Algorithmische Metatheoreme 2.0

Till Tantau

28. September 2012

Gliederung

Worum es geht

- Klassische algorithmische Metatheoreme . . .
- . . . versus neue Varianten für die Komplexitätstheorie

Neue Varianten 1: Platzkomplexität

- Algorithmische Metatheoreme für Platzklassen
- Anwendung: Zyklen gerader Länge
- Anwendung: Pseudopolynomialzeit-Algorithmen

Neue Varianten 2: Schaltkreise

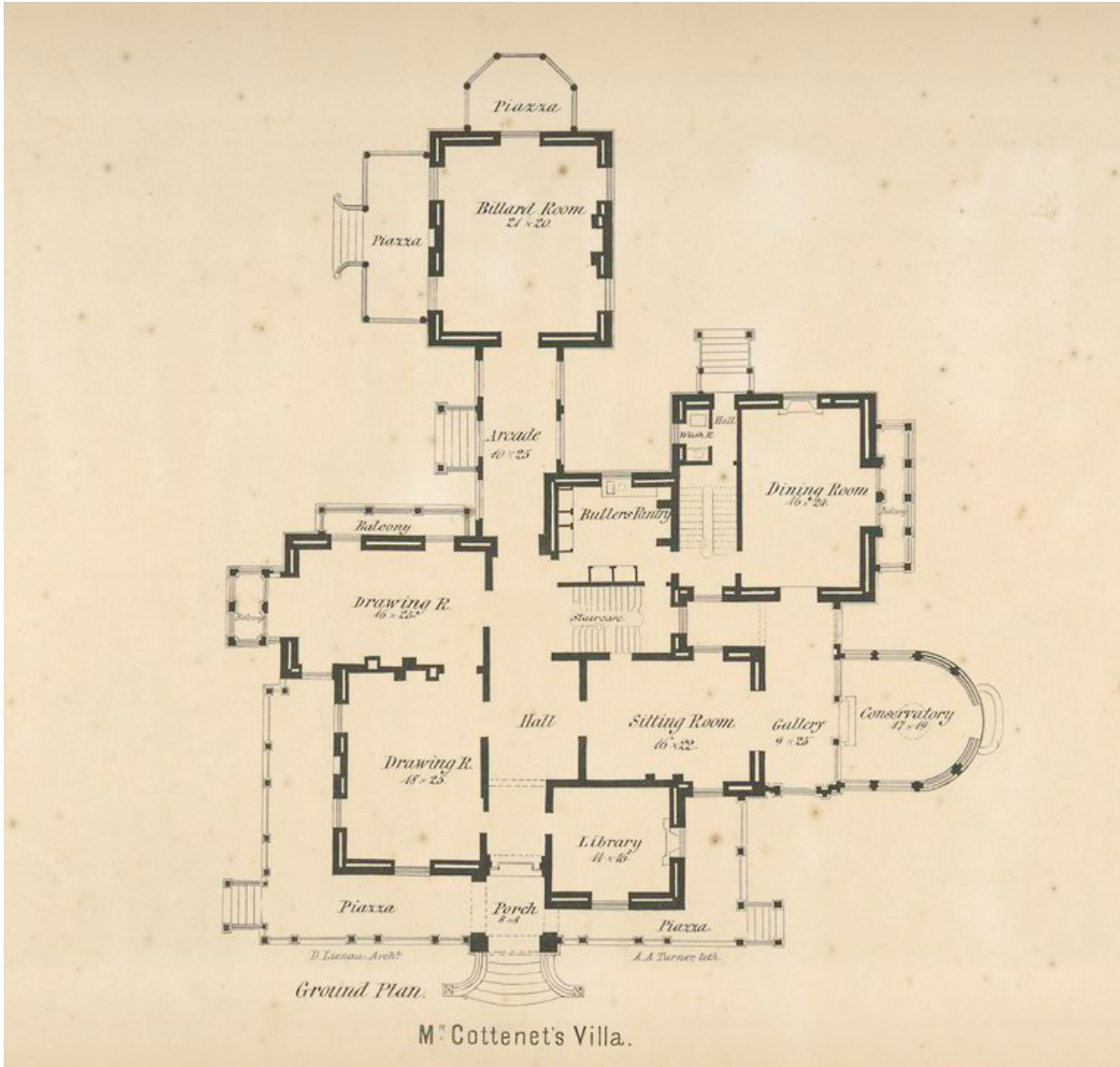
- Algorithmische Metatheoreme für Schaltkreisklassen
- Anwendung: Visible-Pushdown-Sprachen

Zur Einstimmung

Author Daniel Case, Creative Commons Attribution Sharelike License

- Das Cottenet-Brown House soll modernisiert werden.

