CREACOMP

e−Schulung von Kreativität und Problemlösungskompetenz

Prof. Dr. Bruno BUCHBERGER
Prof. Dr. Erich Peter KLEMENT
Prof. Dr. Günter PILZ

und Mitarbeiter der Institute ALGEBRA, FLLL und RISC

Inhalt

CREACOMP

e−Schulung von Kreativität und Problemlösungskompetenz

Endbericht über das Projekt

1 Zusammenfassung

Das Projekt CREACOMP hatte das Ziel, Kreativität und ‘‘computational aspects’’ in der Mathematik−Ausbildung, zusammen mit dem automatischen Beweisen der gewonnenen Sachverhalte zu entwickeln und zu schulen.

Im ersten Teil sollen Studierende im wesentlichen mit dem an der JKU entwickelten Softwaresystem MeetMATH durch angeleitete Experimente *mathematische Sätze und Lösungsverfahren (er)finden und erkennen* und die Freude am eigenen Erkennen empfinden. Im zweiten Teil sollen die gefundenen Sachverhalte *mathematisch exakt formuliert* und schließlich *automatisch (!) bewiesen* werden. Dieser Teil beruht auf dem ebenfalls an der JKU entwickelten System THEOREMA. Im Rahmen des Projektes CREACOMP wurde eine Integration der beiden Systeme MeetMATH und THEOREMA durchgeführt. Da sowohl MeetMATH als auch THEOREMA auf dem bekannten Mathematik−Softwaresystem Mathematica aufbauen, dient Mathematica auch als Plattform für CREACOMP.

In 16 ausgewählten Lerneinheiten (jeweils für etwa eine Doppelstunde gedacht) wurde gezeigt, dass die Verschmelzung von MeetMATH und THEOREMA wirklich und sinnvoll möglich ist. Dies ermöglicht *weltweit erstmalig eine vollständige Integration von Entdeckung, exakter Formulierung und Absicherung* mathematischer Inhalte in Lerneinheiten des E−learning. In der dem Endbericht beiliegenden CD sind alle diese Lerneinheiten (sowohl als Mathematica−Notebooks als auch in pdf−Form) enthalten. Weiters geben wir für jede dieser Units eine verbale Kurzfassung an. Sodann folgt eine Übersicht über die Arbeitsweise der CREACOMP−Entwicklungsgruppe, Angaben über die Dissemination sowie die finanzielle Endabrechnung. Schliesslich legen wir exemplarisch 3 kommentierte Units in Hardcopy bei (die natürlich nicht die wesentlichen Innovationen im e−Bereich zeigen können, dies versuchen wir in den Kommentaren zu kompensieren).

Die Lerneinheiten werden z.T. bereits jetzt in Lehrveranstaltungen verwendet; diese Einsätze sollen noch wesentlich ausgeweitet werden.

Als Zielgruppen standen uns vor Augen: Studierende, Lehrende und AbsolventInnen der JKU, sowie weiterbildungswillige Berufstätige.

Das CREACOMP−Entwicklungsteam bietet auch eine Live−Präsentation an. Ebenso bieten wir wie im Projektantrag an, CREACOMP nach Ende des Projektes weiterzuentwickeln und aufgrund der Rückmeldungen zu verbessern.

2 Benutzung der CREACOMP Software

CREACOMP besteht aus 16 großteils voneinander unabhängigen Lerneinheiten und ist als Standard-AddOn Paket für das bekannte Computeralgebra-System Mathematica konzipiert. Eine funktionsfähige Installation von Mathematica 5.2 (oder höher) ist als Systemvoraussetzung zu Grunde gelegt, es gibt keine weiteren Systemvoraussetzungen. Das CREACOMP-System kann entweder *direkt von der CD* gestartet werden oder der Inhalt der CD wird einmalig auf der Festplatte als Mathematica AddOn Paket *installiert* (Installationsanleitung liegt der CD bei!). Nach erfolgtem Setup kann CREACOMP wie jedes andere Mathematica-Paket durch den Standard-Befehl **Needs** in Mathematica geladen werden. Nach dem Laden von

```
Needs["CreaComp‘"]
```

startet CREACOMP mit der Standard CREACOMP-Navigation in einem eigenen Fenster. Die Navigationsstruktur und ein Autoren-Tool zum Erstellen der konkreten Navigation wurde aus MeetMATH übernommen. Die CREACOMP-Navigation besteht aus einer Java-Applikation, in der der Benutzer auf verschiedenen “Lernpfaden” durch die Lerneinheiten wandern kann. Bild 1 zeigt die Standard CREACOMP-Navigation, in der die aus MeetMATH übernommenen Kategorien “RealWorld” und “MathWorld” als Menüs zur Verfügung stehen, in Untermenüs gruppiert finden sich die jeweiligen Lerneinheiten. Per Mausklick werden die Lerneinheiten als Mathematica-Dokumente (sogenannte “Notebooks”) geöffnet und können sodann vom Lernenden bearbeitet werden.

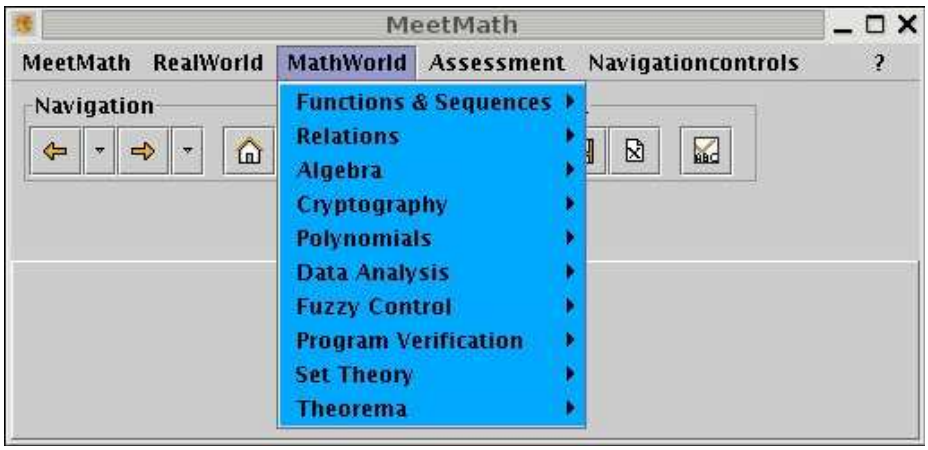


Bild 1: Die Standard CREACOMP-Navigation

Alternativ zur Standard-Navigation können die Lerneinheiten auch in einer Matrix-Struktur nach "RealWorld" und "MathWorld" gruppiert angezeigt werden. Auch aus der Matrix können die Einheiten per Mausklick geöffnet werden.

Im derzeitigen Projektstadium von CREACOMP steht im Gegensatz zum Vorgänger−projekt MeetMATH die Verknüpfung von mathematischen Inhalten ("MathWorld") mit Anwen−dungsbeispielen aus der Praxis ("RealWorld") nicht im Vordergrund. Der Fokus liegt auf der *experimentellen Aufbereitung von mathematischen Inhalten* und deren *Verknüpfung mit dem (automatisierten) Beweisen* von mathematischen Theoremen. Demzufolge sind die derzeit verfügbaren Units ausschließlich in der Kategorie "MathWorld" zu finden, dazu passende "RealWorld"-Units können später hinzukommen.

3 Kurzbeschreibung der Lerneinheiten

3.0 *Theorema*

Dies ist keine Lerneinheit im engeren Sinn, sondern hier sollen die Benutzer die Grundideen des *Theorema* Systems und dessen Benutzung kennenlernen. Die Einheit wurde von W.Windsteiger zusammengestellt.

3.1 Elementary Set Theory

In dieser Lerneinheit werden die elementaren Sprachmittel der Mengenlehre vorgestellt. Die Einheit dient vor allem als Grundlage für andere Units, in denen die Sprache der Mengenle−hre zur Formulierung neuer Theorien verwendet wird. Es geht hier nicht um einen formal exakten Aufbau der Mengenlehre als *die* Basistheorie der Mathematik, unser Hauptaugen−merk liegt auf der Einführung und Erklärung der Sprchkonstrukte, die uns die Mengenlehre zur Verfügung stellt. Die meisten dieser Konstrukte sind dem Studierenden intuitiv vertraut, wir gehen aber auch auf formale Argumentation mit Mengen ein und verwenden dazu den Set Theory Beweiser, der in *Theorema* zur Verfügung steht.

Es werden zuerst Darstellungsformen für Mengen diskutiert, die Problematik eines intuitiven Mengenbegriffs wird mit Hilfe des bekannten Russell'schen Paradoxon beleuchtet. Es wird an dieser Stelle aber kein axiomatischer Aufbau angestrebt, vielmehr werden die wichtigsten Mengenoperationen wie Vereinigung, Durchschnitt, Potenzmenge, etc. definiert, ohne dass auf deren Existenz im Detail eingegangen wird. Verschiedenste Eigenschaften, die das Zusammenspiel dieser Konstrukte beschreiben, werden mit Hilfe von *Theorema* exakt bewiesen, sodass dei Studierenden einen Eindruck des Beweisens in diesem wichti−gen mathematischen Teilgebiet vermittelt bekommen.

Die Einheit wurde von W.Windsteiger zusammengestellt.

3.2 Relations

Wir diskutieren Relationen als allgemeines mathematisches Modell für Beziehungen zwischen Objekten. Im speziellen werden binäre Relationen behandelt, d.h. Relationen, die eine Beziehung zwischen genau zwei Elementen darstellen. Zu Beginn wird erarbeitet, dass binäre Relationen stets Teilmengen Kartesischer Produkte sind. Als solches sind sie wiederum Mengen, so dass auf sie Mengenoperationen angewandt werden können, um neue Relationen zu erhalten. Weiters werden die Hintereinanderausführung von Relationen sowie deren Inverse behandelt.

Das wesentliche Augenmerk dieser Einheit liegt auf der Kombination von explorativen Elementen in Form von interaktiven Experimenten und der Formalisierung sowie des Beweises der mathematischen Zusammenhänge mit Hilfe von Theorema.

Die Einheit wurde von Susanne Saminger gemeinsam mit Bruno Buchberger und Wolfgang Windsteiger entwickelt. Die interaktiven Experimente (Widgets) wurden von Susanne Saminger entwickelt. Die mathematischen Inhalte orientieren sich an Einführungsvorlesungen für Mathematik− und Informatikstudierende an der JKU.

3.3 Equivalence Relations

Basierend auf den Erkenntnissen über Relationen werden weitere Eigenschaften von Relationen untersucht. Für eine beliebige binäre Relation auf einer Grundmenge wird als Klasse eines Elements ganz allgemein jene Menge bezeichnet, die genau die Elemente enthält, welche zu dem gegebenen Element in Relation stehen. Auf diese Art und Weise kann jeder binären Relationen eine Menge von Klassen zugeordnet werden. Ist andererseits eine Menge von (Teil)mengen der Grundmenge gegeben, so kann daraus eine binäre Relationen abgeleitet werden. Beispielsweise indem definiert wird, dass je zwei Elemente der Grundmenge zueinander in Relation stehen, wenn sie gleichzeitig in einer der gegebenen Teilmengen enthalten sind. So erhält man von einer Menge von (Teil)mengen eine binäre Relation.

Für Äquivalenzrelationen, d.h. für reflexive, symmetrische und transitive binären Relationen, bildet die induzierte Menge an Klassen eine Partition der Grundmenge, d.h. dass je zwei Klassen entweder keine gemeinsamen Elemente haben oder vollständig übereinstimmen, sowie dass die Vereinigung aller Klassen die gesamte Grundmenge liefert. Als Konsequenz davon ist jedes Element der Grundmenge in (genau) einer der Klassen enthalten. Weiters ist zentral, dass die binäre Relationen, welche von der Menge von Klassen abgeleitet werden kann, wieder identisch mit der ursprünglichen Äquivalenzrelation ist. Es gibt daher eine eindeutige Zuordung zwischen Partitionen und Äquivalenzrelationen.

In dieser Einheit wird die Menge der Klassen von binären Relationen untersucht, im speziellen wie bestimmte Eigenschaften der Relationen die Struktur der Klassen beeinflussen. Daran anschließend werden binäre Relationen induziert von einer Menge von (Teil)mengen behandelt, wieder unter Berücksichtigung des Einflusses spezieller Eigenschaften der Menge von (Teil)mengen. Abschließend werden beide Schritte zusammen− und hintereinander ausgeführt, um den spezielle Rolle von Äquivalenzrelationen und Parti−

tionen zu verdeutlichen.

Alle diese Aspekte werden mittels interaktiver Experimente und Theorema−Beweisen aufbereitet. Bei der Erstellung der einzelnen Beweise können die Studierenden, wie in anderen Einheiten, eigenständig auswählen, welche Voraussetzungen (vorangegangene Definitionen, Lemmata, Sätze, etc.) für einen erfolgreichen Beweis hinzugezogen werden sollen bzw. müssen. Auf dieser Art und Weise wird implizit ein weiteres Lernziel verfolgt − das Erarbeiten von Wissen über Beweis− und Formalisierungstechniken.

Diese Einheit wurde von Susanne Saminger entwickelt. An der Erstellung der Theorema−Teile hat Wolfgang Windsteiger mitgewirkt, zu den interaktiven Experimente (Widgets) hat Werner Groißböck Beiträge geleistet. Die mathematischen Inhalte orientieren sich an Einführungsvorlesungen für Mathematik− und Informatikstudierende an der JKU.

3.4 Factoring Integers

Das Faktorisieren von großen Zahlen ist ein zentrales Element aktueller kryptografischer Verfahren. Beispielsweise beruht die Sicherheit des RSA−Verfahrens auf der Schwierigkeit des Faktorisierungsproblems. Für sehr große Zahlen gibt es derzeit noch keinen Algorithmus, der in kurzer Zeit die Faktoren findet. Allerdings gibt es zahlreiche Algorithmen, die wesentlich schneller sind, als das simple Durchprobieren aller möglichen Faktoren. Ein solcher Algorithmus, das "Quadratische Sieb", wird hier vorgestellt und die Funktionsweise erläutert. Dieses Verfahren ist geeignet für Zahlen mittlerer Größe, für sehr große Zahlen ist auch das Quadratische Sieb ineffizient. 1994 konnte eine (spezielle) 129−stellige Zahl innerhalb von 8 Monaten mit mehreren Hundert Computern faktorisiert werden.

Das für diesen Algorithmus wichtige Lösen eines lineare Gleichungssystems über dem endlichen Körper $\mathbb{F}_2$ wird im zweiten Teil der Einheit näher beschrieben. Hier wird den Lernenden auch die Möglichkeit gegeben, mit einem interaktiven Werkzeug selbst einen Algorithmus zur Lösung eines solchen Gleichungssystems zu finden.

Die Einheit wurde von Jürgen Ecker and Günther Mayrhofer entwickelt. Der mathematische Inhalt stammt aus der Vorlesung Einführung in die Kryptografie von J. Ecker und aus den in der Einheit zitierten Artikel.

3.5 Polynomial Interpolation 1

In dieser Lerneinheit wird das mathematische Problem der Interpolation mit Polynomfunktionen vorgestellt und exakt spezifiziert. Auf Basis der Spezifikation wird ein erster einfacher Algorithmus mittels Polynomansatz gezeigt, der aber bald als ineffizient erkannt wird. Die Grundidee der Lagrange Interpolation, das Aufstellen einer geeigneten Vektorraum-Basis, wird vorgestellt, ohne den Lagrange Algorithmus gleich zu “verraten”. Mittels geeigneter interaktiver Experimente können die Studierenden die charakteristische Eigenschaft einer Lagrange-Basis experimentell hersufinden und so den Lagrange Algorithmus “selbst erfinden”.

Alle Algorithmen werden in der Sprache von *Theorema* formuliert und anhand einfacher Beispielsrechnungen sofort ausprobiert. Die Studierenden sehen hier, dass die *Theorema*

Sprache einerseits leicht lesbar ist wie eine gewohnte mathematische Formelsprache, andererseits aber auch exekutierbar ist wie eine Programmiersprache.

Die Einheit wurde von W. Windsteiger entwickelt. Der mathematische Inhalt und die Notation basieren auf W.Windsteigers Vorlesungsskriptum *Algorithmische Methoden 1*.

3.6 Polynomial Interpolation 2

In dieser Lerneinheit wird das mathematische Problem der Interpolation mit Polynomfunktionen weiterbehandelt. Die wesentlichen Nachteile des im ersten Teil erarbeiteten Verfahren von Lagrange werden experimentell erkundet und es werden rekursive Lösungsansätze erforscht. Ein erster Ansatz führt dabei auf den rekursiven Algorithmus von Neville. Mittels interaktiver Experimente wird die Effizienz der rekursiven Implementierung studiert, und es fällt auf, dass viele rekursive Aufrufe unnötigerweise mehrmals stattfinden. Dies motiviert die in der Entwicklung von mathematischen Algorithmen wesentliche Strategie der Transformation einer Rekursion in ein Programm mit einer Schleife bzw. die Speicherung von Resultaten in rekursiven Programmierumgebungen. Dies führt zum bekannten Neville Tableau. Die Korrektheit des Neville Algorithmus wird mit Hilfe von *Theorema* exakt bewiesen.

In einem nächsten Schritt wird aufbauend auf den Ideen von Neville der Newton-Algorithmus das Prinzip der dividierten Differenzen vorgestellt. Effizienzüberlegungen analog derer beim Neville Algorithmus werden experimentell unterstützt, und wir gelangen so zum bekannten Schema der dividierten Differenzen.

Alle Algorithmen werden in der Sprache von *Theorema* formuliert und anhand einfacher Beispielsrechnungen sofort ausprobiert. Die Studierenden sehen hier, dass die *Theorema* Sprache einerseits leicht lesbar ist wie eine gewohnte mathematische Formelsprache, andererseits aber auch exekutierbar ist wie eine Programmiersprache.

Die Einheit wurde von W. Windsteiger entwickelt. Der mathematische Inhalt und die Notation basieren auf W.Windsteigers Vorlesungsskriptum *Algorithmische Methoden 1*.

3.7 Real Sequences 1

In dieser Einheit werden die Eigenschaften reeller Folgen untersucht. Die Untersuchungen beginnen mit einigen einfachen speziellen Folgen, etwa *konstanten Folgen*, der *Identitätsfolge*, dem *Quadrat*, dem *Reziproken* etc. Mit Hilfe von interaktiven und grafischen Tools und der Methode des ‘‘Selbst-Explorierens’’ werden elementare Konzepte wie *Beschränktheit*, *Monotonie* und *Konvergenz* eingeführt. Es können erste Erfahrungen gesammelt werden. Erst anschließend werden die Begriffe exakt in der *Theorema* Sprache definiert. Basierend auf den zuvor erfolgten Beobachtungen werden nun Vermutungen über die speziellen Folgen formuliert. Theoreme über Beschränktheit und Konvergenz können dann mit Hilfe der *Theorema* Beweiser *automatisch bewiesen* werden. Nach dem Bearbeiten dieser Einheit sollte der Studierende die grafischen und rechnerischen Experimente mit exaktem mathematischen Beweisen in Zusammenhang bringen können.

Die Einheit wurde von Robert Vajda and Kujtim Avdiu entwickelt. Neben dem mathematischen Inhalt wurden auch grafische Werkzeuge und interaktive Experimente (Widgets)

programmiert, und sogar eine spezielle Beweismethode im Rahmen von *Theorema* wurde entwickelt. Der mathematische Inhalt und die Notation basieren auf B. Buchberger's Arbeitsheft VI. (Calculus).

3.8 Real Sequences 2

Diese Lerneinheit untersucht die Interaktionen mathematischer Operatoren mit den in der Einheit "Real Sequences I" eingeführten Eigenschaften reeller Folgen. Manche Eigenschaften sind *invariant* bzgl. bestimmter Operatoren: z.B. ist die Summe zweier beschränkter Folgen selbst wieder beschränkt, das Produkt zweier konvergenter Folgen ist selbst wieder konvergent, etc. Andere Operatoren hingegen erhalten nicht alle Eigenschaften: z.B. ist die Reziproke einer Nullfolge *keine* Nullfolge mehr. Verschiedenste interessante Interaktionen werden dann exakt formuliert und mit den *Theorema* Beweisern untersucht. Nach dem Bearbeiten dieser Einheit sollte der Studierende die meisten Eigenschafts-erhaltenden Theoreme über reelle Zahlenfolgen kennen und das Grenzwert-Rechnen zum Argumentieren über Folgen-Konvergenz beherrschen.

Die Einheit wurde von Robert Vajda and Kujtim Avdiu entwickelt. Neben dem mathematischen Inhalt wurden auch grafische Werkzeuge und interaktive Experimente (Widgets) programmiert, und sogar eine spezielle Beweismethode im Rahmen von *Theorema* wurde entwickelt. Der mathematische Inhalt und die Notation basieren auf B. Buchberger's Arbeitsheft VI. (Calculus).

3.9 Continuous Functions

In dieser Lerneinheit wird der Begriff der *Stetigkeit einer Funktion* behandelt. Es werden wieder zuerst spezielle Funktionen betrachtet, wie die *konstante Funktion*, die *identische Funktion*, die *Quadratfunktion*, die *Reziprokfunktion*, etc. Mit Hilfe von interaktiven und grafischen Elementen und der Methode des "Selbst-Explorierens" werden elementare Konzepte wie *lokale Stetigkeit*, *Stetigkeit auf Intervallen* und *gleichmäßige Stetigkeit* eingeführt. Mit diesen Begriffen kann dann experimentiert werden und die Studierenden werden mit den Begriffen intuitiv vertraut. Erst dann werden die Begriffe exakt in der *Theorema* Sprache definiert. Basierend auf den zuvor erfolgten Beobachtungen können jetzt Vermutungen über die Eigenschaften der speziellen Funktionen formuliert werden. Die zur Verfügung stehenden *Theorema* Beweiser werden dann zum automatischen Beweisen der vermuteten Sätze zum Einsatz gebracht und die vollautomatisch produzierten Beweise werden in natürlicher Sprache (Englisch) dem Benutzer präsentiert.

Die Einheit wurde von Robert Vajda and Kujtim Avdiu entwickelt. Neben dem mathematischen Inhalt wurden auch grafische Werkzeuge und interaktive Experimente (Widgets) programmiert, und sogar eine spezielle Beweismethode im Rahmen von *Theorema* wurde entwickelt. Der mathematische Inhalt und die Notation basieren auf B. Buchberger's Arbeitsheft VI. (Calculus).

3.10 Markov Chains

Anhand eines anschaulichen Beispiels wird den Lernenden die Anwendung von Markovprozessen näher gebracht. Ausgehend von bestehenden Zuständen, im Beispiel die Verteilung von Konsumenten auf verschiedene Filialen einer Supermarktkette, werden die zukünftigen Zustände berechnet. Dabei wird die Veränderung als konstant angenommen. Dieses Verhalten wird mit Hilfe von Matrizen modelliert. In interaktiven Experimenten können die Lernenden entdecken, dass es Zustände gibt, die sich nicht mehr verändern. Ein solcher Zustand heißt *Equilibrium* und wird unter bestimmten Voraussetzungen unabhängig von Ausgangszustand nach wenigen Schritten erreicht. Wie sich eine solche Konvergenz verhält, kann grafisch untersucht werden. Die interaktiven Grafiken sind so gestaltet, dass auch eigene Beispiele verwendet und veranschaulicht werden können.

Diese Einheit wurde von Günter Pilz und Günther Mayrhofer entwickelt. Zur Unterstützung der Lernenden wurden mehrere interaktive grafische Experimente einbaut. Mit Hilfe von gezielten Beispielen und Aufgabenstellungen können die Lernenden selbst Entdeckungen machen. Der Mathematische Inhalt stammt aus dem Projekt MeetMath und wurde nun für CREACOMP weiterentwickelt.

3.11 Gröbner Bases

Die Lektion "Gröbner Bases" hat das Verständnis der folgenden Aspekte zum Ziel:

- Grundlagen von univariaten und multivariaten Polynomen.
- Repräsentation von Polynomen durch Tupel.
- Multivariate Polynomreduktion.
- Berechnung von Gröbner Basen.
- Lösen multivariater Polynomsysteme mittels Gröbner Basen.

Der Leser wir schrittweise in alle notwendigen Konzepte eingeführt. Diese sind durchwegs in Theorema formalisiert. Zahlreiche Beispiele, die mittels Theorema sofort berechnet werden können, veranschaulichen die Verwendung der vorgestellten Begriffe und Algorithmen und ermöglichen eigene Experimente. Die Lektion ist in folgende Teile aufgegliedert:

- In "Introduction" wird neben einigen einleitenden Worten auch ein motivierendes Problem gestellt (Lösen eines multivariaten Polynomsystems), das am Ende der Lektion mit Hilfe von Gröbner Basen gelöst wird.
- Der kurze Abschnitt "Basic Ingredients from Ideal Theory" führt zwei wichtige Begriffe aus der Algebra ein, die im weiteren Verlauf verwendet werden.

- Der Abschnitt "Univariate Polynomials" führt den Leser in die Grundbegriffe von Polynomen in einer Variablen ein. Das meiste wird ihm wohl schon bekannt, wodurch dieses Kapitel recht schnell erarbeitet werden kann. Insbesondere wird in diesem Teil bereits eine Tupelstruktur zur Repräsentation von Polynomen vorgestellt.
- In "Multivariate Polynomials" werden die Konzepte aus dem vorhergehenden Kapitel verallgemeinert, sodass der Leser mit Polynomen in mehreren Variablen vertraut wird. Außerdem wird die Polynomreduktion behandelt.
- Im letzen Abschnitt "Gröbner Bases and Buchberger's Algorithm" wird letztendlich der Begriff der Gröbner Basen eingeführt und mit dem Buchberger Algorithmus eine Methode, diese zu berechnen. Am Ende dieses Abschnitts wird das motivierende Beispiel vom Beginn dieser Lektion gelöst und darüber hinaus gezeigt, wie Gröbner Basen mit Mathematica berechnet werden können.

In dieser Lektion wird der Leser ausgehend von für ihn bekannten Konzepten Schritt für Schritt in die Theorie der Gröbner Basen eingeführt. Natürlich können dabei nicht alle Aspekte behandelt werden. Dennoch sollte der Leser nach dem Arbeiten mit dieser Lektion mit allen wesentlichen Begriffen vertraut sein, und in der Lage sein, Gröbner Basen sowohl per Hand als auch mit einem Computeralgebrasystem zu berechnen.

Diese Einheit wurde von Alexander Zapletal entwickelt. Zur Unterstützung der Lernenden wurden mehrere interaktive grafische Experimente einbaut. Grundlage der Formalisierung in *Theorema* ist B. Buchbergers Vortrag *Groebner Bases and Automated Theorem Proving* vom 9. Oktober 2004 am Mathematical Institute der Toho University, Tokyo-Tsudanuma bzw. W. Windsteigers Vorlesungsskriptum *Algorithmische Methoden*.

3.12 Cryptography

In dieser Lerneinheit wird das RSA Kryptosystem näher beschrieben. Mit Hilfe von *Theorema* wird der Korrektheitsbeweis des Systems auf ein wichtiges Lemma aus der Zahlentheorie reduziert, das an dieser Stelle als bekannt vorausgesetzt wird. Dem vollautomatischen Beweis dieses Reduktionsschritts wird dann ein klassischer ‘‘per Hand’’-Beweis des Lemmas gegenübergestellt. Mit grafischen Werkzeugen und benutzergesteuerter Exploration werden Standard-Attacken auf das Verschlüsselungssystem (wie z.B. wiederholte Verschlüsselung) diskutiert. Algorithmische Aspekte werden in angeleiteten Übungsbeispielen besprochen, so etwa das ‘‘Quadrieren und Multiplizieren’’ zum modularen Potenzieren.

Explorative Übungen erlauben den Studierenden die Visualisierung von Verschlüsselungsfunktionen und geben ein Gefühl über die geeignete Wahl von Verschlüsselungsparametern. Das Ende bildet ein Beispiel mit Parametern aus echten Anwendungen, um den Studierenden einen Eindruck der Praktikabilität des Systems zu vermitteln.

Diese Lerneinheit kann als mathematische Motivation für das Modul ‘‘Factoring Integers’’ (siehe oben) gesehen werden. Andererseits kommen hier Methoden aus der Einheit ‘‘Fast Computations’’ (siehe nächstes Kapitel) zum Einsatz, sodass zusätzliche Motivation für diese Einheit entsteht.

Die Einheit wurde von Jürgen Ecker and Günther Mayrhofer entwickelt. Neben dem mathematischen Inhalt wurden auch grafische Werkzeuge und interaktive Experimente (Widgets) programmiert. Der mathematische Inhalt und die Notation basieren auf den Vorlesungsunterlagen “Einführung in die Kryptografie” von J. Ecker und den zitierten Artikeln.

3.13 Fast Computations

Im ersten Teil dieser Einheit wird eine Methode entwickelt, um mit großen Zahlen schnell rechnen zu können. Dabei werden die Berechnungen in mehreren kleineren Zahlenbereichen durchgeführt und anschließend zu einem Gesamtresultat zusammengefügt. Der für diese Rückführung zentrale Satz, der so genannte “Chinesische Restsatz”, wird mit Hilfe von *Theorema* bewiesen. Im zweiten Teil der Einheit wird diese Methode auf Polynome übertragen. Hier werden die Polynome zuerst an bestimmten Stellen ausgewertet, dann die gewünschten Berechnungen durchgeführt und dann mit Hilfe der Polynom−Interpolation das Lösungspolynom bestimmt. Die Polynom−Interpolation ist daher verwandt zum “Chinesische Restsatz” für ganze Zahlen. Das Verfahren für Polynome führt dann auf die so genannte “Diskrete Fouriertransformation” und bildet eine Grundlage für effiziente Berechnungen mit Polynomen, die zum Beispiel im Bereich der Gröbner Basen häufig benötigt werden.

In der gesamten Einheit werden die Verfahren an Hand von einfachen Beispielen vorgestellt. Die Beispiele sind so gewählt, dass die Lösung auch mit herkömmlichen Methoden leicht errechnet werden kann und trotzdem die Stärke der Methode verdeutlicht werden kann.

Die Einheit wurde von Günter Pilz und Günther Mayrhofer entwickelt.

3.14 Fuzzy Control

Die prinzipiellen Strukturen von Fuzzy Control werden anhand des Problems des "Invertierten Pendels" erarbeitet. Einen Stab auf einer Hand zu balancieren, ist für jedes Kind, nach ein wenig Training, eine lösbare Herausforderung. Eine analytische Modellierung des Problems führt allerdings zu einem komplexen System von partiellen Differentialgleichungen höherer Ordnung.

Warum ist das Problem für das Kind lösbar? Das Kind balanciert den Stab, ohne die exakte Position oder Geschwindigkeit des Stabes zu kennen und ohne zusätzliches Wissen über die formalen Gesetze der Mechanik. Es passt seine Bewegungen den aktuellen Rahmenbedingungen entsprechend näherungsweise so an, dass der balancierte Stab nicht von der Hand fällt. Und genau dieses Verhalten wird mit Fuzzy Control modelliert: Eingangsgrößen eines Regelungsproblems werden mit Hilfe linguistischer Ausdrücke beschrieben ("groß", "sehr hoch", "ungefähr 5", …) und die Werte der entsprechenden Ausgangsgrößen werden mittels unscharfen Inferenzmechanismen bestimmt.

Für die Modellierung der Inferenzmechanismen gibt es verschiedene theoretische Ansätze, denen unterschiedliche logischen Konzepte zu Grunde liegen. Wesentlicher Schwerpunkt aller Inferenzmechanismen liegt dabei darauf, Beziehungen, die zwischen Eingangs− und Ausgangsgrößen gelten und mittels sprachlicher Regeln wie "wenn der Druck hoch ist, dann muss die Temperatur verringert werden" ausgedrückt werden können, im Regelung−

sprozess zu modellieren. Dieser Schwerpunkt bildet auch die Stärke von Fuzzy Control und bestimmt deren Hauptanwendungsgebiet im Bereich der Industrie − Regelungsprozesse, bei denen Input−Output−Beziehungen vollständig oder auch nur teilweise als rein linguistische Regeln vorliegen bzw. in Form von linguistischen Regeln beschrieben werden können.

Im Rahmen der Einheit werden diese Grundsätze von Fuzzy Control vorgestellt und diskutiert. Die Einheit wurde von Monika Zilkova unter Mitwirkung von Erich Peter Klement, Susanne Saminger und Werner Groißböck entwickelt. Der Inhalt orientiert sich an der Spezial−Vorlesung "Fuzzy Control", wie sie für Studierende eines Masters aus Mathematik, Informatik und Mechatronik regelmäßig an der JKU angeboten werden.

3.15 Clustering

Clustering Methoden zielen darauf ab, große Datenmengen in charakteristischen Gruppen zusammen zu fassen. Im Gegensatz zu reinen Klassifizierungsmethoden, bei denen die einzelnen Gruppen von vornherein bekannt sind, werden beim Clustering auch die Kategorien automatisiert und auf bestmögliche Art und Weise bestimmt. Sie erlauben daher, zu Grunde liegende, aber vorerst versteckte Strukturen der Daten sichtbar zu machen und zu analysieren. Sie gehören daher zu den sogenannten "unbeaufsichtigen" Methoden des machinellen Lernens bzw. der Datenanalyse.

Konkret versuchen Clustering Methoden, Datengruppen so zu finden, dass der Abstand der Datenpunkte innerhalb einer Gruppe möglichst gering und gleichzeitig der Abstand zwischen verschiedenen Gruppen möglichst groß ist. Weiters sollte die Anzahl der gefunden Gruppen möglichst klein gehalten werden. In diesem Sinn ist Clustering eine Optimierungsaufgabe mit mehreren Zielvorgaben.

Vorgestellt wird der bekannteste Clustering Algorithmus, die k−means Methode, anhand dessen die prinzipiellen Schwierigkeiten von Clustering Verfahren untersucht werden. Dazu zählen die Abhängigkeit des Clustering−Ergebnisses einerseits vom zugrunde liegenden Abstandskonzept, und andererseits von Startparametern des Algorithmus sowie der Skalierung der zugrundeliegenden Daten. Weiters werden verschiedene Gütemaße vorgestellt, welche eine Möglichkeit darstellen, verschiedene Clustering−Methoden und deren Ergebnisse zu vergleichen.

Die Einheit wurde von Werner Groißböck gemeinsam mit Susanne Saminger entwickelt. Die Erarbeitung der zuvor angeführten Aspekte von Clustering Methoden erfolgt mittels interaktiver Experimente, welche mit eigenen Datensätze sowie mit bereits vordefinierten Datensätzen durchgeführt werden können. Der theoretisch Inhalt orientiert sich an Spezial−Vorlesungen aus den Bereichen Bildverarbeitung bzw. Datenverarbeitung, wie sie für Studierende eines Masters aus Mathematik, Informatik und Mechatronik regelmäßig an der JKU angeboten werden.

3.16 Imperative Program Verification

Computerprogramme durchdringen unser Leben mehr und mehr, ihr korrektes Verhalten gewinnt immer mehr an Bedeutung. Die Entwicklung von Methoden, die die Korrektheit von Programmen garantieren, stellen eine Herausforderung für die Informatik dar. *Programm-Verifikation* ist ein solches Werkzeug.

Die Lerneinheit über *imperative Programmverifikation* hat die formale Verifikation imperativer Programme zum Inhalt. Dazu wird die "weakest precondition strategy" basierend auf Hoare Logik verwendet. Diese Methode wird dann auf konkrete Programme angewendet, um deren *Verifikationsbedingungen* zu ermitteln. Die Korrektheit des Programms ist gegeben, wenn diese Verifikationsbedingungen bewiesen werden können. Alternativ dazu wird auch auf eine andere Methode, die "strongest postcondition strategy", kurz eingegangen. Abschließend wird die Kombination von algebraischen Techniken zum Auffinden von Schleifeninvarianten mit Techniken des autoomatischen Beweisens gezeigt, die dazu verwendet werden kann, die Korrektheit von *Programmen mit Schleifen* vollautomatisch zu beweisen. Die gesamte Lerneinheit baut auf im Rahmen von *Theorema* entwickelten Methoden auf.

Das Ziel diese Moduls ist es, den Studierenden zu zeigen, wie die Korrektheit von Computerprogrammen bewiesen werden kann. Die "Message" soll sein, dass bei vorhandenem Tool-Support formale Methoden auch für Programmier-Anfänger zugänglich sind. Außerdem soll gezeigt werden, dass *Testen alleine* nicht ausreicht, um die Korrektheit von Computerprogrammen sicherzustellen.

Diese Lerneinheit wurde von Laura Kovacs und Loredana Tec nach einem Konzept von B. Buchberger entwickelt.

4 Arbeitsweise

Zu verschiedenen mathematischen Themenstellungen wurden CREACOMP−Lerneinheiten erstellt. Die einzelnen Lerneinheiten sind thematisch als "stand−alone units" konzipiert, die in verschiedenste bestehende Lehrveranstaltungen eingebaut werden können.

CREACOMP−Einheiten sind in Form von Mathematica Notebook−Dokumenten realisiert und folgen im didaktischen Aufbau den aus MeetMATH bekannten Richtlinien, wonach Lernphasen der *Motivation*, der *Aneignung* und der *Vertiefung* einander abwechseln.

Neu ist, dass die zur Diskussion stehenden mathematischen Begriffe, Sätze und Methoden mit THEOREMA weiter vertieft werden. Die in den Experimenten gewonnenen Vermutungen werden in mathematischer Sprache formalisiert, und es wird dann ein computer−unterstützter Beweis begonnen. Im Fall, dass der Beweis nicht gelingt (weil beispielsweise die Vermutungen falsch sind oder das vorausgesetzte Wissen nicht ausreichend ist), spannt sich der Bogen weiter und die Lernenden werden angeregt, ihre Vermutungen entsprechend zu modifizieren oder zusätzliches Wissen als Voraussetzung einzubringen. Diese Phase des Lernprozesses scheint besonders interessant, dient sie doch auf der Ebene des vermittelten "mathematischen Stoffs" der *Vertiefung* des Gelernten, *gleichzeitig* aber auf einer höheren Ebene des mathematischen Explorationsprozesses der *Aneignung* der mathematischen

Arbeitsweise. Die Unterstützung dieser Lernphase durch den Computer ist nur in Systemen wie THEOREMA möglich, die sowohl die gewohnte mathematische Formelsprache unterstützen als auch das Generieren von Beweisen in einer für den Menschen verständlichen Form erlauben.

Gearbeitet wurde in kleinen Gruppen nach und vor ausführlichen und regelmäßigen Koordinationsgesprächen (inkl. eines ganztägigen Workshops am 10.8.2004). Dadurch wurden die Lerneinheiten zusammengestellt, implementiert und laufend verbessert sowie mit didaktischen Elementen angereichert.

Alle Lehreinheiten beginnen mit einer Liste der Ziele dieser Einheit und den notwendigen Voraussetzungen und der erforderlichen Mathematica- bzw. THEOREMA-Befehle. Die Struktur des Textes und der Zellen, die Verwendung von Farben und Formaten wurde einheitlich gestaltet. Für Autoren wurde ein Leitfaden zur einheitlichen Gestaltung von Lerneinheiten erstellt.

Als Kooperationsplattform wurde am RISC ein *SVN−Server* eingerichtet, der für alle Projektmitarbeiter mit entsprechender Client-Software zugänglich ist. Diese Architektur erleichtert bzw. ermöglicht überhaupt erst eine gemeinsame Bearbeitung der Lerneinheiten und der interaktiven Elemente (den sogenannten Widgets) bei physikalisch voneinander getrennten Arbeitsplätzen der Projektmitarbeiter. Durch SVN war zu jeder Zeit sichergestellt, dass alle Mitarbeiter zu jeder Zeit Zugriff auf die aktuellste Version aller Units und Widgets haben, wodurch Parallelentwicklungen und die dadurch notwendige Synchronisation entfallen. SVN stellte sich darüberhinaus als sehr nützlich beim gemeinsamen Arbeiten an wissenschaftlichen Publikationen über das CREACOMP Projekt (siehe unten) und beim Erstellen dieses Endberichts heraus.

Weiters wurde im WS 2006/07 eine Lehrveranstaltung initiiert.

Das **Konsortium** bestand aus:

- *Buchberger* Bruno (Leiter)
- *Klement* Erich Peter
- *Pilz* Günter.

Die **Mitarbeiter** am Projekt waren (alphabetisch gereiht, ohne Titel):

- *Avdiu* Kujtim
- *Buchberger* Bruno
- *Ecker* Jürgen
- *Groißböck* Werner
- *Klement* Erich Peter
- *Kovacs* Laura

- *Mayrhofer* Günther
- *Pilz* Günter
- *Saminger* Susanne
- *Süss* Patricia
- *Tec* Loredana
- *Vajda* Robert
- *Windsteiger* Wolfgang
- *Zapletal* Alexander
- *Zilkova* Monika.

5 Dissemination

Das System CREACOMP soll in den Lehrveranstaltungen der beteiligten Institute zum Einsatz kommen, die gewonnenen Erfahrungen dokumentiert und für die weitere Verbesserung des Systems zur Verfügung gestellt werden.

Im WS2006/07 wird eine Lehrveranstaltung angeboten, in deren Rahmen die vorliegenden CREACOMP−Lerneinheiten eingesetzt und ihre didaktischen Möglichkeiten diskutiert werden. Dabei werden Studierende der Mathematik, der Informatik und des Lehramtsstudiums Mathematik unter Anleitung mit den CREACOMP Einheiten arbeiten und ihre Lernerfahrungen dokumentieren. Gleichzeitig werden die Projektpartner ihre Lehrerfahrungen anhand dieser Lehraktivitäten beurteilen. Hauptziel der Untersuchungen wird sein zu testen,

- ob die in den interaktiven CREACOMP-Widgets realisierten Interaktionen intuitiv genug sind, um im Studierenden die geplanten Lernprozesse auszulösen,
- ob die Benutzerführung im interaktiven Teil die Studierenden dazu animiert, die von den Autoren gewollten Experimente durchzuführen oder ob die Studierenden eigene Pfade erkunden, die zu anderen oder auch keinen Einsichten führen,
- ob die oben beschriebene experimentelle Herangehensweise an das mathematische Beweisen von den Mathematik−Studierenden angenommen wird und zu einem tieferen Verständnis der Materie führt, vor allem zu einem Gefühl dafür, wie mathematische Theorien entstehen,
- ob das Einbeziehen des Beweis−Aspekts für Studierende andere Fächer, in denen der Anwendungsaspekt der Mathematik im Vordergrund steht, zum besseren Verständnis der hinter den mathematischen Resultaten stehenden Intuitionen beiträgt.

Weitere Diplomarbeiten und Dissertationen zur Weiterentwicklung von CREACOMP sind geplant.

Durch aktive Teilnahme an internationalen Tagungen werden unsere Ergebnisse internationalbekannt gemacht. Umgekehrt wird dadurch das Know−how anderer Arbeitsgruppen in die Weiterentwicklung von CREACOMP einfließen. Die Resultate des CREACOMP-Projekts wurden bisher an folgenden Stellen publiziert:

- B. Buchberger: *Using Theorema in Mathematics Education*. Invited keynote talk at the *Learning Technology and Mathematics Middle East Conference* at Sultan Qaboos University, Muscat, Sultanate of Oman, March 31−2 April, 2007.
- W. Windsteiger. *The CreaComp Project: Theorema for Computer−supported Teaching and Learning of Mathematics*. November 14, 2005. Contributed talk at Theorema−Ultra−Omega'05 Workshop, Saarbrücken, Germany.
- W. Windsteiger. *CreaComp: Neue Möglichkeiten im e−learning für Mathematik*. 22. April, 2005. Invited colloquium talk at Research Net Upper Austria: Brennpunkt Forschung.
- R. Vajda. *E−training of Formal Mathematics: Report on the CreaComp Project at the University of Linz*. 26. Juni, 2006. Contributed talk at ACA-2006 (Applications of Computer-Algebra Conference), Varna, Bulgaria.
- Eine Kurzfassung der Arbeit *CreaComp: Computer−Supported Experiments and Automated Proving in Learning and Teaching Mathematics* ist zur Publikation und Präsentation bei der *8th International Conference on Technology in Mathematics Teaching (ICTMT)* akzeptiert, die Vollversion ist als Beitrag in einem Buch zum *SCE 2006 — International Seminar on Symbolic Computation in Education, Beihang University, Beijing, China, April 12−14, 2006* vorgesehen.
- Ein Paper *Using a Computer−Algebra System and a Theorem Prover to Stimulate Creativity in Learning Mathematics* wurde verfasst.
- Eine weitere wissenschaftliche Arbeit wird für eine internationale Fachzeitschrift gerade vorbereitet und wird nach Fertigstellung nachgereicht.

6 Finanzielle Abrechnung

Die finanzielle Abwicklung des Projekts "CREACOMP: e−Schulung von Kreativität und Problemlösungskompetenz" erfolgte über die Finanzbuchhaltung der JKU über einen Innenauftrag, bei dem die 3 Mitglieder des Konsortiums zeichnungsberechtigt sind.

7 Die CREACOMP CD

- **Installieren** Sie *Mathematica* 5.2 (or later) auf Ihrem Computer.
- **Legen Sie die CD** in das CD-ROM-Laufwerk Ihres Computers ein.
- **Öffnen** Sie die Mathematica Datei “Startup.nb” im Hauptverzeichnis der CD.

Commented Unit: Equivalence Relations

After finishing this module you should

** know the definition of an equivalence relation,*

** know that equivalence classes form a partition of the underlying universe,*

** know that equivalence relations can be induced by partitions.*

Additional Modules

MathWorld : ***Relations***

In the present module we will make use of the following Theorema−commands:

- Definition, Theorem, Proposition, Lemma.

At the beginning of the unit we provide a list of goals to be achieved within this unit as well as a hyperlink to a related unit.

Introduction

The introduction gives a short overview on the contents as well as the expected prerequisites.

The present module focusses on equivalence relations and their relationship to partitions of some universe A. We will investigate the concept of classes of binary relations and their particular structure in case that the binary relation fulfills some additional properties. We will further have a look at the introduction of binary relations on the basis of a collection of sets and investigate for which sets of sets we end up again with an equivalence relation.

In order to profit best from the contents of this module you should be acquainted to the basics of binary relations and their properties as well as of the basics of set theory.

Equivalence relations - basic definitions

Since we assume that the student is acquainted with the basics of set theory and of relations, we immediately start with the necessary definitions for equivalence relations. Afterwards we provide an interactive tool by which the student can recall and explore the basic definitions as well as its influ−ences on the structure of binary relations

An equivalence relation R is a binary relation on some universe A which is additionally reflexive, symmetric and transitive. We recall its definitions by the following Theorema commands.

