Seminarvortrag

Interaktive Beweise und Zero Knowledge

Lars Münzberg, Dirk Hain

Inhalt:

0. Definition der Zielstellung: Was ist Zero Knowledge?

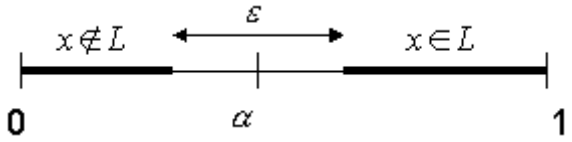
Informal bedeutet Zero Knowledge, daß eine Person (oder Maschine) eine andere davon überzeugen kann, das ihr ein Geheimnis bekannt ist, ohne dieses Geheimnis tatsächlich zu verraten. Diese Fähigkeit hat nicht nur kryptographisch einen hohen Reiz, sondern ist auch für die Praxis äußerst relevant als zentrale Grundlage von Authentikationssystemen. Ziel der folgenden Darlegung ist es, die theoretischen Grundlagen von Zero Knowledge Beweisen zu erklären und somit zu zeigen, dass dieses Verfahren tatsächlich so funktioniert und einsetzbar ist.
Dazu müssen allerdings vorher einige für diese Verfahren unerlässlichen Begriffe definiert werden, wie zum Beispiel die „Interaktiven Beweise“.

0.1 Definition Probabilistischer Algorithmus

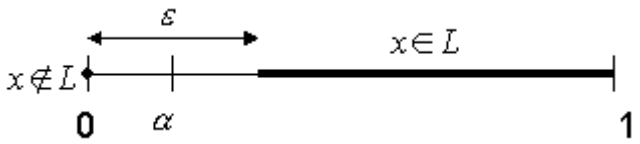
Die Eigenschaft der Probabilität lässt einen Algorithmus seine Entscheidungen von einem Zufallsereignis abhängig machen. Da Algorithmen durch Turingmaschinen simuliert werden können, wird die Probabilität durch n - viele Folgekonfigurationen zu einer Konfiguration, die alle mit gleicher Wahrscheinlichkeit angenommen werden können, nachempfunden. Dies hat aber auch zur Folge, dass die Entscheidung, die die Maschine am Ende der Abarbeitung fällt (akzeptieren oder verwerfen) sowie die Laufzeit der Maschine auch vom Zufall abhängig werden.
Bei dieser Art von Algorithmen (bzw. Maschinen) heißt „akzeptieren“, dass die Wahrscheinlichkeit des letztendlichen Akzeptierens größer als ½ ist. Formuliert man diese Bedingung allgemeiner, so erhält man die Komplexitätsklasse PP.

0.2 Die Komplexitätsklassen PP, BPP und RP

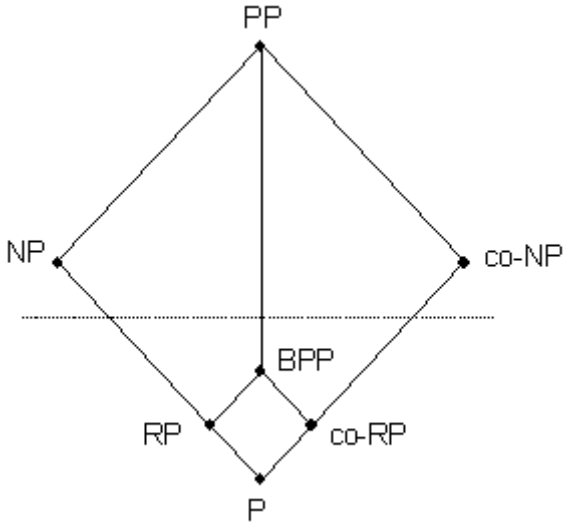
Die Komplexitätsklasse **PP** (probabilistic polynomial time) ist die Menge aller Sprachen, deren Wörter von einer probabislistischen polynomial zeitbeschränkten Turingmaschine mit einer Wahrscheinlichkeit größergleich einem (beliebigen aber festen) Schwellenwert $\alpha \in [0,1]$ akzeptiert werden.
Man kann diese Bedingung noch erweitern, indem man die „Lücke“, die zwischen „akzeptieren“ und „verwerfen“ entsteht auf eine bestimmte Größe festlegt. Zum Beispiel könnte man die Sprache so definieren, dass die Maschine bei $x \in L$ mit einer Wahrscheinlichkeit von $\geq \alpha + \varepsilon/2$ akzeptiert und bei $x \notin L$ mit $\leq \alpha - \varepsilon/2$ Wahrscheinlichkeit verwirft. Auf diese Weise entsteht eine Lücke zwischen Wörtern $x \in L$ und Wörtern $x \notin L$.
Die Komplexitätsklasse **BPP** ist die Menge aller Sprachen, für die es eine probabislistische polynomial zeitbeschränkte Turingmaschine, einen Schwellenwert α und eine Wahrscheinlichkeitslücke ε gibt, so dass es kein Wort in den jeweiligen Sprachen gibt, für dass die Akzeptanzwahrscheinlichkeit im Intervall $[\alpha - \varepsilon/2, \alpha + \varepsilon/2]$ liegt.



Aus den Definitionen und der Abbildung kann man schließen, dass P $\subseteq$ BPP $\subseteq$ PP gilt.
Nun kann man noch Einschränkungen zur Lage der „Lücke“ im Intervall [0,1] machen und falls eine Sprache die Eigenschaft erfüllt, dass $\varepsilon = 2\alpha$ (siehe Skizze), so befindet sie sich in der Komplexitätsklasse **RP**.



Abschließend kann man den Zusammenhang der Inklusionsbeziehungen noch mit folgender grafischer Darstellung veranschaulichen:



1. Interaktive Beweise

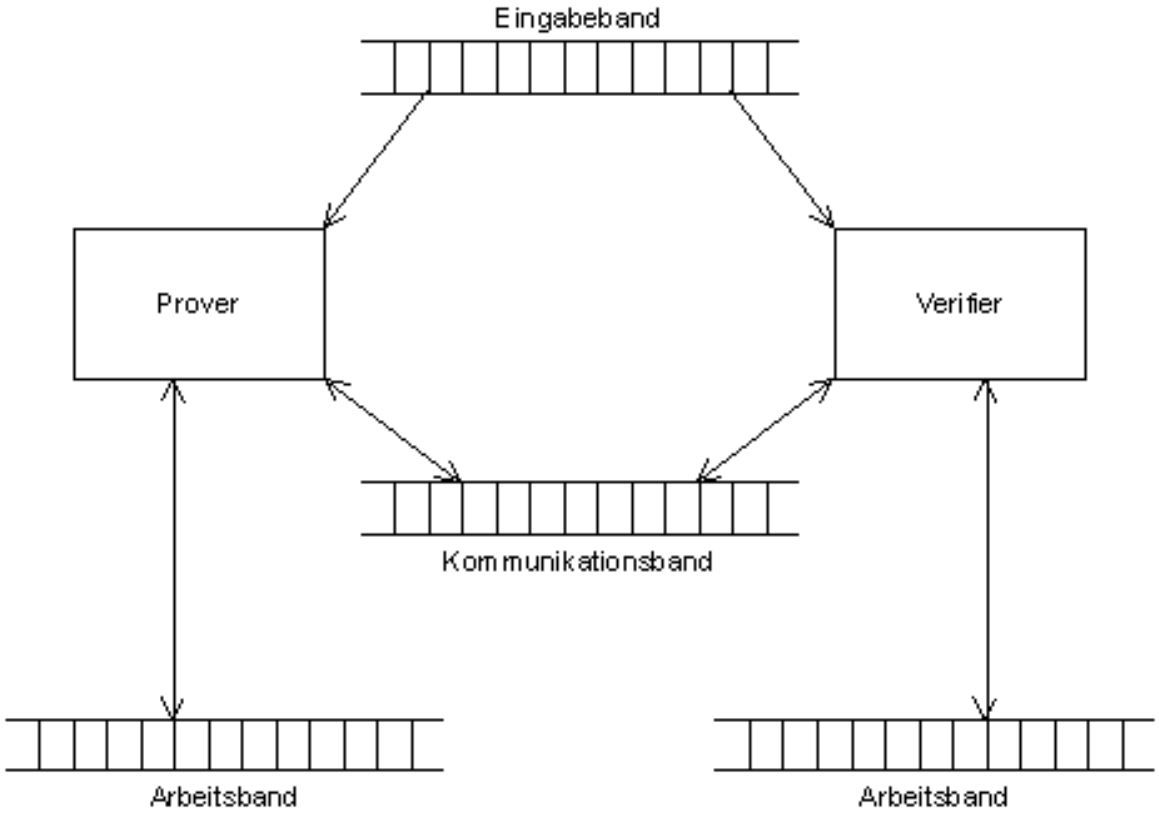
Allgemein verstehen wir unter einem (mathematischen) Beweis eine Argumentationskette, die logisch lückenlos eine neue Aussage aus bereits bewiesenen Aussagen bzw. aus unbeweisbaren Grundaussagen (Axiomen) herleitet. Die Interaktivität in diesen erst in den späten 80er Jahren definierten Beweisen besteht darin, dass der Beweis wie der Dialog zwischen einem Beweisenden (Prover) und einem Unwissenden (Verifier) aufgebaut ist und sich somit in Rede und Gegenrede teilen lässt.

1.1 NP Definition

Die Klasse NP kann als Menge aller Sprachen aufgefasst werden, für die ein nichtdeterministischer endlicher Automat einen höchstens polynomial-langen Beweis „raten“ kann, welcher wiederum in höchstens polynomialer Zeit verifiziert werden kann.

1.2 Kommunikationsproblem

Betrachtet man die in 1.1 aufgeführte Definition, so wird deutlich, dass die Kommunikation zwischen den Prover und Verifier einseitig verläuft, da der Prover seinen Beweis an den Verifier übermittelt und dieser wiederum die Korrektheit überprüft. Von dieser einseitigen Kommunikation zwischen Prover und Verifier, wollen wir nun abstrahieren und das Kommunikationsproblem zwischen beiden Teilnehmern mit Turingmaschinen modellieren. Wir werden zwei Turinmaschinen als Kommunikationspartner auffassen, eben den Prover und den Verifier. Hierbei wird dem Verifier eine polynomiale Zeitbeschränkung auferlegt, dem Prover wird dagegen keine Komplexitätsschranke zugewiesen. Dadurch wird der Existenzquantor in der Definition von NP widergegeben, also das Finden eines Beweises. Mit diesem Beweis wird nachgewiesen, dass die Eingabe x Element einer NP-Sprache A ist.



Beide Maschinen kommunizieren über ein gemeinsames Kommunikationsband, ferner haben beide Zugang zur Eingabe, die gewissermaßen die zu beweisende Aussage x darstellt. Außerdem besitzen beide Turingmaschinen "private" Arbeitsbänder für interne Rechnungen. Die Berechnung erfolgt in Runden, nur eine der beiden Turingmaschinen ist jeweils in einer Runde aktiv. Jede Runde beginnt mit dem Lesen der Information auf dem Kommunikationsband bzw. dem Eingabeband und endet nach eventuellen privaten Berechnungen, die auf dem Arbeitsband getätigt werden, mit dem Schreiben einer neuen höchstens polynomial-langen Information auf das Kommunikationsband. Am Ende akzeptiert oder verwirft der Verifier. Hierbei ist die Anzahl der Runden, sowie die jeweilige Eingabelänge polynomial beschränkt.

Beide Teilnehmer haben die Möglichkeit die bisherige Kommunikationsinformationen auf dem Kommunikationsband mitzuprotokollieren, somit kann die neu berechnete Information für das Kommunikationsband als Funktion der Eingabe, sowie aller bisherigen Kommunikationen aufgefasst werden. Dem Verifier muss diese Berechnung in polynomialer Zeit gelingen.
Diese interaktive Art etwas zu verifizieren, entspringt der Antike. So wurde ein Beweis in Form eines Dialoges zwischen Beweisführer und einer den Beweis anzweifelnden Person niedergeschrieben. 1986 haben Goldwasser, Micali und Rackoff schließlich den Begriff des interaktiven Beweisens eingeführt.

1.3 Die gebildete Sprachmenge klassifizieren

Die Menge der durch so ein interaktives Beweissystem dargestellten Sprachen bildet die Klasse NP. Es muss ein Verifier Algorithmus V geben, so das es für alle $x \in A$ eine Prover-Strategie gibt, mit dem der Verifier bei Eingabe x und Kommunikation mit diesem Prover schließlich akzeptiert bzw. bei $x \notin A$ verwirft. (Definition 1) Sei A durch ein solches Beweissystem berechenbar. Dann liegt A in NP, denn man kann, bei Eingabe von x, die Berechnung des Verifiers simulieren und die potentiellen Antworten des Provers nichtdeterministisch raten. x liegt genau dann in A, wenn diese nichtdeterministische Maschine akzeptiert.