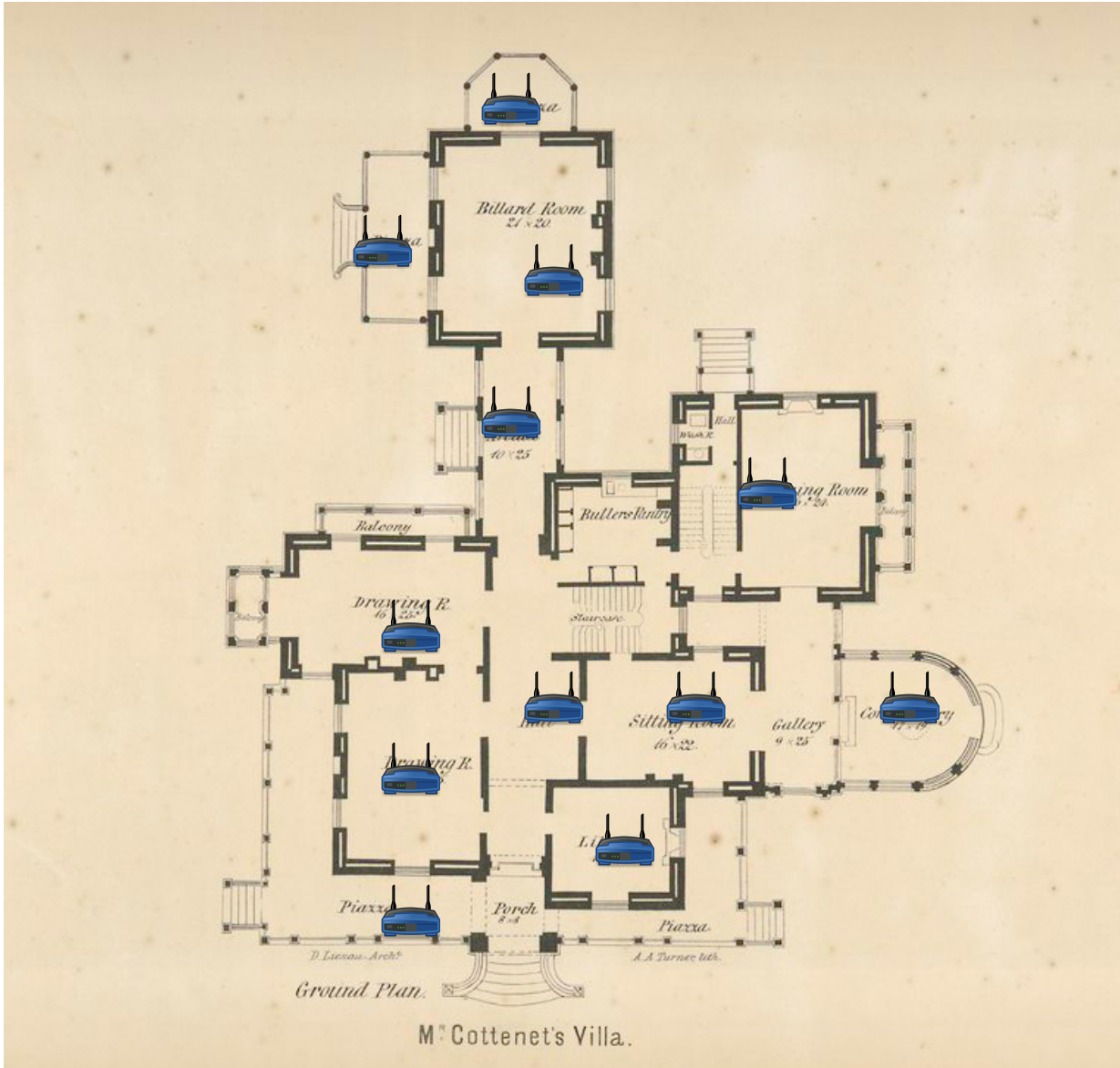
Zur Einstimmung



Public Domain

- Das Cottenet-Brown House soll modernisiert werden.

