Das Lambda-Kalkül und die Montague-Semantic in Spielen

Lydia Steiner

Seminararbeit
Universität Leipzig
Institut für Informatik
E-Mail: mam04inr@studserv.uni-leipzig.de

1 Einleitung

Die Facharbeit ist im Rahmen des Games Summer Camp 2006 der Universität Leipzig entstanden. Sie befasst sich mit Dialogsystemen für Spiele, insbesondere mit der möglichen Anwendung des Lambda-Kalküls und der Montague-Semantic für Spiele. Dazu wird zunächst das Lambda-Kalkül und danach die auf dem Lambda-Kalkül aufbauende Montague-Semantic kurz erklärt. Außerdem wird ein Spiel vorgestellt, indem das Lambda-Kalkül bereits angewandt wurde.

Im letzten Teil werden die Ideen der Autorin über die sinnvolle Anwendung des Lambda-Kalküls und der Montague-Semantic in Spielen diskutiert. Es wird darauf eingegangen, wie das Lambda-Kalkül beziehungsweise die Montague-Semantic die Steuerung von Spielen erleichtern könnte, aber auch wie die Oberfläche des Spiels sinnvoll zu gestalten ist, um eine Verbesserung des Spielerlebnis zu erhalten. Bei der Anwendung des Montague-Semantic in Spielen wird die Autorin nicht nur auf den Bereich Kommunikation mit dem Spieler eingehen, sondern auch eine mögliche Anwendung für die Physik-Engine zeigen.

2 Das Lambda-Kalkül

2.1 Definition des Lambda-Kalküls

Ein Ausdruck des Lambda-Kalküls nennt man Lambda-Ausdruck (λ-Ausdruck). Sie entsprechen folgenden drei Syntaxregeln.

1. Syntaxregel: Variablennamen

<λ-Ausdruck>= <Variable>

Ein Variable ist ein λ-Ausdruck.[1]

[1] http://www.betoerend.de/dasLandHinterDemEndeDesSinns/lambda/folien15bis20.html , 01.09.2006

2. Syntaxregel: Applikation

<λ-Ausdruck>=<λ-Ausdruck>< λ-Ausdruck>

zwei hintereinander geschriebene λ-Ausdrücke ergeben wieder einen λ-Ausdruck.[2]

3. Syntaxregel: Abstraktion

<λ-Ausdruck>= λ<Variable>.< λ-Ausdruck>

Eine λ-Funktion ist ein λ-Ausdruck. λ-Funktionen haben folgende Gestalt: λ gefolgt von einer Variable und einem Punkt und dahinter wiederum ein λ-Ausdruck. Die Variable kann selbst im Lambda-Ausdruck vorkommen, muss aber nicht. Variablen die im λ-Ausdruck vorkommen, aber nicht durch ein außen stehendes λ gebunden sind, nennt man frei.[3]

Für das Lambda-Kalkül gelten folgende zwei Ableitungsregeln:

1. α-Konversion: Variablennamen sind austauschbar.

$$(\lambda x.A) \leftrightarrow (\lambda y.A')$$

λx.A und λy.A‘ sind die selbe λ-Funktion, wenn der Ausdruck A‘ der Ausdruck A ist, indem in A alle Vorkommen x durch y ersetzt wurden und in A kein y vorkommt.

2. β- Konversion: Anwendung einer Funktion auf einen Ausdruck

$$(\lambda x.A)(B) \leftrightarrow A'$$

Die Anwendung der λ-Funktion λx.A auf den Ausdruck B ergibt A‘. Der Ausdruck A‘ ist der Ausdruck A, indem alle Vorkommen von x durch B ersetzt wurden.[4]

2.2 Anwendung des Lambda-Kalküls auf Sprache

Es ist möglich, Sätze natürlicher Sprache beziehungsweise ihre Struktur durch λ-Ausdrücke zu repräsentieren. Durch Ableiten lassen sich die λ-Ausdrücke in eine logischen Formel von Funktionen übertragen, mittels welcher der Wahrheitsgehalt des Satzes geprüft werden kann.

Beispiel:

Satz: Every system crashes.

Repräsentiert wird der Satz durch den λ-Ausdruck:

$$\lambda P.\lambda Q.\forall x: P(x) \rightarrow Q(x)$$

---

[2] http://www.betoerend.de/dasLandHinterDemEndeDesSinns/lambda/folien15bis20.html , 01.09.2006
[3] Ebd.
[4] Ebd.