Wenn sich die Turingmaschinen probabilistisch verhalten, sie dürfen also den Fortgang ihrer Berechnung vom Ausgang eines Zufallsexperiments abhängig machen, so stellt sich die Frage, wie sich die Sachlage nun verhält?
Durch die Zufallssteuerung wird das Kriterium zur Überprüfung, ob x in A liegt, abgeschwächt. Die formale Definition der erzeugten Klasse IP (Interactive Proof) ergibt sich aus der vorhergehenden Definition 1 und der Aussage: falls es eine probabilistische, polynomial zeitbeschränkte Turingmaschine V (den Verifier) gibt, so dass für alle x gilt ([1]):

$x \in A \Rightarrow \exists \text{Prover P: } Pr[(P,V)(x)=1] > \frac{2}{3}$
$x \notin A \Rightarrow \forall \text{Prover P: } Pr[(P,V)(x)=1] < \frac{1}{3}$

Die Konstanten ⅔ und ⅓ erscheinen willkürlich, aber durch die Wahrscheinlichkeitsverstärkung in unserem Modell kommt es auf keinen genauen Wert an, solange die beiden Konstante die Form ½ - ε und ½ + ε erfüllen. Die Wahrscheinlichkeitsverstärkung kommt durch die Anzahl der Ausführungen des Protokolls zustande. Hierbei überlagern sich die jeweiligen Wahrscheinlichkeiten von Ausführung zu Ausführung, es kommt zur Mehrheitsentscheidung!. Das Protokoll wird j-mal ausgeführt, dann wird die Fehlerwahrscheinlichkeit in der Anzahl der Wiederholungen exponentiell klein.

Wie wir bereits gesehen haben, hat jede NP-Sprache einen interaktiven Beweis. Sogar einen trivialen Beweis, wo die Kommunikation nur vom Prover zum Verifier gerichtet ist. Hieraus folgt IP umfasst NP.

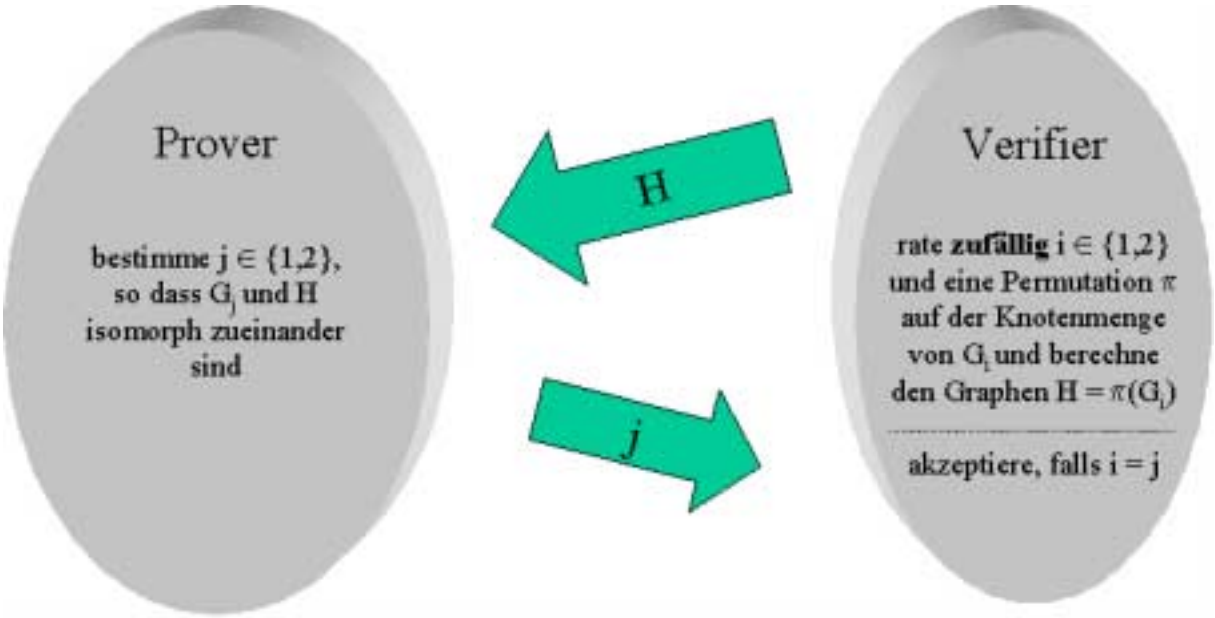
---

[1] Pr[(P,V)(x)=1] ist die Wahrscheinlichkeit (Propability), das bei gegebener Prover-Maschine und gegebener Verifier-Maschine die Eingabe x akzeptiert wird

Es gilt zudem : IP $\subseteq$ PSPACE. Man stelle sich dazu die Kommunikation zwischen Prover und Verifier als einen binären Baum vor, in dem die Knoten Konfigurationen der beiden Maschinen darstellen. Dieser Baum hat höchstens polynomiale Tiefe, da eine Ausprägung der Kommunikation höchstens polynomial viele Schritte hat und ein Knoten höchstens polynomial viel Speicherplatz benötigt (Es wird das polynomial in der Eingabelänge beschränkte Kommunikationsband abgebildet.), ist die Länge eines Astes ebenfalls höchstens polynomial lang. Die Auswertung des Baumes verläuft nun nach der Tiefe-Zuerst-Methode, wodurch höchstens jeweils ein Ast des Baumes gespeichert sein muss. Damit benötigt eine Maschine zum Auswerten des Baumes auch höchstens polynomial viel Platz jedoch exponential viel Zeit, was aber keine Rollen spielt.

1.4 Wie groß ist die Klasse IP und um wie viel wird NP erweitert?

Dieses Problem wollen wir anhand eines Beispieles beleuchten. Wir wählen hierzu die IP-Sprache "Komplement des Graphenisomorphieproblems" von der nicht bekannt ist, ob sie in NP liegt.
Gegeben sind zwei Graphen G_1 und G_2, beweise interaktiv, daß diese nicht isomorph (bijektive Funktion die Knoten von einem Graphen auf die Knoten eines anderen Graphen abbildet) sind, für diese Aufgabe kann man ein Beweis-Protokoll entwerfen:

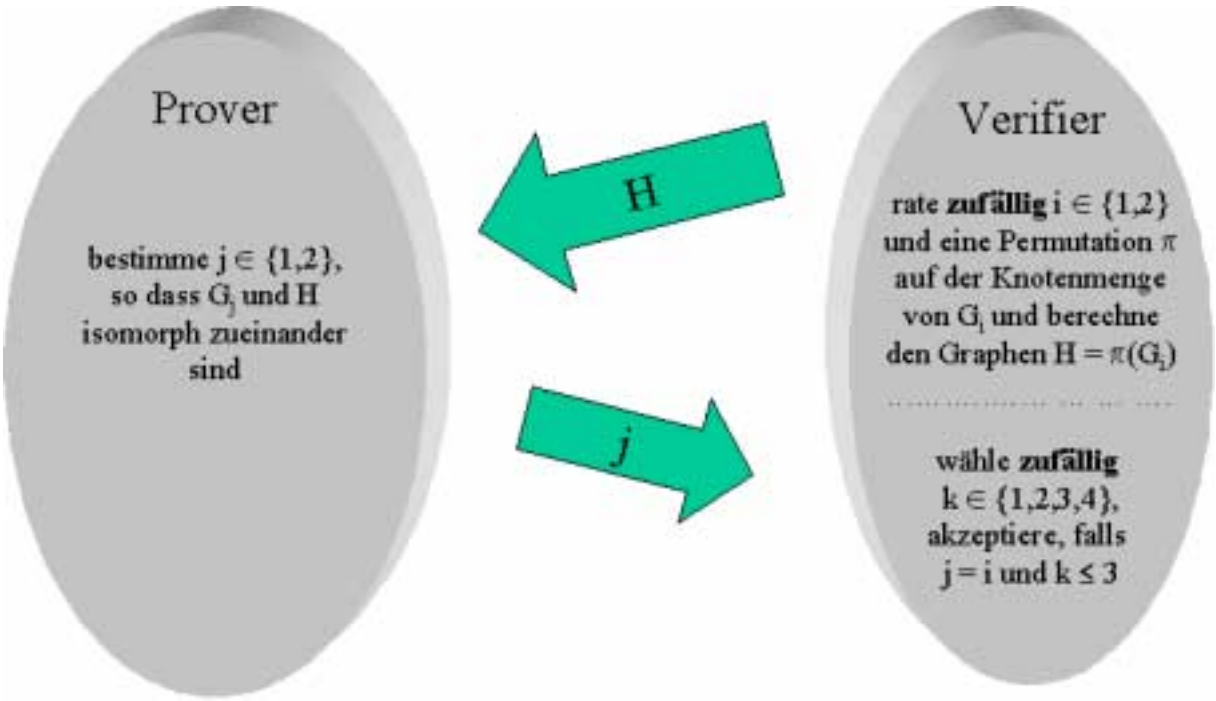


1. Fall G_1 und G_2 nicht isomorph:

   Ein geeigneter Prover-Algorithmus kann immer mit dem richtigen j antworten, da der Prover stets den Graphen G_j, der zu H isomorph ist, findet.
   $\Rightarrow \exists$Prover P: Pr[(P,V)(G_1,G_2) = 1] = 1

2. Fall G_1 und G_2 isomorph:

   Ein Prover hat jetzt höchstens die Chance von ½ für die „richtige“ Antwort. Da beide Graphen G_j zu H isomorph sind, somit muss sich der Prover entscheiden welches j an den Verifier kommuniziert wird.
   $\Rightarrow \forall$Prover P: $Pr[(P,V)(G_1,G_2) = 1] \leq ½$
   Die obige IP-Definition ist aber noch nicht exakt erfüllt, denn die Wahrscheinlichkeit ist nicht kleiner als ⅓. Wir modifizieren nun die Akzeptanzbedingung der Verifiers.



Es gilt also "Komplement des Graphenisomorphieproblems" ist Element in IP. Zu beobachten ist, dass obiges Protokoll mit einer konstanten Anzahl von Runden auskommt (nämlich 2, symbolisch IP(2)). Wenn wir die Möglichkeit von polynomial in der Eingabelänge vielen Runden ausschöpfen, so erreichen wir die Klasse PSPACE.

2. IP = PSPACE

Um diese überraschende Entdeckung zu beweisen bediente sich Shamir der PSPACE – vollständigen Sprache QBF.

2.1 Die PSPACE – vollständige Sprache QBF

Die Sprache QBF ist die Sprache aller quantifizierten boolschen Formeln ohne freie Variablen (d.h. alle in den Formeln vorkommenden Variablen sind quantifiziert), die gültig sind. Die Sprache ist PSPACE – vollständig, das heißt, ein Automat benötigt polynomial viel Speicherplatz, um ein Wort der Sprache zu verifizieren.
Ein Beispiel für ein Element der Sprache ist :

$$\forall a \forall b \forall c(a \vee !b \vee c \vee \exists d((a \wedge d)(s \wedge d)))$$

Die Idee des Beweises ist, einen interaktiven Beweis zur Verifikation der Sprache anzugeben, so dass die Inklusion PSPACE $\subseteq$ IP gezeigt wird. Die Inklusion IP $\subseteq$ PSPACE wurde ja bereits gezeigt. Damit gilt dann die Gleichheit beider Mengen. Um dies jedoch zu beweisen sind für QBF Einschränkungen nötig, die dem Beweis jedoch nicht wiedersprechen.

2.2 Erste Einschränkung von QBF (QBF')

Es sind im Folgenden nur solche Formeln aus QBF zugelassen, in denen sich Negationen nur auf Variablen beziehen. Seien diesen definiert als QBF'.
Die Sprache QBF' ist in Polynomialzeit auf die Sprache QBF reduzierbar (QBF' $\leq$ QBF), da mittels der de Morganschen Regeln die Negationen nach innen gezogen werden können.

$$(!\forall x\ F) = \exists x\ !F$$
$$!\exists x\ F = \forall x\ !F$$
$$\dots$$

Es liegt nun am Prover dem Verifier zu beweisen, daß F $\in$ QBF' gültig ist, das heißt WW (F) = wahr bzw. falls F ungültig ist sollte der Verifier unabhängig davon, was der Prover überträgt verwerfen.

2.3 Die arithmetische Auswertung von boolschen Funktionen

Um die Möglichkeiten des Provers, den Verifier zu betrügen einzuschränken, wird der Lösungsraum {wahr, falsch} erweitert. Dies geschieht durch eine **arithmetische Interpretation** der boolschen Formeln (**arith(F)**).
Dazu wird folgende „Abbildung" definiert:

1. Alle Variablen sind Element der ganzen Zahlen.
2. UND wird als Multiplikation interpretiert.
3. ODER wird als Addition interpretiert.
4. Negierte Variablen werden zu 1-x (x – Wert der Variable) ausgewertet.
5. Formeln der Form $\forall x$ F werden ausgewertet, indem alle freien Variablen x in F einmal mit 0 und einmal mit 1 ersetzt werden. Die sich so ergebenden Zahlenwerte aus der Menge der ganzen Zahlen werden multipliziert.