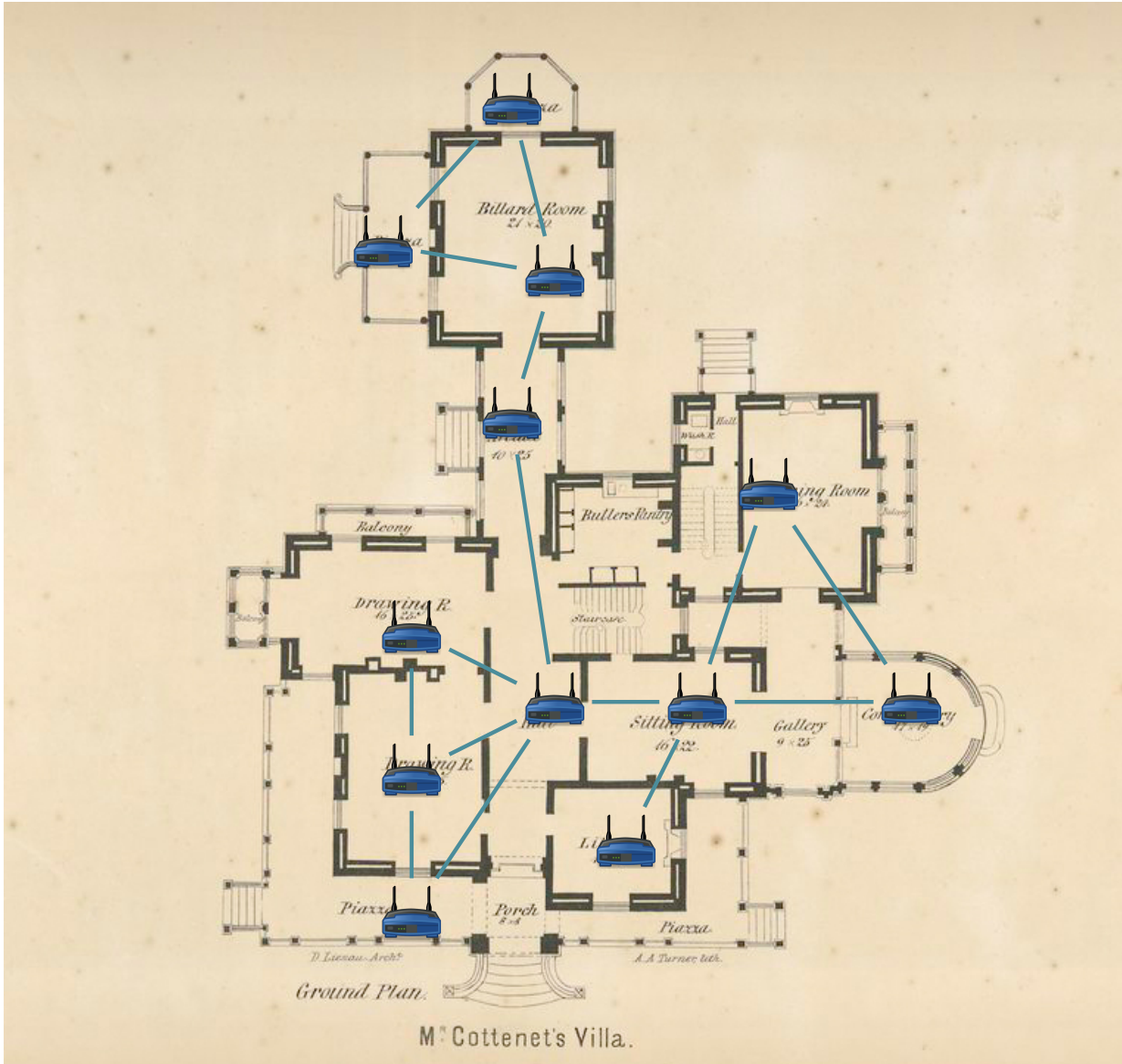
Zur Einstimmung



Public Domain

- Das Cottenet-Brown House soll modernisiert werden.
- In den Räumen werden WLAN-Router aufgestellt.

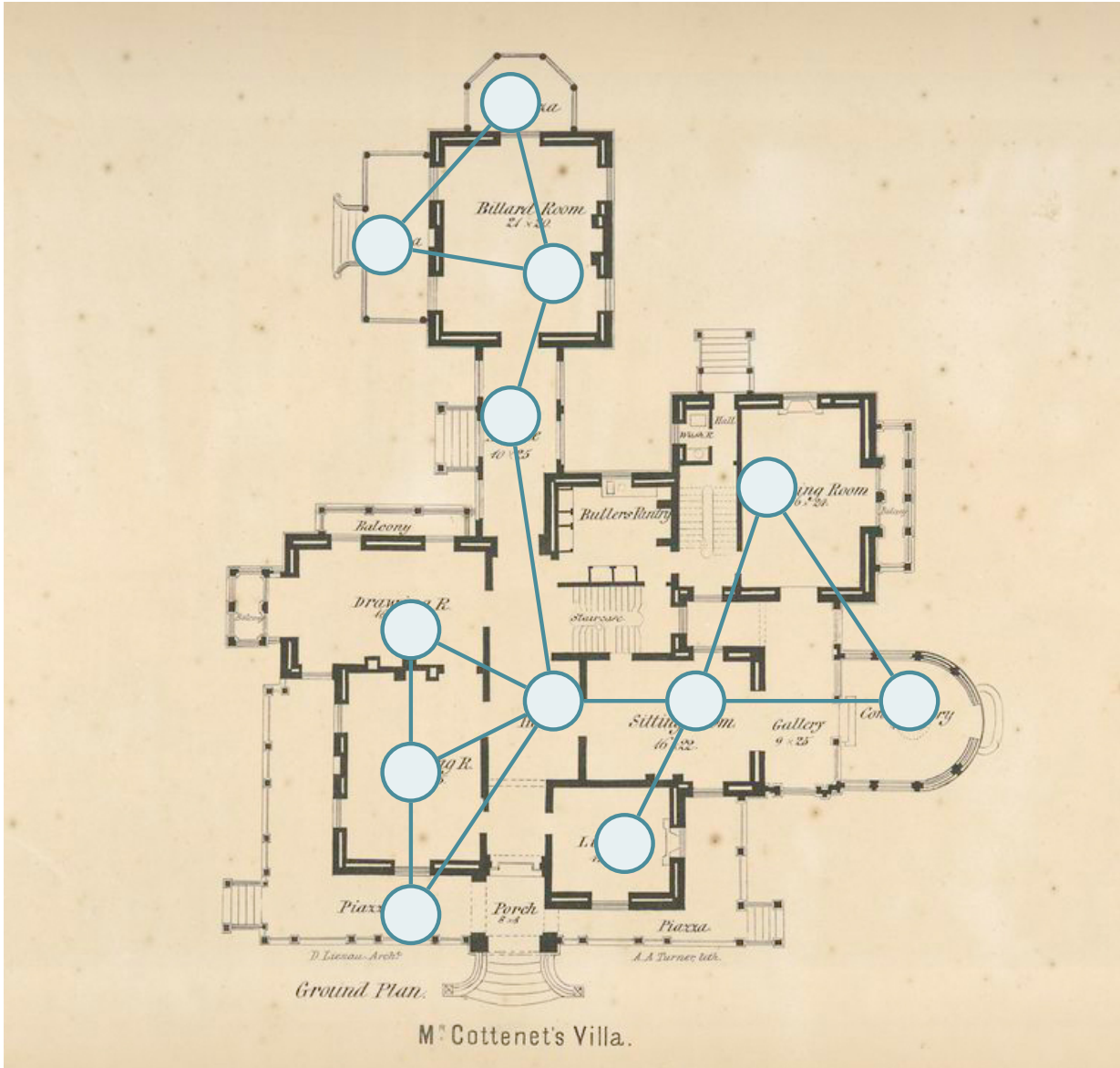
Zur Einstimmung



Public Domain

- Das Cottenet-Brown House soll modernisiert werden.
- In den Räumen werden WLAN-Router aufgestellt.
- In bestimmten Fällen stören sie sich.

