Kapitel C

Kombinatorische Spieltheorie und der Satz von Sprague–Grundy

Teach me how to lose a winning match.
Shakespeare, *Romeo and Juliet* 3.2.12

You got to lose to know how to win.
Aerosmith, *Dream On* (1973)

Inhalt dieses Kapitels C

1. Neutrale kombinatorische Spiele
   - Einzeiliges Nim und dynamische Programmierung
   - Neutrale Spiele und Rückwärtsinduktion
   - Das Spiel Nim und Boutons effiziente Lösung
   - Summen von Spielen und Sprague–Grundy–Satz
2. Anwendungsbeispiele und weitere Aufgaben
   - Lasker–Nim, Grundys Spiel, Fliesentetris
   - Schleifenspiele: Poker-Nim und Northcotts Spiel
   - Das Spiel Chomp! nach David Gale
   - Wie viel Rationalität benötigen wir?
3. Mengen und Logik als Spiele
   - Schach ist determiniert.
   - Mengen als Spiele
   - Quantoren-Spiele

Motivation und Überblick

Dynamische Spiele nutzen Positionen / Zustände und Züge / Aktionen:

- Es gibt verschiedene (meist nur endlich viele) Positionen, oft auch eine bestimmte Startposition zu Beginn des Spiels.
- Die Regeln des Spiels legen für jede Position eindeutig fest, welche Züge möglich sind; diese führen zu neuen Positionen.
- Die Spieler entscheiden sich frei unter den ihnen möglichen Zügen und erhalten Auszahlungen, während des Spiels oder am Ende.

Ausgehend von dieser allgemeinen Beschreibung haben wir im vorigen Kapitel Graphen als tragende Grundstruktur erklärt. Zusammen mit weiteren Daten zur Auszahlung konnten wir die Gewinnfunktion über die Bellman–Gleichung definieren und bestimmen. Dies ist zunächst nur für einen Spieler formuliert, doch erlaubt schon eine erstaunliche Vielfalt an Optimierungsfragen. Wir wollen dies auf mehrere Spieler ausdehnen.

Motivation und Überblick

Viele Spiele erfreuen sich zudem **vollständiger Information**:

- Jeder Spieler kennt das gesamte Spiel: alle möglichen Zustände, Aktionen und Auszahlungen. Diese sind *common knowledge*.
- In jedem aktiven Zustand zieht genau einer der Spieler.
  Es gibt keine Zufallszüge und keine gleichzeitigen Züge.
- Der ziehende Spieler kennt den Startzustand und alle bisherigen Züge und somit den aktuellen Zustand des Spiels. Insbesondere gibt es keine verdeckten Züge oder Vergesslichkeit der Spieler; diese raffinierten Erweiterungen diskutieren wir erst später.

Wir wollen vorerst diese einschränkenden Annahmen machen, denn sie erleichtern sowohl die Beschreibung des Spiels als auch seine Analyse. Später werden auch diese Beschränkungen nach und nach aufgehoben. Es scheint ratsam, zunächst mit einfachen Bei-Spielen zu beginnen.

Motivation und Überblick

Für **kombinatorische Spiele** vereinfachen wir noch weiter:

1. Es gibt genau zwei Spieler (Links / Rechts, Alice / Bob, etc).
2. Sie spielen um Gewinn 1 oder Verlust 0, keine anderen Werte.
3. Jede:r der beiden kann beginnen. Gezogen wird abwechselnd.
4. Das Spiel endet, wenn der Ziehende keine Zugmöglichkeit hat.
5. Normale Konvention: Wer nicht mehr ziehen kann, verliert.
6. Die Spielregeln garantieren, dass jeder Spielverlauf endet.
7. Beide Spieler haben vollständige Information.
8. Es gibt keine Zufallszüge.

Wir betrachten insbesondere **neutrale Spiele** [*impartial games*]:

9. In jeder Position x haben beide Spieler dieselben Zugoptionen.

Das Teilspiel ab x ist daher vollkommen symmetrisch in beiden Spielern.
Allein die Unterscheidung in ziehend und wartend bricht die Symmetrie.

⚠ Diese Ideen sind noch allzu vage und müssen präzisiert werden!

Motivation und Überblick

Aufgabe: Prüfen Sie diese Eigenschaften für Ihre Lieblingsspiele!

Spiel / Eigenschaft	1	2	3	4	5	6	7	8	9
Nim und Varianten	✓	✓	✓	✓	✓	✓	✓	✓	✓
Tic-Tac-Toe	✓	✘	✓	✓	✘	✓	✓	✓	✘
Schach	✓	✘	✓	✘	✘	✓	✓	✓	✘
Backgammon	✓	✘	✓	✘	✘	✓	✓	✘	✘
Schiffe versenken	✓	✓	✓	✘	✘	✓	✘	✓	✘
Poker	?✓	✘	✓	✘	✘	✓	✘	✘	✘
Monopoly	?✓	?✓	✓	✘	✘	✘	✘	✘	✘
Schere-Stein-Papier	✓	✘	✘	✘	✘	✓	✘	✓	✓

Auch Fußball, Handball, Basketball, etc. sind Spiele. Vereinfacht sind die Mannschaften hier die beiden Spieler. Man spricht zwar von Positionen und Spielzügen, doch ist es schwierig, sie mathematisch zu präzisieren. Dennoch lohnt es; diese Bemühungen sind zuletzt enorm erfogreich.

Motivation und Überblick

Die wichtigste Frage zu einem kombinatorischen Spiel ist:
Welcher der beiden Spieler hat eine **Gewinnstrategie**?

Das heißt ausführlich: Welcher Spieler kann gewinnen,
wenn er nur optimal spielt, egal wie der Gegner vorgeht?

Dabei darf er sich nicht auf Fehler des Gegners verlassen,
sondern muss auf alle Gegenstrategien vorbereitet sein.

Eine Position heißt **Gewinnposition**, wenn der ziehende Spieler gewinnt (bei optimalem Spiel). Andernfalls heißt sie **Verlustposition**.

Ein Zug heißt **Gewinnzug**, wenn der ziehende Spieler durch ihn gewinnt (bei weiterhin optimalem Spiel). Andernfalls heißt er **Verlustzug**.

☺ Wir werden in diesem Kapitel Gewinn- und Verlustpositionen erst präzise definieren und anschließend möglichst effizient berechnen.

☺ Damit finden wir auch eine Antwort auf die ganz praktische Frage, wie wir einen Gewinnzug finden, falls es überhaupt einen gibt.

☺ Das Erfolgskriterium für unsere Methoden wird sein:
Können wir damit wirklich besser spielen?

Motivation und Überblick

Die **kombinatorische Spieltheorie** [*combinatorial game theory*, CGT] ist ein umfangreiches und aktives Gebiet der Mathematik und Informatik. Ihre Geschichte beginnt 1901 mit der effizienten Lösung des Spiels Nim:

Charles L. Bouton: *Nim, a game with a complete mathematical theory.* Annals of Mathematics 3 (1901) 35–39

Nim ist der Prototyp für alle Spiele, die in eine Summe unabhängiger Teilspiele zerfallen. Der Satz C1O von Sprague–Grundy überführt dazu jedes neutrale kombinatorische Spiel in ein äquivalentes Nim-Spiel! Damit lassen sich viele Spiele analysieren und effizient lösen.

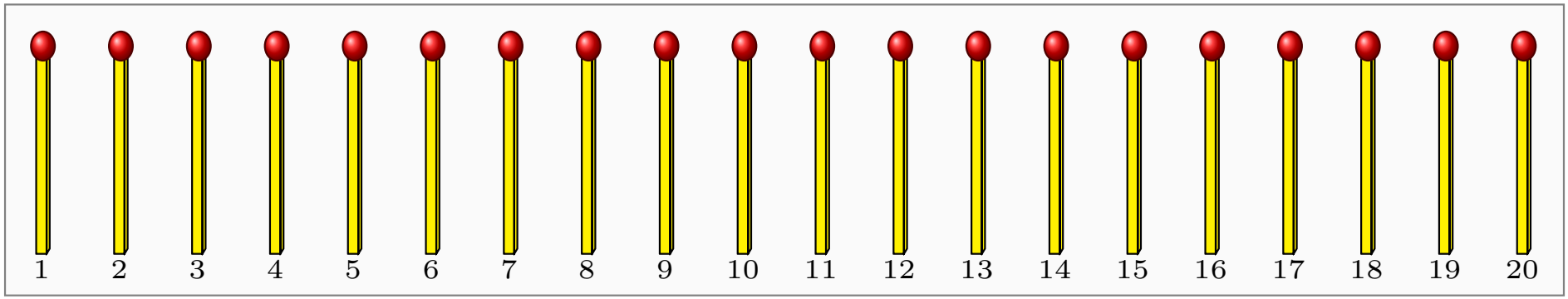
Roland P. Sprague: *Über mathematische Kampfspiele.* Tôhoku Mathematical Journal 41 (1935) 438–444

Roland P. Sprague: *Über zwei Abarten von Nim.* Tôhoku Mathematical Journal 43 (1937) 451–454

Patrick M. Grundy: *Mathematics and Games.* Eureka 2 (1939) 6–8

Einzeiliges Nim und dynamische Programmierung

Auf dem Tisch liegen anfangs $x \in \mathbb{N}$ Streichhölzer / Münzen / Steine.
Die Spieler ziehen abwechselnd, jeder entfernt ein oder zwei Hölzer.
Dies ist ein einzeiliges Subtraktionsspiel mit Zugoptionen $S = \{1, 2\}$.



Bevor Sie weiterlesen sollten Sie dieses Spiel einige Male durchspielen, am besten zu zweit. In Vorlesung und Übung spielen wir es an der Tafel.

Beobachten Sie dabei ihren Lernprozess vom *Whaaa?* bis zum *Aha!* Anfangs werden Sie vermutlich wenig Struktur erkennen. Mit Erfahrung ahnen Sie gewisse Regelmäßigkeiten. Diese können Sie in folgender Aufgabe ausarbeiten und schließlich die allgemeine Regel formulieren.

Am Ende steht ein mathematischer Satz als Extrakt Ihrer Erfahrungen. Diesen können Sie induktiv beweisen und zukünftig getrost anwenden!

Einzeiliges Nim und dynamische Programmierung

Aufgabe: (0) Formulieren Sie dieses Spiel explizit als einen Graphen. Berechnen Sie alle Gewinnzüge (1) naiv-rekursiv (exponentiell in x), (2) memoisiert-rekursiv (linear in x), (3) effizient (linear in $\log x$).

- **Misèrespiel**, $\mu : X \to \{0, 1\}$: Wer nicht mehr ziehen kann, gewinnt.
- **Normalspiel**, $\nu : X \to \{0, 1\}$: Wer nicht mehr ziehen kann, verliert.
- Noch informativer als ν ist die **Sprague–Grundy–Funktion**

$$\gamma : X \to \mathbb{N} : \gamma(x) = \operatorname{mex}\{\, \gamma(y) \mid x \to y \,\},$$
$$\operatorname{mex} : \{\, S \subsetneq \mathbb{N} \,\} \to \mathbb{N} : S \mapsto \min(\mathbb{N} \setminus S).$$

In Worten: Zu jeder Menge $S \subsetneq \mathbb{N}$ definieren wir $\operatorname{mex} S := \min(\mathbb{N} \setminus S)$ als das Minimum des Komplements [engl. *minimal excludant*, kurz *mex*]. Diese Funktion wird sich im Folgenden als überaus nützlich erweisen. Zunächst wollen wir ihre Definition verstehen und konkret anwenden.

Wir werden γ genau so auf jedem Graphen X definieren; dieser muss lokal-endlich sein (das heißt, jede Ecke hat nur endlich viele Nachfolger) und zudem artinsch (das heißt, es gibt keine unendlich langen Wege). Nach den motivierenden Beispielen führen wir diese Definitionen aus.

Einzeiliges Nim und dynamische Programmierung

Lösung: (0) Jede Ecke (Position) ist eine natürliche Zahl $x \in X := \mathbb{N}$. Jede Kante (Zug) $a = (x, s) : x \to y$ erfüllt $y = x - s$ mit $s \in S = \{1, 2\}$.

(1a) Misèrespiel, Gewinnfunktion $\mu : X \to \{0, 1\} : x \mapsto \mu(x)$:

```
1 def mu(x):
2     if x == 0: return 1   # Wer nicht mehr ziehen kann, gewinnt.
3     return 1 - min( mu(y) for y in range(max(0,x-2), x) )
```

(1b) Normalspiel, Gewinnfunktion $\nu : X \to \{0, 1\} : x \mapsto \nu(x)$:

```
1 def nu(x):
2     if x == 0: return 0   # Wer nicht mehr ziehen kann, verliert.
3     return 1 - min( nu(y) for y in range(max(0,x-2), x) )
```

Was ist schlecht an dieser Implementierung? Exponentieller Aufwand!

⚠ Für die Anzahl $f(x)$ der Funktionsaufrufe gilt $f(0) = 0$ und $f(1) = 1$ sowie für alle $x \in \mathbb{N}_{\geq 2}$ rekursiv $f(x) = f(x-1) + f(x-2)$. Dies ist die Fibonacci–Folge! Sie wächst exponentiell, explizit gilt die Binet–Formel:

$$f(x) = \frac{1}{\sqrt{5}} \left[\left(\frac{1+\sqrt{5}}{2} \right)^x - \left(\frac{1-\sqrt{5}}{2} \right)^x \right] \gtrapprox 1.618^x$$

Einzeiliges Nim und dynamische Programmierung

⚠ Naive Rekursion ist zwar korrekt, aber allzu verschwenderisch!
(1) Bei dieser naiven Methode wächst der Aufwand exponentiell mit x!

Algo C1A: Gewinnzüge $@M : x \mapsto M(x)$ im Misèrespiele

Eingabe: Eine natürliche Zahl $x \in \mathbb{N}$ und die Zugoptionen $S \subseteq \mathbb{N}_{\geq 1}$
Ausgabe: Alle Gewinnzüge $M(x) \subseteq S$ in der gegebenen Position x

---

1: if $x = 0$ then return $\{*\}$
2: $M \leftarrow \{\}$
3: for $s \in S$ do if $s \leq x$ and $M(x - s) = \{\}$ then $M \leftarrow M \cup \{s\}$
4: return M

☹ Im Jobinterview machen Sie damit allein keinen guten Eindruck, egal ob in Python (vorige Folie) oder als Pseudo-Code (diese Folie), denn die Werte $M(y)$ werden unnötig immer wieder neu berechnet.
„Never hire a developer who computes the factorial using recursion.“

Aufgabe: (2) Wie kann man die Rekursion effizienter implementieren?

Einzeiliges Nim und dynamische Programmierung

Wir merken uns, was bereits berechnet wurde, und recyclen!
Diese genial-einfache Idee heißt **Memoisation**, abgeleitet von lat. **Memorandum**, kurz **Memo**, *das zu Erinnernde*.

Algo C1B: Gewinnzüge $M : x \mapsto M(x)$ im Misèrespiele

Global: Tabelle memo, anfangs initialisiert durch $\mathrm{memo}[0] \leftarrow \{*\}$
Eingabe: Eine natürliche Zahl $x \in \mathbb{N}$ und die Zugoptionen $S \subseteq \mathbb{N}_{\geq 1}$
Ausgabe: Alle Gewinnzüge $M(x) \subseteq S$ in der gegebenen Position x

---

1: if x not in memo then $\mathrm{memo}[x] \leftarrow @M(x)$
2: return $\mathrm{memo}[x]$

☹ Naive Rekursion ist zwar korrekt, aber allzu verschwenderisch!

☺ Geschickte Buchführung reduziert erheblich den Rechenaufwand!
In diesem Beispiel ist die Rechenzeit für M nur noch linear in x.

☺ Hier wird $@M$ in eine memoisierende Funktion M eingebettet:
Die innere Funktion $@M$ übernimmt dieselbe Rechnung wie zuvor.
Die äußere Funktion M kümmert sich um die Buchhaltung.

Einzeiliges Nim und dynamische Programmierung

Auch für das Normalspiel und die Sprague–Grundy–Funktion lohnt sich, die rekursiven Algorithmen explizit auszuschreiben und zu memoisieren. Wenn Sie möchten, sollten Sie unbedingt Beispiele programmieren! Nur so spüren Sie den Unterschied zwischen effektiv und effizient.

In C/C++ muss Memoisation explizit programmiert werden, wie skizziert. Dazu wird die rekursive Funktion in ein Funktionsobjekt eingebettet, das seine eigene Erinnerung memo als Tabelle [*map*] verwaltet.

Python bietet hierfür einen vorgefertigten Decorator `@memoize`. Viele moderne Programmiersprachen unterstützen Memoisation, manche erledigen dies gar im Hintergrund ganz automatisch.

Beim Normalspiel und der Sprague–Grundy–Funktion bemerken wir: Anders als im Misèrespiel ist der Basisfall $x = 0$ gleich eingearbeitet. Er folgt denselben Regeln wie $x \geq 1$ und verzweigt keinen Sonderfall.

Aus diesem und weiteren Gründen ist das Normalspiel „natürlicher“. Das Hauptargument ist schließlich der Sprague–Grundy–Satz C1o, der für das Normalspiel eine elegante und starke Theorie entwickelt.

Einzeiliges Nim und dynamische Programmierung

Algo C1c: Gewinnzüge $@N : x \mapsto N(x)$ im Normalspiel

Eingabe: Eine natürliche Zahl $x \in \mathbb{N}$ und die Zugoptionen $S \subseteq \mathbb{N}_{\geq 1}$
Ausgabe: Alle Gewinnzüge $N(x) \subseteq S$ in der gegebenen Position x

---

1: $N \leftarrow \{\}$
2: for $s \in S$ do if $s \leq x$ and $N(x-s) = \{\}$ then $N \leftarrow N \cup \{s\}$
3: return N

- Beim Normalspiel ist der Basisfall $x = 0$ gleich eingearbeitet.
- Naive Rekursion ist zwar korrekt, aber allzu verschwenderisch!
- Memoisation reduziert von exponentiellem zu linearem Aufwand:

Algo C1d: Gewinnzüge $N : x \mapsto N(x)$ im Normalspiel

Global: Tabelle memo, anfangs initialisiert als leer
Eingabe: Eine natürliche Zahl $x \in \mathbb{N}$ und die Zugoptionen $S \subseteq \mathbb{N}_{\geq 1}$
Ausgabe: Alle Gewinnzüge $N(x) \subseteq S$ in der gegebenen Position x

---

1: if x not in memo then $\mathrm{memo}[x] \leftarrow @N(x)$
2: return $\mathrm{memo}[x]$

Einzeiliges Nim und dynamische Programmierung

Algo C1E: Grundy–Wert $@\gamma : x \mapsto \gamma(x)$

Eingabe: Eine natürliche Zahl $x \in \mathbb{N}$ und die Zugoptionen $S \subseteq \mathbb{N}_{\geq 1}$
Ausgabe: Der Grundy–Wert $\gamma(x) = \operatorname{mex}\{\, \gamma(y) \mid x \to y \,\}$ der Position x

---

1: $A \leftarrow \{\}$
2: for $s \in S$ do if $s \leq x$ then $A \leftarrow A \cup \{\, \gamma(x - s) \,\}$
3: return $\operatorname{mex} A$

- Beim Grundy–Wert ist der Basisfall $x = 0$ gleich eingearbeitet.
- Naive Rekursion ist zwar korrekt, aber allzu verschwenderisch!
- Memoisation reduziert von exponentiellem zu linearem Aufwand!

Algo C1F: Grundy–Wert $\gamma : x \mapsto \gamma(x)$

Global: Tabelle memo, anfangs initialisiert als leer
Eingabe: Eine natürliche Zahl $x \in \mathbb{N}$ und die Zugoptionen $S \subseteq \mathbb{N}_{\geq 1}$
Ausgabe: Der Grundy–Wert $\gamma(x) = \operatorname{mex}\{\, \gamma(y) \mid x \to y \,\}$ der Position x

---

1: if x not in memo then $\mathrm{memo}[x] \leftarrow @\gamma(x)$
2: return $\mathrm{memo}[x]$

Einzeiliges Nim und dynamische Programmierung

Lösung: (2) Wir berechnen Gewinn und Verlust für das Misèrespiel μ und das Normalspiel ν sowie die Sprague–Grundy–Funktion γ.