Ableitung des λ-Ausdrucks:

Every:

$$\lambda P.\, \lambda Q.\, \forall x : P(x) \rightarrow Q(x)$$

Every system:

$$\lambda P.\, \lambda Q.\, \forall x : P(x) \rightarrow Q(x)(system') \wedge \lambda Q.\, \forall x : system'(x) \rightarrow Q(x)$$

Every system crashes:

$$\lambda Q.\, \forall x : system'(x) \rightarrow Q(x)(crash') \leftrightarrow \forall x : system'(x) \rightarrow crash'(x)$$

$\forall x : system'(x) \rightarrow crash'(x)$ ist die logische Formel, die den Satz „Every system crashes.“ repräsentiert. Sie enthält die Funktionen system‘ und crash‘, welche beide auf der selben Objektmenge operieren und als Ergebnisse Wahrheitswerte zurück liefern. Die Funktion system‘ liefert ‚true‘, wenn das Objekt x ein System ist. Ist x kein System wird ‚false‘ zurück gegeben. ‚true‘ liefert die Funktion crash‘ zurück, wenn x crashed. ‚false‘ wenn x nicht crashed.

Somit ist $\forall x : system'(x) \rightarrow crash'(x)$ genau dann wahr, wenn entweder x System ist und crashed (oder zumindestens die Funktionen system‘ und crash‘ ‚true‘ für das Objekt x zurück liefern) oder x kein System ist (beziehungsweise die Funktion system‘ ‚false‘ zurück liefert).

Aus dem λ-Ausdruck $\lambda P.\, \lambda Q.\, \forall x : P(x) \rightarrow Q(x)$ könnte man auch folgendes ableiten:

Zweite Ableitung des λ-Ausdrucks:

Jedes:

$$\lambda P.\, \lambda Q.\, \forall x : P(x) \rightarrow Q(x)$$

Jedes Auto:

$$\lambda P.\, \lambda Q.\, \forall x : P(x) \rightarrow Q(x)(auto') \leftrightarrow \lambda Q.\, \forall x : auto'(x) \rightarrow Q(x)$$

Jedes Auto fährt:

$$\lambda Q.\, \forall x : auto'(x) \rightarrow Q(x)(fahren') \leftrightarrow \forall x : auto'(x) \rightarrow fahren'(x)$$

$\forall x: auto'(x) \rightarrow fahren'(x)$ ist die logische Formel für „Jedes Auto fährt“.

Es wurde aus dem selben λ-Ausdruck abgeleitet wie $\forall x: system'(x) \rightarrow crash'(x)$.

Jedoch wurden diesmal bei der Ableitung andere Funktionen für P und Q eingesetzt. Statt der Funktion system‘ wird die Funktion auto‘ verwendet und statt der Funktion crashed‘ wird die Funktion fahren‘ verwendet.

2.3 Beispiel: Das Spiel „Monkey Island“

Die Steuerung des Spiels „Monkey Island“ erfolgt mittels Lambda-Kalkül. Der Spieler hat verschiedene Verben und Verb-Phrasen zur Verfügung, welche er mit verschiedenen Objekten kombinieren kann. Die Verben und Verbphrasen kann man dabei als Lambda-Funktionen auffassen, deren Parameter die Objekte sind. Die Objekte des Spiels haben verschiedene Attribute, zum Beispiel: „in Szene“ oder „in Liste“, „tragbar“ oder „nicht tragbar“, „Person“ oder „Gegenstand“. Je nachdem welche Objekte an eine Funktion übergeben werden, werden verschiedene Aktionen im Spiel durchgeführt. Dabei ergibt nicht jede Kombination von Funktion und Objekt oder Objekten einen Sinn für das Spiel. Zum Beispiel besteht kein Sinn darin, mit einer Tür zu reden oder zu einem Gegenstand zu gehen, den der Spieler gerade selbst trägt.

Einige Beispiele:

1. Beispiel: Funktion „Gehe zu“

Die Funktion „Gehe zu“ wird hier mit dem Objekt „aufgeknüpfte Leiche“ verwendet. Das Objekt „aufgeknüpfte Leiche“ hat das Attribut „in Szene“. So macht diese Kombination für das Spiel einen Sinn. Als Aktion wird das Hinlaufen zur Leiche ausgeführt.

Abb. 1: Szene aus dem Spiel „Mokey Island 1“

2. Beispiel: Funktion „Schau an“

Die Funktion „Schau an“ wird hier mit dem Objekt „Bananen“ verwendet. Das Objekt „Bananen“ hat das Attribut „in Szene“. Diese Kombination ergibt im Spiel eine Sinn. Als Aktion wird ein Satz über das Objekt „Bananen“ angezeigt werden.

Abb. 2: Szene aus dem Spiel „Monkey Island 1“

3. Beispiel: Funktion „Schau an“

Die Funktion „Schau an“ wird hier mit den Objekt „Huhn“ verwendet, welches das Attribut „in Liste“ hat. Auch das ergibt für das Spiel einen Sinn. Als Aktion wird ein Satz über das Objekt „Huhn“ angezeigt werden.