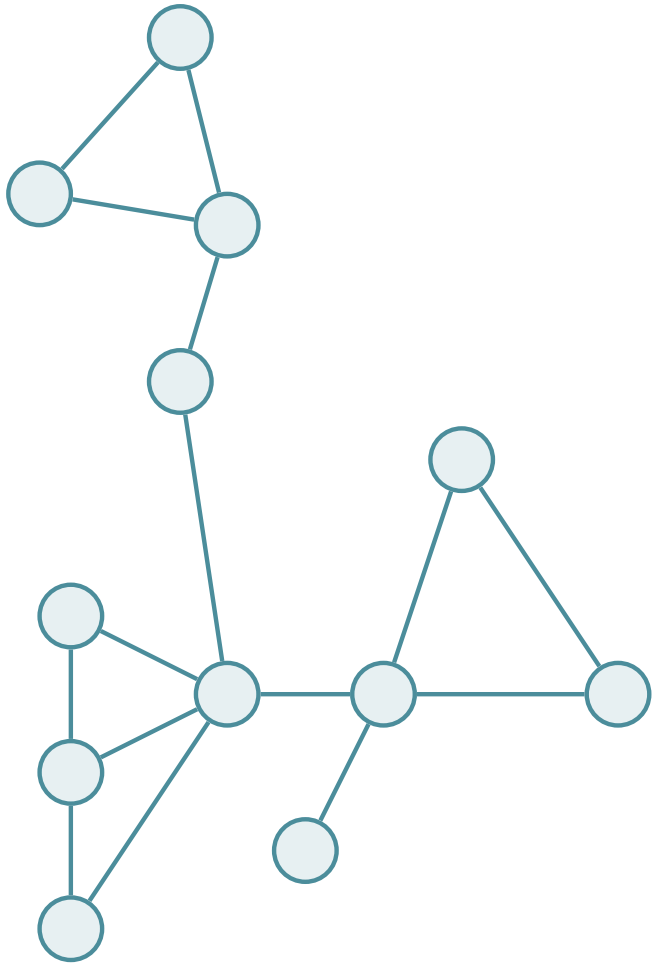
Zur Einstimmung



Public Domain

- Das Cottenet-Brown House soll modernisiert werden.
- In den Räumen werden WLAN-Router aufgestellt.
- In bestimmten Fällen stören sie sich.

Zur Einstimmung



- Das Cottenet-Brown House soll modernisiert werden.
- In den Räumen werden WLAN-Router aufgestellt.
- In bestimmten Fällen stören sie sich.
- Das abstrakte Problem ist *3-Färbbarkeit*.

Was man über 3-Färbbarkeit weiß.

Definition (3-Färbbarkeit)

Die Sprache 3-COLORABLE enthält alle (Codes von) Graphen,
deren Knotenmenge sich mit drei Farben einfärben lässt,
so dass keine zwei benachbarte Knoten dieselbe Farbe haben.

- Das Problem ist ein klassisches NP-vollständiges Problem.
- Falls also P $\neq$ NP, so lässt es sich *nicht* in polynomieller Zeit lösen, aber vielleicht *approximieren* oder *mit niedrigem Exponenten* lösen oder *im Schnitt schnell lösen* oder, oder, oder . . .
- Wir betrachten den Fall, dass sich die Eingabe *baumartig zerlegen* lässt, damit wir *Teilen-und-Herrschen* anwenden können.

Das Räuber-und-Gendarmen-Spiel

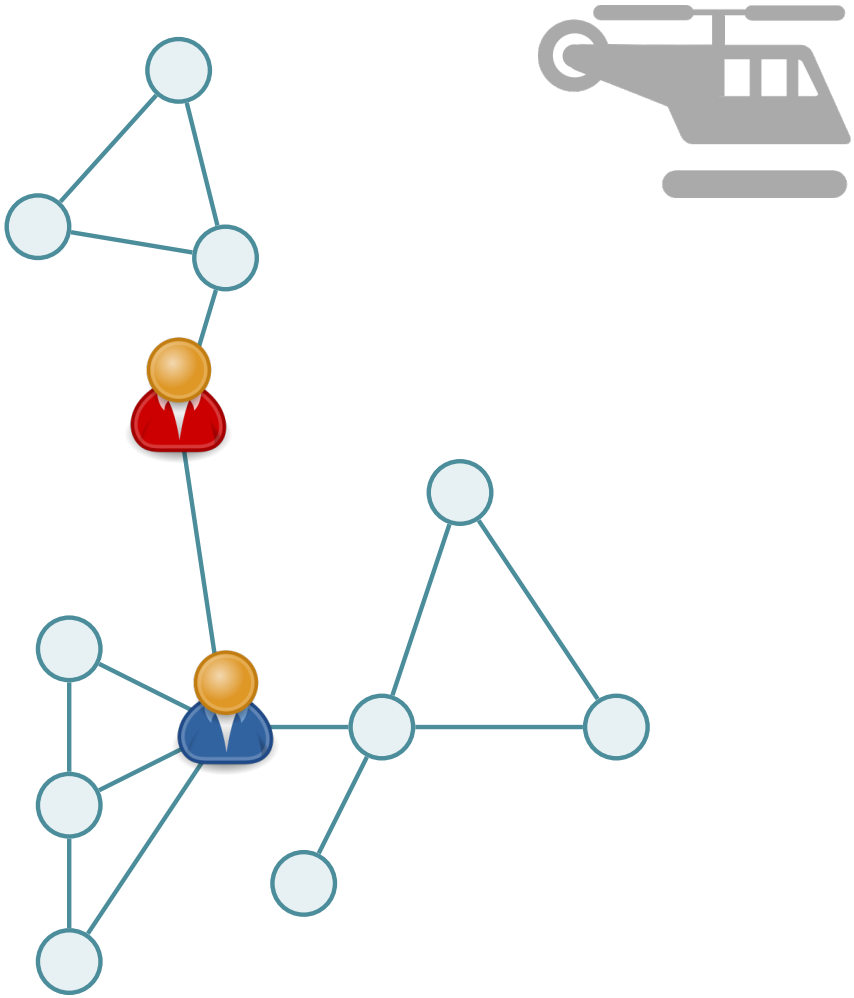
Spielziel

Eine Truppe von *k Gendarmen* wollen einen *Räuber* fassen, indem sie auf denselben Knoten wie der Räuber kommen.