$x=$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
$\mu=$	1	0	1	1	0	1	1	0	1	1	0	1	1	0	1	1	0	1	1	0	1
$\nu=$	0	1	1	0	1	1	0	1	1	0	1	1	0	1	1	0	1	1	0	1	1
$\gamma=$	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2

Bottom-up sortiert nach Größe: Jedes Teilproblem nutzt nur kleinere.
Ebenso gut gelingt der rekursive Ansatz top-down mit Memoisation.

(3) Wie lautet die allgemeine Regel? Wir krönen unsere Bemühungen:

Satz C1G: einzeiliges Nim mit Zugoptionen $S = \{1, 2, \ldots, n-1\}$

Misèrespiel: Genau dann ist x eine Verlustposition, wenn $x \bmod n = 1$.
Normalspiel: Genau dann ist x eine Verlustposition, wenn $x \bmod n = 0$.
Die Sprague–Grundy–Funktion des Spiels ist $\gamma : \mathbb{N} \to \mathbb{N} : x \mapsto x \bmod n$.

Einzeiliges Nim und dynamische Programmierung

Die **Dynamische Programmierung** [*Dynamic Programming*, DP] ist ein Werkzeugkasten zur Optimierung rekursiver Berechnungen. Siehe Cormen, Stein, Leiserson, Rivest: *Introduction to Algorithms*, §15.

Top-down mit Memoisation: Wir formulieren unsere Funktion rekursiv und speichern das Ergebnis jedes gelösten Teilproblems, etwa in einem assoziativen Array [*map*, *dictionary*] oder einer Tabelle [*hash table*].

Bottom-up topologisch sortiert: Jedes Teilproblem nutzt jeweils nur kleinere, bereits gelöste. Beide Formulierungen leisten meist dasselbe; Ausnahmen entstehen, wenn top-down große Lücken lassen kann.

Idealerweise können Algorithmen mit exponentiellen Aufwand so auf polynomiellen Aufwand reduziert werden, wie in unserem Beispiel: Unsere Lösung ist zunächst exponentiell in x, dann polynomiell in x, hier sogar linear in x, dank Satz C1G schließlich sogar nur logarithmisch in x.

Das natürliche **Komplexitätsmaß** ist hier die Bitlänge $\mathrm{len}(x) \sim \log_2(x)$. Unsere Lösung verbessert sich von doppelt exponentiell zu exponentiell, schließlich zu polynomiell, sogar linear in der Länge $\mathrm{len}(x) \sim \log_2(x)$.

Einzeiliges Nim und dynamische Programmierung

Aufgabe: Untersuchen Sie ebenso einzeiliges Nim mit den Zugoptionen (3) $S = \{1, 2, 4\}$ und (4) $S = \{1, 3, 4\}$ und (5) $S = \{1, 3, 6\}$. **Lösung:**

$x=$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
$\mu=$	1	0	1	1	0	1	1	0	1	1	0	1	1	0	1	1	0	1	1	0	1
$\nu=$	0	1	1	0	1	1	0	1	1	0	1	1	0	1	1	0	1	1	0	1	1
$\gamma=$	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2

$x=$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
$\mu=$	1	0	1	0	1	1	1	1	0	1	0	1	1	1	1	0	1	0	1	1	1
$\nu=$	0	1	0	1	1	1	1	0	1	0	1	1	1	1	0	1	0	1	1	1	1
$\gamma=$	0	1	0	1	2	3	2	0	1	0	1	2	3	2	0	1	0	1	2	3	2

$x=$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
$\mu=$	1	0	1	0	1	0	1	1	1	1	0	1	0	1	0	1	1	1	1	0	1
$\nu=$	0	1	0	1	0	1	1	1	1	0	1	0	1	0	1	1	1	1	0	1	0
$\gamma=$	0	1	0	1	0	1	2	3	2	0	1	0	1	0	1	2	3	2	0	1	0

Einzeiliges Nim und dynamische Programmierung

Richard Bellman (1920–1984) war ein US-amerikanischer Mathematiker. Er entwickelte 1953 die Kernidee der **Dynamischen Programmierung**. In „Dynamische Programmierung“ sowie in „Lineare Programmierung“ bedeutet das Wort „Programmierung“ so viel wie „Optimierung“.

Ökonomen bezeichnen die Dynamische Programmierung schlicht als **Rekursionsmethode**. Sie tritt bei zahlreichen Optimierungsproblemen natürlich auf und wird gerne und erfolgreich angewendet. Sie ist daher ein beliebtes Universalwerkzeug der Wirtschaftswissenschaften.

Idealerweise entspringt die Rekursion unmittelbar der ökonomischen / spieltheoretischen Fragestellung. In vereinfachter Form findet sich die Rekursionsmethode daher bereits Ernst Zermelo 1913 sowie später bei John von Neumann und Oskar Morgenstern 1944.

Die Dynamische Programmierung ist in der Informatik eine universelle Strategie, um Probleme rekursiv zu lösen. Zur gewünschten Funktion muss eine geeignete Rekursionsstruktur aber erst gefunden werden. Diese Kunst erfordert sowohl Erfahrung als auch Kreativität.

Neutrale Spiele und Rückwärtsinduktion

Definition C1H: neutrales Spiel

Ein **neutrales Spiel** (G, v) mit konstanter Summe $c \in \mathbb{R}$ besteht aus einem artinschen Graphen $G = (X, A, \sigma, \tau)$ mit Auszahlung $v : \partial X \to \bar{\mathbb{R}}$.

Beide Spieler ziehen abwechselnd gemäß den Kanten A des Graphen.
Neutral bedeutet: Beide Spieler haben dieselben Aktionen zur Auswahl; allein die Unterscheidung in ziehend und wartend bricht die Symmetrie.

In jedem terminalen Zustand $x \in \partial X$ erhält der Ziehende schließlich die Auszahlung $v(x)$ und der Wartende das Komplement $c - v(x)$.
Was ist die Auszahlung des ziehenden Spielers bei optimalem Spiel?

Satz C1H: Gewinnfunktion durch Rückwärtsinduktion

Dazu existiert genau eine **Gewinnfunktion** $u : X \to \bar{\mathbb{R}}$ mit $u|_{\partial X} = v$ und

$$u(x) \overset{!}{=} \sup_{x \to y} \big[c - u(y)\big] = c - \inf_{x \to y} u(y)$$

Eine **Lösung** oder **optimale Strategie** s ordnet jedem aktiven Zustand $x \in X^\circ$ eine optimale Aktion $s(x) = a : x \to y$ mit $u(x) = c - u(y)$ zu.

Neutrale Spiele und Rückwärtsinduktion

Übung: Beweisen Sie Existenz und Eindeutigkeit der Gewinnfunktion u.

☺ Die Beweisidee ist anschaulich klar und aus Beispielen vertraut:
Wir beweisen Existenz durch Rekursion, Eindeutigkeit durch Induktion.
Allerdings ist nicht offensichtlich, worüber hier induziert werden soll...
Versuchen Sie es zunächst selbst als Fingerübung, dann hilft Satz B1F.

Bemerkung: Formal ist die Max-Min-Bedingung die Bellman–Gleichung mit konstanter sofortiger Belohnung $r(a) = c$ und Diskontfaktor $\delta = -1$.
Diese Parameterwahl codiert unser Modell der konstanten Summe c.

Der Diskontfaktor $\delta \in [0, 1]$ beschreibt den schrittweisen **Wertverlust**, zum Beispiel durch Inflation, Abbruchrisiko oder allgemein Ungeduld.

Der Faktor $\delta = -1$ codiert die schrittweise **Wertumkehr** wie in C1H:
Beide Spieler verfolgen dasselbe Ziel, mit umgekehrten Vorzeichen.

☺ Diese simple Beobachtung ist zunächst nur eine formale Analogie, doch sie dient zur Wiederverwendung der Techniken aus Kapitel B.

Neutrale Spiele und Rückwärtsinduktion

Ein **Nullsummenspiel** ist ein Spiel (G, v) mit konstanter Summe $c = 0$. Jeder Spieler versucht, wie immer, seinen Gewinn zu maximieren; hier ist dies äquivalent dazu, den Gewinn seines Gegners zu minimieren. Solche Spiele sind strikt kompetitiv, sie erlauben keine Kooperation.

Spiele mit konstanter Summe $c \in \mathbb{R}$ sind flexibler in der Formulierung. Allgemein: Was der eine Spieler gewinnt, geht dem anderen verloren. Sie sind äquivalent zu Nullsummenspielen durch Translation:

Übung: Sei (G, v) ein neutrales Spiel mit konstanter Summe $c \in \mathbb{R}$ und Gewinnfunktion u. Sei $\tilde{v} = \lambda v + \kappa$ mit $\lambda \in \mathbb{R}_{>0}$ und $\kappa \in \mathbb{R}$. Dann ist $(G, \tilde{v})$ ebenfalls ein neutrales Spiel mit konstanter Summe $\tilde{c} = \lambda c + 2\kappa$ und Gewinnfunktionen $\tilde{u} = \lambda u + \kappa$. Speziell für $(\lambda, \kappa) = (2, -c)$ folgt $\tilde{c} = 0$.

Beispiele: Bei kombinatorischen Spielen nutzen wir im Folgenden die Auszahlungen $0 =$ *Verlust* und $1 =$ *Gewinn* mit konstanter Summe $c = 1$. Ebenso möglich wäre $-1 =$ *Verlust* und $+1 =$ *Gewinn* mit Summe $c = 0$. In manchen Spielen ist auch ein $0 =$ *Unentschieden* möglich.

Neutrale Spiele und Rückwärtsinduktion

Bemerkung: Wir können alle Aktionen mit gemeinsamem Start und Ziel zusammenfassen, da es keine sofortigen Belohnungen gibt, sondern nur die terminale Auszahlung $v : \partial X \to \bar{\mathbb{R}}$, auf die wir alles ausrichten.

Für jedes neutrale Spiel (G, v) genügt also ein **einfacher Graph** G: Von jedem Start $x \in X$ zu jedem Ziel $y \in X$ existiert höchstens eine Kante, die Randabbildung $\partial = (\sigma, \tau) : A \to X \times X$ ist somit injektiv.

Die obige Beweis zeigt Eindeutigkeit und Existenz der Gewinnfunktion $u : X \to \bar{\mathbb{R}}$. In vielen Anwendungen ist der Graph zudem lokal-endlich, das heißt: Jeder Zustand $x \in X$ erlaubt nur endlich viele Aktionen.

Für jeden lokal-endlichen Graphen wird das Maximum/Minimum jeweils angenommen: Es existiert also (mindestens) eine optimale Strategie. Diese explizit zu berechnen, kann jedoch beliebig komplex sein.

Genau darum geht es uns im Folgenden: Wenn wir das Spiel gewinnen wollen, dann müssen wir eine Gewinnstrategie nicht nur prinzipiell berechnen können, sondern möglichst effizient!

Was ist eine effiziente Lösung?

Definition C1I: Lösung, allgemein und polynomiell

Sei (G, v) ein neutrales Spiel mit Summe $c \in \mathbb{R}$ und der Gewinnfunktion $u : X \to \bar{\mathbb{R}}$, also $u|_{\partial X} = v$ und $u(x) = \sup_{x \to y}[c - u(y)]$ für alle $x \in X^\circ$.

Eine **Lösung** oder **optimale Strategie** s ordnet jedem aktiven Zustand $x \in X^\circ$ eine optimale Aktion $s(x) = a : x \to y$ mit $u(x) = c - u(y)$ zu.

Unter einer **polynomiellen Lösung** verstehen wir einen Algorithmus mit polynomieller Laufzeit, der eine optimale Aktion $x \mapsto a$ berechnet.

Beispiel: Die Eingabe einer natürlichen Zahl messen wir in Bitlänge:

$$\operatorname{len} : \mathbb{N} \to \mathbb{N} : x \mapsto \min\{\, \ell \in \mathbb{N} \mid a < 2^\ell \,\}$$

Wir betrachten einzeiliges Nim mit Aktionen $a \in S = \{1, 2, \ldots, n-1\}$: Zum Zustand $x \in \mathbb{N}$ erfordert die memoisierte Rekursion x Schritte, ist also exponentiell in $\operatorname{len}(x)$. Satz C1G bietet eine polynomielle Lösung: Es genügt $a = x \bmod n$ zu berechnen. **Übung:** Führen Sie dies aus! Warum ist der Zeitaufwand bei festem n höchstens linear in $\operatorname{len}(x)$?

Was ist eine effiziente Lösung?

Ist der Graph G **lokal-endlich und artinsch**, so existiert eine Lösung: Die Rückwärtsinduktion C1H konstruiert eine optimale Strategie $x \mapsto a$. Oft erfordert das exponentiellen Aufwand. *Ain't nobody got time for that!*

Was bedeutet das genau? Wie messen wir Komplexität und Aufwand? Zustände und Aktionen seien codiert durch $X, A \hookrightarrow \{0, 1, |, (,), \ldots\}^{(\mathbb{N})}$. Wir messen die Komplexität $\operatorname{len}(x)$ als Länge über diesem Alphabet. Im binären Falle, wie oben erklärt, messen wir die Länge in Bits.

Ein vorgelegter **Algorithmus** zur Berechnung einer Strategie $s : x \mapsto a$ hat zur Eingabe x eine gewisse Laufzeit $T(x) \in \mathbb{R}_{\geq 0}$, etwa gemessen in Sekunden oder Taktzyklen. Gilt $T(x) \leq c_0 + c_1 \operatorname{len}(x)^\alpha$ mit Konstanten $c_0, c_1, \alpha \in \mathbb{R}_{\geq 0}$, so ist die Laufzeit (höchstens) polynomiell vom Grad α. Gilt hingegen $T(x) \geq c_0 \exp(c_1 \operatorname{len}(x))$ mit Konstanten $c_0, c_1 \in \mathbb{R}_{>0}$, so ist die Laufzeit (mindestens) exponentiell, und somit nicht polynomiell.

Unter einer **polynomiellen Lösung** verstehen wir einen Algorithmus mit polynomieller Laufzeit, der jedem aktiven Zustand $x \in X^\circ$ eine optimale Aktion $a : x \to y$ zuordnet, sodass $u(x) = c - u(y)$ gilt.

Normalspiel und Sprague–Grundy–Funktion

Wir vereinfachen weiter und erlauben nur zwei mögliche Auszahlungen: entweder 0 für null / falsch / Niederlage oder 1 für eins / wahr / Gewinn. Die konstante Summe sei $c = 1$, also $u \mapsto 1 - u$ die logische Negation. Wir nennen dann (G, v) ein **neutrales kombinatorisches Spiel**.

Satz C1J: Normalspiel und Sprague–Grundy–Funktion

Auf dem artinschen Graphen $G = (X, A, \sigma, \tau)$ betrachten wir:

- Das **Misèrespiel** $(G, 1)$: Wer nicht mehr ziehen kann, gewinnt.
- Das **Normalspiel** $(G, 0)$: Wer nicht mehr ziehen kann, verliert.

Hierzu existieren die eindeutigen Gewinnfunktionen $\mu, \nu : X \to \{0, 1\}$ mit den vorgegebenen konstanten Randwerten $\mu|_{\partial X} = 1$ und $\nu|_{\partial X} = 0$.

Ist G zudem lokal-endlich, so existiert die **Sprague–Grundy–Funktion**

$$\gamma : X \to \mathbb{N} : \gamma(x) = \operatorname{mex}\{\, \gamma(y) \mid x \to y \,\},$$
$$\operatorname{mex} : \{\, S \subsetneq \mathbb{N} \,\} \to \mathbb{N} : S \mapsto \min(\mathbb{N} \setminus S).$$

Es gilt $\nu = \gamma \wedge 1$, also $\nu(x) = 0$ gdw $\gamma(x) = 0$ und $\nu(x) = 1$ gdw $\gamma(x) \geq 1$.

Normalspiel und Sprague–Grundy–Funktion

Beweis: Rückwärtsinduktion C1H garantiert Existenz und Eindeutigkeit von $\mu, \nu : X \to \{0, 1\}$, ebenso für $\gamma : X \to \mathbb{N}$ dank lokaler Endlichkeit. (Andernfalls formulieren wir mex und γ für Ordinalzahlen.)

Wir nutzen hier die übliche und bequeme Notation $a \wedge b := \min\{a, b\}$.
Wir vergleichen $\nu : X \to \{0, 1\}$ und $\gamma : X \to \mathbb{N}$ und zeigen $\nu = \gamma \wedge 1$:

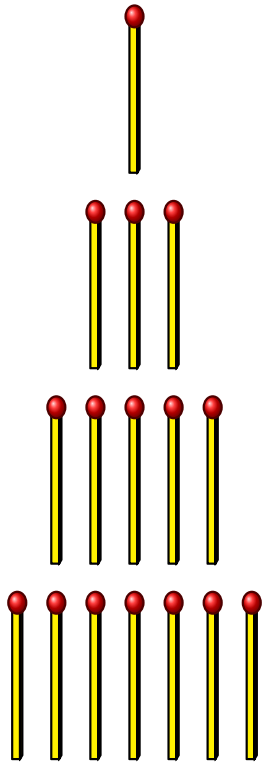
Wir betrachten eine Ecke $x \in X$. Gilt $\nu(y) = \gamma(y) \wedge 1$ für alle $x \to y$, dann folgt $\nu(x) = \gamma(x) \wedge 1$. Hierzu unterscheiden wir die beiden Fälle:

Gewinnposition: Führt ein Zug $x \to y$ zu $\nu(y) = 0$, so gilt $\nu(x) = 1$.
In diesem Falle gilt $\gamma(y) = 0$, also $\gamma(x) \geq 1$, und somit $\nu(x) = \gamma(x) \wedge 1$.

Verlustposition: Führen alle Züge $x \to y$ zu $\nu(y) = 1$, so gilt $\nu(x) = 0$.
In diesem Falle gilt $\gamma(y) \geq 1$, also $\gamma(x) = 0$, und somit $\nu(x) = \gamma(x) \wedge 1$.

Daraus folgt $\nu = \gamma \wedge 1$: Gäbe es $x_0 \in X$ mit $\nu(x_0) \neq \gamma(x_0) \wedge 1$, so gilt $x_0 \in X^\circ$, und es gibt einen Nachfolger $x_0 \to x_1$ mit $\nu(x_1) \neq \gamma(x_1) \wedge 1$.
So fortfahrend erhalten wir einen unendlichen Weg $x_0 \to x_1 \to x_2 \to \ldots$.
Das widerspricht unserer Voraussetzung, dass der Graph G artinsch ist.

Das Spiel Nim: Formalisierung als Graph

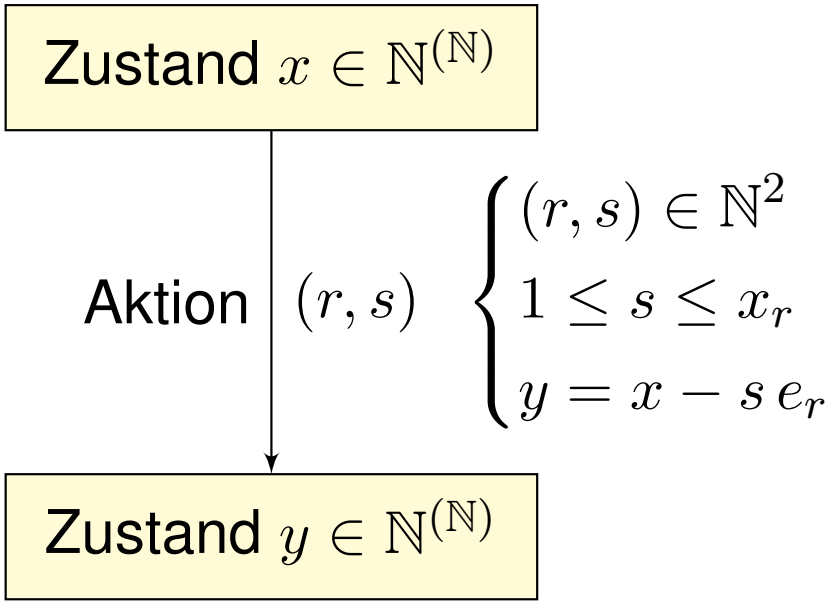


$x_4 = 1$

$x_3 = 3$

$x_2 = 5$

$x_1 = 7$



Aufgabe: Formulieren Sie dieses Spiel in Worten und als Graph.