$\forall x\ F \rightarrow$ a_1 = arith(F), wobei x = 1 $\rightarrow \forall x\ F = a_0 * a_1$
a_0 = arith(F), wobei x = 0

Bei Existenzquantoren wird entsprechend addiert.
$\exists x\ F \rightarrow$ a_1 = arith(F), wobei x = 1 $\rightarrow \exists x\ F = a_0 + a_1$
a_0 = arith(F), wobei x = 0

Es gilt nun bool(F) = wahr $\Rightarrow$ arith(F) > 0
bool(F) = falsch $\Rightarrow$ arith(F) = 0.

Beweis (Induktion über den Aufbau von boolschen Formeln):

- Bestehen die Formeln nur aus einzelnen Variablen, so werden diese auch nur durch 0 oder 1 ersetzt.
- Bei einer negierten Variable wird durch (1-x) aus einer 1 eine 0 oder umgekehrt. (Es können wie oben erwähnt nur einzelne Variablen negiert sein.)
- In Formeln der Form $F \wedge G$ wird UND mit der Multiplikation korrekt simuliert, analog gilt dies für die Simulation von ODER mit Addition.

*/AND	F	G	AW	WW
	0	0	0	0
	0	1	0	0
	1	1	1	1
	1	0	0	0

+/OR	F	G	AW	WW
	0	0	0	0
	0	1	1	1
	1	1	2	1
	1	0	1	1

- Bei quantifizierten Formeln der Form $\forall x\ F$ ($\exists x\ F$) wird durch das angegebene Verfahren des Einsetzens von 0 und 1 und anschließendes multiplizieren (addieren) auch der Allquantor (Existenzquantor) richtig repräsentiert.

$\forall x\ F$: F wird nur dann wahr, falls es für x = 0 und x = 1 wahr ist
$\exists x\ F$: F wird wahr wenn es entweder für x = 0 oder x = 1 wahr ist

Durch die oben aufgeführte Änderung der Interpretation der Formeln wird das Protokoll dahingehend geändert, dass der Prover dem Verifier keinen Wahrheitswert zurückgibt, sondern eine Zahl ganze Zahl a, $a \geq 0$ überträgt und ihm anschließend beweist, dass a = arith(F).

2.4 Die Modulo-Operation zur Einschränkung auf PSPACE

Jetzt tritt erneut ein Problem des Protokolls auf. a = arith(F), $a \in Z$ wird vom Prover (unbeschränkte Rechenressourcen) berechnet, muß aber auch übertragen werden. Die Binärdarstellung von a kann aber unter Umständen überpolynomial viel Speicher erfordern und würde damit PSPACE überschreiten.
Ein Beispiel für eine solche Formel, deren Ergebnis PSPACE sprengt ist:

$$F = \forall x_1 \ldots \forall x_m \exists y \exists z\ (y \vee z)$$

Es gilt : arith(F) = 4^{2^m}

Beweis: arith(F) = $\forall x_1 \ldots \forall x_m \exists y ((y+0)+(y+1))$
= $\forall x_1 \ldots \forall x_m (((0+0)+(1+0))+((0+1)+(1+1)))$
= $\forall x_1 \ldots \forall x_{m-1} (4*4)$
= $\forall x_1 \ldots \forall x_{m-2} (4^2*4^2)$
= $\forall x_1 \ldots \forall x_{m-2} \; 4^{2^{(m-(m-2))}}$
= 4^{2^m} ($\rightarrow$ nicht in PSPACE darstellbar $\forall$m)

Es gilt aber andererseits, dass die Maximalgröße des arithmetischen Wertes einer Formel F der Länge n die Größe 2^{2^n} nicht überschreitet.

Beweis: (Induktion)

- einfache Variablen: $1 \leq 2^{2^1}$
- $\wedge$- bzw. $\vee$-verknüpfte boolsche Funktionen:
  die einzelnen Elemente einer $\wedge$- bzw. $\vee$ Verknüpfung haben alle Werte $\leq 2^{2^u}$ bzw. $\leq 2^{2^v}$. Da u + v ≤ n (n – Formellänge) kann durch
  $2^{2^u} + 2^{2^v}$ bzw.
  $2^{2^u} * 2^{2^v} = 2^{2^v+2^v}$
  die Größe 2^{2^n} nicht überschritten werden.
- quantifizierte Formeln der Form $\forall$x F ($\exists$x F):
  F hat nach obigen Schritten eine Größe von 2^{2^m} mit m < n (der Quantor fehlt ja noch). Damit ergibt sich für den größeren der beiden Fälle:
  $\forall$x F kann höchstens die Länge $2^{2^m} * 2^{2^m} = 2^{2*2^m}$
  $= 2^{2^{m+1}} \leq 2^{2^n}$
  haben.

Damit brauchen wir das Ergebnis der arithmetischen Auswertung nur modulo einer Zahl der Größe $2^{O(n)}$ rechnen und das Ergebnis wird zur Speicherung linear viele Bits benötigen (da der Exponent beliebig aber fest ist).
Nun kann es aber unter Umständen sein, dass die Aussagen

bool(F) = wahr $\Rightarrow$ arith(F) > 0
bool(F) = falsch $\Rightarrow$ arith(F) = 0.

keine Gültigkeit mehr haben, da nun der Wert der arithmetischen Auswertung genau dem gewählten Moduloperator entsprechen kann, womit

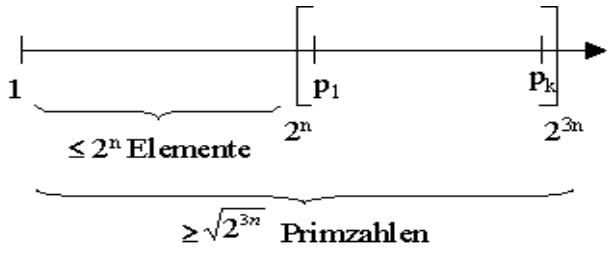
bool(F) = wahr aber arith(F) = 0

gelten würde.

Die Lösung dieser Situation erfolgt über die „günstige“ Wahl der Anzahl der Restklassen. Da a = arith(F) im Intervall [1, 2^{2^n}] liegt (a$\in$Z) wählen wir den Moduloperator k aus den Primzahlen des Intervalls [2^n, 2^{3n}]. Die Einschränkung auf das hier angegebene Intervall erfolgt aus dem Grund, dass alle nun möglichen Primzahlen garantiert größer als 2^n sind, was im späteren Verlauf des Beweises noch eine wichtige Rolle spielen wird.

Es gilt : a mod k ≠ 0.

Beweis: Es wird hier die Tatsache angewendet dass es für jedes m mehr als $\sqrt{m}$ -viele Primzahlen mit der Eigenschaft, dass diese kleiner als m sind gibt. Das heißt es gibt im angegebenen Intervall $\sqrt{2^{3m}} - 2^n > 2^n$ viele Primzahlen $p_1, ..., p_k$



Annahme: Für alle p_i (i < k) gilt a mod p_i = 0.