Definition["relation", any[R, A],
$\text{is-relation}_A[R] :\Leftrightarrow (R \subseteq A \times A)$]

Definition["reflexivity", any[A, R],
$\text{is-reflexive}_A[R] :\Leftrightarrow \forall_{x \in A} \langle x, x\rangle \in R$]

Definition["symmetry", any[A, R],
$\text{is-symmetric}_A[R] :\Leftrightarrow \forall_{x,y \in A} (\langle x, y\rangle \in R \Rightarrow \langle y, x\rangle \in R)$]

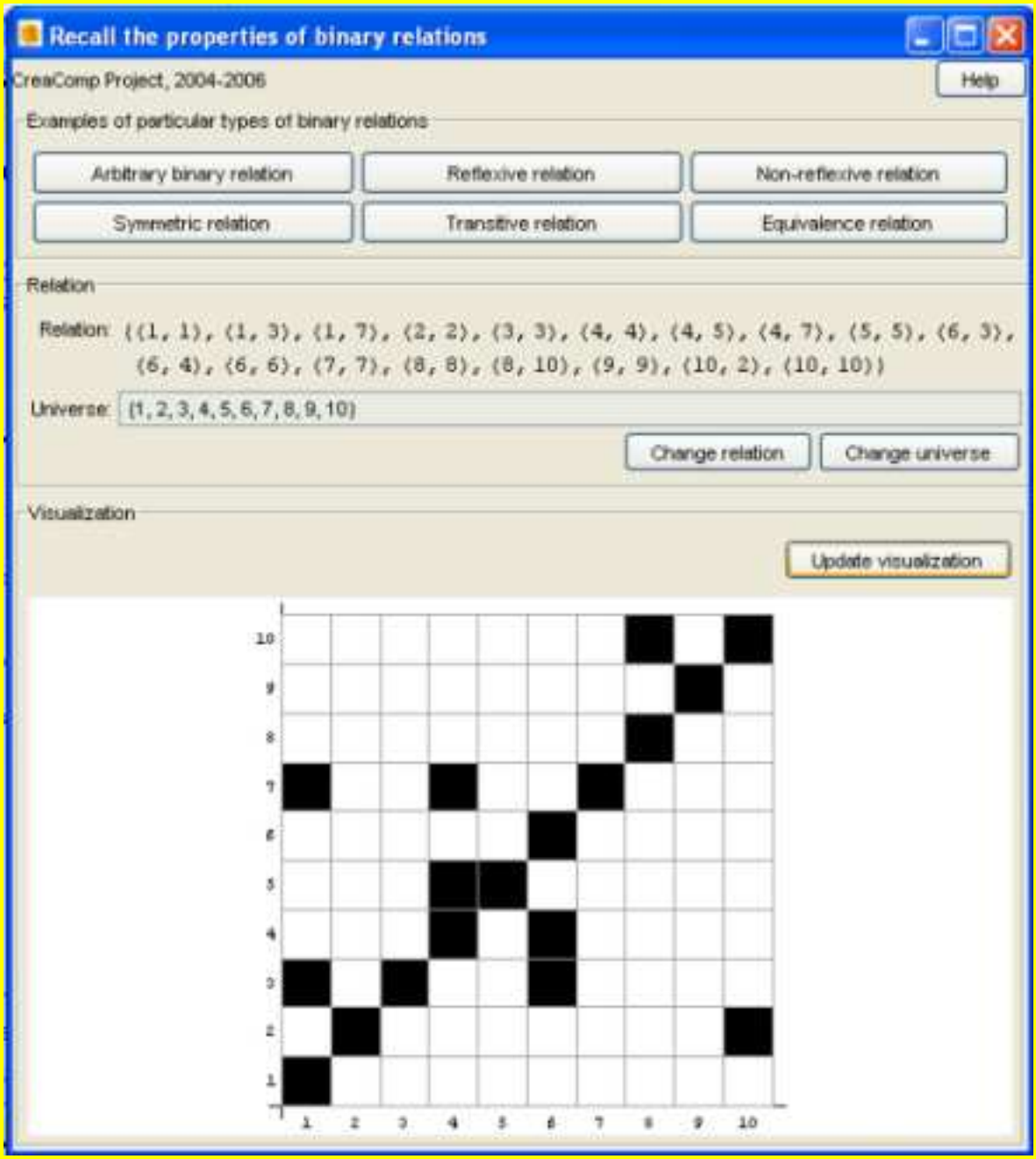
Definition["transitivity", any[A, R],
$\text{is-transitive}_A[R] :\Leftrightarrow \forall_{x,y,z \in A} (\langle x, y\rangle \in R \wedge \langle y, z\rangle \in R \Rightarrow \langle x, z\rangle \in R)$]

Definition["equivalence", any[A, R],
$\text{is-equivalence}_A[R] :\Leftrightarrow \bigwedge \begin{Bmatrix} \text{is-relation}_A[R] \\ \text{is-reflexive}_A[R] \\ \text{is-symmetric}_A[R] \\ \text{is-transitive}_A[R] \end{Bmatrix}$]

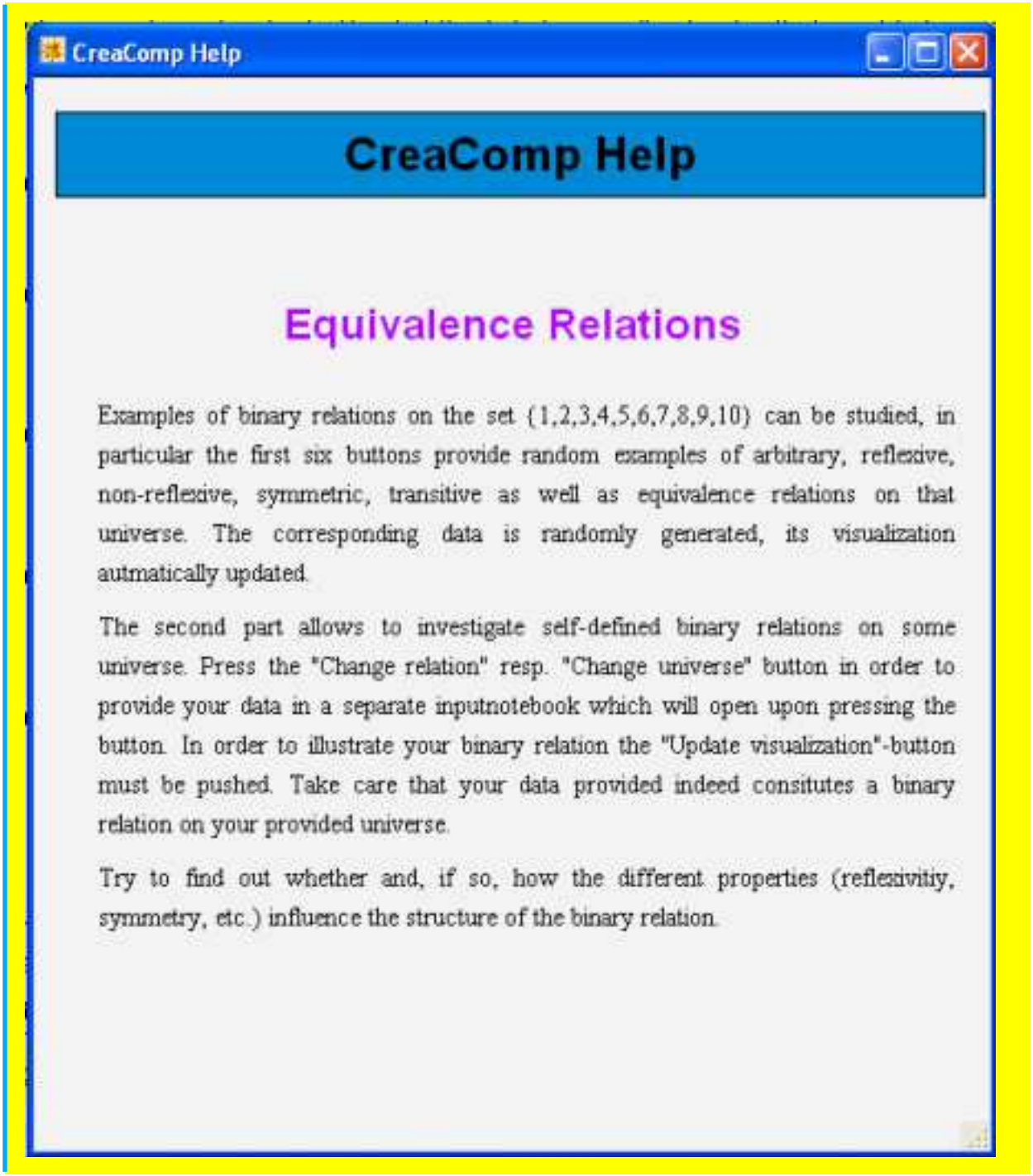
The following link provides illustrations of some examples of binary relations on the set {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}. You can use this tool to illustrate also other binary relations by providing the corre−sponding list of pairs and pressing the "update visualization"−button.

Visualize binary relations

Clicking the previous button will open the following widget.



The buttons in the first part allow to generate randomly generate relations of the described type, the visualization part will be automatically updated and displays by black squares all pairs representing element which are related to each other. The second part of the widget provides the student the opportunity to visualize own binary relations. The data must be inserted throught the interfaces provided when clicking on the corresponding buttons. Since by this elements contained in the relation resp. of the correpsonding universe it is necessary to push the "update visualization"−button after both entries have been changed accordingly. If the data provided does not match the demanded criteria an error-message will be displayed. When clicking on the help-button the following Mathemat−ica notebook will open.



If you would like to explore the concept of relations in more detail, we recommend to have a look at the module on **relations**. In this module we will focus next on the concept of classes of binary relations.

The concept of classes and their properties

When introducing relations we have stressed that they allow to model relationships between objects. On the other hand, if we consider some binary relation R on some universe A we can associate with each element $x \in A$ the set $\text{class}_{R,A}[x]$ of all objects that are related to x by relation R, i.e., $\text{class}_{R,A}[x] = \{a \in A \mid \langle a, x\rangle \in R\}$. We will refer to x as the *representative* of the class $\text{class}_{R,A}[x]$.

Definition["class", any[x, A, R],
$\text{class}_{R,A}[x] := \{a \in A \mid \langle a, x\rangle \in R\}$]

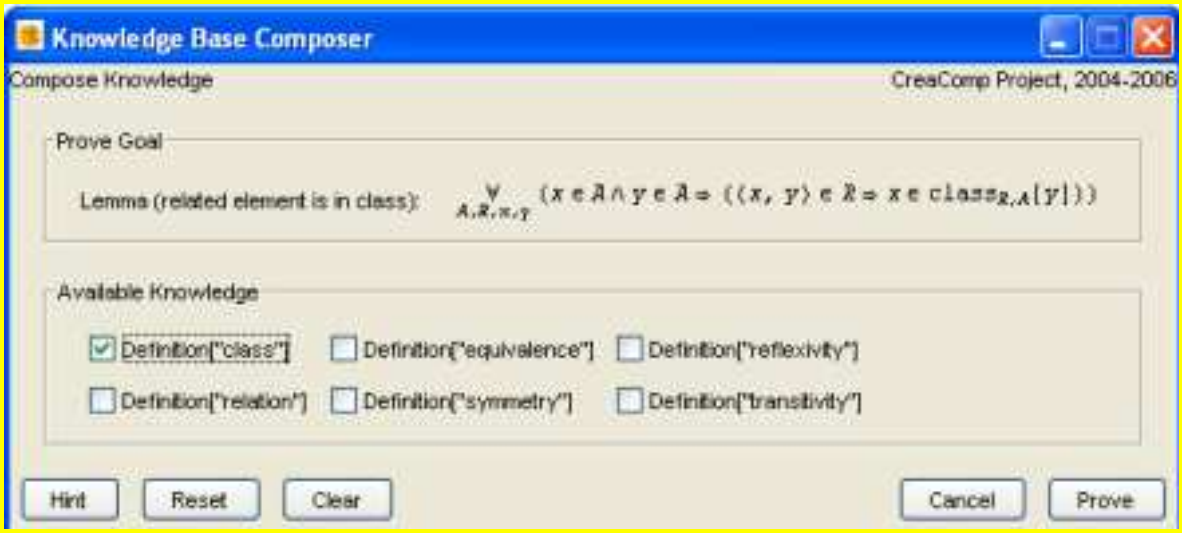
We introduce the concept of a class of a binary relation and immediately draw first basic conclusions based on the definition. The student is asked to prove the provided lemmata himself. The contents are very basic, so the proofs will not be diffifult to find, however, like the student already gets acquainted with the button-bar for theorema-proofs as it will be used later also for proving more complex properties.

As a consequence of the definition of a class we can immediately formulate the following Lemma.

Lemma["related element is in class", any[A, R; x, y ∈ A],
$\langle x, y\rangle \in R \Rightarrow x \in \text{class}_{R,A}[y]$]

Prove it! **Abort** **Show** **Off-line**

The "Abort"−button allows to interrupt a proof generation in case that it might be too time consuming or, due to the selected knowledge-based, not successful. By "Show" the proof attempt can be visual−ized. "Off-line" provides a pre-generated Theorema-proof of the result. The standard behaviour will be to push the "Prove it!"−button what will lead the student to the following input mask:



The student is asked to select from a list of available knowledge parts those which might be relevant for the proof of the displayed theorem. By pushing the "Prove"−button the proof-search in Theorema will be initiated. By pressing "Cancel" the window will close, no proof-search is started. By "Hint" the student can ask for advice, which selection might lead to a successful proof. In case of the present example the following proof will be generated:

Begin of the automatically generated proof by theorema

Prove:

(Lemma (related element is in class)) $\forall_{A,R,x,y} (x \in A \wedge y \in A \Rightarrow (\langle x, y\rangle \in R \Rightarrow x \in \text{class}_{R,A}[y]))$,

under the assumption:

(Definition (class)) $\forall_{A,R,x} \left(\text{class}_{R,A}[x] := \left\{a \mid_{a} a \in A \wedge \langle a, x\rangle \in R\right\}\right)$.

We assume

(1) $x_0 \in A_0 \wedge y_0 \in A_0$,

and show

(2) $\langle x_0, y_0\rangle \in R_0 \Rightarrow x_0 \in \text{class}_{R_0,A_0}[y_0]$.

We prove (2) by the deduction rule.

We assume

(3) $\langle x_0, y_0\rangle \in R_0$

and show

(4) $x_0 \in \text{class}_{R_0,A_0}[y_0]$.

Formula (4), using (Definition (class)), is implied by:

(5) $x_0 \in \left\{a \mid_{a} a \in A_0 \wedge \langle a, y_0\rangle \in R_0\right\}$.

In order to prove (5) we have to show:

(6) $x_0 \in A_0 \wedge \langle x_0, y_0\rangle \in R_0$.

We prove the individual conjunctive parts of (6):

Proof of (6.1) $x_0 \in A_0$:

Formula (6.1) is true because it is identical to (1.1).

Proof of (6.2) $\langle x_0, y_0\rangle \in R_0$:

Formula (6.2) is true because it is identical to (3).

□

End of automatically generated proof by Theorema.

Lemma["any class is subset", any[x, A, R],
$\text{class}_{R,A}[x] \subseteq A$]

Prove it! **Abort** **Show** **Off-line**

Based on these basic experiments the student shall execute another interactive experiment (widget) to explore the properties of classes. The following text describes how the widget shall be used. Similar information can also be found in the help-file corresponding to the widget which can be opened by pressing the corresponding button.

The basic idea of this experiment is two-fold: On the one hand, the student shall try to find out how different sets of classes might look like. Are elements always included in classes? Might it be that

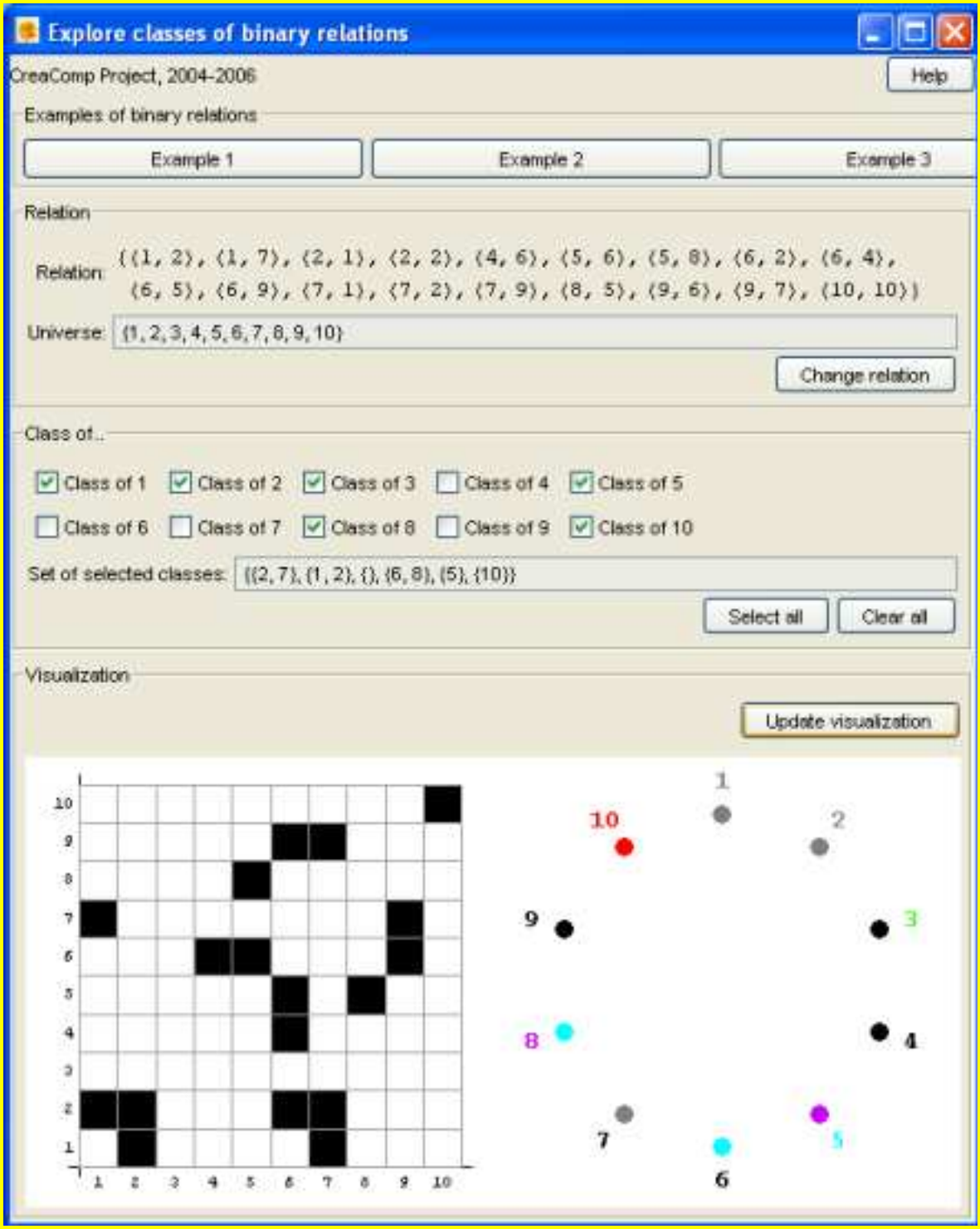
some classes coincide? That they are empty sets? On the other hand the students should get acquainted with the visualization tool which will be used later again for investigating the classes of particular types of binary relations.

Use the following visualization tool to have a look at randomly generated examples as well as self-defined binary relations and illustrate several of its classes. You might provide the binary relation as a list of tuples as well a list of all those elements whose classes shall be illustrated. To each representative will be assigned its own color by which its label will be colored. The elements of the corresponding classes will be displayed as accordingly colored points. In case that two classes are different and overlap as well, the color of the class elements and its representatives is set to grey. For the time being we recommend to illustrate classes of arbitrary binary relations, not necessarily equivalence relations.

Try to get acquainted with the tool visualizing binary relations and particular classes. Example 1 and 2 pro−vide randomly generated binary relations. If you would like to illustrate different classes of these randomly generated relations change the parameters and use the "update visualization"−button by which just the illustration is adopted but not the underlying binary relation.

Visualize classes of binary relations

By clicking the button the following experiment will open:



One sees immediately that the class of element 1 and 2 have common joint elements, but do not coincide. As such their labels as well as class elements are displayed in grey color. Differently the classes of element 8 and 5: None of their classes overlaps with one of the other selected classes. As such the labels of the representatives are colored and the class elements are visualized as correspon−digly colored points. In case of element 10, the representative is also the only element in the class as such both are displayed in the same color.

Note that the displayed relation is of arbitary type. The student can select between different randomly generated relations by pressing one of the buttons on top of the experiment.

Based on these experiences we will turn to particular binary relations and investigate whether and if so how properties of relations might influence the classes itself resp. the relationships among them. Our aim is to derive conjectures about mathematical properties which can subsequently be proven with Theorema. Since, bear in mind that, these experiments provide some insight into structures and properties of binary relations and its classes, however, they can only demonstrate a finite number of particular cases and do not prove mathematical properties valid for arbitrary binary relations and their classes in general. Let us now turn to the property of reflexivity and later on to symmetry and transitivity.

After investigation classes of arbitrary binary relations we turn next to particular subclasses of relations, namely reflexive relations resp. symmetric and transitive relations. The students shall explore how the property of the relation influences the structure of the set of sets. Based on the experiments he shall be able to formulate conjectures about the relationsships which will be later be proven with Theorema. As such each of the following sections contains a part for the experiments followed by general properties proven with Mathematica.

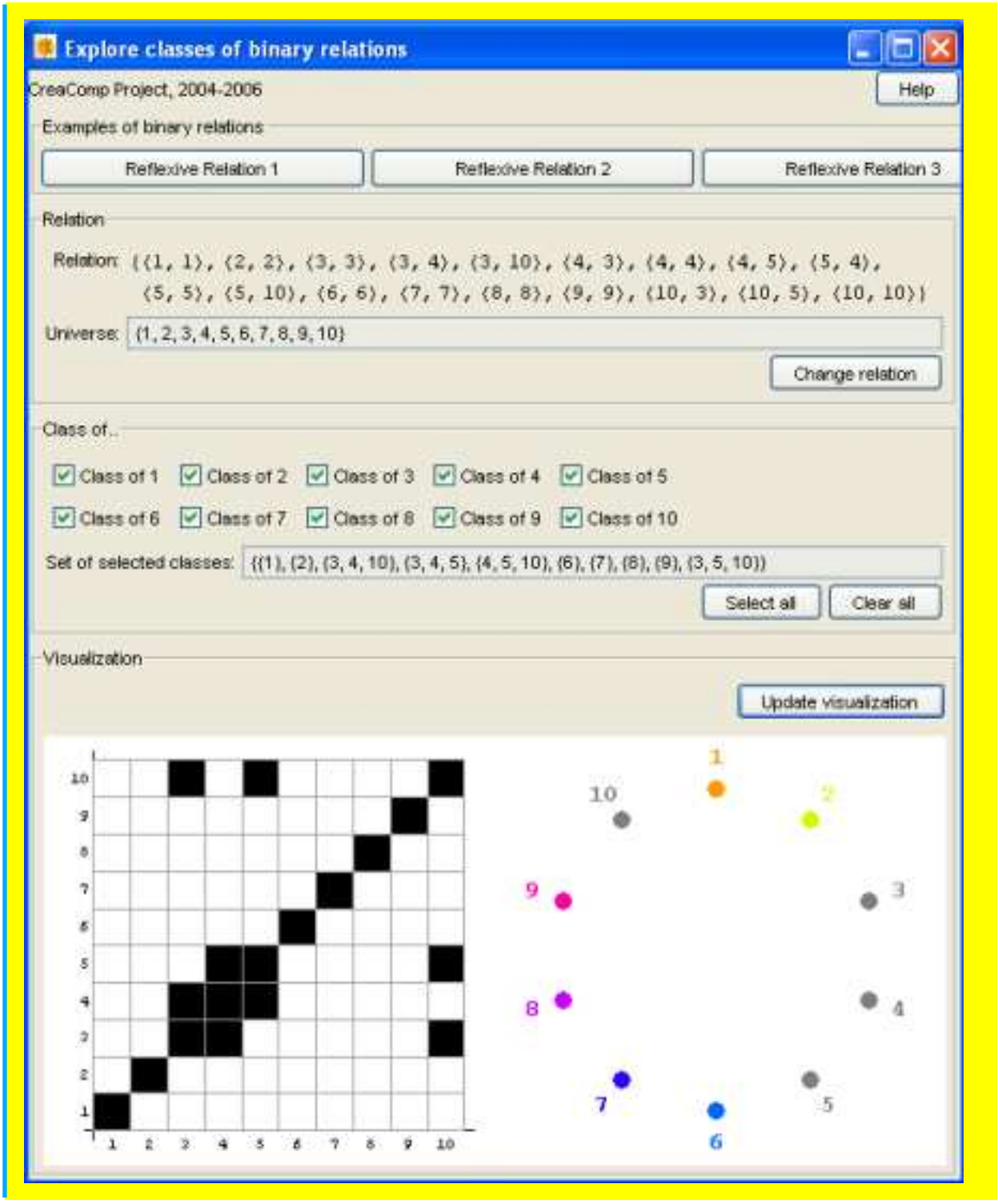
In order to keep these comments as short as possible, we will just provide screenshots of the experiments, which resemble the experiment described just above. Moreover, we will provide just one Theorema proof for each of the subsections. The button bar interfaces are, up to the increased knowledge-based in the knowledge-base selector, the same as previsouly described.

Reflexivity

The following illustration tools provides interfaces for reflexive binary (but not necessarily equivalence) relations. Again you can modify the binary relation, the underlying universe as well as the classes intended for demonstration. Take care that the relation illustrated and investigated is indeed reflexive.

Try to find out, e.g., whether each element is at least once element of a class and whether there exists a relationsship between single representatives and their classes.

Visualize classes of reflexive binary relations



Conjectures

Based on these experiments we can formulate the following properties of arbitrary reflexive binary relations and prove it with Theorema.

- Every element is contained in its class.

Proposition["in own class", any[A, R], $\forall_{x \in A}\ x \in \text{class}_{R,A}[x]$]

Prove it! Abort Show Off-line

Beginn of Theorema proof attempt

Prove:

(Proposition (in own class)) $\forall_{A,R,x}\ (x \in A \Rightarrow x \in \text{class}_{R,A}[x])$,

under the assumptions:

(Definition (reflexivity)) $\forall_{A,R}\left(\text{is-reflexive}_A[R] :\Leftrightarrow \forall_x\ (x \in A \Rightarrow \langle x, x\rangle \in R)\right)$,

(Definition (class)) $\forall_{A,R,x}\left(\text{class}_{R,A}[x] := \left\{a \mid_a a \in A \wedge \langle a, x\rangle \in R\right\}\right)$.

Alternative proof 1: failed

We assume

(1) $x_0 \in A_0$,

and show

(2) $x_0 \in \text{class}_{R_0,A_0}[x_0]$.

Formula (2), using (Definition (class)), is implied by:

(3) $x_0 \in \left\{a \mid_a a \in A_0 \wedge \langle a, x_0\rangle \in R_0\right\}$.

In order to prove (3) we have to show:

(4) $x_0 \in A_0 \wedge \langle x_0, x_0\rangle \in R_0$.

Alternative proof 1: failed

Not all the conjunctive parts of (4)can be proved.

Proof of (4.1) $x_0 \in A_0$:

Formula (4.1) is true because it is identical to (1).

Proof of (4.2) $\langle x_0, x_0\rangle \in R_0$:

The proof of (4.2)fails. (The prover "QR" was unable to transform the proof situation.)

Alternative proof 2: failed

The proof of (4)fails. (The prover "QR" was unable to transform the proof situation.)

Alternative proof 2: failed

The proof of (Proposition (in own class))fails. (The prover "QR" was unable to transform the proof situation.)

□

End of Theorema proof attempt

By inspecting the proof attempt the student realizes that the prover is not able to resolve $\langle x_0, x_0\rangle \in R$. Therefore, for a successful proof generation, further knowledge/prerequisites are needed. In the following part, the student shall try to modify accordingly. Afterwards we provide the correct solution.

Note that the proof does not succeed! Is the proposition really true as it is stated?

Why does the proof fail? What property of R is necessary to succeed with the proof?

Reformulate the proposition and *specify an appropriate property* using the with[□] field in the proposition.

Proposition["in own class", any[A, R], with[□],
$\forall_{x \in A} \; x \in \text{class}_{R,A}[x]$]

Prove it! **Abort** **Show**

Proposition["in own class", any[A, R], with[is-reflexive$_A$[R]],
$\forall_{x \in A} \; x \in \text{class}_{R,A}[x]$]

Prove it! **Abort** **Show** **Off-line**

It could further be seen from the experiments that for reflexive relations no class is empty, it contains as proven before at least its representative.

- All classes are non−empty.

Proposition["non−empty class", any[A, R], with[is-reflexive$_A$[R]],
$\forall_{x \in A} \; \text{class}_{R,A}[x] \neq \emptyset$]

Prove it! **Abort** **Show** **Off-line**

It could further be seen from the experiments that for reflexive relations it never happens that an element is not contained in any of the classes. There are different ways how to prove this result. As such the student is encouraged to try different variants.

- The union of all classes equals the underlying universe.

Proposition["union of all classes", any[A, R], with[is-reflexive$_A$[R]],
$\bigcup_{x \in A} \text{class}_{R,A}[x] = A$]

Prove it! **Abort** **Show** **Off-line**

Inspect the second part of the proof, in particular the proof of $x1_0 \in \text{class}_{R_0,A_0}[x1_0]$, and try to make this part simpler: Try to use more knowledge when specifying the knowledge base. Use knowledge that we have already proven!

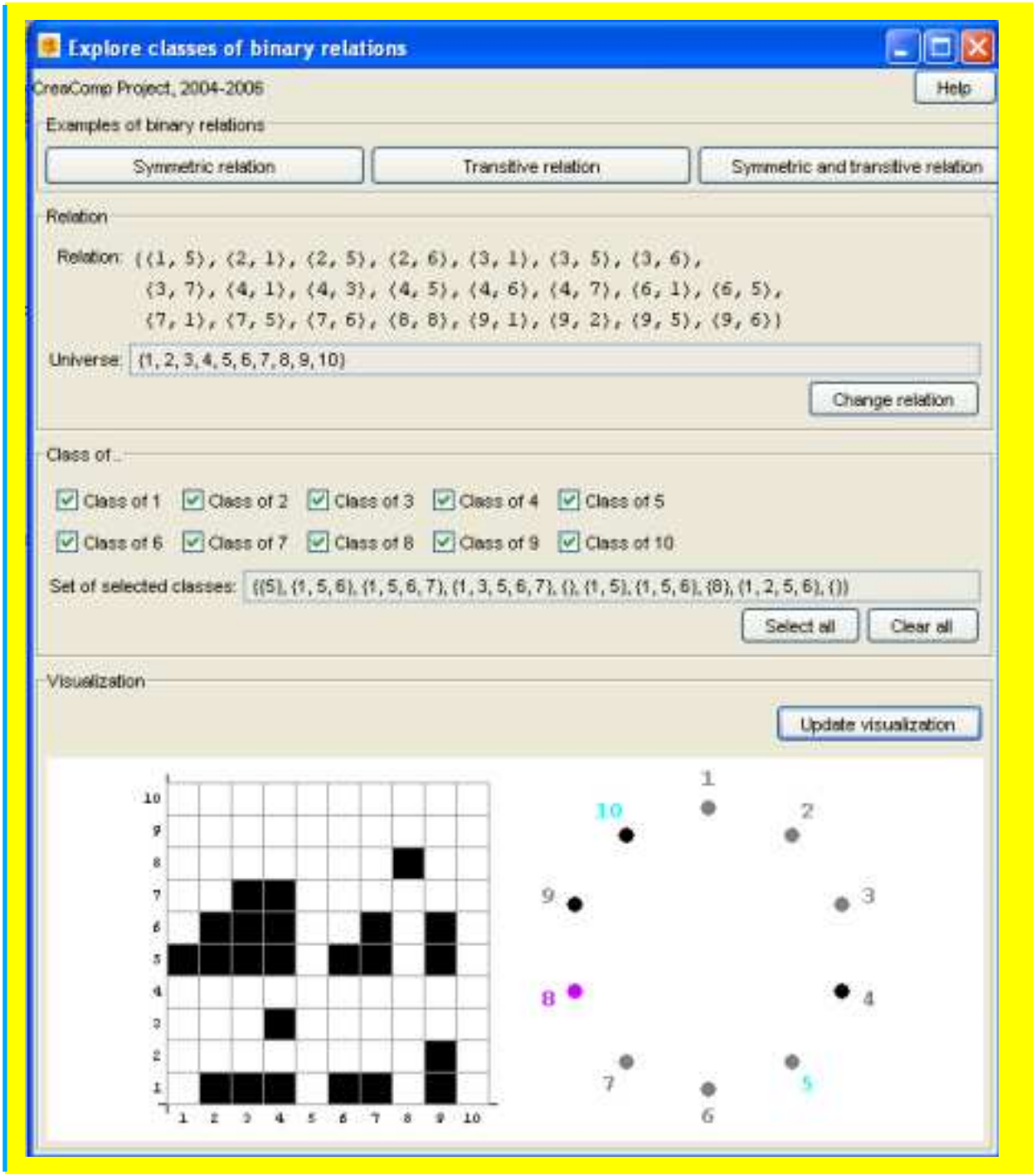
Prove it! **Abort** **Show** **Off-line**

Symmetry and transitivity.

The following illustration tools provides interfaces for symmetric and/or transitive binary (but not necessarily equivalence) relations. Again you can modify the binary relation as well as the classes intended for demonstra−tion. Take again care that the binary relations defined by yourself are indeed symmetric resp. transitive.

Try to find out, e.g., whether some classes are subsets of other classes or whether there are relationsships between elements and representatives of different classes.

Visualize classes of symmetric and transitive binary relations



Conjectures

Due to the symmetry, it holds that whenever y is in the class of x, then x is vice versa contained in the class of y.

- Symmetrie: $\forall\ x, y \in A : y \in \text{class}_{R,A}[x] \Rightarrow x \in \text{class}_{R,A}[y]$

Proposition["symmetric classes", any[A, R; x, y ∈ A], with[is-symmetric$_A$[R]], y ∈ class$_{R,A}$[x] ⇒ x ∈ class$_{R,A}$[y]]

Prove it! Abort Show Off-line

If one looks at the previous screenshot, which displays the classes of a transitive, but not symmetric relation, one sees that whenever the element 1 occurs among a class then also the element 5 is contained. This is due to the following theorem: If for some x it holds that $1 \in class_{R,A}[x]$ then it follows that $class_{R,A}[1] = \{5\} \subseteq class_{R,A}[x]$. By the experiments such relationships can just be tested for a (finite) number of examples. What follows is the proof that indeed it holds for any arbitrary transitive relation.

- Transitivity: $\forall\ x, y \in A : y \in \text{class}_{R,A}[x] \Rightarrow \text{class}_{R,A}[y] \subseteq \text{class}_{R,A}[x]$

Proposition["subclasses", any[A, R; x, y ∈ A], with[is-transitive$_A$[R]], y ∈ class$_{R,A}$[x] ⇒ class$_{R,A}$[y] ⊆ class$_{R,A}$[x]]

Prove it! Abort Show Off-line

Beginn of Theorema proof

Prove:

(Proposition (subclasses))

$\underset{A,R,x,y}{\forall}\ (x \in A \wedge y \in A \wedge \text{is-transitive}_A[R] \Rightarrow (y \in \text{class}_{R,A}[x] \Rightarrow \text{class}_{R,A}[y] \subseteq \text{class}_{R,A}[x]))$,

under the assumptions:

....

We assume

(1) $x_0 \in A_0 \wedge y_0 \in A_0 \wedge \text{is-transitive}_{A_0}[R_0]$,

and show

(2) $y_0 \in \text{class}_{R_0,A_0}[x_0] \Rightarrow \text{class}_{R_0,A_0}[y_0] \subseteq \text{class}_{R_0,A_0}[x_0]$.

.....

For proving (4) we choose

(5) $y1_0 \in \text{class}_{R_0,A_0}[y_0]$,

and show:

(6) $y1_0 \in \text{class}_{R_0,A_0}[x_0]$.

Formula (6), using (Definition (class)), is implied by:

(7) $y1_0 \in \{a \mid_a a \in A_0 \wedge \langle a, x_0\rangle \in R_0\}$.

In order to prove (7) we have to show:

(8) $y1_0 \in A_0 \bigwedge \langle y1_0, x_0\rangle \in R_0$.

We prove the individual conjunctive parts of (8):

Proof of (8.1) $y1_0 \in A_0$:

Formula (5), by (Definition (class)), implies:

(9) $y1_0 \in \{a \mid_a a \in A_0 \wedge \langle a, y_0\rangle \in R_0\}$.

From what we already know follows:

From (9) we can infer

(10) $y1_0 \in A_0 \bigwedge \langle y1_0, y_0\rangle \in R_0$.

Formula (8.1) is true because it is identical to (10.1).

Proof of (8.2) $\langle y1_0, x_0\rangle \in R_0$:

Formula (5), by (Definition (class)), implies:

(11) $y1_0 \in \{a \mid_a a \in A_0 \wedge \langle a, y_0\rangle \in R_0\}$.

From what we already know follows:

From (11) we can infer

(12) $y1_0 \in A_0 \bigwedge \langle y1_0, y_0\rangle \in R_0$.

Formula (3), by (Definition (class)), implies:

(13) $y_0 \in \{a \mid_a a \in A_0 \wedge \langle a, x_0\rangle \in R_0\}$.

From what we already know follows:

From (13) we can infer

(14) $y_0 \in A_0 \wedge \langle y_0, x_0\rangle \in R_0$.

Formula (1.3), by (Definition (transitivity)), implies:

(15) $\forall_{x,y,z} (x \in A_0 \wedge y \in A_0 \wedge z \in A_0 \Rightarrow (\langle x, y\rangle \in R_0 \wedge \langle y, z\rangle \in R_0 \Rightarrow \langle x, z\rangle \in R_0))$.

Formula (8.2), using (15), is implied by:

(16) $\exists_y (x_0 \in A_0 \bigwedge y1_0 \in A_0 \bigwedge y \in A_0 \bigwedge \langle y1_0, y\rangle \in R_0 \bigwedge \langle y, x_0\rangle \in R_0)$.

Now, let $y := y_0$. Thus, for proving (16) it is sufficient to prove:

(17) $x_0 \in A_0 \bigwedge y1_0 \in A_0 \bigwedge y_0 \in A_0 \bigwedge \langle y1_0, y_0\rangle \in R_0 \bigwedge \langle y_0, x_0\rangle \in R_0$.

We prove the individual conjunctive parts of (17):

.... which are all true, because they are equivalent to previous (sub)formulae.

□

End of Theorema proof

As a consquence of the previous two propositions we can conclude that whenever x and y are related w.r.t. a symmetric and transitive relation R, i.e., $\langle x, y\rangle \in R$ resp. $x \in \text{class}_{R,A}[y]$, then their classes coincide.

Proposition["equal classes", any[A, R; x, y ∈ A], with[is-transitive$_A$[R] ⋀ is-symmetric$_A$[R]],

x ∈ class$_{R,A}$[y] ⇒ (class$_{R,A}$[x] = class$_{R,A}$[y])]

Prove it! Abort Show Off-line

Moreover, as a consequence, if the intersection of two classes is not empty, i.e., contains an common element, then the classes coincide.

Lemma["common element leads to equal classes",

any[A, R / x, y, z ∈ A], with[is-transitive$_A$[R] ⋀ is-symmetric$_A$[R]],

x ∈ class$_{R,A}$[y] ⋂ class$_{R,A}$[z] ⇒ (class$_{R,A}$[z] = class$_{R,A}$[y])]

Prove it! Abort Show Off-line

Beginn of Theorema proof

Prove:

.....

We assume

(1) $x_0 \in A_0 \bigwedge y_0 \in A_0 \bigwedge z_0 \in A_0 \bigwedge (\text{is-transitive}_{A_0}[R_0] \bigwedge \text{is-symmetric}_{A_0}[R_0])$,

and show

(2) $x_0 \in (\text{class}_{R_0,A_0}[y_0]) \bigcap (\text{class}_{R_0,A_0}[z_0]) \Rightarrow (\text{class}_{R_0,A_0}[z_0] = \text{class}_{R_0,A_0}[y_0])$.

We prove (2) by the deduction rule.

We assume

(3) $x_0 \in (\text{class}_{R_0,A_0}[y_0]) \bigcap (\text{class}_{R_0,A_0}[z_0])$

and show

(4) $\text{class}_{R_0,A_0}[z_0] = \text{class}_{R_0,A_0}[y_0]$.

From what we already know follows:

From (3) we can infer

(6) $x_0 \in \text{class}_{R_0,A_0}[y_0]$,

(7) $x_0 \in \text{class}_{R_0,A_0}[z_0]$,

(5) $(\text{class}_{R_0,A_0}[y_0]) \bigcap (\text{class}_{R_0,A_0}[z_0]) \neq \{\}$.

Formula (6), by (Proposition (equal classes)), implies:

(8) $x_0 \in A_0 \bigwedge y_0 \in A_0 \bigwedge \text{is-symmetric}_{A_0}[R_0] \bigwedge \text{is-transitive}_{A_0}[R_0] \Rightarrow (\text{class}_{R_0,A_0}[x_0] = \text{class}_{R_0,A_0}[y_0])$.

Formula (7), by (Proposition (equal classes)), implies:

(9) $x_0 \in A_0 \bigwedge z_0 \in A_0 \bigwedge \text{is-symmetric}_{A_0}[R_0] \bigwedge \text{is-transitive}_{A_0}[R_0] \Rightarrow (\text{class}_{R_0,A_0}[x_0] = \text{class}_{R_0,A_0}[z_0])$.

....

From (1.4.2) and (14) we obtain by modus ponens

(15) $\text{class}_{R_0,A_0}[x_0] = \text{class}_{R_0,A_0}[z_0]$.

....

From (1.4.2) and (16) we obtain by modus ponens

(17) $\text{class}_{R_0,A_0}[x_0] = \text{class}_{R_0,A_0}[y_0]$.

Formula (17), by (15), implies:

(18) $\text{class}_{R_0,A_0}[z_0] = \text{class}_{R_0,A_0}[y_0]$.

Formula (4) is true because it is identical to (18).

□

End of Theorema proof

Equivalence classes and partitions

Based on the previous investigations on the set of classes, which is also referred to as factor set, we turn in this section to the concept of a partition. The main theorem in this section will be that every factor set is a partition. Again we combine experiments with formal definitions and proofs such that the students shall have the possibility for first having ideas and conjectures about possible relation−ships and then formalizing and proving them in the framework of Theorema. The proof of the main theorem is preceded by three lemmata and as such demonstrates the student how proofs can be divided into substeps in order to become more readable in the end.

Again we will provide screenshots for the experiments and two selected proofs, in order to keep these comments insightful but not disturbing.

We have seen that in case of equivalence relations all classes contain at least one element, namely its representative and that the union of all classes yields the underlying universe. Further that if two elements are related by the equivalence relation, their classes coincide. Subsuming all equivalence classes in one set, the so-called *factor set*,

Definition["factor set", any[A, R],
$\text{factor-set}_{A}[R] := \left\{\text{class}_{R,A}[x] \mid_{x \in A}\right\}$]

provides us with a set of sets of a particular structure.

In general, if for some set M of subsets of a universe A, each element x of A is contained in at least one of these subsets, the set M is called a *covering*. A fact, which can equivalently be expressed as

Definition["covering", any[A, M],
$\text{is-covering}_{A}[M] :\Leftrightarrow \forall_{x \in A}\, x \in \bigcup M$]

The pairwise disjointness in M is defined by

Definition["disjoint", any[M],
$\text{are-disjoint}[M] :\Leftrightarrow \forall_{\substack{A,B \in M \\ A \neq B}} (A \cap B = \emptyset)$]

and for describing the fact that none of the elements of M is the empty set, we put

Definition["all nonempty", any[M],

$$\text{all-nonempty}[M] :\Leftrightarrow \underset{X\in M}{\forall} X \neq \emptyset\Big]$$

A set M of subsets of a universe which are all non-empty, pairwise disjoint and which forms a covering of the universe, is called a *partition* of the underlying universe.

Definition["partition", any[A, M],

$$\text{is-partition}_A[M] :\Leftrightarrow \bigwedge \begin{Bmatrix} \text{is-covering}_A[M] \\ \text{are-disjoint}[M] \\ \text{all-nonempty}[M] \end{Bmatrix}\Big]$$

Partitions possess the characteristic property that whenever two sets with common elements have been selected that the sets have to be equal.

Proposition["intersecting are equal", any[P], with[is-partition[P]],

$$\underset{A,B\in P}{\forall} (A \cap B \neq \emptyset \Rightarrow (A = B))\Big]$$

In fact this is a direct consequence of the property of pairwise disjointness which can be equivalently expressed as follows:

Proposition["are disjoint equivalence", any[M],

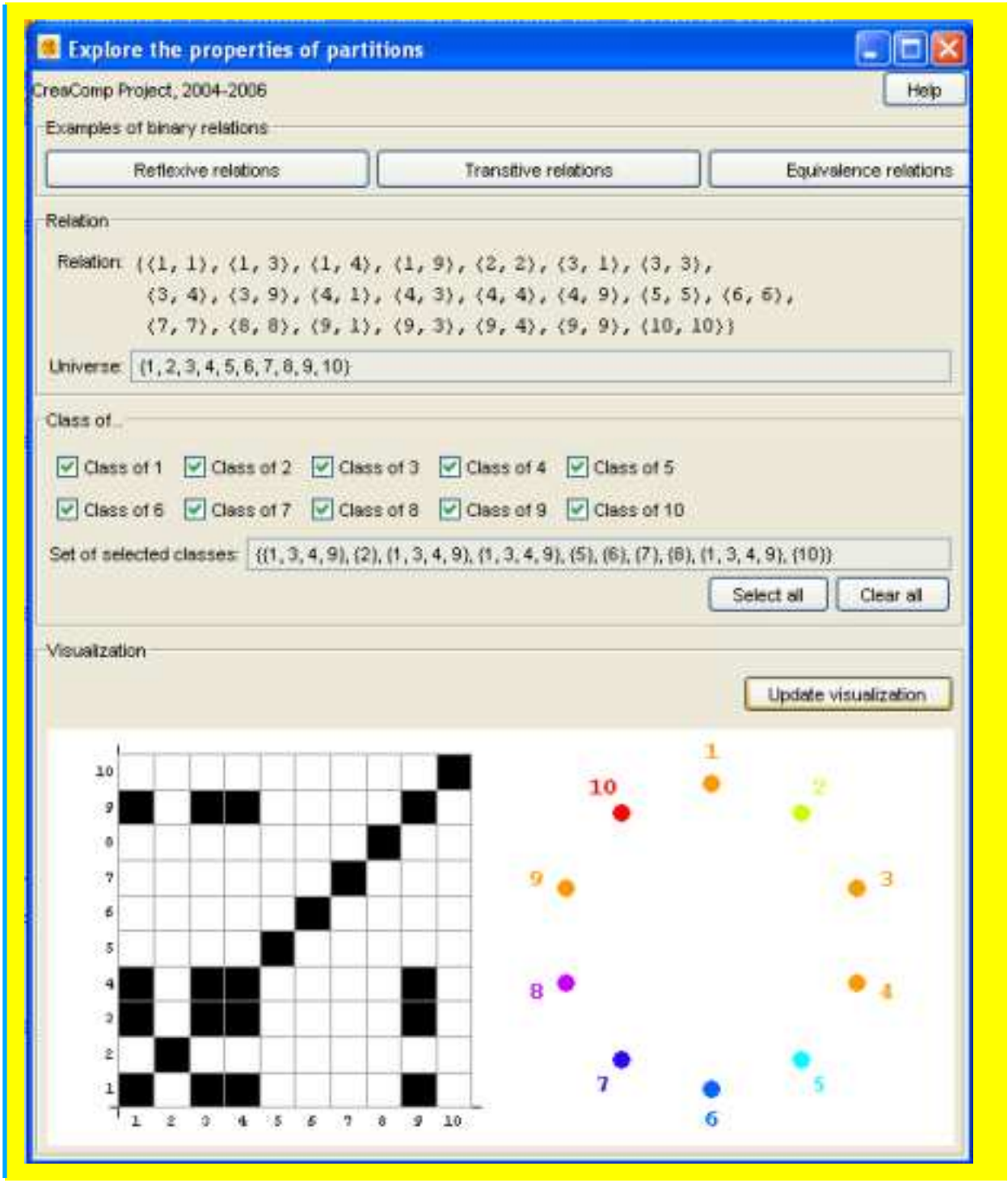
$$\text{are-disjoint}[M] \Leftrightarrow \underset{A,B\in M}{\forall} (A \cap B \neq \emptyset \Rightarrow (A = B))\Big]$$

Prove it! **Abort** **Show** **Off-line**

The properties of partitions seem to resemble the properties of the factor set of an equivalence relation. Therefore, try to verify the properties of a partition for the set of classes for some (arbitrary) binary relation as well as for an equivalence relation by using the following visualization tool. What do you conjecture?

Explore the properties of partitions by sets of classes

By pushing the button the following widget will open in order to carry out the experiment and to investigate the properties as described just before.



Main theorem

Based on the properties of a partition and our experiences with the factor set of an equivalence relation, we immediately conclude the following:

Theorem["factor set is partition", any[A, R], with[is-equivalence$_A$[R]], is-partition$_A$[factor-set$_A$[R]]]

Instead of showing the theorem immediately, let us formulate and prove auxiliary lemmata and prove the theorem later.

Lemma["factor set is covering", any[A, R], with[is-equivalence$_A$[R]], is-covering$_A$[factor-set$_A$[R]]]

Prove it! Abort Show Off-line

Beginn of Theorema proof

....

We assume

(1) $\text{is-equivalence}_{A_0}[R_0]$,

and show

(2) $\text{is-covering}_{A_0}[\text{factor-set}_{A_0}[R_0]]$.

Formula (2), using (Definition (factor set)), is implied by:

$$\text{is-covering}_{A_0}\left[\left\{\text{class}_{R_0,A_0}[x] \mid_x x \in A_0\right\}\right],$$

which, using (Definition (covering)), is implied by:

(3) $$\forall_{x1}\left(x1 \in A_0 \Rightarrow x1 \in \bigcup\left\{\text{class}_{R_0,A_0}[x] \mid_x x \in A_0\right\}\right).$$

We assume

(4) $x1_0 \in A_0$,

and show

(5) $$x1_0 \in \bigcup\left\{\text{class}_{R_0,A_0}[x] \mid_x x \in A_0\right\}.$$

In order to show (5) we have to show

(6) $$\exists_{x2}\left(x1_0 \in x2 \bigwedge x2 \in \left\{\text{class}_{R_0,A_0}[x] \mid_x x \in A_0\right\}\right).$$

In order to solve (6) we have to find x2^* such that

(7) $$x1_0 \in \text{x2}^* \bigwedge \exists_x (x \in A_0 \bigwedge (\text{x2}^* = \text{class}_{R_0,A_0}[x])).$$

Since (4) matches a part of (7) we try to instantiate, i.e. let now $x := x1_0$.

Thus, by (7), we choose $\text{x2}^* := \text{class}_{R_0,A_0}[x1_0]$.

Now, it suffices to show

(9) $x1_0 \in A_0 \bigwedge x1_0 \in \text{class}_{R_0,A_0}[x1_0]$.

We prove the individual conjunctive parts of (9):

Proof of (9.1) $x1_0 \in A_0$:

Formula (9.1) is true because it is identical to (4).

Proof of (9.2) $x1_0 \in \text{class}_{R_0,A_0}[x1_0]$:

Formula (9.2), using (Proposition (in own class)), is implied by:

(10) $x1_0 \in A_0 \bigwedge \text{is-reflexive}_{A_0}[R_0]$.