Lösung: Nim ist ein Spiel für zwei Spieler; beide ziehen abwechselnd.
Wir betrachten das Normalspiel: Wer nicht mehr ziehen kann, verliert.
Die Zustandsmenge ist $X = \mathbb{N}^{(\mathbb{N})}$. Gegeben sei ein Zustand $x \neq 0$.
Der ziehende Spieler wählt einen Zug $(r, s) \in \mathbb{N}^2$ mit $1 \leq s \leq x_r$.
Neuer Zustand ist $y = x - s\, e_r$. Nun zieht der andere Spieler.

Das Spiel Nim: Formalisierung als Graph

Dieses Spiel ist sehr einfach, doch wenn Sie es zum ersten Mal sehen, werden Sie anfangs vermutlich wenig oder keine Struktur erkennen. Erinnern Sie sich an Ihre Erfahrung mit einzeiligem Nim!

Zur Demonstration spielen wir dieses Spiel ausgiebig in der Vorlesung, damit Sie das Problem und seine elegante Lösung wirklich spüren. In hoffe diese experimentelle Spielphase ist gut investierte Zeit.

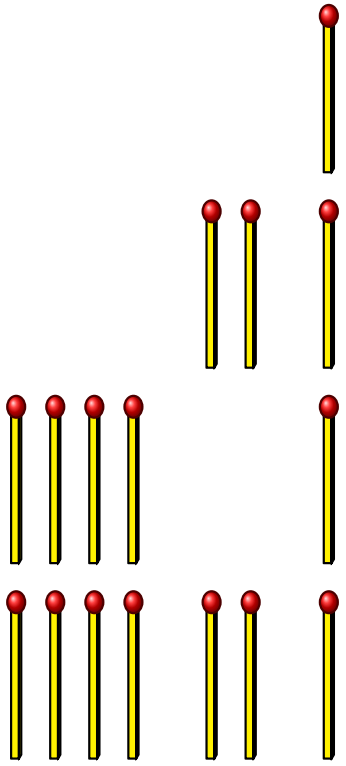
Manche Spezialfälle entdecken und verstehen Sie leicht im Spiel: Zum Beispiel ist $(x_1, x_2) \in \mathbb{N}^2$ mit $x_1 = x_2$ eine Verlustposition, denn der zweite Spieler kann jeden Zug des ersten spiegeln.

Ein allgemeine Position $(x_1, x_2, \ldots, x_n) \in \mathbb{N}^n$ hingegen ist nicht so einfach zu durchschauen. Das müssen Sie selbst ausprobieren und persönlich erfahren! Erst dann können Sie die geniale Lösung würdigen.

Das Spiel Nim wurde gelöst von Charles L. Bouton: *Nim, a game with a complete mathematical theory*, Annals of Mathematics 3 (1901) 35–39.

Der Satz C1o von Sprague–Grundy (1935–39) überführt allgemein jedes neutrale Spiel in ein Nim-Spiel. Dies ist der Ausgangspunkt der kombinatorischen Spieltheorie, die seit etwa 1970 entwickelt wurde.

Das Spiel Nim: effiziente Lösung nach Charles Bouton 1901



$x_4 = 001$

$x_3 = 011$

$x_2 = 101$

$x_1 = 111$

Wir entwickeln x_i im Binärsystem

$$x_i = \sum_{k=0}^{m} \langle x_i \rangle_k \, 2^k \quad \text{mit} \quad \langle x_i \rangle_k \in \{0, 1\}$$

und bilden die Spaltensumme

$$\langle s \rangle_k = \sum_{i=1}^{n} \langle x_i \rangle_k \bmod 2.$$

Binäre Summe ohne Übertrag:

$$s = \sum_{k=0}^{m} \langle s \rangle_k \, 2^k =: x_1 \oplus x_2 \oplus \cdots \oplus x_n$$

Satz C1K: effiziente Lösung des Nim-Spiels, Bouton 1901

Im Normalspiel sind die Verlustpositionen x genau die Nullpositionen:

(0) Ist $x \in \mathbb{N}^n$ eine Null-Position, also $x_1 \oplus x_2 \oplus \cdots \oplus x_n = 0$, so führt jeder Zug $x \to y$ in eine Nicht-Null-Position, $y_1 \oplus y_2 \oplus \cdots \oplus y_n \neq 0$.

(1) Ist x eine Nicht-Null-Position, $x_1 \oplus x_2 \oplus \cdots \oplus x_n \neq 0$, so existiert ein Zug $x \to y$ in eine Null-Position, also $y_1 \oplus y_2 \oplus \cdots \oplus y_n = 0$.

Das Spiel Nim: effiziente Lösung nach Charles Bouton 1901

Den Beweis führen wir allgemein für Satz C1L, noch besser Satz C1O.

☺ Mit diesem simplen Algorithmus können wir die Gewinnfunktion $\nu : X \to \{0, 1\}$ des Normalspiels schnell und einfach berechnen.

Das ist eine effiziente Lösung im Sinne unserer Definition C1I:
Der Zeitaufwand ist linear in der Bitlänge der Eingabe $x \in \mathbb{N}^n$:

$$\begin{aligned} \operatorname{len} &: \mathbb{N} \to \mathbb{N} \quad : x \mapsto \min\{\, \ell \in \mathbb{N} \mid a < 2^\ell \,\} \\ \operatorname{len} &: \mathbb{N}^n \to \mathbb{N} \; : \; x \mapsto \operatorname{len}(x_1) + \cdots + \operatorname{len}(x_n) \end{aligned}$$

Linearer Aufwand ist das Beste, was wir je erwarten können: Die Lösung zu berechnen ist somit nicht aufwändiger, als das Problem zu lesen!

☺ Die Funktion ν liefert uns die optimale Auszahlung $\nu(x)$ und zudem optimale Aktionen $x \mapsto a \in \operatorname{Arg\,max}[\dots]$, also eine optimale Strategie!

Optimale Züge erkennen Sie leicht sobald Sie das Optimum kennen:
Wir haben dies zuvor bereits für die Bellman–Gleichung illustriert.

Für das tatsächliche *Spielen* ist dies der entscheidende Schritt.
Daher führe ich dies im folgenden Satz gebrauchsfertig aus.

Das Spiel Nim: effiziente Lösung nach Charles Bouton 1901

Wir nutzen die **Binärentwicklung**: Jede natürliche Zahl $z \in \mathbb{N}$ schreibt sich eindeutig als Summe $z = \sum_{k\in\mathbb{N}} z_k 2^k$ mit Ziffern $(z_k)_{k\in\mathbb{N}} \in \{0,1\}^{(\mathbb{N})}$.

$$\begin{array}{ccccc} \mathbb{N} & \underset{|-|}{\overset{\langle - \rangle}{\rightleftarrows}} \cong & \{0,1\}^{(\mathbb{N})} & \overset{\cong}{\rightleftarrows} & \mathbb{F}_2[T] \\ \sum_{k\in\mathbb{N}} z_k 2^k & \leftrightarrow & (z_k)_{k\in\mathbb{N}} & \leftrightarrow & \sum_{k\in\mathbb{N}} z_k T^k \end{array}$$

Diese Bijektion $\mathbb{N} \cong \mathbb{F}_2[T]$ ist kein Isomorphismus: $(\mathbb{N},+) \not\cong (\mathbb{F}_2[T],+)$.
Algebraisch: $(\mathbb{F}_2[T],+)$ ist eine Gruppe, hingegen $(\mathbb{N},+)$ nur ein Monoid.
Arithmetisch: In $(\mathbb{F}_2[T],+)$ wird ohne, in $(\mathbb{N},+)$ mit Übertrag addiert.

Wir definieren auf $\mathbb{N}$ die **binäre Addition ohne Übertrag** durch

$$\oplus \;:\; \mathbb{N}\times\mathbb{N} \to \mathbb{N} \;:\; (x,y) \mapsto |\langle x\rangle + \langle y\rangle|.$$

Damit ist $(\mathbb{N},\oplus) \cong (\mathbb{F}_2[T],+)$ eine abelsche Gruppe, gar $\mathbb{F}_2$–Vektorraum. Wir definieren $\bigoplus_{i=0}^{n} x_i = x_0 \oplus x_1 \oplus \cdots \oplus x_n$ wie in jedem abelschen Monoid induktiv für $n = 0,1,2,\dots$ und allgemein für $x \in \mathbb{N}^{(I)}$ mit endlichem Träger $J \subseteq I$ ebenso $\bigoplus x := \bigoplus_{i\in I} x_i := \bigoplus_{i\in J} x_i$.

Das Spiel Nim: effiziente Lösung nach Charles Bouton 1901

In der Informatik ist dies eine sehr vertraute Operation: bitweises XOR! Sie ist schnell und einfach zu berechnen, für Mensch wie für Computer.

Auch in der Mathematik ist $(\mathbb{N}, \oplus) \cong (\mathbb{F}_2[T], +)$ eine vertraute Struktur, nämlich eine abelsche Gruppe, genauer sogar ein $\mathbb{F}_2$–Vektorraum.

Aufgrund von Satz C1κ nennen manche Autoren (im Rahmen der Spieltheorie) diese Verknüpfung $\oplus : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ auch **Nim-Summe**.

Fun fact: Die Koeffizienten $x_1, x_2, \ldots, x_n \in \mathbb{N}$ sind natürliche Zahlen, engl. *natural numbers*. Im Kontext des Nim-Spiels nennen manche Autoren dies auch *nimbers*, als Wortspiel und als Erinnerung.

Elwyn R. Berlekamp, John H. Conway, Richard K. Guy:
Winning Ways for Your Mathematical Plays. A K Peters 2001-2004
Gewinnen: Strategien für mathematische Spiele. Vieweg 1985-1986

John H. Conway: *On Numbers and Games.* A K Peters 2000

Aaron N. Siegel: *Combinatorial Game Theory.* AMS 2013

Das Spiel Nim: effiziente Lösung nach Charles Bouton 1901

Auch die Polynommultiplikation überträgt sich zu $x \odot y = |\langle x \rangle \cdot \langle y \rangle|$.
So wird $(\mathbb{N}, \oplus, \odot)$ eine isomorphe Kopie des Polynomrings $(\mathbb{F}_2[T], +, \cdot)$.

Auf den ersten Blick scheinen Polynomringe vielleicht kompliziert, selbst der einfachsten Vertreter $\mathbb{F}_2[T]$. Das ist keine objektive Schwierigkeit:

Neutral betrachtet ist die Arithmetik in $\mathbb{F}_2[T]$ viel einfacher als in $\mathbb{N}$:
In $\mathbb{F}_2[T]$ wird **ohne Übertrag** addiert und multipliziert.

Aus mathematischer Sicht ist die Definition von $(\mathbb{N}, \oplus)$ nicht tiefsinnig.
Überaus bemerkenswert ist hingegen, dass die binäre Addition $(\mathbb{N}, \oplus)$ und die Anordnung $(\mathbb{N}, <)$ für Spiele so trickreich zusammenarbeiten.
Für das Produkt $(\mathbb{N}, \odot)$ ist mir keine ähnliche Anwendung bekannt.

L'algèbre est généreuse, elle donne souvent plus qu'on lui demande.
[Die Algebra ist großzügig, sie gibt oft mehr, als wir von ihr verlangen.]
Jean Le Rond d'Alembert (1717–1783)

Das Spiel Nim: effiziente Lösung nach Charles Bouton 1901

Nim: starke Lösung nach Sprague 1935, Grundy 1939

Boutons Lösung C1K erklärt Gewinn- und Verlustpositionen für Nim.
Als Verfeinerung bestimmen wir nun die Sprague–Grundy–Funktion:

Satz C1L: Sprague–Grundy–Funktion des Nim-Spiels

Wir untersuchen das Nim-Spiel $G = (X, A, \sigma, \tau)$ mit den Zuständen $X = \mathbb{N}^{(\mathbb{N})}$ und Aktionen $(r, s) : x \to y$ mit $1 \leq s \leq x_r$ und $y = x - s\, e_r$.

Für seine Sprague–Grundy–Funktion $\gamma : X \to \mathbb{N}$ gilt:

$$\gamma(x) = \bigoplus\nolimits_{i \in \mathbb{N}} x_i = x_0 \oplus x_1 \oplus x_2 \oplus \dots.$$

Das löst das Spiel mit polynomiellem Zeitaufwand. Ausführlich gilt:

1. Für jede Aktion $(r, s) : x \to y$ gilt $x_r \neq y_r$, also $\gamma(x) \neq \gamma(y)$.
2. Zu $0 \leq n < \gamma(x)$ existiert eine Aktion $(r, s) : x \to y$ mit $\gamma(y) = n$:
   Wir berechnen $z := \gamma(x) \oplus n = 2^\ell + \sum_{k=0}^{\ell-1} z_k 2^k$ und wählen $r \in \mathbb{N}$ mit $\langle x_r \rangle_\ell = 1$. Somit gilt $y_r := x_r \oplus z < x_r$ und $\gamma(y) = \gamma(x) \oplus z = n$.

Das beschreibt insbesondere alle Gewinnzüge $x \to y$ mit $\gamma(y) = 0$.

Nim: starke Lösung nach Sprague 1935, Grundy 1939

☺ Dieser Satz beinhaltet Boutons Lösung (Satz C1K):
Jede Position $x \in X$ mit $\gamma(x) = 0$ ist eine Verlustposition.
Jede Position $x \in X$ mit $\gamma(x) \geq 1$ ist eine Gewinnposition.

☺ Wir werden diese geniale Methode in Satz C1O perfektionieren.
Der Beweis ist jeweils leicht und ich formuliere ihn detailliert aus.

☹ Wenn Sie zuerst den Beweis lesen, sagt er Ihnen noch allzu wenig.
Der Beweis ist einfach genial und genial einfach: Sie können ihn Schritt für Schritt leicht durcharbeiten, doch dabei besteht die Gefahr, dass der zündende Funke nicht überspringt. Das wäre hier besonders schade!

☺ Zunächst empfehle ich, den Algorithmus in zahlreichen Bei-Spielen zu erproben. Durch eigenständiges Probieren wird Ihnen schnell klar, wie das Verfahren funktioniert und wie ein Beweis aussehen könnte.

Übung: Experimentieren Sie mit dem genialen Algorithmus des Satzes!
Wenden Sie das Verfahren mehrfach an, dann beweisen sie den Satz.
So können Sie selbst den Beweis entdecken, entwickeln und verstehen.

Nim: starke Lösung nach Sprague 1935, Grundy 1939

Beweis: Die erste Aussage (1) ist klar, da $(\mathbb{N}, \oplus)$ eine Gruppe ist.

(2) Vom Start $m = \gamma(x)$ zum Ziel $n < m$ steht das höchste zu ändernde Bit an der angegebenen Stelle ℓ. Dank $m > n$ gilt $\langle m \rangle_\ell = 1$ und $\langle n \rangle_\ell = 0$.

Demnach existiert mindestens eine Koordinate $r \in \mathbb{N}$ mit $\langle x_r \rangle_\ell = 1$.
Genauer existiert eine ungerade Anzahl dieser Wahlmöglichkeiten.

Dies stellt sicher, dass $y_r := x_r \oplus z$ die Ungleichung $y_r < x_r$ erfüllt.
Diese Wahl r und $s = x_r - y_r$ definieren die Aktion $(r, s) : x \to y$.
Nach Konstruktion gilt $\gamma(y) = \bigoplus_{i \in \mathbb{N}} y_i = (\bigoplus_{i \in \mathbb{N}} x_i) \oplus z = \gamma(x) \oplus z = n$.

Die beiden Eigenschaften (1–2) zeigen, dass die Funktion $\gamma : X \to \mathbb{N}$ tatsächlich die Sprague–Grundy–Funktion des Nim-Spiels G ist. QED

Lemma C1M: Umformulierung der mex–Eigenschaft

Vorgelegt sei ein Graph $G = (X, A, \sigma, \tau)$ mit einer Funktion $\gamma : X \to \mathbb{N}$.
Dann sind (1–2) äquivalent zu $\gamma(x) = \mathrm{mex}\{\, \gamma(y) \mid x \to y \,\}$ für alle $x \in X$.

Ist G zudem artinsch und lokal-endlich, so existiert hierzu genau eine Lösung $\gamma : X \to \mathbb{N}$, und dies ist die Sprage–Grundy–Funktion (C1J).

Nim: starke Lösung nach Sprague 1935, Grundy 1939

Aufgabe: Vorgelegt sei eine Position $x \in X = \mathbb{N}^{(\mathbb{N})}$ im Nim-Spiel.
(1) Wie finden Sie von x aus alle möglichen Gewinnzüge $(r,s) : x \to y$?
(2) Wie finden Sie zu $0 \leq n < \gamma(x)$ alle Züge $(r,s) : x \to y$ mit $\gamma(y) = n$?

Lösung: Frage (1) ist ein Spezialfall von (2), nämlich für $n = 0$.
Es genügt daher, die allgemeinere Fragestellung (2) zu lösen.

Der Zug $(r,s) : x \to y$ bedeutet $x_r \to y_r = x_r - s e_r$.
Somit gilt $\gamma(y) = \gamma(x) \oplus z$ mit $z = \gamma_i(x_i) \oplus \gamma_i(y_i)$. Wir wollen $\gamma(y) = n$.

Wir berechnen also zunächst $z := \gamma(x) \oplus n = 2^\ell + \sum_{k=0}^{\ell-1} z_k 2^k$
und suchen nun alle Züge $x_r \to y_r$ mit $y_r = x_r \oplus z$.

Im Falle $\langle r_r \rangle_\ell = 0$ gilt $x_r \oplus z > x_r$: Das ist im Nim–Spiel nicht erlaubt.

Im Falle $\langle x_r \rangle_\ell = 1$ gilt $x_r \oplus z < x_r$: Diese Verringerung können wir im Nim-Spiel realisieren durch den Zug $(r,s) : x \to y$ mit $s = x_r - (x_r \oplus z)$.

(Dies zeigt, dass es mindestens eine solche Koordinate $r \in \mathbb{N}$ gibt, genauer existiert immer eine ungerade Anzahl solcher Koordinaten.)

Wir finden so alle Züge $(r,s) : x \to y$ mit $y_r = x_r \oplus z$ und somit $\gamma(y) = n$.

Erinnerung: Produkt und Summe von Gruppen

Zu jedem Index $i \in I$ sei $G_i = (X_i, +_i, 0_i, -_i)$ eine abelsche Gruppe mit Grundmenge X_i und darauf der Addition $+_i : X_i \times X_i \to X_i$ mit neutralem Element $0_i \in X_i$ und Negation $-_i : X_i \to X_i$.

Das **Produkt** $P = \prod_{i \in I} G_i := (X, +, 0, -)$ hat als Grundmenge

$$X = \prod_{i \in I} X_i = \left\{ x : I \to \bigcup_{i \in I} X_i : i \mapsto x_i \;\middle|\; x_i \in X_i \right\}.$$

Hierauf definieren wir koordinatenweise die Addition $x + y = (x_i +_i y_i)_{i \in I}$, das neutrale Element $0 = (0_i)_{i \in I}$ und die Negation $-x = (-_i x_i)_{i \in I}$.