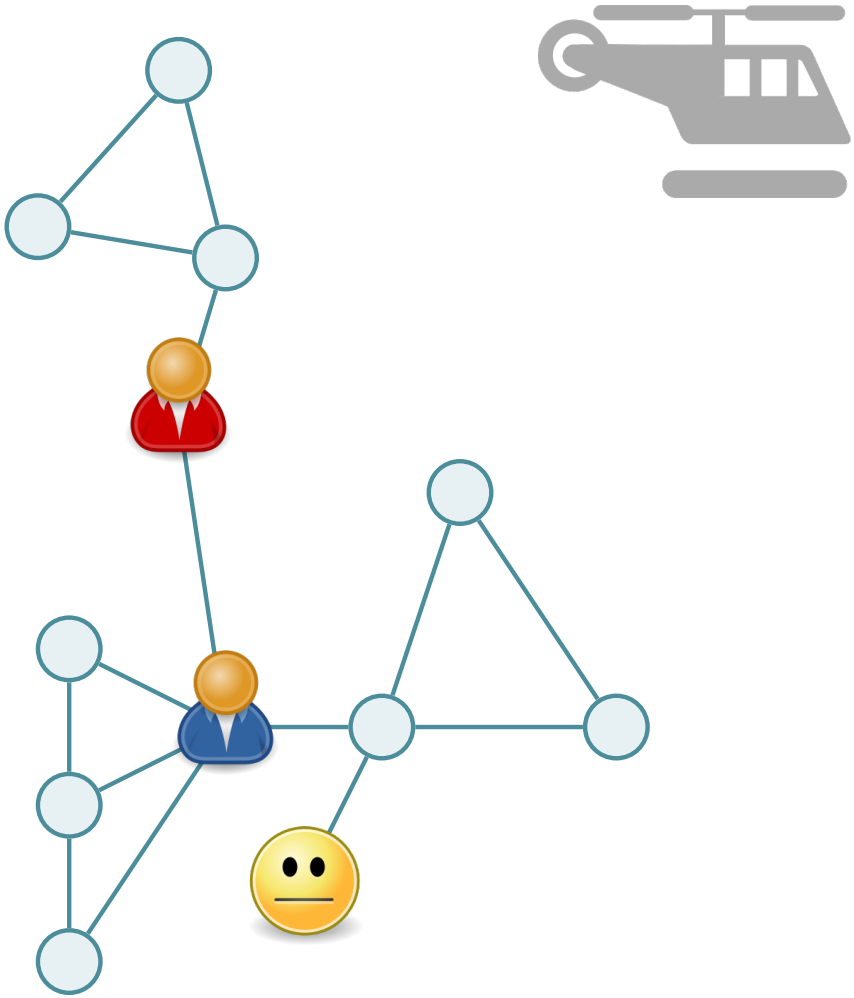
Spielregeln

1. Erst die Gendarmen, dann der Räuber suchen sich Startknoten.
2. Ein Gendarm steigt in einen *Hubschrauber* und steuert einen *noch nicht besuchten Knoten* an.
3. Währenddessen *bewegt sich der Räuber beliebig entlang Kanten*, wobei er das Ziel des Hubschraubers kennt.

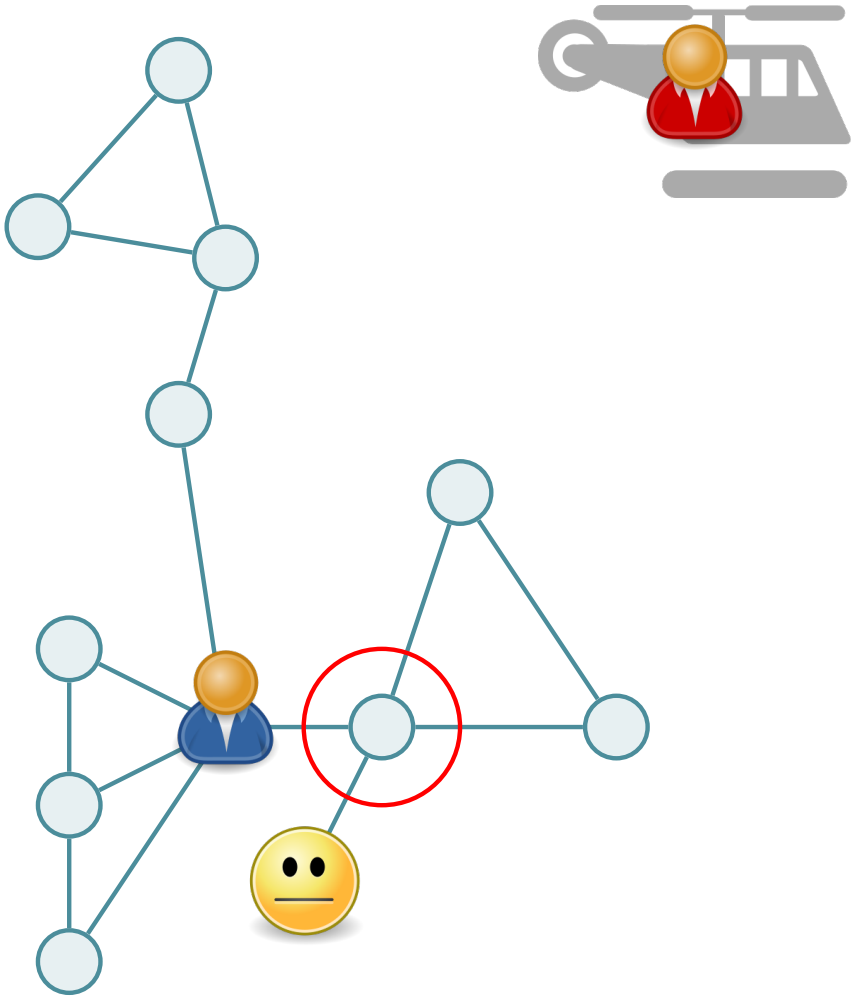
Das Räuber-und-Gendarmen-Spiel
Zwei Gendarmen reichen nicht.



