AMSEL – ein Lernsystem zum Algorithmenentwurf

Jörg Desel[1], Leo von Klenze[2]

[1]FernUniversität in Hagen, 58084 Hagen
[2]TNG - Technology Consulting GmbH, 85774 Unterföhring
joerg.desel@fernuni-hagen.de, leo.vonklenze@tngtech.com

Abstract: Der in dieser Arbeit vorgestellte Ansatz vereinfacht den Entwurf eines Algorithmus. Anwender erstellen zunächst Beispielabläufe passend zur Problembeschreibung. Diese Beispielabläufe werden formalisiert und zu einem Algorithmus synthetisiert. Das Lernsystem AMSEL unterstützt den gesamten Vorgang von der Erstellung der Beispielabläufe bis hin zum synthetisierten Algorithmus. Die interaktive Gestaltung bindet den Benutzer aktiv ein und erhöht so den Lerneffekt beim Algorithmenentwurf. Eine Evaluierung schließt die Arbeit ab.

1 Einleitung

Die Fähigkeit, einen Algorithmus zu entwerfen, muss ebenso erworben werden, wie die Fähigkeit, seine formale Darstellung zu lesen und zu verstehen. Es reicht nicht aus, nur die Syntax einer Programmiersprache sowie die Bedeutung ihrer Konstrukte zu erklären, obwohl gerade dies in der Lehre oft im Vordergrund steht. Der Doppelschritt der gleichzeitigen Abstraktion von verstandenen einzelnen Abläufen auf allgemeinere Ablaufstrukturen und die Formalisierung der Strukturen auf Befehle einer Programmiersprache überfordert Anfänger bei kleineren Aufgabenstellungen und auch Fortgeschrittene bei größeren. Auf einem alternativen, ablauforientierten Ansatz [Des08] basiert das in dieser Arbeit vor gestellte System AMSEL, welches den Lernenden beim Algorithmenentwurf unterstützt.

Das Ziel dieser Arbeit ist somit eine Machbarkeitsstudie, die die Tauglichkeit dieses Ansatzes zunächst wenigstens für eine eingeschränkte Problemklasse untersucht. Der Ansatz verlangt nicht das gleichzeitige Abstrahieren und Formalisieren. Er fordert zunächst die Erstellung von Beispielabläufen aus der Problembeschreibung. Anschließend müssen die Abläufe formalisiert werden. Dieser Schritt ist im Allgemeinen nicht einfach, da der formale Rahmen zuerst gefunden werden muss (dies ist aber auch beim traditionellen Vorgehen notwendig). Sind die Beispielabläufe formalisiert, werden sie werkzeugunterstützt zu einem Algorithmus synthetisiert.

Das System AMSEL ist auf Algorithmen eingeschränkt, die einen Roboter in einer zweidimensionalen Welt steuern. Diese Umgebung wurde als Rahmen gewählt, da sie besonders leicht verständlich ist. Das System gibt dem Lernenden die Möglichkeit, Beispielabläufe einfach zu generieren. Die Formalisierung geschieht implizit durch das System. Aus den Beispielabläufen wird systemunterstützt ein Algorithmus entwickelt. Dadurch kann der Lernende spielerisch den Umgang mit Algorithmen erlernen und erfahren, wie sich bestimmte Kontrollstrukturen, z. B. Schleifen, aus den Problembeschreibungen ergeben.

Einige Ansätze setzen sich mit ähnlichen Ideen auseinander. Beim Process Mining wird versucht, aus protokollierten betrieblichen Abläufen (sog. event logs) ein Prozessmodell zu synthetisieren [Aa03]. Dieses Modell basiert jedoch lediglich auf den Informationen der Eingabeprotokolle. Das Einfügen von Bedingungen, die zusätzliche Informationen liefern, ist nicht interaktiv, sondern erst im fertigen Prozessmodell möglich.

Ein exaktes Verfahren zur Synthese aus Abläufen beschreibt [Be07]. Hier werden Abläufe zu LPO-Netzen (Labeled Partial Order) formalisiert und anschließend zu einem Petrinetz synthetisiert. Auch dieses Verfahren basiert rein auf Eingabeabläufen, zusätzliche Informationen werden nicht berücksichtigt. Ziel ist zwar auch die Erstellung eines korrekten Algorithmus (im obigen Kontext: eines korrekten Netzes), jedoch nicht die Vermittlung von Entwurfskompetenz. Der Anwender unseres Ansatzes soll nicht nur einen fertigen Algorithmus erhalten, sondern verstehen, wie dieser Algorithmus zustande kam.

In [FK] wird mit Robot Karol auf den ersten Blick ein zu unserer Arbeit ähnlicher Ansatz verfolgt. Dort wird ein Werkzeug vorgestellt, das ebenfalls eine Umgebung für einen Roboter bereitstellt. Diese hat sogar drei Dimensionen, und der Roboter kann mit Objekten in seiner Welt interagieren. Der wesentliche Unterschied besteht jedoch in der Art und Weise, wie dieses Programm verwendet wird. Der Roboter wird nicht direkt gesteuert. Vielmehr bietet das Programm eine besonders einfache Programmiersprache, um den Roboter zu programmieren. Der Anwender muss sich also über die Try-and-error-Methode an den richtigen Algorithmus herantasten.