Wir definieren den **Träger** $\operatorname{supp} : P \to \mathfrak{P}I : x \mapsto \{ i \in I \mid x_i \neq 0_i \}$.
Wir definieren die **Summe** der Gruppen $(G_i)_{i \in I}$ durch

$$S = \bigoplus_{i \in I} G_i := \left\{ x \in P \;\middle|\; \operatorname{supp}(x) \text{ endlich} \right\}.$$

Übung: Mit dieser Konstruktion ist $(P, +, 0, -)$ eine abelsche Gruppe. Hierin ist die Summe $S \subseteq P$ eine Untergruppe. Dasselbe gilt für darauf aufbauende Strukturen wie R–Moduln, K–Vektorräume, etc., ebenso für Ringe und K–Algebren, nicht jedoch mit Eins, denn für $\sharp I = \infty$ hat das Einselement $1 = (1_i)_{i \in I}$ keinen endlichen Träger, also $1 \in P$, aber $1 \notin S$.

Erinnerung: Produkt und Summe von Gruppen

Diese Konstruktion von Produkt und Summe begegnet uns sehr häufig. Allgemeiner gelingt diese Konstruktion für abelsche Monoide $(X_i, +_i, 0_i)$, noch allgemeiner Tripel aus einer Menge X_i mit innerer Verknüpfung $+_i : X_i \times X_i \to X_i$ und idempotenten Element $0_i \in X_i$, also $0_i +_i 0_i = 0_i$.

Im Falle $G_i = G$ für alle $i \in I$ haben wir Produkt und Summe:

$$P = G^I \quad := \{\, x : I \to G \,\}$$
$$S = G^{(I)} := \{\, x : I \to G \mid \operatorname{supp}(x) \text{ endlich} \,\}$$

Nur auf letzterem haben wir die Summenabbildung

$$\sum : G^{(I)} \to G : (g_i)_{i \in I} \mapsto \textstyle\sum_{i \in I} g_i.$$

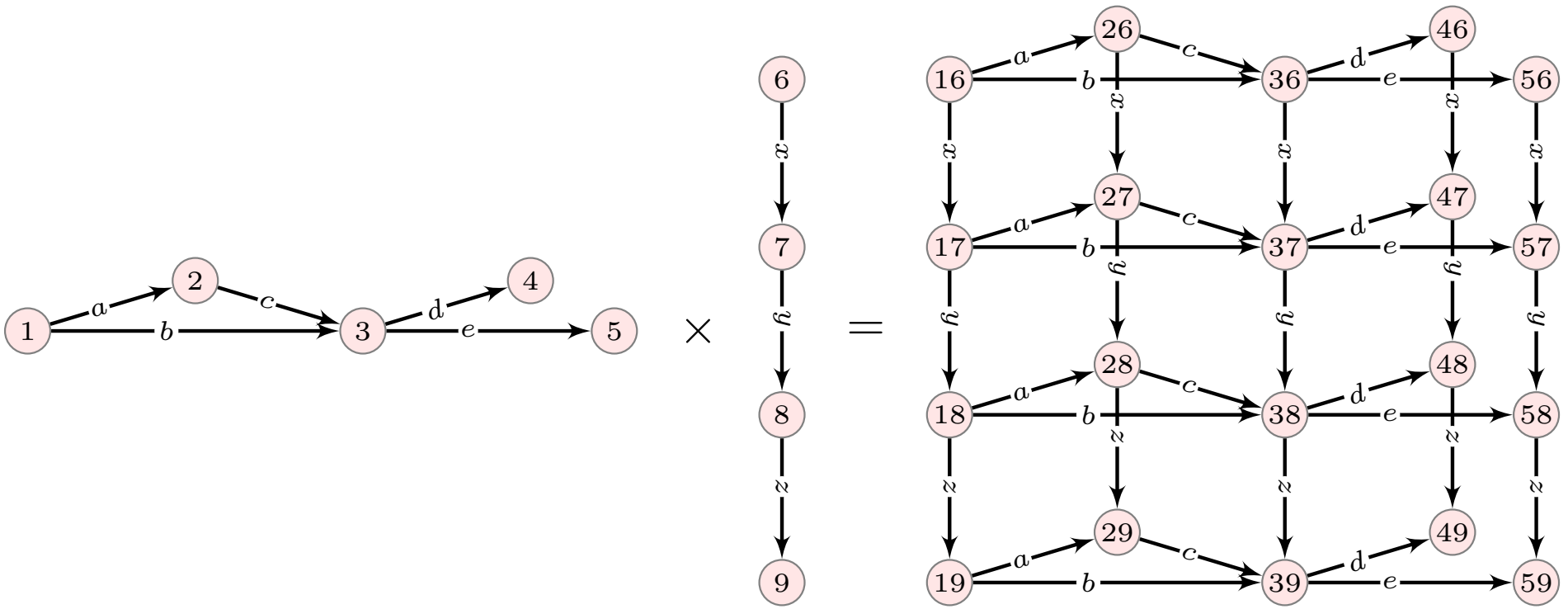
Beispiel: Für den Sprague–Grundy–Satz nutzen wir insbesondere:

Aus der abelschen Gruppe $(\mathbb{N}, \oplus, 0)$ erhalten wir die Gruppe $\mathbb{N}^{(\mathbb{N})}$ mit koordinatenweiser Addition $\oplus$ und hierauf $\bigoplus : \mathbb{N}^{(\mathbb{N})} \to \mathbb{N}$.

Aus dem abelschen Monoid $(\mathbb{N}, +, 0)$ erhalten wir das Monoid $\mathbb{N}^{(\mathbb{N})}$ mit koordinatenweiser Addition $+$ und hierauf $\sum : \mathbb{N}^{(\mathbb{N})} \to \mathbb{N}$.

Produkt und Summe von Spielen

Wir spielen mehrere Spiele G_i parallel, also gleichzeitig nebeneinander.
Der ziehende Spieler darf sich aussuchen, in welchem Spiel G_i er zieht.
Der gemeinsame Spielgraph ist somit das Produkt bzw. die Summe:



Seien $G_1 = (X_1, A_1, \sigma_1, \tau_1)$ und $G_2 = (X_2, A_2, \sigma_2, \tau_2)$ zwei Graphen.
Ihr Produkt $G = G_1 \times G_2$ hat die Eckenmenge $X = X_1 \times X_2$.
Eine Kante $(i, a_i) : (x_1, x_2) \to (y_1, y_2)$ ist entweder von der Form
$a_1 : x_1 \to y_1$ mit $x_2 = y_2$ oder von der Form $a_2 : x_2 \to y_2$ mit $x_1 = y_1$.

Produkt und Summe von Spielen

⚠ Es gibt mehrere Möglichkeiten, Produkte von Graphen zu definieren. Verschiedene Autoren verwenden verschiedene Namen und Notationen. Gegen mögliche Verwirrung hilft nur eine präzise Definition:

Definition C1N: Produkt und Summe von Spielen

Gegeben sei eine Familie $(G_i)_{i\in I}$ von Graphen $G_i = (X_i, A_i, \sigma_i, \tau_i)$.

Ihr **Produkt** ist der Graph $P = \prod_{i\in I} G_i = (X, A, \sigma, \tau)$
mit der Eckenmenge $X = \prod_{i\in I} X_i$ und der Kantenmenge

$$A = \left\{ (x, i, a_i, y) \;\middle|\; x, y \in X,\ i \in I,\ a_i : x_i \to y_i,\ \forall j \neq i : x_j = y_j \right\}$$

sowie den Projektionen $\sigma(x, i, a_i, y) = x$ und $\tau(x, i, a_i, y) = y$.

Die **Summe** $S = \bigoplus_{i\in I} G_i = (X', A', \sigma', \tau')$ ist der volle Teilgraph $S \leq P$ der Zustände $x \in X$ mit endlichem Träger $\operatorname{supp}(x) := \{ i \in I \mid x_i \in X_i^\circ \}$.

Der Träger $\operatorname{supp}(x) \subseteq I$ indiziert aktive Koordinaten $i \in I$ mit $x_i \in X_i^\circ$.
Ist I endlich, so ist das Produkt gleich der Summe, doch in unendlichen Summen verlangen wir, dass nur endlich viele Koordinaten aktiv sind.

Der Sprague–Grundy–Satz

Satz C1O: Sprague 1935, Grundy 1939

Gegeben seien lokal-endliche artinsche Graphen $G_i = (X_i, A_i, \sigma_i, \tau_i)$.
Dann ist auch ihre Summe $G = \bigoplus_{i \in I} G_i$ lokal-endlich artinsch,
und für ihre Sprague–Grundy–Funktion gilt $\gamma(x) = \bigoplus_{i \in I} \gamma_i(x_i)$.

Ausführlich folgt $\gamma(x) = \operatorname{mex}\{\, \gamma(y) \mid x \to y \,\}$ aus zwei Eigenschaften:

1. Für jede Aktion $(i, a_i) : x \to y$ gilt $a_i : x_i \to y_i$ im Graphen G_i.
   Für die Grundy–Zahlen folgt $\gamma_i(x_i) \neq \gamma_i(y_i)$, somit $\gamma(x) \neq \gamma(y)$.
2. Zu $0 \leq n < \gamma(x)$ existiert eine Aktion $(i, a_i) : x \to y$ mit $\gamma(y) = n$:

   Wir lösen $\gamma(x) \oplus z = n$ durch $z = \gamma(x) \oplus n = 2^\ell + \sum_{k=0}^{\ell-1} z_k 2^k$
   und wählen $i \in I$ mit $\langle \gamma_i(x_i) \rangle_\ell = 1$. Somit gilt $\gamma_i(x_i) \oplus z < \gamma_i(x_i)$.

   In G_i existiert eine Aktion $a_i : x_i \to y_i$ mit $\gamma_i(y_i) = \gamma_i(x_i) \oplus z$.
   In G erhalten wir $(i, a_i) : x \to y$ mit $\gamma(y) = \gamma(x) \oplus z = n$.

Dank (2) finden wir insbesondere Gewinnzüge $x \to y$ mit $\gamma(y) = 0$.

Beispiel: Im Spiel Nim erhalten wir die Nim-Summe aus Satz C1K/C1L.

Der Sprague–Grundy–Satz

☺ Der Satz ist konstruktiv als gebrauchsfertiger Algorithmus formuliert. Sie können dieses Verfahren direkt gewinnbringend anwenden. . . und sich dann nachfolgend den allgemeinen Beweis erarbeiten.

☹ Wenn Sie zuerst den Beweis lesen, sagt er Ihnen noch allzu wenig. Der Beweis ist einfach genial und genial einfach: Sie können ihn Schritt für Schritt leicht durcharbeiten, doch dabei besteht die Gefahr, dass der zündende Funke nicht überspringt. Das wäre hier besonders schade!

☺ Zunächst empfehle ich, den Algorithmus in zahlreichen Bei-Spielen zu erproben. Durch eigenständiges Probieren wird Ihnen schnell klar, wie das Verfahren funktioniert und wie ein Beweis aussehen könnte.

☺ Als konkretes Beispiel und methodische Vorarbeit haben wir oben das Nim-Spiel diskutiert. Dieses leuchtende Vorbild hilft Ihnen!

Übung: Experimentieren Sie mit dem genialen Algorithmus des Satzes! Wenden Sie das Verfahren mehrfach an, dann beweisen sie den Satz. So können Sie selbst den Beweis entdecken, entwickeln und verstehen.

Der Sprague–Grundy–Satz

Beweis: Zunächst ist G lokal-endlich und artinsch: Gäbe es in G einen unendlichen Weg $x^0 \to x^1 \to x^2 \to \dots$, so ist $J = \operatorname{supp}(x^0)$ endlich, und in mindestens einer Koordinate $i \in J$ finden wir einen unendlichen Weg $x_i^{n_0} \to x_i^{n_1} \to x_i^{n_2} \to \dots$, und somit wäre G_i nicht artinsch. Widerspruch!

Für jeden Zustand $x \in X$ ist $\gamma(x) := \bigoplus_{i \in I} \gamma_i(x_i)$ wohldefiniert, denn für fast alle $i \in I$ gilt $x_i \in \partial X_i$ und somit $\gamma_i(x_i) = 0$.

Die erste Aussage (1) ist klar, da $(\mathbb{N}, \oplus)$ eine abelsche Gruppe ist.

(2) Vom Start $m = \gamma(x)$ zum Ziel $n < m$ steht das höchste zu ändernde Bit an der angegebenen Stelle ℓ. Dank $m > n$ gilt $\langle m \rangle_\ell = 1$ und $\langle n \rangle_\ell = 0$.

Demnach existiert mindestens eine Koordinate $i \in \mathbb{N}$ mit $\langle \gamma_i(x_i) \rangle_\ell = 1$. Genauer existiert eine ungerade Anzahl dieser Wahlmöglichkeiten.

Dies garantiert $\gamma(x_i) \oplus z < \gamma(x_i)$. In G_i existiert demnach eine Aktion $a_i : x_i \to y_i$ mit $\gamma(y_i) = \gamma(x_i) \oplus z$. In G erhalten wir daraus $(i, a_i) : x \to y$. Nach Konstruktion gilt $\gamma(y) = \bigoplus_{i \in \mathbb{N}} y_i = (\bigoplus_{i \in \mathbb{N}} x_i) \oplus z = \gamma(x) \oplus z = n$.

Die beiden Eigenschaften (1–2) zeigen, dass die Funktion $\gamma : X \to \mathbb{N}$ tatsächlich die Sprague–Grundy–Funktion des Spiels G ist (C1M). QED

Der Sprague–Grundy–Satz

Aufgabe: Vorgelegt sei eine Position $x \in X$ im Spiel $G = (X, A, \sigma, \tau)$.
(1) Wie finden Sie von x aus alle möglichen Gewinnzüge (x, i, a_i, y)?
(2) Wie finden Sie zu $0 \leq n < \gamma(x)$ alle Züge (x, i, a_i, y) mit $\gamma(y) = n$?

Lösung: Frage (1) ist ein Spezialfall von (2), nämlich für $n = 0$.
Es genügt daher, die allgemeinere Fragestellung (2) zu lösen.

Der Zug (x, i, a_i, y) im Spiel G bedeutet $a_i : x_i \to y_i$ im Spiel G_i.
Somit gilt $\gamma(y) = \gamma(x) \oplus z$ mit $z = \gamma_i(x_i) \oplus \gamma_i(y_i)$. Wir wollen $\gamma(y) = n$.

Wir berechnen also zunächst $z := \gamma(x) \oplus n = 2^\ell + \sum_{k=0}^{\ell-1} z_k 2^k$
und suchen nun alle Züge $a_i : x_i \to y_i$ mit $\gamma_i(y_i) = \gamma_i(x_i) \oplus z$.

Im Falle $\langle \gamma_i(x_i) \rangle_\ell = 0$ gilt $\gamma_i(x_i) \oplus z > \gamma_i(x_i)$: Geeignete Züge $a_i : x_i \to y_i$ können im Spiel G_i existieren, sie sind möglich, aber nicht zwingend.

Im Falle $\langle \gamma_i(x_i) \rangle_\ell = 1$ gilt $\gamma_i(x_i) \oplus z < \gamma_i(x_i)$: Nach Definition der Grundy–Zahl γ_i existiert im Spiel G_i mindestens ein Zug $a_i : x_i \to y_i$.

(Der Beweis zeigt, dass es mindestens eine solche Koordinate $i \in I$ gibt, genauer existiert immer eine ungerade Anzahl solcher Koordinaten.)

Wir finden so alle Züge (x, i, a_i, y) mit $\gamma_i(y_i) = \gamma_i(x_i) \oplus z$, also $\gamma(y) = n$.

Lasker–Nim: Berechnung dank Sprague–Grundy

„Eine interessante Variation entsteht, wenn man die Spielregel wie folgt festsetzt: Der am Zuge Befindliche darf irgendeinen der Haufen in zwei Haufen zerteilen oder aber, nach seinem freien Ermessen, verkleinern.“
Emanuel Lasker: *Brettspiele der Völker.* Scherl Verlag, Berlin 1931.

Aufgabe: Dieses Spiel heißt **Lasker–Nim**. Lösen Sie es!
Berechnen Sie zu jeder Position $x = (x_1, \ldots, x_n)$ die Grundy–Zahl $\gamma(x)$.

Lösung: Das Spiel ist eine Summe, dank Sprague–Grundy gilt also

$$\gamma : \mathbb{N}^n \to \mathbb{N} : (x_1, \ldots, x_n) \mapsto \gamma(x_1) \oplus \cdots \oplus \gamma(x_n).$$

Wir benötigen demnach nur noch $\gamma : \mathbb{N} \to \mathbb{N}$. Laut Spielregel gilt:

$$\gamma(x) = \operatorname{mex}\Big[\big\{\, \gamma(y) \;\big|\; 0 \le y < x \,\big\} \cup \big\{\, \gamma(y) \oplus \gamma(z) \;\big|\; x = y + z,\ 1 \le y \le z \,\big\}\Big]$$

Wir berechnen folgende Tabelle (bottom up / memoisierte Rekursion):

x	0	1	2	3	4	5	6	7	8	9	10	11	12
γ	0	1	2	4	3	5	6	8	7	9	10	12	11

Lasker–Nim: Berechnung dank Sprague–Grundy

Übung: Rechnen Sie die obigen Fälle $x = 0, 1, 2, \ldots, 12$ sorgfältig nach. Formulieren Sie die allgemeine Regel. Beweisen Sie dies per Induktion.

Übung: Welchen Zeitaufwand $T(x)$ haben beide Methoden für $\gamma(x)$?

1. Die memoisierte Rekursion, wie in obiger Tabelle illustriert.
2. Die allgemeine Regel, die Sie soeben bewiesen haben.

Vergleichen Sie dies mit unserer Diskussion zu einzeiligem Nim.

Übung: Denken Sie sich „zufällig“ einige Spielpositionen $x \in \mathbb{N}^{(\mathbb{N})}$ aus. Ist dies eine Gewinn- oder Verlustposition in Lasker–Nim? Wie nutzen Sie hier Ihre Vorarbeit? Finden Sie alle Gewinnzüge in dieser Position.

Ähnliche Formeln gelten für alle **Take-and-Break-Spiele**: Der ziehende Spieler darf Objekte entfernen und/oder einen Haufen teilen, wozu viele verschiedene Regeln denkbar sind. In *Winning Ways*, Kapitel 4 „Taking and Breaking“ wird für die Familie der sogenannten **Octalen Spiele** eine konzise Notation eingeführt. (`en.wikipedia.org/wiki/Octal_game`)

Grundys Spiel: eine ungelöste Vermutung

Das Spiel beginnt mit einem Haufen von $x \in \mathbb{N}$ Objekten. Der ziehende Spieler teilt einen Haufen seiner Wahl in zwei Haufen ungleicher Größe. Wir vereinbaren Normalspiel: Wer nicht mehr ziehen kann, verliert.

Aufgabe: Dieses Spiel heißt **Grundys Spiel**. Lösen Sie es für kleine x! Berechnen Sie zur Position x die Grundy–Zahl $\gamma(x)$, soweit möglich.

Lösung: Die Sprague–Grundy–Funktion $\gamma : \mathbb{N} \to \mathbb{N}$ ist gegeben durch:

$$\gamma(x) = \operatorname{mex}\Big[\big\{\, \gamma(y) \oplus \gamma(z) \;\big|\; x = y + z,\ 1 \le y < z \,\big\}\Big]$$

Wir berechnen folgende Tabelle (bottom up / memoisierte Rekursion):

x	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
γ	0	0	0	1	0	2	1	0	2	1	0	2	1	3	2	1	3	2	4	3	0

⚠ Anders als bei Lasker–Nim erkennen wir hier kein einfaches Muster, das unsere Rekursion zu einer effizienten Lösung abkürzen könnte.

Frage: Ist diese Sprague–Grundy–Funktion $\gamma : \mathbb{N} \to \mathbb{N}$ periodisch?

