# FREIE UNIVERSITÄT BERLIN

General Game Playing mit stochastischen Spielen

Johannes Kulick, Marco Block und Raul Rojas

B-09-08
July 2009


FACHBEREICH MATHEMATIK UND INFORMATIK
SERIE B · INFORMATIK

# General Game Playing mit stochastischen Spielen

Johannes Kulick, Marco Block und Raúl Rojas
Freie Universität Berlin
Institut für Informatik
Takustr. 9, 14195 Berlin, Germany
{kulick,block,rojas}@inf.fu-berlin.de

Juli 2009

***Zusammenfassung*—In der Game Description Language lassen sich rundenbasierte Spiele mit vollständiger Information ohne stochastische Ereignisse beschreiben. Obwohl die Sprache erst 2005 entwickelt wurde, hat sie eine hohe Akzeptanz innerhalb der Wissenschaftgemeinde. Es werden Programme entwickelt, die aus den Spielbeschreibungen Strategien ableiten und die Spiele erfolgreich spielen. In dieser Arbeit wird eine Erweiterung der Sprache GDL vorgestellt, mit der nun auch Spiele mit stochastischen Elementen beschrieben und gespielt werden können.**

## I. Motivation und Einführung

Vor einigen Jahren war Schach noch die *Drosophila melanogaster* der Künstliche Intelligenz (KI). Heute sind spezialisierte Maschinen in vielen Gebieten bereits mindestens so gut oder sogar besser als professionelle, menschliche Spieler, so z.B. in Dame [5], Schach [10] oder Poker [8], so dass Schach allein als Forschungsgebiet keine große Herausforderung mehr darstellt. Eine Maschine, die zwar regelmäßig Großmeister im Schach besiegen kann, ist aber nicht automatisch in der Lage diese Fähigkeiten auf andere Spiele zu übertragen und anzuwenden, so also nicht einmal in der Lage, TicTacToe zu spielen.

Abbildung 1. Ziel der GGP ist es, analog zum spielenden Menschen eine *Machina ludens* zu entwickeln, die alle Spiele erfolgreich spielen kann (Abbildung aus [1]).

General Game Playing (GGP) ist ein modernes Forschungsgebiet im Bereich der KI. In der klassischen Spieleprogrammierung wird die Spielanalyse und damit die Codierung von Expertenwissen von Menschen durchgeführt und auf Basis dieses Wissens innerhalb einer Suche über die beste Züge entschieden. Bei GGP ist das Ziel, aus der maschinenlesbaren Regelbeschreibung ein Spiel zu „verstehen“, abstrakte Gewinnstrategien für dieses zu entwickeln und das Spiel erfolgreich zu spielen (siehe Abbildung 1).

Eine formale Sprache, mit der Spiele beschrieben werden können, ist dafür notwendig. Die Game Description Language (GDL), die 2005 an der Universität Stanford von Michael Genesereth entwickelt wurde, besitzt dabei die größte Akzeptanz innerhalb der Wissenschaftsgemeinde [2].

Es wird seit 2005 einmal im Jahr eine Weltmeisterschaft im Rahmen der AAAI-Konferenz ausgetragen. Zu den aktuell fünf besten Programmen zählen der Fluxplayer (Technische Universität Dresden) [17], Ary (University of Paris), der ClunePlayer (University of California) [4] und Maligne (University of Alberta). In den letzen beiden Jahren dominierte die Universität Reykjavík mit dem Programm CADIAPlayer [3]. In Tabelle I sind die Weltmeisterprogramme der AAAI seit 2005 aufgelistet.

| Jahr | Programm | Arbeitsgruppe |
|---|---|---|
| 2005 | ClunePlayer | University of California |
| 2006 | Fluxplayer | Technische Universität Dresden |
| 2007 | CADIAPlayer | Reykjavik University |
| 2008 | CADIAPlayer | Reykjavik University |

Tabelle I
Weltmeisterprogramme der AAAI-Competition seit 2005.