Abb. 3: Szene aus dem Spiel „Monkey Island 1“

4. Beispiel: Änderung von Attributen

Während des Spieles können sich Attribute von Objekten ändern. So wird bei Objekten, die das Attribut „in Szene“ haben und auf die die Funktion „Nimm“ angewandt wird, das Attribut „in Szene“ zu „in Liste“ geändert. Objekte, die das Attribut „in Liste haben“ und auf die die Funktion „Gib“ angewendet wird, erhalten danach das Attribut „in Szene“ und haben nicht mehr das Attribut „in Liste“.

Abb. 4: Szene aus dem Spiel „Monkey Island 1“

5. Beispiel: verfügbare Funktionen

Genau wie Attribute können sich auch die verfügbaren Funktionen während des Spiels ändern. Nachdem man gestorben ist, ergibt es keinen Sinn mit jemanden zu reden oder irgendwohin zu gehen. Deshalb werden solche Funktionen nicht mehr angezeigt. Stattdessen werden Funktionen wie „Schaukle“, „Schwebe“ oder „Verwese“ zur Verfügung gestellt.

3 Die Montague- Semantic

3.1 Was ist Montague-Semantic?

Richard Montague vertrat die Meinung, dass es keinen prinzipiellen Unterschied zwischen der Semantik von natürlichen und künstlichen Sprachen gibt. Er versuchte, die logische Struktur natürlicher Sprachen offen zu legen. Dazu verwendet er Kategorien, in die er die Wörter der englischen Sprache einteilt. Außerdem gibt es noch Regeln, mit deren Hilfe man Sätze und Ausdrücke in logische Formeln übersetzt. Diese Regeln sind unterteilt in Syntax- und Übersetzungsregeln. Die Syntaxregeln sind Regeln, die den gültigen Aufbau von Sätzen und Ausdrücken beschreiben. Sie werden mit Sx abgekürzt, wobei x die Nummer der Regel ist. Die Übersetzungsregeln sind Regeln für die Übersetzung in logische Formeln. Sie werden

mit Üx abgekürzt, wobei x die Nummer der Regel ist. Wendet man für den Satzaufbau die Regel Sx an so muss man für die Übersetzung die Regel Üx anwenden.

3.2 Kategorien und Regeln

3.2.1 Kategorien

Kategorie	**Beispiele**	**Bemerkung**
e	-	-
t	-	Satz
t//e (Abk.: N)	man, woman, park	Nominalphrasen
t/e (Abk.: IV)	run, walk, talk, change	Verbalphrasen
t/IV (Abk.: T)	John, Mary, he0	Termphrasen
IV/IV (Abk.: IAV)	rapidly, slowly	Verbalphrasen Adverb
IV/T (Abk.: TV)	find, lose	Transitive Verben
T/N (Abk.: DET)	every, the, a, an	Determiner
t/t	necessarily	Sententielle Adverben
IV/t	believe, assert	Satzkomplement Verben
IV//IV	try, wish	Infinitiykomplement
IAV/T	in, about	Präpositionen

Quelle: Indroduction to Montague Semantic[5]

Zu jeder Kategorie a gibt es noch die Menge der Denotationen P_a. Oft wird die Menge auch „Menge der wohlgeformten Phrasen“ genannt. P_a enthält also alle Phrasen, die sowohl syntaktisch richtig sind als auch einen Sinn haben.

Außerdem gibt es ein Lexikon, in dem jedem Wort seine Kategorie zugeordnet wird (so ähnlich wie in der vorangegangenen Tabelle) .

3.2.2 Regeln

Richard Montague entwickelte noch einige Regeln für Syntax und Übersetzung der Sätze. So kann er jeden Satz in eine logische Formel übersetzen. Es gehören immer eine Syntaxregel und eine Übersetzungsregel zusammen.

Synaxaregel 1: Falls α im Lexikon als A kategorisiert ist, dann gilt $\in \alpha \ P_A$.

Übersetzungsregel 1: Falls α lexikalisch ist, dann gilt (α)‘=(α)‘0. Wobei (...)‘ die Übersetzungsfunktion ist und (.)‘0 der Teil davon, der lexikalische Teile übersetzt.

Syntaxregel 2: Ist $\alpha \in P_{t/IV/N}$ und $\beta \in P_N$, dann ist $F_2(\alpha,\beta) \in P_{t/IV}$, mit $F_2(\alpha,\beta)=\alpha\beta$.

[5] „Indroduction to Montague Semantic“, David R. Dowty, Robert E. Wall, Stanley Peters, 7. Kapitel „The Grammer of PTQ“, Seite 183

Übersetzungsregel 2: Ist α‘ die Übersetzung von α und β‘ die Übersetzung von β, dann ist α‘(^β‘) die Übersetzung von $F_2(\alpha,\beta)$.