Grundys Spiel: eine ungelöste Vermutung

Elwyn Berlekamp, John Horton Conway und Richard Guy vermuten in ihrem Buch *Winning Ways* (1982), dass γ tatsächlich periodisch ist.
Richard Guy: *Unsolved Problems in Combinatorial Games* (1996), `library.msri.org/books/Book29/files/unsolved.pdf`

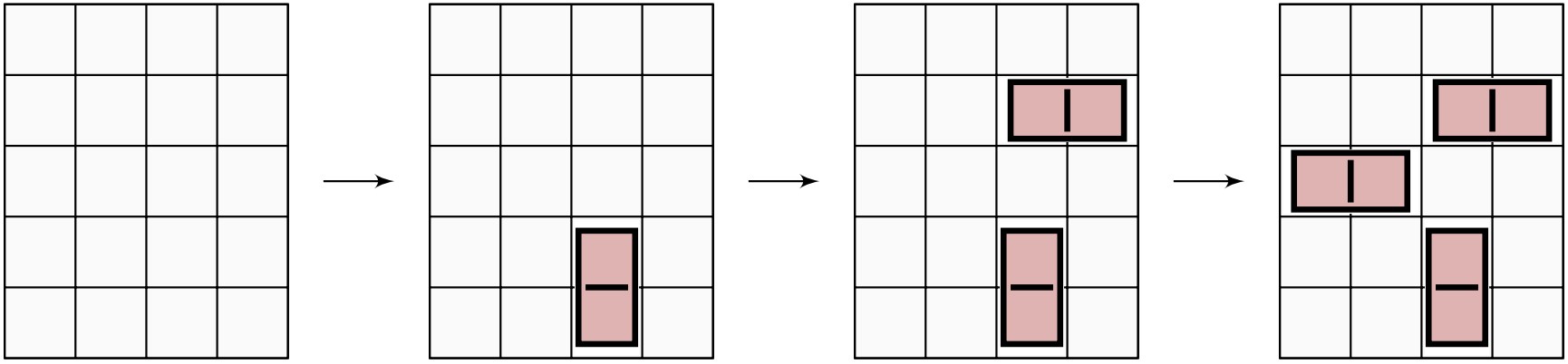
Sie können selbst weitere Werte berechnen, leichter mit Computerhilfe. So hat Achim Flammenkamp die ersten $2^{35} \approx 34 \cdot 10^9$ Werte berechnet, summarisch unter `wwwhomes.uni-bielefeld.de/achim/grundy.html`. Ein periodisches Verhalten konnte allerdings noch niemand entdecken.

Es ist recht erstaunlich, dass eine doch relativ einfache Rekursion ein so kompliziertes Verhalten der Folge nach sich ziehen kann.

Übung: Denken Sie sich „zufällig“ einige Spielpositionen $x \in \mathbb{N}^{(\mathbb{N})}$ aus. Ist dies eine Gewinn- oder Verlustposition in Grundys Spiel? Wie nutzen Sie hier Ihre Vorarbeit? Finden Sie alle Gewinnzüge in dieser Position.

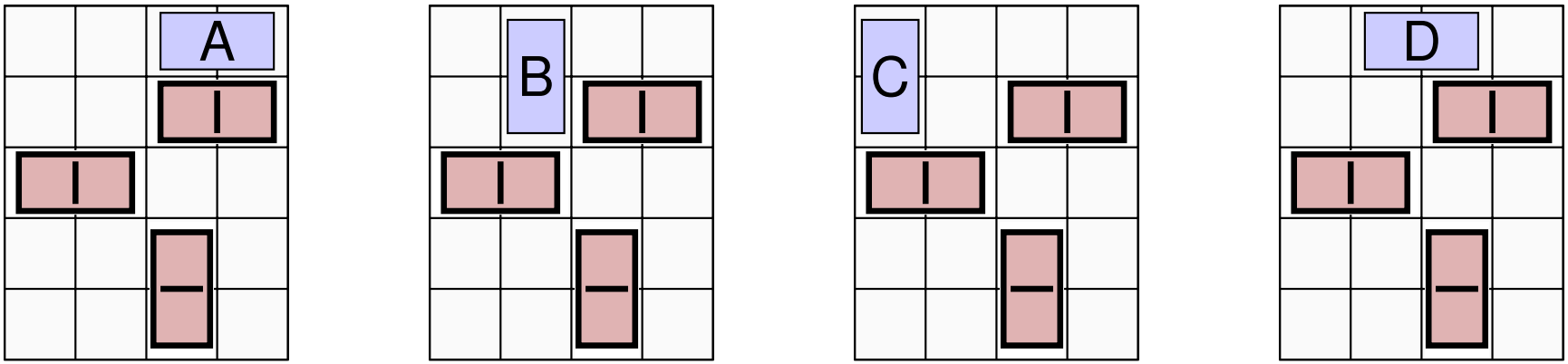
Übung: Spielpositionen $x \in \mathbb{N}^{(\mathbb{N})}$ beschreiben wir am besten sortiert, etwa $(x_1 \geq x_2 \geq \cdots \geq x_n)$ wie für Partitionen üblich. Schreiben Sie den Spielgraphen für einige kleine Startwerte (x_1) möglichst explizit aus.

Fliesentetris: Endspiel-Analyse dank Sprague–Grundy



Das Spielfeld besteht aus Quadraten, zum Beispiel rechteckig 4×5.
Beide Spieler ziehen abwechselnd, der Ziehende legt ein Domino auf zwei benachbarte freie Quadrate. Wer nicht mehr ziehen kann, verliert.

Aufgabe: Wie lässt sich hier der Sprague–Grundy–Satz anwenden?
Im oben skizzierten konkreten Beispiel? Allgemein als Algorithmus?
Welcher der folgenden vier Züge A–D führt zum Gewinn?



Fliesentetris: Endspiel-Analyse dank Sprague–Grundy

Wir parkettieren hier mit Dominos: Das ist Tetris für Fliesenleger!

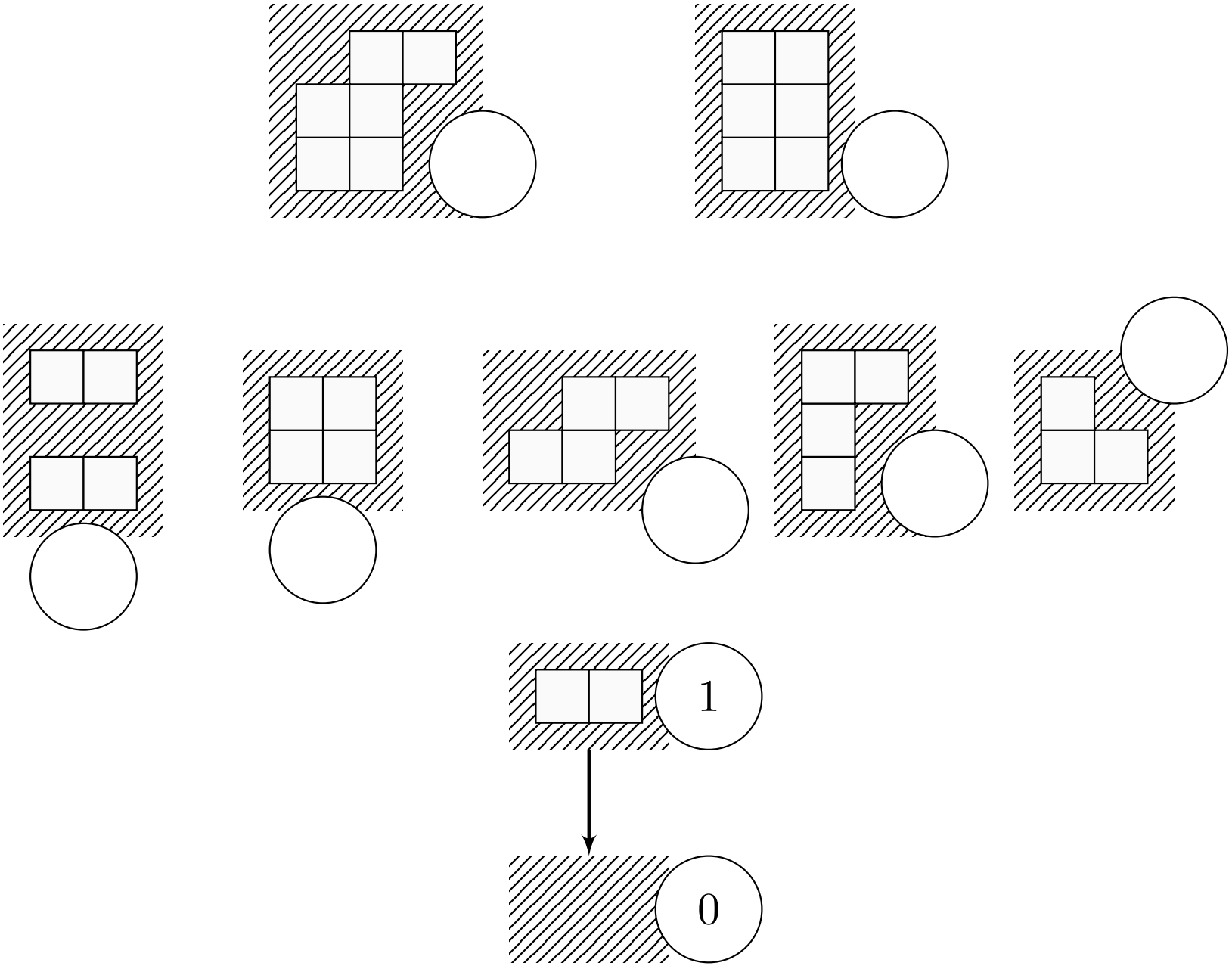
Der hier entdeckte Trick gilt ganz allgemein für **Positionsspiele**:
Die Spieler erobern Positionen mit Spielsteinen und behalten diese.
Im Verlauf entstehen Inseln, also Zusammenhangskomponenten,
die sich nicht mehr gegenseitig beeinflussen. Das nutzen wir gerne:
Das Spiel zerfällt nachfolgend in die Summe seiner Komponenten!

Es genügt daher, jede **Komponente** zu analysieren, den Graphen G_i
zu erstellen und seine Sprague–Grundy–Funktion γ_i zu berechnen.
Für das gesamte (End-)Spiel gilt dann $G = \bigoplus_{i \in I} G_i$ und $\gamma = \bigoplus_{i \in I} \gamma_i$.
Die Berechnung von γ gelingt auf diesem Wege wesentlich effizienter.
Anschließend lesen wir aus $x \mapsto \gamma(x)$ alle Gewinnzüge ab. Voilà!

🙂 Der Sprague–Grundy–Satz lässt sich überall bei Summen einsetzen.
Diese bilden wir willkürlich, indem wir beliebige Spiele parallel spielen.
Manchmal entstehen Summen auch von ganz alleine, ohne unser Zutun.

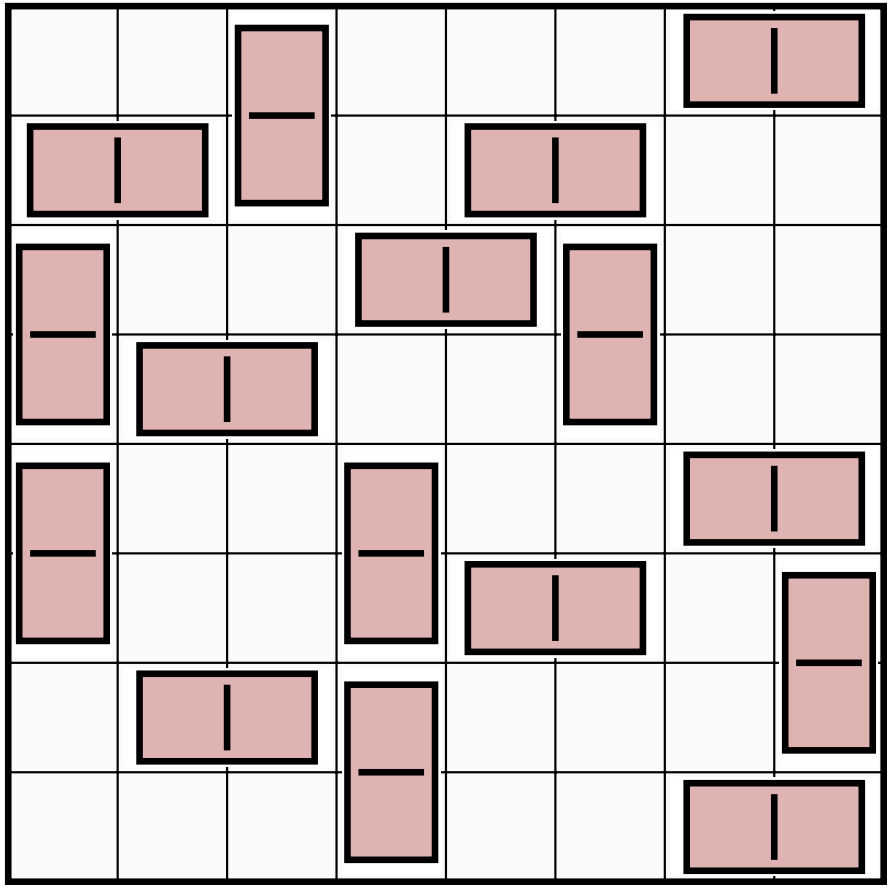
🙂 Das entspricht übrigens der **externen** und der **internen** Summe,
wie Sie dies von Vektorräumen und ähnlichen Strukturen kennen.

Fliesentetris: Endspiel-Analyse dank Sprague–Grundy

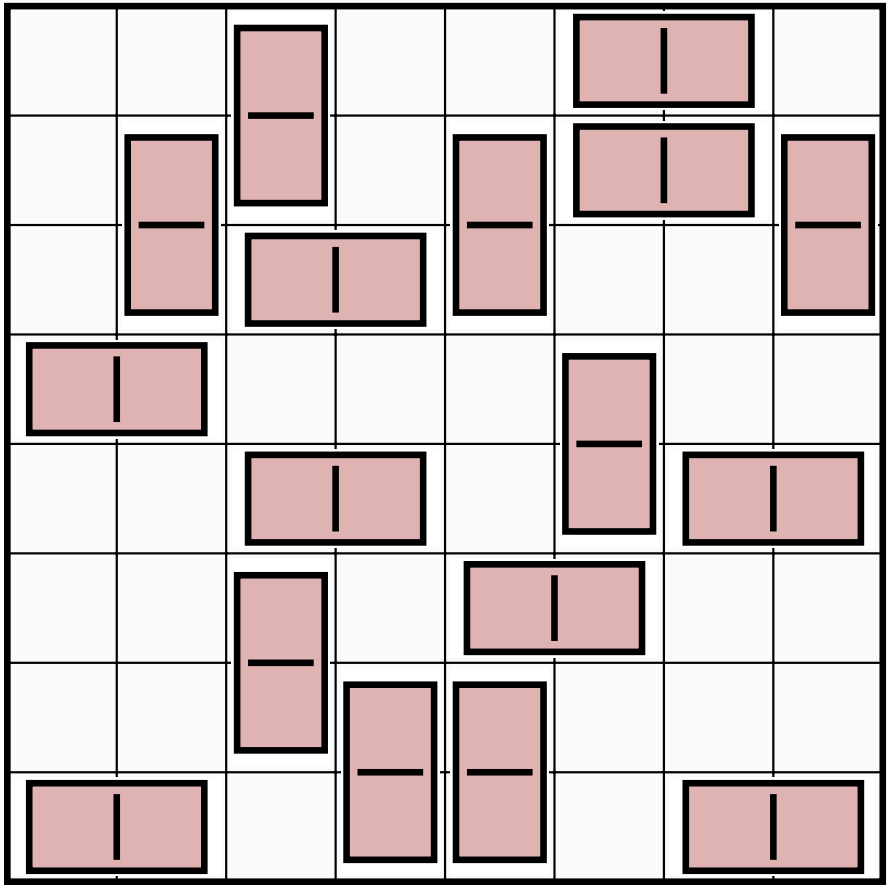


Fliesentetris: Endspiel-Analyse dank Sprague–Grundy

Aufgabe: (1) Bestimmen Sie den Spielgraphen und die Grundy–Zahlen.
(2) Sind die folgenden Spielstände Gewinn- oder Verlustpositionen?
(3) Finden Sie alle Gewinnzüge! Wie viele gibt es?

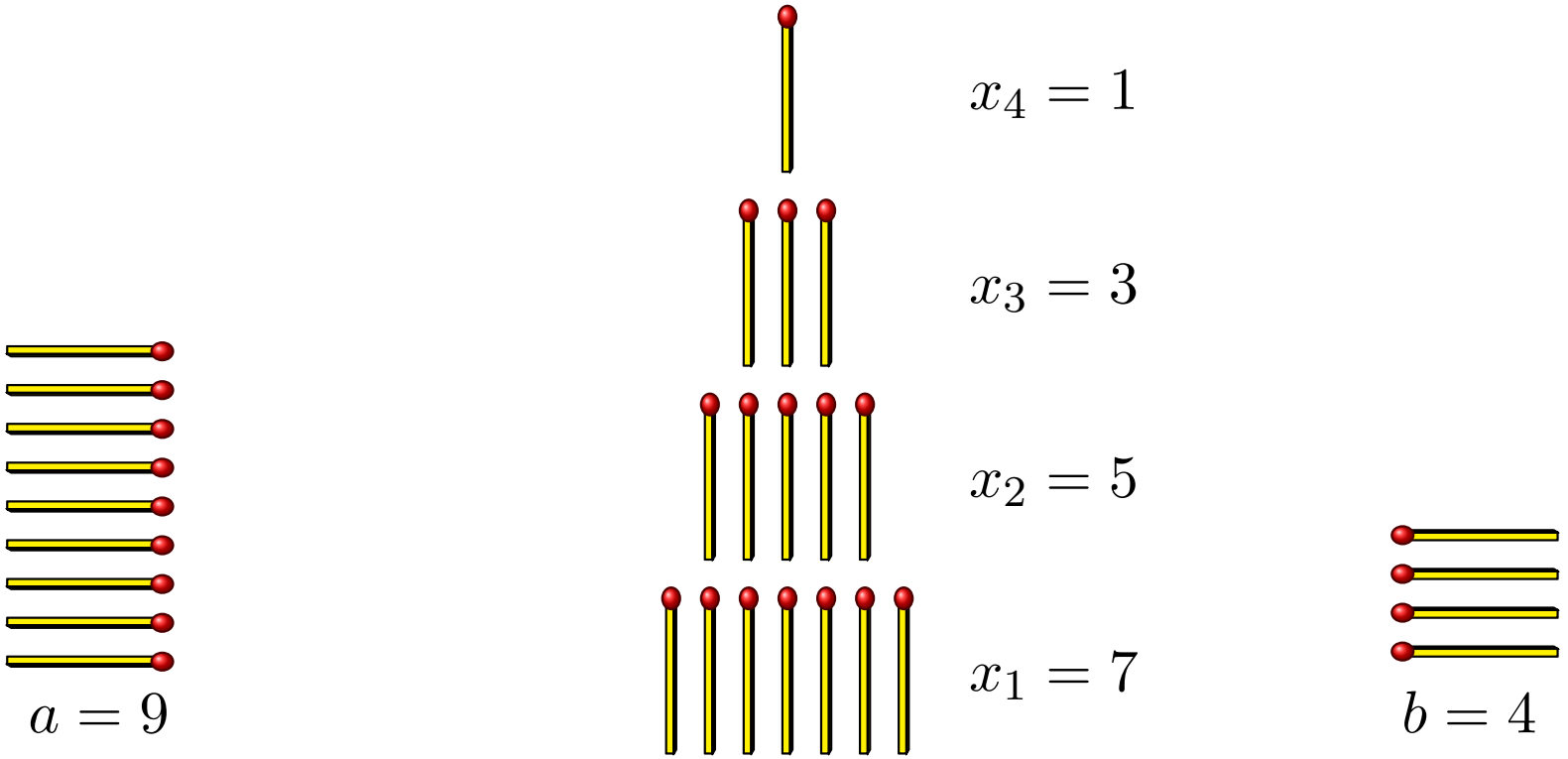


Links: Grundy–Zahl $\gamma(x) = 0$.



Rechts: Grundy–Zahl $\gamma(y) = 1$.

Poker-Nim: Geben ist seliger als Nehmen?



Das Spiel **Poker-Nim** wird gespielt wie Nim mit Spielständen $x \in \mathbb{N}^{(\mathbb{N})}$: Der ziehende Spieler A nimmt $s \geq 1$ Streichhölzer eines Haufens x_r zu seinem Haufen a, oder legt umgekehrt $s \geq 1$ Streichhölzer von a zu x_r. Entsprechend für Spieler B und seinen Haufen b.