Ebenfalls auf einfache Programmierung ausgelegt ist das in [HM03] vorgestellte Konzept, welches erweiterte Live-Sequence-Charts als Sprache verwendet. Auch hier kann der Anwender konkrete Anwendungsfälle modellieren. Dieses Programm ist jedoch mehr für die Systementwicklung als für die Lehre geeignet.

Eine weitere Parallele findet sich in [Lie01]. Dort wird beschrieben, wie unterschiedliche Software so gestaltet werden kann, dass sie durch Vorspielen von Aktionen durch den Anwender, lernt und diese Vorgänge wiederholen kann. Dies spielt eher in den Bereich der Künstlichen Intelligenz als in den Bereich des Algorithmenentwurfs hinein.

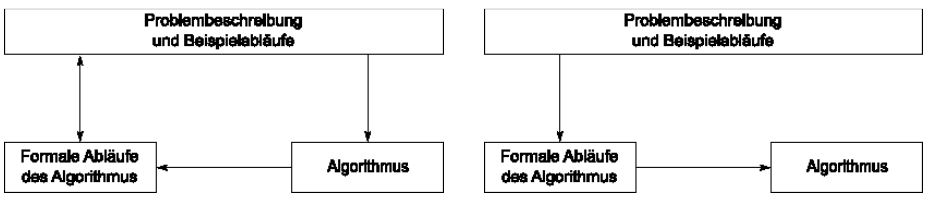
Erste Vorarbeiten zu dem hier vorgestellten Konzept sind in [Des08] und [DN08] beschrieben. [Des08] beschreibt den Ansatz allgemein für die Modellierung. Bei [DN08] handelt es sich um eine Studie, die in einer Schule durchgeführt wurde. Bei dieser Studie wurden zwei Schulklassen auf verschiedene Art unterrichtet: eine nach dem klassischen Lehransatz und die andere nach dem auch hier verwendeten ablauforientierten Ansatz. Das Ergebnis ist zwar nicht eindeutig; allerdings wird durchaus deutlich, dass der ablauforientierte Ansatz einige Vorteile gegenüber der traditionellen Vorgehensweise bietet.

In Abschnitt 2 dieser Arbeit wird der ablauforientierte Ansatz beschrieben. Abschnitt 3 beschreibt das System AMSEL, Abschnitt 4 geht auf seine Implementierung ein und Abschnitt 5 enthält eine Kurzfassung der Evaluierungsergebnisse.

Diese Arbeit basiert auf [vK09]. Das System AMSEL, seine Bedienungsanleitung, die verwendeten Fragebögen und weitere Informationen findet man auch unter [vK].

2 Ablauforientierter Ansatz

Ein Algorithmus beschreibt eine Vorgehensweise, um eine Klasse von Problemen zu lösen. Eine formale Beschreibung eines Algorithmus wird notwendig, z.B. in Form einer Programmiersprache, wenn der Algorithmus von einem Computer verstanden werden soll. Bei der Vermittlung von Algorithmen steht üblicherweise die Vermittlung formaler Konstrukte, wie Schleifen oder Verzweigungen, im Vordergrund. Häufig werden diese Konstrukte anhand einer Programmiersprache eingeführt, damit sie danach leicht übertragen werden können. Damit ein spezielles Problem oder gar eine Klasse von Problemen gelöst werden kann, reichen diese Kenntnisse jedoch nicht aus. Der Problemlöser benötigt ebenso Wissen darüber, wie er ein Problem so erfassen kann, dass er in die Lage kommt, einen lösenden Algorithmus zu finden. Er benötigt also eine Strategie, um von einem Problem auf den zugehörigen Algorithmus zu schließen. Natürlich gibt es sehr unterschiedliche Probleme, und es gibt kein generell gültiges, allgemeines Lösungsschema. Es gibt jedoch verschiedene Herangehensweisen, um einen Algorithmus für ein Problem zu finden.



(a) Traditioneller Ansatz (b) Ablauforientierter Ansatz

Abbildung 1: Traditioneller und ablauforientierter Ansatz im Vergleich

Eine häufig praktizierte Herangehensweise zeigt Abbildung 1a. Um einen Algorithmus aus einer Problembeschreibung zu finden, muss der Programmierer zwei Schritte auf einmal durchführen: informelle Beispielabläufe, die aus der Problembeschreibung folgen, müssen analysiert und zu einem allgemeineren Verfahren abstrahiert werden, und gleichzeitig muss ein Algorithmus in einer bestimmten formalen Sprache erzeugt werden, der das beschriebene Problem löst. Diese Vorgehensweise ist bei einfachen Problemen noch leicht zu bewerkstelligen. Bei komplexen Problemen scheitern ungeübte Programmierer meist schon bei der Abstraktion, bei der viele Fälle berücksichtigt werden müssen. Diese Fälle müssen nämlich geeignet zusammengefasst werden, um einen allgemeinen Algorithmus zu entwickeln. Die von diesem Algorithmus erzeugten Abläufe werden schließlich mit den Beispielabläufen verglichen.