Derzeit wird eine weitere Competition im Rahmen der GIGA 2009 ausgetragen, es bleibt abzuwarten, ob auch dort der CADIAPlayer seine dominierende Stellung beibehalten wird [20].

Mit der Sprache GDL lassen sich rundenbasierte Spiele mit vollständiger Information ohne stochastische Ereignisse modellieren, in denen Agenten nicht kommunizieren können. Damit ist zwar eine große Klasse von Spielen beschreibbar, aber einfache Spiele, wie z.B. Poker, Monopoly oder Backgammon sind es nicht. Gerade im Hinblick auf die Anwendbarkeit der Spieltheorie auf andere Gebiete, wie beispielsweise die Wirtschaftswissenschaften, fehlt es der GDL an Ausdrucksfähigkeit.

Die vorliegende Arbeit stellt eine Erweiterung der GDL um das Konzept stochastischer Ereignisse vor. Die bisherige Sprachkonstrukte bleiben dabei erhalten. In Abschnitt II werden verwandte Arbeiten und Projekte vorgestellt, sowie ein

kurzer historischer Abriss zum Thema General Game Playing. Anschließend werden in Abschnitt III die Erweiterungen vorgestellt, mit denen sich in GDL auch stochastische Ereignisse modellieren lassen. Abschließend werden in Abschnitt IV die Ergebnisse diskutiert und ein Ausblick auf zukünftige Erweiterungen gegeben.

## II. Verwandte Arbeiten und Projekte

### A. General Game Playing

In den 60er Jahren gab es bereits durch Jacques Pitrat eine erste Bewegung in Richtung General Game Playing, aber sie wurde erstmal aufgrund weniger Fortschritte wissenschaftlich vernachlässigt [15].

Einen ersten richtigen Ansatz eine Sprache für Regelbeschreibungen zu entwickeln machte Barney Pell 1992 [12]. Er schlug mit *Metagame* eine Sprache vor, die eine Klasse von Spielen beschreiben kann, die er symmetrische Schach-ähnliche Spiele nennt. Dabei handelt es sich um Zwei-Spieler-Brettspiele mit vollständiger Information ohne Zufallsereignisse. Zwar können so unterschiedliche Spiele generiert werden, die Einschränkungen sind aber so groß, dass die meisten klassischen Brettspiele damit nicht abgebildet werden können.

Ein ähnlicher Ansatz wie er bei Metagame verfolgt wurde, findet sich bei *Multigame* wieder [9]. Auch hier können nur Brettspiele, allerdings mit einem oder mit zwei Spielern modelliert werden. Zufallsereignisse lassen sich damit aber ebenfalls nicht abbilden.

Das kommerzielle Produkt *Zillions of Games* stellt eine auf Lisp basierende Sprache zur Verfügung, mit der Spiele beschrieben werden können [19]. Über einen unsichtbaren Computerspieler, der zufällige Züge spielt, ist es hier möglich Zufall zu simulieren. Es ist allerdings nicht möglich eigene KIs zu schreiben, die diese Spiele dann spielen, sondern eine vorgefertigte, nicht änderbare KI wird mitgeliefert, die die Beschreibungen lesen und verarbeiten kann.

In dem 2007 erschienenen Artikel von Quenault und Cazenave wird das *Extended General Gaming Model* vorgeschlagen [6]. Damit soll es möglich sein, Zufallsereignisse zu modellieren. Es handelt sich dabei aber nur um eine Pythonbibliothek für die Spieleprogrammierung, die sowohl für die Spielbeschreibung als auch für die Agenten Schnittstellen bereitstellt. Das zugrundeliegende Framework ist nicht verfügbar und war bei Veröffentlichung des Artikels noch nicht fertig. Zufallsereignisse waren zu diesem Zeitpunkt ebenfalls noch nicht implementiert.

Die Sprache mit der größten Akzeptanz im Bereich des GGP ist die *Game Description Language* [2]. Es handelt sich um eine Datalog-Variante, die mit Hilfe von Aussagenlogik Spielregeln formal definiert. Für sie existiert ein frei verfügbares Framework [17]. In der sehr allgemein gefassten GDL können rundenbasierte, deterministische Spiele mit vollständiger Information beschrieben werden. Obwohl dies schon sehr viele Spiele umfasst, können viele klassische Spiele damit nicht modelliert werden.

