerten als korrekt oder falsch erkannt werden
- Ein Existenzquantor, der als ersten Parameter eine Ecke des Dreiecks und als zweiten Parameter mehrere Ecken des Dreiecks beinhaltet, sollte nach den Regeln der Allen-Logik als korrekt oder falsch erkannt werden
- Ein Existenzquantor, der mehrere Ecken des Dreiecks im ersten und mehrere Ecken im zweiten Parameter beinhaltet, sollte nach der Allen-Logik als korrekt oder falsch erkannt werden
- Ein Existenzquantor, der mehrere Ecken des Dreiecks im ersten Parameter und nur eine Ecke im zweiten Parameter beinhaltet, sollte nach den Regeln der Allen-Logik als korrekt oder falsch erkannt werden
- Mehrere Existenzquantoren sollten nach den Regeln der Allen-Logik als korrekt oder falsch erkannt werden
- Ein Existenzquantor mit einer Beziehung zu sich selbst, wie etwa (r_exist 'A 'A '(X)), wobei X für eine beliebige Beziehung steht, sollte fehlschlagen

Daraus wurden Äquivalenzklassen gebildet und aus jeder Äquivalenzklasse ein Testfall erstellt. Wenn möglich wurden dabei mit einem Testfall mehrere Möglichkeiten abgedeckt. Diese Testfälle wurden mit dem Vier-Augen-Prinzip von Hand durchgerechnet. Anschließend wurde getestet, ob das in LISP umgesetzte Programm dieselbe Lösung liefert.

Folgende Testfälle wurden erstellt:

fail

- fail01: Ein einfache Dreiecksbeziehung, die scheitert. Ein Existenzquantor ist hier nicht enthalten.
- fail02: Eine einfache Dreiecksbeziehung, die scheitert. Zusätzlich ein Existenzquantor mit einer Ecke des Dreiecks als erstem Parameter und einer Ecke des Dreiecks als zweitem Parameter. Der Existenzquantor ändert jedoch nichts am Scheitern der Prüfung.
- fail03: Eine einfache Dreiecksbeziehung, die scheitert. Zusätzlich ein Existenzquantor, der als ersten Parameter eine einzelne Ecke des Dreiecks und als zweite Bedingung die beiden anderen Ecken des Dreiecks bekommt. Trotz der durch den Existenzquantor hinzugefügten Beziehungen, scheitert die Prüfung nach den Regeln der Allen-Logik.
- fail04: Ein komplexerer Testfall mit prinzipiell gleichem Aufbau wie fail04, aber einer komplexeren Dreiecksbeziehung, um auch das korrekte Zusammenspiel komplexerer Bedingungen zu testen.
- fail05: Eine einfache Dreiecksbeziehung, die für sich alleine scheitert und zusätzlich ein Existenzquantor von einer Ecke des Dreiecks auf sich selbst. Dieser Testfall sollte auch scheitern.

success

- success01: Ein einfacher Testfall einer Dreiecksbeziehung, der nach den Regeln der Allen-Logik korrekt ist. Ein Existenzquantor ist hier nicht enthalten.
- success02: Ein einfacher Testfall mit einer Dreiecksbeziehung, die ohne den Existenzquantor nach den Regeln der Allen-Logik scheitern würde, mit den zusätzlichen Bedingungen des Existenzquantor jedoch valide ist. Der

Existenzquantor ist ein simpler Existenzquantor mit nur einer einzelnen Beziehung zwischen nur zwei Ecken des Dreiecks.

- success03: Wie Testfall success02, aber mit einem weiteren Existenzquantor, der Beziehungen zwischen einer Ecke des Dreiecks mit den beiden anderen Ecken des Dreiecks herstellen kann und so das Dreieck nach den Regeln der Allen-Logik konsistent ist.
- success04: Eine Existenzquantoren-Bedingung von einer Ecke des Dreiecks zu zwei anderen Ecken des Dreiecks mit mehreren Beziehungen. Zusätzlich ein weiterer Existenzquantor, der von mehreren Ecken des Dreiecks Beziehungen zu mehreren Ecken des Dreiecks herstellen kann.

Durch diese Testfälle konnten alle Möglichkeiten und Randbedingungen der Äquivalenzklassen getestet werden.

Um den Erfolg oder das Scheitern aller Tests auf einen Blick zu sehen, wurde eine Ausgabe erstellt, die erfolgreich ist, wenn alle Tests erfolgreich sind und fehlschlägt, wenn einer der Tests ein anderes als das spezifizierte Ergebnis liefert.

Da sämtliche Tests automatisiert ausgeführt wurden, erscheinen die Testergebnisse nach der Ausführung der entsprechenden Datei make.lsp. Der Gesamttest in der letzten Zeile zeigt wie bereits oben beschrieben, ob irgendeiner der Tests fehlschlug oder nicht.

Alle Testszenarien wurden in unseren Tests mit dem gewünschten Testergebnis durchlaufen.