We prove the individual conjunctive parts of (10):

Proof of (10.1) $x1_0 \in A_0$:

Formula (10.1) is true because it is identical to (4).

Proof of (10.2) $\text{is-reflexive}_{A_0}[R_0]$:

Formula (1), by (Definition (equivalence)), implies:

(11) $\text{is-reflexive}_{A_0}[R_0] \bigwedge \text{is-relation}_{A_0}[R_0] \bigwedge \text{is-symmetric}_{A_0}[R_0] \bigwedge \text{is-transitive}_{A_0}[R_0]$.

Formula (10.2) is true because it is identical to (11.1).

□

Enof Theorema proof

Lemma["factor set of symm, trans relation fulfills disjointness",
any[A, R], with[is-symmetric_A[R] ⋀ is-transitive_A[R]],
$\text{are-disjoint}[\text{factor-set}_A[R]]$]

Prove it! **Abort** **Show** **Off-line**

And as a consquence we can immediately state also the following lemma.

Lemma["factor set of equivalence fulfills disjointness", any[A, R], with[is-equivalence_A[R]],
$\text{are-disjoint}[\text{factor-set}_A[R]]$]

Prove it! **Abort** **Show** **Off-line**

Finally we show that factor sets do not contain empty sets.

Lemma["factor set fulfills all−nonempty", any[A, R], with[is-equivalence_A[R]],
$\text{all-nonempty}[\text{factor-set}_A[R]]$]

Prove it! **Abort** **Show** **Off-line**

Prove the theorem! **Abort** **Show** **Off-line**

Begin of the Theorema proof of the final theorem

Prove:

(Theorem (factor set is partition)) $\underset{A,R}{\forall}(\text{is-equivalence}_A[R] \Rightarrow \text{is-partition}_A[\text{factor-set}_A[R]])$,

under the assumptions:

(Definition (partition)) $\underset{A,M}{\forall}(\text{is-partition}_A[M] :\Leftrightarrow \text{is-covering}_A[M] \bigwedge \text{are-disjoint}[M] \bigwedge \text{all-nonempty}[M])$,

(Lemma (factor set of equivalence fulfills disjointness)) $\underset{A,R}{\forall}(\text{is-equivalence}_A[R] \Rightarrow \text{are-disjoint}[\text{factor-set}_A[R]])$

,

(Lemma (factor set fulfills all−nonempty)) $\underset{A,R}{\forall}(\text{is-equivalence}_A[R] \Rightarrow \text{all-nonempty}[\text{factor-set}_A[R]])$,

(Lemma (factor set is covering)) $\underset{A,R}{\forall}(\text{is-equivalence}_A[R] \Rightarrow \text{is-covering}_A[\text{factor-set}_A[R]])$.

We assume

(1) $\text{is-equivalence}_{A_0}[R_0]$,

and show

(2) $\text{is-partition}_{A_0}[\text{factor-set}_{A_0}[R_0]]$.

Formula (2), using (Definition (partition)), is implied by:

$\text{all-nonempty}[\text{factor-set}_{A_0}[R_0]] \bigwedge \text{are-disjoint}[\text{factor-set}_{A_0}[R_0]] \bigwedge \text{is-covering}_{A_0}[\text{factor-set}_{A_0}[R_0]]$,

which, using (Lemma (factor set of equivalence fulfills disjointness)), is implied by:

$\text{all-nonempty}[\text{factor-set}_{A_0}[R_0]] \bigwedge \text{is-covering}_{A_0}[\text{factor-set}_{A_0}[R_0]] \bigwedge \text{is-equivalence}_{A_0}[R_0]$,

which, using (Lemma (factor set fulfills all−nonempty)), is implied by:

$\text{is-covering}_{A_0}[\text{factor-set}_{A_0}[R_0]] \bigwedge \text{is-equivalence}_{A_0}[R_0]$,

which, using (Lemma (factor set is covering)), is implied by:

(6) $\text{is-equivalence}_{A_0}[R_0]$.

Formula (6) is true because it is identical to (1).

□

End of the Theorema proof of the final theorem

Resumee

So far we have seen that from equivalence relations we can determine equivalence classes. These can be summarized in a factor set which in turn forms a partition of the underlying universe. What about going the reversed way, namely from sets of sets to the definition of a binary relation? For which types of sets of sets do we again end up with an equivalence relation?

Relations induced by sets of sets

As indicated just before we will now look on binary relations induced by a set of sets. The student shall first investigate, how different sets of sets influence the property of the relation by some experi−ments, finally conjectures about relations induced by a partition shall be formulated and proven. Again

One possibilty of inducing a binary relation from a set of sets is to state that two elements are related to each other whenever there exists at least one set in the collection of sets which contains both these elements.

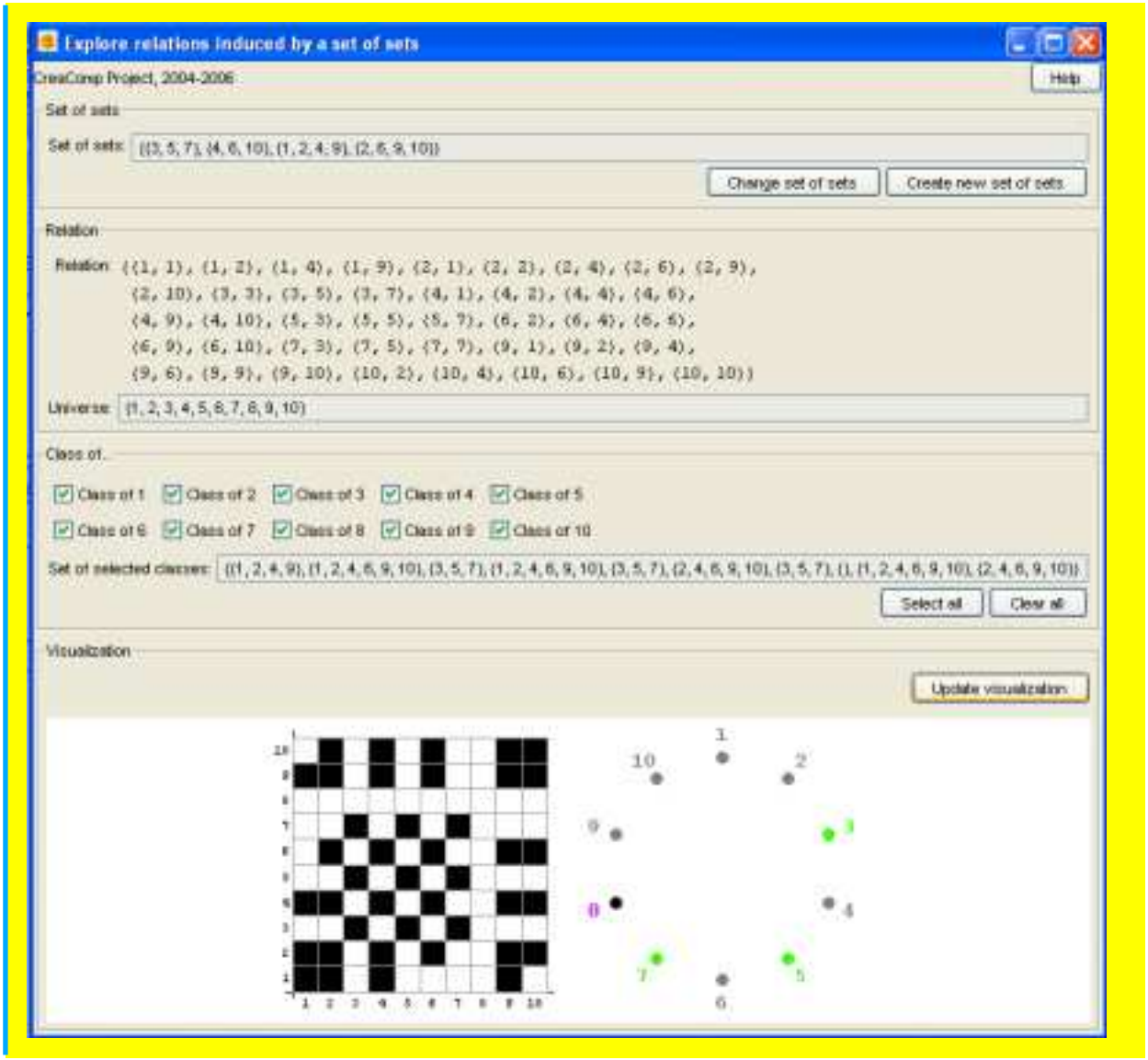
Definition["induced relation", any[A, S],

$$\text{induced-relation}_A[S] :\Leftrightarrow \left\{\langle x, y\rangle \;\middle|_{x,y\in A}\; \exists_{M\in S}\,(x \in M \wedge y \in M)\right\}]$$

The following link allows for illustrating binary relations on the universe $\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$ by means of a set of sets. Several predefined such collections are provided, however, you can and should provide your own sets of sets and illustrate the relation as done previously. For which types of sets is the induced relation reflexive, symmetric resp. transitive? When is it an equivalence relation?

Explore relations induced by sets of sets

Clicking the button will open a widget of the following type. It allows to generate sets of sets ran−domly, but also to modify the current set of sets manually. Classes and visualization tool can be handled as in the previous experiments.



Conjecture

As we could see by our previous experiments, partitions induce equivalence relations. We state the formal theorem. After a first attempt we again divide the proof into four subparts each of them referring to the particular properties of an equivalence relation, namely being a binary relation, being reflexive, symmetric and transitive.

Theorem["induced by partition is equivalence", any[A, P], with[is-partition$_A$[P]], is-equivalence$_A$[induced-relation$_A$[P]]]

Prove it! **Abort** **Show** **Off-line**

The immediate proof fails, so we formulate and prove auxiliary lemmata that will allow to prove the theorem later.

Lemma["induced by partition is reflexive", any[A, P], with[is-partition$_A$[P]], is-reflexive$_A$[induced-relation$_A$[P]]]

Prove it! Abort Show Off-line

Try the theorem again Abort Show Off-line

Lemma["induced by partition is relation", any[A, P],
is-relation$_A$[induced-relation$_A$[P]]]

Prove it! Abort Show Off-line

Beginn of Theorema proof

....

For proving (Lemma (induced by partition is relation)) we take all variables arbitrary but fixed and prove:

(1) $\text{is-relation}_{A_0}[\text{induced-relation}_{A_0}[P_0]]$.

Formula (1), using (Definition (relation)), is implied by:

$\text{induced-relation}_{A_0}[P_0] \subseteq A_0 \times A_0$,

which, using (Definition (induced relation)), is implied by:

(2) $\left\{\langle x, y\rangle \Big|_{x,y} x \in A_0 \bigwedge y \in A_0 \bigwedge \exists_{M \in P_0} (x \in M \wedge y \in M)\right\} \subseteq A_0 \times A_0$.

For proving (2) we choose

(3) $x1_0 \in \left\{\langle x, y\rangle \Big|_{x,y} x \in A_0 \bigwedge y \in A_0 \bigwedge \exists_{M \in P_0} (x \in M \wedge y \in M)\right\}$,

and show:

(4) $x1_0 \in A_0 \times A_0$.

From what we already know follows:

From (3) we know by definition of $\{T_x \mid_x P\}$ that we can choose an appropriate value such that

(5) $x2_0 \in A_0 \bigwedge x3_0 \in A_0 \bigwedge \exists_M (M \in P_0 \wedge (x2_0 \in M \wedge x3_0 \in M))$,

(6) $x1_0 = \langle x2_0, x3_0\rangle$.

We prove (4) by checking the individual components:

We have to prove:

(9) $|x1_0| = 2$.

Formula (9), using (6), is implied by:

(12) $|\langle x2_0, x3_0\rangle| = 2$.

Using built-in simplification rules we simplify (12) to

(13) True.

We have to prove:

(7) $x1_{0\,1} \in A_0$.

Formula (7), using (6), is implied by:

(16) $(\langle x2_0, x3_0\rangle)_1 \in A_0$.

Using built-in simplification rules we simplify (16) to

(17) $x2_0 \in A_0$.

Formula (17) is true because it is identical to (5.1).

We have to prove:

(8) $x1_{0\,2} \in A_0$.

Formula (8), using (6), is implied by:

(20) $(\langle x2_0, x3_0 \rangle)_2 \in A_0$.

Using built-in simplification rules we simplify (20) to

(21) $x3_0 \in A_0$.

Formula (21) is true because it is identical to (5.2).

□

End of Theorema proof

Since new properties are now available for the knowledge base the student is again encouraged to try to prove the theorem again. By analyzing the unsuccessful proof attempt further conjectures shall be found and proven.

Try the theorem again **Abort** **Show** **Off-line**

Beginn of an unsuccessful proof attempt

Prove:

(Theorem (induced by partition is equivalence)) $\forall_{A,P}$ (is-partition$_A$[P] ⇒ is-equivalence$_A$[induced-relation$_A$[P]])

,

under the assumptions:

(Lemma (induced by partition is reflexive)) $\forall_{A,P}$ (is-partition$_A$[P] ⇒ is-reflexive$_A$[induced-relation$_A$[P]]),

(Lemma (induced by partition is relation)) $\forall_{A,P}$ is-relation$_A$[induced-relation$_A$[P]],

(Definition (equivalence))

$\forall_{A,R}$ (is-equivalence$_A$[R] :⇔ is-relation$_A$[R] ⋀ is-reflexive$_A$[R] ⋀ is-symmetric$_A$[R] ⋀ is-transitive$_A$[R]).

As there are several methods which can be applied, we have different choices to proceed with the proof.

Alternative proof 1: failed

We assume

(1) is-partition$_{A_0}$[P_0],

and show

(2) is-equivalence$_{A_0}$[induced-relation$_{A_0}$[P_0]].

Formula (2), using (Definition (equivalence)), is implied by:

is-reflexive$_{A_0}$[induced-relation$_{A_0}$[P_0]] ⋀ is-relation$_{A_0}$[induced-relation$_{A_0}$[P_0]] ⋀ ,
is-symmetric$_{A_0}$[induced-relation$_{A_0}$[P_0]] ⋀ is-transitive$_{A_0}$[induced-relation$_{A_0}$[P_0]]

which, using (Lemma (induced by partition is reflexive)), is implied by:

(3) is-partition$_{A_0}$[P_0] ⋀ is-relation$_{A_0}$[induced-relation$_{A_0}$[P_0]] ⋀ .
is-symmetric$_{A_0}$[induced-relation$_{A_0}$[P_0]] ⋀ is-transitive$_{A_0}$[induced-relation$_{A_0}$[P_0]]

As there are several methods which can be applied, we have different choices to proceed with the proof.

Alternative proof 1: failed

Not all the conjunctive parts of (3)can be proved.

Proof of (3.1) is-partition$_{A_0}$[P_0]:

Formula (3.1) is true because it is identical to (1).

Proof of (3.2) is-relation$_{A_0}$ [induced-relation$_{A_0}$ [P_0]]:

Formula (3.2) is proved because it is an instance of (Lemma (induced by partition is relation)).

Proof of (3.3) is-symmetric$_{A_0}$ [induced-relation$_{A_0}$ [P_0]]:

The proof of (3.3)fails. (The prover "QR" was unable to transform the proof situation.)

Proof of (3.4) is-transitive$_{A_0}$ [induced-relation$_{A_0}$ [P_0]]:

The proof of (3.4)fails. (The prover "QR" was unable to transform the proof situation.)

.....

□

End of the unsuccessful proof attempt

Lemma["induced by partition is symmetric", any[A, P],
is-symmetric$_A$ [induced-relation$_A$ [P]]]

Prove it! **Abort** **Show** **Off-line**

Try the theorem again **Abort** **Show** **Off-line**

Lemma["induced by partition is transitive", any[A, P], with[is-partition$_A$ [P]],
is-transitive$_A$ [induced-relation$_A$ [P]]]

Try the theorem again **Abort** **Show** **Off-line**

Beginn of finally successful Theorema proof

(Theorem (induced by partition is equivalence)) $\forall_{A,P}$ (is-partition$_A$ [P] ⇒ is-equivalence$_A$ [induced-relation$_A$ [P]])

,

.....

We assume

(1) is-partition$_{A_0}$ [P_0],

and show

(2) is-equivalence$_{A_0}$ [induced-relation$_{A_0}$ [P_0]].

Formula (2), using (Definition (equivalence)), is implied by:

is-reflexive$_{A_0}$ [induced-relation$_{A_0}$ [P_0]] ⋀ is-relation$_{A_0}$ [induced-relation$_{A_0}$ [P_0]] ⋀ ,
is-symmetric$_{A_0}$ [induced-relation$_{A_0}$ [P_0]] ⋀ is-transitive$_{A_0}$ [induced-relation$_{A_0}$ [P_0]]

which, using (Lemma (induced by partition is reflexive)), is implied by:

is-partition$_{A_0}$ [P_0] ⋀ is-relation$_{A_0}$ [induced-relation$_{A_0}$ [P_0]] ⋀ ,
is-symmetric$_{A_0}$ [induced-relation$_{A_0}$ [P_0]] ⋀ is-transitive$_{A_0}$ [induced-relation$_{A_0}$ [P_0]]

which, using (Lemma (induced by partition is transitive)), is implied by:

(3) is-partition$_{A_0}$ [P_0] ⋀ is-relation$_{A_0}$ [induced-relation$_{A_0}$ [P_0]] ⋀ is-symmetric$_{A_0}$ [induced-relation$_{A_0}$ [P_0]].

We prove the individual conjunctive parts of (3):

Proof of (3.1) is-partition$_{A_0}$ [P_0]:

Formula (3.1) is true because it is identical to (1).

Proof of (3.2) is-relation$_{A_0}$ [induced-relation$_{A_0}$ [P_0]]:

Formula (3.2) is proved because it is an instance of (Lemma (induced by partition is relation)).

Proof of (3.3) is-symmetric$_{A_0}$ [induced-relation$_{A_0}$ [P_0]]:

Formula (3.3) is proved because it is an instance of (Lemma (induced by partition is symmetric)).

□

End of Theorema proof

From relations to sets of sets back to relations

In the previous sections the student has seen that from equivalence relations partitions can be deduced as well as that partitions again induce equivalence relation. This last section is dedicated to experiments showing that applying these both steps consecutively leads to the original relation resp. partition. As such there is a one-to-one-correspondence between equivalence relations and partitions. The final theorems are stated just to formalize this relationsship. The proofs are carried out by two basic lemmata. Since they are similar we just display one of them.

Finally, we have a look whether in case of equivalence relations resp. partitions we can apply these processes - "from relations to sets of sets" and "from sets of sets to relations" - consecutively such that finally we end up with the same result. Again we invite you to do some experiments first in order to build up new conjectures, before we provide the final theorems.

Explore relations induced by set of classes and classes of relations induced by a set of sets

In order to increase the readability we have rotated the following screenshot of the experiment which consists of two panels - one dealing with relations induced by sets of sets, the other one with classes of relations. The contents of the helpfile explain how to use the experiment, which we quote here:

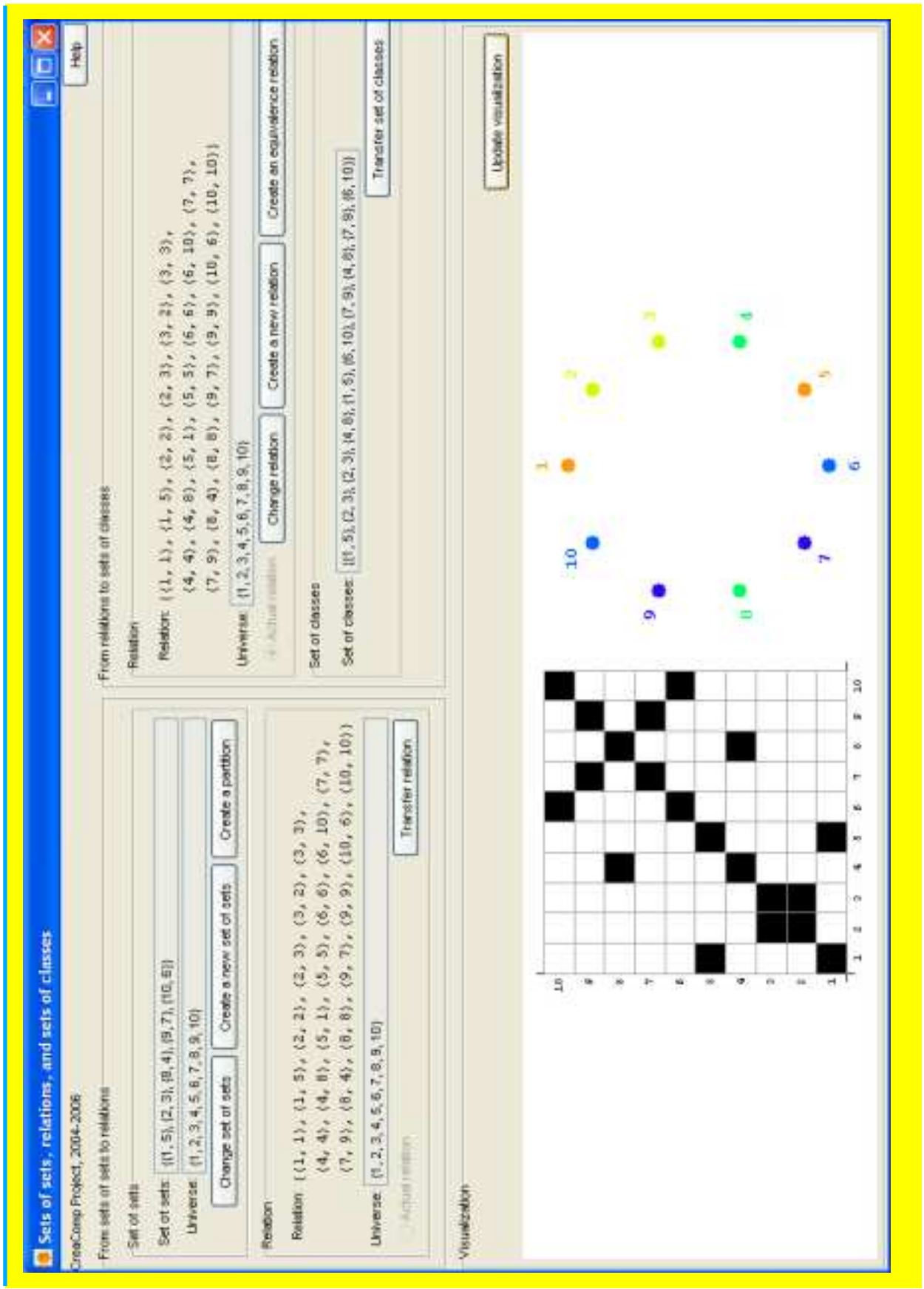
"Examples of binary relations induced by subsets of {1,2,3,4,5,6,7,8,9,10} can be studied. The left panel allows randomly generated sets of subsets and partitions as well as for changing a given set of subsets manually. Take care that indeed the provided sets are subsets of $\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$*. Any change will automatically update the display of the relation.*

The right panel part provides interfaces for investigating relations. Binary relations on $\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$ *can be defined manually, random examples of binary relations as well as equivalence relations can be generated by pressing the corresponding buttons. The set of corresponding classes will be updated automatically.*

In order to illustrate the binary relation currently in charge (it is highlighted by the "actual relation"-button) push the "Update visualization"-button.

By the corresponding "transfer"-buttons data can be shifted from one panel to the other, i.e., the deduced relation of the left panel is pasted into the right panel, vice versa, the set of classes of the right panel can be pasted in the set of sets field of the left panel.

Try to find out whether classes of relations induced by a set of sets coincide with the original set of sets. Further investigate whether relations induced by a set of classes yield again the same relation."


Sets of sets, relations, and sets of classes
CreaCongr Project, 2004-2006
Help
From sets of sets to relations
Set of sets
Set of sets: {{1, 5}, {2, 3}, {8, 4}, {9, 7}, {10, 6}}
Universe: {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
Change set of sets
Create a new set of sets
Create a partition
Relation
Relation: {{1, 1}, {1, 5}, {2, 2}, {2, 3}, {3, 2}, {3, 3}, {4, 4}, {4, 8}, {5, 1}, {5, 5}, {6, 6}, {6, 10}, {7, 7}, {7, 9}, {8, 4}, {8, 8}, {9, 7}, {9, 9}, {10, 6}, {10, 10}}
Universe: {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
Actual relation
Transfer relation
From relations to sets of classes
Relation
Relation: {{1, 1}, {1, 5}, {2, 2}, {2, 3}, {3, 2}, {3, 3}, {4, 4}, {4, 8}, {5, 1}, {5, 5}, {6, 6}, {6, 10}, {7, 7}, {7, 9}, {8, 4}, {8, 8}, {9, 7}, {9, 9}, {10, 6}, {10, 10}}
Universe: {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
Change relation
Create a new relation
Create an equivalence relation
Set of classes
Set of classes: {{1, 5}, {2, 3}, {2, 3}, {4, 8}, {1, 5}, {6, 10}, {7, 9}, {4, 8}, {7, 9}, {6, 10}}
Transfer set of classes
Visualization
Update visualization

Theorem["induced by factor set of R is R itself", any[A, R], with[is-equivalence$_A$[R]],
induced-relation$_A$[factor-set$_A$[R]] = R]

Lemma["induced by factor set of R contained in R",
any[A, R], with[is-symmetric$_A$[R] ⋀ is-transitive$_A$[R]],
induced-relation$_A$[factor-set$_A$[R]] ⊆ R]

Lemma["induced by factor set of R contains R", any[A, R], with[is-equivalence$_A$[R]], induced-relation$_A$[factor-set$_A$[R]] ⊇ R]

Prove the theorem! **Abort** **Show** **Off-line**

Theorem["factor set of induced by P is P itself", any[A, P], with[is-partition$_A$[P]], factor-set$_A$[induced-relation$_A$[P]] = P]

Lemma["factor set of induced by P contained in P", any[A, P], with[is-partition$_A$[P]], factor-set$_A$[induced-relation$_A$[P]] ⊆ P]

Lemma["factor set of induced by P is contained in P", any[A, P], with[is-partition$_A$[P]], factor-set$_A$[induced-relation$_A$[P]] ⊇ P]

Prove the theorem! **Abort** **Show** **Off-line**

Beginn of Theorema proof

Prove:

(Theorem (factor set of induced by P is P itself))

$\forall_{A,P}$ (is-partition$_A$[P] ⇒ (factor-set$_A$[induced-relation$_A$[P]] = P)),

under the assumptions:

(Lemma (factor set of induced by P contained in P))

$\forall_{A,P}$ (is-partition$_A$[P] ⇒ factor-set$_A$[induced-relation$_A$[P]] ⊆ P),

(Lemma (factor set of induced by P is contained in P))

$\forall_{A,P}$ (is-partition$_A$[P] ⇒ factor-set$_A$[induced-relation$_A$[P]] ⊇ P).

We assume

(1) is-partition$_{A_0}$[P_0],

and show

(2) factor-set$_{A_0}$[induced-relation$_{A_0}$[P_0]] = P_0.

Formula (1), by (Lemma (factor set of induced by P contained in P)), implies:

(3) factor-set$_{A_0}$[induced-relation$_{A_0}$[P_0]] ⊆ P_0.

In order to show (2), by (3), it is sufficient to show

(5) P_0 ⊆ factor-set$_{A_0}$[induced-relation$_{A_0}$[P_0]].

In order to show (5), by (Lemma (factor set of induced by P is contained in P)), it is sufficient to show

(6) is-partition$_{A_0}$[P_0].

Formula (6) is true because it is identical to (1).

□

End of Theorema proof

Summary

Finally we summarize the main topics of this unit.

We have investigated the fact that from binary relations we can deduce sets of classes. Further that from a collection of subsets we can determine a binary relation. In case that the relation is reflexive, symmetric and/or transitive, the induced classes have a particular structure or fulfill further properties. Moreover, in case of equivalence relations the set of classes forms a so called partition and the relation induced by this partition coincides with the equivalence relation. Vice versa coincides the factor set of a relation induced by partition with the original partition.

Commented Unit: Polynomial Interpolation

After finishing this module you should

** be familiar with the mathemtical problem setting of interpolation,*

** be familiar with standard algorithms for polynomial interpolation,*

** have an idea how to develop mathematical algorithms given their specification,*

** be able to implement algorithms for polynomial interpolation in an arbitrary programming language.*

Additional Modules

MathWorld ***Fast Computations***

MathWorld ***Polynomials in Theorema***

In the present module we will make use of the following *Mathematica*− and Theorema−commands:

- Definition, Theorem, Proposition, Lemma
- UseAlso, Compute, ComputationalSession, Prove

At the beginning of the unit we give an overview on its content and hyperlink to related units.

Introduction

Interpolation is an elementary problem in Mathematics. Consider a function $f : A \to B$ where the only knowl−edge available about f is a finite set of sample points $I \subseteq A \times B$. The problem consists of finding a representa−tion of f that "describes" the function "outside of I", ideally on the entire domain A. Reasonable descriptions of a function in the above sense could be:

- A term f_x describing the function values depending on $x \in A$, such that $f : A \to B, x \mapsto f_x$ is a term description of f.
- A program $P[x]$ that returns for every $x \in A$ a suitable $y \in B$ such that $f[x] = y$.

A term description is in general very powerful, because it not only allows to compute function values but also allows other operations like differentiation, intergration, computing limits, etc. Sometimes, however, the computation of a term representation is too expensive and the possibility of computing arbitrary function values is sufficient for the concrete application.
In this chapter, we want to present different algorithms for solving the interpolation problem, and we also want to give some advice in how to develop algorithms that solve an exactly specified problem.

Here we just give textual introduction to the mathematical setting treated in this unit.

The Problem of Interpolation

The classical interpolation problem consists of

- given two tuples x, a of same length and a set of interpolation functions Φ
- find $f \in \Phi$ such that $\forall_{i=1,\ldots,|x|} f[x_i] = a_i$.

In general, of course, there are always infinitely many functions that run through finitely many given pairs $\langle x_i, a_i \rangle$, therefore interesting classes of interpolation functions Φ are those that allow *unique solutions* to the interpolation problem. Let K be a field and let $\Pi[K]$ denote the set of polynomial functions over K. The set

$$\Pi[K, n] := \left\{ f \in \Pi[K] \,\middle|\, \deg_{\Pi[K]}[f] \leq n \right\}$$

is the set of polynomial functions over K of degree less equal n. For tuples x, a over some field K (with $x_i \neq x_j$ provided $i \neq j$) the set of polynomial functions of degree less $|x|$, i.e. the set $\Pi[K, |x| - 1]$, is an appropriate candidate for Φ. This leads to *polynomial interpolation*, i.e.

- given a field K and two tuples x, a of same length with $\forall_{\substack{i,j=1,\ldots,|x| \\ i \neq j}} x_i \neq x_j$
- find $f \in \Pi[K, |x| - 1]$ such that $\forall_{i=1,\ldots,|x|} f[x_i] = a_i$.

This is the exact problem specification for which we want to develop solution algorithms.

In a first step, we translate the problem stated for *polynomial functions* into a problem for *polynomials* only. The input stays the same, but now we need to

- find p such that $\deg_{\mathfrak{P}[K]}[p] \leq |x| - 1$ and $\forall_{i=1,\ldots,|x|} \operatorname{eval}_{\mathfrak{P}[K]}[p, x_i] = a_i$.

For notation and basic arithmetic in the domain of polynomials and polynomial functions in Theorema, see **Polynomials in Theorema**.

Here we hyperlink to the unit, where the representation of polynomials and their basic arithmetic is defined in Theorema language. We will use Theorema polynomials for computation in the rest of this unit. This should also explain the meaning of the unconventional notation (always writing $\mathfrak{P}[K]$ under the operators) as the functor notation in the Theorema system.

A First Solution to the Problem

For a first solution algorithm we observe that the domain of polynomials over K of degree less n forms a vector space of dimension n and thus has a basis B. In this vector space, the interpolating polynomial p for x and a can be written as

$$\sum_{j=1,\ldots,n} \lambda_j \cdot B_j \,.$$

As soon as we fix a basis B, this is an Ansatz with yet to be determined coefficients λ_j. Clearly, the λ_j need to be adjusted such that p satisfies the interpolation conditions given in the problem specification above. A first natural choice for B is the canonical basis of dimension n over K.

Definition["Canonical Basis", any[n, i, j, K],

$$\mathcal{B}[n, K] = \left\langle \underset{P[K]}{\text{canonic}}\left[\left\langle \delta_{i,j} \;|_{i=1,\ldots,n}\right\rangle\right] \;\middle|\; \right\rangle_{j=1,\ldots,n} \quad \text{"Canonic"}$$

$$\delta_{i,j} = \begin{cases} 1 & \Leftarrow \; i = j \\ 0 & \Leftarrow \; \text{otherwise} \end{cases} \quad \text{"Kronecker-Delta"}$$

]

UseAlso[Definition["Canonical Basis"]]

Compute[$\mathcal{B}$[5, $\mathbb{Q}$]]

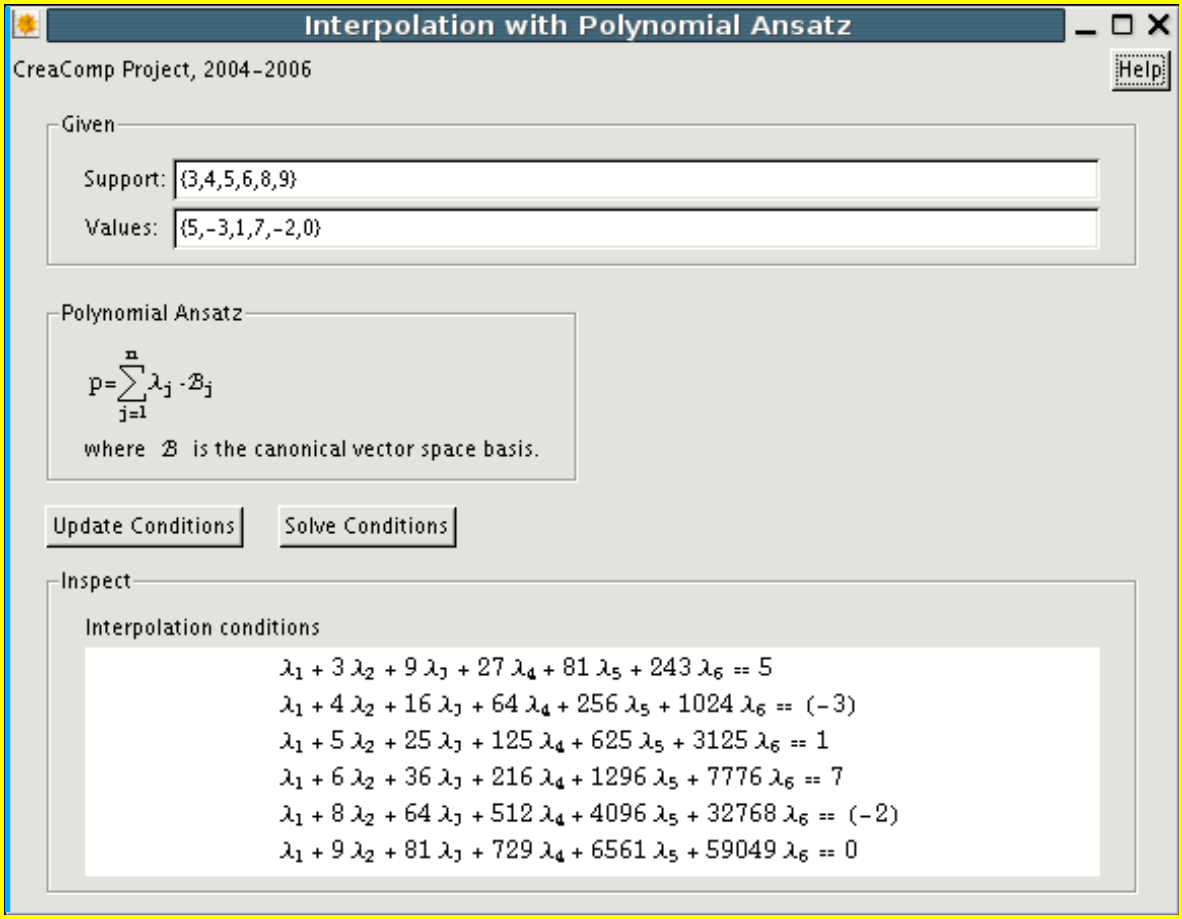
$\langle\langle 1\rangle, \langle 0, 1\rangle, \langle 0, 0, 1\rangle, \langle 0, 0, 0, 1\rangle, \langle 0, 0, 0, 0, 1\rangle\rangle$

The Theorema computation reveals the mathematical representation of polynomials as tuples of coefficients.

In the language of polynomials this can be read as $\langle 1, x, x^2, x^3, x^4\rangle$ as the canonical basis for the vector space of polynomials over $\mathbb{Q}$ of degree less than 5. Let's plug in this Ansatz into the interpolation conditions and try to identify the unknown coefficients λ_j.

Experiment with interpolation conditions

Pressing this button will open an interactive widget, in which the user can experiment with the condi−tions resulting from that polynomial Ansatz.



The widget has a Help−button in the top−right corner, which explains the intended use of that widget:

This widget should help exploring the method of computing the interpolation polynomial using a canonic "polynomial Ansatz".

The two input fields allow to enter a list of ordinate values (as in all other widgets this will be referred to as "Support") and a list of function values (as in all other widgets this will be referred to as "Val−ues"). The support corresponds to the tuple x in the problem specification, and the values correspond to the tuple a in the problem specification.

The following box shows the polynomial Ansatz with unknown coefficients λ_j. The bottom box shows the resulting conditions for the λ_j for the given input lists.

When changing the input, the conditions are updated when pressing the "Update Conditions"−Button or when hitting the RET−key on the keyboard in an input field. Pressing the "Solve Conditions"−Button computes the solutions for the λ_j.

The observation should be that the conditions become more and more complicated with growing length of the input lists though always being a system of linear equations.

So we could solve the interpolation problem in this fashion. Does it always work? After all, it is much effort to determine the coefficients λ_j with this method. We therefore try to invent a better method.

Lagrange Interpolation

For Lagrange interpolation we want the students to capture the basic idea of that approach: choose basis polynomials in such a way that the coefficients in the linear combination can be determined with ease. In fact, the basis polynomials can be chosen such that the coefficients are just the given a_j. For this the basis polynomials need to be chosen such that they evaluate to 0 at all support points but one, where they need to evaluate to 1. Using appropriate interactive experiments, the students should be lead to this insight themselves.

For a new interpolation algorithm we start out with the same idea, namely to represent the interpolation polynomial as

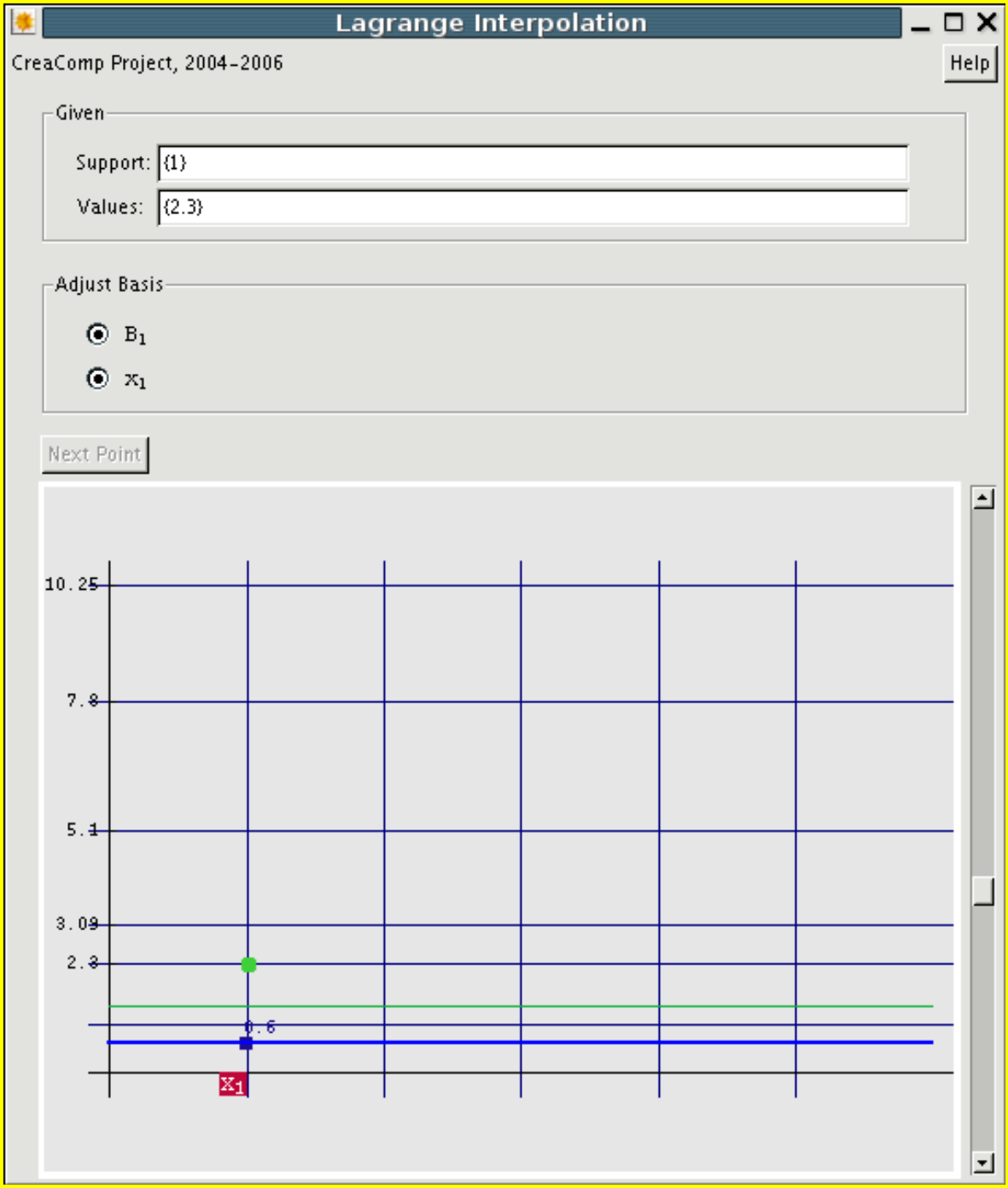
$$\sum_{j=1,\ldots,n} \lambda_j \cdot B_j \, .$$

However, now we want to use a basis B that allows to determine the coefficients λ_j very easily. It will turn out that for given x and a we can choose B such that $\lambda_j = a_j$, for all $j = 1, \ldots, n$, so that our interpolation polynomial can be represented as

$$\sum_{j=1,\ldots,n} a_j \cdot B_j \, .$$

How to choose the B_j

Pressing this button will open an interactive widget, in which the user can adjust the basis polynomials. This should give the basic motivation for Lagrange interpolation, it should provide the specification for the Lagrange basis polynomials.



The widget has a Help−button in the top−right corner, which explains the intended use of that widget:

This widget should help exploring the key properties of the Lagrange basis polynomials.

The basis polynomials B_j should be adjusted in such a way that the linear combination $\sum_{j=1}^{n} a_j \cdot B_j$ is the interpolationg polynomial for x and a. In this widget, the tuples x and a are predefined, i.e. they must be specified as parameters when the widget is called. The input fields for "Support" and "Val−ues" can therefore not be edited.

The idea of this experiment is to explore the interpolation property step by step with increasing n starting with $n = 1$. In this phase, there is only B_1 and we want to adjust the evaluation properties of B_1 such that $\sum_{j=1}^{1} a_j \cdot B_j = a_1 \cdot B_1$ is the desired interpolation polynomial. The widget allows to adjust the value of B_1 at x_1 using the slider bar at the right margin. Moving the slider updates the graphics and shows the current B_1 in blue and the resulting linear combination $a_1 \cdot B_1$ in green.

In each step, the picture marks all interpolation points with thick green points, each interpolation

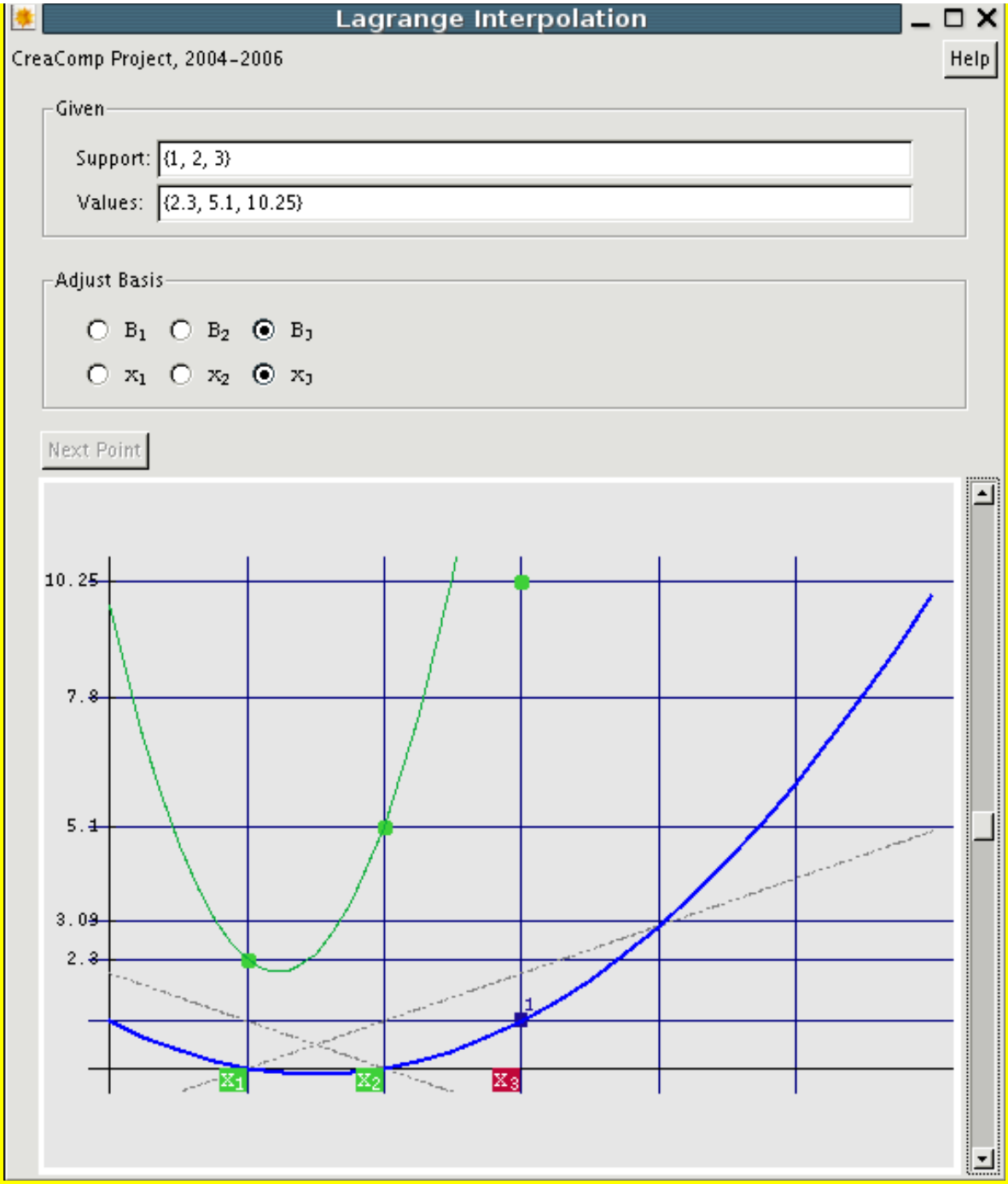
ordinate is labeled near the first axis. Red background means that the interpolation condition is violated at that interpolation point, green background indicates that the chosen linear combination interpolates at that point. Once the interpolation conditions are satisfied at all points (all labels show green background), the "Next Point"−Button turns active indicating that one may proceed to the next phase increasing n *by 1, i.e. considering the next interpolation point in addition and try to re−adjust the basis.*

Proceeding to the next point adds an additional basis polynomial and an additional interpolation point with the corresponding radio buttons in the "Adjust Basis"−area. Using the radio buttons the user can guide to which basis polynomial and to which interpolation point the adjustments of the slider bar apply. The active basis polynomial is always rendered in blue, all inactive basis polynomials are rendered dashed gray.