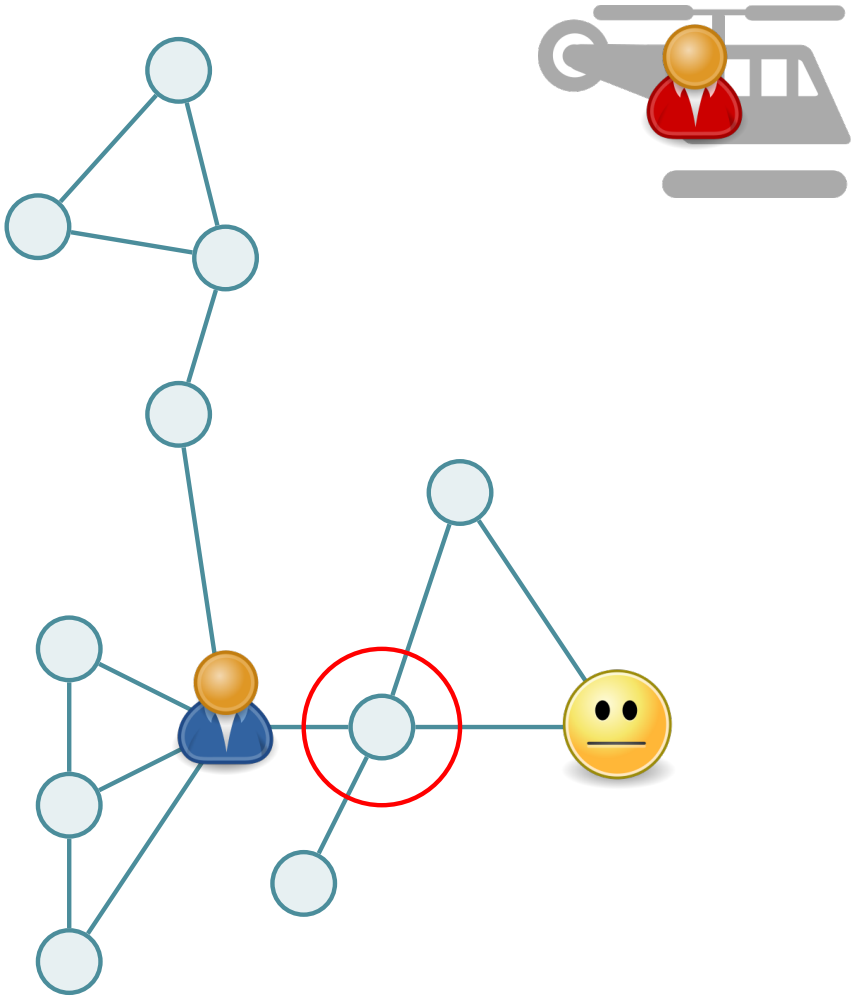
Das Räuber-und-Gendarmen-Spiel
Zwei Gendarmen reichen nicht.



Das Räuber-und-Gendarmen-Spiel
Zwei Gendarmen reichen nicht.

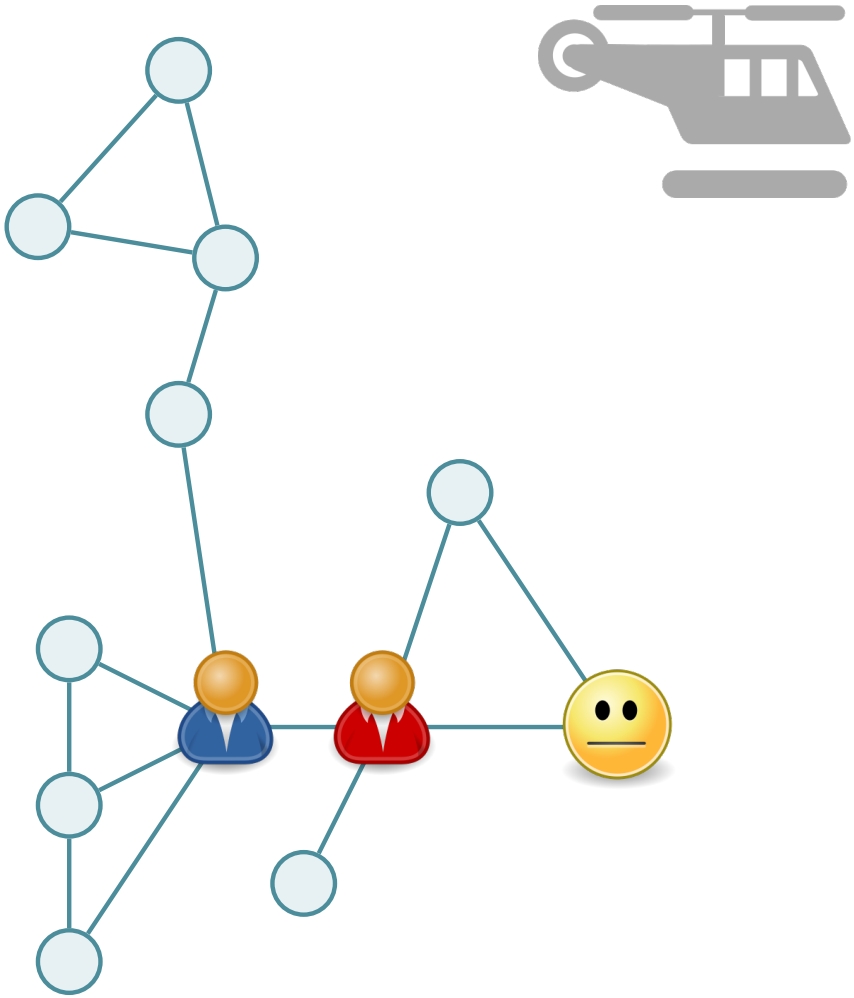


Das Räuber-und-Gendarmen-Spiel
Zwei Gendarmen reichen nicht.