Da es schon sehr viele starke Programme für die GDL gibt und sich das logische Sprachmodell als sowohl sehr mächtig als auch flexibel herausgestellt hat, ist es sinnvoll die GDL zu erweitern und nicht eine neue Sprache zu entwickeln.

Aktuell gibt es mit Palamedes ein Werkzeug (Plugin für Eclipse), das die Entwicklung von Spielen in GDL unterstützt (siehe Abbildung 2) [21].

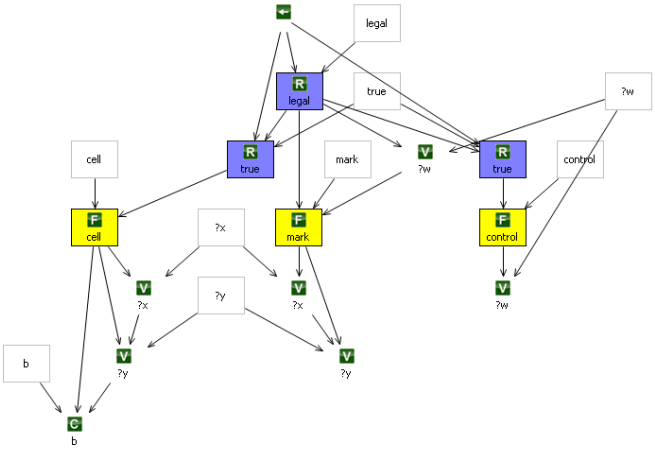


Abbildung 2. Mit Hilfe der Palamedes IDE lassen sich Relationen und die damit auftretenden logischen Abhängigkeiten in Syntaxbäumen visualisieren.

### B. Expertensysteme und Unsicherheit

Wie die GDL basieren auch viele Expertensysteme auf logischem Schließen. Expertensysteme wurden in den 1950er Jahren entwickelt und sollen über Wissen einer entsprechenden Domain verfügen, das durch einen Experten gegeben wurde. Im Anschluß soll das Expertensystem nicht nur in der Lage sein, dieses Wissen Eins-zu-Eins wiederzugeben, sondern basierend auf gegebene Daten mit der sogenannten Inferenzmaschine Wissen abzuleiten und daraus logisch zu schließen [7].

Die Schwierigkeit besteht oft darin, dass der Domainexperte (menschlicher Experte) das Wissen nicht immer mit absoluter Sicherheit angeben kann, sondern das Wissen eher so formuliert: „*Wenn der Wagen nicht anspringt und das Radio funktioniert, liegt es in 90% der Fälle an einem leeren Tank.*“. Expertensysteme müssen also auch mit stochastischen Ereignissen umgehen können.

Das in Lisp 1972 an der Stanford-Universität entwickelte medizinische Expertensystem MYCIN beispielsweise, arbeitet mit dem Konzept der Unsicherheitsfaktoren, denn gerade bei oft auftretenden Krankheitssymptomen kann die korrekte Diagnose nicht zu 100% erfolgen [14].

Wir wollen uns im folgenden Beispiel zur Fahrzeugdiagnose den Einsatz von Unsicherheitsfaktoren anschauen (Beispiel entnommen aus [13]). Dabei können Unsicherheitsfaktoren an Fakten

```
turn_over cf 100
smell_gas cf 100
```

und Regeln

```
rule 4
if turn_over and smell_gas
then problem is flooded cf 80
```

angegeben werden. Diese Faktoren liegen im Bereich von $[-100, 100]$, wobei $-100$ der Wahrscheinlichkeit $p(e) = 0$ und 100 der Wahrscheinlichkeit $p(e) = 1$ entspricht.