In phase $n = 1$ *it is straightforward to realize that fixing* B_1 *to evaluate to 1 at* x_1 *is the right choice.*

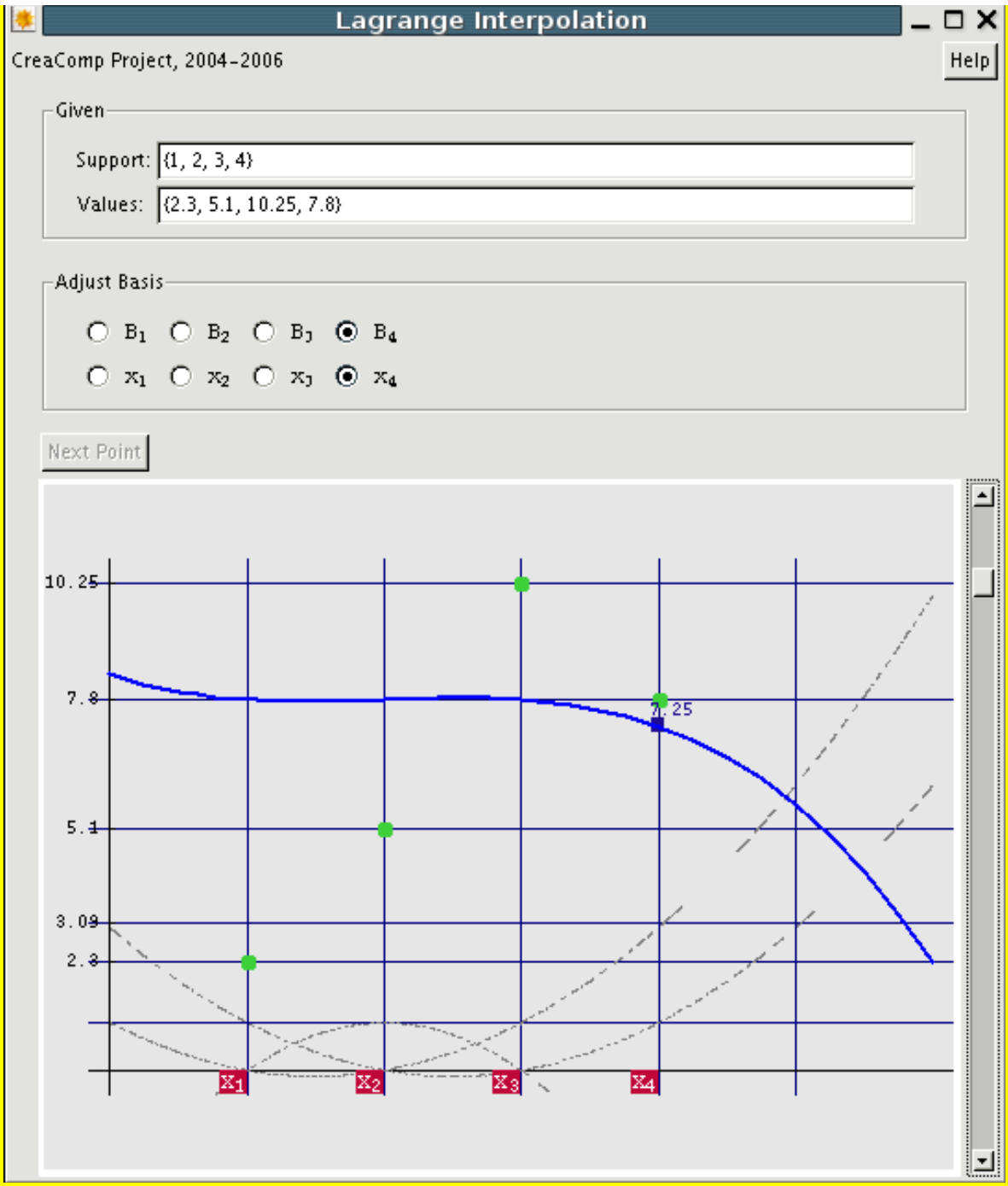
Proceeding to $n = 2$ *automatically initializes* B_1 *to 0 at* x_2 *whereas it leaves the previously successful choice 1 at* x_1*.* B_2 *is constant 0 initially. Raising* B_2 *to 1 at* x_2 *is what needs to be done in order to succeed in that phase.*

Proceeding to $n = 3$ *is the crucial moment in this experiment:* B_1 *and* B_2 *are left as they were,* B_3 *is again constant 0. The successful move from the previous rounds is to increase the "youngest" basis polynomial (in this case* B_3*) to 1 at the youngest interpolation point (in this case* x_3*). This is not sufficient anymore! However, one sees that* B_3 *is now proposed to be a parabula, i.e. a polynomial of degree 2.*



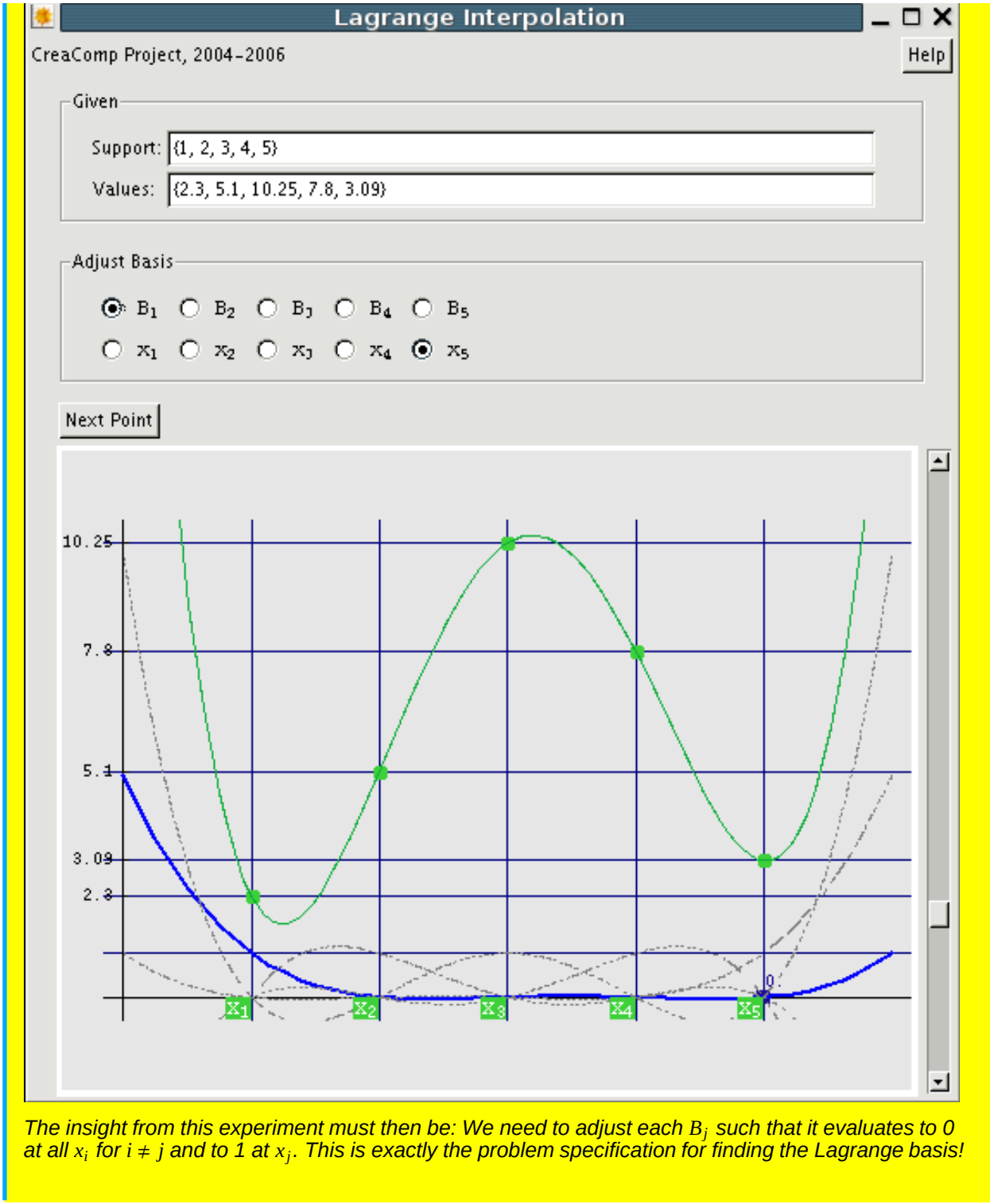
Since the widget allows to adjust B_1 and B_2 also, the student might want to re−adjust the values of these polynomials at x_3. Forcing B_1 and B_2 to evaluate to 0 at x_3 might not be the first thing they try but from the previous phases this is probably a reasonable strategy. If they manage, then they have understood the key concept!

Proceeding to $n = 4$ initializes B_4 differently to the previous stages: it is not initialized constant 0 but it is initialized as a certain degree 3 polynomial. The interpolation conditions are violated at all points inititially!



With the experience from the previous rounds the student may re−adjust all polynomials at all points in order to fit the linear combination through all points again! Setting B_4 to 1 at x_4 is the reasonable first choice. Once B_4 is set to 0 at some x_j the interpolation condition is satisfied at that point turning its background into green, thus setting B_4 to 0 at all x_1, x_2, x_3 is near at hand. What remains then is to fulfill the interpolation condition at x_4, which will be established by setting B_1, B_2, B_3 all to 0 at x_4, analogous to the previous phases.

In the optimal case the student finishes phase $n = 5$ as follows:



The insight from this experiment must then be: We need to adjust each B_j such that it evaluates to 0 at all x_i for $i \neq j$ and to 1 at x_j. This is exactly the problem specification for finding the Lagrange basis!

Thus, in order to compute the basis polynomial B_j, we need to find a polynomial of degree $n-1$, which evaluates

- to 0 at all x_i ($i \neq j$) and
- to 1 at x_j

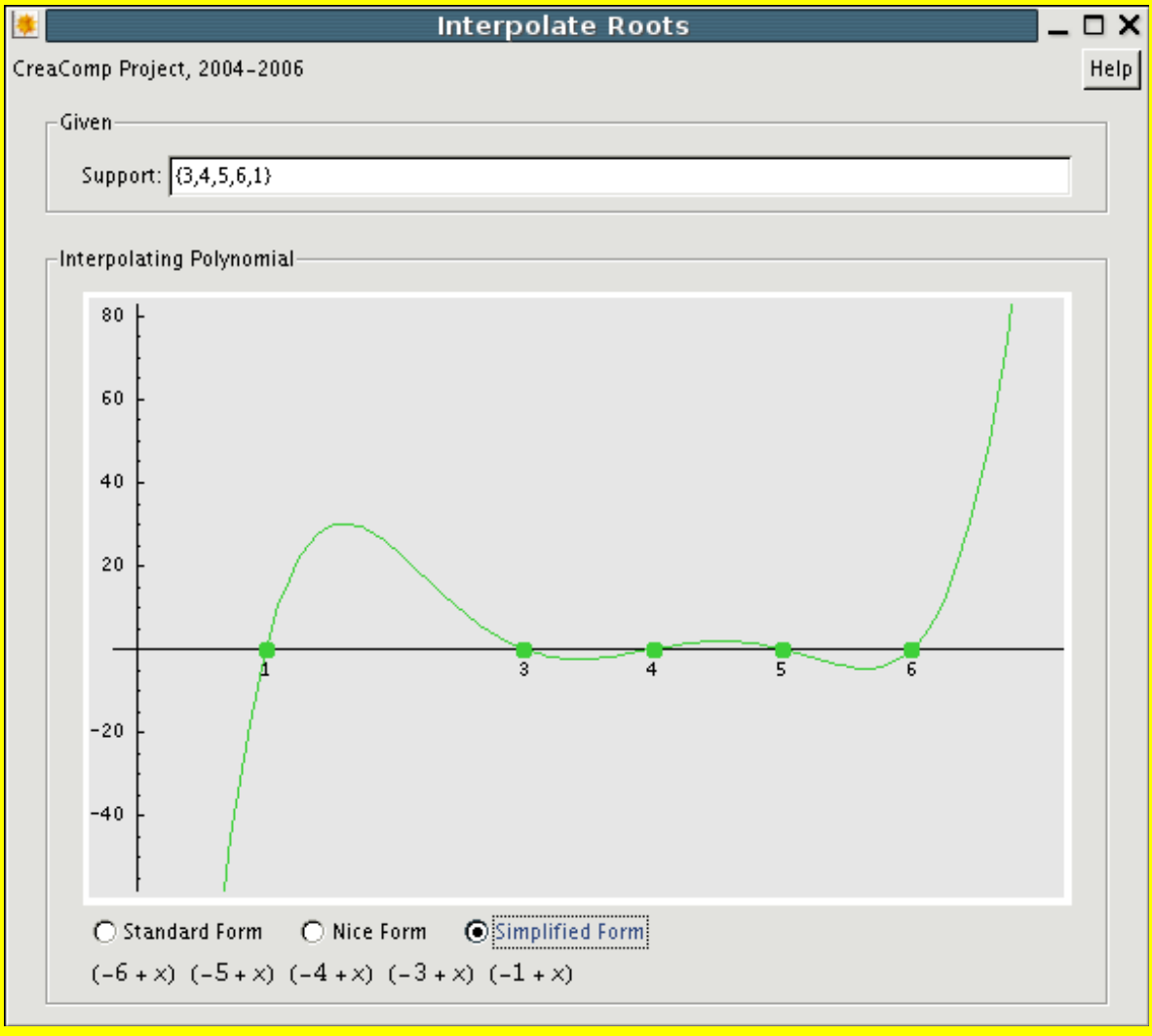
in other words

$$\underset{\mathbb{P}[K]}{\text{eval}}[B_j, x_i] = \delta_{i,j} .$$

For finally solving the interpolation problem using this approach we now need to find a method to compute the Lagrange basis, for which we've just obtained its chracteristic property. The following experiment is a first step in this direction.

Polynomial with given roots

Pressing this button will open an interactive widget, in which the user can find a polynomial that runs through given zeros.



The widget has a Help−button in the top−right corner, which explains the intended use of that widget:

This widget should help in finding out how to construct the Lagrange basis polynomials.

A root of a polynomial is an element where the polynomial evaluates to 0.

The widget has an input field for choosing the "Support", a graphical visualization of the polynomial that has "Support" as the set of its roots, and it displays the polynomial in the bottom area. Using the radio buttons the user can decide how to display the polynomial:

"Standard Form" is the tuple of coefficients,

"Nice Form" is the common form like $c_0 + c_1\, x + c_2\, x^2 + \ldots + c_n\, x^n$*, and*

"Simplified Form" is the factorized polynomial (written as a product of factors).

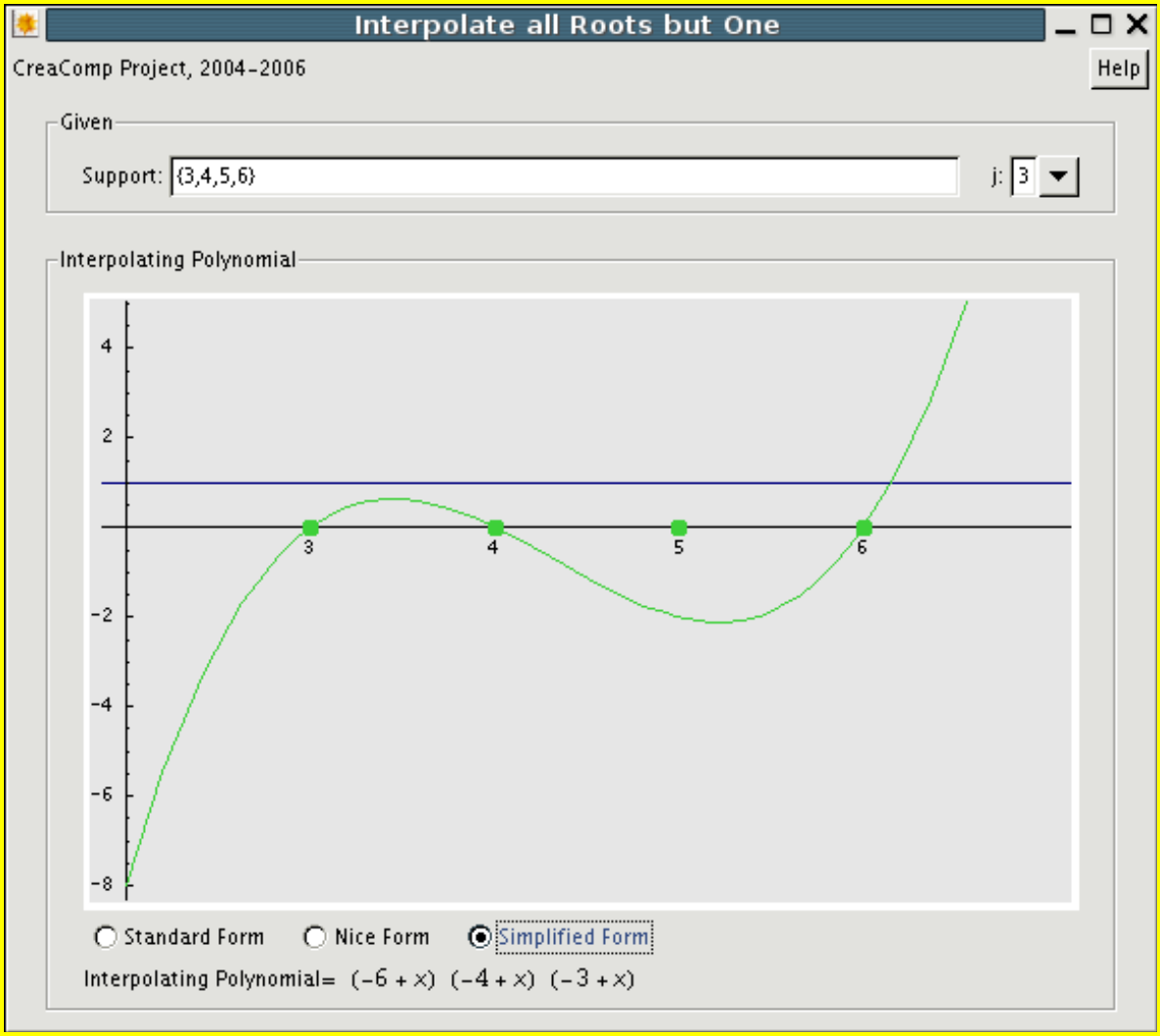
The widget updates whenever the support is changed and the RET *−key is pressed on the keyboard in the input field.*

Switching display forms should first of all make the student familiar with the tuple notation. The more important aspect is that the student may observe that the polynomial always factors completely into linear factors, and that each factor has the form $(x - r_i)$, where r_i are the chosen roots.

This should give a general idea how to construct a polynomial with given roots (i.e. values, where the polynomial evaluates to 0). Given support points $x_1, \ldots, x_n$, let's now find a polynomial that evaluates to 0 at all x_i for $i \neq j$.

Polynomial with roots x_i for $i \neq j$

Pressing this button will open an interactive widget, in which the user can find a polynomial that runs through all given zeros but one.



The widget has a Help−button in the top−right corner, which explains the intended use of that widget:

Again we have an input field for choosing the "Support". Then we have a "Pull Down"−menu for choosing j. The j−th Lagrange basis polynomial B_j should evaluate to 0 at x_i for $i \neq j$, thus $B_j = \prod_{i \neq j} (x - x_i)$ seems a good idea. Furthermore, the widget computes and visualizes this "guess" for B_j. Using the radio buttons the user can decide how to display the polynomial:

"Standard Form" is the tuple of coefficients,

"Nice Form" is the common form like $c_0 + c_1\, x + c_2\, x^2 + \ldots + c_n\, x^n$, and

"Simplified Form" is the factorized polynomial (written as a product of factors).

The widget updates whenever the support is changed and the RET−key is pressed on the keyboard in the input field or when a new j is selected.

Switching display forms should first of all make the student familiar with the tuple notation. The more important aspect is that the student may observe that the polynomial always factors completely into linear factors, and that each factor has the form $(x - r_i)$, where r_i are the chosen roots, only $(x - r_j)$ is missing!

Conclusion: the polynomials $x - x_i$ evaluate to 0 at x_i, respectively, hence *their product* for $i \neq j$ will evaluate to 0 *at all* x_i for $i \neq j$. Let's write this in our language for polynomials: the "polynomial" $x - x_i$ writes as the tuple $\langle -x_i, 1 \rangle$, thus the following algorithm computes the polynomial $B[j, x, K]$ with roots x_i for $i \neq j$.

Algorithm["Given Roots", any[j, x, K],

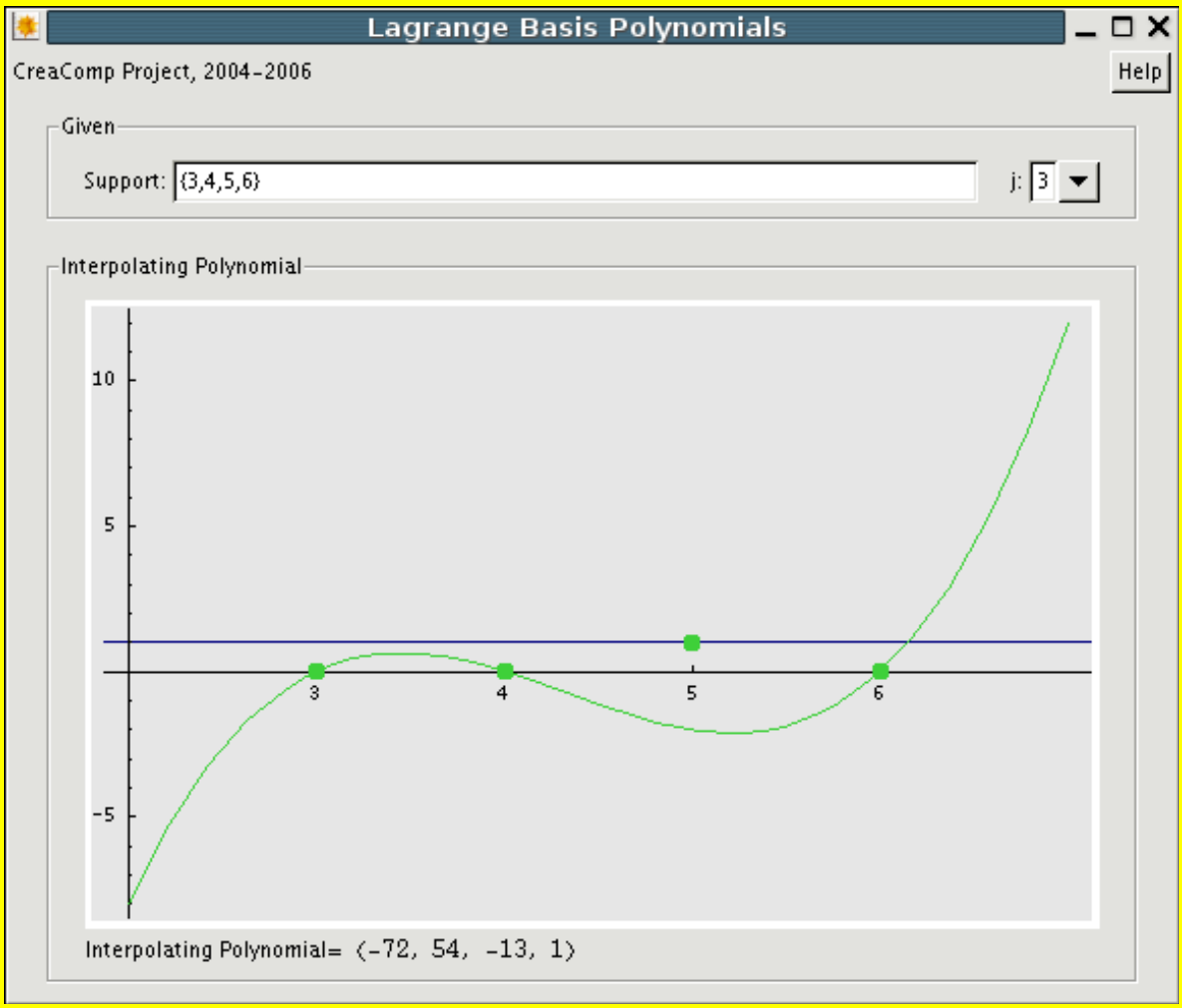
B[j, x, K] := $\prod_{\substack{i=1,\ldots,|x| \\ i \neq j}} \left\langle \frac{}{K} x_i, \frac{1}{K} \right\rangle$]

The algorithm just formulates the conclusion from above in the language of Theorema. It looks slightly more complicated than in a plain mathematical formula, but it can immediately be used for computation. Of course, the polynomial computed and displayed is exactly the same is in the previ−ous experiment!

Try out the new algorithm

Pressing this button will open an interactive widget, in which the user can check the properties of the polynomial B[j,x,K] defined above in

Algorithm["Given Roots"].



The widget has a Help−button in the top−right corner, which explains the intended use of that widget:

This widget should help in finding out how to rigorously define the Lagrange basis polynomials.

Again we have an input field for choosing the "Support". Then we have a "Pull Down"−menu for choosing j. Furthermore, the widget computes and visualizes the polynomial $B[j, x, K]$ as defined in Algorithm["Given Roots"].

The widget updates whenever the support is changed and the RET−key is pressed on the keyboard in the input field or when a new j is selected.

The polynomial $B[j, x, K]$ "fits" at x_i for $i \neq j$, but it needs normalization in order to fit also at x_j.

In order to adjust the evaluation of $B[j, x, K]$ at x_j we need to *normalize*.

```
DoNotUse[Algorithm["Given Roots"]];
Algorithm["Given Roots", any[j, x, K],
  B[j, x, K] := (∏_(i=1,…,|x|, i≠j) ⟨x_i, 1⟩_K) / P[K]]
```

The normalization above is "stupid", but see the result! The widget is the same as the one above, but it refers to the new definnition of $B[j, x, K]$.

Try out the new algorithm

Try to enter the correct normalization at the "□" below and check the new algorithm in the visualization.

```
DoNotUse[Algorithm["Given Roots"]];
Algorithm["Given Roots", any[j, x, K],
  B[j, x, K] := (∏_{i=1,…,|x|, i≠j} ⟨x_i ⊖_K, 1/K⟩) / □]
```

The definition of $B[j, x, K]$ now contains a "□" as an indication that the user needs to fill in something. In this example, the appropriate normalization factor needs to be given. Whatever the user writes instead of the "□" will be used for computation in the widget below.

Try out the new algorithm

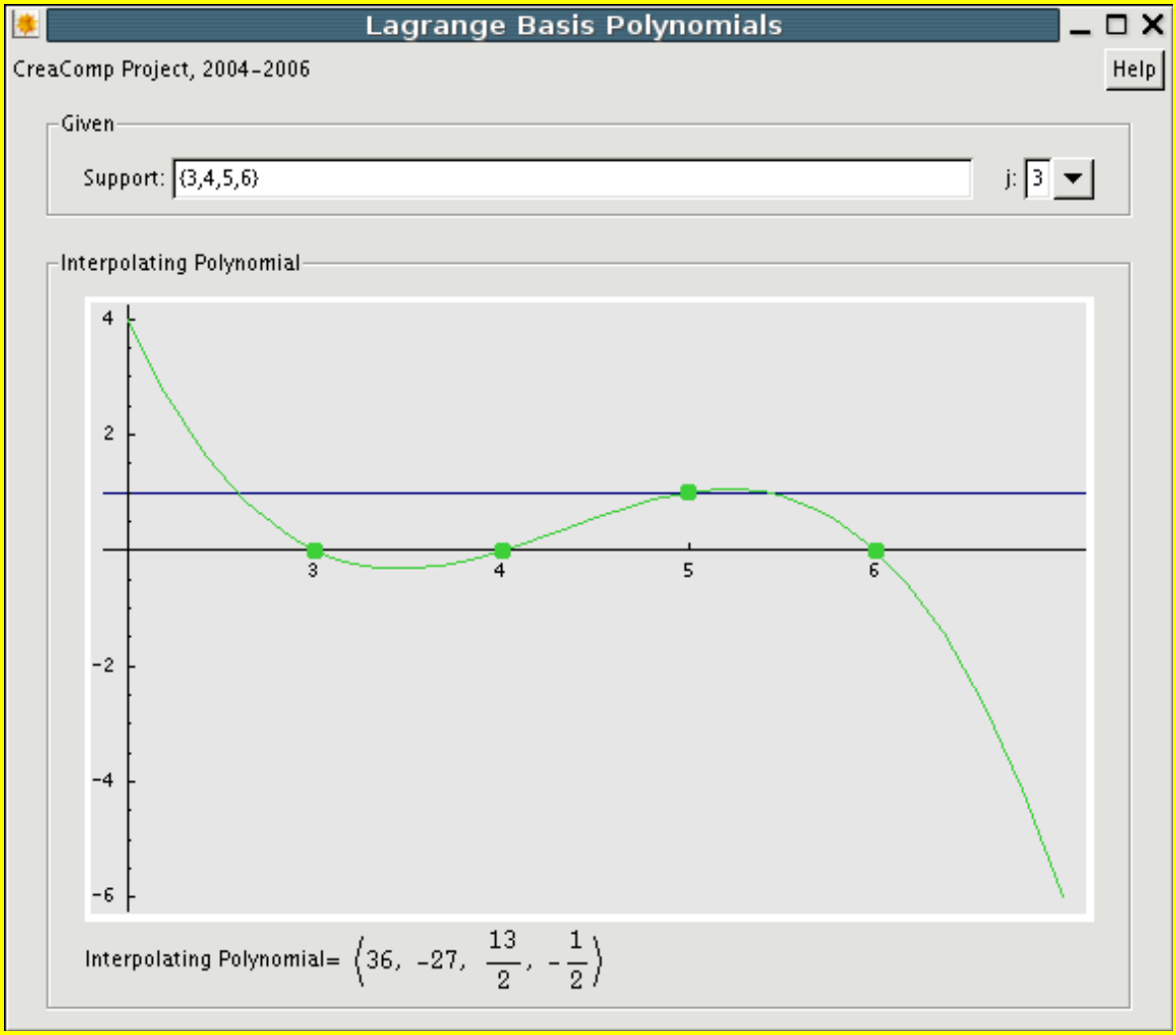
```
DoNotUse[Algorithm["Given Roots"]]
```

At x_j the product of the polynomials $x - x_i$ will evaluate to the product of $x_j - x_i$ for $i \neq j$, so the algorithm below will show the desired behavior. We call $B[j, x, K]$ the j−th *Lagrange basis polynomial* for x (in K).

```
Algorithm["Lagrange Basis Polynomial", any[j, x, K],
  B[j, x, K] := (∏_{i=1,…,|x|, i≠j} ⟨x_i ⊖_K, 1/K⟩) / ∏_{i=1,…,|x|, i≠j} (x_j ⊖_K x_i)]
```

Regardless of all experiments above this is the correct definition of $B[j, x, K]$. Let's examine in the widget.

Try out again



The $B[j, x, K]$ as defined now has the desired properties!

The collection of all Lagrange basis polynomials is then called the *Lagrange basis* for x (in K).

```
Algorithm["Lagrange Basis", any[x, K],
  LagrangeBasis[x, K] := ⟨B[j, x, K] | j=1,...,|x|⟩]
```

We can immediately perform computations with this definition in the Theorema system. We enter a computational session.

```
ComputationalSession["Interpolation"]
```

```
Use[⟨Algorithm["Lagrange Basis Polynomial"],
    Algorithm["Lagrange Basis"], Definition["Polynomial Domain"]⟩]
```

```
B[1, ⟨3, 4, 5⟩, ℚ]
```

$$\left\langle 10,\ \frac{-9}{2},\ \frac{1}{2}\right\rangle$$

```
B[3, ⟨3, 4, 5, 7, 9⟩, ℚ]
```

$$\left\langle \frac{189}{4},\ \frac{-633}{16},\ \frac{187}{16},\ \frac{-23}{16},\ \frac{1}{16}\right\rangle$$

```
LagrangeBasis[⟨3, 4, 5⟩, ℚ]
```

$$\left\langle\left\langle 10,\ \frac{-9}{2},\ \frac{1}{2}\right\rangle, \langle -15, 8, -1\rangle, \left\langle 6,\ \frac{-7}{2},\ \frac{1}{2}\right\rangle\right\rangle$$

```
LagrangeBasis[⟨3, 4, 5, 7, 9⟩, ℚ]
```

$$\left\langle\left\langle \frac{105}{4},\ \frac{-887}{48},\ \frac{227}{48},\ \frac{-25}{48},\ \frac{1}{48}\right\rangle, \left\langle -63,\ \frac{248}{5},\ \frac{-206}{15},\ \frac{8}{5},\ \frac{-1}{15}\right\rangle,\right.$$
$$\left.\left\langle \frac{189}{4},\ \frac{-633}{16},\ \frac{187}{16},\ \frac{-23}{16},\ \frac{1}{16}\right\rangle, \left\langle \frac{-45}{4},\ \frac{161}{16},\ \frac{-155}{48},\ \frac{7}{16},\ \frac{-1}{48}\right\rangle, \left\langle \frac{7}{4},\ \frac{-389}{240},\ \frac{131}{240},\ \frac{-19}{240},\ \frac{1}{240}\right\rangle\right\rangle$$

```
eval[B[3, ⟨3, 4, 5, 7, 9⟩, ℚ], 5]
 P[ℚ]
```

1

```
eval[B[3, ⟨3, 4, 5, 7, 9⟩, ℚ], 9]
 P[ℚ]
```

0

```
eval[B[3, ⟨3, 4, 5, 7, 9⟩, ℚ], 1]
 P[ℚ]
```

18

```
EndComputationalSession[]
```

From the basis representation we now get an easy interpolation algorithm: We call the interpolating polynomial for x and a (over K) computed in this way the *Lagrange polynomial* for x and a (over K) abbreviated as $L[x, a, K]$.

Algorithm["Lagrange Polynomial", any[x, a, K],

L[x, a, K] := where[B = LagrangeBasis[x, K], $\sum_{j=1,\ldots,|x|}$ a_j $\cdot_{P[K]}$ B_j]]

We can immediately perform computations with this definition in the Theorema system. We re−enter the computational session from above.

```
ComputationalSession["Interpolation"]
```

```
Use[Algorithm["Lagrange Polynomial"]]
```

```
L[⟨3, 4, 5⟩, ⟨2, 4, 0⟩, ℚ]
```

$\langle -40, 23, -3 \rangle$

```
eval[L[⟨3, 4, 5⟩, ⟨2, 4, 0⟩, ℚ], 3]
P[Q]
```

2

The interpolation polynomial for $\langle 3, 4, 5 \rangle$ and $\langle 2, 4, 0 \rangle$ evaluates to 2 at $x_1 = 3$, as desired.

```
⟨eval[L[⟨3, 4, 5⟩, ⟨2, 4, 0⟩, ℚ], x] | ⟩
P[Q]                                  x∈{3,4,5}
```

$\langle 2, 4, 0 \rangle$

The interpolation polynomial for x and a evaluates to a_i at x_i, as desired.

```
L[⟨3, 4, 5, 7, 9⟩, ⟨2, 4, 0, −3, 5⟩, ℚ]
```

$\left\langle -157, \frac{2463}{20}, \frac{-3967}{120}, \frac{73}{20}, \frac{-17}{120} \right\rangle$

```
eval[L[⟨3, 4, 5, 7, 9⟩, ⟨2, 4, 0, −3, 5⟩, ℚ], 3]
P[Q]
```

2

```
eval[L[⟨3, 4, 5, 7, 9⟩, ⟨2, 4, 0, −3, 5⟩, Q], 7]
P[Q]
```

```
−3
```

```
⟨eval[L[⟨3, 4, 5, 7, 9⟩, ⟨2, 4, 0, −3, 5⟩, Q], x] | ⟩
P[Q]                                              x∈{3,4,5,7,9}
```

```
⟨2, 4, 0, −3, 5⟩
```

```
EndComputationalSession[]
```

Lagrange interpolation should now be understood.

Neville Interpolation

The main disadvantage of Lagrange interpolation shows up if *new support points* are adjoined. As we saw already in our experiments above, *all Lagrange basis polynomials need to be recomputed* once the tuple of support points changes.

Try out again

The widget from above. Just to re−compute Lagrange basis polynomials and observe changes when the support changes.

The interpolating polynomial for x and a (over K), however, shows an interesting recursive property, which will lead to a *recursive interpolation algorithm*. Namely, there is a property connecting the interpolating polynomial for $\langle x_1, \ldots, x_n \rangle$ and $\langle a_1, \ldots, a_n \rangle$ with the interpolating polynomial for $\langle x_1, \ldots, x_{n-1} \rangle$ and $\langle a_1, \ldots, a_{n-1} \rangle$ and the interpolating polynomial for $\langle x_2, \ldots, x_n \rangle$ and $\langle a_2, \ldots, a_n \rangle$, respectively. By this property, the computation of the interpolation polynomial for tuples of length n goes back to the computation of *two interpolation polynomials* for tuples of length $n-1$, respectively. Finally, the computation of the 0-degree interpolation polynomial through exactly one given point needed as the recursion base is easy.

```
Algorithm["Neville", any[c, α, x, a, K],
  NP[⟨c⟩, ⟨α⟩, K] = const[α]
  NP[x, a, K] = where[n = |x|,
    ((x ⊘ const[x1]) ⊛ NP[x1→⬚, a1→⬚, K]) ⊘
      ((x ⊘ const[xn]) ⊛ NP[xn→⬚, an→⬚, K]) // (xn − x1)]]
```

Again, the Theorema language immediately serves for executing the algorithm defined by the above formula. The notation $x_{1\to\square}$ for deleting at position 1 in tuple x and similar notation used above is assumed to be known from the introduction to the Theorema language.

```
ComputationalSession["Interpolation"]
```

```
Use[Algorithm["Neville"]]
```

```
NP[⟨3, 4, 5⟩, ⟨2, 4, 0⟩, ℚ]
```

$$\langle -40, 23, -3 \rangle$$

```
NP[⟨3, 4, 5, 7, 9⟩, ⟨2, 4, 0, −3, 5⟩, ℚ]
```

$$\left\langle -157, \frac{2463}{20}, \frac{-3967}{120}, \frac{73}{20}, \frac{-17}{120} \right\rangle$$

The new algorithm computes the same interpolation polynomials as did Lagrange interpolation above! No surprise, the interpolation polynomial is uniquely determined!

```
EndComputationalSession[]
```

In contrast to Lagrange interpolation, where we tried to invent the algorithm through guided experiments, we just presented the recursive algorithm for Neville interpolation. We now go for a rigorous proof of the recursive property used in the algorithm. In fact, we will prove the key property fully automatically using the Theorema system.

So, the algorithm seems to work correctly in the examples shown above. We now want to provide a *proof of its correctness*. Given a domain of coefficients K and tuples x and a (of same length). For tuples $\langle c \rangle$ and $\langle \alpha \rangle$ of length 1 it is easy to show that $\underset{\wp[K]}{\text{const}}[\alpha]$ satisfies the interpolation conditions, hence the algorithm's recursion base is easy. Now suppose that P is the interpolating polynomial for $\langle x_1, \ldots, x_{n-1} \rangle$ and $\langle a_1, \ldots, a_{n-1} \rangle$ and Q is the interpolating polynomial for $\langle x_2, \ldots, x_n \rangle$ and $\langle a_2, \ldots, a_n \rangle$, in other words

Assumption["Interpolation",

$\underset{P[K]}{\text{eval}}[P, x_1] = a_1$ "1"

$\underset{P[K]}{\text{eval}}[Q, x_n] = a_n$ "n"

$i \neq 1 \wedge i \neq n \Rightarrow \left(\underset{P[K]}{\text{eval}}[P, x_i] = a_i\right) \bigwedge \left(\underset{P[K]}{\text{eval}}[Q, x_i] = a_i\right)$ "inner"]

Ideally, a proof by induction over the tuples x and a would be performed. However, we concentrate only on the induction step, hence we assume the induction hypothesis and prove the induction step. We implicitly fix x and a and use n for $|x|$.

We need to prove that *the combination* of P and Q as given in Neville's algorithm is the interpolating polyno−mial for $\langle x_1, \ldots, x_n\rangle$ and $\langle a_1, \ldots, a_n\rangle$, i.e. it evaluates to a_i at x_i (for all $i = 1, \ldots, |x|$).

Theorem["Neville",

$\underset{P[K]}{\text{eval}}\left[\left(\left(x \underset{P[K]}{\bar{}} \underset{P[K]}{\text{const}}[x_1]\right) \underset{P[K]}{*} Q \underset{P[K]}{\bar{}} \left(x \underset{P[K]}{\bar{}} \underset{P[K]}{\text{const}}[x_n]\right)\right) \underset{P[K]}{*} P\right) \underset{P[K]}{/} \left(x_n \underset{K}{\bar{}} x_1\right), x_i\right] = a_i\Big]$

This is the correctness statement for the recursive algorithm of Neville.

Of course, we cannot prove this theorem without any knowldge about the concepts involved. In particular, we need information about polynomial evaluation:

Lemma["Evaluation", any[p, q, n, a, K],

$\underset{P[K]}{\text{eval}}\left[p \underset{P[K]}{/} n, a\right] = \underset{P[K]}{\text{eval}}[p, a] \underset{K}{/} n$ "/"

$\underset{P[K]}{\text{eval}}\left[p \underset{P[K]}{\bar{}} q, a\right] = \underset{P[K]}{\text{eval}}[p, a] \underset{K}{\bar{}} \underset{P[K]}{\text{eval}}[q, a]$ "−"

$\underset{P[K]}{\text{eval}}\left[p \underset{P[K]}{*} q, a\right] = \underset{P[K]}{\text{eval}}[p, a] \underset{K}{*} \underset{P[K]}{\text{eval}}[q, a]$ "∗"]

$\underset{P[K]}{\text{eval}}\left[x_{P[K]}, a\right] = a$ "x"

$\underset{P[K]}{\text{eval}}\left[\underset{P[K]}{\text{const}}[p], a\right] = p$ "const"

It is completely clear, though instructive for students, that in most of the cases a proof cannot be given from zero knowledge. Also Theorema needs explicit knowledge that is available in a proof. This is knowledge about the evaluation of polynomials, more specifically, it is knowledge how evaluation interacts with polynomial arithmetic operations. In the spirit of theory exploration, these properties are typically studied when the notion of polynomial evaluation is introduced. Basically, the above proper−ties tell that evaluation is a homomorphism from the domain of polynomials over K into K.

Secondly, we need knowledge about arithmetic in the coefficient field K:

Lemma["Field arithmetic", any[x, y, z, i, j, K],

$$(x_i \underset{K}{-} x_j) \underset{K}{*} z = \begin{cases} 0 & \Leftarrow \; i = j \\ x_i \underset{K}{*} z \underset{K}{-} x_j \underset{K}{*} z & \Leftarrow \; \text{otherwise} \end{cases} \qquad \text{"dist"}$$

$$0 \underset{K}{-} x = \underset{K}{-} x \qquad \text{"0−"}$$

$$x \underset{K}{-} 0 = x \qquad \text{"−0"}$$

$$\underset{K}{-}\left((x \underset{K}{-} y) \underset{K}{*} z\right) \Big/_{K} (y \underset{K}{-} x) = z \qquad \text{"cancel1"}$$

$$\left(x \underset{K}{*} z \underset{K}{-} y \underset{K}{*} z\right) \Big/_{K} (x \underset{K}{-} y) = z \qquad \text{"cancel2"}$$

$$(x \underset{K}{-} y) \underset{K}{-} (x \underset{K}{-} z) = z \underset{K}{-} y \qquad \text{"−"}$$

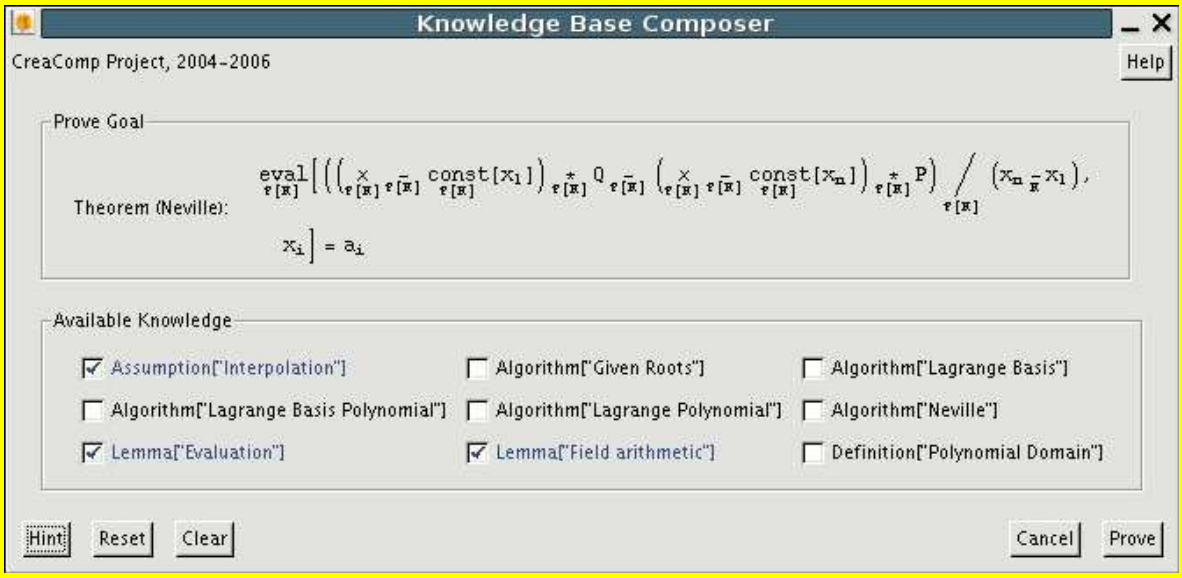
]

We use a simple simplification prover that has no built−in knowledge about field arithmetic, therefore we need to give these elementary properties. This might look unusual, but it has the advantage that all simplification steps will be shown in detail during the proof.

We use the well−known button bar as prove interface to the Theorema prover. "Prove it!" initiates the proof dialog for composing the knowledge base and then starts an automatic proof. "Abort" allows to abort, and "Show" allows to show an aborted incomplete proof. The successful proof should appear in a separate window after a few seconds. If something goes wrong, an off−line version of the suc−cessful proof can be viewed via the "Off−line" button.

Prove it! **Abort** **Show** **Off-line**

Pressing this button will open an interactive widget, in which the user can compose the knowledge base for the proof.



The widget has a Help−button in the top−right corner, which explains the intended use of that widget:

This widget should help in composing the knowledge base for a Theorema proof.

The widget shows the formula to be proven and a collection of all knowledge available in the current Theorema session. Checkboxes allow to select items to go into the knowledge base for the proof.

If multiple proof attempts for a formula are made, the widget remembers the knowledge chosen in the

previous proof and pre−selects those itmes at startup.

The bottom area of the widget is made up of buttons for knowledge selection in the left part and buttons for passing on to the proof in the right. The "Hint"−button selects the knowledge needed for a successful proof, "Reset" resets the selection to the startup configuration, "Clear" unselects all. The "Cancel"−button cancels the current proof attempt, whereas "Prove" initiates a fully automatic proof using the chosen knowledge base.

A few seconds after pressing the "Prove"−button, a new appears on the screen containing a com−plete proof of the above theorem. The proof contains all details and it is structured in hierarchically nested cells in a Mathematica notebook. Nested cells can be opened/closed in order to show/hide parts of the proof (we cannot resemble this feature in the printed version, of course).

We simplify (Theorem (Neville)).

Simplification of the lhs term:

$$\underset{P[K]}{\text{eval}}\left[\left(\left(x \underset{P[K]}{\ast} \overline{P[K]}\ \underset{P[K]}{\text{const}}[x_1]\right) \underset{P[K]}{\ast} Q \underset{P[K]}{\ast} \overline{P[K]}\ \left(x \underset{P[K]}{\ast} \overline{P[K]}\ \underset{P[K]}{\text{const}}[x_n]\right) \underset{P[K]}{\ast} P\right) \Big/ \underset{P[K]}{} (x_n \underset{K}{-} x_1),\ x_i\right] = \quad \text{by (Lemma (Evaluation): /)}$$

$$\underset{P[K]}{\text{eval}}\left[\left(x \underset{P[K]}{\ast} \overline{P[K]}\ \underset{P[K]}{\text{const}}[x_1]\right) \underset{P[K]}{\ast} Q \underset{P[K]}{\ast} \overline{P[K]}\ \left(x \underset{P[K]}{\ast} \overline{P[K]}\ \underset{P[K]}{\text{const}}[x_n]\right) \underset{P[K]}{\ast} P,\ x_i\right] \Big/ (x_n \underset{K}{-} x_1) = \quad \text{by (Lemma (Evaluation): −)}$$

Every individual simplification step is shown and explained, by which knowledge the simplification step is justified. Formula references are hyperlinks, which display the referenced formula when clicked.

We leave out details …

$$\left((x_i \underset{K}{-} x_1) \underset{K}{\ast} \underset{P[K]}{\text{eval}}[Q,\ x_i] \underset{K}{-} (x_i \underset{K}{-} x_n) \underset{K}{\ast} \underset{P[K]}{\text{eval}}[P,\ x_i]\right) \Big/ \underset{K}{} (x_n \underset{K}{-} x_1)\rfloor$$

Simplification of the rhs term:

$$a_i\rfloor$$

Hence, it is sufficient to prove:

(1) $$\left((x_i \underset{K}{-} x_1) \underset{K}{\ast} \underset{P[K]}{\text{eval}}[Q,\ x_i] \underset{K}{-} (x_i \underset{K}{-} x_n) \underset{K}{\ast} \underset{P[K]}{\text{eval}}[P,\ x_i]\right) \Big/ \underset{K}{} (x_n \underset{K}{-} x_1) = a_i.$$

We prove (1) by case distinction using (Lemma (Field arithmetic): dist).

Case 1:

(2) $i = 1$.

Hence, we have to prove

(3) $$\left(0 \underset{K}{-} (x_i \underset{K}{-} x_n) \underset{K}{\ast} \underset{P[K]}{\text{eval}}[P,\ x_i]\right) \Big/ \underset{K}{} (x_n \underset{K}{-} x_1) = a_i.$$

A proof by simplification of (3) works.

Simplification of the lhs term:

$$\left(0 \underset{K}{-} (x_i \underset{K}{-} x_n) \underset{K}{\ast} \underset{P[K]}{\text{eval}}[P,\ x_i]\right) \Big/ \underset{K}{} (x_n \underset{K}{-} x_1) = \quad \text{by (2)}$$

Details skipped …

$$a_1\rfloor$$

Simplification of the rhs term:

$$a_i = \quad \text{by (2)}$$

$$a_1\rfloor$$

Case 2:

(4) $i \neq 1$.