Syntaxregel 3: Ist $\alpha \in P_N$ und $\beta \in P_t$, dann ist $F_{3,n}(\alpha,\beta) \in P_N$, mit $F_{3,n}(\alpha,\beta)=\alpha$ WH β‘‘, wobei WH=which, falls α Genus Neutrum ist, und wo sonst, und β‘‘= β mit erstem Vorkommen von he_i/him_i getilgt wird und weitere durch he/him, she/her oder it, je nach Genus von α.

Übersetzungsregel 3: Ist α‘ die Übersetzung von α und β‘ die Übersetzung von β, dann ist $\lambda x_n.[\alpha‘(x_n) \wedge \beta‘]$ die Übersetzung von $F_{3,n}(\alpha,\beta)$.

Syntaxregel 4: Ist $\alpha \in P_{t/IV}$ und $\beta \in P_{IV}$, dann ist $F_4(\alpha,\beta) \in P_N$, mit $F_4(\alpha,\beta)=\alpha\beta‘‘$, wobei β‘‘ aus β kommt, indem man das erste Verb au durch seine Variante in 3. Person Singular Präsens ersetzt.

Übersetzungsregel 4: Ist α‘ die Übersetzung von α und β‘ die Übersetzung von β, dann ist α‘(^β‘) die Übersetzung von $F_4(\alpha,\beta)$.

Syntaxregel 5: Wenn $\alpha \in P_{TV}$ und $\beta \in P_T$, dann ist $F_5(\alpha,\beta) \in P_{IV}$, wobei $F_5(\alpha,\beta)=\alpha\beta$ wenn β nicht die Form he_n and $F_5(\alpha, hen)=\alpha him_n$ ist.

Übersetzungsregel 5: Ist α‘ die Übersetzung von α und β‘ die Übersetzung von β, dann ist α‘(^β‘) die Übersetzung von $F_5(\alpha,\beta)$.

Syntaxregel 6: Wenn $\alpha \in P_{IAV/T}$ und $\beta \in P_T$, dann ist $F_5(\alpha,\beta) \in P_{IAV}$.

Übersetzungsregel 6: Ist α‘ die Übersetzung von α und β‘ die Übersetzung von β, dann ist α‘(^β‘) die Übersetzung von $F_5(\alpha,\beta)$.

Syntaxregel 7: Ist $\alpha \in P_{IV/t}$ und $\beta \in P_t$, dann ist $F_{11}(\alpha,\beta) \in P_{IV}$ mit $F_{11}(\alpha,\beta)=\alpha$ that β.

Übersetzungsregel 7: Ist α‘ die Übersetzung von α und β‘ die Übersetzung von β, dann ist α‘(^β‘) die Übersetzung von $F_{11}(\alpha,\beta)$.

Syntaxregel 8: Ist $\alpha \in P_{IV//IV}$ und $\beta \in P_{IV}$, dann ist $F_{17}(\alpha,\beta) \in P_{IV}$ mit $F_{17}(\alpha,\beta)=\alpha$ to β.

Übersetzungsregel 8: Ist α‘ die Übersetzung von α und β‘ die Übersetzung von β, dann ist α‘(^β‘) die Übersetzung von $F_{17}(\alpha,\beta)$.

Syntaxregel 9: Ist $\alpha \in P_{t/t}$ und ist $\beta \in P_t$, dann ist $F_6(\alpha,\beta) \in P_t$ mit $F_6(\alpha,\beta)=\alpha\beta$.

Übersetzungsregel 9: Ist α‘ die Übersetzung von α und β‘ die Übersetzung von β, dann ist α‘(^β‘) die Übersetzung von $F_6(\alpha,\beta)$.

Syntaxregel 10: Ist $\alpha \in P_{IV/IV}$ und $\beta \in P_{IV}$, dann ist $F_7(\alpha,\beta) \in P_{IV}$ mit $F_7(\alpha,\beta) = \beta\alpha$.

Übersetzungsregel 10: Ist α‘ die Übersetzung von α und β‘ die Übersetzung von β, dann ist α‘(^β‘) die Übersetzung von $F_7(\alpha,\beta)$.

Syntaxregel 11a: Ist α, $\beta \in P_t$, dann ist $F_8(\alpha,\beta) \in P_t$ mit $F_8(\alpha,\beta) =$ α and β.

Übersetzungsregel 11a: Ist α‘ die Übersetzung von α und β‘ die Übersetzung von β, dann ist [α‘ ∧ β‘] die Übersetzung von $F_8(\alpha,\beta)$.

Syntaxregel 11b: Ist α, $\beta \in P_t$, dann ist $F_9(\alpha,\beta) \in P_t$ mit $F_9(\alpha,\beta) =$ α or β.

Übersetzungsregel 11b: Ist α‘ die Übersetzung von α und β‘ die Übersetzung von β, dann ist [α‘ ∨ β‘] die Übersetzung von $F_9(\alpha,\beta)$.

Syntaxregel 12a: Ist α, $\beta \in P_{IV}$, dann ist $F_8(\alpha,\beta) \in P_{IV}$.