Aufgabe: (1) Formalisieren Sie Poker-Nim als ein neutrales Spiel (G, v). (2) Ist die oben gezeigte Position eine Gewinn- oder Verlustposition? Allgemein: Wie erkennen Sie Gewinnpositionen und Gewinnzüge?

Poker-Nim: Geben ist seliger als Nehmen?

Bislang betrachteten wir nur Spiele (G, v) auf artinschen Graphen G, also ohne unendliche Wege $x_0 \to x_1 \to x_2 \to \ldots$, somit ohne Schleifen. Egal wie gespielt wird, das Spiel endet nach endlich vielen Zügen.

Ausgehend von der terminalen Auszahlung $v : \partial G \to \{0, 1\}$ konstruieren wir per Rückwärtsinduktion (C1H) die **Gewinnfunktion** $u : X \to \{0, 1\}$:

(T) Für jeden terminalen Zustand $x \in \partial X$ gilt $u(x) = v(x)$.
(A) Für jeden aktiven Zustand $x \in X^\circ$ gilt $u(x) = \sup_{x \to y}[1 - u(y)]$.

Das bedeutet ausführlich als Fallunterscheidung:
(A0) Falls $u(x) = 0$: Für jeden Zug $x \to y$ gilt $u(y) = 1$.
(A1) Falls $u(x) = 1$: Es existiert ein Zug $x \to y$ mit $u(y) = 0$.

Poker-Nim und Northcotts Spiel (siehe unten) sind erste Beispiele für **Schleifenspiele** (engl. *loopy games* nach John H. Conway 1978).

A priori ist es hier möglich, unendlich lange zu spielen. Bei optimalem Spiel kann jedoch einer der beiden Spieler seinen Gewinn erzwingen!

 Zur Lösung beschränken wir explizit die Zeit bis zum Spielende.

Poker-Nim: Geben ist seliger als Nehmen?

Definition C2A: erweiterte Gewinnfunktion mit Zeitschranke

Sei $G = (X, A, \sigma, \tau)$ ein Graph mit Auszahlung $v : \partial G \to \{0, 1\}$.

Eine **erweiterte Gewinnfunktion** $(u, w) : X \to \{0, 1\} \times \mathbb{N}$ besteht aus einer Gewinnfunktion $u : X \to \{0, 1\}$ und einer Zeitschranke $w : X \to \mathbb{N}$.

(T) Für jeden terminalen Zustand $x \in \partial X$ gilt $w(x) = 0$ und $u(x) = v(x)$.
(A) Für jeden aktiven Zustand $x \in X^\circ$ gilt $w(x) \geq 1$ und zudem:
(A0) Falls $u(x) = 0$: Für jeden Zug $x \to y$ gilt $u(y) = 1$ und $w(y) \leq w(x)$.
(A1) Falls $u(x) = 1$: Es gibt $x \to y$ mit $u(y) = 0$ und $w(y) < w(x)$.

Lösung: Mit dieser genial-einfachen Technik lösen wir Poker-Nim: Zustand $(x, a, b) \in X := \mathbb{N}^{(\mathbb{N})} \times \mathbb{N}^2$, Zug $(r, s) : (x, a, b) \to (x', a', b')$ mit $r \in \mathbb{N}, s \in \mathbb{Z}, 1 \leq s \leq x_r$ oder $1 \leq -s \leq a, x' = x - s\, e_r, a' = b, b' = a + s$.

Die Gewinnfunktion ist $u : X \to \{0, 1\} : (x, a, b) \mapsto 1 \wedge \bigoplus_{i \in \mathbb{N}} x_i$, genau wie beim klassischen Nim-Spiel, nun erweitert durch die explizite Zeitschranke $w : X \to \mathbb{N} : (x, a, b) \mapsto [1 - u(x)]a + u(x)b + \sum_{i \in \mathbb{N}} x_i$.
Dieses Paar (u, w) erfüllt alle Forderungen aus Definition C2A.

Poker-Nim: Geben ist seliger als Nehmen?

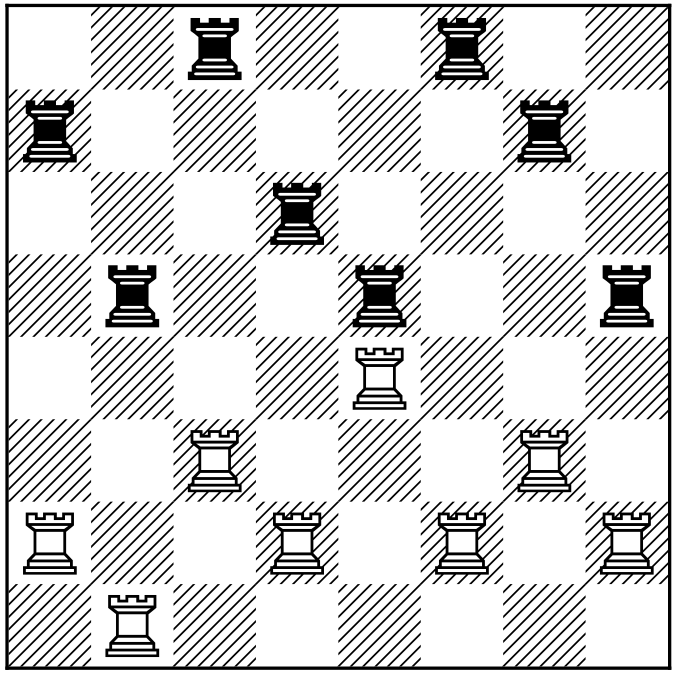
Interpretation: Beim Spielstand $x \in X$ mit $u(x) = 1$ kann der ziehende Spieler seinen Gewinn erzwingen, indem er stets reduzierend zieht (A1); er benötigt dann $\leq w(x)$ Züge bis zum Spielende und seinem Gewinn.

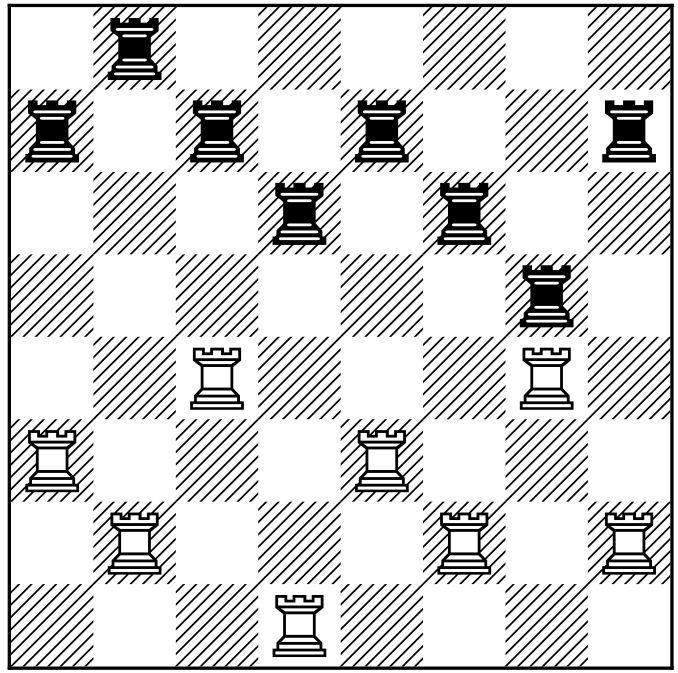
Im Falle $u(x) = 0$ wird der ziehende Spieler verlieren – wie immer bei optimalem Spiel seines Gegners. Der ziehende Spieler kann dies nicht verhindern, sondern höchstens auf $\leq w(x)$ Züge hinauszögern.

 Erlaubt unser Spiel (G, v) eine erweiterte Gewinnfunktion (u, w), so ist damit das Spiel gelöst: Jeder optimale Spielverlauf ist endlich, wir erkennen die Gewinnpositionen an der Eigenschaft $u(x) = 1$ und die (reduzierenden) Gewinnzüge $x \to y$ an $u(y) = 0$ und $w(y) < w(x)$.

Für $w : X \to \mathbb{N}$ genügt statt $(\mathbb{N}, <)$ jede wohlgeordnete Menge $(W, <)$. Dies garantiert bei optimalem Spiel ein Ende nach endlich vielen Zügen. Wir werden diese Verallgemeinerung im Folgenden nicht nutzen. Für lokal-endliche Graphen genügen die natürlichen Zahlen.

Northcotts Spiel: Türme ziehen ohne Ende?





Gezogen wird abwechselnd weiß und schwarz (Markierung am Rand).
Der Ziehende bewegt einen Turm nur vertikal und soweit Platz ist.
Wir vereinbaren Normalspiel: Wer nicht mehr ziehen kann, verliert.

Aufgabe: (1) Formalisieren Sie dies als ein neutrales Spiel (G, v).
Ist der Spielgraph G hier endlich? Wie viele Zustände hat er?
Ist der Spielgraph artinsch? Endet jeder (optimale) Spielverlauf?
(2) Sind die beiden obigen Positionen Gewinn- oder Verlustposition?
Allgemein: Wie erkennen Sie Gewinnpositionen und Gewinnzüge?

Northcotts Spiel: Türme ziehen ohne Ende?

Dieses bekannte Beispiel heißt auch **Northcotts Spiel**.
Es ist insofern ungewöhnlich (und interessant!), als die Regeln allein nicht garantieren, dass jeder Spielverlauf wirklich endet.
Es ist hier durchaus möglich, unendlich lange zu spielen!

Überraschend zeigt sich jedoch, dass bei optimalem Spiel einer der beiden Spieler seinen Gewinn erzwingen kann. Sehen Sie wer und wie?
Dahinter versteckt sich eine weitere Variante des obigen Poker-Nim.
Das ist genau das Ziel der obigen Aufgabe. Probieren Sie es!

Übung: Ist dieses Spiel die Summe seiner Spalten? Falls ja, so lässt sich der Sprague–Grundy–Satz C1o direkt anwenden. Falls nein, so lässt sich das Spiel vielleicht umformulieren in eine äquivalente Summe. Das vereinfacht die Analyse und ermöglicht eine effiziente Lösung!

Übung: Untersuchen Sie folgende Variante: Über bzw. unter jeder Zeile markiert eine Münze, welcher der beiden Türme als nächstes gezogen wird. Nach jedem Zug in dieser Spalte wird die Münze nach unten bzw. oben umgelegt. Ist dieses variierte Spiel die Summe seiner Spalten?

Northcotts Spiel: Türme ziehen ohne Ende?

Definition C2B: erweiterte Grundy–Funktion mit Zeitschranke

Sei $G = (X, A, \sigma, \tau)$ ein Graph. Eine **erweiterte Grundy–Funktion** $(\gamma, w) : X \to \mathbb{N} \times \mathbb{N}$ besteht aus einer Grundy–Funktion $\gamma : X \to \mathbb{N}$ und einer Zeitschranke $w : X \to \mathbb{N}$ mit folgenden Eigenschaften:

(T) Für jeden terminalen Zustand $x \in \partial X$ gilt $w(x) = 0$ und $u(x) = 0$.
(A) Für jeden aktiven Zustand $x \in X^\circ$ gilt $w(x) \geq 1$ und zudem:
(A0) Für jeden Zug $x \to y$ gilt $\gamma(y) \neq \gamma(x)$ und $w(y) \leq w(x)$.
(A1) Zu $0 \leq n < \gamma(x)$ existiert $x \to y$ mit $\gamma(y) = n$ und $w(y) < w(x)$.

Interpretation: Bei $\gamma(x) > 0$ gewinnt der ziehende Spieler das Spiel in $\leq w(x)$ Zügen, wenn er immer reduzierend zieht wie in (A1) erklärt.

☺ Erlaubt unser Spiel $(G, 0)$ eine erweiterte Grundy–Funktion (γ, w), so ist damit das Spiel gelöst: Jeder optimale Spielverlauf ist endlich, wir erkennen die Gewinnpositionen an der Eigenschaft $\gamma(x) \geq 1$ und die (reduzierenden) Gewinnzüge $x \to y$ an $\gamma(y) = 0$ und $w(y) < w(x)$.

Northcotts Spiel: Türme ziehen ohne Ende?

Lemma C2C: Existenz und Eindeutigkeit

Sind (u, w) und (u', w') erw. Gewinnfunktionen zu (G, v), so gilt $u = u'$.
Sind (γ, w) und (γ', w') erw. Grundy–Funktionen zu $(G, 0)$, so gilt $\gamma = \gamma'$.

Ist der Graph G artinsch, so existiert zu (G, v) eine erw. Gewinnfunktion (u, w), und zu $(G, 0)$ eine erw. Grundy–Funktion (γ, w).

☺ Für lösbare Schleifenspiele gilt der Satz von Sprague–Grundy C1O:

Satz C2D: Sprague–Grundy für Schleifenspiele

Gegeben sei eine Familie von Graphen G_i indiziert durch $i \in I$,
jeder mit einer erweiterten Grundy–Funktion $(\gamma_i, w_i) : X_i \to \mathbb{N} \times \mathbb{N}$.

Dann erlaubt ihre Summe $G = \bigoplus_{i \in I} G_i$ die erweiterte Grundy–Funktion $(\gamma, w) : X \to \mathbb{N} \times \mathbb{N}$ mit $\gamma(x) = \bigoplus_{i \in I} \gamma_i(x_i)$ und $w(x) = \sum_{i \in I} w_i(x_i)$.

Aufgabe: Rechnen Sie Satz und Lemma sorgfältig nach!

☺ Damit lösen Sie Summen von Schleifenspielen!

Das Spiel Chomp! nach David Gale

Beim Spiel $\mathrm{Chomp}(m \times n)$ geht es um eine $m \times n$–Tafel Schokolade. Alice und Bob ziehen abwechselnd; die ziehende Spieler:in isst ein Feld $(i, j) \in \{0, \ldots, m-1\} \times \{0, \ldots, n-1\}$ und alle rechts-oben davon. Das Feld $(0, 0)$ links unten ist jedoch bitter; wer es isst, verliert.

Aufgabe: (1) Welcher der beiden Spieler kann den Gewinn erzwingen? (2) Welche Startzüge sind gewinnend? Finden Sie Gewinnstrategien! Inwiefern können Sie hier die Sprague–Grundy–Theorie anwenden?

Das Spiel Chomp! nach David Gale

Fun fact: „Chomp!“ heißt soviel wie „Mampf!“ oder „Schmatz!“.
Consommer avec modération! (Mit Mäßigung zu genießen!)

Das Spiel Chomp ähnelt Nim, mit einem entscheidenden Unterschied:
Beim Nim-Spiel sind die Zeilen unabhängig und werden parallel gespielt.
Bei Chomp hingegen sind Zeilen und Spalten miteinander verbunden,
das Spiel ist daher keine Summe von Zeilen / Spalten-Teilspielen!

Eine Anwendung von Summen ist, wie wir wissen, die Endspiel-Analyse:
Im Spiel Chomp tritt sie nur auf, wenn zwei getrennte Arme übrig sind.

Die geschickte Zerlegung und überaus effiziente Berechnung des
Sprague–Grundy–Satzes steht uns hier sonst nicht zur Verfügung.
Ohne dieses Werkzeug müssen wir meist mühsam rekursiv rechnen.
Das erklärt die erhöhte Komplexität, die wir im Folgenden spüren.

Natürlich können wir dennoch die Sprague–Grundy–Funktion dieses
Spiels berechnen: Wir nutzen sie wie nach wie vor zur externen Summe,
etwa wenn wir mehrere Chomp-Spiele (oder andere) parallel spielen.
Aber sie nützt uns eben leider nicht für eine interne Summenzerlegung.

Das Spiel Chomp: Gewinnen ist möglich, aber wie?

Satz C2E: Symmetrie und Strategieklau

(0) Das Spiel $\mathrm{Chomp}(1 \times n)$ entspricht einzeiligem Nim, kurz $\mathrm{Nim}(n-1)$.

(1) Beim quadratischen Spiel $\mathrm{Chomp}(n \times n)$ beliebiger Größe $n \in \mathbb{N}_{\geq 2}$ hat Alice eine Gewinnstrategie: Sie zieht $(1,1)$ und erhält das Nim-Spiel $\mathrm{Nim}(n-1) \oplus \mathrm{Nim}(n-1)$. Anschließend spiegelt sie jeden Zug von Bob.

(2) Beim rechteckigen Spiel $\mathrm{Chomp}(m \times n)$ mit $m, n \in \mathbb{N}_{\geq 2}$ hat Alice eine Gewinnstrategie, jedoch nur implizit, im Allgemeinen unbekannt.

Beweis: Aussagen (0) und (1) sind klar.

(2) Angenommen, Bob hätte eine Gewinnstrategie $s_B : X^\circ \to A$.
Das gilt insbesondere nach Alice' erstem Zug $(m-1, n-1) : x_0 \to x_1$,
das heißt, Bob hat einen Gewinnzug $s_B(x_1) = (i,j) : x_1 \to x_2$.
Dann hat auch Alice den Gewinnzug $(i,j) : x_0 \to x_2$. QED

Das Spiel Chomp: Gewinnen ist möglich, aber wie?

 Quadratisches Chomp ist gemäß dieser Lösung trivial, als Spiel somit langweilig. Schon rechteckiges Chomp ist notorisch schwierig!

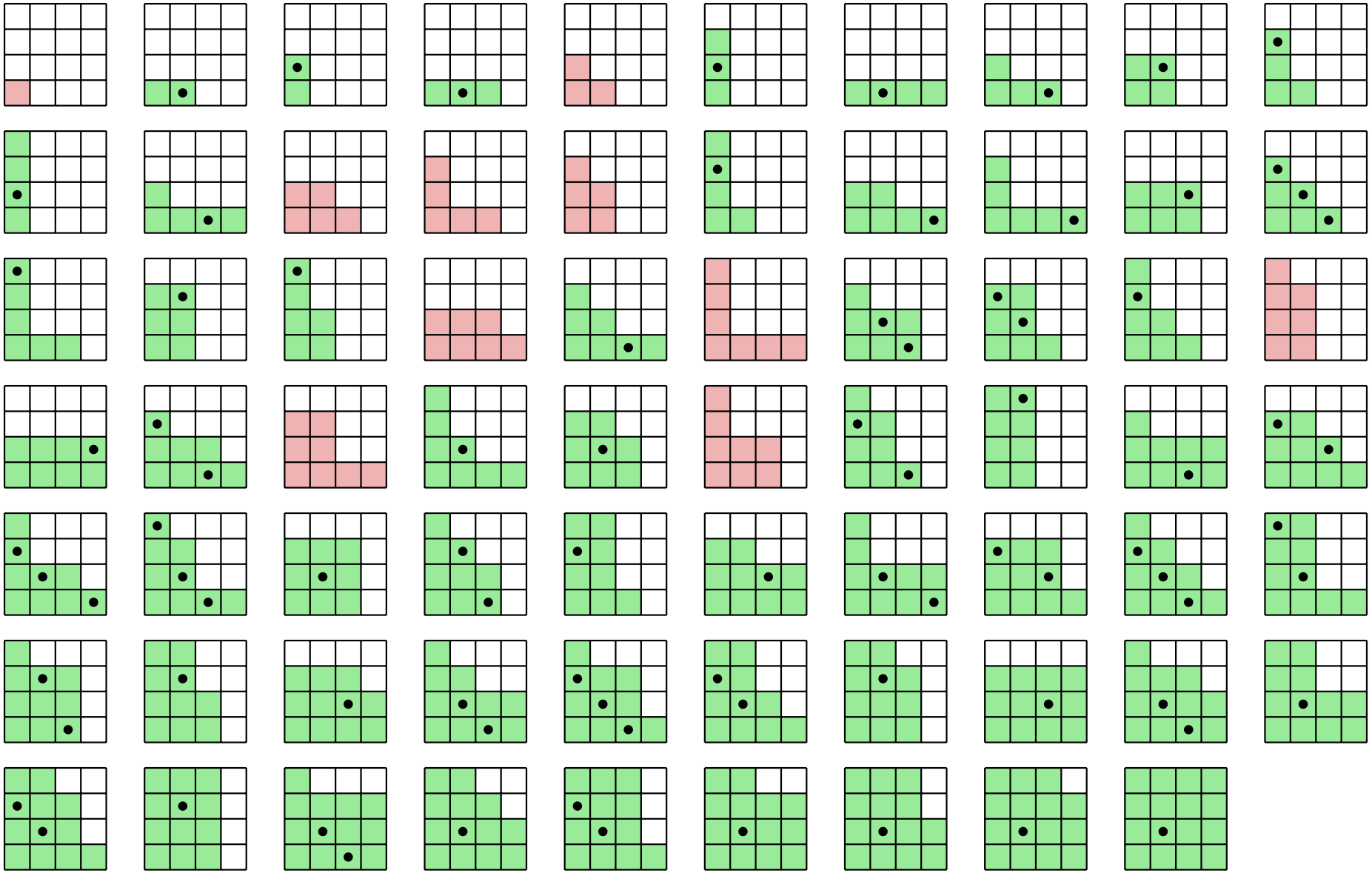
 Wir stehen vor einer erstaunlichen Situation: Alice hat nachweislich eine Gewinnstratgie, diese ist im Allgemeinen jedoch unbekannt!