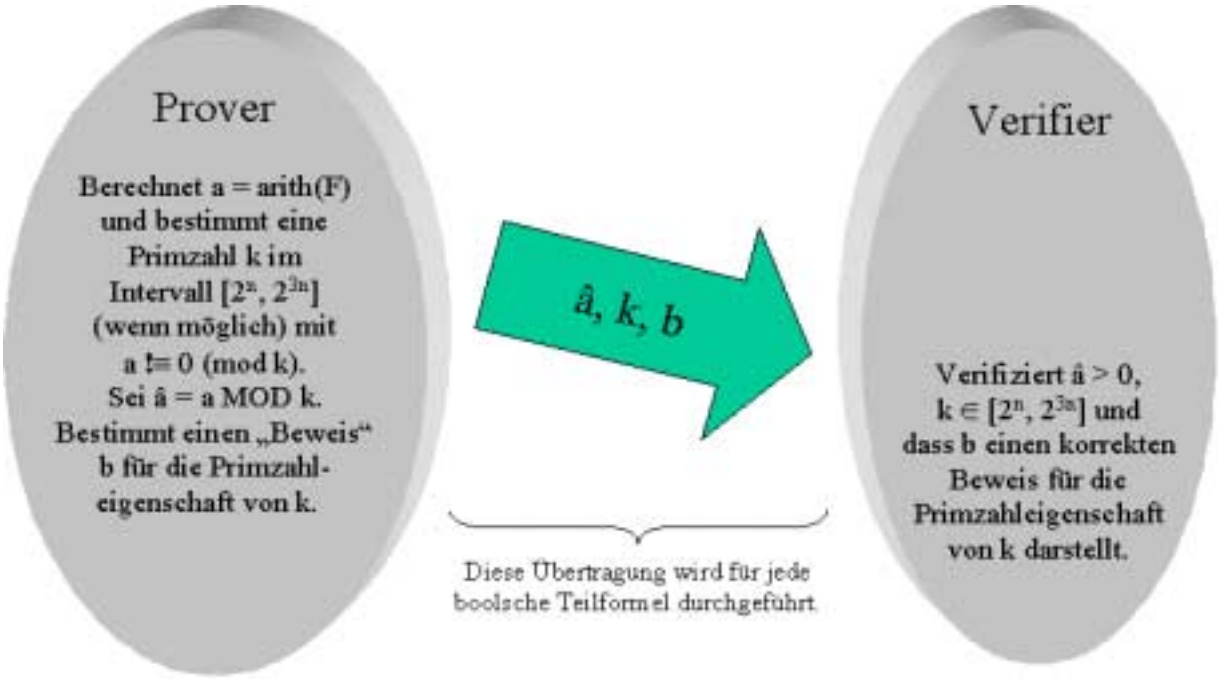
Dann gilt a mod $(\prod_{i=1}^{k} p_i) = 0$ (da jedes p_i (i < k) Teiler von a ist).

Weiterhin ist $\prod_{i=1}^{k} p_i > \prod_{i=1}^{k} 2^n$ (jedes p_i ist größer als 2^n)

und $\prod_{i=1}^{k} 2^n > 2^{2^n}$ (da n beliebig aber fest).

Da aber a < 2^{2^n} muss aufgrund von a mod$(\prod_{i=1}^{k} p_i) = 0$ a = 0 gelten, was zu einem Widerspruch zu a ∈ [1, 2^{2^n}] führt.

Daraus folgt, es existiert ein p_i mit a mod $(p_i) \neq 0$.
Die Zahlen, die laut dem Protokoll nun noch übertragen werden müssen sind nun alle in PSPACE darstellbar. Damit hat das Protokoll nun folgende Form (n ist beiden Maschinen bekannt, da es die Länge der Eingabe ist) :



2.5 Die Primzahleigenschaft der Anzahl der Restklassen

Der Beweis der Primzahleigenschaft des Moduloperators ist in der Klasse NP, daraus folgt, dass wie in Teil 1 gezeigt wurde dieser Beweis auch in PSPACE liegt. Aufgrund der Primzahleigenschaft von k erfüllt damit die Menge der Restklassen ({0 , ... , k-1} = GF(k)) die Körperaxiome (wäre k keine Primzahl, so wäre die Nullteilerfreiheit nicht erfüllt) [2]. Wäre GF(k) kein Körper, so könnte man nicht behaupten, dass Polynome des Grades g höchstens g viele Nullstellen haben, womit die Wahrscheinlichkeit, das ein Betrug auffällt sinkt.

Bis jetzt haben wir nur einen elementaren Teilschritt des Ablauf des Protokolls beschrieben, wie er für alle Einzelteile der Formel durchgeführt werden muss, d.h. wir zerlegen die Formel Schritt für Schritt in Teilformeln und beweisen deren Richtigkeit bis wir auf der untersten Ebene nur noch Variablen vorfinden.
Formeln, die mit $\wedge$ bzw. $\vee$ verknüpft sind (z.B. $F = F_1 \wedge F_2$) führen zur Übertragung von zwei Zahlen (a_1, a_2), die die Werte der einzelnen Teilformeln repräsentieren. Der Verifier prüft nun, ob $\hat{a} = a_1 * a_2 \pmod k$. Anschließend muss bewiesen werden, dass die einzelnen Werte der Teilformeln tatsächlich a_1 bzw. a_2 entsprechen, d.h.:

$$a_1 = \text{arith}(F_1) \pmod k \text{ und}$$
$$a_2 = \text{arith}(F_2) \pmod k.$$

Dies wird rekursiv weiterbetrieben, bis zur untersten Ebene.

2.6 Behandlung der Quantoren

In Formeln der Form $\forall x\ G$ ($\exists x\ G$) ist x in G freie Variable, d.h. bei arithmetischer Auswertung könnte für x jede Zahl aus GF(k) eingesetzt werden, woraus sich eine Funktion mit der Variablen $x \in GF(k)$ ergibt.
Es wird in jeder Runde für die jeweils unquantifizierte Variable eine Konstante eingesetzt, die zufällig gewählt wird. Dadurch entsteht ein variablenfreier arithmetischer Ausdruck.