Übersetzungsregel 12a: Ist α‘ die Übersetzung von α und β‘ die Übersetzung von β, dann ist λx.[α‘(x) ∧ β‘(x)] die Übersetzung von $F_8(\alpha,\beta)$.

Syntaxregel 12b: Ist α, $\beta \in P_{IV}$, dann ist $F_9(\alpha,\beta) \in P_{IV}$.

Übersetzungsregel 12b: Ist α‘ die Übersetzung von α und β‘ die Übersetzung von β, dann ist λx.[α‘(x) ∨ β‘(x)] die Übersetzung von $F_9(\alpha,\beta)$.

Syntaxregel 13: Ist α, $\beta \in P_T$, dann ist $F_9(\alpha,\beta) \in P_T$.

Übersetzungsregel 13: Ist α‘ die Übersetzung von α und β‘ die Übersetzung von β, dann ist λP.[α‘(P) ∨ β‘(P)] die Übersetzung von $F_9(\alpha,\beta)$.

Syntaxregel 14: Ist $\alpha \in P_{t/IV}$ und $\beta \in P_t$, dann ist $F_{10,n}(\alpha,\beta) \in P_t$ mit $F_{10,n}(\alpha,\beta) = \beta$ mit erstem Vorkommen von he_i/him_i durch α ersetzt wird und weitere durch he/him, she/her oder it, je nach Genus von α.

Übersetzungsregel 14: Ist α‘ die Übersetzung von α und β‘ die Übersetzung von β, dann ist α‘(^λx_n.[β‘]) die Übersetzung von $F_{10,n}(\alpha,\beta)$.

Syntaxregel 15: Ist $\alpha \in P_T$ und ist $\beta \in P_{CN}$, dann ist $F_{10,n}(\alpha,\beta) \in P_{CN}$.

Übersetzungsregel 15: Ist α‘ die Übersetzung von α und β‘ die Übersetzung von β, dann ist λy.a´(^λx_n.[β´(y)]) die Übersetzung von $F_{10,n}(\alpha,\beta)$.

Syntaxregel 16: Ist $\alpha \in P_T$ und ist $\beta \in P_{IV}$, dann ist $F_{10,n}(\alpha,\beta) \in P_{IV}$.

Übersetzungsregel 16: Ist α‘ die Übersetzung von α und β‘ die Übersetzung von β, dann ist $\lambda y.a'(\hat{}\lambda x_n.[\beta'(y)])$ die Übersetzung von $F_{10,n}(\alpha,\beta)$.

Syntaxregel 17: Ist $\alpha \in P_T$ und ist $\beta \in P_{IV}$, dann ist $F_{12}(\alpha,\beta)$, $F_{13}(\alpha,\beta)$, $F_{14}(\alpha,\beta)$, $F_{15}(\alpha,\beta)$, $F_{16}(\alpha,\beta) \in P_t$ mit: $F_{12}(\alpha,\beta)$=αβ‘ und β‘ ist β indem das erste Verb durch seine negierte 3. Person Singular Präsens ersetzt wird, $F_{13}(\alpha,\beta)$=αβ‘‘ und β‘‘ ist β indem das erste Verb durch seine 3. Person Singular Futur ersetzt wird, $F_{14}(\alpha,\beta)$=αβ‘‘‘ und β‘‘‘ ist β indem das erste Verb durch seine negierte 3. Person Singular Futur ersetzt wird, $F_{15}(\alpha,\beta)$=αβ‘‘‘‘ und β‘‘‘‘ ist β indem das erste Verb durch seine 3. Person Singular Präsens Perfekt ersetzt wird, $F_{16}(\alpha,\beta)$=αβ‘ und β‘ ist β indem das erste Verb durch seine negierte 3. Person Singular Perfekt ersetzt wird.

Übersetzungsregel 17: Ist α‘ die Übersetzung von α und β‘ die Übersetzung von β, dann ist:

¬α‘(^β‘) die Übersetzung von $F_{12}(\alpha,\beta)$,

Fα‘(^β‘) die Übersetzung von $F_{13}(\alpha,\beta)$,

¬Fα‘(^β‘) die Übersetzung von $F_{14}(\alpha,\beta)$,

Pα‘(^β‘) die Übersetzung von $F_{15}(\alpha,\beta)$,

¬Pα‘(^β‘) die Übersetzung von $F_{16}(\alpha,\beta)$.