Der Ansatz aus Abbildung 1b wirkt diesem Problem entgegen, indem er den Formalisierungsschritt vom Abstraktionsschritt trennt. Dieser Ansatz versteht den Algorithmus als die Zusammenfassung aller möglichen Abläufe. Daher werden die Beispielabläufe zuerst formalisiert. Dazu ist auch eine formale Sprache nötig, allerdings entfällt die gleichzeitige Abstraktion. Ein formalisierter Ablauf besteht nur aus einer Folge von Ereignissen. Er beinhaltet keine Schleifen oder Verzweigungen, die beachtet werden müssen. Daher lässt er sich leicht erfassen und verstehen. Ein Problem mag die Identifizierung und Abgrenzung der einzelnen Ereignisse darstellen, um eine geeignete formale Sprache zu finden. Sind die Abläufe formalisiert, können sie zu einem Algorithmus abstrahiert, also zu-

sammengefasst werden. Dieser Schritt kann auf Grundlage der vorliegenden formalen Abläufe werkzeugunterstützt durchgeführt werden. Dies ermöglicht auch die Konstruktion umfangreicher Algorithmen. Nachdem der Algorithmus erstellt wurde, können zudem seine Abläufe einfach mit der Vorgabe verglichen werden, da sie in derselben formalen Sprache wie die formalisierten Beispielabläufe vorliegen.

Der ablauforientierte Ansatz soll durch die eben vorgestellte Zerlegung in zwei Schritte die Modellierung von Algorithmen vereinfachen und dadurch verständlicher für Anfänger ein. Die Vermittlung formaler Sprachen und Konstrukte wie Schleifen ist natürlich weiterhin notwendig, da sie im Algorithmus zur Abstraktion benötigt werden, ihre Vermittlung steht jedoch nicht im Vordergrund. Vielmehr liegt der Schwerpunkt auf der Vermittlung von Kompetenz zur Erstellung von Algorithmen.

3 Das Konzept des Lernsystems AMSEL

Mit dem Lernsystem AMSEL[1] soll festgestellt werden, ob der Ansatz wenigstens für ein kleines Beispiel realisierbar und gewinnbringend ist. Es soll Lernende im Umgang mit Algorithmen und deren Entwurf unterstützen. Ein Schwerpunkt ist die Vermittlung des Konzepts bedingter Schleifen. Das System geht von Beispielabläufen für gewisse Konfigurationen aus. Der generierte Algorithmus soll dann sämtliche Abläufe, auch für andere Konfigurationen, erzeugen können. Der gesamte Vorgang ist interaktiv, damit Lernende die einzelnen Schritte nachvollziehen können. Primäres Ziel ist nicht die erfolgreiche Algorithmensynthese, sondern die Vermittlung der Vorgehensweise und die damit verbundene Steigerung der Kompetenz zum Entwurf von Algorithmen.

Als Umgebung wurden zweidimensionale Welten gewählt, ähnlich Schachbrettern, in der sich ein Roboter bewegen und Aktionen durchführen kann. Die betrachteten Algorithmen steuern also einen Roboter, in Abhängigkeit von seiner Umgebung. Die Anzahl der Felder in beiden Dimensionen ist variabel und definiert unterschiedliche Welten. Das System ist im Hinblick auf die Konfiguration der Welten und die möglichen Aktionen des Roboters erweiterbar. Eine Grundkonfiguration, die umgesetzt wurde, ist im Folgenden beschrieben. Eine Roboterwelt $W_{b,h}$ hat die Breite b und die Höhe h. Für einzelne Felder sind die Objekte Wand und Startfeld definiert:

Wand: Der Roboter kann Felder mit einer Wand nicht befahren, und es können auch keine weiteren Objekte auf solchen Feldern platziert werden.

Startfeld: Wird ein Roboter in einer Welt mit einem Startfeld initialisiert, soll er auf dem Startfeld mit einer bestimmten Blickrichtung starten; die Position des Anfangszustandes soll also der Position des Startfeldes entsprechen.

Der Roboter besitzt eine Position und eine Blickrichtung. Er soll die folgenden drei Anweisungen verstehen. Die Tabelle listet auch die entstehenden Ereignisse auf.

[1] Algorithmen modellieren spielend erlernen

Anweisung	Beschreibung	Ereignis
Schritt vorwärts:	Der Roboter bewegt sich in Blickrichtung ein Feld nach Vorne, wenn dieses nicht blockiert ist	Schritt vorwärts
Drehe rechts:	Der Roboter dreht sich um 90 nach rechts.	Drehung rechts
Drehe links:	Der Roboter dreht sich um 90 nach links.	Drehung links

Wenn der Roboter auf dem Feld (x, y) steht, gibt sein Blickfeld an, auf welches Feld er bei der entsprechenden Richtung „schaut“.

r	Beschreibung	Blickfeld
0	Norden	(x, y + 1)
1	Osten	(x + 1, y)
2	Süden	(x, y - 1)
3	Westen	(x - 1, y)

Der Roboter kann Bedingungen prüfen; z.B. kann er feststellen, ob er vor einer Wand steht. Wir definieren für die gewählte Konfiguration folgende Elementarbedingungen:

Wand: Diese Bedingung prüft, ob ein Feld von einer Wand blockiert ist. Die Auswertung liefert wahr, wenn das entsprechende Feld durch eine Wand blockiert ist.

besuchtes Feld: Diese Bedingung prüft, ob das Feld bereits vom Roboter besucht wurde. Dafür muss der Roboter bei Besuch eines Feldes dieses als besucht markieren.