Das Spiel Chomp! nach David Gale

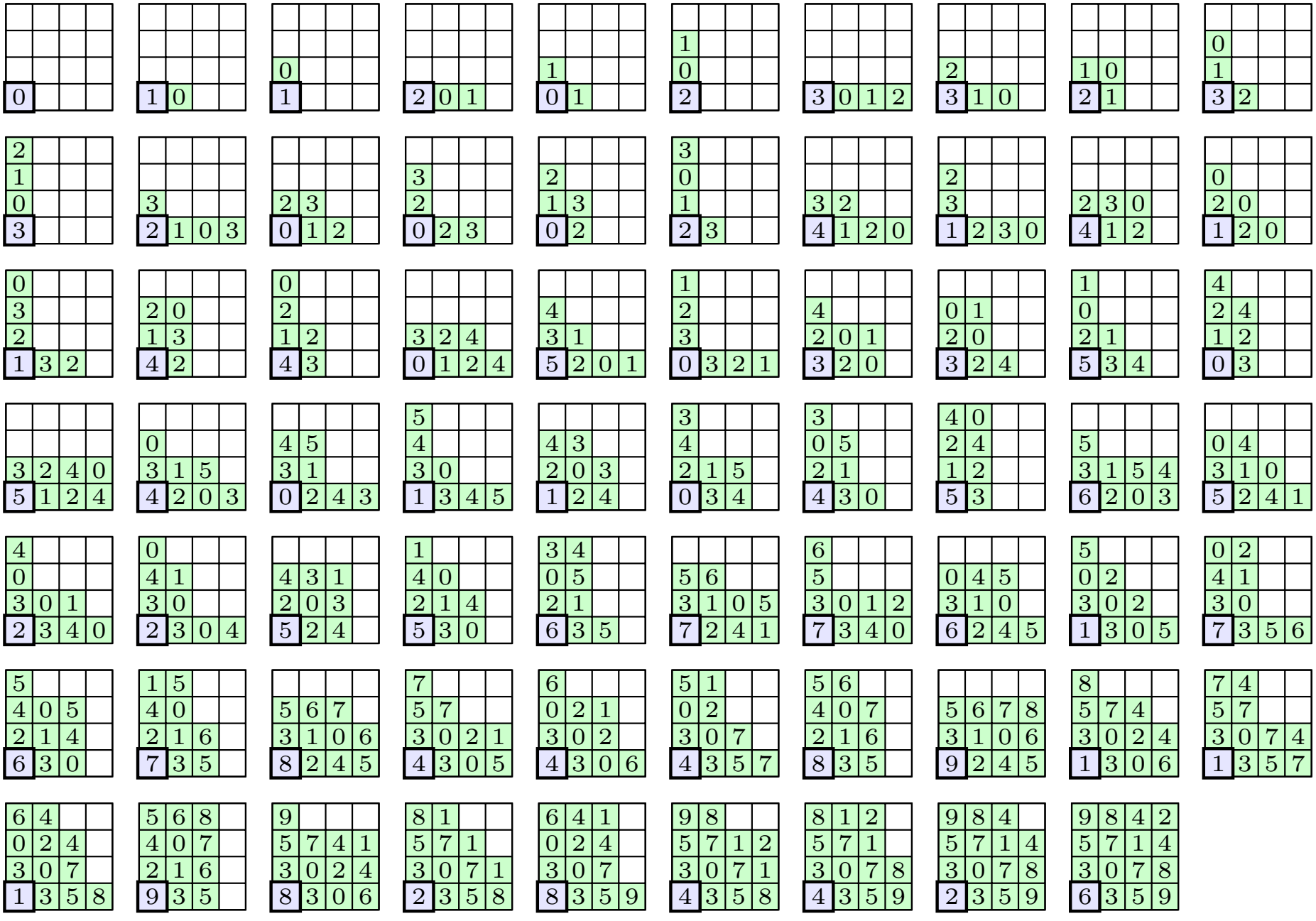
Lösung des Spiels $\mathrm{Chomp}(4 \times 3)$; markiert sind alle Gewinnzüge.

Das Spiel Chomp! nach David Gale

Lösung des Spiels $\mathrm{Chomp}(4 \times 4)$; markiert sind alle Gewinnzüge.



Sprague–Grundy–Zahlen des Spiels $\mathrm{Chomp}(4 \times 4)$



Sprague–Grundy–Zahlen des Spiels $\mathrm{Chomp}(4 \times 4)$

Welche Information beinhalten diese Zahlen? Wozu sind sie gut?

Für jede Position x steht links unten ihre Sprague–Grundy–Zahl $\gamma(x)$. Sie berechnet sich rekursiv wie gewohnt: Für jeden Zug $x \to y$ schauen wir bei der Position y ihren zuvor bereits berechneten Wert $\gamma(y)$ nach. Nach Definition (Satz C1J) finden wir damit $\gamma(x) = \mathrm{mex}\{\, \gamma(y) \mid x \to y \,\}$. Dank $\nu = \gamma \wedge 1$ berechnen wir so zugleich auch alle Gewinnpositionen x mit $\gamma(x) \geq 1$, und auch alle möglichen Gewinnzüge $x \to y$ mit $\gamma(y) = 0$.

Mit Geduld und Sorgfalt können wir also alle Werte $\gamma(x)$ berechnen, wie in obiger Tabelle. Leider erkennen wir hier kein einfaches Muster, das mühsame Rekursion zu einer effizienten Lösung abkürzen könnte.

Nach unserem derzeitigen Erkenntnisstand ist demnach Chomp spürbar komplexer als Nim: Für Nim haben wir eine effiziente Lösung, denn die Rechnung lässt sich in polynomieller Zeit durchführen, sogar linear!

Für Chomp hingegen kommen wir um die rekursive Rechnung nicht herum (soweit wir wissen), daher ist Zeitaufwand zur Berechnung von $x \mapsto \gamma(x)$ exponentiell, und so nur für kleine x realistisch durchführbar.

Das allgemeine Spiel Chomp!

Das Spiel $\mathrm{Chomp}(P,\leq)$ wird gespielt auf einer (partiell) geordneten Menge $(P,\leq)$. Gezogen wird abwechselnd. Die ziehende Spieler:in wählt $a \in P$, neues Teilspiel ist $\mathrm{Chomp}(Q,\leq)$ auf der Restmenge

$$Q = P \setminus P_{\geq a} = \{\, b \in P \mid b \not\geq a \,\}.$$

Misèrespiel: Wir fordern ein kleinstes Element $0 \in P$, also $0 \leq P$.
Wer dieses Element 0 zieht (als somit letzten Zug), verliert.
Das entspricht dem **Normalspiel** auf $P' = P \setminus \{0\}$.

Beispiel: Das Spiel $\mathrm{Chomp}(n)$ mit $n = \{\, 0 \leq 1 \leq 2 \leq \cdots \leq n-1 \,\}$ ist einzeiliges $\mathrm{Nim}(n-1)$. Ebenso für die unendliche Menge $(\mathbb{N},\leq)$.

Bemerkung: Sei $(P,\leq)$ total geordnet. Dann ist das Spiel $\mathrm{Chomp}(P,\leq)$ genau dann artinsch, wenn die Ordnung artinsch ist, das heißt: In $(P,\leq)$ existiert keine unendliche absteigende Kette $a_0 > a_1 > a_2 > \ldots$.

Beispiel: Das Spiel $\mathrm{Chomp}(m \times n)$ mit der Produktordnung entspricht der Schokoladentafel. Ebenso Quader $p \times q \times r$ oder allgemein $(\mathbb{N},\leq)^n$.
Im Produkt $(P,\leq) = \prod_{i\in I}(P_i,\leq_i)$ ist die Ordnung $a \leq b$ erklärt durch $a_i \leq b_i$ für jede Koordinate $i \in I$: punktweiser Vergleich von Funktionen.

Wer gewinnt bei Chomp! und wie?

Für jedes rechteckige Spiel $\mathrm{Chomp}(p \times q)$ existiert eine Gewinnstrategie, sie ist im Allgemeinen jedoch nicht explizit bekannt! Allgemein gilt hierzu:

Satz C2F: Strategieklau im Spiel Chomp

(1) Existiert in $P' = P \setminus \{0\}$ ein Element $m \in P'$, das mit allen $x \in P'$ vergleichbar ist, so hat Bob keine Gewinnstrategie. (2) Ist das Spiel $\mathrm{Chomp}(P, \leq)$ zudem artinsch, so hat Alice eine Gewinnstrategie.

Beweis: (1) Angenommen, Bob hätte eine Gewinnstrategie. Das gilt insbesondere nach Alice' erstem Zug $m : P \to P_1 = P \setminus P_{\geq m}$, das heißt, Bob hat einen Gewinnzug $a : P_1 \to P_2 = P_1 \setminus P_{\geq a}$. Dann hat auch Alice den Gewinnzug $a : P \to P_2$. Das ist ein Widerspruch. QED

Offene Probleme: Für $\mathrm{Chomp}(n \times n \times n)$ ist die Lösung unbekannt.

David Gale bot $\$100$ für eine vollständige Lösung von 3D-Chomp, also allen Spielen $\mathrm{Chomp}(p \times q \times r)$; das beinhaltet 2D-Chomp.

David Gale bot zudem $\$200$ für die Antwort zu folgender Frage:
Hat der erste Spieler in $\mathrm{Chomp}(\mathbb{N} \times \mathbb{N} \times \mathbb{N})$ eine Gewinnstrategie?

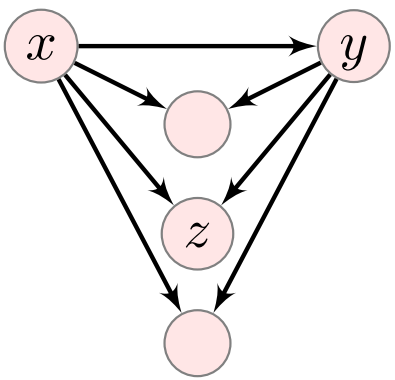
Strategieklau als allgemeine Methode

Diese Beweismethode durch Strategieklau gilt recht allgemein:

Satz C2G: Strategieklau in neutralen Spielen

Sei (G, v) ein neutrales kombinatorisches Spiel auf dem artinschen Graphen $G = (X, A, \sigma, \tau)$ mit terminaler Auszahlung $v : \partial X \to \{0, 1\}$.

Vorgelegt sei $x \in X$. Angenommen, es gibt einen Zug $x \to y$ mit $y \in X^\circ$, sodass für jeden Folgezug $y \to z$ auch $x \to z$ gilt. Dann folgt $u(x) = 1$.



Beweis: Zu (G, v) sei $u : X \to \{0, 1\}$ die Gewinnfunktion. Angenommen, es gälte $u(x) = 0$. Daraus folgt $u(y) = 1$. Somit existiert mindestens ein Zug $y \to z$ mit $u(z) = 0$. Dank $x \to z$ gilt dann $u(x) = 1$. Widerspruch! QED

Für jeden solchen (in y vollen) Teilgraphen ist x eine Gewinnposition.
Übung: Welche weiteren Teilgraphen mit dieser Eigenschaft finden Sie?

- ☺ Der Beweis zeigt $u(x) = 1$, denn ein Gewinnzug $x \to ?$ existiert.
- ☹ Der indirekte Beweis gibt keinen Gewinnzug $x \to ?$ explizit an.

Strategieklau als allgemeine Methode

Übung: (1) Vergleichen Sie die abstrakten Existenzaussagen der beiden Sätze mit explizit-konstruktiven Lösungen. Was nützt wozu?

(2) Warum ist das unendliche Spiel Chomp auf $(\mathbb{N}, \leq)^d$ artinsch? Wir versehen $\mathbb{N}^d$ mit der Produktordnung $x \leq y$ gdw $\forall i : x_i \leq y_i$.

(3) Hat in $\mathrm{Chomp}(\mathbb{N}^2)$ der erste Spieler eine Gewinnstrategie? Beweisen Sie dies konstruktiv-explizit oder abstrakt-existentiell?

Hermann Weyl (1885–1955) formulierte die Problematik sehr treffend: Ein Existenzsatz verkündet „das Vorhandensein eines Schatzes, ohne jedoch zu verraten, an welchem Ort. [. . .] Nicht das Existenztheorem ist das Wertvolle, sondern die im Beweise geführte Konstruktion.“ (*Über die neue Grundlagenkrise der Mathematik*, 1921)

Die Existenz einer Lösung ist wichtig, doch meist nur ein erster Schritt. Die explizite Lösung ist eine ungleich schwierigere Frage, doch gerade für Anwendungen unabdingbar. Das gilt insbesondere für Spiele!

Wie viel Rationalität benötigen wir?

Wir erinnern an unsere Annahme unbeschränkter Rationalität (A2A):

$\mathscr{R}_0$: Jeder Spieler will sein Ergebnis (Nutzen, Gewinn, . . .) maximieren.

$\mathscr{R}_1$: Jeder Spieler versteht zudem alle Spielregeln und Konsequenzen.

$\mathscr{R}_2$: Es gilt die vorige Aussage $\mathscr{R}_1$, und jeder Spieler weiß dies.

Wie viel Rationalität benötigen wir speziell für kombinatorische Spiele?

Angenommen die ziehende Spielerin ist in einer Gewinnposition.
Dann genügt ihr eine optimale Strategie, also eigene Rationalität ($\mathscr{R}_1$):
Damit kann sie gewinnen, unabhängig davon wie ihr Gegner sich verhält.

Angenommen jedoch der ziehende Spieler ist in einer Verlustposition.
Wenn er rational ist ($\mathscr{R}_1$) und dies auch von seiner Gegnerin weiß ($\mathscr{R}_2$),
dann kann er das aussichtslose Spiel aufgeben, denn er wird verlieren.

Wenn er hingegen hofft, dass seine Gegnerin Fehler machen könnte,
dann lohnt sich erwartungsgemäß weiterhin, geschickt zu spielen ($\mathscr{R}_1$).
Sprichwörtlich: *„It ain't over till it's over."* / *„. . . till the fat lady sings."*
Optimist: „Wunder geschehen." Pessimist: „Die Hoffnung stirbt zuletzt."

Wie viel Rationalität benötigen wir?

Rationalität ist eine zentrale, aber manchmal allzu starke Annahme:
Im Schach kenne ich zwar alle Regeln, aber nicht alle Konsequenzen;
mir fehlt die Rechenkapazität, ausreichend viele Züge vorauszudenken.
Für Menschen wie für Computer ist die Komplexität entscheidend!

Unsere Vorlesungsexperimente zeigen: Fehler treten tatsächlich auf,
selbst in einfachen Spielen wie einzeiligem Nim (vor dessen Lösung).
Teilnehmer verstehen vollständig die Regeln und wollen gewinnen ($\mathscr{R}_0$),
sie wissen auch, wie die Berechnung prinzipiell durchzuführen wäre. . .

Doch ohne genügend Rechenzeit, ohne Hilfsmittel wie Stift und Papier,
überschreitet der exponentielle Aufwand die verfügbare Kapazität:
Schon bei Spielbeginn in Position $x = 20$ scheint der Weg meist lang
und schwierig genug, um den einen oder anderen Fehler zu provozieren.

Befindet man sich in einer Verlustposition, so sind zwar in der *Theorie*
alle Züge gleich schlecht, aber in der *Praxis* eben nicht! Zum Beispiel
lohnen sich Züge, die dem Gegner die Analyse *erschweren*, damit kann
man ihn vielleicht *überraschen* und *verwirren* und so Fehler *provozieren*.

Wie viel Rationalität benötigen wir?

Nach der Lösung C1G sollte einzeiliges Nim fehlerfrei gespielt werden.
Vernichtet die mathematische Lösung jeglichen Spielspaß? Vielleicht.
Sie gewinnen dafür das intellektuelle Vergnügen, ein Problem zu lösen.
Zudem gibt es noch viele weitere Spiele, der Spielspaß dauert also an!

Dieselben Beobachtungen wiederholen sich noch dramatischer bei Nim.
Anfangs erkennen Sie wenig Struktur. Selbst wenn Sie sehen, dass Ihr Gegenüber über eine Gewinnstrategie verfügt, so können Sie diese schwerlich erraten. Alles ändert sich nach Boutons genialem Satz C1K.

Beobachten Sie dabei ihren Lernprozess vom *Whaaa?* bis zum *Aha!*
Diese Interaktion in der Vorlesung kostet enorm viel Zeit und Aufwand.
Viel schneller wäre, die Antworten nacheinander paradieren zu lassen, selbst wenn Sie sich zu diesem Zeitpunkt noch gar keine Fragen stellen.

Es fordert eine große Investition für alle Beteiligten, aber es lohnt sich:
Sie spüren die Probleme, Sie stellen Fragen, Sie äußern erste Ideen,
dafür will ich Ihnen Zeit lassen und genügend Raum für Diskussion.
Im Nachgang sollen Sie alles gründlich nacharbeiten und festigen.

Wie viel Rationalität benötigen wir?

Sie verstehen jetzt auch praktisch und anschaulich, warum ich Spiele in der Vorlesung (bzw. begleitend im Casino) tatsächlich spielen will: Sie sammeln dort Erfahrungen, die die rationale Theorie illustrieren, oft genug bestätigen, aber manchmal auch darüber hinausgehen!

Alle Informationen liegen offen, Spielregeln und Zielsetzung sind klar. Um das Verhalten vorherzusagen, benötigen wir weitere Annahmen, vereinfachend setzen wir hierzu unbegrenzte Rationalität voraus. Doch Menschen verhalten sich nicht immer rational, aus diversen Gründen.

Das gilt selbst für sehr einfache Spiele, wie unsere ersten Beispiele: Denken Sie an Ihre eigene Erfahrung und an ihre ersten Versuche! Die Spannung zwischen (rationaler) Theorie und (oft sehr begrenzt rationaler) Praxis ist immer wieder überraschend und faszinierend.

Die große Herausforderung der Spieltheorie ist es, diese Kluft zu verringern, und in günstigen Fällen die Lücke gänzlich zu schließen. Sie sollten beide Sichtweisen verstehen, beurteilen und nutzen lernen. Es gelingt sicher nicht immer, aber es lohnt sich danach zu streben.

Zermelos Satz: Schach ist determiniert.

Satz C3A: Zermelo 1913

Schach ist determiniert: Entweder besitzt Weiß eine Gewinnstrategie, oder Schwarz, oder jeder Spieler kann ein Unentschieden erzwingen.

Aufgabe: (0) Formalisieren Sie das Spiel Schach als einen Graphen. (1) Ist dieser Graph artinsch? (2) Beweisen Sie damit Zermelos Satz!