Übersetzung von Eigennamen:

$(John)'_0 = \lambda P.[\check{} P(j)]$ [6]

Übersetzung von Pronomen:

$(he_i)'_0 = \lambda P.[\check{} P(x_i)]$

Übersetzung von Determiner:

$(every)'_0 = \lambda Q.[\lambda P.[ALL(\check{}Q)(\check{}P)]]$

$(a(n))'_0 = \lambda Q.[\lambda P.[EXISTS(\check{}Q)(\check{}P)]]$

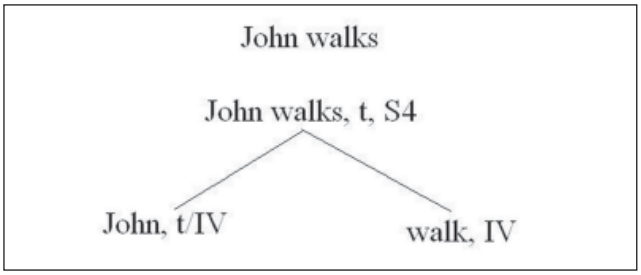
$(the)'_0 = \lambda Q.[\lambda P.[THE(\check{}Q)(\check{}P)]]$ [7]

[6] „Indroduction to Montague Semantic“, David R. Dowty, Robert E. Wall, Stanley Peters, 7. Kapitel „The Grammer of PTQ“, Seite 193

[7] Ebd., Seite 192-251

3.3 Beispiele für die Montague-Semantik

1. Beispiel: John walks.



Der Satz „John walks“ (Kategorie: t) wird durch die Syntaxregel 4 aus „John“ (Kategorie: John) und aus „walk“ (Kategorie: IV) gebildet. Auf „John“ und „walks“ kann man Syntaxregel 1 anwenden, da sie im Lexikon stehen.

Für die Übersetzung des Satzes „John walks“ müssen zuerst die lexikalischen Teile übersetzt werden (Übersetzungsregel 1), also „John“ und „walks“.

1) (John)‘0=λ P.[ˇ P(j)]

2) (walks)‘0=walk‘

Da der Satz „John walks“ mittels Syntaxregel 4 aus „John“ und „walks“ gebildet wurde, benutzt man Übersetzungsregel 4 für die Übersetzung von „John walks“.

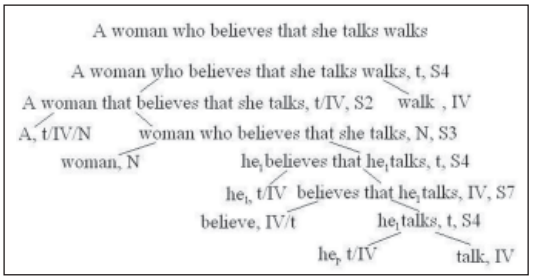
3) (John walks)‘=λ P.[ˇ P(j)] (^walk‘)

4) ˇ^walk‘(j) *(β-Konversion)*

5) walk‘(j) *(Down-Up-Konversion)*[8]

Down-Up-Konversion/Up-Down-Konversion: Wenn ein Up(^) gleich nach einem Down(ˇ) oder ein Down gleich nach einem Up kommt, können diese gleich weggelassen werden.

2. Beispiel: A woman who believes that she talks walks.



[8] http://www.ifi.unizh.ch/CL/hess/classes/ecl2/semantik.pdf#search=%22Montague%20Semantik%20PTQ%22 vom 01.09.2006, Seite 115

Übersetzung:

1. (he$_1$ talks)‘=λP.[ˇP(x$_1$)](^talk‘)= ˇ^ talk‘(x$_1$)=talk(x$_1$)

(Ü1,Ü4, β-Konversion, Down-Up-Konversion)

2. (believe that he$_1$ talks)‘=bel‘(^talk (x$_1$)) *(Ü7)*

3. (he$_1$ believes that he$_1$ talks)‘

=λP.[ˇP(x$_1$)](^ bel‘(^talk (x$_1$)))

= ˇ^ bel‘(^talk (x$_1$)) (x$_1$)

= bel‘(^talk (x$_1$)) (x$_1$) *(Ü4, β-Konversion, Down-Up-Konversion)*

4. (woman)‘=woman‘ *(Ü1)*

5. (woman who believes that he$_1$ talks)‘ =λ x$_1$.[woman‘(x$_1$) bel‘(^talk (x$_1$)) (x$_1$)]

(Ü3)

6. (a)‘= λ Q.[λ P.[EXISTS(ˇQ) (ˇP)]] *(Ü1)*

7. (a woman who believes that he$_1$ talks)‘

=λ Q.[λ P.[EXISTS(ˇQ) (ˇP)]] (^λ x$_1$.[woman‘(x$_1$) bel‘(^talk (x1)) (x1)])

= λ P.[EXISTS(ˇ^λ x1.[woman‘(x1) bel‘(^talk (x1)) (x1)]) (ˇP)]

= λ P.[EXISTS(λ x1.[woman‘(x1) bel‘(^talk (x1)) (x1)]) (ˇP)]

(Ü2, β-Konversion, Down-Up-Konversion)