Hence, we have to prove

(6) $\left(\left(x_i \underset{K}{*}_{P[K]} \text{eval}[Q, x_i]\right) \underset{K}{-} x_1 \underset{K}{*}_{P[K]} \text{eval}[Q, x_i]\right) \underset{K}{-} \left(x_i \underset{K}{-} x_n\right) \underset{K}{*}_{P[K]} \text{eval}[P, x_i]\Big) \Big/_K \left(x_n \underset{K}{-} x_1\right) = a_i$.

We prove (6) by case distinction using (Lemma (Field arithmetic): dist).

Case 1:

(7) $i = n$.

Hence, we have to prove

(8) $\left(\left(x_i \underset{K}{*}_{P[K]} \text{eval}[Q, x_i]\right) \underset{K}{-} x_1 \underset{K}{*}_{P[K]} \text{eval}[Q, x_i]\right) \underset{K}{-} 0\Big) \Big/_K \left(x_n \underset{K}{-} x_1\right) = a_i$.

We simplify (8).

Simplification of the lhs term:

$\left(\left(x_i \underset{K}{*}_{P[K]} \text{eval}[Q, x_i]\right) \underset{K}{-} x_1 \underset{K}{*}_{P[K]} \text{eval}[Q, x_i]\right) \underset{K}{-} 0\Big) \Big/_K \left(x_n \underset{K}{-} x_1\right) =$ by (Lemma (Field arithmetic): −0)

$\left(x_i \underset{K}{*}_{P[K]} \text{eval}[Q, x_i]\right) \underset{K}{-} x_1 \underset{K}{*}_{P[K]} \text{eval}[Q, x_i]\Big) \Big/_K \left(x_n \underset{K}{-} x_1\right)\rfloor$

Simplification of the rhs term:

$a_i\rfloor$

Hence, it is sufficient to prove:

(12) $\left(x_i \underset{K}{*}_{P[K]} \text{eval}[Q, x_i]\right) \underset{K}{-} x_1 \underset{K}{*}_{P[K]} \text{eval}[Q, x_i]\Big) \Big/_K \left(x_n \underset{K}{-} x_1\right) = a_i$.

Formula (12), using (7), is implied by:

$\left(x_n \underset{K}{*}_{P[K]} \text{eval}[Q, x_n]\right) \underset{K}{-} x_1 \underset{K}{*}_{P[K]} \text{eval}[Q, x_n]\Big) \Big/_K \left(x_n \underset{K}{-} x_1\right) = a_n$,

which, using (Assumption (Interpolation): n), is implied by:

$\left(x_n \underset{K}{*} a_n \underset{K}{-} x_1 \underset{K}{*} a_n\right) \Big/_K \left(x_n \underset{K}{-} x_1\right) = a_n$,

which, using (Lemma (Field arithmetic): cancel2), is implied by:

(14) $a_n = a_n$.

By reflexivity of =, formula (14) is proved.

Case 2:

(9) $i \neq n$.

Hence, we have to prove

(11) $\left(\left(x_i \underset{K}{*}_{P[K]} \text{eval}[Q, x_i]\right) \underset{K}{-} x_1 \underset{K}{*}_{P[K]} \text{eval}[Q, x_i]\right) \underset{K}{-} \left(x_i \underset{K}{*}_{P[K]} \text{eval}[P, x_i]\right) \underset{K}{-} x_n \underset{K}{*}_{P[K]} \text{eval}[P, x_i]\Big)\Big) \Big/_K \left(x_n \underset{K}{-} x_1\right) = a_i$.

Formula (9), by (Assumption (Interpolation): inner), implies:

(15) $i \neq 1 \Rightarrow \left(\text{eval}_{P[K]}[P, x_i] = a_i\right) \bigwedge \left(\text{eval}_{P[K]}[Q, x_i] = a_i\right)$.

Formula (4), by (15), implies:

(16) $\left(\text{eval}_{P[K]}[P, x_i] = a_i\right) \bigwedge \left(\text{eval}_{P[K]}[Q, x_i] = a_i\right)$.

We simplify (11).

Simplification of the lhs term:

$\left(\left(x_i \underset{K}{*}_{P[K]} \text{eval}[Q, x_i]\right) \underset{K}{-} x_1 \underset{K}{*}_{P[K]} \text{eval}[Q, x_i]\right) \underset{K}{-} \left(x_i \underset{K}{*}_{P[K]} \text{eval}[P, x_i]\right) \underset{K}{-} x_n \underset{K}{*}_{P[K]} \text{eval}[P, x_i]\Big)\Big) \Big/_K \left(x_n \underset{K}{-} x_1\right) =$ by (16.1)

$\left(\left(x_i \underset{K}{*}_{P[K]} \text{eval}[Q, x_i]\right) \underset{K}{-} x_1 \underset{K}{*}_{P[K]} \text{eval}[Q, x_i]\right) \underset{K}{-} \left(x_i \underset{K}{*} a_i \underset{K}{-} x_n \underset{K}{*} a_i\right)\Big) \Big/_K \left(x_n \underset{K}{-} x_1\right) =$ by (16.2)

$$\left(\left(x_i \underset{K}{*} a_i \underset{K}{-} x_1 \underset{K}{*} a_i\right) \underset{K}{-} \left(x_i \underset{K}{*} a_i \underset{K}{-} x_n \underset{K}{*} a_i\right)\right) \Big/_{K} \left(x_n \underset{K}{-} x_1\right)\rfloor$$

Simplification of the rhs term:

$$a_i\rfloor$$

Hence, it is sufficient to prove:

(17) $$\left(\left(x_i \underset{K}{*} a_i \underset{K}{-} x_1 \underset{K}{*} a_i\right) \underset{K}{-} \left(x_i \underset{K}{*} a_i \underset{K}{-} x_n \underset{K}{*} a_i\right)\right) \Big/_{K} \left(x_n \underset{K}{-} x_1\right) = a_i .$$

Formula (17), using (Lemma (Field arithmetic): −), is implied by:

$$\left(x_n \underset{K}{*} a_i \underset{K}{-} x_1 \underset{K}{*} a_i\right) \Big/_{K} \left(x_n \underset{K}{-} x_1\right) = a_i ,$$

which, using (Lemma (Field arithmetic): cancel2), is implied by:

(18) $$a_i = a_i .$$

By reflexivity of =, formula (18) is proved.

□

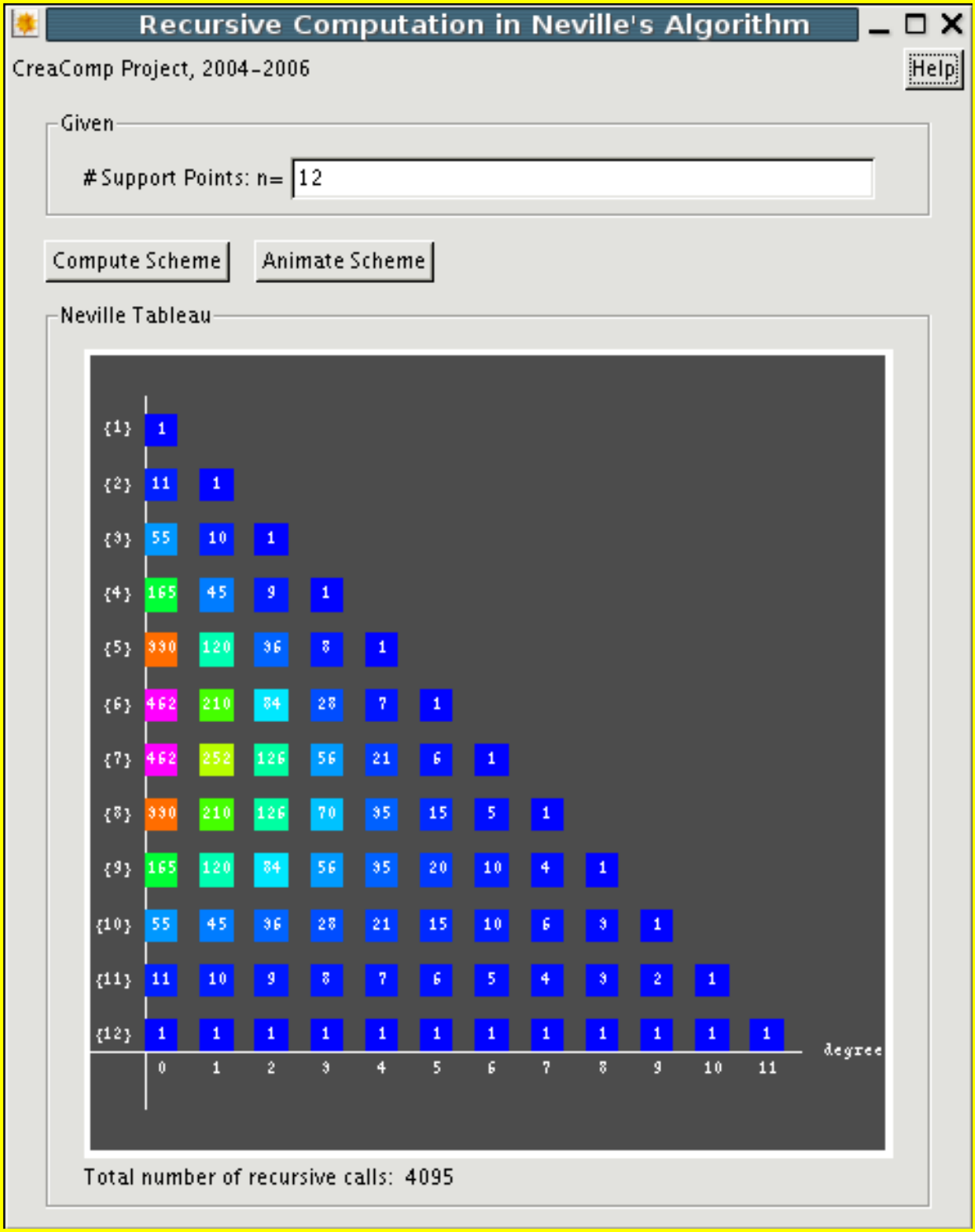
Performance of Neville's Algorithm

We now want to analyze the performance of the above recursive algorithm. If we run the recursive program as written in Algorithm["Neville"], we see that many of the recursive function calls will be executed multiple times with the same input! The recursion flow follows the arrows from right to left in the so-called *Neville-Tab−leau*. We abbreviate $\mathrm{NP}[\langle x_i, \ldots, x_j\rangle, \langle a_i, \ldots, a_j\rangle, K]$ by $\mathrm{NP}_{i,j}$.

degree →	0	1	⋯	$n-2$	$n-1$
$\langle a_1\rangle = \mathrm{NP}_1$	↖				
$\langle a_2\rangle = \mathrm{NP}_2$	←	$\mathrm{NP}_{1,2}$	⋱		
⋮		⋮			
⋮		⋮	⋯	↖	
$\langle a_{n-1}\rangle = \mathrm{NP}_{n-1}$		⋮	⋱	← $\mathrm{NP}_{1,n-1}$	
	↖	⋮	⋮	↖	↖
$\langle a_n\rangle = \mathrm{NP}_n$	←	$\mathrm{NP}_{n-1,n}$	⋯	← $\mathrm{NP}_{2,n}$	← $\mathrm{NP}_{1,n}$

Performance Neville Algorithm I

Pressing this button will open an interactive widget, in which the user can investigate performance issues of the recursive Neville algorithm.



The widget has a Help−button in the top−right corner, which explains the intended use of that widget:

This widget should help in exploring performance issues in the recursive Neville algorithm.

The widget has an input field for choosing the number of support points n *and a graphical visualiza− tion of the Neville tableau. The concrete values to be interpolated are not of interest, though they are shown along the second axis.*

Each entry in the tableau is symbolized as a colored square carrying the number of calls of that particular "cell" during the execution of the recursive algorithm. Initially, all numbers are 0, of course. The color of a cell turns from dark blue through green, yellow, and red to pink depending on the number of recursive calls during execution.

Pressing "Compute Scheme" re−computes and re−colors the entire scheme in one stroke. The same happens when hitting the RET−key of the keyboard in the input field for n.

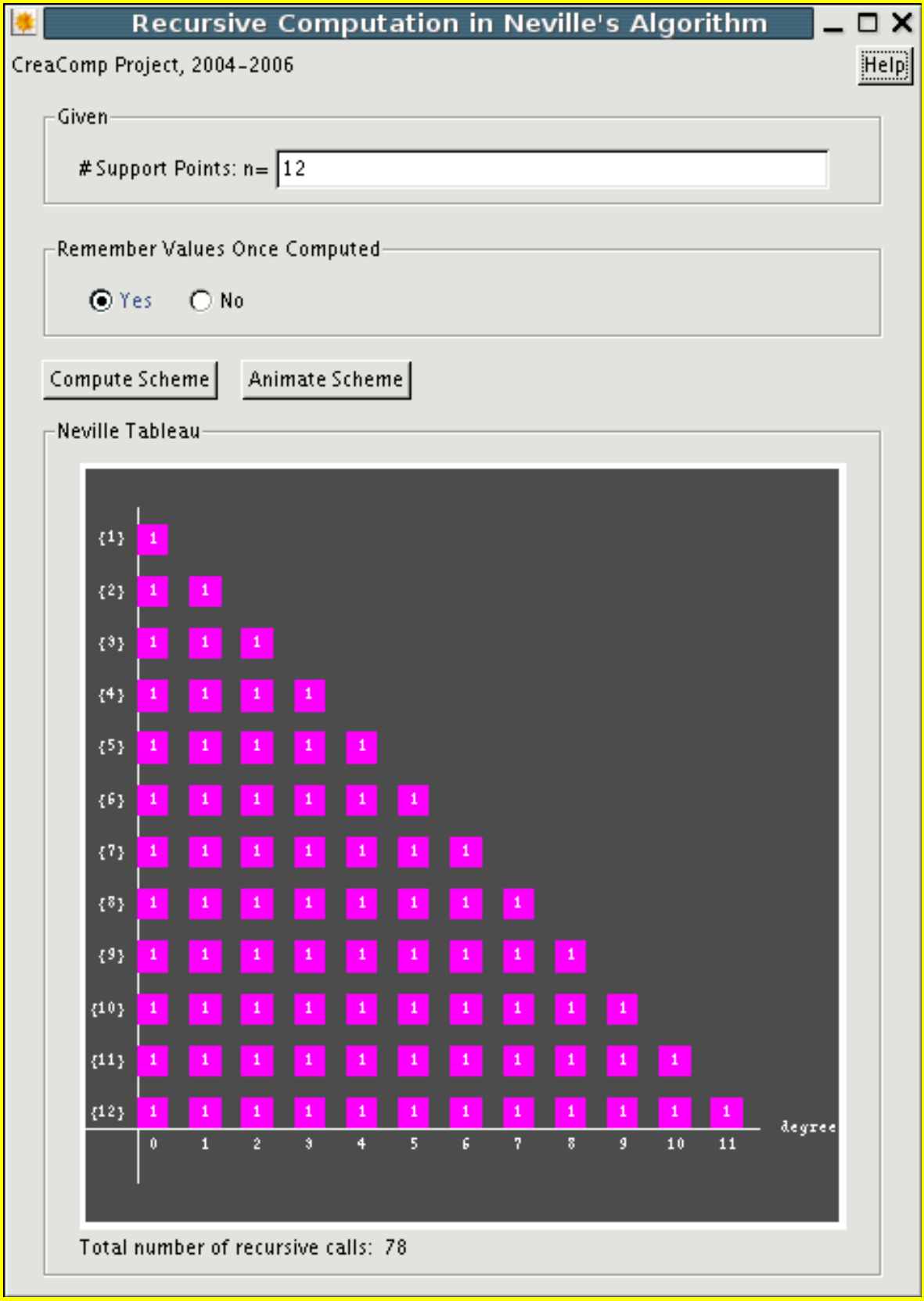
Pressing "Animate Scheme" re−computes and re−colors the entire scheme step by step.

The widget shows an impressive number of superfluous recursive function calls, in particular if n becomes larger. The colors moreover indicate the critical area of the tableau.

Therefore, any reasonable implementation of the Neville algorithm must take this fact into account in order to run efficiently. This can be done, for instance, by remembering values in the recursive calls or by computing the Neville tableaux "left to right" and storing the values in a table. Compare its performance now:

Performance Neville Algorithm II

Pressing this button will open an interactive widget, in which the user can investigate performance issues of the recursive Neville algorithm.



The widget has a Help−button in the top−right corner, which explains the intended use of that widget:

This widget should help in exploring performance issues in the recursive Neville algorithm.

The widget has an input field for choosing the number of support points n, radio buttons for choosing whether the recursive computation should store values once computed, and a graphical visualization

of the Neville tableau. The concrete values to be interpolated are not of interest, though they are shown along the second axis.

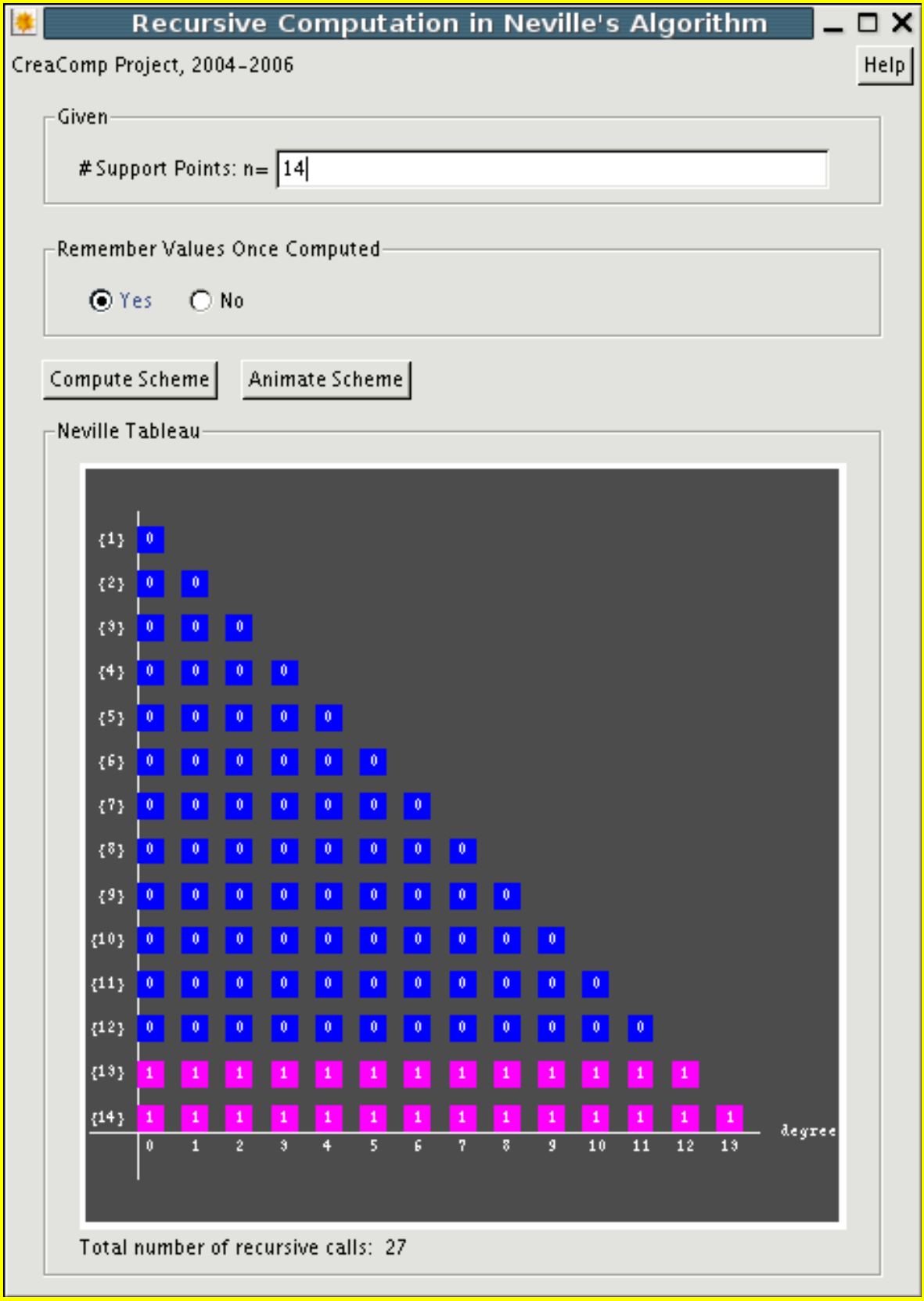
Each entry in the tableau is symbolized as a colored square carrying the number of calls of that particular "cell" during the execution of the recursive algorithm. Initially, all numbers are 0, of course. The color of a cell turns from dark blue through green, yellow, and red to pink depending on the number of recursive calls during execution.

Pressing "Compute Scheme" re−computes and re−colors the entire scheme in one stroke. The same happens when hitting the RET *−key of the keyboard in the input field for* n.

Pressing "Animate Scheme" re−computes and re−colors the entire scheme step by step.

Checkout the behaviour when increasing n.

First we see a dramatic difference in total recursive calls, in particular, there are no superfluous calls anymore. We then switch from $n = 12$ *to* $n = 14$.



Only the new rows in the tableau need to be computed, all others can be reused from before.

Newton Interpolation

Finally, we want to present *another resursive algorithm* for computing the interpolation polynomial. The main idea this time is to find a recursive property that links the interpolating polynomial for $\langle x_1, \ldots, x_n\rangle$ and $\langle a_1, \ldots, a_n\rangle$ with the interpolating polynomial for $\langle x_1, \ldots, x_{n-1}\rangle$ and $\langle a_1, \ldots, a_{n-1}\rangle$ only. If we can find such a relation then we can easily devise a recursive interpolation algorithm, which uses only a *single recursive call* instead of the *more expensive tow-stage recursion* necessary in Neville's algorithm! The key idea in Newton interpolation is as follows: Suppose we have the interpolating polynomial for $\langle x_1, \ldots, x_{n-1}\rangle$ and $\langle a_1, \ldots, a_{n-1}\rangle$, let's abbreviate it by $P_{1,\ldots,n-1}$. Now take another polynomial, say Z_{n-1}, which evaluates to 0 at all x_i for $i = 1, \ldots, n-1$. $P_{1,\ldots,n-1}$ has degree less $n-1$, if Z_{n-1} has degree less n then we can represent $P_{1,\ldots,n}$ as

$$P_{1,\ldots,n} = P_{1,\ldots,n-1} + \Delta_{1,\ldots,n} \cdot Z_{n-1}$$

with $\Delta_{1,\ldots,n} \in K$. We have already investigated earlier the problem of finding Z_{n-1}, i.e. a polynomial with pre−defined roots!
Remember?

Polynomial with given roots

This is the widget already used when exploring Lagrange interpolation.

Thus,

$$Z_{n-1} = \prod_{i=1,\ldots,n-1} (x - x_i)$$

hence

$$\deg[Z_{n-1}] = n - 1$$

and since $\deg[P_{1,\ldots,n-1}] < n-1$ we see that $\Delta_{1,\ldots,n}$ *must be the leading coefficient* of $P_{1,\ldots,n}$.
At the same time, using Neville's algorithm (!), we see that

$$P_{1,\ldots,n} = ((x - x_1)\, P_{2,\ldots,n} + (x - x_n)\, P_{1,\ldots,n-1})\,/\,(x_n - x_1).$$

Comparing the leading coefficients of both sides tells us that

$$\begin{aligned}\Delta_{1,\ldots,n} &= \text{leading\_coefficient}[((x - x_1)\, P_{2,\ldots,n} + (x - x_n)\, P_{1,\ldots,n-1})\,/\,(x_n - x_1)] \\ &= (\Delta_{2,\ldots,n} - \Delta_{1,\ldots,n-1})\,/\,(x_n - x_1)\end{aligned}$$

meaning a recursion formula for the Δ's! We call the Δ's *divided differences*.

```
Algorithm["Divided Differences", any[c, α, x, a, K],
  Δ[⟨c⟩, ⟨α⟩, K] = α
  Δ[x, a, K] = where[n = |x|,
    (Δ[x1→□, a1→□, K] -K Δ[xn→□, an→□, K]) /K (xn -K x1)]]
```

This is a recursive algorithm written Theorema language, the scheme is very much like the one in Neville's algorithm. We can immediately perform computations in the Theorema system.

```
Use[{Built-in["Tuples"], Built-in["Quantifiers"],
    Built-in["Numbers"], Built-in["Number Domains"], Built-in["Connectives"]}]
```

```
ComputationalSession["Interpolation"]
```

```
Use[Algorithm["Divided Differences"]]
```

```
Δ[⟨3⟩, ⟨2⟩, ℚ]
```

$$2$$

```
Δ[⟨3, 4, 5⟩, ⟨2, 4, 0⟩, ℚ]
```

$$-3$$

```
Δ[⟨3, 4, 5, 7, 9⟩, ⟨2, 4, 0, -3, 5⟩, ℚ]
```

$$\frac{-17}{120}$$

```
EndComputationalSession[]
```

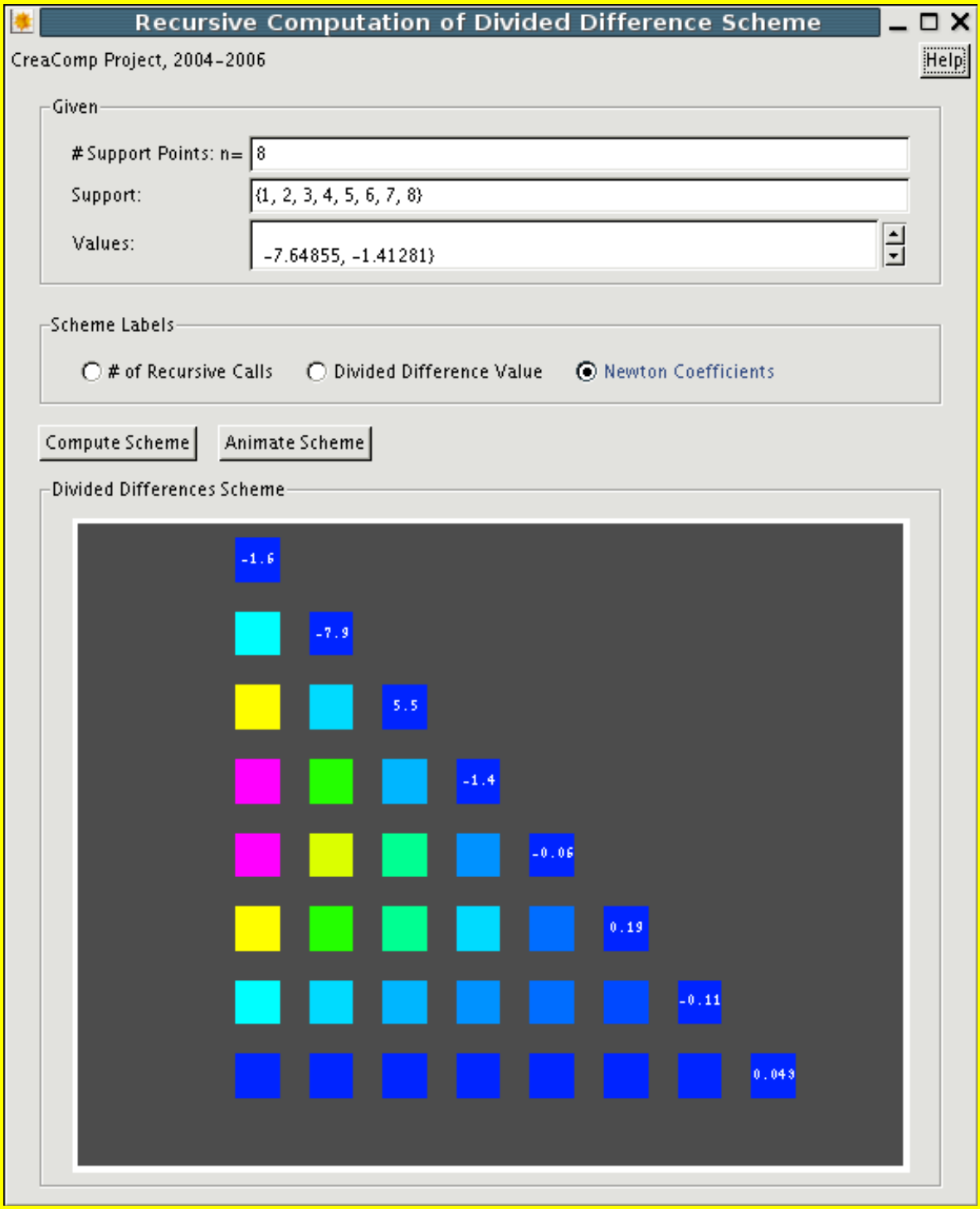
The divided differences recursion flow is the same as the one in Neville's algorithm. Note, that in the final interpolation polynomial we will only need the $\Delta_{1,\ldots,j}$, i.e. the divided differences along the upper diagonal in the tableau.

$$\begin{array}{lccccccc}
\text{degree} \rightarrow & 0 & & 1 & \cdots & n-2 & & n-1 \\
\hline
 & \langle a_1 \rangle = \mathbf{\Delta_1} & & & & & & \\
 & & \nwarrow & & & & & \\
 & \langle a_2 \rangle = \Delta_2 & \leftarrow & \mathbf{\Delta_{1,2}} & & & & \\
 & \vdots & & \vdots & \ddots & & & \\
 & \vdots & & \vdots & \cdots & \nwarrow & & \\
 & \langle a_{n-1} \rangle = \Delta_{n-1} & & \vdots & \ddots & \leftarrow & \mathbf{\Delta_{1,\ldots,n-1}} & \\
 & & \nwarrow & \vdots & \vdots & \nwarrow & & \nwarrow \\
 & \langle a_n \rangle = \Delta_n & \leftarrow & \Delta_{n-1,n} & \cdots & \leftarrow & \Delta_{2,\ldots,n} & \leftarrow & \mathbf{\Delta_{1,\ldots,n}}
\end{array}$$

Compare the performance of the recursive algorithm for computing the divided differences with the original Neville algorithm.

Divided Differences Scheme

Pressing this button will open an interactive widget, in which the user can investigate performance issues of the recursive algorithm for divided differences.



The widget has a Help−button in the top−right corner, which explains the intended use of that widget:

This widget should help in exploring performance issues in the recursive algorithm for computing divided differences.

The widget has an input field for choosing the number of support points n, we automatically choose the range $1, \ldots, n$ as support and choose n random values to be interpolated. The main part is a graphical visualization of the divided differences tableau.

Each entry in the tableau is symbolized as a colored square carrying the number of calls of that particular "cell" during the execution of the recursive algorithm. Initially, all numbers are 0, of course.

The color of a cell turns from dark blue through green, yellow, and red to pink depending on the number of recursive calls during execution. Radio buttons can be used for choosing the labelling of the cells. The cells can be labelled with the number of recursive calls (like in the Neville widget), the actual value of the divided difference in that cell, or only the divided differences needed in the Newton interpolation polynomial.

Pressing "Compute Scheme" re−computes and re−colors the entire scheme in one stroke. The same happens when hitting the RET−key of the keyboard in the input field for n.

Pressing "Animate Scheme" re−computes and re−colors the entire scheme step by step.

Algorithm["Newton Interpolation", any[c, α, x, a, K],

NI[⟨c⟩, ⟨α⟩, K] = $\underset{\mathcal{P}[K]}{\text{const}}$[Δ[⟨c⟩, ⟨$\alpha$⟩, K]]

NI[x, a, K] = where[n = |x|, NI[$x_{n\to\square}$, $a_{n\to\square}$, K] $\underset{\mathcal{P}[K]}{+}$ Δ[x, a, K] $\underset{\mathcal{P}[K]}{\cdot}$ Z[$x_{n\to\square}$, K]]]

Algorithm["Zeros", any[c, α, x, a, K],

$Z[x, K] = \prod_{\substack{\mathcal{P}[K] \\ i=1,\ldots,|x|}} \left\langle \overline{K} x_i, \frac{1}{K} \right\rangle$]

As usual, we immediately explore the new algorithms by doing some computations in the Theorema system.

ComputationalSession["Interpolation"]

Use[⟨Algorithm["Newton Interpolation"], Algorithm["Zeros"], Definition["Polynomial Domain"]⟩]

NI[⟨3⟩, ⟨2⟩, ℚ]

⟨2⟩

NI[⟨3, 4, 5⟩, ⟨2, 4, 0⟩, ℚ]

⟨−40, 23, −3⟩

NI[⟨3, 4, 5, 7, 9⟩, ⟨2, 4, 0, −3, 5⟩, ℚ]

$\left\langle -157, \frac{2463}{20}, \frac{-3967}{120}, \frac{73}{20}, \frac{-17}{120} \right\rangle$

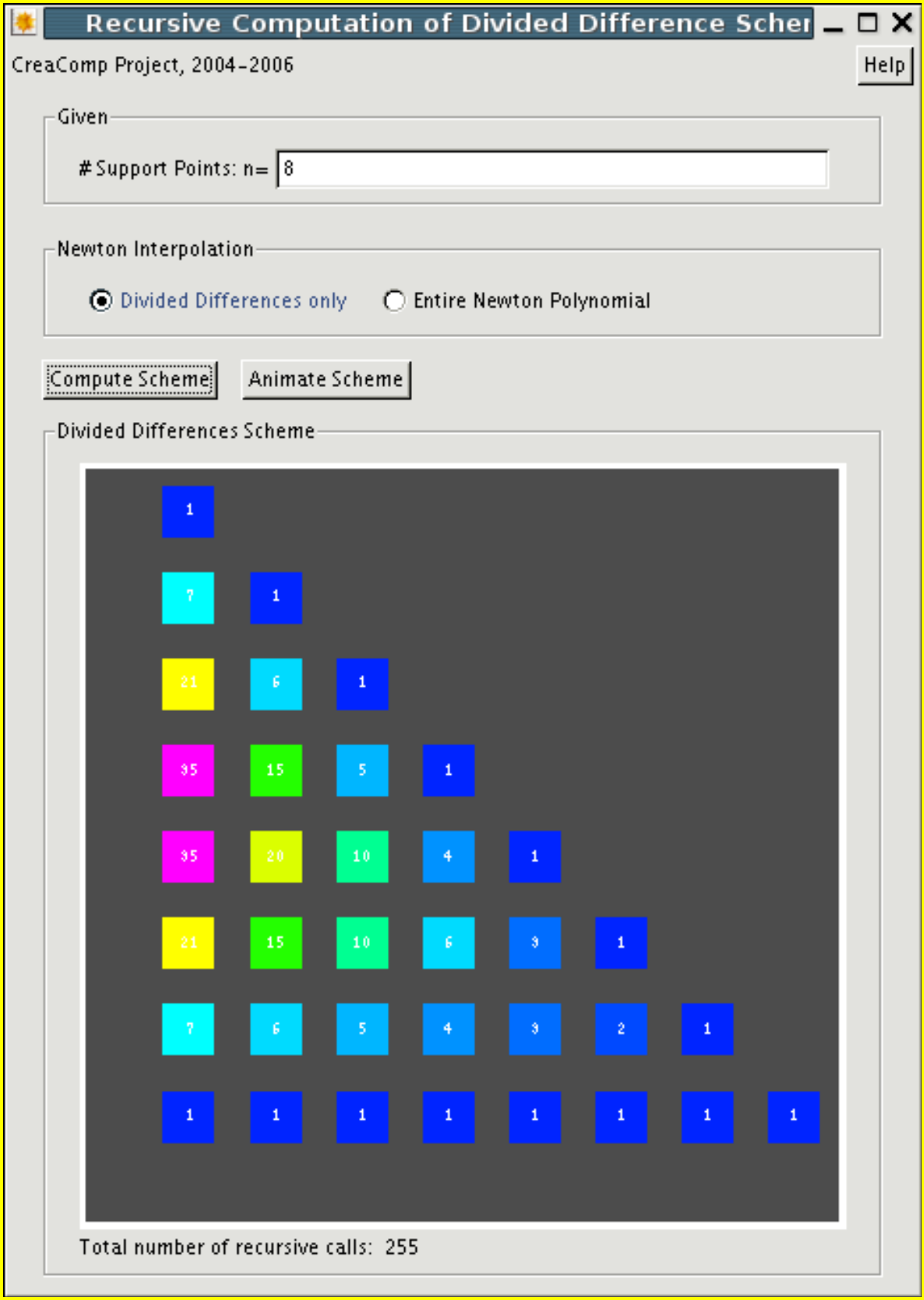
EndComputationalSession[]

The results are the same as with the other interpolation algorithms.

Performance of the recursive algorithms becomes more critical now: The interpolation algorithm is recursive and generates many superfluous recursive calls for computing divided differences, the recursion for comput−ing divided differences is in itself inefficient in this respect. Try out!

Recursive Newton Interpolation

Pressing this button will open an interactive widget, in which the user can investigate performance issues of the recursive algorithm for Newton interpolation.



The widget has a Help−button in the top−right corner, which explains the intended use of that widget:

This widget should help in exploring performance issues in the recursive algorithm for Newton interpo−lation.

The widget has an input field for choosing the number of support points n, we automatically choose

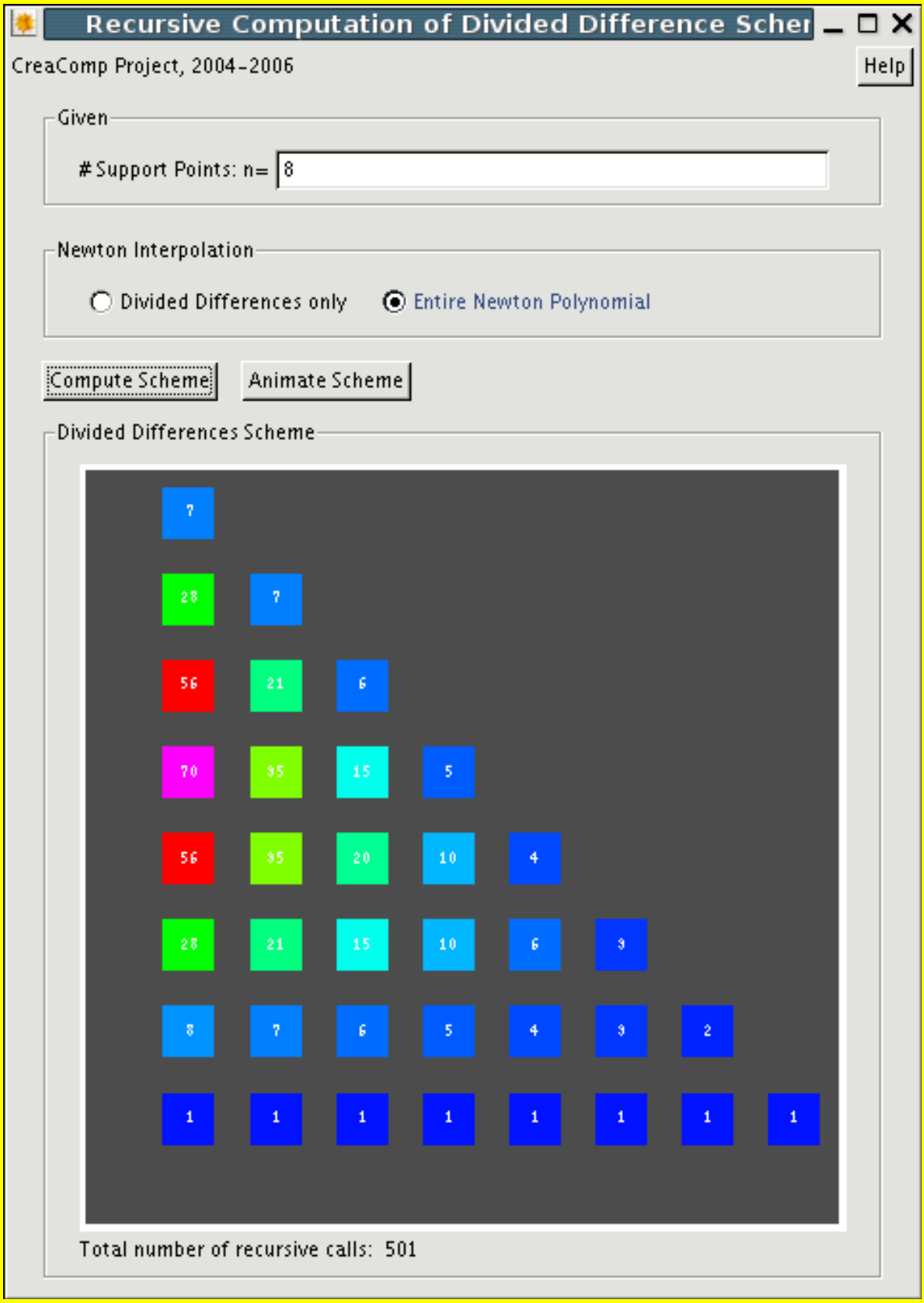
the range 1, …, n as support and choose n random values to be interpolated. The concrete values to be interpolated are not of interest.The main part is a graphical visualization of the divided differences tableau.

Each entry in the tableau is symbolized as a colored square carrying the number of calls of that particular "cell" during the execution of the recursive algorithm. Initially, all numbers are 0, of course. The color of a cell turns from dark blue through green, yellow, and red to pink depending on the number of recursive calls during execution. Radio buttons can be used for selecting the computation to be performed: we can either only compute the divided differences scheme (recursively), or we can compute the entire Newton interpolation polynomial (again recursively).

Pressing "Compute Scheme" re−computes and re−colors the entire scheme in one stroke. The same happens when hitting the RET *−key of the keyboard in the input field for n.*

Pressing "Animate Scheme" re−computes and re−colors the entire scheme step by step.

Compare this pattern to the pattern left after computing the entire Newton polynomial.



Efficient implementation is an important issue!

For evaluating the Newton polynomial, a procedure similar to Horner's algorithm can be used *without actually computing* the Newton polynomial! It only needs the divided differences $\Delta_1, \ldots, \Delta_{1,\ldots n}$ and the tuple x. like in Neville's algorithm, the recursive implementation for computing the divided differences must store the values once computed, i.e. it must compute the entire scheme of divided differences and store it. This will eliminate all superfluous function calls.

Summary

In this unit, we presented the mathematical problem of *interpolation* and several algorithms for computing the uniquely determined interpolation polynomial for given tuples of ordinates and abscissas. The emphasis has been put on correctness of the algorithms and their efficiency.

Commented Unit:
Public Key Cryptography - RSA

After finishing this module you should

** know about the RSA public key encryption scheme.*

Additional Modules

MathWorld : ***FactoringIntegers*** , ***FastComputations***

In the present module we will make use of the following *Mathematica*- and Theorema-commands:

- `Mod`, `PowerMod`, `ListPlot`, `Random`
- Definition, Theorem, Lemma

The header of the unit defines the goal of the unit and links to some related units as well as to the help system for Mathematica commands used in this unit.

Introduction

The introduction gives some definition so that the students know what the unit is about.

Definition:
A cryptosystem is a quintuple (P, C, K, E, D) *where:*
P *is a set of possible cleartext messages*
C *is a set of possible ciphertext messages*
K *is a set of possible keys*
E *is a function from* $P \times K$ *to* C*, called encryption function*
D *is a function from* $C \times K$ *to* P*, called decryption function*
and the following holds:
For every (encryption key) $e \in K$ *there exists a (decryption key)* $d \in K$ *such that for every (plaintext)* $p \in P$ *:* $D(E(p, e), d) = p$.

Of course, it should not be feasible to compute the plaintext from a ciphertext. This should hold, even when the functions E and D are known, as long as the decryption key is unknown.

Definition:
A cryptosystem is called asymmetric or a public key scheme, if there is no algorithm which given $e \in K$ computes in time polynomial in $\log(e)$ a key $d \in K$ such that $D(E(p, e), d) = p$ holds for every $p \in P$. Otherwise the cryptosystem is called symmetric.

Usually, in a symmetric cryptosystem the encryption key e and the corresponding decryption d key are equal. The sender and the receiver share this secret key, which both can use to encrypt or decrypt. In an asymmetric system, knowledge of the encryption key does not help with decryption, because the decryption key can not be computed from the encryption key. In such a system, the encryption key can be published and only the decryption key has to be kept secret. This is the reason, why such cryptosystems are often called public key cryptosystems and the keys the public key (for encryption) and the private key (for decryption). In this unit we are going to study one of the most important asymmetric cryptosystems in use, the RSA cryptosystem.

RSA

Here we introduce the encryption scheme which we will treat in more details in this unit.

One of the earliest asymmetric encryption schemes is RSA, named after its inventors Rivest, Shamir and Adleman (c.f. [RSA78]).

Let p and q be two primes and let $n = p\,q$.
Let $P = C = \{m \in \mathbb{N} \mid m < n \wedge gcd(m, n) = 1\}$ and let
$K = \{(n, e) \mid 1 < e < (p-1)(q-1) \wedge gcd(e, (p-1)(q-1)) = 1\}$.
Define $E(m, (n, e)) := D(m, (n, e)) := m^e \bmod n$.

For a fixed key (n, e) the encryption function E maps numbers in P to numbers in C.

Example

In this example–section we touch on some computational aspects. The students should think about how this computation can be done in a fast way.

The encryption of the message $m = 8$ under the key $(77, 53)$ is

```
c = Mod[8^53, 77]
```

or, much faster:

```
c = PowerMod[8, 53, 77]
```

Why is the second method faster? What is a fast method to compute $m^e \bmod n$ even for large n and e?

> *The following explorative exercises shall guide the student in developing a "square and multiply" algorithm for modular exponentiation.*

Explore:

1. Compute the following numbers using as few multiplications as possible:

$7^{2^1},\ 7^{2^2},\ 7^{2^3},\ 7^{2^4},\ 7^{2^5},\ 7^{2^6},\ 7^{2^7}$

2. Compute the following numbers using only one multiplication and the results from 1.

$7^3,\ 7^5,\ 7^{10},\ 7^{96}$

3. Compute the following numbers using as few multiplications as possible and the results from 1.

$7^{100},\ 7^{200},\ 7^{127}$

How many multiplications are needed?

4. Repeat 1. and 2. but compute the values modulo 32.

5. Write down your method of exponentiation as an algorithm.

6. How many multiplications does your algorithm require for a k-bit exponent in the optimal case, in the average case, and in the worst case?

RSA is a bijective function

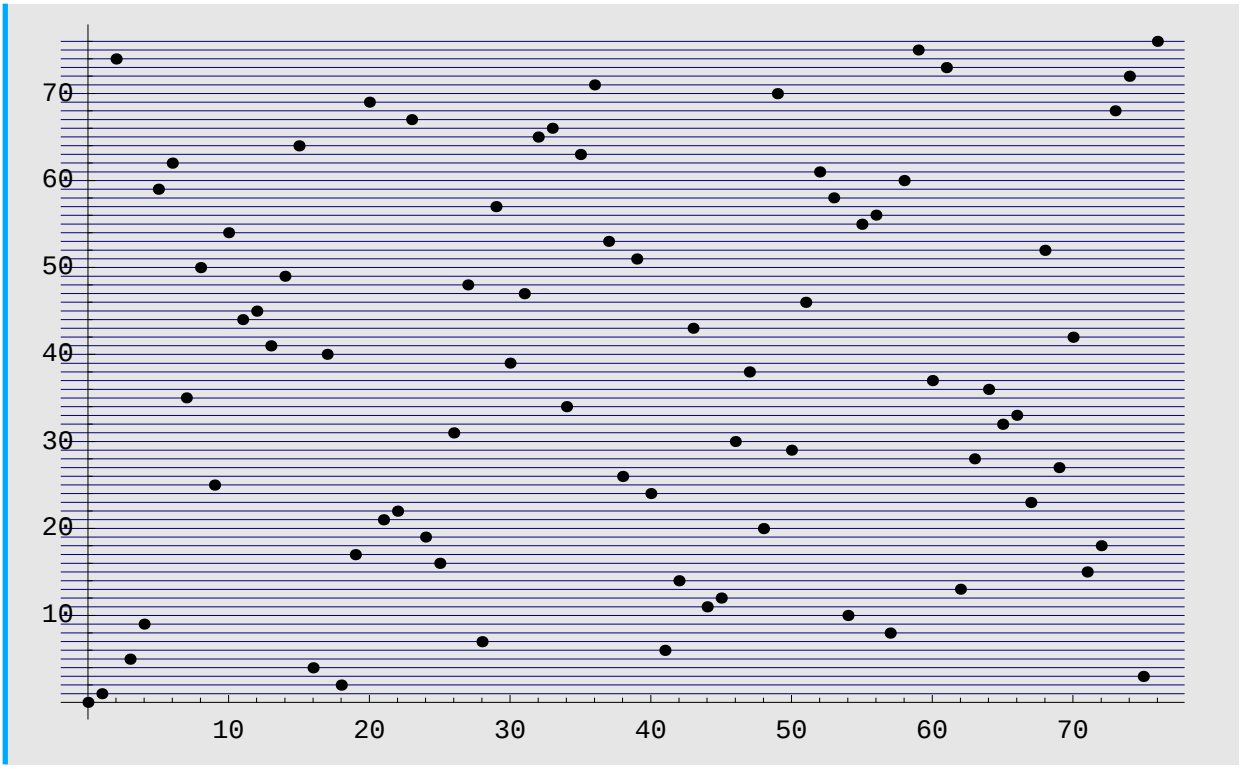
Here we discuss the bijectivity of the encryption function. The plots you can see now are not part of the original unit, but they can be produced by the students at runtime just by evaluating the Mathematica input cells. So the students can change the input and they will see the new results immediately. This is a kind of experiments that should initiate a detailed thinking process.