Die Herausforderung ist es nun, die gegebenen Unsicherheitsfaktoren in der Suche sinnvoll weiterzuleiten und zu kombinieren. Da es kein Wissen über den kompletten Weg zwischen Wurzel und Blatt im Entscheidungsbaum gibt, muss die Überführung der Faktoren lokal an jeder Stelle vorgenommen werden. Dazu wurde bei MYCIN die folgende Regel, die in der Praxis gute Resultate geliefert hat, für die Kombination von Regeln und Prämissen verwendet

```
CF = RuleCF * PremiseCF / 100.
```

Wenn mehrere Regeln aufeinander stoßen (innerer Knoten des Entscheidungsbaumes), dann werden die folgenden Regeln angewendet

```
CF(X,Y)=X+Y(100-X)/100; X und Y>0
CF(X,Y)=X+Y/(1-min(|X|,|Y|)); X oder Y<0
CF(X,Y)=-CF(-X,-Y); X und Y<0
```

Die resultierenden Unsicherheiten entsprechen dabei nicht den exakten Werten und so kann das System nur einen heuristischen Wert angeben. Besonders negative Unsicherheitsfaktoren können zu unbrauchbaren Ergebnissen führen.

In Abbildung 3 sind beispielsweise die Regeln

```
rule 1
if fact_1
then problem is conclusion cf -80
```

und

```
rule 2
if fact_2
then problem is conclusion cf 50
```

visualisiert.

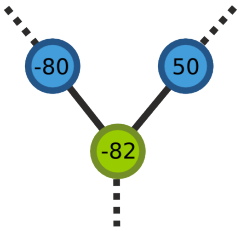


Abbildung 3. Unsicherheitsfaktoren werden nur lokal berechnet. Fehler in der Heuristik werden so weiterpropagiert und akkumulieren sich entsprechend. In diesem Beispiel wurde die Vorschrift `CF(X,Y)=X+Y/(1-min(|X|,|Y|)); X oder Y<0` verwendet.

Aus diesen Regeln ergibt sich nach der vorher genannten Vorschrift, der Unsicherheitsfaktor für die Schlussfolgerung

$$\begin{aligned} CF_{Conclusion}(50,-80) &= 50 + \frac{-80}{1-\min(|-80|,|50|)} \\ &= 51.63 \end{aligned} .$$

Diese lokal von einer Heuristik berechneten Unsicherheitsfaktoren werden nun in den folgenden Schlussfolgerungen weiter verwendet. Auftretende Fehler werden also weiterpropagiert und akkumulieren sich entsprechend.

Ein weiteres Problem ist die Assymetrie der Regeln (siehe Abbildung 4).

So führt die Umkehrung der Unsicherheiten in den Eingaben für $X$ und $Y$ mit

$$\begin{aligned} CF_{Conclusion}(-80,50) &= -80 + \frac{50}{1-\min(|-80|,|50|)} \\ &= -82.02 \\ &\neq CF_{Conclusion}(50,-80) \end{aligned}$$

zu einem anderen Ergebnis.

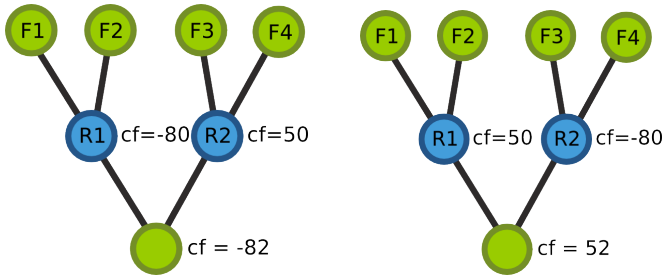


Abbildung 4. Aus dem linken Teilbaum ergibt sich der Unsicherheitsfaktor $cf = -82$, was in etwa einer Wahrscheinlichkeit $p(e) = 0.09$ entspricht. Aus dem rechten Teilbaum, in dem die Reihenfolge der oberen Knoten vertauscht wurde, ergibt sich der Unsicherheitsfaktor $cf = 52$, was einer Wahrscheinlichkeit von $p(e) = 0.76$ entspricht. Die MYCIN Regeln sind assymmetrisch und entsprechen nicht der exakten Wahrscheinlichkeit $p(e) = 0.775$.