8. (walk)‘=walk‘ *(Ü1)*

9. (a woman who believes that he$_1$ talks walks)‘

=λ P.[EXISTS(λ x$_1$.[woman‘(x$_1$) bel‘(^talk (x$_1$)) (x$_1$)]) (ˇP)](^walk‘)

=EXISTS(λx1.[woman‘(x$_1$) bel‘(^talk (x$_1$)) (x$_1$)]) (ˇ^walk‘)

=EXISTS(λ x$_1$.[woman‘(x$_1$) bel‘(^talk (x$_1$)) (x$_1$)]) (walk‘)]

(Ü4, β-Konversion, Down-Up-Konversion)[9]

4 Schlussbemerkung: Verwendungsmöglichkeiten

Wie schon beim Beispiel „Monkey Island“ gesehen, kann das Lambda-Kalkül in Adventure Games zur Steuerung der Spiele genutzt werden. Dabei müssen nicht unbedingt wie in „Monkey Island“ Worte für die Aktionen verwendet werden. Ebenso sinnvoll und vom Spieler vielleicht einfacher zu verstehen, sind Symbole (Icons) für die Aktionen. Um eine Aktion im Spiel auszuführen klickt der Spieler nun erst das Symbol an und anschließend die Objekte

[9] http://www.ifi.unizh.ch/CL/hess/classes/ecl2/semantik.pdf#search=%22Montague%20Semantik%20PTQ%22 vom 01.09.2006, Seite 115

mit denen er die Aktion ausführen will. Ebenso könnten Strategiespiele gesteuert werden. So hat der Spieler zum Beispiel ein Symbol „Straße“, mit welchem er eine Straße bauen kann. Er klickt es an und anschließend noch Start- und Endpunkt der Straße. Nun werden genügend viele Einheiten an die entsprechende Stelle geschickt und die Straße wird gebaut.

An manchen Stellen in Strategiespielen ist es jedoch sinnvoller ganze Sätze zu tippen als die Aktion mit Klicks durchzuführen. Dazu ein Beispiel: Im Spiel „Schlacht um Mittelerde 2“ geht es darum, seinen Gegner oder seine Gegner zu vernichten. Zunächst baut der Spieler alle möglichen Gebäude und erschafft seine Einheiten. Mit diesen kann er schließlich seine Gegner angreifen. Da dieser nicht nur in seinem Gebiet herumsitzt und darauf wartet, dass er angegriffen wird, kann es sein, dass es sinnvoll ist an mehreren Orten gleichzeitig anzugreifen. Oder der Spieler muss sich gegen einen Gegner verteidigen und greift zusätzlich noch einen anderen oder den selben Gegner an. Das heißt, dass er nun seine Einheiten auf der Karte an verschiedenen Stellen verteilt. Aus verschiedenen Gründen ist es nun manchmal sinnvoll, alle seine Einheiten (wenn man sich bei der Stärke des Gegners verschätzt hat und nur ein Ziel schafft, wenn der Gegner an einer Stelle besiegt ist) oder alle Einheiten eines bestimmten Typs (wenn der Einheitentyp an ursprünglichen Ziel nichts mehr ausrichten kann) an einem Punkt zu sammeln. Bisher sucht der Spieler alle seine Einheiten, markiert sie in Gruppen soweit es möglich ist und schickt sie dort hin, wo sie sich Sammeln sollen. Wenn der Spieler Pech hat, muss er dies mit jeder Einheit einzeln machen. Außerdem muss er sich den Sammelpunkt merken. Zwar könnten für diesen Zweck wieder Symbole eingesetzt werden, aber dann würde ein Symbol für alle Einheiten und je ein Symbol pro Einheitentyp gebraucht werden. Dann müsste jedoch auch erst geschaut werden, welche Typen dem Spieler zur Verfügung stehen (unterschiedliche Typen bei unterschiedlichen Rassen, unterschiedliche Ausbaustufen). Außerdem könnten dann nicht Befehle wie „alle Einheiten außer die Baumeister“ oder „alle Ents und alle Baumbarts“ ausgeführt werden. Beziehungsweise dies würde wieder zum selben Problem wie schon ohne die Symbole führen. Wesentlich schneller ginge es, wenn der Spieler ein Befehlsfenster hätte und dort zum Beispiel hineinschreiben könnte „Wähle alle Einheiten aus“ oder „Wähle alle Einheiten außer den Baumeistern aus“ oder „Wähle alle Ents und alle Baumbarts aus“. Sind diese nun ausgewählt, könnte der Spieler sie mit einem Rechtsklick auf die entsprechende Stelle auf die Karte an einen Ort schicken oder mit einem Linksklick auf ein gegnerisches Gebäude zum Angriff gegen den Gegner schicken. Jedoch ist diese Variante wesentlich schneller als die Variante mit Symbolen oder die jetzige Variante mit Klicks.