Der Roboter kann, unabhängig von seiner Position, die Bedingungen für jedes Feld überprüfen. Formal besteht die Bedingung aus der Aussage und dem Feld, für das die Aussage ausgewertet werden soll. Die Bedingungen „Wand (1 Feld) vorne“ und „Wand (1 Feld) rechts“ sind also verschieden. Es gibt für eine Welt $W_{b,h}$ also $2 \bullet (b+2) \bullet (h+2)$ Bedingungen (die Wand, die die Welt umgibt, muss mitgerechnet werden).

Grundsätzlich unterstützt das System die folgenden vier Schritte, die im folgenden Abschnitt detaillierter beschrieben werden.

1. Erstellen von Beispielabläufen aus der Problembeschreibung
2. Formalisieren der Beispielabläufe
3. Interaktive Synthese des Algorithmus
4. Testen des synthetisierten Algorithmus

Neben diesen Schritten gibt es einige weitere notwendige Aufgaben:

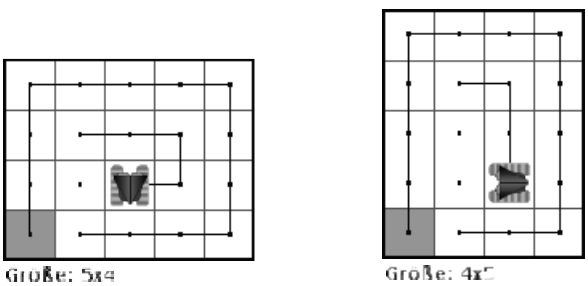
Welten erstellen: Um die Beispielabläufe ausgehend von der Problembeschreibung aufzeichnen zu können, sind entsprechende Welten notwendig, in denen mit dem Roboter das gewünschte Verhalten eingespielt werden kann. Das Programm benötigt also einen Editor, um solche Welten zu erstellen.

Szenariensammlung erstellen: Wird ein Beispielablauf in einer bestimmten Welt mit einem Roboter aufgezeichnet, so entsteht ein Szenario. Der Benutzer hat die Möglichkeit, mehrere dieser Szenarien zu einer Sammlung zusammenzufassen, aus der

dann der Algorithmus synthetisiert wird. Des Weiteren können Benutzer Syntheseparameter anpassen, z.B. bzgl. Bedingungen, die während des Synthesevorgangs verwendet werden können.

4 Vorgehensweise von AMSEL

Ein Beispiel einer Problembeschreibung unserer Konfiguration ist: „Fahre die komplette Welt so von außen nach innen ab, dass die Fahrspur eine Rechtsspirale darstellt“. Diese Problembeschreibung wird durch den Ablauf aus Abbildung 2a verdeutlicht (der Roboter startet auf dem Feld links unten mit Blick nach Norden). Abbildung 2b präzisiert die Aufgabenstellung weiter. Nun ist erkennbar, dass sich der Roboter am Ende noch einmal nach rechts drehen soll. Wird nur der Ablauf aus Abbildung 2a betrachtet, könnte auch gemeint sein, dass der Roboter am Ende nach Norden blicken soll. Beide Abläufe zusammen bilden jedoch eine hinreichende Problembeschreibung.



(a) (b)

Abbildung 2: Beispielabläufe für eine Rechtsspirale

Der Lernende soll auf einfache Weise derartige Beispielabläufe visuell unterstützt erstellen können. Dazu bietet das System ein „Spielfeld“, auf dem der Roboter gesteuert werden kann. Die durch den Benutzer ausgeführten Aktionen werden aufgezeichnet und in einem sichtbaren Protokoll festgehalten. Es stellt den aufgezeichneten Ablauf dar. Dieser Ablauf kann wieder abgespielt und verändert werden, bis der gewünschte Ablauf gefunden ist. Ein Beispielablauf in einer bestimmten Welt wird Szenario genannt. Da mehrere Beispiele nötig sind, um eine Verallgemeinerung durchführen zu können, kann der Benutzer mehrere Szenarien zu einer Szenariensammlung zusammenzufassen.

4.1 Formalisieren erstellter Beispielabläufe

Dieser Schritt geschieht implizit bei der Aufzeichnung der Beispielabläufe. Für die Eingabe der Beispielabläufe werden Roboteraktionen in einer bestimmten Welt ausgeführt, dem Roboter werden also entsprechende Anweisungen erteilt. Jede dieser Aktionen führt zu einem Ereignis, welches im Protokoll festgehalten wird. Wird ein Ereignis als Zeichen aus einem Alphabet aufgefasst, so kann das Protokoll als Wort über diesem Alphabet betrachtet werden.

Das System stellt sicher, dass der Benutzer nur vordefinierte Ereignisse auslösen kann, da er nur bestimmte Aktionen zur Verfügung hat. Hier besteht das Alphabet A aus drei Zeichen: „Schritt vorwärts“ (v), „drehe rechts“ (r) und „drehe links“ (l), also A = {v, r, l}. Ein Beispielablauf wird damit automatisch zu einem Wort aus A* formalisiert.