## III. Erweiterte Syntax der Sprache GDL

### A. Grundzüge der GDL

Die Syntax der Sprache GDL basiert auf dem Knowledge Interchange Format (KIF), das zum Austausch von maschinenlesbarem Wissen konzipiert wurde [11]. Sie nutzt Aussagenlogik in Prefix-Notation zur Darstellung von Spielregeln. Es gibt drei Konstrukte: Terme, Relationen und Implikationen.

Terme sind entweder Atome, also unveränderliche Objekte, oder Variablen, die unifiziert werden können. Variablen beginnen mit einem Fragezeichen. Relationen bestehen aus einem Namen und $n$ Termen. Implikationen bestehen aus einem Kopf (der Schlussfolgerung) und der Konjunktion von $m$ Relationen (den Vorraussetzungen).

Zusätzlich wird in der GDL die Semantik der acht Relationen `role`, `init`, `true`, `does`, `next`, `legal`, `goal` und `terminal` definiert. Jede dieser Relation beschreibt typische Spieleigenschaften. Tabelle II zeigt die Funktionen der einzelnen Relationen und gibt eine ensprechende Kurzbeschreibung.

| Relation | Funktionalität |
|---|---|
| `role <player>` | gibt die Namen der Spieler an und definiert die Spieleranzahl |
| `init <state>` | beschreibt den initialen Spielzustand |
| `true <state>` | überprüft ob Terme oder Relationen im Spielzustand vorkommen |
| `does <player> <move>` | beschreibt die zuletzt gezogenen Züge |
| `next <state>` | beschreibt den Spielzustand in der nächsten Runde |
| `legal <player> <move>` | definiert legale Züge für einen Spieler zu einem bestimmten Zustand |
| `goal <player> <value>` | definiert Bewertungen für Ziele mit Werten zwischen 0 und 100 |
| `terminal` | definiert terminale Zustände in der Spielbeschreibung |

Tabelle II
Liste der reservierten Relationen in GDL mit entsprechender Kurzbeschreibung.

Ein typischer Ausschnitt aus einer Spielbeschreibung sieht zum Beispiel so aus.

```
(<= (next (cell ?x ?y ?marker))
    (does ?player (mark ?x ?y))
    (true (cell ?x ?y b))
    (playerpiece ?player ?marker))
```

Diese Regel besagt, dass die Zelle mit den Koordinaten `?x` und `?y`, die vorher mit einem `b` markiert war, durch den Zug `mark ?x ?y` von dem Spieler `?player` markiert wird. Zu beachten ist dabei, dass die Relationen `cell`, `mark` und `playerpiece` keine GDL-Schlüsselwörter sind, sondern in der Spielbeschreibung definiert werden müssen.

Um gegeneinander spielen zu können wird noch eine Infrastruktur benötigt. Das GGP-Framework der Universität Dresden bietet dafür den GameController [17]. Dieser fungiert als Vermittler zwischen den Spielern. Die Kommunikation ist dabei auch in der GDL-Sprachspezifikation definiert.

Zu Spielbeginn sendet der GameController allen Spielern eine Startnachricht mit der Rolle, die der jeweilige Spieler übernimmt, der Vorbereitungszeit, der Zugzeit und der Spielspezifikation in GDL. Zusätzlich bekommt jedes Spiel hier eine eindeutige Identifikationsnummer. Die Nachricht hat das Format

```
(START <MATCHID> <ROLE> <DESCRIPTION>
       <STARTCLOCK> <PLAYCLOCK>)
```

Nach der Vorbereitungszeit müssen alle Spieler ihren Zug zum GameController geschickt haben. Für jeden weiteren Zug haben sie nur die Zugzeit zur Verfügung. Sollten sie zu spät oder illegal ziehen, so entscheidet der GameController über einen zufälligen legalen Zug, den der Spieler stattdessen zieht. Die tatsächlich gezogenen Züge werden allen Spielern wiederum vom GameController in einer Playnachricht bekanntgegeben.

```
(PLAY <MATCHID> (<MOVE1> <MOVE2>
                ... <MOVEn>))
```

Ist ein terminaler Zustand erreicht, so werden die letzten Züge nicht in einer Playnachricht sondern in einer Stopnachricht bekannt gegeben. Daraus können die Spieler den terminalen Zustand und damit den Sieger berechnen.