Lösung: Es gibt mehrere (äquivalente) Möglichkeiten, ein gegebenes Spiel wie Schach durch einen Graphen $G = (X, A, \sigma, \tau)$ zu codieren. In weiser Voraussicht notieren wir den gesamten Verlauf der Partie:

Wir konstruieren einen **Spielbaum** ausgehend vom Startzustand 0 durch Aufzeichnen aller bisher gespielten Halb/Züge $x = z_1 z_2 \ldots z_n$. Jeder nun legale Halb/Zug z ergibt eine Fortsetzung $y = z_1 z_2 \ldots z_n z$. (Was wir einen Zug nennen, heißt beim Schach traditionell Halbzug.)

(0) Die Zustandsmenge X besteht aus allen legalen Halb/Zugfolgen. Jeder Halb/Zug ist eine legale Fortsetzung, formal ausgeschrieben $A = X \setminus \{0\}$ mit $\sigma(z_1 \ldots z_n) = z_1 \ldots z_{n-1}$ und $\tau(z_1 \ldots z_n) = z_1 \ldots z_n$.

Zermelos Satz: Schach ist determiniert.

(1) Beim Schach gilt die **50-Züge-Regel**: Die Partie endet remis, wenn in den letzten 100 Halb/Zügen weder ein Stein geschlagen noch ein Bauer gezogen wurde (`de.wikipedia.org/wiki/50-Züge-Regel`).

Genauer: Die Partie endet nach 50 Zügen noch nicht automatisch, sondern das Remis muss von einem der Spieler reklamiert werden. Erst nach 75 Zügen wie in (1) beendet der Schiedsrichter die Partie.

Daraus folgt sofort: Der oben konstruierte Spielgraph G ist artinsch! (Die Stellung genügt dazu nicht, die Regeln benötigen den Verlauf.)

(2) Wir codieren Schach als den Spielgraphen $G = (X, A, \sigma, \tau)$.
Die terminalen Zustände bewerten wir mit $v : \partial X \to \{-1, 0, +1\}$
als Verlust, Remis, Gewinn für den gerade ziehenden Spieler,
und $-v$ für den Gegner. Dies ist demnach ein Nullsummenspiel.

Dank C1H existiert hierzu genau eine Gewinnfunktion $u : X \to \{\pm 1, 0\}$.
Im Falle $u(0) = +1$ besitzt Weiß eine Gewinnstrategie.
Im Falle $u(0) = -1$ besitzt Schwarz eine Gewinnstrategie.
Im Falle $u(0) = 0$ kann jeder Spieler ein Unentschieden erzwingen.

Algorithmische Komplexität: Wie schwierig ist Schach?

Zermelos grundlegendes Ergebnis zeigt, dass Schach determiniert ist; es definiert die Funktion u, sagt jedoch nichts über den Wert $u(0)$ aus. Viele haben versucht, diesen Wert zu bestimmen, und erhebliche Mühe investiert. Bis heute ist unbekannt, welcher der drei Fälle tatsächlich gilt.

Im Prinzip können wir die Gewinnfunktion u durch Rückwärtsinduktion berechnen. Der obige Beweis ist *konstruktiv*, das Vorgehen ist *effektiv*, doch die Berechnung nicht ausreichend *effizient*. Dieser Erfolg und die gleichzeitige Enttäuschung hat seit Zermelo viele Menschen fasziniert:

> *With chess it is possible, in principle, to play a perfect game or construct a machine to do so as follows: One considers in a given position all possible moves, then all moves for the opponent, etc., to the end of the game (in each variation). The end must occur, by the rules of the games after a finite number of moves (remembering the 50 move drawing rule). Each of these variations ends in win, loss or draw. By working backward from the end one can determine whether there is a forced win, the position is a draw or is lost.*
>
> Claude Shannon, *Programming a Computer for Playing Chess* (1950)

Algorithmische Komplexität: Wie schwierig ist Schach?

Shannon schätzte die Größe des Spielbaums G grob ab: Ein typisches Schachspiel dauert etwa 80 Halb/Züge, der ziehende Spieler hat etwa 30 mögliche Halb/Züge zur Auswahl, das ergibt grob $\sharp X \approx 30^{80} \approx 10^{120}$. Genaueres finden Sie unter `en.wikipedia.org/wiki/Shannon_number`.

Das ist eine astronomisch große Zahl, im wahrsten Sinne des Wortes: Die Anzahl aller Atome im beobachtbaren Universum wird auf etwa 10^{80} geschätzt. Shannons Zahl 10^{120} ist noch um 40 Zehnerpotenzen größer. Bei solch großen Zahlen versagt schnell unsere menschliche Intuition.

Die Berechnung von u mit brutaler Gewalt [*brute force*] ist demnach ganz offensichtlich ausgeschlossen. Nichtsdestotrotz ist es denkbar, durch geschickte, tiefsinnige Optimierung eine schnelle Berechnung zu finden und Zermelos Ergebnis C3A schließlich explizit zu berechnen.

Dazu betone ich einen naiv-optimistischen Vergleich: Auch Nim mit 40 Zeilen zu jeweils 1000 Objekten hat etwa $1000^{40} \approx 10^{120}$ Spielzustände. Dennoch können wir dieses Spiel effizient lösen, denn es hat Struktur (als Summe), und wir haben Werkzeuge (Satz C1O). Mathematik hilft!

Jede Menge ist ein kombinatorisches Spiel...

 Jede Zermelo-Fraenkel-Menge M ist ein kombinatorisches Spiel! Die Spieler, Alice und Bob, ziehen abwechselnd durch Wahl eines Elements $N \in M$ als neuen Zustand. Wer nicht ziehen kann, verliert.

Beispiele: Die leere Menge $\emptyset$ ist terminal, also eine Verlustposition. Somit ist $\{\emptyset\}$ eine Gewinnposition, ebenso jede Menge M mit $\emptyset \in G$.

Nach John von Neumann definieren wir die natürlichen Zahlen durch $0 = \emptyset$, $1 = \{0\}$, $2 = \{0, 1\}$, allgemein $n + 1 = \{0, 1, \ldots, n\}$. Das ist Nim!

Satz C3B: Jede Menge definiert einen Graphen.

Jede Menge M definiert einen artinschen Graphen $G = (X, A, \sigma, \tau)$ mit Wurzel M und den Kanten $x \to y$ genau dann wenn $x \ni y$.

Beweis: Warum ist G artinsch? Hier retten uns die ZF-Axiome der Mengenlehre: Sie verbieten unendlich lange Ketten $x_0 \ni x_1 \ni x_2 \ni \ldots$.

. . . und umgekehrt!

Satz C3C: Jeder Graph definiert eine Menge.

Sei $G = (X, A, \sigma, \tau)$ ein artinscher Graph. Rekursiv können wir jedem Zustand $x \in X$ eine Menge $\zeta(x) = \{\, \zeta(y) \mid x \to y \,\}$ zuordnen.

Beweis: Auch dies verdanken wir der Rückwärtsinduktion C1H, leider nicht direkt dem Satz, doch unmittelbar seinem Beweis.

Das Graphenspiel G im Zustand x ist äquivalent zum Mengenspiel $\zeta(x)$:
(1) Jeder Zug $x \to y$ entspricht dem zugehörigen Zug $\zeta(x) \ni \zeta(y)$.
(2) Umgekehrt, zu jedem $\zeta(x) \ni z$ existiert $x \to y$ mit $\zeta(y) = z$.

Übung: Betrachten Sie das Spiel Nim mit (kleinem) Startzustand $x \in \mathbb{N}^{(\mathbb{N})}$ und codieren Sie dies durch die zugehörige Menge $\zeta(x)$.

Übung: Sind die oben erklärten Zuordnungen $\xi : M \mapsto (G, M)$ und $\zeta : (G, x) \mapsto \zeta(x)$ zwischen Mengen und Graphen zueinander invers?

Von Mengen und Spielen. . .

Mit **Mengen** können wir alle mathematischen Objekte formulieren, von den Zahlbereichen $\mathbb{N} \subset \mathbb{Z} \subset \mathbb{Q} \subset \mathbb{R} \subset \mathbb{C}$ über Funktionen $f : X \to Y$ hin zu weiteren Strukturen wie zum Beispiel Skalarprodukte, Normen, Metriken, Topologien, σ–Algebren, Wahrscheinlichkeits/Maße, usw.

Der Phantasie sind dabei keine Grenzen gesetzt. Erstaunlicherweise lässt sich all dies dank Mengen präzise formulieren und bequem nutzen. Die Mengenlehre dient so der gesamten Mathematik als Fundament, auch nach über einhundert Jahren erfüllt sie treu ihren Zweck.

Es ist daher zunächst überhaupt nicht verwunderlich, dass wir auch Spiele im bewährten Rahmen der Mengenlehre formulieren können. Genau das haben wir mit Graphen und ggf. weiteren Daten erreicht. Die Sprache der Mengen bewährt sich auch hier erneut wunderbar.

Durchaus überraschend ist jedoch, wie nahe sich die Grundstrukturen stehen: Mengen auf der einen Seite, artinsche Graphen auf der anderen. Hier haben wir eine direkte, einfache Übersetzung in beide Richtungen, die alle wesentlichen Aspekte bewahrt, wie oben skizziert.

. . . zur mathematischen Grundlagenforschung

John H. Conway perfektionierte die enge Verbindung von Spielen und Mengen in seinem berühmten Buch *On Numbers and Games* (1976).

Als sympathisch-spielerische Einführung schrieb Donald E. Knuth 1974 *Surreal Numbers*, ein Dialog zwischen Alice und Bob über die ersten 20 Seiten von *ONAG*. Er richtet sich an Studienanfänger:innen, mit dem Wunsch, ihnen spielerisch mathematische Forschung nahezubringen.

Of course, I wrote this mostly for fun, [. . .] but I must admit that I also had a serious purpose in the back of my mind. Namely, I wanted to provide some material that would help to overcome one of the most serious shortcomings in our present educational system, the lack of training for research work. [. . .] Conway's recent approach to numbers struck me as the perfect vehicle for illustrating the important aspects of mathematical explorations, because it is a rich theory that is almost self-contained, yet with close ties to both algebra and analysis, and because it is still largely unexplored.

Donald E. Knuth, *Surreal Numbers* (1974)

Als schönes Video von Prof. Edmund Weitz, `youtu.be/Q9gGVUiMo04`.

Die Frucht der Erkenntnis. . . von wahr und falsch

Adam und Eva, Gemälde von Tizian um 1550, Museo del Prado

Die Frucht der Erkenntnis. . . von wahr und falsch

Dieses Gemälde von Tizian (Tiziano Vecellio, 1488–1576) zeigt den Sündenfall des Menschen. Die Paradieserzählung im Buch Genesis der Bibel spricht vom **Baum der Erkenntnis von Gut und Böse** (Gen 2,9). Ich interpretiere dies hier freizügig als Erkenntnis von **wahr und falsch**.

Im Folgenden spielen ausnahmsweise mal nicht Alice und Bob, sondern zwecks Diversität und Abwechslung nun Eva und Adam, genauer ∃va und ∀dam, denn es geht uns hier um Quantoren. Diese beiden hauchen bewährter Mathematik neues Leben ein!

Das hier vorgestellte Spielprinzip ist eine Interpretation der Logik, und inzwischen gibt es hierzu eine umfangreiche mathematische Literatur, speziell in der (infinitären) Logik und der (deskriptiven) Mengenlehre, siehe `en.wikipedia.org/wiki/Axiom_of_determinacy`.

Erste solche Spiele kennt jede:r Studierende seit dem ersten Semester der Analysis. Wir gehen nun einen mutigen Schritt weiter und wollen diese Quantorenspiele tatsächlich *spielen*, live und interaktiv im Hörsaal. Wie immer machen wir uns auf einige Überraschungen gefasst!

Quantoren-Spiele: ∃va gegen ∀dam

Jede Aussage mit All- und Existenz-Quantoren ist ein Spiel!

∃va spielt die **E**xistenzquantoren und möchte die Behauptung **e**rfüllen.
∀dam spielt die **A**llquantoren und möchte die Behauptung **a**nfechten.

Nulltes Beispiel: Zum Aufwärmen betrachten wir

$$\text{(1)} \quad \forall q \in \mathbb{Q}_{>0}\ \exists z \in \mathbb{Z} \ : \ 2^z > q$$

$$\text{(2)} \quad \forall q \in \mathbb{Q}_{>0}\ \exists z \in \mathbb{Z} \ : \ 2^{z-1} \leq q < 2^z$$

Sei $h_n = \sum_{k=1}^{n} 1/k$ die harmonische Reihe, mit $h_0 = 0$ und $h_{-1} = -\infty$.

$$\text{(3)} \quad \forall q \in \mathbb{Q}\ \exists n \in \mathbb{N} \ : \ h_n \geq q$$

$$\text{(4)} \quad \forall q \in \mathbb{Q}\ \exists n \in \mathbb{N} \ : \ h_{n-1} < q \leq h_n$$

Je nach Formulierung des Spiels ist die Rechenlast / Beweislast
für ∀dam und ∃va und die Spielleitung (!) spürbar verschieden.

Quantoren-Spiele: ∃va gegen ∀dam

Unsere ersten Quantorenspiele sind (theoretisch!) sehr simpel und daher genau richtig um sich (praktisch!) einzugewöhnen.

Sobald alle die Spielregeln verstehen und auch die Schwierigkeiten spüren, können wir schrittweise zu interessanteren Spielen über.

Eventuell müssen weitere Regeln diskutiert und abgestimmt werden. Es ist jedenfalls gut, sich gut einzustimmen, bevor es ernst wird.

Sei $\mathbb{P} = \{p_0 < p_1 < p_2 < \ldots\}$ die Menge der Primzahlen und $p_{-1} = 0$.

(5) $\forall n \in \mathbb{N}\ \exists p \in \mathbb{P}\ :\ p > n$

(6) $\forall n \in \mathbb{N}\ \exists k \in \mathbb{N}\ :\ p_{k-1} \leq n < p_k$

Wir betrachten die Reihe $x_n = \sum_{k=1}^{n} 1/k^2$.

(7) $\exists q \in \mathbb{Q}\ \forall \varepsilon \in \mathbb{Q}_{>0}\ \exists m \in \mathbb{N}\ \forall n \in \mathbb{N}_{\geq m}\ :\ q - \varepsilon < x_n \leq q$

(8) $\forall \varepsilon \in \mathbb{Q}_{>0}\ \exists q \in \mathbb{Q}\ \exists m \in \mathbb{N}\ \forall n \in \mathbb{N}_{\geq m}\ :\ q - \varepsilon < x_n \leq q$

Quantoren-Spiele: ∃va gegen ∀dam

Jede Aussage mit All- und Existenz-Quantoren ist ein Spiel!

∃va spielt die **E**xistenzquantoren und möchte die Behauptung **e**rfüllen.
∀dam spielt die **A**llquantoren und möchte die Behauptung **a**nfechten.

Erstes Beispiel: Als Spieldaten betrachten wir die Folge

$$(1) \quad x_n := \sum_{k=0}^{n} 2^{-k}.$$

Zwei mögliche Spielregeln sind:

$$(K) \quad \exists a \in \mathbb{Q} \;\; \forall \varepsilon \in \mathbb{Q}_{>0} \;\; \exists m \in \mathbb{N} \;\; \forall n \in \mathbb{N}_{\geq m} \;\; : \;\; |x_n - a| < \varepsilon$$

$$(C) \quad \forall \varepsilon \in \mathbb{Q}_{>0} \;\; \exists a \in \mathbb{Q} \;\; \exists m \in \mathbb{N} \;\; \forall n \in \mathbb{N}_{\geq m} \;\; : \;\; |x_n - a| < \varepsilon$$

Weitere Spieldaten, für noch mehr Spielspaß:

$$(2) \quad x_n := \sum_{k=0}^{n} (-2)^k$$

$$(3) \quad x_n := \sum_{k=0}^{n} 1/k!$$

Quantoren-Spiele: ∃va gegen ∀dam

Die „Epsilontik“ ist eine geniale Errungenschaft der Analysis. Sie hat im 19. Jh. der intuitiven Vorstellung des infinitesimal Kleinen eine tragfähige Grundlage geschaffen und ist seither überall phantastisch erfolgreich. Es lohnt, dieses Juwel menschlicher Erkenntnis zu verstehen.

Egal ob in der Mathematik oder der Informatik, den Natur-, Ingenieur- oder Wirtschaftswissenschaften, die Analysis wird überall gerne genutzt, doch ihr Studium verlangt von Anfänger:innen eine große Investition. Vielleicht hilft es, Quantorenfolgen als Spiele zu begreifen.

Meist spielt leider nur eine Person, in der Vorlesung, beim Rechnen, als Hausaufgabe, in der Klausur, etc. Wir spielen dies im Casino Royal als „Team ∃va“ gegen „Team ∀dam“. Selbst wer theoretisch alles versteht, erlebt interaktiv erfahrungsgemäß neue Aspekte und Aha-Momente.

Was winkt als Bonus? Das Spiel spornt an. Konkrete Zahlen üben das Rechnen. Der Wunsch nach Systematisierung vertieft das Verständnis. Die Schwierigkeiten werden greifbar: schrittweise Lernen und Üben, formales Protokol, rechnerisch-algorithmische Komplexität, usw.

Quantoren-Spiele: ∃va gegen ∀dam

Jede Aussage mit All- und Existenz-Quantoren ist ein Spiel!

∃va spielt die **E**xistenzquantoren und möchte die Behauptung **e**rfüllen.
∀dam spielt die **A**llquantoren und möchte die Behauptung **a**nfechten.

Zweites Beispiel: Als Spieldaten betrachten wir die Funktion

$$f : \mathbb{Q} \to \mathbb{Q} : x \mapsto \operatorname{sign}(x^2 - 2) \qquad \text{auf } X = \mathbb{Q}.$$

Zwei mögliche Spielregeln sind:

$$\forall a \in X \;\; \forall \varepsilon \in \mathbb{Q}_{>0} \;\; \exists \delta \in \mathbb{Q}_{>0} \;\; \forall x \in X \;\; : \; |a - x| < \delta \Rightarrow |f(a) - f(x)| < \varepsilon$$

$$\forall \varepsilon \in \mathbb{Q}_{>0} \;\; \exists \delta \in \mathbb{Q}_{>0} \;\; \forall a \in X \;\; \forall x \in X \;\; : \; |a - x| < \delta \Rightarrow |f(a) - f(x)| < \varepsilon$$

Weitere Spieldaten, für noch mehr Spielspaß:

$$g : \mathbb{R} \to \mathbb{R} : x \mapsto \operatorname{sign}(x^2 - 2) \qquad \text{auf } X = \mathbb{R},$$

$$h : \mathbb{R} \to \mathbb{R} : x \mapsto \textstyle\sum_{k=1}^{\infty} \operatorname{dist}(x, \frac{1}{k!}\mathbb{Z}) \qquad \text{auf } X = \mathbb{R}.$$

Quantoren-Spiele: ∃va gegen ∀dam

Eine bewährte mathematische Weisheit besagt: Wenn wir eine Aussage beweisen wollen, sollten wir zugleich versuchen, unsere Argumente zu widerlegen. Das erzieht uns, ja zwingt uns, zu (selbst)kritischer Prüfung. Wir verteilen nun die Rollen explizit und erleben manche Überraschung!

Was ist nach einer Episode bewiesen? Vermutlich noch nahezu nichts. Nach vielen Episoden wächst die Überzeugung. Gewinnt immer ∃va? Gewinnt immer ∀dam? Können wir eine Gewinnstrategie formulieren? Jede vollständig ausgearbeitete Gewinnstrategie ist ein formaler Beweis!

Die Durchführung des Spiels kann auch als **Null-Wissen-Beweis** [*zero knowledge proof*] dienen: ∃va kann ∀dam überzeugen, dass sie einen Beweis hat, ohne den Beweis preiszugeben; der Nachweis geschieht durch wiederholtes Spiel, bis ∀dam überzeugt ist (oder umgekehrt).

Ich habe oben bewusst auf Stichworte wie „konvergente Folge“ oder „Cauchy–Folge“, „stetige Funktion“ oder „gleichmäßig stetig“ verzichtet. Das ist Teil des Entdeckens, Ausprobierens, Spielens, . . . Wer sie kennt, genießt gewisse Vorteile, und verspürt doch erneut Anfängerfreuden.