4.2 Synthese des Algorithmus

Hat der Benutzer die gewünschten Beispielabläufe aufgezeichnet (und damit gleichzeitig formalisiert) und in einer Szenariensammlung zusammengefasst, kann aus dieser Sammlung der Algorithmus synthetisiert werden. Der Anwender wird konsequent in den Vorgang eingebunden, indem er interaktiv die einzelnen Schritte ergänzen und bestätigen muss. Auf diese Weise soll er nicht nur das Endergebnis, nämlich den synthetisierten Algorithmus, sondern auch die Verfahrensweise kennenlernen.

Ziel der Synthese ist es, dass der synthetisierte Algorithmus mindestens die Abläufe aus den Szenarien der Szenariensammlung erzeugen kann (und möglichst nicht viel mehr). Dies hängt jedoch stark von den eingegebenen Beispielabläufen ab und kann nicht allgemein garantiert werden. Allerdings ist für eine bestimmte Klasse von Szenariensammlungen sichergestellt, dass der synthetisierte Algorithmus für alle Abläufe aus den enthaltenen Szenarien gültig ist und gewisse Korrektheitskriterien erfüllt. Diese Klasse wird in [vK09] genauer beschrieben. Das Syntheseverfahren besteht aus den folgenden fünf Schritten:

1. **Erzeugung von Syntheseobjekten:** Die Eingabe ist eine Szenariensammlung. In diesen sind Abläufe und die dazugehörigen Welten enthalten. Die Abläufe werden zuerst in elementare Algorithmen umgewandelt, die nur den Ablauf erzeugen, aus dem sie entstanden sind (reine Sequenzen von Anweisungen und Zählschleifen). Sie werden mit ihrem dazugehörigen Szenario in einem Syntheseobjekt gespeichert.

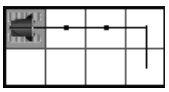
2. **Horizontaler Abgleich:** Beim ersten horizontalen Abgleich wird versucht, die einzelnen Algorithmen aus den Syntheseobjekten so zu verallgemeinern, dass sie nicht mehr nur den Ablauf aus ihrem zugehörigen Szenario erzeugen können. Dazu werden die Algorithmen „horizontal angeordnet“ und Anweisung für Anweisung miteinander verglichen. Mit Hilfe der zugehörigen Welten, die zusätzliche Informationen liefern, werden möglichst viele Zählschleifen durch Bedingungsschleifen ersetzt.

3. **Vertikale Kompression:** In einer anschließenden vertikalen Kompression werden die Algorithmen jeweils „vertikal“ untersucht. Durch das Auffinden von sich wiederholenden Mustern und das Einbetten dieser Muster in Zählschleifen werden die Algorithmen verkürzt. In diesem Schritt werden keine Zählschleifen zu Bedingungsschleifen verallgemeinert. Allerdings werden ggf. Bedingungsschleifen miteinander kombiniert.

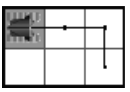
4. **Horizontaler Abgleich:** Nach der vertikalen Kompression bestehen die Algorithmen unter Umständen wieder aus Zählschleifen. Diese können bei dem horizontalen Abgleich wieder verallgemeinert werden.

5. Verschmelzung: Alle vorigen Schritte haben auf einer Menge von Syntheseobjekten gearbeitet und auch ein solches erzeugt. In diesem Schritt werden die Algorithmen aus den verschiedenen Syntheseobjekten miteinander kombiniert. Für gewisse Szenariensammlungen sind alle Algorithmen in den verschiedenen Syntheseobjekten gleich, und es kann ein beliebiger ausgesucht werden. Andernfalls sind die Algorithmen jedoch unterschiedlich, und es muss der „richtige“ ausgewählt werden.

Exemplarisch soll der zweite Schritt „Horizontaler Abgleich“ an einem Beispiel genauer dargestellt werden. Ausgangspunkt sind zwei Algorithmen, die jeweils aus Szenarien hervorgegangen sind. In Abbildung 3 sind über den Algorithmen die zugehörigen Welten dargestellt. Die graue Linie markiert den vom Benutzer gewählten Ablauf. Es werden



1 *Schritt vorwärts*
2 Von 1 Bis 1 (1x)
3 *Drehe rechts*
4 Von 1 Bis 1 (1x)
5 *Schritt vorwärts*



1 *Schritt vorwärts*
2 Von 1 Bis 1 (1x)
3 *Drehe rechts*
4 Von 1 Bis 1 (1x)
5 *Schritt vorwärts*

Abbildung 3: Abläufe und ihre zugehörigen elementaren Algorithmen

nun die ersten beiden Zählschleifen miteinander verglichen. Da sich die Zählgrenzen unterscheiden, müssen sie verallgemeinert werden, indem sie in Bedingungsschleifen umgewandelt werden. Beide Zählschleifen können durch eine in beiden Welten gültige Bedingungsschleife mit der Bedingung „Nicht Wand (1 Feld) vorne“ ersetzt werden. Der folgende Algorithmus kann daher beide Abläufe erzeugen.

1 Solange Nicht Wand (1 Feld) vorne
2 *Schritt vorwärts*
3 Von 1 Bis 1 (1x)
4 *Drehe rechts*
5 Von 1 Bis 1 (1x)
6 *Schritt vorwärts*

Die beiden restlichen Schleifen haben dagegen dieselben Zählgrenzen und müssen deshalb nicht verallgemeinert werden.