Die genaue Definition der Sprache GDL und der Kommunikation zwischen Spielern und GameController kann der Sprachspezifikation entnommen werden [2].

### *B. Erweiterung um Zufallsereignisse*

Um Zufallsereignisse und Unsicherheit nutzen zu können, führen wir die neue Relation `random` ein. Sie hat die folgende Syntax

```
random <name> <value> <expression>
```

Der erste Parameter ist ein Term, der die Zufallsvariable bezeichnet. Alle $n$ Ereignisse die von dem selben Zufallsexperiment abhängen, werden der selben Zufallsvariable zugeordnet, bekommen also den selben Namen. Der zweite Parameter ist eine ganze Zahl $v_e$ und definiert die Wahrscheinlichkeit für das Ereignis $e$. Der dritte Parameter ist ein GDL-Ausdruck, der das Ereignis $e$ beschreibt, das in Abhängigkeit eines Zufallsexperiments auftreten kann.

Die Wahrscheinlichkeit für ein Ereignis $e$ berechnet sich aus

$$p(e) = \frac{v_e}{\sum_{i=0}^{n} v_i}.$$

Die Unifizierung von Zufallsvariablen erfolgt (pseudo-) zufällig. Damit dies für alle Spieler gleich geschieht und sicher gestellt ist, dass tatsächlich Pseudo-Zufallsgeneratoren eingesetzt werden, muss die Unifizierung an einer zentralen Stelle vorgenommen werden und nicht von den Spielern selbst. Aus diesem Grund erhalten die Zufallsvariablen keinen KIF-konformen Variablennamen mit voranstehendem Fragezeichen, sondern sind Atome.

Da der einzige Austausch von Informationen in einem GDL-basierten Spiel bei der Übertragung der Züge stattfindet, stellen wir die folgende Restriktion auf. Eine Zufallsvariable darf neben ihrer Definition in einer `random` Relation nur in der Definition von legalen Zügen, also im Zug-Teil von `legal` Klauseln, vorkommen (Zufalls-Restriktion).

Ein einfaches Beispiel für ein Zufallsereignis ist der Würfelwurf in einem Brettspiel.

```
(random dice 1 1)
(random dice 1 2)
(random dice 1 3)
(random dice 1 4)
(random dice 1 5)
(random dice 1 6)
(<= (legal ?player (move dice))
    (control ?player)
)
```

Der Spieler hat hier den legalen Zug `move dice`. Die Zufallsvariable `dice` wird dabei erst an zentraler Stelle unifiziert, wenn er sich für diesen Zug entscheidet. Dabei wird einer der Ausdrücke $e \in M = \{1,2,...,6\}$ mit jeweils der Wahrscheinlichkeit $p(e) = \frac{1}{6}$ ausgewählt.

Die `random` Relation kann dabei nicht nur in konstanten Ausdrücken vorkommen, sondern auch in Implikationen. So können Wahrscheinlichkeiten vom bisherigen Spielverlauf abhängig sein.

Beispielsweise ist die Wahrscheinlichkeit eine bereits gezogene Karte nochmals zu ziehen in den meisten Kartenspielen $p(e) = 0$.

```
(<= (random drawCard 0 ?card)
    (card ?card)
    (true (drawn ?card))
)
(<= (random drawCard 1 ?card)
    (card ?card)
    (not (true (drawn ?card)))
)
(card spades-ace)
(card spades-king)
...
```

Die zentrale Stelle bei GDL-Spielen ist der GameController, der den Spielern die Regeln sendet, die Zeiten kontrolliert und die Züge auf Legalität überprüft. Dieser wurde erweitert, so dass er die Unifizierung der Zufallsvariablen übernimmt (siehe Abbildung 5).