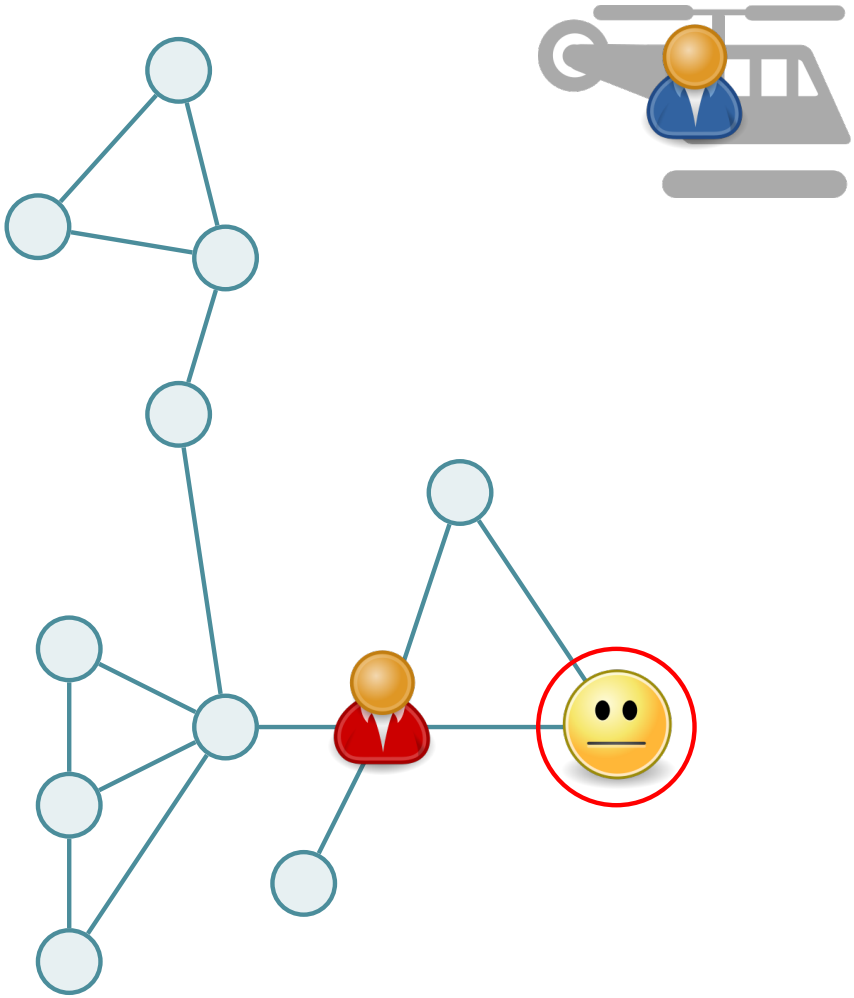


Das Räuber-und-Gendarmen-Spiel

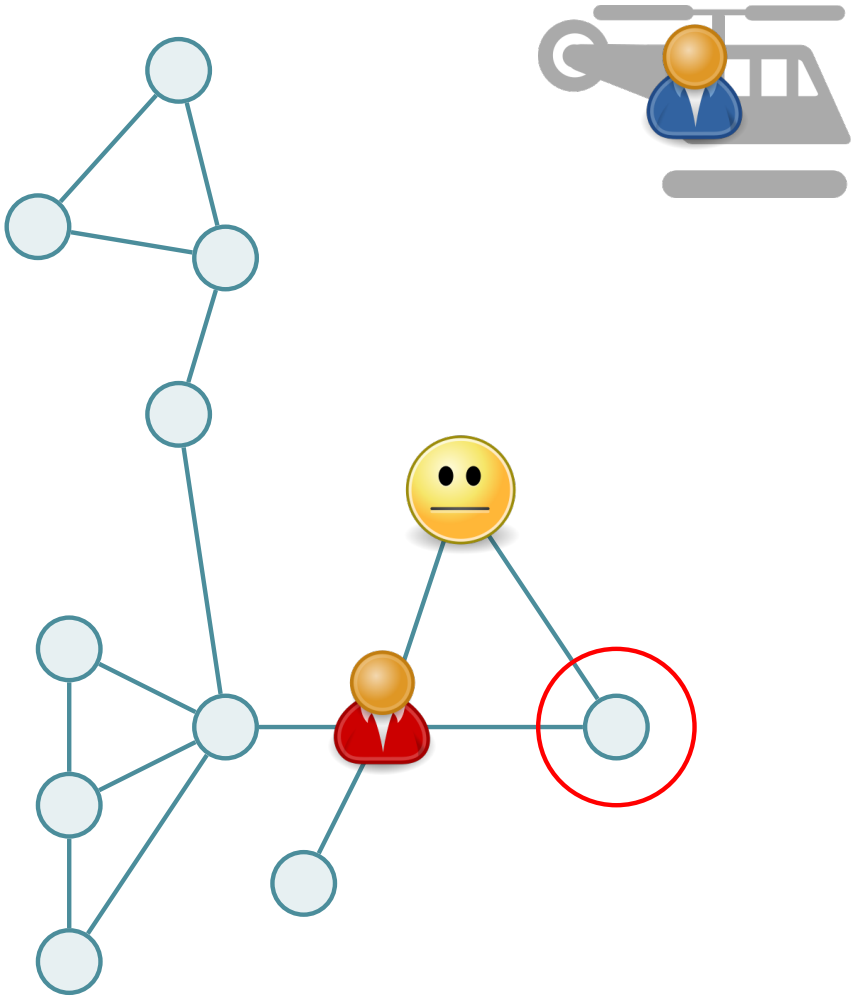
Zwei Gendarmen reichen nicht.