Um eine Zählschleife durch eine Bedingungsschleife ersetzen zu können, muss eine passende Bedingung für die neue Schleife gefunden werden. Für sinnvolle und in den Kontext passende Bedingungen sind nicht nur die Algorithmen der Syntheseobjektmenge relevant, sondern auch die zugehörigen Welten, da eine Bedingung sich immer auf eine Welt bezieht (z. B. „Wand (1 Feld) vorne“).

Mögliche Bedingungen werden dem Lernenden von AMSEL angeboten. Die entsprechende Systemfunktion besteht im Wesentlichen aus drei Teilen: Generieren von Bedingungen, Filtern von Bedingungen und Kombinieren und Testen von Bedingungen. Das

Filtern reduziert die Anzahl der später zu kombinierenden Bedingungen. Dies ist aus Komplexitätsgründen wichtig, weil jede erzeugte Kombination in jedem Szenario getestet werden muss.

Um die Bedingungen zu filtern, wird ein Roboter erstellt, der alle Anweisungen des Algorithmus in der zugehörigen Welt von Anfang bis zur betrachteten Stelle ausführt. Für die Schleife wird eine Bedingung gesucht, die bei jedem bisherigen Schleifeneingang dieses Ablaufs zutraf und in dem aktuellen Zustand des Roboters nicht mehr zutrifft. Zunächst werden potentielle Abbruchbedingungen gespeichert, also Bedingungen, die genau dann wahr sind, wenn die Schleife nicht weiter ausgeführt werden soll. Dazu müssen die Bedingungen getestet werden. Ist eine Bedingung wahr, so wird noch geprüft, ob sie in einem der vorigen Algorithmen nicht falsch war. In diesem Fall wird die Bedingung gespeichert. Ist die Bedingung für keinen der folgenden Algorithmen falsch, so wird sie schließlich zu einer Menge Bed hinzugefügt. Analog wird mit der Negation der Bedingung vorgegangen, wenn der Test „falsch“ zurücklieferte. Auf diese Weise werden alle Abbruchbedingungen in Bed gespeichert.

Für das oben angeführte Beispiel werden dem Benutzer die folgenden Abbruchbedingungen angeboten: Bed = {„Nicht Wand vorne“, „Nicht Wand vorne Oder Wand rechts“}. Er kann nun entscheiden, welcher Grund für das Abbiegen des Roboters im allgemeinen Fall anzugeben ist und entsprechend eine Bedingung auswählen.

Nach vollständiger Ausführung des ersten horizontalen Abgleichs sehen die Algorithmen für das Spiralbeispiel aus Abbildung 2 wie folgt aus:

Für Abbildung 2a:

```
1        Schritt vorwärts
2    Von 1 Bis 1 (1x)
3        Drehe rechts
4    Zeilen 1-4 wiederholen sich dreimal
5    Solange nicht besuchtes Feld v
6        Schritt vorwärts
7    Von 1 Bis 1 (1x)
8        Drehe rechts
9    Zeilen 6-9 wiederholen sich fünf mal
```

Für Abbildung 2b:

```
1        Schritt vorwärts
2    Von 1 Bis 1 (1x)
3        Drehe rechts
4    Zeilen 1-4 wiederholen sich dreimal
5    Solange nicht besuchtes Feld v
6        Schritt vorwärts
7    Von 1 Bis 1 (1x)
8        Drehe rechts
9    Zeilen 6-9 wiederholen sich fünf mal
```

Bei der vertikalen Kompression entstehen wieder zwei Algorithmen. Der folgende Algorithmus ist aus dem linken der eben dargestellten Algorithmen entstanden:

```
1    Von 1 Bis 8 (8x)
2        Solange (Nicht W and vorne Und Nicht besuchtes Feld vorne )
3            Schritt vorwärts
4        Von 1 B i s 1 (1x)
5            Drehe rechts
```

Der Algorithmus, der aus der rechten Seite durch die vertikale Kompression hervorgeht, unterscheidet sich durch eine Zählgrenze von 7 statt 8 bei der ersten Schleife. Ein erneuter horizontaler Abgleich bringt schließlich die „optimalen“ Algorithmen hervor, die für

beide Welten gleich sind. Der Verschmelzungsschritt ist hier nicht mehr notwendig, da bereits alle Algorithmen gleich sind und somit auch jeder Algorithmus für alle Welten gültig ist:

```
1  Solange Nicht besuchtes Feld vorne
2          Solange (Nicht Wand vorne Und Nicht besuchtes Feld vorne)
3                  Schritt vorwärts
4          Von 1 Bis 1 (1x)
5                  Drehe rechts
```

5 Evaluation

Für eine erste Evaluation des Systems AMSEL bekam eine Gruppe von Testpersonen, überwiegend Studenten, einen Fragebogen und die Vorgabe, das Programm für zwei gestellte Probleme zu testen. Ziel der Evaluation war es, herauszufinden, ob das Konzept und dessen Umsetzung sich eignen, die Kompetenz des Algorithmenentwurf insbesondere bei Studenten zu erhöhen. Dabei sollte lediglich eine Tendenz festgestellt werden, weshalb die Ergebnisse keinen statistischen Tests unterworfen wurden.