Die Zufallsvariable wird dabei in der Mitteilung der gezogenen Züge durch das Ereignis ersetzt. So erhalten alle Spieler das Ergebnis des Zufallsexperiments.

Das Beispiel am Ende des Abschnittes II-B läßt sich nun mit Hilfe der Erweiterung mathematisch korrekt nachvollziehen (siehe Abbildung 6).

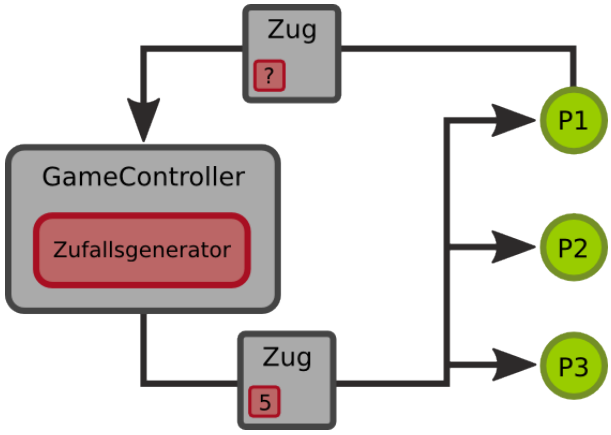


Abbildung 5. Die Architektur der Zufallsereignisse: Der GameController unifiziert die Zufallsvariablen in einem Zug und sendet den Zug mit dem aufgetretenen Ereignis an alle Spieler. So werden alle Spieler über den Ausgang des Zufallsereignisses informiert und können die Folgezustände berechnen. So wird auch sichergestellt, dass tatsächlich Pseudo-Zufallsereignisse eingesetzt werden.

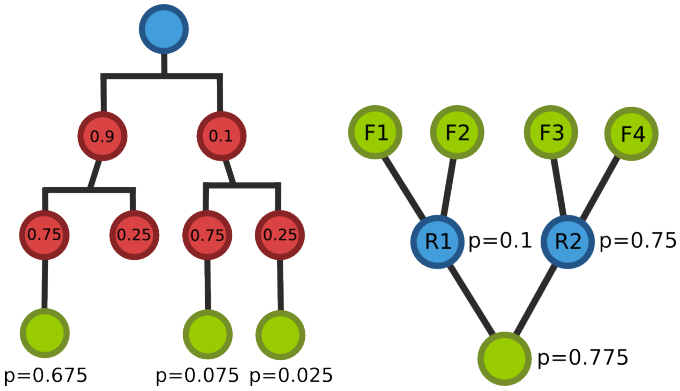


Abbildung 6. Unsicherheiten an Regeln oder Fakten können durch Zufallsereignisse ausgedrückt werden. Die beiden Regeln $R_1$ und $R_2$ sind stochastisch unabhängig. Daraus folgt, dass die Wahrscheinlichkeit der Schlussfolgerung $p(e) = 0.1 + 0.9 \cdot 0.75 = 0.775$ ist. Das ist im linken Wahrscheinlichkeitsbaum dargestellt. Die Summe der Wahrscheinlichkeiten in diesem Beispiel beträgt $\sum p(e) = 0.775$. Jeder Wahrscheinlichkeitsknoten steht hier für ein Zufallsereignis in GDL. Die sich ergebenen Wahrscheinlichkeiten sind demnach korrekt.

## IV. Diskussion und Ausblick auf zukünftige Arbeiten

Die Erweiterung der Sprache GDL um das Konzept der Unsicherheit, gibt jetzt die Möglichkeit, auch stochastische Spiele zu beschreiben. Im Rahmen dieser Erweiterung wurden bereits einige Spiele formalisiert, diese stehen auf der Projektwebseite zum Download bereit [16]. Zusätzlich wurde der Basicplayer der Technischen Universität Dresden erweitert und steht ebenfalls auf der Projektseite zur Verfügung.