Das im ersten Schritt aus G entstehende Polynom p_G (es gibt ja nur + bzw. * Operationen in dem so entstehenden Polynom) wird vom Prover nun dem Verifier übertragen, sowie â und dieser verifiziert nur :

$$p_G(0) * p_G(1) = \hat{a} \pmod k \quad [\text{bzw. } p_G(0) + p_G(1) = \hat{a} \pmod k].$$

Anschließend wählt der Verifier ein Element aus GF(k) und der Prover soll ihm beweisen, dass $\text{arith}(G_{x=z}) = p_G(z)$. (Ist p_G das zu G gehörige Polynom, so ist arith(G) nur eine andere Darstellung von p_G und die Werte beider Terme nach Einsetzen der Zahl z für die Variable x sind gleich.)

---

[2] Definition Körper:
Menge von Elementen, auf denen zwei Operationen (Körpermultiplikation und Körperaddition) definiert sind, die folgende Eigenschaften erfüllen:
- beide Operationen sind kommutativ, assoziativ und distributiv
- es existiert jeweils ein Neutrales und ein Inverses Element bezüglich der Operationen
- die Nullteilerfreiheit von Körpern lässt sich aus den Axiomen ableiten (siehe Mathe 1)

Jetzt ist die Beweisarbeit des Provers, dem Verifier zu zeigen, dass das Polynom tatsächlich zu der boolschen Formel äquivalent ist. Diese Stelle des Portokolls realisiert eine Betrugskontrolle. Da der Verifier die Formel G und somit auch arith(G) kennt, verifiziert er an dieser Stelle, dass p mit hoher Wahrscheinlichkeit das zu G gehörige Polynom ist, was beweist, dass der Prover richtig gerechnet hat. Jedoch besteht noch die Chance, dass das Polynom falsch war und nur zufällig genau an dieser Stelle mit dem „richtigen“ übereinstimmt. Die Wahrscheinlichkeit für diesen Fall wird nun gezeigt.

Nehmen wir Polynome p und p’ vom Grad d < k an, wobei der Prover den Verifier davon zu überzeugen hat, dass p’ zu $G_{x=z}$ gehört, tatsächlich jedoch p das zu $G_{x=z}$ gehörige Polynom ist. Dann ist die Wahrscheinlichkeit, mit der der Betrug nicht auffällt genau so groß, wie das Verhältnis der Anzahl der Stellen, an denen die Polynome p und p’ gleich sind, zur Kardinalität des Körpers GF(k). Betrachtet man das Differenzpolynom p – p’, so entsprechen die Stellen der Gleichheit von p und p’ den Nullstellen. Da das Differenzpolynom maximal den Grad d hat, können dies höchstens d-viele sein. Die Wahrscheinlichkeit, eine dieser Stellen im Bereich von 0 bis k-1 zu „treffen“ liegt damit bei $\frac{d}{k} \leq \frac{d}{2^n}$ (da $k > 2^n$).

Sind wir auf der untersten Ebenen einer Formel G angekommen (d.h. es sind nur noch Variablen in G zu ersetzen), so müssen sich die am Anfang zufällig für die freien Variablen eingesetzten Zahlen jetzt aus der Rechnung ergeben, da diese Zahlen für die Variablen in jedem Schritt beibehalten wurden.

Betrachtet man die oben erwähnten Polynome, so liegt keine Einschränkung bezüglich deren Grad vor. Da aber alle Koeffizienten übertragen und damit gespeichert werden müssen können es höchstens polynomial viele sein. Die Länge eines einzelnen Koeffizienten überschreitet nicht PSPACE da er aus GF(k) stammt. Daraus folgt, es muss eine weitere Einschränkung für die zugrunde liegende Sprache QBF’ vorgenommen werden.

2.7 Zweite Einschränkung von QBF (QBF’’)

Es sei QBF’’ die Menge aller Formeln in QBF’, die folgende Eigenschaft erfüllen:

- Für alle in einer Formel aus QBF’ vorkommenden Variablen ist zwischen der Quantifizierung der Variablen und allen Vorkommen höchstens ein Allquantor.

Für alle arithmetischen Interpretationen der Formeln aus QBF’’ ist der Polynomgrad höchstens 2n.
Beweis:

- Da zwischen der Allquantifizierung einer Variable und ihrem Auftreten maximal ein weiterer Allquantor liegt, kann dies höchstens eine Verdoppelung des Grades verursachen. In der Formel selbst bewirken nur UND-Operationen eine Inkrementierung des Grades. ODER-Operationen sowie Existenzquantoren bewirken keine Erhöhung des Grades. Somit kann der Grad die Größe 2n (n – Stringlänge) nicht überschreiten.

QBF'' ist in polynomialer Zeit auf QBF' reduzierbar. Der folgende Beweis rechtfertigt für die IP = PSPACE Betrachtung eine Einschränkung auf QBF''.
Beweis:

- Man ersetzt jede Teilformel der Form Qx....$\forall$y H(x,...) Q $\in$ {$\forall$, $\exists$} durch die äquivalente Formel
  Qx....$\forall$y$\exists$x' ((x $\leftrightarrow$ x') $\wedge$ H(x',….))
  $\Leftrightarrow$ Qx....$\forall$y$\exists$x' (((¬x $\wedge$ ¬x') $\vee$ (x $\wedge$ x')) $\wedge$ H(x',….))
- Wie man sieht wird jede Variable durch eine neue ersetzt, die äquivalent zu der Alten ist. Damit sind die Formeln gleichwertig, zudem ist die Ersetzungszeit maximal polynomial. Die entstandene Formel erfüllt die Definition von QBF''.

Betrachtet man nun die möglichen Eingabeformeln, so wird im Verlauf des Interaktiven Beweises die polynomiale Platzschranke nicht verletzt. Es können zwei Fälle auftreten:

1. G ist gültig.
   In diesem Fall wird der Prover sowohl den korrekten arithmetischen Wert der Gesamtformel als auch aller Teilformeln übertragen und der Verifier wird mit Wahrscheinlichkeit 1 (Übertagungsfehler ausgeschlossen) akzeptieren.
2. G ist nicht gültig.
   Der Verifier wird in diesem Fall nur dann akzeptieren, wenn der Prover ihm in einem Schritt ein falsches Polynom p' liefert und der Verifier eine Zufallszahl z so wählt, dass p' mit dem richtigen Polynom p in z übereinstimmt. Da p und p' an maximal 2n Stellen übereinstimmen, ist die Chance in einer Runde dafür $2n/2^n$. Bei einer Laufzeit von kleiner gleich n Runden sinkt damit die Wahrscheinlichkeit auf $2n^2/2^n$.

Damit ist abschließend die von IP geforderte geringe (< ⅓) Wahrscheinlichkeit eines fälschlichen Akzeptierens erfüllt. Es folgt IP = PSPACE.