Der erstellte und verwendete Fragebogen ist in [vK] zu finden. Der Fragebogen wurde nach den Richtlinien von [Pra] und [Sta] erstellt. Die Evaluation wurde mit Hilfe des Fragebogens und des Programms AMSEL durchgeführt, welches jedem Teilnehmer auf einem Rechner während der Evaluation zur Verfügung stand. Für die Evaluation war eine gute Stunde als Zeitrahmen vorgesehen.

Zu Beginn der Evaluation wurden Fragen bezogen auf die bereits vorhandenen Kenntnisse in der Programmierung und im Algorithmenentwurf gestellt. Auf diese Weise kann festgestellt werden, ob ein Lerneffekt durch die Nutzung des Programms eintritt. Anschließend wurde eine kurze Einführung in die genaue Thematik gegeben. Es wurde das Ziel erklärt, nämlich Bewegungsalgorithmen für einen Roboter in einer bestimmten Umgebung zu entwerfen, sowie die vorhandene Konfiguration (welche Aktionen hat der Roboter zur Verfügung, welche Bedingungen kann er abfragen). Die nächste Frage zielte auf die Fähigkeit ab, einen vorhandenen Algorithmus zu lesen und zu interpretieren. Bei der folgenden Frage musste aus der Problembeschreibung ein Algorithmus erstellt werden.

Der nächste Teil der Evaluation bestand aus dem eigentlichen Test des Programms AMSEL. Dazu wurden zwei Problemstellungen und eine kurze Anleitung, wie vorzugehen ist, verteilt. Zum Schluss wurden Fragen über die Bedienung und Benutzerfreundlichkeit des Programms gestellt. Außerdem wurde abgefragt, ob der Benutzer die Prinzipien der einzelnen Schritte verstanden hat (horizontaler Abgleich, vertikale Kompression). Zum Schluss war wieder ein Algorithmus zu entwerfen, dessen Problembeschreibung eine Kombination der zuvor gestellten Aufgaben war.

An der Evaluation nahmen 18 Probanden teil, die größtenteils ein mathematisches Studium absolvieren oder absolvierten. Drei der Probanden kamen aus anderen Bereichen: der Journalistik, der Geographie und der Politik. Unter den 18 Probanden befanden sich 13

Studenten und 5 mit einem abgeschlossenem Studiengang. Die meisten hatten also bereits Vorkenntnisse in diesem Gebiet; meist, weil sie schon einmal programmiert hatten. Lediglich ein Proband hatte keinerlei Vorkenntnisse.

Die Erklärung der Umgebung und des Ziels wurde von den meisten richtig verstanden. Fast alle interpretierten einen vorgegebenen Algorithmus richtig. Eine Folgefrage diente dazu, die Entwurfskompetenz festzustellen, bevor mit dem Programm AMSEL gearbeitet wurde. Hierbei wurde ersichtlich, dass viele nicht von sich aus abstrahieren und versuchen einen allgemeinen Algorithmus zu finden. Stattdessen wurde eine Art Ablauf angegeben, d. h. die Fahrspur wurde Schritt für Schritt in Anweisungen „umgewandelt“ ohne in Betracht zu ziehen, dass Schleifen, oder gar geschachtelte Schleifen, für eine kompaktere und allgemeingültige Darstellung verwendet werden können. Keiner der Probanden hat dabei den Algorithmus in seiner allgemein möglichen und kürzesten Form aufgeschrieben.

Bei der Anwendung von AMSEL musste etwas Hilfestellung bei der Bedienung gegeben werden. Allerdings verfügte das System über keine Hilfefunktion. Die Erstellung der Beispielabläufe lief problemlos; viele Probanden haben aber nicht erkannt, dass sie mehr als einen Beispielablauf erzeugen müssen. Da das Programm in diesem Fall nicht warnt, kam es an dieser Stelle zu Fehlern. Nach einer erneuten Erklärung bewältigten die Probanden die Aufgabe mühelos. Während des Synthesevorgangs gab es kaum inhaltliche Fragen. Einige Bedienungsschritte wurden nicht sofort erkannt, etwa warum der Vorgang nach Beendigung des horizontalen Abgleichs noch nicht beendet war. Der Synthesevorgang stellte keine größeren Probleme dar. Fast alle Probanden haben gültige Algorithmen erzeugt, die meisten sogar in der allgemeinsten und kürzesten Form. Die Testmöglichkeit nach der vertikalen Kompression wurde von allen Probanden wahrgenommen. Die vertikale Kompression bereitete im Vergleich zum horizontalen Abgleich mehr Schwierigkeiten. So zeigt eine Verständnisfrage zum horizontalen Abgleich, dass zumindest der Grundgedanke verstanden wurde, während eine entsprechende Frage zur vertikalen Kompression auf ein geringeres Verständnis schließen lässt.

Die gemessene Entwurfskompetenz ist bei 16 der 18 Probanden leicht gestiegen. Dies lässt zumindest eine positive Tendenz erkennen. Viele verwendeten nach Gebrauch des Programms verschachtelte Schleifen und griffen die Möglichkeiten der allgemeinen Bedingungen auf. Zudem glaubten 10 Probanden, das Werkzeug sei eine Hilfe gewesen, und weitere 6 beantworteten diese Frage mit „eher schon“.