In order to be able to correctly decrypt a ciphertext, the encryption function has to be injective for every chosen key (n, e). Moreover, it must be feasible to "invert" these functions. Let us look at two small examples, first.

```
p = 7; q = 11; n = p q; e = 53;
GCD[e, (p – 1) (q – 1)]
```

```
1
```

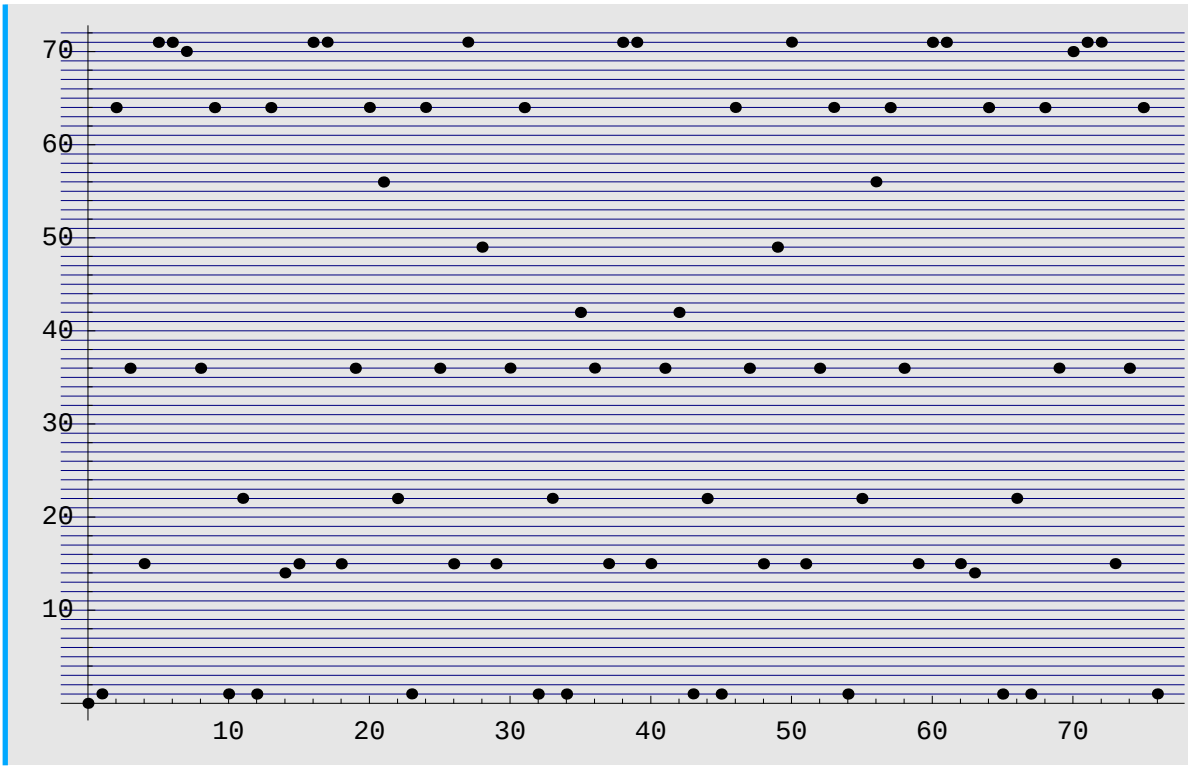
```
ListPlot[Table[{x, PowerMod[x, e, n]}, {x, 0, n – 1}],
    PlotStyle → {PointSize[.01]}, GridLines → {{}, Range[n – 1]}];
```




```
p = 7; q = 11; n = p q; e = 6;
GCD[e, (p – 1) (q – 1)]
```

```
6
```

```
ListPlot[Table[{x, PowerMod[x, e, n]}, {x, 0, n – 1}],
    PlotStyle → {PointSize[.01]}, GridLines → {{}, Range[n – 1]}];
```



In the first example, $\gcd(e, (p-1)(q-1)) = 1$ and from the graph of the encryption function we can see that it is injective (even surjective). In the second example, $\gcd(e, (p-1)(q-1)) = 6 \neq 1$, and it is easy to see that this encryption function is not injective.

In the following section we are going to prove that the condition $\gcd(e, (p-1)(q-1)) = 1$ is sufficient for the RSA encryption function to be injective (and surjective).

RSA looks random (for most keys)

The randomness of an encryption function is an important security feature. If the image of an encryption function has too much structure this can be used for so called repeated encryption attacks to get the message or worse the private keys.

So we demonstrate in examples how the image of the encryption function should look like and how it must not look like.

```
p = Prime[20] ; q = Prime[30] ; n = p q ; e = 2727 ;
```

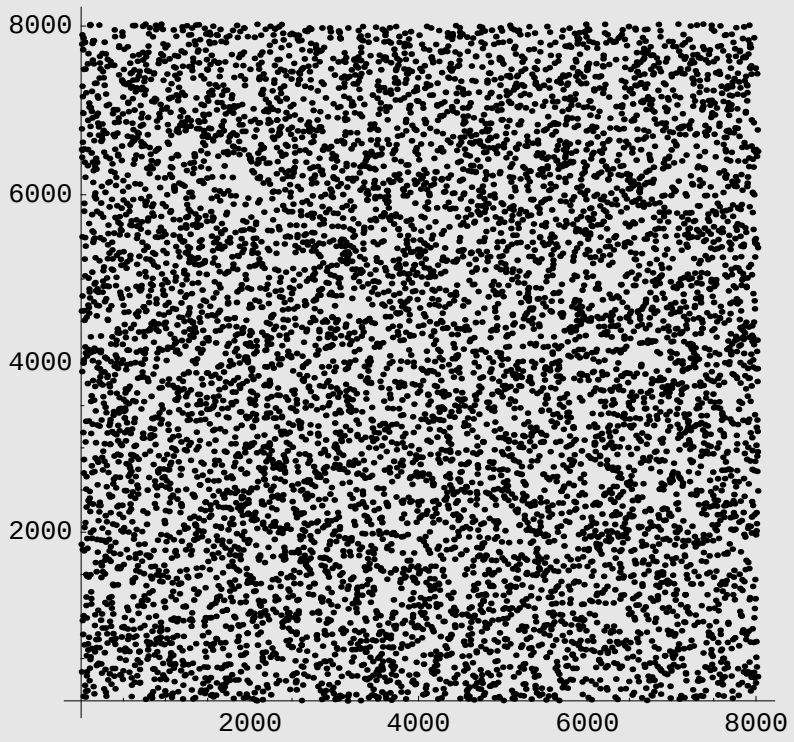
```
PowerMod[e, 84, (p – 1) (q – 1)]
```

```
1
```

A random function on the set $\{0, \ldots, n-1\}$ looks as follows

For easier comparision we plot the image of a random function (uniformly distributed).

```
ListPlot[Table[Random[Integer, {0, n – 1}], {i, 0, n – 1}], AspectRatio → Automatic];
```

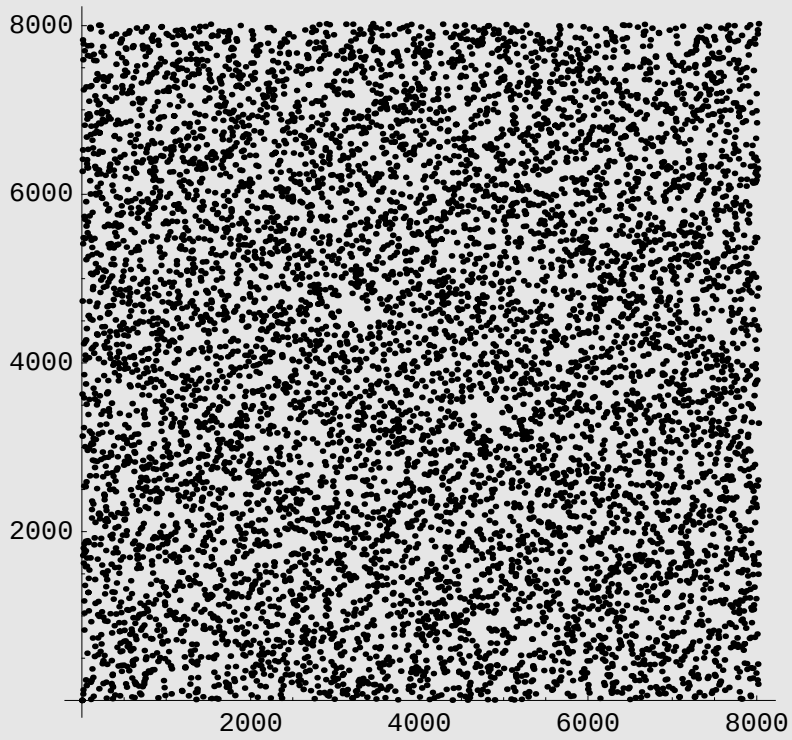


The RSA encryption function for the key $(n, 2727)$ looks similar.

Here is the plot of the encryption function with a certain key. At this point the students can see that both plots look similar.

Again, the students have to evaluate the inputs on their own in order to get some experiences with it. We want to animate the students to do observations on their own.

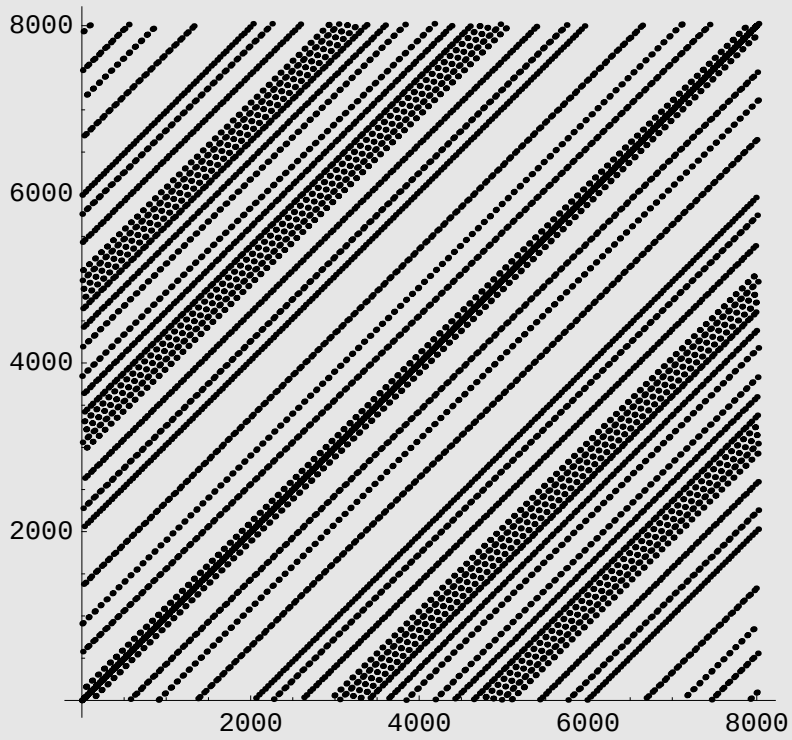
```
ListPlot[Table[{x, PowerMod[x, 2727, n]}, {x, 0, n – 1}], AspectRatio → Automatic];
```



Explore this property

Using another key we get some completely different results. Here it is obvious that the image does not look like a random function as the example above.

```
ListPlot[Table[{x, PowerMod[x, 5489, n]}, {x, 0, n – 1}], AspectRatio → Automatic];
```



What is the reason that for $e = 4971$, the graph does not look random? Find the "bad" exponents. How can they be characterized. How many are there? Is this a security issue?

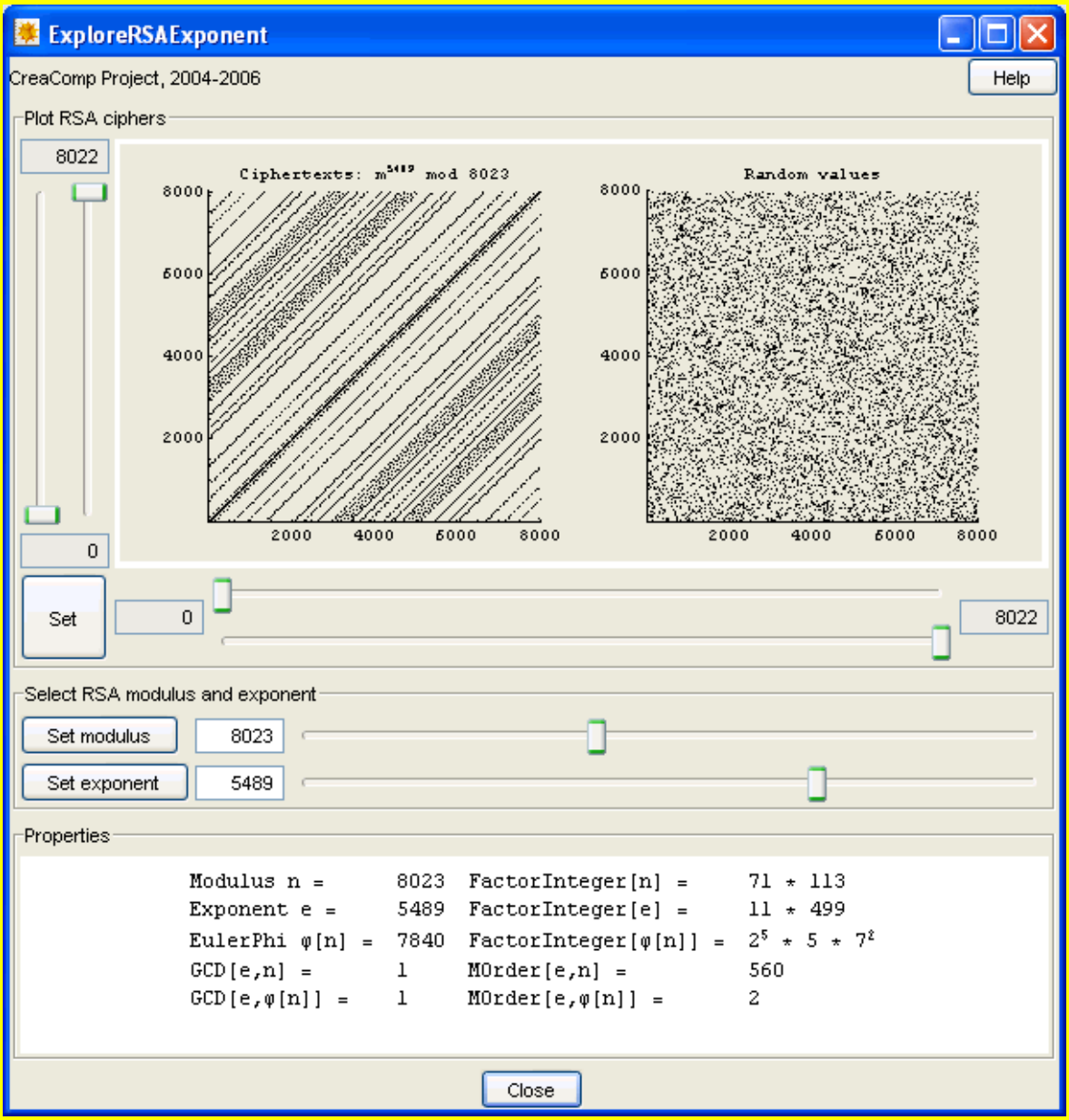
Explore RSA Exponents.

You can use the command ExploreRSAExponent[n,e] to start with your own modulus and exponent.

```
ExploreRSAExponent[n, e];
```

Now it is of great interest to explore this property in more examples. Therefore we provide an interactive tool to do this. After pressing the button or evaluating the input cell `Explore RSAExponent[n,e]` *a dialog window will be opened.*

Since a printed version can not be interactive we provide only a screen shot with some explanations:



In this window the students can compare the output of the RSA encryption function (plot on the left) with the image of an random function (plot on the right).

The plot area provides some zoom functionality to look closer at some interesting parts of the image. Using the horizontal sliders the range of the domain can be restricted, using the vertical sliders the range in the image is changed. The "Set" button updates both plots to the new ranges.

The modulus n can be changed by entering a number into the text field and hitting the RET-key or by pressing the button "Set modulus". Alternatively changing the modulus can be done by using the corresponding slider.

The button "Set exponent" does the same for the exponent e. The slider allows to select values from $\{1, \ldots \varphi[n]\}$.

In the special case when n is the product of two primes ($n = p * q$) there is $\varphi[n] = (p-1)(q-1)$. This case is used for the RSA encrytion scheme. The interactive window is powerful enough so that other cases can be observed, too.

In the last part there are listed some properties of n and e:

• Modulus n and exponent e with prime factorization.

• The EulerPhi function $\varphi[n]$ with prime factorization. $\varphi[n]$ is the quantity of all numbers smaller than n which are relatively prime with n.

• Further there is the greatest common divisor (in short: gcd) of e and n as well as the multiplicative order of e in $(\mathbb{Z}_n^*, *)$.

• $MultiplicativeOrder[e,n]$ gives the smallest integer a such that $e^a \equiv 1 \bmod n$.

• The gcd of e and $\varphi[n]$ has to be 1 in order to provide the correctness of RSA encryption.

The task of the students is to find out which properties of n and e are important for the randomness of the RSA encryption function.

Hints

1. Compare results for $e = 2727,\ 4183,\ 6449,\ 5489$.
2. Why is it a bad idea to choose e with small multiplicative order modulo $(p-1)(q-1) = \varphi(p*q)$ from a security point of view?

Finding the corresponding decryption key

A necessary property of an encryption scheme is the correctness. In this section we discuss this property in detail. First we formulate this property as a theorem. Then we formalize it and try to find the proof using the Theorema system.

If we want use the RSA encryption scheme, we would like to be sure, that every encrypted message can be decrypted by the holder of the private key. This property is called "correctness".

RSA encryption is very easy, given a key (n, e) and a message m, which is coded as a number between 0 and $n-1$ simply compute $c = m^e \bmod n$. To decrypt such a message, this process of taking the e^{th} power modulo n has to be reversed. So decrypting is find the e^{th} root modulo n of a given ciphertext c.

Theorem:
Let p and q be two primes, let e be a natural number such that $gcd(e, (p-1)(q-1)) = 1$ and let d be a natural number such that $ed = 1 \bmod (p-1)(q-1)$. Then for every integer c the unique solution x of the equation $c = x^e \bmod (p\,q)$ is $x = c^d \bmod (p\,q)$.

This means that it is easy to find a decryption key, when the factors of n are known.

Proof (Theorema):
We formalize the property in the language of Theorema. We use the variable m for the plaintext message, c for the ciphertext, r for the received message. n is the RSA modulus, e is the exponent, and d the inverse of e modulo $\varphi[n]$. The first assumption describes the encryption, the second the decryption, and the third one describes the properity of the inverse. RSA encryption is correct, if the received message r is equal to the plaintext message m. Equality in $\mathbb{Z}_n$ means r is congruent m modulo n.

Before we can use variables and symbols in Theorema, we have to be sure, that they have no value, e.g. from computations before:

```
Clear[m, c, r, e, d, n, a, b, φ];
```

```
Theorem["correctness of RSA", any[m, c, r, e, d, n],
  with[ ⋀ { c ≡n m^e
          { r ≡n c^d          ],
          { e ∗ d ≡φ[n] 1
  r ≡n m
]
```

Prove theorem! **Abort** **Show** **Off–line**

In the interactive version of this unit the students can use the button "Prove theorem!" to start the Theorema prover searching for a proof of the theorem. The students can use this button bar to control the Theorema system. The button "Abort" aborts a proof generation e.g. if it needs too much time. Using the button "Show" a generated proof, or at least an incompleted proof attempt, is shown in a separate window. If a proof was found or the prover was not able to do further steps the proof or the proof attempt is shown automatically in a separate window.

The button "Off–line" shows a pregenerated version of the proof. This ensures that the students can look at the proof even if the theorem or the knowledge base is incorrect or the online proof generation would take too much time on a certain maschine.

In the printed version we provide some parts of the generated proofs, in the interactive version the following will be generated automatically by the Theorema system:

Prove:

(Theorem (correctness of RSA)) $\forall_{c,d,e,m,n,r} (c \equiv_n m^e \wedge r \equiv_n c^d \wedge e * d \equiv_{\varphi[n]} 1 \Rightarrow r \equiv_n m)$,

with no assumptions.

We assume

(1) $c_0 \equiv_{n_0} m_0^{e_0} \wedge r_0 \equiv_{n_0} c_0^{d_0} \wedge e_0 * d_0 \equiv_{\varphi[n_0]} 1$,

and show

(2) $r_0 \equiv_{n_0} m_0$.

The proof of (2)fails. (The prover "QR" was unable to transform the proof situation.)

□

Now we discuss this failed proof attempt in order to support the automated prover to find the proof of the theorem.

The first attempt to prove the theorem failed. Of course, we did not use any additional knowledge. Theorema uses only knowledge we explicitly add to the knowledge base. If one tries to prove this theorem

by hand, the first idea would be to compound the first and the second equation in the assumption. Let's formalize this, to give Theorema the chance to do this, too:

Lemma["substitute", any[a, b, c, d, e, n],
$(a \equiv_n b^d \bigwedge b \equiv_n c^e) \quad \Rightarrow \quad a \equiv_n (c^e)^d$]

Prove theorem! **Abort** **Show** **Off–line**

Now the automated prover can do much more and comes closer to the proof goal. The following is the generated proof attempt, again usually this will be displayed automatically in a separate window:

Prove:

(Theorem (correctness of RSA)) $\forall_{c,d,e,m,n,r} (c \equiv_n m^e \bigwedge r \equiv_n c^d \bigwedge e * d \equiv_{\varphi[n]} 1 \Rightarrow r \equiv_n m)$,

under the assumption:

(Lemma (substitute)) $\forall_{a,b,c,d,e,n} (a \equiv_n b^d \bigwedge b \equiv_n c^e \Rightarrow a \equiv_n (c^e)^d)$.

We assume

(1) $c_0 \equiv_{n_0} m_0^{e_0} \bigwedge r_0 \equiv_{n_0} c_0^{d_0} \bigwedge e_0 * d_0 \equiv_{\varphi[n_0]} 1$,

and show

(2) $r_0 \equiv_{n_0} m_0$.

Formula (1.1), by (Lemma (substitute)), implies:

(3) $\forall_{c,e} (m_0 \equiv_{n_0} c^e \Rightarrow c_0 \equiv_{n_0} (c^e)^{e_0})$.

Formula (1.2), by (Lemma (substitute)), implies:

(4) $\forall_{c,e} (c_0 \equiv_{n_0} c^e \Rightarrow r_0 \equiv_{n_0} (c^e)^{d_0})$.

Formula (1.1), by (4), implies:

(5) $r_0 \equiv_{n_0} (m_0^{e_0})^{d_0}$.

Formula (5), by (Lemma (substitute)), implies:

(6) $\forall_{c,e} (m_0^{e_0} \equiv_{n_0} c^e \Rightarrow r_0 \equiv_{n_0} (c^e)^{d_0})$.

Formula (2), using (6), is implied by:

(7) $\exists_{\substack{c,e \\ (c^e)^{d_0} = m_0}} (r_0 \equiv_{n_0} m_0)$.

The proof of (7)fails. (The prover "QR" was unable to transform the proof situation.)

□

Again we analyse the failed proof attempt and we find some more knowledge we have to use in the proof.

Now the prover finds several steps, but the proof also fails. Looking at the last formula, the current proof goal, or looking at formula (5) we detect, that the proof fails, because the prover has no knowledge to simplify the power of a power. So let's add this lemma:

Lemma["power∗", any[a, b, c],
$(a^b)^c = a^{b*c}$]

Prove theorem! **Abort** **Show** **Off–line**

Here we insert only those lines which are different from the last proof attempt:

...

Formula (1.1), by (4), implies:

$r_0 \equiv_{n_0} (m_0{}^{e_0})^{d_0}$,

which, by (Lemma (power*)), implies:

(5) $r_0 \equiv_{n_0} m_0{}^{e_0 * d_0}$.

Formula (5), by (Lemma (substitute)), implies:

(6) $\underset{c,e}{\forall} (m_0 \equiv_{n_0} c^e \Rightarrow r_0 \equiv_{n_0} (c^e)^{e_0 * d_0})$.

...

The last lemma was only a small step forward. But now the proof is nearly done.

The proof fails again, but formula (5) is near to our proof goal. The prover never used the third assumption, formula (1.2). Let us introduce a lemma, which allows the prover to use this assumption:

Lemma["powerφ", any[a, b, c, n],
$(a \equiv_n b^c \bigwedge c \equiv_{\varphi[n]} 1) \Rightarrow (a \equiv_n b)$]

Prove theorem! **Abort** **Show** **Off–line**

Here the complete proof generated automatically by the Theorema system is shown:

Prove:

(Theorem (correctness of RSA)) $\forall_{c,d,e,m,n,r} (c \equiv_n m^e \bigwedge r \equiv_n c^d \bigwedge e * d \equiv_{\varphi[n]} 1 \Rightarrow r \equiv_n m)$,

under the assumptions:

(Lemma (substitute)) $\forall_{a,b,c,d,e,n} (a \equiv_n b^d \bigwedge b \equiv_n c^e \Rightarrow a \equiv_n (c^e)^d)$,

(Lemma (power*)) $\forall_{a,b,c} ((a^b)^c = a^{b*c})$,

(Lemma (powerφ)) $\forall_{a,b,c,n} (a \equiv_n b^c \bigwedge c \equiv_{\varphi[n]} 1 \Rightarrow a \equiv_n b)$.

We assume

(1) $c_0 \equiv_{n_0} {m_0}^{e_0} \bigwedge r_0 \equiv_{n_0} {c_0}^{d_0} \bigwedge e_0 * d_0 \equiv_{\varphi[n_0]} 1$,

and show

(2) $r_0 \equiv_{n_0} m_0$.

Formula (1.1), by (Lemma (substitute)), implies:

(3) $\forall_{c,e} (m_0 \equiv_{n_0} c^e \Rightarrow c_0 \equiv_{n_0} (c^e)^{e_0})$.

Formula (1.1), by (Lemma (powerφ)), implies:

(4) $e_0 \equiv_{\varphi[n_0]} 1 \Rightarrow c_0 \equiv_{n_0} m_0$.

Formula (1.2), by (Lemma (substitute)), implies:

(5) $\forall_{c,e} (c_0 \equiv_{n_0} c^e \Rightarrow r_0 \equiv_{n_0} (c^e)^{d_0})$.

Formula (1.1), by (5), implies:

$r_0 \equiv_{n_0} ({m_0}^{e_0})^{d_0}$,

which, by (Lemma (power*)), implies:

(6) $r_0 \equiv_{n_0} {m_0}^{e_0 * d_0}$.

Formula (6), by (Lemma (substitute)), implies:

(7) $\forall_{c,e} (m_0 \equiv_{n_0} c^e \Rightarrow r_0 \equiv_{n_0} (c^e)^{e_0 * d_0})$.

Formula (6), by (Lemma (powerφ)), implies:

(8) $e_0 * d_0 \equiv_{\varphi[n_0]} 1 \Rightarrow r_0 \equiv_{n_0} m_0$.

From (1.3) and (8) we obtain by modus ponens

(9) $r_0 \equiv_{n_0} m_0$.

Formula (2) is true because it is identical to (9).

□

Now we get the complete proof of correctness. The validity of first two lemmas is obvious. The third lemma is a result of the theory of residue classes, which we are going to prove "by hand" for the case where n is an odd prime and for the case where n is the product of two odd primes.

The following lemmata prove one of the lemmata used in the proof above. The proofs are done in special cases and by hand only because these are results from Number Theory and not the main interest of this unit. Still, they give us the chance to demonstrate the usefulness of the Chinese Remainder Theorem not only for fast integer arithmetic but also for proofs in number theory.

***Lemma**:*
Let b be an integer, let n be an odd prime number, and let c be an integer such that $c \equiv 1 \bmod (n-1)$. Then $b^c \equiv b \bmod n$.

Proof:
Let b be an arbitrary integer and let $c \equiv 1 \bmod (n-1)$. If $b \equiv 0 \bmod n$, then $b^c \equiv b \bmod n$. (Note, that $c \neq 0$.)
Thus, for the rest of the proof we may assume that $\gcd(b, n) = 1$. As a first step we are going to prove that the mapping $\phi : \mathbb{Z}_n \to \mathbb{Z}_n$, $\phi(x) := b\,x \mod n$, is a bijection. In order to prove that ϕ is injective, suppose we have $x, y \in \mathbb{Z}_n$ such that $b\,x \equiv \phi(x) \equiv \phi(y) \equiv b\,y \bmod n$. Then $b(x-y) \equiv bx - by \equiv 0 \bmod n$. Since $\gcd(b, n) = 1$, b has an inverse $b^{-1} \bmod n$. Multiplying the equation with b^{-1} yields $x - y \equiv 0 \bmod n$. Thus ϕ is injective. Since $\mathbb{Z}_n$ is a finite set, ϕ is also bijective.
Now

$$\prod_{x=1}^{n-1} x \equiv \prod_{x=1}^{n-1} \phi(x) \equiv \prod_{x=1}^{n-1} b\,x \equiv b^{n-1} \prod_{x=1}^{n-1} x \bmod n.$$

Dividing by $\prod_{x=1}^{n-1} x$ we obtain $1 = b^{n-1} \bmod n$.
Since $c \equiv 1 \bmod n - 1$, there exists an integer v, such that $c = 1 + v(n-1)$. Then
$b^c \equiv b^{1+v(n-1)} \equiv b.(b^{n-1})^v \equiv b.1^v \equiv b \mod n$.
□

***Lemma**:*
Let b be an integer, let p and q be two odd prime numbers, and let c be an integer such that $c \equiv 1 \bmod (p-1)(q-1)$. Then $b^c \equiv b \mod n$.

Proof:
Since $c \equiv 1 \bmod (p-1)(q-1)$, there exists an integer v, such that $c = 1 + v(p-1)(q-1)$.
By the previous lemma we have

$b^c \equiv b^{1+v(p-1)(q-1)} \equiv b^{1-v(q-1)+pv(q-1)} \equiv b^{1-v(q-1)}(b^p)^{v(q-1)} \equiv b^{1-v(q-1)}\,b^{v(q-1)} \equiv b \bmod p$, if $b \not\equiv 0 \bmod p$ and
$0^c \equiv 0 \bmod p$

Analogously,

$b^c \equiv b^{1+v(p-1)(q-1)} \equiv b^{1-v(p-1)+qv(p-1)} \equiv b^{1-v(p-1)}(b^q)^{v(p-1)} \equiv b^{1-v(p-1)}\,b^{v(p-1)} \equiv b \bmod q$, if $b \not\equiv 0 \bmod q$ and
$0^c \equiv 0 \bmod q$.

Thus, by the **Chinese Remainder Theorem**, $b^c \equiv b \bmod p\,q$.

□

Explore

$(77, 6)$ is not a valid key, because $\gcd(60, 6) = 6 \neq 1$. Why can $(77, 6)$ not be used as a key? What would be the problem when this key is used for encryption?

Example

This example is an introduction to the next section, practical computations of RSA encryptions. Here we discuss the raising of integers to high powers modulo a number, which is the product of two primes, and the difficultness to find the inverse operation to this. Here the "fast computation approach" to the Chinese Remainder Theorem is taken.

The students can evaluate input cells at runtime. Therefore they have the possibility to experiment with different examples.

The decryption key corresponding to the encryption key $(77, 53)$ is

```
PowerMod[53, -1, (7 - 1) (11 - 1)]
```

```
17
```

Decrypting the ciphertext $c = 50$ using the key $d = 17$ yields

```
m = PowerMod[50, 17, 77]
```

```
8
```

At this point an idea from the module **Fast Computations** may be very helpful. Explore this in the following exercise.

Explore

In the example above, compute m modulo 7 and modulo 11 and use the Chinese Remainder Theorem to reconstruct m modulo 77.

This makes the size of the numbers that have to be multiplied smaller. Does it allow you to reduce the size of the exponent, too?

Why can this method not be used for encryption?

With respect to security not very much more than assumptions can be made. To give the student an impression, one assumption and two theorems are stated here.

A second important property besides "correctness" is "security", it describes how difficult the decryption of a message is, if one does not know the private key.

RSA assumption:
There is no algorithm with running time polynomial in $\log(\min(p, q))$ *which, given* $p * q$, e *and* c, *computes the solution of the equation* $c = x^e \bmod (p\,q)$.

Theorem:
The RSA assumption implies hardness of factoring.

Proof:
Suppose, there exists a polynomial time algorithm for factoring. Then given a public key (n,e) one simply factors n to obtain its prime factors p and q and computes $d = e^{-1} \bmod (p-1)\,(q-1)$. Now, $x = c^d \bmod n$.
□

It is not known, whether hardness of factoring implies the RSA assumption, but ...

Theorem *(A. May, 2005):*
Given an RSA key pair $((n, e), (n, d))$ *it is feasible to compute the prime factors of* n *in time polynomial in* $\log(n)$.

Practical RSA

The last section has suggested a link between the RSA system and the problem of factoring integers. A little exercise shall give the student an impression of the hardness of factoring large numbers. As the remark "Use any tool, book or internet help." indicates the student shall learn about famous composite numbers which are not composed into factors, yet.

If the modulus n used for RSA encryption is small, it is easy to factor. How big does n have to be for factorisation to be infeasible?

Explore

Try to factor the Fermat numbers

$$2^{2^n} + 1$$

for $n = 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12$. Use any tool, book or internet help.

After this exercise, what might be a reasonable size for an RSA modulus?

The last section describes how one can do RSA encryption in a practical way.

The discussed example uses only Mathematica inputs. Hence it is possible that the students can use this to build up some programs to do RSA encryption within Mathematica.

Example

The results of the Mathematica commands are not in the notebook until the students will evaluate this commands. For this printed version we have added these results.

In 1978 Rivest, Shamir and Adleman computed the following RSA key

```
n =
    114381625757888867669235779976146612010218296721242362562561842935706935245733889 ⋱
      7830597123563958705058989075147599290026879543541;
e = 9007;
```

This key is known as RSA-129, the modulus n has 129 digits (426 bits).
The plaintext

```
plaintext = "THE MAGIC WORDS ARE SQUEAMISH OSSIFRAGE";
```

was coded using the scheme "A=01, B=02,...,Z=26, SPC=00"

```
m = Map[IntegerDigits[#, 10, 2] &, (ToCharacterCode[plaintext] − 64) /. {−32 → 0}] // Flatten //
    FromDigits
```

```
200805001301070903002315180419000118050019172105011309190800151919090618010705
```

The resulting number was then RSA-encrypted

```
c = PowerMod[m, e, n]
```

```
968696137546220614771409222543558829057599911245743198746951209308162982251457083 ⋱
  569314766228839896280133919905518299451578151 54
```

A prize of $100 was set out for the first person to decrypt this ciphertext. In 1994, Derek Atkins, Michael Graff, Arjen Lenstra, and Paul Leyland announced that they had succeeded in computing the factors of n [AG+94]. Knowing the factors of n, it is possible to compute the decryption key

```
p = 3490529510847650949147849619903898133417764638493387843990820577;
q = 3276913299326670954996198819083446141317764296799294253979828853 3;
p q == n
```

```
True
```

```
d = PowerMod[e, −1, (p − 1) (q − 1)]
```

```
10669861436857802444286877132892015478070990663393786280122622449663106312591177 4 ⋱
  4708733401685974623065539685445132771090536060 95
```

and decrypt

```
mm = PowerMod[c, d, n]
```

```
200805001301070903002315180419000118050019172105011309190800151919090618010705
```

Decoding using the above scheme yields the plaintext

```
(Map[FromDigits, Partition[IntegerDigits[mm, 10], 2]] /. {0 → −32}) + 64 // FromCharacterCode
```

```
THE MAGIC WORDS ARE SQUEAMISH OSSIFRAGE
```

The module **Factoring Integers** describes the method used for factoring the RSA-129 modulus.

The module ends with a hyperlink to the related module Factoring Integers. Since the security of the RSA encryption scheme is related to the difficultness of factoring large integers this is a interesting continuation. In this module a method for factoring of medium sized integers will be discussed.

Remarks

The following are references to the internet and an easy to read scientific publication. The student should not stop here at the end of the module. Reading scientific articles is another important step towards developing an idea of mathematical reasoning.

[AG+94] Atkins, D., Graff, M., Lenstra, A.K., Leyland, P., announcement, 1994, online at http://www.mit.edu:8001/people/warlord/RSA129-announce.txt

[RSA78] Rivest, R. L., Shamir, A., Adleman, L. A., "A Method for Obtaining Digital Signatures and Public-key Cryptosystems", Communications of the ACM Vol.21, Nr.2 (1978), 120-126.

Using a Computer-Algebra System and a Theorem Prover to Stimulate Creativity in Learning Mathematics*

Susanne Saminger† Robert Vajda‡

Wolfgang Windsteiger‡

Johannes Kepler University, A-4040 Linz, Austria

† FLLL – Institut für Wissensbasierte Mathematische Systeme
‡ RISC – Institut für Symbolisches Rechnen

Susanne.Saminger@jku.at
Robert.Vajda@risc.uni-linz.ac.at
Wolfgang.Windsteiger@risc.uni-linz.ac.at

Abstract

We present an environment for learning and teaching mathematics that aims at inspiring the creative potential of students by enabling the learners to perform various kinds of interactive experiments during their learning process. Computer interactions are both of visual and purely formal mathematical nature, where the computer-algebra system *Mathematica* powers the visualization of mathematical concepts and the tools provided by the theorem proving system *Theorema* are used for the formal counterparts. We present a case study on the concept of convergence of real-valued sequences, in which we demonstrate the entire bandwidth of computer-support that we envision for modern learning environments for mathematics ranging from "getting first ideas and intuitions" over "checking the validity of first ideas on a wide variety of examples" until "rigorously proving or disproving one's own conjectures".

Keywords: Computer Algebra, Automated Theorem Proving, E-learning, MeetMath, Theorema.

1 Introduction

Both in classroom and in scenarios of distance learning of mathematics the use of computer-algebra systems has become more and more popular. Re-

*Corresponding author: Wolfgang Windsteiger, Wolfgang.Windsteiger@risc.uni-linz.ac.at

cently they support computation and visualization: the availability of powerful algorithms allows to perform calculations even in situations when the method would require a tremendous computational effort when executed "by hand" or when the algorithms are not known to the students at a certain level of education. Additionally, different visual representations of the computational objects could contribute to qualitative data analysis and exploration of non-obvious mathematical relationships. Nevertheless it is clear that the acquisition of knowledge about how to pose and prove theorems, moreover the question "What makes a proof a proof?" have to be addressed during mathematics education, see e.g. [Hanna, 2000, Housman and Porter, 2003, Recio and Godino, 2001]. On the other hand, most of the available electronic learning material for mathematics only rarely focus on the support for acquiring this crucial skill for a mathematician, for exceptions see e.g. [Andrews *et al.*, 2004, Sommer and Nuckols, 2004]. Besides, the availability of an engine that can perform thousands of computations within just a fraction of a second sometimes leads students to the attitude that *checking the truth of an arbitrary property* on many concrete examples makes them *believe to having proved* that the property necessarily always holds, see [Hanna, 2000, Housman and Porter, 2003].

In this paper we present the CreaComp project, whose main goal is to develop electronic course material for self-study and also for use in classroom. Besides being electronically available and distributable the material also offers computer-support for the mathematical topics it covers. The computer-aid provided in CreaComp units follows the didactical guidelines set up in the MeetMath project, see Section 2, and it promotes an approach of self-paced learning that has been centered around "gaining insight into mathematical concepts through computational and graphical interaction". MeetMath uses *Mathematica*, see [Wolfram, 2003], as its computation engine. In addition to computational and graphical interaction the CreaComp approach now adds *formal reasoning* as a third important component by integrating the *Theorema* system, see [Buchberger *et al.*, 2005, Buchberger *et al.*, 2000, Buchberger *et al.*, 1997], a mathematical assistant system also based upon *Mathematica*. *Theorema* is designed to become a uniform environment in which a mathematician gets support during all periods of her mathematical occupation. For the CreaComp project, however, *Theorema* mainly provides the mathematical language and the possibility of fully automated or interactive generation of mathematical proofs.

In contrast to most of the approaches for using computers to assist learning and teaching of mathematics, which see the application of the computer solely for visualization and computation, we will show that the computer can be of help also when teaching formal rigor. There is often the distinction between *experimental mathematics* (standing for visualization and computation) and *formal mathematics* (standing for proving) and computer-supported teaching/learning is primarily associated with experimental mathematics whereas all proving-related mathematics is predominantly considered to be done "by hand". The pedagogic discussion is often "experimental mathematics vs. for-

mal mathematics" and the question is to what extent can formal mathematics be supplemented/accompanied/enriched/replaced by experimental mathematics and vice versa, see [Borwein, 2005]. The CreaComp approach should be seen much more as "experimental formal mathematics", i.e. we demonstrate that

- both teachers and students can make untold fruitful experiences when computer-support enters also formal mathematics and
- also formal mathematics reveals many experimental aspects that are certainly worth teaching and learning, e.g. how to build up mathematical theories, how to formulate useful mathematical knowledge, or how to invent new mathematical knowledge.

In other words, we argue that formal mathematics can well be taught with an experimental flavor.

In the sequel we will introduce the constituent components MeetMath and *Theorema* in more detail in Sections 2 and 3 and describe the way they combine to CreaComp in Section 4. The main part will be a case study presenting parts of a CreaComp unit on the convergence of real-valued sequences in Section 5.

2 MeetMATH

MeetMath denotes a family of interactive mathematics courseware based on *Mathematica* equipped with a Java™-based navigation. The basic course MeetMath@Business&Economy and the didactic concepts for MeetMath courseware have been initially developed in the framework of the project IMMENSE (Interactive Multimedia Mathematics Education in Networked universities for Social and Economic sciences) initiated by the Johannes Kepler University Linz already in 1999 (for more detailed information about the project and partners see `http://www.flll.jku.at/meetmath`).

The development of any courses including electronic materials or support demands to focus not only on, possibly new, technology but on the corresponding didactic framework. By the use of computers in learning scenarios the role of a teacher has changed. Moreover and more important, objectives and students themselves have achieved a more central role in the learning process, accompanied with an increased awareness of the students' responsibility.

One of the main guidelines throughout the development of MeetMath has been:

> "If our aim is to help students to become competent, autonomously acting experts in their field of practice, we must give them the opportunity to behave competent and autonomous as learners, too."

As a consequence, a simple transfer of static mathematical knowledge as, for instance, presented in books to electronic documents would not be sufficient to reach the project goals. The same applies to just building a collection of single

interactive documents not being related to each other. Structuring a learning process is an important and crucial aspect for its successfulness, but fixing a unique linear structure would restrict the possibilities and responsibilities of the students in how to organize their learning process. Therefore, didactical demands for the development of courses with electronic components are different from classical guided training situations. MeetMath provided an answer by the concept of "didactical rooms". Students can then decide on the sequence and the duration by which they like to visit different parts of the materials, each of which dedicated to some of these didactical room. Basically, four different types of rooms have been distinguished:

- "motivation/linking": Motivation/linking addresses the students pre-knowledge about and their attitude to deal with certain contents. Motivation rooms should be accessible in an easy way. Moreover, the attendance of these rooms must be optional, since obligatory motivation where no motivation might be needed is discouraging and can be sensed as annoying. Therefore movies—illustrating real-life problems, animations—presenting complex situations, sounds—remembering of certain situation in everyday life, pictures—showing typically situations, examples and experiments should be optional.

- "acquisition/confrontation": The central concepts must be presented in a complete and carefully paced argumentation. The student should get the possibility to add new knowledge to the existing knowledge base. Within confrontation rooms relevant information, thoughts and illustrations should be offered in a way that students are able to build up their own ideas and concepts and/or to modify existing, incorrect concepts.

- "strengthening": Strengthening rooms allow to verify and inspect newly developed concepts and ideas. Experiments and interactive elements enable students to stabilize the concepts. Experiments allow to falsify incorrectly developed concepts or parts of concepts by trial and error. Both success and failure are possible and allowed during strengthening. Failure is important and indicates that returning to a room of acquisition might be necessary. The level and the design of strengthening parts must be considered well, such that strengthening rooms do not degenerate into "rooms of frustration", since the new aspects about the contents to be investigated are too complex or seem to be unrelated.

- "assessment": Assessment is a necessary tool for supervising a learning process. Since students are responsible for their processes and their knowledge development, they need some tools for measuring the progress and/or for finding out the status quo. Assessment can be necessary on different stages of the learning process—at the beginning, in the middle or the end. As a consequence the accessability of assessment rooms throughout the process is a relevant aspect when designing electronic course materials.

MeetMath courses have been developed by taking into account these principles. Moreover, to overcome a linear structure of the documents, the units themselves are structured into the categories MathWorld and RealWorld and provide an approach to mathematical topics based either on the underlying mathematical concepts or on convincing real-life problems. By a Java™-based navigation, they are arranged in a matrix navigation allowing access to topics from different points. Since MeetMath is based on *Mathematica* movies, interactive experiments as well as animations and the computational power of *Mathematica* are integrated in a way to facilitate and support the acquisition of insights into mathematical concepts.

MeetMath@Business&Economy has been integrated into the mathematics education of students of business administration and economy at the Johannes Kepler University Linz since winter semester 2001/2002. In this context, it has been applied and adopted for different learning scenarios, like self-paced learning, large scale lectures, tutorials. However, parts of it have also been been employed in the education of mathematics students and students of computer science, respectively. Exactly, those experiences showed that MeetMath— or computer-algebra systems in general—allow to gain insight into mathematical concepts, to solve and inspect a lot of examples in a quick manner, to provide support if properties of objects are to be proven for a finite number of cases or can be proven by simple calculations or algebraic transformations. However, their capabilities seem somewhat limited if it comes to reasoning about general properties or to applying specialized proof techniques.

3 The Theorema System

Theorema is a system that intends to bring computer-support during all phases of mathematical activity. Typically, the every-day work of mathematicians consists of alternating phases of

- defining new mathematical objects,
- performing experiments on the new objects,
- conjecturing properties of the new objects,
- proving the correctness of the conjectured properties,
- developing algorithms,
- running and improving existing algorithms,
- visualizing mathematical data,
- and many more.

Some of these activities can already be computer-supported by existing mathematical software, but most of the current systems specialize in one of the above

aspects and give only weak support for the others. Hence, computer-oriented mathematics often needs a combination of several software components, which results in cumbersome conversions from data available in one system into data usable in the other systems. Much more important in the context of combined systems, though not always evident and often neglected, is the question of *correctness* since there is no common underlying logic upon which the different systems are built up. There is no guarantee that the meaning of a symbol is preserved when converting from one system to an other. Note that this source of risk comes *in addition* to the risk of errors in the individual component systems.

The *Theorema* system provides a uniform language and logic, in which many of the above activities can be carried out, see [Buchberger, 1996]. The overall design principle of the *Theorema* system is to communicate with the user in "mathematical textbook style". The syntax of the language in both input and output is close to "common mathematical notation" including special mathematical characters and two-dimensional syntax as typically used in mathematics. The *Theorema* language is a version of higher-order predicate logic with pre-defined basic mathematical objects such as numbers, sets, and tuples. For algorithmic language constructs such as numbers, finite sets and tuples, and quantifiers with finite ranges the language provides semantics in form of algorithms, which can be accessed in particular during computations. The main focus in the development of the *Theorema* system over the past years has, however, been put on the development of various general and special-purpose theorem provers. In its current state, *Theorema* contains fully automated provers for propositional and first order predicate logic, a prover for equational theories, a set theory prover, several induction provers, a prover for geometry based on the Gröbner basis method, and a prover for combinatorial identities based on the Paule-Schorn method.

The *Theorema* system is built on top of the well-known *Mathematica* system. In particular, we use the *Mathematica* notebook front-end as the user-interface for *Theorema* and the *Mathematica* programming language as the implementation platform for both the *Theorema* language and the *Theorema* provers. When generating mathematical proofs, *Theorema* displays the full proof in a separate notebook document with each proof step explained in natural language. The structure of the proof is reflected in the cell structure of the proof notebook, so that the standard *Mathematica* technology of opening/closing nested cells by mouse-click can be used to collapse entire proof branches. This allows the reader to get an overview over the structure of the proof and then to "zoom into" parts of the proof by subsequently opening just the relevant cells. As an alternative to fully automated proof generation the *Theorema* system also allows for interactive proving. In this situation, the user may influence the proof generation in various ways, e.g. by selecting the next proof step to be applied, by helping the prover to instantiate formulae during the proof, or by even introducing entirely new formulae during the proof. For details on the *Theorema* system, the language, and examples of provers implemented in the *Theorema* system we refer to the introductory papers [Buchberger *et al.*, 2005, Buchberger *et al.*, 2000, Buchberger *et al.*, 1997].

4 The Combination of MeetMath and Theorema

Traditional teaching of mathematics often leads to frustration among the students. The reasons for this are manifold:

- The relevance of mathematical methods for real-world problems cannot be recognized. Only toy examples can be presented in class because the real-world examples
  - would exceed the calculation capabilities of the students, or
  - would need some additional knowledge or algorithms that are not available on the respective level of education, or
  - would require complex mathematical modelling capabilities that the students are not aware of.
- Concise visualization of mathematical concepts is difficult to achieve using traditional media such as blackboard or transparencies. Moreover, it is hardly possible for the students to reproduce visualizations properly.
- The students' role in discovering mathematics is often a passive one, only reproducing what has previously been presented.