3. Zero Knowledge

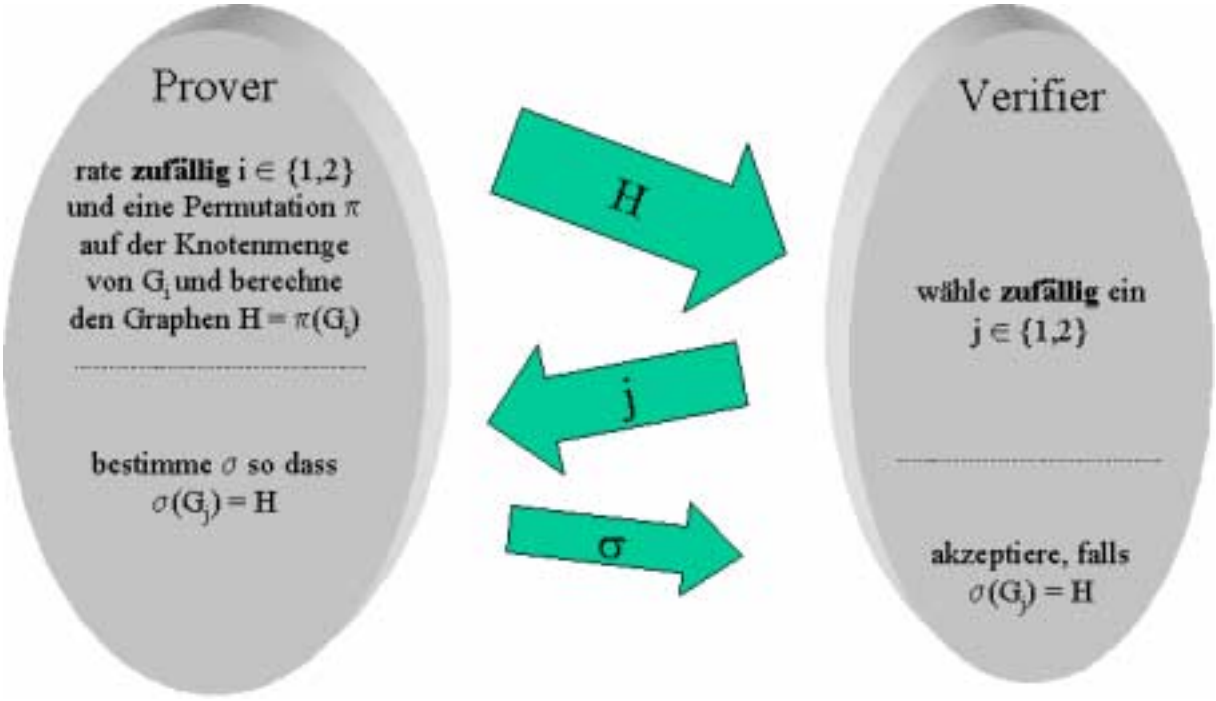
Zero Knowledge bedeutet, dass ein IP-Beweis geführt werden kann, ohne dass dem Verifier irgendeine Information über den Beweis selbst gegeben wird. Trotzdem muss dieser von der Existenz des Beweises überzeugt sein, denn es handelt sich ja um ein korrektes IP-Protokoll.

3.1 Definition Zero Knowledge

Die Zero Knowledge Eigenschaft eines Beweises sagt aus, dass ein interaktiver Beweis geführt werden kann, der dem Verifier keine Information über den Beweis selbst gibt, ihn aber dennoch von der Existenz des Beweises überzeugt.
Etwas anschaulicher kann man diese Eigenschaft auch so definieren:
Ein Protokoll besitzt die Zero Knowledge – Eigenschaft, wenn es einer am Protokoll nicht beteiligten Instanz, einem „Simulator“ gelingt, eine Kopie der Sicht auf das Protokoll zu erstellen, wie es der Verifier mitbekommt, ohne dass der Simulator dabei mit dem Prover kommuniziert oder das Geheimnis kennt. Diese Kopie darf für Außenstehende nicht von der originalen Sicht des Verifiers in Interaktion mit dem Prover zu unterscheiden sein.
Letztere Definition soll ausdrücken, dass der Verifier bei der Ausführung des Protokolls nicht mehr mitbekommt als ein Außenstehender ihm präsentieren könnte, der nicht daran beteiligt war. Dies wiederum impliziert, dass der Verifier keine für die Entschlüsselung des Geheimnisses relevanten Hinweise bei der Protokollausführung erfährt (ein Außenstehender hat ja auch kein Wissen über das Geheimnis), was bei dieser Definition der Zero Knowledge – Eigenschaft entspricht.

3.2 Anwendungsbeispiel

Ein bekanntes Beispiel ist das Graphenisomorphieproblem. Um zu beweisen, dass zwei gegebene Graphen isomorph sind, wäre es das einfachste einen Isomorphismus zwischen den beiden Graphen dem Verifier zu übermitteln. Dabei werden auch nur polynomial viele Bits verwendet. Somit ist aber das Geheimnis verraten, der Verifier könnte das Geheimnis auch noch an Dritte weitergeben. Dies ist aber im allgemeinen nicht erwünscht. Ein solchen Ansatz umzusetzen vermag ein Zero-Knowledge Beweis.
Die Modellierung erfolgt mittels eines IP-Protokolls, in dem der Prover beginnt und beide Kommunikationspartner probabilistisches Verhalten zeigen.



Wie aus der Grafik zu erkennen ist, handelt es sich bei dem Protokoll um ein IP(3) Protokoll, da genau drei Runden benötigt werden, um den Beweis durchzuführen. Sind die Eingangsgraphen isomorph, so sind sie auch beide isomorph zu H, deshalb existiert für jede Kombination ein Isomorphismus σ, womit die Akzeptanzwahrscheinlichkeit 1 annimmt. Im Fall der Nichtisomorphie ist die Akzeptanzwahrscheilichkeit ½, da jeder beliebige Prover eine Chance von 50% hat, im ersten Schritt den „richtigen" Graphen zu raten und somit den Isomorphismus σ zu bilden.
Der Verifier kann aus den ihm kommunizierten Informationen kein Isomorphismus zwischen den Eingabegraphen ableiten, da er ihn selbst berechnen müsste. Er hat lediglich einen Isomorphismus zwischen einem der Eingabegraphen und einem Graphen H, der (mit hoher Wahrscheinlichkeit) nicht dem anderen Eingabegraphen entspricht. Dies ermöglicht somit keinen Schluss auf die Isomorphie der Eingabegraphen. Eine Berechnung ist für den Verifier aufgrund seiner polynomiellen Zeitbeschränkung ebenfalls ausgeschlossen.

4. Quellen

Perlen der Theoretischen Informatik, Schöning
Moderne Verfahren der Kryptographie, Beutelspacher, Schwenk, Wolfenstetter