Die Bedienung wurde von mehr als 70% mit „eher leicht“ oder leichter bewertet. Einen Ausreißer bildeten die während des Vorgangs eingeblendeten Texte, deren Verständlichkeit von etwa der Hälfte als „eher schwer“ oder schwerer eingestuft wurde. Der durchschnittliche Wert für die gesamte Bedienung von 3, 44 auf einer Skala von 0 - 5 ist aber positiv. Es gab die Möglichkeit, Verbesserungsvorschläge anzugeben. Diese Möglichkeit wurde nicht von allen Probanden genutzt, lieferte aber einige nützliche Hinweise.

Insgesamt ist das Ergebnis der Evaluation positiv zu werten. Das Konzept und das erstellte Werkzeug bieten die Möglichkeit, den Anwender bei der Entwicklung eines Algorithmus zu unterstützen und ihm gleichzeitig entsprechende Kompetenzen zu vermitteln.

6 Fazit und Ausblick

Mit dem System AMSEL wurde ein Werkzeug erstellt, das für einen einfachen und eingeschränkten Anwendungsbereich den Entwurf von Algorithmen unterstützt. Es dient damit einerseits dem Entwurf von Algorithmen und andererseits der Unterstützung bei der Vermittlung von Entwurfskompetenz. Für das zweitgenannte Ziel wird der Lernende bei jedem Schritt des Algorithmusentwurfs einbezogen, die einzelnen Phasen des Entstehens eines Algorithmus sind sehr transparent. AMSEL setzt konsequent einen ablauforientierten Ansatz um, der davon ausgeht, dass Lernende die Funktionsweise eines Algorithmus in Form von (informellen) Beispielabläufen verstanden haben, bevor sie in der Lage sind, einen entsprechenden Algorithmus zu konstruieren. Eine erste Evaluation von AMSEL zeigt, dass das System die genannten Ziele erfüllt, aber bezogen auf Benutzungsschnittstelle und Hilfestellungen noch verbessert werden kann. Sie beweist jedoch nicht die Verallgemeinerbarkeit des Ansatzes auf beliebige Algorithmentypen und auch nicht die Überlegenheit des Ansatzes gegen über anderen Lehrmethoden für bestimmte oder gar alle Lernertypen.
Wir verstehen AMSEL als ersten Schritt eines größeren Projektvorhabens, in dem der in der Arbeit genannte ablauforientierte Ansatz verfeinert und in einem allgemeineren Kontext erprobt und angewandt werden soll. Insbesondere haben wir vor, nicht nur sehr einfache, sondern auch kompliziertere Algorithmen aus ihren Abläufen zu generieren. Bisherige Erfahrungen legen nahe, dass der Ansatz nicht für alle Lernertypen in gleichem Maße gewinnbringend ist; während einige Lernende zunächst die Abläufe der zu entwickelnden Algorithmen vertiefen, denken andere gleich in algorithmischen Strukturen und betrachten die Abläufe erst anschließend. Gegenstand weiterer Untersuchungen ist daher nicht nur eine Verfeinerung des Konzepts, sondern auch die Charakterisierung potentiell geeigneter Anwendergruppen.

Literaturverzeichnis

[Aa03] Aalst, van der, W. M. P. et al.: A survey of issues and approaches. In: Data & Knowledge Engineering, Vol 47, November 2003; S. 237–267.

[Be07] Bergenthum, R. et al.: Process Mining Based on Regions of Languages. BPM 2007, LNCS 4714, Springer, 2007; S. 375–383.

[Des08] Desel, J.: Modellieren lernen über Formalisieren von Ablaufbeschreibungen. Modellierung in Lehre und Weiterbildung, Workshop auf der Modellierung 2008, S. 37–46.

[DN08] Desel, J.; Neumair, C.: Entwicklung und Bewertung einer Unterrichtssequenz zur ablauforientierten Sichtweise von Algorithmen. LNI 135, GI, DDI 2008; S. 151–152.

[FK] Freiberger, U.; Krsko, O.: Robot Karol. http://www.schule.bayern.de/Karol/download.htm (Stand: 28. Juni 2009)

[HM03] Harel, D.; Marelly, R.: Come, let's play. Springer, Berlin, 2003.

[Lie01] Lieberman, H.: Your wish is my command. Morgan Kaufmann, San Francisco, 2001.

[Pra] Pratzner, A.: Schriftliche Evaluation in der Erwachsenenbildung. http://www.fragebogen.de/schriftliche-evaluation.htm (Stand: 8. Mai 2009)

[Sta] Stangl, W: Schritte der Fragebogen Konstruktion. http://arbeitsblaetter.stangltaller.at/FORSCHUNGSMETHODEN/Fragebogen.shtml (Stand: 8. Mai. 2009)

[vK] Klenze, von, L.: AMSEL online. http://informatik.ku-eichstaett.de/amsel

[vK09] Klenze, von, L.: Werkzeugunterstützter Algorithmenentwurf durch Synthese aus Beispielabläufen. Diplomarbeit, Katholische Universität Eichstätt-Ingolstadt, 2009.