The use of electronic media in classroom has helped to overcome some of the above problems, in particular with respect to computational tasks and visualization matters by applying the "Black-Box White-Box principle", see [Buchberger, 1990]. It should be mentioned, however, that additional problems just due to the use of computers have been encountered, see e.g. [Drijvers, 2002], but are not in the focus of this work. In our view of mathematics the most dangerous scenario of computer-supported mathematics is that the extensive use of computation and visualization leads to a tradition of "proof by inspecting particular examples" (though many!) instead of "proof by mathematical proving". It is one of the main goals of this project to highlight the importance of rigorous formal mathematical arguments in all facets of mathematical work, including for instance also software development.

The combination of MeetMath and *Theorema*, which is investigated in the frame of the CreaComp-project, therefore aims at providing computer aid for visualization, computation, but most importantly also for proving. Whereas computation and visualization ought to fertilize the students' *intuition* about mathematical objects, the proving phase should establish and enhance their *understanding* of mathematical argumentation, in particular that "validity in some examples" does not necessarily always coincide with "validity in all cases". An additional benefit of a theorem proving system as the student's assistant is that the students must think carefully about tacit assumptions they use in their argumentation, because the automated prover forces them to state all usable knowledge explicitly, see also e.g. [Hanna, 2000].

CreaComp educational units are distributed in the form of *Mathematica* notebooks containing normal text intermixed with formal mathematical texts

(definitions, theorems, lemmata, etc.) written in the *Theorema* language. Computational experiments involving the concepts introduced in the formal parts can be carried out using *Theorema*'s `Compute` command. *Theorema* computations exploit the computational power of the underlying *Mathematica* system for computations involving numbers but, additionally, also truth values of formulae involving quantifiers can be computed as long as the quantifiers involve only finite ranges. Since visualization is not yet included in native form in the current version of *Theorema*, i.e. there is not yet an integrated interface from *Theorema* to the various plotting capabilities available in *Mathematica*, we provide certain visualization commands for certain mathematical objects on demand. In addition to static plots we try to involve the students in actively exploring mathematical properties by providing interactive graphics based on *Mathematica*'s GUIKit, a toolbox supporting the implementation of Java applications fed with *Mathematica* data. Finally, we encourage students to also prove the conjectures they may come up with after their experiments. In this stage they may use the *Theorema* provers both in interactive or in fully automated mode.

CreaComp-RealWorld units will be organized very much along the lines of MeetMath-RealWorld units, i.e. they will demonstrate the application of mathematical algorithms developed in some MathWorld unit to real-world examples. Algorithms will be used in the form as available in *Mathematica*, thus the RealWorld units shall teach the students how to solve given problems using mathematical algorithms available in a computer-algebra system. In this sense, the RealWorld units serve the didactic goals of "acquisition" and "strengthening" on the level of the students' general problem solving capabilities. The main purpose of the RealWorld units is, however, to provide "motivation" to study some mathematical ideas and to show where and how mathematics is used in practice. The CreaComp-MathWorld units, on the other hand, will show the development of mathematical theories and therefore they relate more to the didactic guidelines of "acquisition" and "strengthening" on the level of mathematical knowledge. These will be the places where the combination of MeetMath and *Theorema* will be most prominent. Thus, we will show the flavor of a typical CreaComp-MathWorld unit in the remainder of this paper in a case study on the evolution of a theory on real-valued sequences.

5 Case Study: Exploring Real-Valued Sequences

Following the didactic principles taken from MeetMath, the typical flow of a CreaComp-MathWorld unit is to

- motivate the students by some real-world example,
- present new mathematical concepts by defining new objects or properties,
- let the students experiment with the new entities on concrete data,

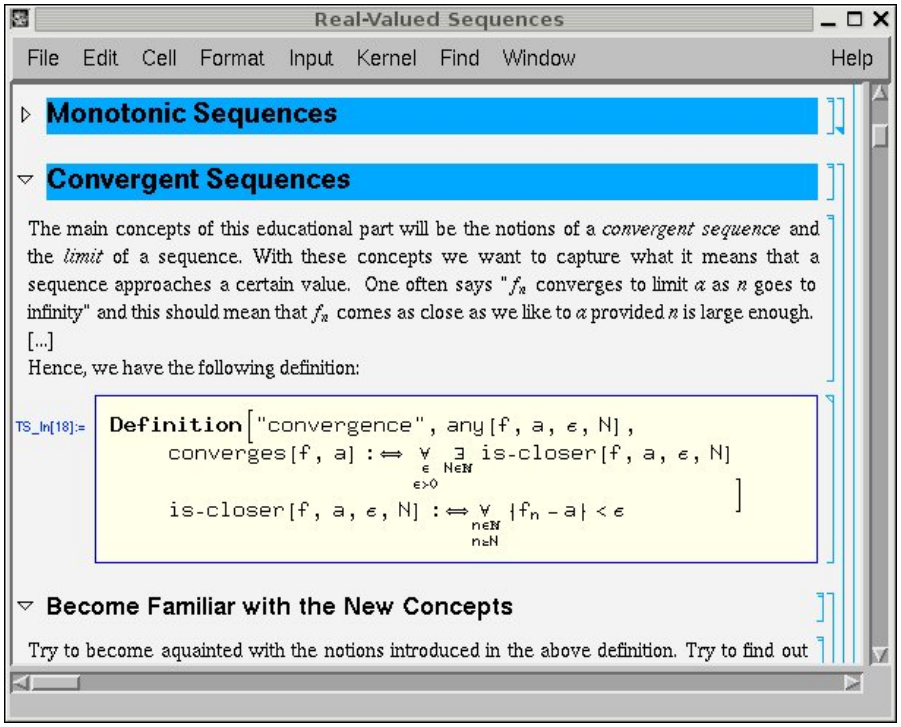


Figure 1: Educational unit showing structured text with *Theorema* formal text.

- guide the students in their experiments such that possibly they are able to conjecture new properties,
- guide the students in rigorous proofs of their conjectures.

We try to illustrate the flavor of computer-support given in a CreaComp-MathWorld unit in the example chapter on "convergence of real-valued sequences". Figure 1 shows a screen-shot of a part of the notebook on real-valued sequences. The chapter on "Convergent Sequences" is opened as indicated by the cell structure rendered on the right margin, the preceding chapter on "Monotonic Sequences" is closed. We see that a CreaComp unit contains structured text containing also in-line formulae, but formal parts (e.g. definitions, theorems, algorithms, etc.) are written in the *Theorema* language. *Theorema* blocks are written in *Mathematica* input cells and must be evaluated in order to be accessible. Note, however, that *Theorema* input is quite different from usual *Mathematica* input! *Theorema* understands most of the common mathematical notation, notably all sorts of quantifiers written in appealing two-dimensional form.

The students' context when entering the chapter on convergent sequences is as follows: A motivating real-world example has been presented by hyper-linking

this unit to an appropriate RealWorld unit at the beginning of the notebook, real-valued sequences have been introduced as functions from the naturals to the reals, and different representations for sequences have been introduced such as the graph of a sequence or the range of values. Some properties of sequences such as boundedness and monotonicity have already been presented, but in the spirit of self-paced learning, we do not require that every student has gone through these chapters already. Moreover, of course, students are familiar with the standard principles and the basic usage of *Theorema* and *Mathematica*.

At various points throughout the presentation of this case study, in particular always after the explanation of computer experiments available for the students, we presuppose certain "behavior" of the students. The experiment's effects, which we describe in this case study, should be understood as *our intended goals* for the experiment. We will refer to insights or ideas, which we expect to be in reach for at least some of the students. In other words, we strive to design the experiments and the guidance for the students before, during, and after the experiments in such a way that possibly every student can reach the intended goal even in complete self-study. It is clear that in a learner-centered environment with some latitude upon what portions of the material the student actually studies, we must also foresee the student's deviation of our anticipated flow of ideas. Putting intermediate questions in order to route ideas into a desired direction before proceeding with further experiments is considered as an appropriate didactical remedy for this problem. In this sense, in the sequel, phrases like "the student might see ..." or "the student may conjecture ..." refer to our didactic goals behind the respective computer interactions.

5.1 Motivation and Introduction of the New Concept

The definition of the new mathematical property "the sequence f converges to a" is written in the *Theorema* language in the form

$$\textbf{Definition}[\text{"convergence"}, \text{any}[f, a, \varepsilon, N],$$
$$\text{converges}[f, a] :\Longleftrightarrow \underset{\substack{\varepsilon \\ \varepsilon > 0}}{\forall} \underset{N \in \mathbb{N}}{\exists} \text{is-closer}[f, a, \varepsilon, N]$$
$$\text{is-closer}[f, a, \varepsilon, N] :\Longleftrightarrow \underset{\substack{n \in \mathbb{N} \\ n \geq N}}{\forall} |f_n - a| < \varepsilon \qquad] .$$

5.1.1 Visual Exploration of the New Concept

Before proceeding further we want the students at this moment to get some intuition *what* it actually means when $\text{converges}[f, a]$. Informally, in the introductory text of this chapter, it has already been suggested that $\text{converges}[f, a]$ should capture that f_n comes close to a for large n. For this purpose we provide a large collection of example sequences, which the students can substitute for f in the definition. We will now show some of the expected experiments for just two sequences from this collection, namely g and h defined by

$$g_n \quad := \quad \frac{2}{n^2 + 3n}$$

$$h_n \quad := \quad \mathrm{Cos}[n\pi] + \frac{1}{n} \; .$$

When referring to experiments performed on concrete example sequences in the rest of the paper, we will always refer to only those two examples, although we expect the students to perform the respective experiments on many more of the supplied examples.

In this moment, the students need the insight that a decision upon truth or falsity of a statement like converges$[f, a]$ needs at least concrete values for both f and a. For f they will try out the sequences g and h, but for a they need to *guess appropriate values*. Visualizing g and h is a reasonable strategy, therefore the functions `PrintSequence` for producing a nicely formatted two-dimensional table of values and `PlotGraphSequence` for two-dimensional graphical representation from the introductory sections on sequences may appear useful. In order to guide the experiments into a "reasonable" direction, the appropriate input lines are already supplied in the notebook.

In[1]:= `PrintSequence[g,{20,30}]`

Out[1]=

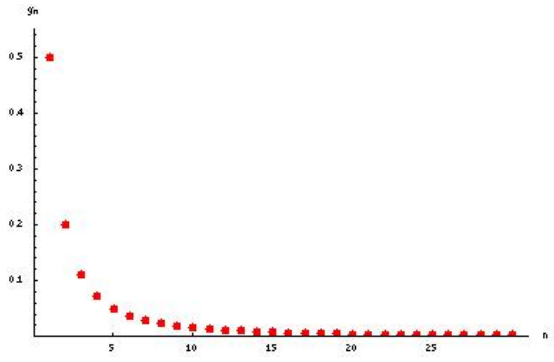
n	g_n
20	0.00434
21	0.00396
22	0.00363
23	0.00334
24	0.00308
25	0.00285
26	0.00265
27	0.00246
28	0.00230
29	0.00215
30	0.00202

In[2]:= `PrintSequence[h,{20,30}]`

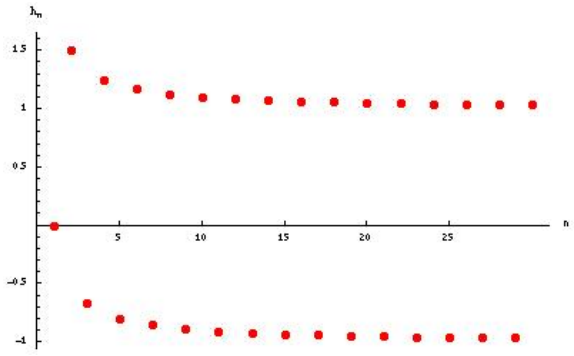
Out[2]=

n	h_n
20	1.05
21	−0.95238
22	1.04545
23	−0.95652
24	1.04166
25	−0.96
26	1.03846
27	−0.96296
28	1.03571
29	−0.96551
30	1.03333

In[3]:= `PlotGraphSequence[g,{1,30}]`



In[4]:= `PlotGraphSequence[h,{1,30}]`



From the visualization a student might get the impression that "g_n comes close to 0" and "h_n comes close to both 1 and -1". Hence, from the informal meaning of "converges" given before, a student might speculate now $\mathsf{converges}[g, 0]$ and both $\mathsf{converges}[h, 1]$ as well as $\mathsf{converges}[h, -1]$.

The hypotheses set up in their first experiments need now to be assessed. By definition, $\mathsf{converges}[f, a]$ stands for

$$\forall_{\substack{\varepsilon \\ \varepsilon>0}} \exists_{N\in\mathbb{N}} \mathsf{is\text{-}closer}[f, a, \varepsilon, N] \tag{1}$$

and we want the students to get a feeling what this actually means. We provide an interactive widget implemented using *Mathematica*'s GUIKit for exploring formula (1) for given f and a, which must be specified by the student when calling the widget. The core idea of this widget (and all other interactive graphics elements in CreaComp) is to *visualize the quantified formula for concrete values of the quantified variables* and provide means to *interactively change the values* substituted for the quantified variables. Thus, the widget as shown in Figure 2 contains input fields for ε and N and the graphics area displays a visualization of $\mathsf{is\text{-}closer}[g, 0, \varepsilon, N]$ and $\mathsf{is\text{-}closer}[h, 1, \varepsilon, N]$, respectively. Changing ε moves the horizontal lines representing the "ε-corridor around a", whereas changing N moves the vertical line symbolizing the threshold. The graphics is re-rendered with every change in one of the input fields, thus providing immediate feedback. As a general rule for interactive graphics, the students know that values for the quantified variables need to be chosen left-to-right, i.e. in a formula of the form $\forall_{\varepsilon} \exists_{N} \ldots$ one needs to choose ε first and only then choose N. From the picture, it is intended that the student is able to read off whether $\mathsf{is\text{-}closer}[f, a, \varepsilon, N]$ for the respective f and a and for the chosen values of ε and N. The definition of $\mathsf{is\text{-}closer}$ is displayed at the top of the widget, thus, the student needs to decide whether

$$\forall_{\substack{n\in\mathbb{N} \\ n\geq N}} |f_n - a| < \varepsilon, \tag{2}$$

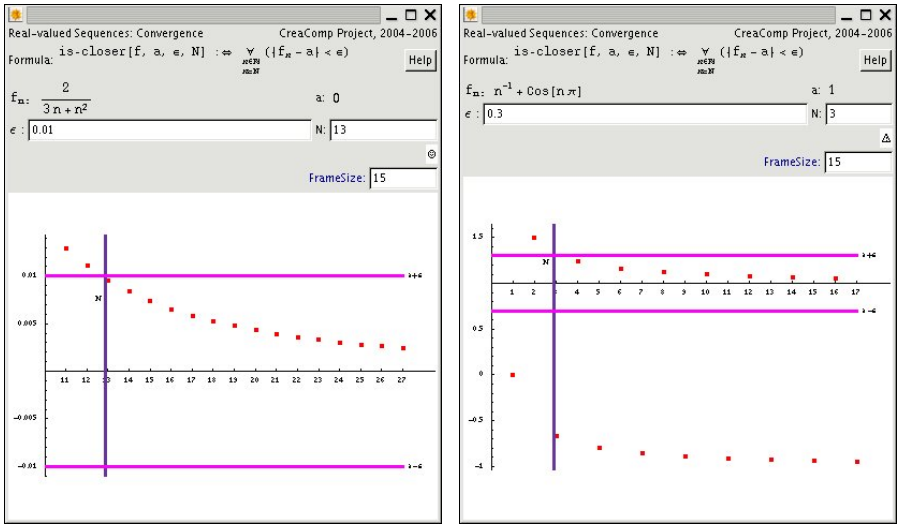


Figure 2: Interactive exploration of $\mathsf{is\text{-}closer}[g, 0, \varepsilon, N]$ and $\mathsf{is\text{-}closer}[h, 1, \varepsilon, N]$.

i.e. whether *all sequence elements* lying to the right of the threshold stay within the marked ε-corridor. Of course, we cannot display *all sequence elements* right of the threshold, therefore the widget contains one more input field for specifying how many elements to display. If the investigated property holds for the finite portion of the sequence in the display, then this is indicated with a "smiley" (on the right margin next to the input fields), otherwise a "warning sign" indicates the violation of the property. After having played with the input values, the student might conjecture for instance:

- for whatever choice for ε it is always possible to choose N such that $\mathsf{is\text{-}closer}[g, 0, \varepsilon, N]$, i.e. $\mathsf{converges}[g, 0]$.
- for small ε (e.g. $\varepsilon = 0.3$), whatever we choose for N, we always have some elements of the sequence within the ε-corridor and some of them outside, i.e. $\neg\mathsf{converges}[h, 1]$.

We then ask the students to also try alternative values for a, e.g. to try whether $\mathsf{converges}[g, 0.001]$ or $\mathsf{converges}[h, -1]$. Possible situations resulting from the former are shown in Figure 3, whereas the latter exhibits "basically the same behavior" as what we already presented in Figure 2. The insights gained from these exercises should then be:

- for certain values of ε (e.g. $\varepsilon = 0.001$) it is also possible to determine N such that $\mathsf{is\text{-}closer}[g, 0.001, \varepsilon, N]$, but as ε becomes sufficiently small (e.g. $\varepsilon = 0.0001$) this is not anymore possible, i.e. $\neg\mathsf{converges}[g, 0.001]$.
- analogously to above also $\neg\mathsf{converges}[h, -1]$.

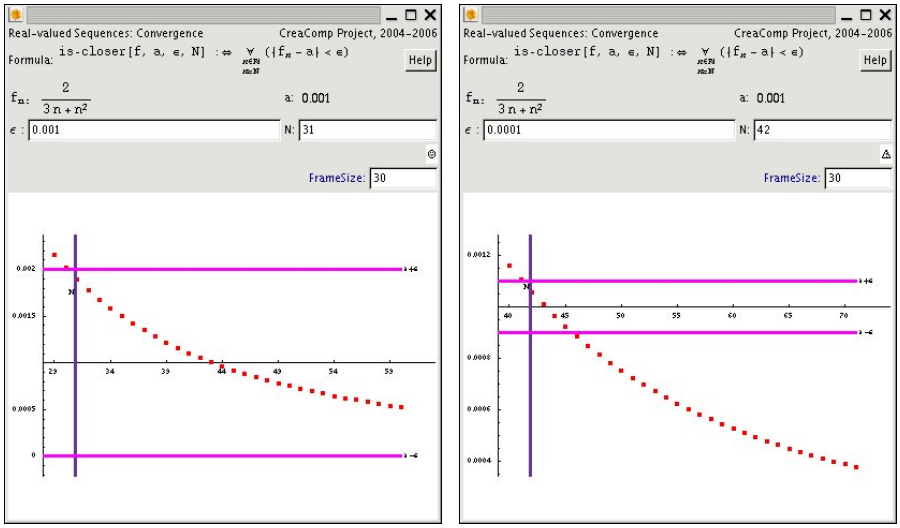


Figure 3: Interactive exploration of $\mathsf{is\text{-}closer}[g, 0.001, \varepsilon, N]$.

It must become clear to the students, however, that these experiments cannot prove the conjecture $\mathsf{converges}[g, 0]$, because only finitely many values for ε and only a finite segment of the sequence can be tested! For the conjectures $\mathsf{converges}[g, 0.001]$ and $\mathsf{converges}[h, 1]$, on the other hand, the experiment already gives the main ingredients for constructing a counter-example, e.g. for the former take $\varepsilon = 0.3$ and an arbitrary $N \in \mathbb{N}$, assume $\mathsf{is\text{-}closer}[h, 1, 0.3, N]$ and it will be not too complicated to derive a contradiction.

Although presumably $\neg\mathsf{converges}[h, 1]$ the above experiments might still reveal some essential features of the sequence h.

- We can choose ε as small as we want and we will always find some sequence elements, in fact infinitely many, in the ε-corridor around 1. A fruitful discussion on the subtle difference between "infinitely many" and "all starting from some index N" can evolve from that.
- If we consider only the "even elements of the sequence" then "this sequence $\bar{h}$" would probably fulfil $\mathsf{converges}[\bar{h}, 1]$.

These observations lead naturally to the concepts of *accumulation point of a sequence* and *subsequence*. In the CreaComp unit we will hyper-link to the respective MathWorld units covering these topics. Following these links will lead students to a new perspective of the topic, i.e. to a different didactical room (see the MeetMath concept of "didactical rooms" described in Section 2), which we will not pursue further in this paper. The main path of exploring the notion "converges" will be described in the subsequent sections.

5.1.2 Formal Exploration of the New Concept

In the next phase of the unit, we encourage the learner to experiment also on the formula level. Moreover, the formal exploration described in this section can be understood also as a substitute for the visual exploration described above for those persons who are not so much of visual nature. As a general strategy for becoming familiar with new mathematical material (definitions, algorithms, etc.), we propose to students throughout the entire material if possible (and reasonable) to "first investigate some finite cases by computation". In this example, this can be achieved by changing all infinite ranges in the quantifiers to finite ranges, namely ranges over finite sets or ranges over finitely many integers. We recommend to introduce additional parameters in order to adjust the finite ranges during the computational experiments. In a first attempt this would lead to a finitary version of convergence[1] such as for instance

$$\mathsf{converges\#}[f,a,B,max] : \Longleftrightarrow \underset{\varepsilon\in B}{\forall}\ \underset{N=1,\ldots,max}{\exists}\ \mathsf{is\text{-}closer\#}[f,a,\varepsilon,N]$$

with is-closer# being a finitary version of is-closer in a similar fashion. However, the experiments possible on the level of converges# seem to be not eminently helpful. We consider it more valuable to direct the students more into the "choose ε–find N"-game similar to the interactive graphic widget described earlier. For this we introduce a new predicate stays-closer#, which checks *for given* ε the existence of a proper N such that is-closer#$[f,a,\varepsilon,N]$.

Definition["stays closer on finite segment", any$[f,a,\varepsilon,max,N]$,

$$\mathsf{stays\text{-}closer\#}[f,a,\varepsilon,max] : \Longleftrightarrow \underset{N=1,\ldots,max}{\exists}\ \mathsf{is\text{-}closer\#}[f,a,\varepsilon,N]$$
$$\mathsf{is\text{-}closer\#}[f,a,\varepsilon,N] : \Longleftrightarrow \underset{n=N,\ldots,N+30}{\forall}\ |f_n-a|<\varepsilon \qquad]$$

Note that for the universal quantifier in is-closer# we hard-coded the upper limit instead of introducing another parameter. This serves for keeping the focus of the computations on the essential things. It is the responsibility of the designer of a learning unit to configure the experiments in an instructive manner so that the students can benefit from their investigations. Of course, there is also room for the students to invent their own experiments. Changing the $\exists$-quantifier in the definition of stays-closer# into the $\ni$-quantifier[2], we can even let *Theorema* compute "such an N", for which we have is-closer#$[f,a,\varepsilon,N]$, i.e. we define a function that actually finds the threshold for given ε.

[1] We append a '#' to the names in order to distinguish the finitary notions from the original definitions. In *Theorema*, however, we could even leave the names unchanged in order to emphasize the connection to the underlying infinite concept, because in every *Theorema* command such as `Compute` or `Prove` there is the possibility to explicitly refer to the knowledge being used. It is a question of didactics whether to use the same names or different names, where both approaches have their pros and cons.

[2] The *Theorema* language provides the special quantifier $\underset{x}{\ni} P$ to denote "such an x for which P holds". When using a finite range for x then expressions involving an $\ni$-quantifier can effectively be computed using the *Theorema* command `Compute` and they return in fact the first such x for which P holds. This language construct is mostly used in what is commonly called an *implicit definition*.

Definition["threshold on finite segment", any$[f, a, \varepsilon, max]$,
threshold#$[f, a, \varepsilon, max] :=$ $\underset{N=1,\ldots,max}{\ni}$ is-closer#$[f, a, \varepsilon, N]$]

The *Theorema* command `Compute` can be used to perform computations, and we stimulate *Theorema* computations of the following fashion:

In[5]:= `Compute`$[\langle$stays-closer#$[g, 0, 0.1^i, 200]_{i=1,\ldots,5} \rangle]$

Out[5]= $\langle$True, True, True, True, False$\rangle$

In[6]:= `Compute`$[\langle$stays-closer#$[g, 0, 0.1^i, 500]_{i=1,\ldots,5} \rangle]$

Out[6]= $\langle$True, True, True, True, True$\rangle$

In[7]:= `Compute`$[\langle$threshold#$[g, 0, 0.1^i, 500]_{i=1,\ldots,5} \rangle]$

Out[7]= $\langle 4, 13, 44, 140, 446\rangle$

In these computations we chose a fixed bound up to which we search for an N. Indeed, an additional didactic goal in these computational experiments, which is not particulary connected to "convergence of sequences", is that students also experience an important limitation of this "naive finite search" technique: In case of not finding an N up to some bound *max* one can never be secure, whether there *really does not exist* an N or just *max* needs to be increased. The two computations above illustrate this effect and turn a False into True when raising the upper search limit from 200 to 500. The "dual phenomenon" can be observed of course with finitary versions of the universal quantifier, i.e. if a formula is true for all natural numbers up to some bound *max*, then this does not necessarily force the formula to be true *for all natural numbers*. In fact, we could also just tacitly choose a "sufficiently large upper bound" once, but on the one hand one will observe the increase in computation time for the search and on the other hand we consider it a vital insight to come across the semantics of "there exists" and "for all" this way. This experiment can also serve as a motivation for *upper bounds*, but we do not go into details on this aspect. We now continue for even smaller ε but in the subsequent examples we choose the upper bound for searching the N in dependence of ε.

In[8]:= `Compute`$[\langle$stays-closer#$[g, 0, 0.009^i, 100^i]_{i=1,\ldots,5} \rangle]$

Out[8]= $\langle$True, True, True, True, True$\rangle$

In[9]:= `Compute`$[\langle$threshold#$[g, 0, 0.009^i, 100^i]_{i=1,\ldots,5} \rangle]$

Out[9]= $\langle 14, 156, 1655, 17458, 184038\rangle$

In[10]:= `Compute`$[\langle$stays-closer#$[g, 0.001, 0.1^i, 10^i]_{i=1,\ldots,5} \rangle]$

Out[10]= $\langle$True, True, True, False, False$\rangle$

In[11]:= $\texttt{Compute}[\langle \mathsf{threshold}\#[g, 0.001, 0.1^i, 10^i]_{i=1,\ldots,5} \rangle]$

Theorema::impossible: "such a ..." cannot be computed if such an object does not exist.
Out[11]= $\langle 4, 13, 31, \text{Null}, \text{Null} \rangle$

These computations reflect exactly the inspection of finite segments of the sequence in the interactive graphical widget described in the preceding section. Figure 3 shows that g seems to stay closer than 0.1^3 to 0.001 for $n \geq 31$, hence

$$\mathsf{stays\text{-}closer}\#[g, 0.001, 0.1^3, 10^3] = \mathsf{True} \quad \text{and}$$

$$\mathsf{threshold}\#[g, 0.001, 0.1^3, 10^3] = 31,$$

whereas it does not anymore stay closer than 0.1^4 resulting in

$$\mathsf{stays\text{-}closer}\#[g, 0.001, 0.1^4, 10^4] = \mathsf{False}$$

and the error message when trying to compute the threshold. We urge the students to apply this type of cross-check whenever possible: *check (by computation)* the properties read-off a visualization and try to also *visualize* results obtained in computations. The above computations together with previous visualizations should strengthen the conjectures already guessed before, e.g. $\mathsf{converges}[g, 0]$ and $\neg\mathsf{converges}[g, 0.001]$.

5.1.3 Towards Proving the Conjectures

Inspired by the computation of the threshold on a finite segment of the sequence, the students might now wonder whether they can compute the threshold for *the entire sequence*. For the conjecture $\mathsf{converges}[g, 0]$ the question is, as already elaborated in equation (2), whether it is possible to find a natural number N, such that for chosen values of ε

$$\left| \frac{2}{n^2 + 3n} - 0 \right| < \varepsilon \tag{3}$$

holds *for all* $n \geq N$. We do not expect that an "untrained student" recognizes formula (3) as the problem of "solving an inequality"—note that the quantity N to be found does not even occur in the inequality itself! We therefore provide a hyperlink to the CreaComp-MathWorld unit on inequalities, where typical variants of what it can mean to "solve an inequality" are presented. In particular, we expect that students are familiar with computer-support for solving inequalities in *Mathematica* using the standard add-on package `Algebra‘InequalitySolve‘`. We hope that those tools might be of help for solving our problem (3) and recommend the following experiments to the students:

In[12]:= $\texttt{InequalitySolve}[\left|\frac{2}{n^2+3n} - 0\right| < 0.01^2, n]$
Out[12]= $n < -142.929 \vee n > 139.929$

In[13]:= `InequalitySolve`$[n > 0 \wedge \left|\frac{2}{n^2+3n} - 0\right| < 0.01^2, n]$
Out[13]= $n > 139.929$

This says that the given inequality holds for all (real numbers[3]) n as soon as $n > 139.929$, therefore, it holds also for all natural numbers $n \geq \lceil 139.929 \rceil$, i.e. $n \geq 140$, hence one possible choice for the requested natural number N is 140 and every natural number greater 140 would do as well. A comparison with the *Theorema* computations on finite segments done before is of course suggested.

In[14]:= `Table`[`InequalitySolve`$[n > 0 \wedge \left|\frac{2}{n^2+3n} - 0\right| < 0.01^i, n], \{i, 1, 5\}]$
Out[14]= $n > 12.7215, n > 139.929, n > 1412.71, n > 14140.6, n > 141419.9$

and we now go for non-numeric solutions. In a first attempt, we specify non-numeric values for ε and see what happens[4]:

In[15]:= `Table`[`InequalitySolve`$[n > 0 \wedge \left|\frac{2}{n^2+3n} - 0\right| < \left(\frac{1}{100}\right)^i, n], \{i, 1, 5\}]$
Out[15]=
$$n > \tfrac{1}{2}\left(-3 + \sqrt{809}\right)$$
$$n > \tfrac{1}{2}\left(-3 + \sqrt{80009}\right)$$
$$n > \tfrac{1}{2}\left(-3 + \sqrt{8000009}\right)$$
$$n > \tfrac{1}{2}\left(-3 + \sqrt{800000009}\right)$$
$$n > \tfrac{1}{2}\left(-3 + \sqrt{80000000009}\right)$$

It looks as if the solution follows a certain pattern depending on the given ε or, in other words, for every ε we would find a solution by just substituting in the above pattern. Of course, also this is only a guess based on finitely many samples. But, using the full power of computer-algebra hidden in `InequalitySolve`, we get the pattern valid for all $\varepsilon > 0$:

In[16]:= `InequalitySolve`$[n > 0 \wedge \left|\frac{2}{n^2+3n} - 0\right| < \varepsilon, \{\varepsilon, n\}]$
Out[16]= $\varepsilon > 0 \wedge n > -\frac{3}{2} + \frac{1}{2}\sqrt{\frac{9\varepsilon+8}{\varepsilon}}$

This tells us for every $\varepsilon > 0$ how we need to choose N such that formula (3) holds for all $n \geq N$, i.e. $\mathsf{converges}[g, 0]$. On the other hand,

In[17]:= `InequalitySolve`$[n > 0 \wedge \left|\frac{2}{n^2+3n} - \left(\frac{1}{10}\right)^3\right| < \varepsilon, \{\varepsilon, n\}]$
Out[17]=
$$\left(0 < \varepsilon < \tfrac{1}{1000} \wedge -\tfrac{3}{2} + \tfrac{1}{2}\sqrt{\tfrac{9000\varepsilon+8009}{1000\varepsilon+1}} < n < -\tfrac{3}{2} + \tfrac{1}{2}\sqrt{\tfrac{9000\varepsilon-8009}{1000\varepsilon-1}}\right) \vee$$
$$\left(\varepsilon \geq \tfrac{1}{1000} \wedge n > -\tfrac{3}{2} + \tfrac{1}{2}\sqrt{\tfrac{9000\varepsilon+8009}{1000\varepsilon+1}}\right)$$

shows that for $\mathsf{converges}[g, \left(\frac{1}{10}\right)^3]$ we find an appropriate N only for $\varepsilon \geq \frac{1}{1000}$, whereas for $0 < \varepsilon < \frac{1}{1000}$ the inequality always only holds for numbers up to $-\frac{3}{2} + \frac{1}{2}\sqrt{\frac{9000\varepsilon-8009}{1000\varepsilon-1}}$.

[3] As an interesting sideline, it needs to be discussed that *Mathematica*'s `InequalitySolve` solves inequalities over the reals, whereas our concrete problem, in fact, is to solve an inequality over the natural numbers. How do these two problems relate to each other?

[4] The students are familiar with *Mathematica*'s strategy to give symbolic answers for purely symbolic input and to automatically switch to numerical output as soon as decimals appear in the input.

5.1.4 A Complete Proof of the Conjecture

In the previous section, we concentrated only on the final inequality (3), but the reduction of the original statement converges$[g, 0]$ to formula (3) has been given only on an informal level. We now switch back to *Theorema* in order to produce a complete proof of the following conjecture:

Proposition["g converges to 0", converges$[g, 0]$]

In a first attempt, we try to let the *Theorema* PCS-prover, see [Buchberger, 2001], generate a proof of this proposition using only the relevant definitions in the knowledge base. The *Theorema* command for this looks as follows:

In[18]:= `Prove`[Proposition["g converges to 0"],
using→ {Definition["g"], Definition["convergence"]}, by → PCS],

where Definition["g"] refers to the definition of our example sequence g. Of course, the above proof fails, because the PCS prover does not have any knowledge about absolute value, inequalities, or arithmetic built in. However, the student can inspect the nicely formatted failing proof in a separate *Mathematica* notebook and, thus, again enter an interactive phase. As the students can easily see in the proof, the prover misses to find an N^* such that for arbitrary but fixed positive ε_0

$$N^* \in \mathbb{N} \wedge \underset{n}{\forall}\, n \in \mathbb{N} \wedge n \geq N^* \Rightarrow \left| \frac{2}{n^2 + 3n} - 0 \right| < \varepsilon_0. \tag{4}$$

From the session playing with *Mathematica*'s `InequalitySolve` as discussed in Section 5.1.3 we know, that this inequality is fulfilled as soon as $n > X$ with X as an abbreviation for $-\frac{3}{2} + \frac{1}{2}\sqrt{\frac{9\varepsilon+8}{\varepsilon}}$. We can formulate an auxiliary lemma and allow the prover to use it.

Lemma["inequality:abs", any$[n, \varepsilon]$,

$\varepsilon > 0 \wedge n > -\frac{3}{2} + \frac{1}{2}\sqrt{\frac{9\varepsilon+8}{\varepsilon}} \Rightarrow \left| \frac{2}{n^2+3n} - 0 \right| < \varepsilon$]

In order to facilitate experiments with the provers the CreaComp units also supply skeletons for calling *Theorema* provers correctly, e.g.

`Prove`[Proposition["g converges to 0"], using→ {Definition["example"],
Definition["convergence"], □}, by → PCS]

where a '□' indicates the open positions that need to be filled by the students. Filling in Lemma["inequality:abs"] into the `Prove`-skeleton and executing the resulting command will generate a new proof notebook, but still the generated proof is not successful! Figure 4 shows the essential part of the failing proof. The main line of explanation how to find a successful proof will then be:

- Observe that the failing goal (5) as displayed in Figure 4 has the form

$$\text{find an } N^* \text{ such that } \ldots \wedge \underset{n}{\forall}\, (\ldots \wedge n \geq N^* \Rightarrow n > X).$$

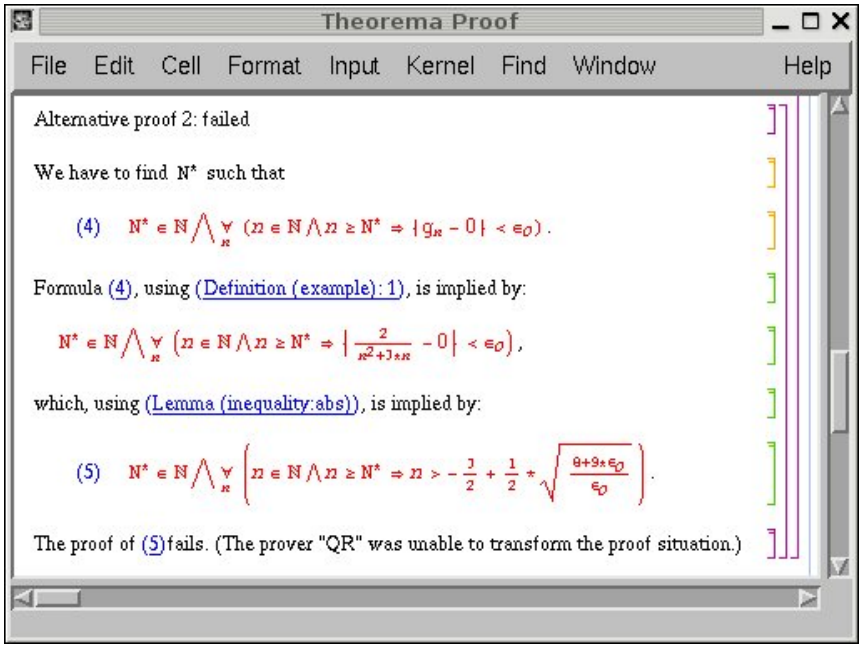


Figure 4: The crucial steps in a failing *Theorema* proof.

Turning this into a proof goal of the form

$$\text{find an } N^* \text{ such that } \ldots \wedge \underset{n}{\forall} (\ldots \wedge n \geq N^* \Rightarrow n \geq \bar{X})$$

would certainly make $\bar{X}$ a plausible candidate for N^*.

- Observe how Lemma["inequality:abs"] has been applied in the proof: the lemma is a universally quantified implication and a sub-formula of the proof goal is *an instance of its right-hand side*, hence, the sub-formula is replaced *by the corresponding instance of the lemma's left-hand side*[5].
- Try to find another auxiliary lemma of essentially the form

$$\underset{x}{\forall}\, x \geq \bar{X} \Rightarrow x > X$$

in order to further rewrite the goal (5) in a similar fashion.

For the last step—finding appropriate auxiliary knowledge—there are again various possibilities to continue depending primarily on didactic considerations.

[5]We want to teach methods how to prove mathematical statements, thus we assume that these students are familiar with notions like "implication", "instance", etc.

i. The appropriate auxiliary lemmata can simply be communicated to the students (with or without proof) and *Theorema* would automatically generate a successful and correct proof. Still, the students can study the proof and possibly learn from that.

ii. Large databases of mathematical knowledge can be provided or the students learn how to access external databases in order to *search for appropriate* knowledge. Knowledge deemed to be of help can be passed to the prover and the students can experience the influence on the resulting proof.

iii. Arbitrary additional knowledge can be given to the prover, but proofs must be supplied for all statements that are used in the prover's knowledge base[6]. This approach is very instructive if not only an isolated proof of Proposition["g converges to 0"] is of interest but the educational goal is also to expose "how to build up mathematics hierarchically" in the spirit of theory exploration, see [Buchberger, 1999].

Possible Stages when Building up the Necessary Theory

This section will not be part of the main flow of the CreaComp unit on convergence. It will only be hyper-linked in order to be accessible for students and teachers that want to follow the didactic approach (iii) described above. Here we will animate the students to try to find useful available knowledge for completing the proof. As already pointed out above, the structure of such a lemma should be $\underset{x}{\forall}\ x \geq \bar{X} \Rightarrow x > X$, i.e. we need to come up with some $\bar{X}$ such that this formula holds for X as introduced above. There is ample room for experiments and we again govern the students' creativity by offering a certain skeleton for the lemma, namely a generalization that is valid for arbitrary real numbers in place of X like

$$\underset{x,y}{\forall}\ x \geq \square \Rightarrow x > y.$$

The easiest such lemma would probably be something like
Lemma["inequality:reals", any[x, y],
$x \geq y + 1 \Rightarrow x > y$].
Plugging Lemma["inequality:reals"] into the PCS prover skeleton would reduce the original proof to prove

$$-\frac{3}{2} + \frac{1}{2}\sqrt{\frac{9\varepsilon_0 + 8}{\varepsilon_0}} + 1 \in \mathbb{N},$$

[6]Note, that this is not necessarily required in *Theorema*. Even unproved formulae can go into the knowledge base, and a *Theorema* proof must always be understood *relative* to the knowledge base given in the `Prove`-command. This is quite useful, because through this mechanism not every proof needs to go back to the axioms, which is the natural approach also in human proving.

which, for arbitrary ε_0 cannot be proven. We see that this lemma will not be of further help for the main proof and we must go for some *natural number* instead of $y+1$ on the left-hand side in our lemma. Inspired by the experiments shown in Section 5.1.3 one might come up with e.g.

Lemma["ceiling", any$[x, y]$,

$x \geq \lceil y \rceil \Rightarrow x > y$]

and provide this as additional knowledge for the prover. *Theorema* will now produce a successful proof! However, when proceeding now with a proof of Lemma["ceiling"] it will turn out that this lemma cannot be proven because, in fact, the statement is *not true* for all x and y. As before, thorough analysis of the failing proof could lead to the following reformulation:

Lemma["ceiling", any$[x, y]$,

$x \geq \lceil y+1 \rceil \Rightarrow x > y$ "ineq"
$\lceil y \rceil \in \mathbb{N}$ "nat"]

This lemma will then allow a successful proof of converges$[g, 0]$.

The Successful Proof

If the previous section on how to build up the appropriate knowledge has been skipped then only the final version of Lemma["ceiling"] together with the skeleton for calling the prover will be offered here. Only a few seconds after calling the prover, *Theorema* will present the successful proof[7] in a separate notebook in the style as shown in Figure 4. For better readability, we exhibit the proof typeset in LaTeX instead of taking a screen-shot. We thereby loose some of the didactically outstanding features of *Theorema*'s proof presentation in the notebook such as collapsing entire proof branches by mouse-click, the distinction between proof goals and proof assumptions by using different colors, or cross-references to formulae used in a proof step as active elements that display the referenced formula when clicked. We present the proof of the original proposition converges$[g, 0]$ together with the proofs of the auxiliary Lemma["ceiling"]. We want to emphasize that the entire proof as displayed in this paper including all explanatory text, the formula labels etc. can be generated completely automatically in the *Theorema* system.

Prove:

(Proposition (g converges to 0)) converges$[g, 0]$,

under the assumptions:

(Definition (g)) $\underset{n}{\forall}\, g_n := \frac{2}{n^2+3*n}$,

---

[7] We show a proof where the definition of "convergence" does not use the auxiliary notion "is-closer" in order to save one "uninteresting" rewrite step in the proof. However, the message to students on "how to build up mathematics hierarchically" will just be the contrary: If possible, introduce auxiliary notions *for each quantifier* in the formula and explore these concepts one after the other concentrating on only one quantifier!

(Definition (convergence)) $\underset{f,a}{\forall} \mathsf{converges}[f,a] :\Longleftrightarrow \underset{\substack{\varepsilon\\ \varepsilon>0}}{\forall} \underset{N\in\mathbb{N}}{\exists} \underset{\substack{n\in\mathbb{N}\\ n\geq N}}{\forall} |f_n - a| < \varepsilon,$

(Lemma (inequality:abs)) $\underset{n,\varepsilon}{\forall} \varepsilon > 0 \wedge n > -\frac{3}{2} + \frac{1}{2}\sqrt{\frac{9\varepsilon+8}{\varepsilon}} \Rightarrow \left|\frac{2}{n^2+3n} - 0\right| < \varepsilon,$

(Lemma (ceiling): ineq) $\underset{x,y}{\forall} x \geq \lceil y+1 \rceil \Rightarrow x > y,$

(Lemma (ceiling): nat) $\underset{y}{\forall} \lceil y \rceil \in \mathbb{N}.$

Formula (Proposition (g converges to 0)), using (Definition (convergence)), is implied by:

(1) $\underset{\substack{\varepsilon\\ \varepsilon>0}}{\forall} \underset{N\in\mathbb{N}}{\exists} \underset{\substack{n\in\mathbb{N}\\ n\geq N}}{\forall} |g_n - 0| < \varepsilon.$

We assume

(2) $\varepsilon_0 > 0,$

and show

(3) $\underset{N\in\mathbb{N}}{\exists} \underset{\substack{n\in\mathbb{N}\\ n\geq N}}{\forall} |g_n - 0| < \varepsilon_0.$

We have to find N^* such that

(4) $N^* \in \mathbb{N} \wedge \underset{n}{\forall} n \in \mathbb{N} \wedge n \geq N^* \Rightarrow |g_n - 0| < \varepsilon_0.$

Formula (4), using (Definition (g)), is implied by:

$N^* \in \mathbb{N} \wedge \underset{n}{\forall} n \in \mathbb{N} \wedge n \geq N^* \Rightarrow \left|\frac{2}{n^2+3*n} - 0\right| < \varepsilon_0,$

which, using (Lemma (inequality:abs)) and (2), is implied by:

$N^* \in \mathbb{N} \wedge \underset{n}{\forall} n \in \mathbb{N} \wedge n \geq N^* \Rightarrow n > -\frac{3}{2} + \frac{1}{2}\sqrt{\frac{9\varepsilon_0+8}{\varepsilon_0}},$

which, using (Lemma (ceiling): ineq), is implied by:

(5) $N^* \in \mathbb{N} \wedge \underset{n}{\forall} n \in \mathbb{N} \wedge n \geq N^* \Rightarrow n \geq \left\lceil -\frac{3}{2} + \frac{1}{2}\sqrt{\frac{9\varepsilon_0+8}{\varepsilon_0}} + 1 \right\rceil.$

Partially solving it, formula (5) is implied by

(6) $\left\lceil -\frac{3}{2} + \frac{1}{2}\sqrt{\frac{9\varepsilon_0+8}{\varepsilon_0}} + 1 \right\rceil \in \mathbb{N} \wedge N^* = \left\lceil -\frac{3}{2} + \frac{1}{2}\sqrt{\frac{9\varepsilon_0+8}{\varepsilon_0}} + 1 \right\rceil.$

We can partially solve (6). By taking $N^* \leftarrow \left\lceil -\frac{3}{2} + \frac{1}{2}\sqrt{\frac{9\varepsilon_0+8}{\varepsilon_0}} + 1 \right\rceil$, formula (6) is implied by

(7) $\left\lceil -\frac{3}{2} + \frac{1}{2}\sqrt{\frac{9\varepsilon_0+8}{\varepsilon_0}} + 1 \right\rceil \in \mathbb{N}.$

Formula (7) is proved because it is an instance of (Lemma (ceiling): nat).

□

Prove:

(Lemma (ceiling): ineq) $\underset{x,y}{\forall} x \geq \lceil y+1 \rceil \Rightarrow x > y$,

under the assumptions:

(Lemma (inequality:ceiling)) $\underset{x,y}{\forall} x \geq \lceil y \rceil \Rightarrow x \geq y$,

(Lemma (inequality:reals)) $\underset{x,y}{\forall} x \geq y+1 \Rightarrow x > y$.

We assume

(5) $x_0 \geq \lceil y_0 + 1 \rceil$,

and show

(6) $x_0 > y_0$.

Formula (5), by (Lemma (inequality:ceiling)), implies:

$x_0 \geq y_0 + 1$,

which, by (Lemma (inequality:reals)), implies:

(7) $x_0 > y_0$.

Formula (6) is true because it is identical to (7).

□

Prove:

(Lemma (ceiling): nat) $\underset{y}{\forall} \lceil y \rceil \in \mathbb{N}$,

under the assumptions:

(Definition (ceiling)) $\underset{y}{\forall} \lceil y \rceil := \underset{n \in \mathbb{N}}{\ni} n \geq y \wedge \underset{m \in \mathbb{N}}{\forall} m \geq y \Rightarrow m \geq n$.

From (Definition (ceiling)) we can infer by expansion of the "such that"-quantifier

(1) $\underset{y,n}{\forall} \lceil y \rceil = n \Rightarrow n \in \mathbb{N} \wedge n \geq y \wedge \underset{m \in \mathbb{N}}{\forall} m \geq y \Rightarrow m \geq n$,

(2) $\underset{y}{\forall} \lceil y \rceil \in \mathbb{N} \wedge \lceil y \rceil \geq y \wedge \underset{m \in \mathbb{N}}{\forall} m \geq y \Rightarrow m \geq \lceil y \rceil$.

Formula (2) is simplified to:

(4) $\underset{y}{\forall} \lceil y \rceil \in \mathbb{N} \wedge \underset{y}{\forall} \lceil y \rceil \geq y \wedge \underset{m \in \mathbb{N}}{\forall} m \geq y \Rightarrow m \geq \lceil y \rceil$.

Formula (Lemma (ceiling): nat) is true because it is identical to (4.1).

□

Note that in the proof of the first part of the lemma we use an additional property Lemma["inequality:ceiling"], whereas the proof of the second part goes back to the definition of ceiling using again the "such that"-quantifier already mentioned in Section 5.1.2. We do not show the proof of Lemma["inequality:ceiling"] anymore because we also want to demonstrate that "formal proving" need not necessarily always mean "proving by going back to the axioms". Of course, if students are eager to also prove this lemma, they may try to do so.