Würde in Spielen Spracherkennung einsetzen werden, könnten nicht nur Befehle des Spielers über Icons und geschriebener Sprache entgegen genommen werden, sonder auch über gesprochene Sprache. Jedoch ist der Einsatz der Spracherkennung noch sehr umstritten. Es wird gesagt, dass es für die Spieler unnatürlich ist, mit ihrem Computer während des Spiels zu reden. Jedoch könnten sich einige Spieler auch vorstellen ihre Spiele über Sprache zu steuern, wenn die Technik entsprechend gut wäre.[10] So könnten zum Beispiel Actionspiele mit Sprache gesteuert werden. Man klickt nun nicht mehr die Gegner an, sondern gibt den

[10] Informationen von Befragungen der Entwickler und Besucher auf der Games Convention 2006 in Leipzig vom 28.08.2006-27.08.2007

Befehl seinen Gegner anzugreifen. In Strategiespielen würde der Spieler nicht schreiben oder auf Icons klicken, sondern sprechen. Das Spielerlebnis würde dadurch natürlicher wirken.

Das Lambda-Kalkül bietet auch nicht-linguistische Einsatzmöglichkeiten in Computerspielen. Die Entwicklung neuer Titel tendiert immer mehr zu möglichst großer Handlungsfreiheit für den Spieler. Dazu zählt es auch, alle möglichen Gegenstände der Spielwelt nutzen oder manipulieren zu können. In der Regel müssen aber all diese Interaktionsmöglichkeiten einzeln programmiert werden.

Eine (wahrscheinlich) wesentlich einfachere Variante wäre die Verwendung des Lambda-Kalküls. Jeder Gegenstandstyp müsste dabei mit Eigenschaften und einem „Aktionsinterface“ versehen werden. Dieses Interface bestimmt, mit welchen Gegenstandstypen der betreffende Gegenstand wie interagieren kann. Dabei beschränkt sich diese Technik nicht auf 1:1 Beziehungen, sondern lässt im Prinzip beliebig viele Kombinationen zu. So könnte ein Spieler beispielsweise einen Stock mit einer Rolle, einem Faden und einem Haken kombinieren und erhält dann eine funktionstüchtige Angel. Wird diese nun mit einem Köder versehen und in ein See oder Fluß gehängt, könnte dessen Interface wiederum besagen, dass mit einer Wahrscheinlichkeit von w nach x-Sekunden ein Fisch anbeißt. Um dies zu erreichen, könnten mehrere „Sockel“ im Interface vorgesehen werden. Über das Kalkül würde dabei immer geprüft werden, ob die gewünschte Kombination gültig ist. Da viele Gegenstände ähnliche Interaktionsmöglichkeiten besitzen, bietet es sich außerdem an, Vererbung einzusetzen. Ein Stein könnte so ein Felsen mit zusätzlichen Interaktionsmöglichkeiten sein. Der Spiele könnte ihn - im Gegensatz zum Felsen - auch tragen oder werfen. Mit diesen beiden Sockeln können andere Gegenstände verwendet werden, diese Aktion unterstützen. Hände, ein Eimer oder eine Schleuder böten sich hier an.

Der Vorteil besteht in einem klaren Regelwerk, das dem Spieler viele Freiheiten gibt und - zusammen mit einer vernünftigen Physik-Engine - sämtliche Interaktionsmöglichkeiten, also nicht nur die sprachlichen, im Spiel kontrolliert. Nicht jede Kombinationsmöglichkeit müsste einzeln aufgezählt und definiert werden, sondern ergibt sich aus den Handlungssockeln der Typen und den Regeln des Lambda-Kalküls.

5 Abbildungsverzeichnis

Abbildung 1:

http://www.kultpower.de/external_frameset.php3?site=%2Famigajoker_testbericht.php3%3Fnumber_of%3D10%26start%3D0%26game_id%3D%26letter%3DT%26width%3D1133%26height%3D859%26im%3Dthesecretofmonkeyisland.jpg&from_referer=http%3A%2F%2Fwww.kultpower.de%2Famigajoker_datenbank.php3%3Fhits%3D%26number_of%3D10%26letter%3DT

Datum: 01.09.2006

Abbildung 2:

http://www.kultpower.de/external_frameset.php3?site=%2Famigajoker_testbericht.php3%3Fnumber_of%3D10%26start%3D0%26game_id%3D%26letter%3DT%26width%3D1133%26height%3D859%26im%3Dthesecretofmonkeyisland.jpg&from_referer=http%3A%2F%2Fwww.kultpower.de%2Famigajoker_datenbank.php3%3Fhits%3D%26number_of%3D10%26letter%3DT

Datum: 01.09.2006

Abbildung 3:

http://content.answers.com/main/content/wp/en/e/ef/The_Secret_of_Monkey_Island.PNG

Datum: 01.09.2006

Abbildung 4:

http://www.scummvm.org/screenshots/scummvm_0_4_5.jpg

Datum: 01.09.2006