Neben Forschung und Entwicklung an Programmen, die in der Lage sind, aus maschinenlesbaren Regelbeschreibungen, abstrakte Gewinnstrategien abzuleiten und diese Spiele erfolgreich zu spielen, muss auch die formale Sprache ständig erweitert werden, um eine größere Problemklasse abzudecken.

Zusätzlich zu unvollständigen Informationen und der Möglichkeit ein Echtzeitsystem zu beschreiben, muss auch Kommunikation zwischen Agenten ermöglicht werden. Nur dann können Kooperationen entstehen, die widerum eine Basis für größere, komplexere Problemstellungen sind.

## Literatur


[1] Michulke D., Schiffel S.: „*Matt bei Vier gewinnt*“, C't Computer und Technik Magazin, Vol.1, pp.174, 2009

[2] Love N., Hinrichs T., Haley D., Schkufza E., Genesereth M.: „*General Game Playing: Game Description Language Specication*“, Stanford Logic Group Computer Science Department Stanford University, Technical Report LG-2006-01, 2008

[3] Finnsson H., Björnsson Y.: „*Simulation-Based Approach to General Game Playing*“, In The Twenty-Third AAAI Conference on Artificial Intelligence, pp. 259-264, 2008

[4] Clune J.: „*Heuristic evaluation functions for General Game Playing*“, In Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence, pp. 1134-1139, 2007

[5] Schaeffer J., Burch N., Bjornsson Y., Kishimoto A., Muller M., Lake R., Lu P., Sutphen S.: „*Checkers is Solved*“, Magazin Science, Vol.317, No.5844, pp. 1518-1522, 2007

[6] Quenault M., Cazenave T.: „*Extended General Gaming Model*“, Computer Games Workshop, pp. 195-204, 2007

[7] Kurbel K.: „*Entwicklung und Einsatz von Expertensystemen: Eine anwendungsorientierte Einführung in wissensbasierte Systeme*“, ISBN: 978-3540552376, Springer Verlag, 2007

[8] Billings D.: „*Algorithms and Assessment in Computer Poker*“, Dissertation an der University of Alberta/Kanada, 2006

[9] Romein J.: „*Multigame - An environment for Distributed Game-Tree Search.*“, Dissertation an der Vrije Universiteit, Amsterdam/Niederlande, 2001

[10] King D.: „*Kasparow gegen Deep Blue. Die Auseinandersetzung Mensch gegen Maschine*“, ISBN: 978-3888052828, Beyer-Verlag, 1997

[11] Genesereth M., Fikes R., et al.: „*Knowledge Interchange Format - Version 3.0 Reference Manual*“, Stanford Logic Group Computer Science Department Stanford University, Technical Report Logic-92-1, 1992

[12] Pell B.: „*Metagame in Symmetric, Chess-Like Games.*“, In van den Herik H. und Allis L. eds.: Heuristic Programming in Artificial Intelligence 3 - The Third Computer Olympiad, Ellis Horwood, 1992

[13] Merritt D.: „*Building Expert Systems in Prolog*“, ISBN: 978-3540970163, Springer Verlag, 1989

[14] Buchanan B.G.: „*Rule Based Expert Systems: The Mycin Experiments of the Stanford Heuristic Programming Project*“, ISBN: 978-0201101720, Addison-Wesley, 1984

[15] Pitrat J.: „*Realization of a general game-playing program*“, 4th IFIP Congress, Vol.2, pp.1570-1574, 1968

[16] Webseite der GGP-Gruppe der FU Berlin: *http://gameai.mi.fu-berlin.de/ggp/index.html*

[17] Webseite der GGP-Gruppe der TU Dresden: *http://www.general-game-playing.de/*

[18] Webseite der GGP-Gruppe der Universität Reykjavík: *http://cadia.ru.is/wiki/public:cadiaplayer:main*

[19] Webseite zu Zillions of games: *http://www.zillions-of-games. com/index.html*

[20] Webseite der GIGA 2009: *http://www.ru.is/GIGA09/ggpc2009.html*

[21] Webseite zur Palamedes IDE: *http://palamedes-ide.sourceforge.net/*


Die angegebenen Links waren gültig am 03.07.2009.