The reader may notice in the main proof that the choice $N^* \leftarrow \lceil \ldots \rceil$, which comes from the auxiliary Lemma["ceiling"], is probably not "the best possible" choice for N^*, because it gives in general not necessarily the smallest possible N^*. We want to emphasize here—but also when teaching students—that this is not an error! It is only in traditional maths education that both teachers and students are often spoilt to believe that only the best possible choice is the correct one. Our setting now gives the students the opportunity to experiment with different auxiliary knowledge and to experience how the choice for N^* depends on the available knowledge. Since the reproduction of various variants of the proof is no effort, students can try out what they believe to be valid choices. They might come up with a lemma stating for any x, y

$$x \geq \lfloor y \rfloor + 1 \Rightarrow x > y,$$

which then results in a different instantiation for N^*. Whatever auxiliary knowledge they provide, we will encourage them to also prove what they claim. Others might realize that *any natural number* greater than a given real number would do. By the unboundedness of the natural numbers and the transitivity of $<$ we can describe the existence of such a natural number by

$$\underset{y \in \mathbb{R}}{\forall}\ \underset{n \in \mathbb{N}}{\exists}\ \underset{x}{\forall}\ x \geq n \Rightarrow x > y.$$

The PCS prover will then introduce a Skolem function $n_0[y]$ expressing the n depending on y and, thus, it will instantiate N^* by a term constructed with the help of this Skolem function, i.e. $N^* \leftarrow n_0[-\frac{3}{2} + \frac{1}{2}\sqrt{\frac{9\varepsilon_0+8}{\varepsilon_0}}]$. We think that this type of experiments offers an entirely new perspective on how mathematical theories evolve and these experiments can only be undertaken with computer-supported proof generation.

6 Conclusion

In this paper we describe a computer-based learning environment that aims at the stimulation of the students' creativity when learning mathematics. The key ingredient in this framework is the interaction of the student with a computer-algebra system and a theorem proving system. We use the computer-algebra system—unlike other approaches that utilize the system's command language in order to execute available computer-algebra algorithms—mainly for enabling

interactive graphical experiments, which help the students to sharpen their intuition about mathematical concepts. Whereas we see the computer algebra system's role predominantly in the phase of "aquisition of new knowledge", the theorem proving system should assist the students in veryfing their intutions acquired during their visual and computational experiments, thus serving mostly the didactic principle of "strengthening". In the use of the theorem proving system we also follow the paradigm of student-centered learning by designing interaction patterns according to which the students play with the prover. In these experiments, we intend to teach the methodology in which mathematical knowledge is built up, i.e. learning from failing proofs *how* and *which* auxiliary knowledge must be formulated in order to succeed with the proof.

The methods envisaged have been described in all detail in a case study on some aspects of convergent real-valued sequences. The emphasis of our approach lies on the approach that computer-support is not only given through visualization, animation, and computation but also for the formal argumentation about truth or falsity of mathematical properties. In this case study we presented a fully automated prover available in the *Theorema* system, and we focused on user interaction on the level of building up the appropriate knowledge base needed for a successful proof. All provers provided in the *Theorema* system can however be run also in interactive mode, where additional interaction patterns can be applied, such as instantiating formulae during the proof, adding/removing formulae during the proof, etc. We will of course exploit these capabilities in other parts of the material.

Another important apsect of computer-based learning, which has not been discussed in this case study, is testing and assessment. Tools for computer-supported assessment have already been developed in the frame of MeetMath, see [Saminger, 2002], and we will re-use available technology also in our CreaComp units. Finally, we want to provide educational material that can be used both in classroom and for self-study and that involves the learner in computer experiments that enhance both their intuition and their formal argumentation capabilities.

Acknowledgements

This work is done within the project "CreaComp: E-Schulung von Kreativität und Problemlösekompetenz" at the Johannes Kepler University of Linz sponsored by the Upper Austrian government. The authors would like to thank the project leaders B. Buchberger, E. P. Klement, and G. Pilz for providing the frame for this enjoyable research.

References

[Andrews *et al.*, 2004] R.B. Andrews, C.E. Brown, F. Pfenning, M. Bishop, S. Issar, and H.Xi. ETPS: A System to Help Students Write Formal Proofs. *Journal of Automated Reasoning*, 32:75–92, 2004.

[Borwein, 2005] J. M. Borwein. The Experimental Mathematician: The Pleasure of Discovery and the Role of Proof. *International Journal of Computers for Mathematical Learning*, 10(2):75–108, May 2005.

[Buchberger *et al.*, 1997] B. Buchberger, T. Jebelean, F. Kriftner, M. Marin, E. Tomuta, and D. Vasaru. A Survey of the Theorema project. In W. Kuechlin, editor, *Proceedings of ISSAC'97 (International Symposium on Symbolic and Algebraic Computation, Maui, Hawaii, July 21-23, 1997), ACM Press 1997.*, pages 384–391, 1997.

[Buchberger *et al.*, 2000] B. Buchberger, C. Dupre, T. Jebelean, F. Kriftner, K. Nakagawa, D. Vasaru, and W. Windsteiger. The Theorema Project: A Progress Report. In M. Kerber and M. Kohlhase, editors, *Symbolic Computation and Automated Reasoning (Proceedings of CALCULEMUS 2000, Symposium on the Integration of Symbolic Computation and Mechanized Reasoning)*, pages 98–113. St. Andrews, Scotland, Copyright: A.K. Peters, Natick, Massachusetts, 6-7 August 2000.

[Buchberger *et al.*, 2005] B. Buchberger, A. Craciun, T. Jebelean, L. Kovacs, T. Kutsia, K. Nakagawa, F. Piroi, N. Popov, J. Robu, M. Rosenkranz, and W. Windsteiger. Theorema: Towards Computer-Aided Mathematical Theory Exploration. *Journal of Applied Logic*, 2005. To appear.

[Buchberger, 1990] B. Buchberger. Should Students Learn Integration Rules? *ACM SIGSAM Bulletin*, 24(1):10–17, January 1990.

[Buchberger, 1996] B. Buchberger. Symbolic Computation: Computer Algebra and Logic. In F. Bader and K.U. Schulz, editors, *Frontiers of Combining Systems, Proceedings of FROCOS 1996 (1st International Workshop on Frontiers of Combining Systems), March 26-28, 1996, Munich*, volume 3 of *Applied Logic Series*, pages 193–220. Kluwer Academic Publisher, Dordrecht - Boston - London, The Netherlands, 1996.

[Buchberger, 1999] B. Buchberger. Theory Exploration Versus Theorem Proving. In A. Armando and T. Jebelean, editors, *Electronic Notes in Theoretical Computer Science*, volume 23-3, pages 67–69. Elsevier, 1999. CALCULEMUS Workshop, University of Trento, Trento, Italy.

[Buchberger, 2001] B. Buchberger. The PCS Prover in Theorema. In R. Moreno-Diaz, B. Buchberger, and J.L. Freire, editors, *Proceedings of EUROCAST 2001 (8th International Conference on Computer Aided Systems Theory - Formal Methods and Tools for Computer Science)*, Lecture Notes in Computer Science 2178, pages 469–478. Las Palmas de Gran Canaria, Copyright: Springer - Verlag Berlin, 19-23 February 2001.

[Drijvers, 2002] P. Drijvers. Learning Mathematics in a Computer Algebra Environment: Obstacles are Opportunities. *Zentralblatt für Didaktik der Mathematik*, 34(5):221–228, 2002.

[Hanna, 2000] G. Hanna. Proof, Explanation and Exploration: An overview. *Educational Studies in Mathematics*, 44:5–23, 2000.

[Housman and Porter, 2003] D. Housman and M. Porter. Proof Schemes and Learning Strategies of Above-average Mathematics Students. *Educational Studies in Mathematics*, 53:139–158, 2003.

[Recio and Godino, 2001] A.M. Recio and J.D. Godino. Institutional and Personal Meanings of Mathematical Proof. *Educational Studies in Mathematics*, 48:83–99, 2001.

[Saminger, 2002] S. Saminger. MeetMATH — visualizations and animations in a didactic framework. In M. Borovcnik and H. Kautschitsch, editors, *Technology in Mathematics Teaching (Special groups and working groups). Proceedings of ICTMT 5, Klagenfurt (Austria)*, volume 26 of *Schriftenreihe Didaktik der Mathematik*, pages 217–222, Wien, 2002. öbv & hpt Verlagsgesellschaft.

[Sommer and Nuckols, 2004] R. Sommer and G. Nuckols. A Proof Environment for Teaching Mathematics. *Journal of Automated Reasoning*, 32:227–258, 2004.

[Wolfram, 2003] Stephen Wolfram. *The Mathematica Book*. Wolfram Media, Inc., 5th edition, 2003.

CreaComp: Computer-Supported Experiments and Automated Proving in Learning and Teaching Mathematics*

Günther Mayrhofer[1] and Susanne Saminger[2] and Wolfgang Windsteiger[3]

[1] *Institut für Algebra, guenther.mayrhofer@students.jku.at*
[2] *Institut für Wissensbasierte Mathematische Systeme, susanne.saminger@jku.at*
[3] *Institut für Symbolisches Rechnen, wolfgang.windsteiger@risc.uni-linz.ac.at*
JKU Linz, A-4040 Linz, Austria

We present an environment for learning and teaching mathematics that aims at inspiring the creative potential of students by enabling the learners to perform various kinds of interactive experiments during their learning process. Computer interactions are both of visual and purely formal mathematical nature, where the computer-algebra system *Mathematica* powers the visualization of mathematical concepts and the tools provided by the theorem proving system *Theorema* are used for the formal counterparts. We present a case study on the concept of equivalence relations and set partitions, in which we demonstrate the entire bandwidth of computer-support that we envision for modern learning environments for mathematics ranging from "getting first ideas and intuitions" over "checking the validity of first ideas on a wide variety of examples" until "rigorously proving one's own conjectures".

Keywords: Automated Theorem Proving, Computer-Algebra Systems, Maths Education

1. Introduction

Both in classroom and in scenarios of distance learning of mathematics the use of computer-algebra systems has become more and more popular. Recently computer-support focuses on *(symbolic) computations*, e.g. calculating limits, derivatives, integrals, and *visualization*, e.g. plotting real-valued sequences or functions: the availability of powerful algorithms allows to perform calculations even in situations when the method would re-

*This work is done within the project "CreaComp: E-Schulung von Kreativität und Problemlösekompetenz" at the Johannes Kepler University of Linz sponsored by the Upper Austrian government.

quire a tremendous computational effort when executed "by hand" or when the algorithms are not known to the students at a certain level of education. Additionally, different visual representations of the computational objects contribute to qualitative data analysis and exploration of non-obvious mathematical relationships. It is clear that modern symbolic computation systems are powerful enough to carry out all computations done in high-school mathematics and most of the computations taught at undergraduate university level. Hence, the question of which methods should be taught to students in the first place and for which tasks we rely on the help of the computer is of utmost importance. There is no absolute answer to this and the "White-Box Black-Box principle", see [1], usually serves as the didactic guideline, by which, depending on the didactical goals certain methods are taught in detail in one phase of education (the white-box phase) whereas they can be applied as black-boxes in later phases.

On the other hand, most of the available electronic learning material for mathematics only rarely focus on computer-support for acquiring such crucial mathematical skills as "exact formulation of mathematical properties" and "rigorously proving mathematical properties correct", for exceptions see e.g. [2,3].

In this paper we present the CreaComp project, whose main goal is to develop electronic course material for self-study and also for use in classroom. In its current state, CreaComp comprises approximately 15 (only loosely connected) learning units on an undergraduate level, such as e.g. equivalence relations, polynomial interpolation, or Markov processes. The computer-aid provided in CreaComp units follows the didactical guidelines set up in the MeetMath project, see Section 2, and it promotes an approach of self-paced learning that has been centered around "gaining insight into mathematical concepts through computational and graphical interaction". In addition to computational and graphical interaction the CreaComp approach now adds *formal reasoning* as a third important component by integrating the *Theorema* system, see [4–6]. *Theorema* is designed to become a uniform environment in which a mathematician gets support during all periods of his/her mathematical occupation. For the CreaComp project, however, *Theorema* mainly provides the mathematical language and the possibility of fully automated or interactive generation of mathematical proofs. Both MeetMath and *Theorema* are based on the well-known computer algebra system *Mathematica*, see [7].

The CreaComp approach aims to combine the experimental approach of discovering mathematics through interactive visualizations and compu-

tations with the rigorous approach of proving every claim that is made. By using the *Theorema* system for automatically generating human-readable proofs, an experimental flavor can also be given to the formal part, i.e. students can observe with little effort how the *available knowledge influences success or non-success of a proof*, how *tacit assumptions* are often used in human arguments, or how *mathematical theories evolve* from sometimes simple definitions.

In the sequel we will briefly introduce the constituent components MeetMath and *Theorema* and their combination in Section 2, the main part will be an exemplary case study presenting parts of a CreaComp unit on equivalence relations and partitions in Section 3.

Mathematically, the learning unit on equivalence relations and set partitions starts from

- the *definition of binary relations* and elementary properties such as *reflexivity*, *symmetry*, and *transitivity*,
- then introduces *classes* and *factor sets*,
- proceeds with *set partitions* and *induced relations*,
- develops the theorems that *the factor set of an equivalence forms a set partition* and that *the induced relation of a set partition is an equivalence relation*, and
- finally concludes with the theorems that *building the factor set (of an equivalence relation)* and *building the induced relation (of a set partition)* are *inverse* to each other.

At all stages, interactive visualization tools are provided to illustrate the new concepts and their properties in small and easily comprehensible examples. For all theorems as well as for all auxiliary lemmata required in the proofs, we then provide fully automated proofs in natural language to be generated interactively by the students, i.e. we provide an interface to the *Theorema*-provers that allows the student to easily generate fully automated proofs.

2. The CreaComp Project — An Overview

2.1. *MeetMath*

MeetMath denotes a family of interactive mathematics courseware based on *Mathematica* equipped with a Java^{TM}-based navigation. The basic course MeetMath@Business&Economy and the didactic concepts for MeetMath courseware have been initially developed in the

framework of the project IMMENSE (Interactive Multimedia Mathematics Education in Networked universities for Social and Economic sciences) initiated by the Johannes Kepler University Linz already in 1999 (for more detailed information about the project and partners see `http://www.flll.jku.at/meetmath`).

The development of any course material including electronic supplement demands to focus not only on, possibly new, technology but on the corresponding didactic framework as well. MEETMATH course units are structured as "didactical rooms", where different rooms reflect different phases of the learning process. Students can then decide on the sequence and the duration in which they like to visit different parts of the material. Basically, four different types of didactical rooms have been distinguished:

(i) "Motivation/linking": motivation/linking addresses the students knowledge about and their attitude to deal with certain content.
(ii) "Acquisition/confrontation": the central new concepts are presented in a complete and carefully paced argumentation. Within confrontation rooms, relevant information, thoughts, and illustrations should be offered in a way that students are able to build up their own ideas and concepts and/or to modify existing, incorrect concepts.
(iii) "Strengthening": these rooms allow to verify and inspect newly developed concepts and ideas. Experiments and interactive elements enable students to stabilize the concepts. Experiments allow to falsify incorrectly developed concepts or parts of concepts by trial and error. Both success and failure are possible and allowed during strengthening.
(iv) "Assessment": assessment is a necessary tool for supervising the learning process. Since students are responsible for the development of their knowledge, they need some tools for measuring the progress and/or for finding out the status quo.

For more details, we refer to [8].

2.2. *Theorema*

Theorema is a system that intends to bring computer-support during all phases of mathematical activity, such as *defining* new mathematical objects, *performing experiments* on the new objects, *conjecturing* properties of the new objects, *proving* the correctness of the conjectured properties, *developing algorithms*, *running* and *improving* existing algorithms, *visualizing* mathematical data, and many more. The *Theorema* system provides a uniform language and logic, in which many of the above activities can be

carried out, see [9]. The overall design principle of the *Theorema* system is to communicate with the user in "mathematical textbook style". The syntax of the language in *both input and output* is close to "common mathematical notation" including special mathematical characters and two-dimensional syntax as typically used in mathematics. The *Theorema* language is a version of higher-order predicate logic with pre-defined basic mathematical objects such as numbers, sets, and tuples. For algorithmic language constructs such as numbers, finite sets and tuples, and quantifiers with finite ranges the language provides computational semantics in form of algorithms for basic operations in these domains. The main focus in the development of the *Theorema* system over the past years has been put on the development of various general and special-purpose fully automated theorem provers.

Since *Theorema* is built on top of the well-known *Mathematica* system, we use the *Mathematica* notebook front-end as the user-interface for *Theorema*. When generating mathematical proofs, *Theorema* displays the full proof in a separate notebook document with each proof step explained in natural language. The structure of the proof is reflected in the cell structure of the proof notebook, so that the standard *Mathematica* technology of opening/closing nested cells by mouse-click can be used to collapse entire proof branches. This allows the reader to get an overview over the structure of the proof and then to "zoom into" parts of the proof by subsequently opening just the relevant cells. As an alternative to fully automated proof generation the *Theorema* system also allows for interactive proving, see e.g. [10]. For details on the *Theorema* system, the language, and examples of provers implemented in the *Theorema* system we refer to the introductory papers [4–6].

2.3. *The Combination of MeetMath and Theorema*

The most dangerous scenario of computer-supported mathematics is that the extensive use of computation and visualization leads to a tradition of "proof by inspecting particular examples" instead of "proof by mathematical proving". In our view, examples can and should accompany the development of mathematical content, they can contribute to shaping the students' intuition about mathematics, and the content presented in examples is typically well memorized. However, examples—even if many and well-chosen—can (almost) never substitute a proof of a proposition! It is one of the main goals of this project to highlight the importance of rigorous formal mathematical arguments in all facets of mathematical work, including for instance also software development.

The combination of MeetMath and *Theorema*, which is investigated in the frame of the CreaComp-project, therefore aims at providing computer aid for visualization, computation, but most importantly also for proving. Whereas computation and visualization ought to fertilize the *intuition* about mathematical objects, the proving phase should establish and enhance the *understanding* of mathematical argumentation, in particular that "validity in some examples" does not necessarily always coincide with "validity in *all cases*". An additional benefit of a theorem proving system as the student's assistant is that the students must think carefully about tacit assumptions they use in their argumentation, because the automated prover forces them to state all usable knowledge explicitly, see also e.g. [11]. Moreover, we think that by having a machine to give formal proofs of all statements and therefore being forced to fill all gaps in the proofs, the students can better understand the evolution of mathematical theories. This experience, even if maybe not necessary for a pure user of mathematics, we consider as very enlightening for students of mathematics.

The interplay of MeetMath and *Theorema* is facilitated by the common underlying *Mathematica* technology, since both are based on the capabilities of the *Mathematica* notebook-frontend. CreaComp educational units are distributed in the form of *Mathematica* notebooks containing normal text intermixed with formal mathematical texts (definitions, theorems, lemmata, etc.) written in the *Theorema* language. Furthermore we provide visualization commands for certain mathematical objects. In addition to static plots we try to involve the students in actively exploring mathematical properties by providing *interactive visualizations* based on *Mathematica*'s GUIKit, a toolbox supporting the implementation of Java applications fed with *Mathematica* data. Finally, we encourage students to also prove their conjectures after their experiments. In this stage they may use the *Theorema* provers both in interactive or in fully automated mode.

Although the *Mathematica* notebook-frontend is often called a a graphical user interface (GUI), *Mathematica* propagates a command-centered interaction pattern. In order to trigger a *Mathematica* computation, a command has to be typed into the notebook and then needs to be evaluated by pressing certain keys. For the CreaComp interface we make heavy use of interactive notebook elements like buttons and hyperlinks in order to prevent students from struggling with unfamiliar input syntax. Students' experiments are mainly based on common modern user interface actions such as selecting items by clicking radio-buttons or checkboxes, opening dialogs by button-click, etc.

In the spirit of MeetMath's didactical framework, the interactive visualizations serve mainly for motivation and acquisition. After introducing a new mathematical concept we provide tools aiming at graphical visualizations of important properties that are to be investigated in the current unit. The user can interactively "play" with the tool, the on-line help gives instructions which phenomena can and should be observed. These interactive tools are always designed in such a fashion that in addition to pre-defined examples they allow to run user-defined examples as well as randomly generated examples. A symbolic computation system is indispensable as the engine behind these tools, because an instructive visualization often needs the computation of the problem's solution in the background. Symbolic computation methods can be applied for generating pre-computed symbolic solutions depending on example parameters, such that e.g. for visualization of a random example only the example parameters need to be instantiated in the symbolic solution.

During this phase the students get an intuition about the new concept and in the best case they observe some of the intended properties during their experiments. Still staying in an "acquisition room" (see Section 2.1) we then formulate some conjectures in the *Theorema* language, which looks very much like standard mathematical formula language. Changing into a "strengthening room" we then ask the user to prove the conjecture using *Theorema*. Again, we provide a button interface for the prover call in order not to confuse the students with syntax details of calling the appropriate prover in the appropriate fashion thereby distracting them from their main focus, the proof! The learning goal and, thus, the user interaction in this phase consists of choosing the appropriate knowledge base and observing its influence on the generated proof and of investigating possible modifications in the formulation of the conjecture and/or parts of the knowledge in order to obtain a successful proof.

CreaComp consists of several thematic units that are intended for use in standard undergraduate courses for studies in mathematics and computer science as well as in mathematics courses for non-mathematical studies, e.g. business, marketing, social sciences, etc. Units are available for basic set theory, relations, functions, real-valued sequences and limits, continuous functions, fast computations using modular arithmetic, polynomial interpolation, Markov chains, cryptology, Gröbner bases, and other topics to be developed.

In the remainder of this paper, we want to illustrate the structures described above in a case study showing parts of the CreaComp unit on

equivalence relations and set partitions.

3. Case Study - Equivalence Relations and Set Partitions

Following the didactic principles taken from MeetMath, the typical flow of a CreaComp-MathWorld unit is to

- motivate the students by some real-world example,
- present new mathematical concepts by defining new objects or properties,
- let the students experiment with the new entities on concrete data,
- guide the students in their experiments such that possibly they are able to conjecture new properties,
- guide the students in rigorous proofs of their conjectures.

We try to illustrate the flavor of computer-support given in a CreaComp unit in the example chapter on "equivalence relations and set partitions".

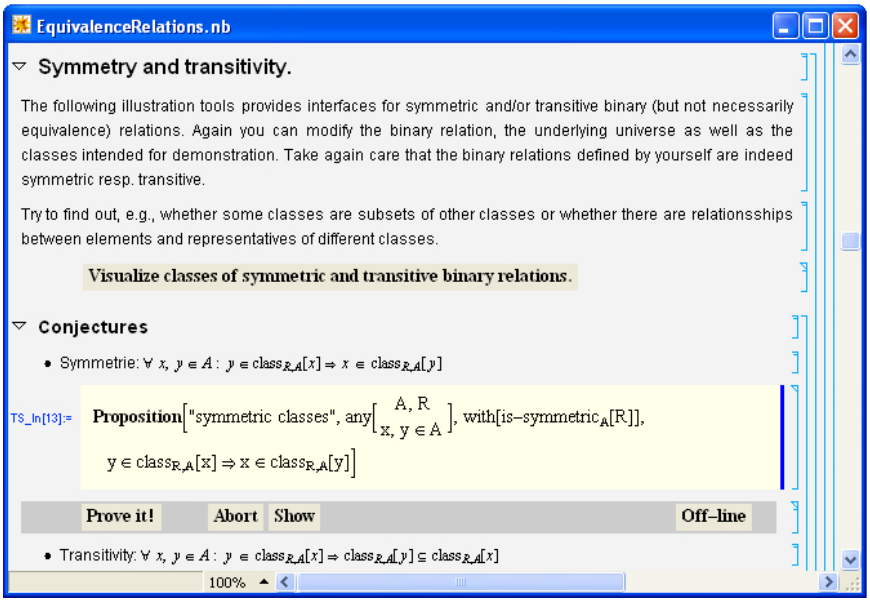


Fig. 1. A typical CreaComp educational unit.

Figure 1 shows a screen-shot of a part of the notebook on equivalence relations containing the most important interactive interface elements. We

see structured text containing inline mathematical formulae hierarchically grouped in nested cells intermixed with formal parts (e.g. definitions, theorems, etc.) written in the *Theorema* language. *Theorema* blocks are written in *Mathematica* input cells and they differ in layout from the surrounding text blocks so that they can easily be recognized as active content. These cells must be evaluated in order for their content to be accessible in the *Mathematica*-kernel later. Note, however, that *Theorema* input is quite different from usual *Mathematica* input! *Theorema* understands most of the common mathematical notation, notably all sorts of quantifiers written in appealing two-dimensional form and, thus, *Theorema* input hardly differs from inlined mathematical formulae in the text.

Although *Mathematica* and *Theorema* provide palettes and keyboard shortcuts for inputting two-dimensional expressions, *Theorema* input is always prepared in advance and the users are usually not required to type *Theorema* formulae. Only occasionally, we leave parts of a formula blank and indicate with a "□"-placeholder that formula parts need to be inserted for the placeholder! CreaComp *interaction buttons* indicate the availability of an interactive experiment and they appear as gray boxes, the text above and on the button roughly explains the associated experiment. The unit shown in Figure 1 contains an interaction button for visualizing classes of symmetric and transitive relations, which will be described in more detail in Section 3.1. Mathematical conjectures are formulated in *Theorema* language immediately followed by a CreaComp *prove panel* containing a prove-button, an abort-button, a show-button, and an off-line-button, see Section 3.2 for details.

3.1. *The Interface to Computer-Supported Experiments*

The interface to interactive experiments is always provided by so-called CreaComp interaction buttons. As an example, we describe the visualization tool behind the interaction button shown in Figure 1. The notions "symmetry" and "transitivity" of a binary relation R on a universe A and the notion of a "class of an element x w.r.t. R and A" have been introduced earlier. At this point we want to investigate properties of classes when R is symmetric and/or transitive. When pressing the interaction button, a new window as shown in Figure 2 appears on the screen. The window essentially contains 4 components:

(i) The top box displays the relation R as a set of pairs in *Theorema* notation and the universe set A. Pressing the button below allows to

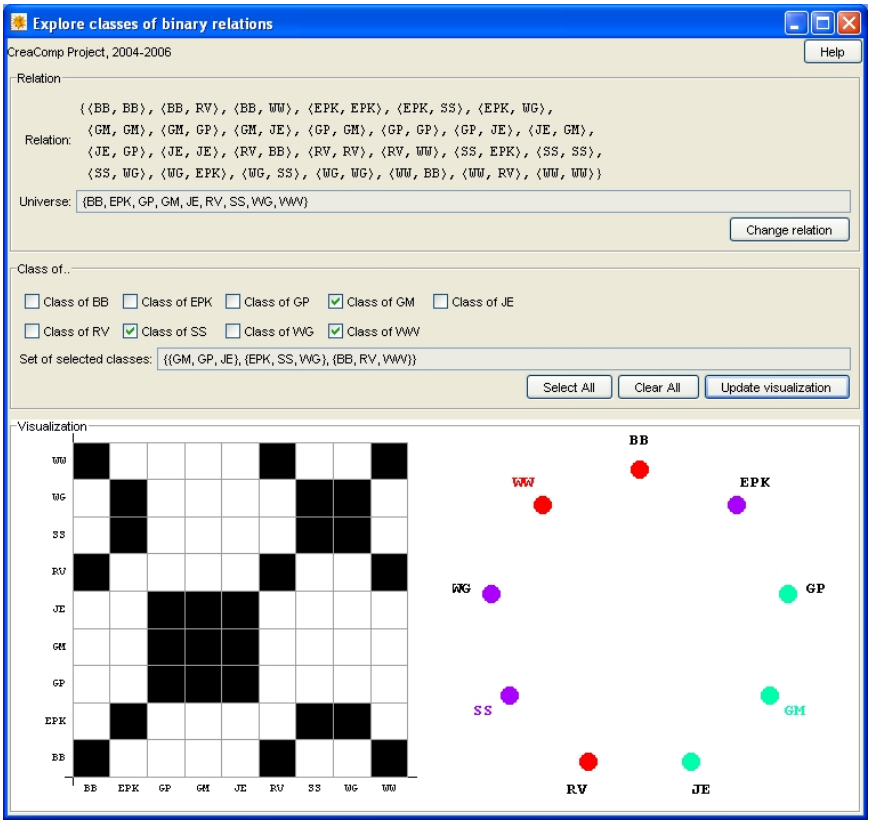


Fig. 2. Interactive visualization of classes.

change the relation R by either explicitly giving a new set of pairs or by having a random symmetric and/or transitive relation automatically generated by the system.

(ii) The middle box allows to select the classes to be visualized and it displays the respective classes as a set of sets in *Theorema* notation.

(iii) The bottom box shows graphical visulaizations of the relation and the selected classes. The raster on the left corresponds to R's adjacency matrix and the arrangment of disks on the right indicates A's elements and each of the selected elements' class by color: each element is assigned a unique color shown by the color of its label and all elements in its class have their disk in the same color. Since intersecting classes are a key-feature to be discovered by this experiment, we display elements

belonging to more than one class by a grey disk. The absence of grey disks in the visualization in Figure 2 indicates disjoint classes in this example.

(iv) The help button in the top-right corner displays individual help for this experiment by explaining which interactions can be made and which phenomena the student is expected to investigate.

Interactive visualizations are implemented using *Mathematica*'s GUIKit, which allows to build Java GUIs containing *Mathematica* data. In the example, the bottom graphics are re-calculated and re-drawn whenever there is user-interaction in one of the top boxes and this is the prototypical interaction pattern for CreaComp experiments: mathematical objects are visualized by *Mathematica* graphics that adapt to user-input given through intuitive interface elements such as check-boxes, radio-buttons, roll-down menues, etc., and special dialog windows that allow *Theorema* input that will be correctly parsed and processed before the respective graphics are re-generated. The on-line help explains the possible interactions and, as a side-effect, it is meant to lead the students' experiments into a "reasonable direction" so that they might conjecture "relevant knowledge". Students have the freedom in exploiting the interactive tools in arbitrary manner, but it is important to provide them also some guidelines in what to try and what to observe, otherwise there is the danger that they get lost if they deviate from the intended track through the unit.

3.2. *The Interface to Automated Proving*

The uniform interface to *Theorema*'s automated provers is always provided by a so-called CreaComp prove panel. As an example, we take again the proof of the proposition shown in Figure 1. The prove-button on the left has a call to a *Theorema* prove method with appropriate parameters associated to its "button-pressed"-event. Every *Theorema* prover needs the knowledge base to be used in the proof as a parameter, thus, before actually starting the proof the user must compose the knowledge base in an interactive dialog. Figure 3 shows such a dialog window: it displays the formula to be proven and it lists all definitions, propositions, theorems, etc. available at this stage. The user simply selects by mouse-click and sends the knowledge base to the prover by pressing the "Prove"-button in the dialog's bottom-right corner. Pressing the "Hint"-button in the bottom-left corner selects just the "appropriate" portion of knowledge for a successful proof. The appropriate knowledge for a certain proof cannot be detected automatically,

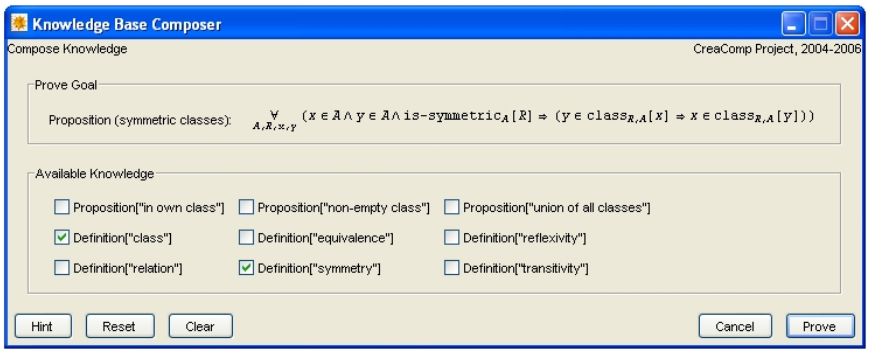


Fig. 3. Interactive knowledge base composition.

the developer of the unit needs to hide this information within the prove panel so that the knowledge base composer can access it from there.

The abort-button in the prove panel aborts a running proof, the show-button shows the proof attempt typically after having aborted the prover. The off-line-button on the right-margin of the prove panel allows to view a pre-generated successful proof. Both pre-generated and live-generated proofs appear in a separate window as shown in Figure 4. The proof comes in human-readable format and explains each proof step in natural language.

3.3. *The Entire Unit*

After having explained the interaction possibilities that are spread all over the material whenever appropriate we can now browse through the unit "Equivalence Relations and Set Partitions", which introduces the students to the correspondence between equivalence relations and set partitions. First of all we introduce the mathematical objects to work with in this theory. The definitions are given in *Theorema* language and start with the definition of relations as a subset of some cartesion product. Since we want to study classes and factor sets of equivalence relations, we restrict our theory to binary relations on some universe set. For other kinds of relations there are links to other CreaComp units, which focus on e.g. general relations or order relations. Furthermore, the first section of this unit defines the main properties interesting for equivalence relations, namely "reflexivity", "symmetry", and "transitivity".

In order to develop some intuition on these properties of relations, we provide a visualization tool to illustrate these properties. The students

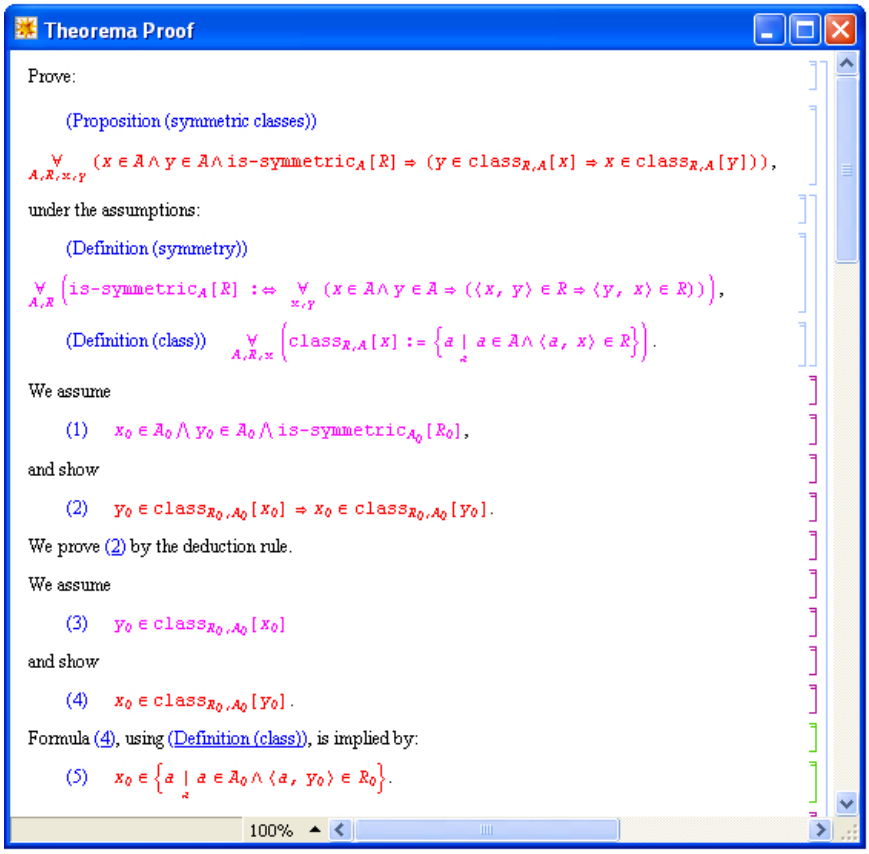


Fig. 4. *Theorema* proof.

can experiment with different relations, some are pre-defined, some are randomly generated or user-defined. This tool is similar in nature to the one shown in Figure 2 only that there is no mentioning of the concept of "classes" yet.

The next step is to introduce the concept of "classes" for binary relations. In *Theorema* language we can do this close to the common mathematical language, so the students can read and understand this definition easily without learning additional syntax:

Definition["class", any$[x, A, R]$,
$\text{class}_{R,A}[x] := \{a \in A | \langle a, x\rangle \in R\}$]

At this point again the students can do some experiments with relations

and classes. While they study some examples they may develop some conjectures on how the properties of a relation effect certain classes. Some possible conjectures are listed in the unit. They are formulated in *Theorema* language, so in a first step the students can see how a conjecture is formalized and in a second step they try to prove it automatically with *Theorema*.

Since our focus is on the mathematical content and not on how to input a proposition in formal syntax and control a theorem prover we provide both the formalization within the notebook and the call for the *Theorema*-prover through the CreaComp prove panel. After evaluating the input cell containing the proposition in *Mathematica* a click on the prove-button initiates a *Theorema* proof of the proposition. The knowledge base needs to be composed interactively as described in Section 3.2 and then an automated proof is attempted by *Theorema*.

Of course, not all conjectures are really true. Some are false in general, but they may be valid in special cases. For example, after inspecting relations in the visualization tool some students might believe the following conjecture to be true:

Proposition["in own class", any$[A, R]$,

$$\forall_{x \in A} \; x \in \mathsf{class}_{R,A}[x] \;]$$

After trying the proof they would realize that the prover fails to prove this proposition. In general, failure of a *Theorema* proof can have various reasons: the provided knowledge is insufficient or the proposition as stated is not provable, maybe not even true! This is just what we want to emphasize in teaching mathematics, and conventional learning material does not provide room for this experience. Finally, failure can also be due to an inappropriate proving method or inappropriate parameters for the method, but we eliminate these sources by packing the call to the prover into the prove-button.

Theorema's possibility to inspect failing proof attempts comes very handy at this point, so students can investigate, which part of the proof led to failure. In the example above, they detect that the prover closes all branches successfully only $\langle x_0, x_0 \rangle \in R_0$ needs to be proven for arbitrary x_0 and R_0. Thorough inspection of the available knowledge at that stage shows that only $x_0 \in A_0$ is known and the student might learn that the propsition as stated *cannot be proven* unless more is known about R_0. Thus, they have to add more side-conditions to the proposition. In *Theorema* this can be done using the with[...] expression. At this place, the unit now contains

a proposition with the "□"-placeholder included, where the students can insert the additional side-condition:

Proposition["in own class", any[A, R], with[□],

$\underset{x\in A}{\forall}\ x \in \mathsf{class}_{R,A}[x]$]

Now a user can try different conditions in order to succeed in proving the proposition. Of course, in order to succeed with the above proof, R_0 must have the property that $\langle x, x\rangle \in R_0$ is true for all $x \in A_0$, i.e. R_0 must be reflexive on A_0. Hence, we must put the side-condition "is-reflexive$_A[R]$" and can then successfully prove the adapted proposition. This proof is not particularly difficult, still the formal rigor in which it is carried out is instructive for students.

The remaining content of this unit explains some properties of sets of sets. In particular, the students can learn about the factor set of a relation

Definition["factor set", any[A, R],

$\mathsf{factor\text{-}set}_A[R] := \{\mathsf{class}_{R,A}[x] \underset{x\in A}{|}\ \}$]

and which properties a set of sets make it a "partition". Moreover, the concept of an "induced relation" of a set of sets is introduced, namely

Definition["induced relation", any[A, S],

$\mathsf{induced\text{-}relation}_A[S] := \{\langle x, y\rangle \underset{x,y\in A}{|}\ \underset{M\in S}{\exists}\ x \in M \wedge y \in M\}$]

In each step the procedure is similar:

- *introduce* new objects or properties,
- *visualize* the properties in order to gain insights,
- *guess and propose* conjectures,
- *formalize* the conjectures precisely, and
- *prove* them automatically with *Theorema*.

The key observations are then:

- if R is an equivalence relation on A then factor-set$_A[R]$ is a partition of A and induced-relation$_A$[factor-set$_A[R]] = R$.
- if P is a partition of A then induced-relation$_A[P]$ is an equivalence relation and factor-set$_A$[induced-relation$_A[P]] = P$.

All these theorems including also all auxiliary lemmata necessary for compact proofs of the main statements are proved fully automatically within this learning unit. In order to demonstrate the readability of *Theorema* proofs, we show one proof in all details *as it is generated in the* Theorema

system, compare also to Figure 4.

Lemma[“induced relation is transitive”, any[A, P], with[is-partition$_A$[P]], is-transitive$_A$[induced-relation$_A$[P]]]

In the knowledge base for this proof, we have the definitions of is-transitive$_A$ and induced-relation$_A$ and an auxiliary proposition proved earlier, namely

Proposition[“intersecting are equal”, any[A, P], with[is-partition$_A$[P]], $\underset{X,Y\in P}{\forall} X \cap Y \neq \emptyset \Rightarrow X = Y$]

Proof. We assume

(1) is-partition$_{A_0}$[P_0]

and show

(2) is-transitive$_{A_0}$[induced-relation$_{A_0}$[P_0]].

Formula (2), using (Definition (transitivity)), is implied by:

$$(3)\quad \underset{x,y,z\in A_0}{\forall} \langle x,y\rangle \in \text{induced-relation}_{A_0}[P_0] \wedge \langle y,z\rangle \in \text{induced-relation}_{A_0}[P_0] \Rightarrow \langle x,z\rangle \in \text{induced-relation}_{A_0}[P_0].$$

We assume

$$(4)\quad \begin{array}{l} x_0 \in A_0 \wedge y_0 \in A_0 \wedge z_0 \in A_0 \wedge \\ \langle x_0,y_0\rangle \in \text{induced-relation}_{A_0}[P_0] \wedge \langle y_0,z_0\rangle \in \text{induced-relation}_{A_0}[P_0] \end{array}$$

and show

(5) $\langle x_0, z_0\rangle \in$ induced-relation$_{A_0}$[P_0].

Formula (5), using (Definition (induced relation)), is implied by:

$$(8)\quad \langle x_0,z_0\rangle \in \{\langle x,y\rangle \underset{x,y\in A_0}{|} \underset{M\in P_0}{\exists} x \in M \wedge y \in M\}.$$

In order to prove (8) we have to show

$$(9)\quad \underset{x,y\in A_0}{\exists} (\underset{M}{\exists} M \in P_0 \wedge x \in M \wedge y \in M) \wedge \langle x_0,z_0\rangle = \langle x,y\rangle.$$

Since $x := x_0$ and $y := z_0$ solves the equational part of (9) it suffices to show

$$(10)\quad x_0 \in A_0 \wedge z_0 \in A_0 \wedge \underset{M}{\exists} M \in P_0 \wedge x_0 \in M \wedge z_0 \in M.$$

Formula (10.1) is true because it is identical to (4.1)
Formula (10.2) is true because it is identical to (4.3)

Formula (4.4), by (Definition (induced relation)), implies:

(12) $\langle x_0, y_0\rangle \in \{\langle x, y\rangle \underset{x,y\in A_0}{|} \underset{M\in P_0}{\exists} x \in M \wedge y \in M\}$.

From (12) we know by definition of $\{T_x \underset{x}{|} P\}$ that we can choose an appropriate value such that

(13) $\underset{M}{\exists} M \in P_0 \wedge x1_0 \in M \wedge x2_0 \in M$,

(14) $\langle x_0, y_0\rangle = \langle x1_0, x2_0\rangle$.

Formula (14) simplifies to

(16) $x_0 = x1_0 \wedge y_0 = x2_0$.

By (13) we can take appropriate values such that:

(17) $M_0 \in P_0 \wedge x1_0 \in M_0 \wedge x2_0 \in M_0$.

Now, let $M := M_0$. Thus, for proving (10.3) it is sufficient to prove:

(21) $M_0 \in P_0 \wedge x_0 \in M_0 \wedge z_0 \in M_0$.

Formula (21.1) is true because it is identical to (17.1).
Formula (21.2), using (16.1), is implied by:

(22) $x1_0 \in M_0$.

Formula (22) is true because it is identical to (17.2).
Proof of (21.3) $z_0 \in M_0$: Formula (4.5), by (16.2), implies:

$\langle x2_0, z_0\rangle \in \text{induced-relation}_{A_0}[P_0]$

which, by (Definition (induced relation)), implies:

(23) $\langle x2_0, z_0\rangle \in \{\langle x, y\rangle \underset{x,y\in A_0}{|} \underset{M\in P_0}{\exists} x \in M \wedge y \in M\}$.

From (23) we know by definition of $\{T_x \underset{x}{|} P\}$ that we can choose an appropriate value such that

(24) $\underset{M}{\exists} M \in P_0 \wedge x3_0 \in M \wedge x4_0 \in M$,

(25) $\langle x2_0, z_0\rangle = \langle x3_0, x4_0\rangle$.

Formula (25) simplifies to

(27) $x2_0 = x3_0 \wedge z_0 = x4_0$.

By (24) we can take appropriate values such that:

(28) $M_1 \in P_0 \wedge x3_0 \in M_1 \wedge x4_0 \in M_1$.

Formula (21.3), using (27.2), is implied by:

(32) $x4_0 \in M_0$.

Formula (17.3), by (27.1), implies:

(33) $x3_0 \in M_0$.

From (28.2) together with (33) we know

(35) $x3_0 \in M_1 \cap M_0$.

From (35) we can infer

(36) $M_1 \cap M_0 \neq \emptyset$.

Formula (36), by (Proposition (intersecting are equal)), implies:

(37) $\underset{A,P}{\forall}$ is-partition$_A[P] \wedge M_0 \in P \wedge M_1 \in P \Rightarrow M_1 = M_0$.

Formula (1), by (37), implies:

(71) $M_0 \in P_0 \Rightarrow (M_1 \in P_0 \Rightarrow M_1 = M_0)$.

From (17.1) and (71) we obtain by modus ponens

(72) $M_1 \in P_0 \Rightarrow M_1 = M_0$.

From (28.1) and (72) we obtain by modus ponens

(73) $M_1 = M_0$.

Formula (32) is true because of (28.3) and (73). □

The proof as shown above will appear in a separate window and features interactive elements that cannot be rendered in the above "static reproduction": All formula references are active button elements, which will display the referenced formula in a separate window, proof goals and assumptions can easily be distinguished by color, see also Figure 4, and the structure of the proof tree is reflected by the nested cell structure of the proof notebook, so that entire proof branches can be collapsed by a single mouse-click.

4. Conclusion and Future Work

CreaComp is work in progress. Therefore we do not have results on evaluation of the units in classroom yet. Further work will go into computer-

supported assessment, which has already been implemented in the frame of MeetMath, see [8]. Assessment is heavily based on randomly generated test exercises based on example patterns, where the power of a symbolic computation system in the background is essential for checking correctness of user solutions. Also, the use of *Theorema* provers for checking user answers can be investigated.

References

1. B. Buchberger, *ACM SIGSAM Bulletin* **24**, 10(January 1990).
2. R. Andrews, C. Brown, F. Pfenning, M. Bishop, S. Issar and H.Xi, *Journal of Automated Reasoning* **32**, 75 (2004).
3. R. Sommer and G. Nuckols, *Journal of Automated Reasoning* **32**, 227 (2004).
4. B. Buchberger, A. Craciun, T. Jebelean, L. Kovacs, T. Kutsia, K. Nakagawa, F. Piroi, N. Popov, J. Robu, M. Rosenkranz and W. Windsteiger, *Journal of Applied Logic* (2005), To appear.
5. B. Buchberger, C. Dupre, T. Jebelean, F. Kriftner, K. Nakagawa, D. Vasaru and W. Windsteiger, The Theorema Project: A Progress Report, in *Symbolic Computation and Automated Reasoning (Proceedings of CALCULEMUS 2000, Symposium on the Integration of Symbolic Computation and Mechanized Reasoning)*, eds. M. Kerber and M. Kohlhase (Copyright: A.K. Peters, Natick, Massachusetts, 6-7 August 2000).
6. B. Buchberger, T. Jebelean, F. Kriftner, M. Marin, E. Tomuta and D. Vasaru, A Survey of the Theorema project., in *Proceedings of ISSAC'97 (International Symposium on Symbolic and Algebraic Computation, Maui, Hawaii, July 21-23, 1997), ACM Press 1997.*, ed. W. Kuechlin1997.
7. S. Wolfram, *The Mathematica Book*, 5th edn. (Wolfram Media, Inc., 2003).
8. S. Saminger, MeetMATH — visualizations and animations in a didactic framework, in *Technology in Mathematics Teaching (Special groups and working groups). Proceedings of ICTMT 5, Klagenfurt (Austria)*, eds. M. Borovcnik and H. Kautschitsch, Schriftenreihe Didaktik der Mathematik, Vol. 26 (öbv & hpt Verlagsgesellschaft, Wien, 2002).
9. B. Buchberger, Symbolic Computation: Computer Algebra and Logic, in *Frontiers of Combining Systems, Proceedings of FROCOS 1996 (1st International Workshop on Frontiers of Combining Systems), March 26-28, 1996, Munich*, eds. F. Bader and K. Schulz, Applied Logic Series, Vol. 3 (Kluwer Academic Publisher, Dordrecht - Boston - London, The Netherlands, 1996).
10. F. Piroi and T. Kutsia, The Theorema Environment for Interactive Proof Development, in *Logic for Programming, Artificial Intelligence, and Reasoning. Proceedings of the 12th International Conference, LPAR'05*, eds. G. Sutcliffe and A. Voronkov, Lecture Notes in Artificial Intelligence, Vol. 3835 (Springer Verlag, 2005).
11. G. Hanna, *Educational Studies in Mathematics* **44**, 5 (2000).