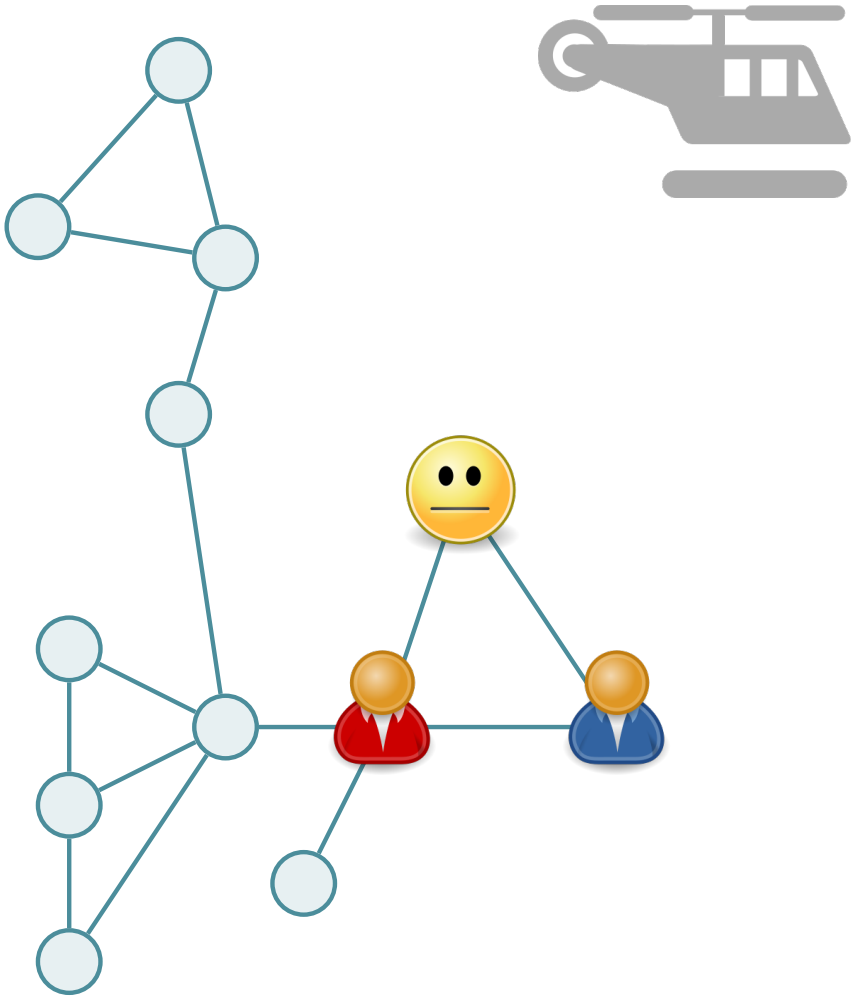
Das Räuber-und-Gendarmen-Spiel
Zwei Gendarmen reichen nicht.



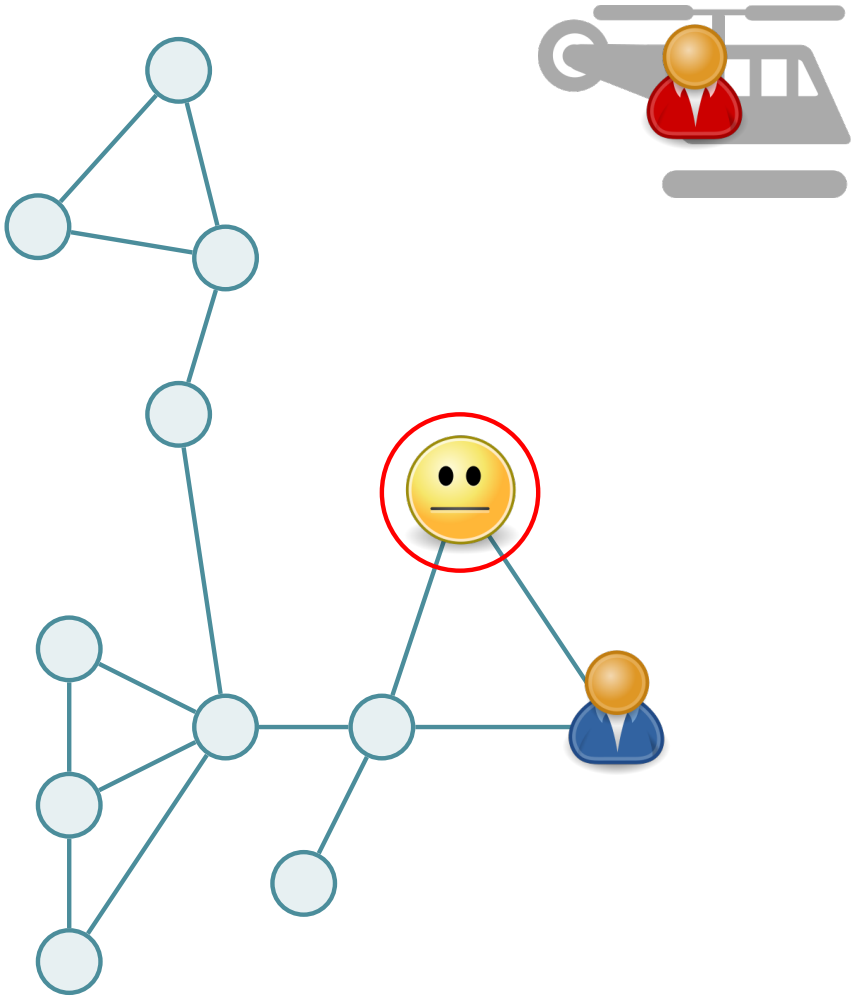
Das Räuber-und-Gendarmen-Spiel
Zwei Gendarmen reichen nicht.



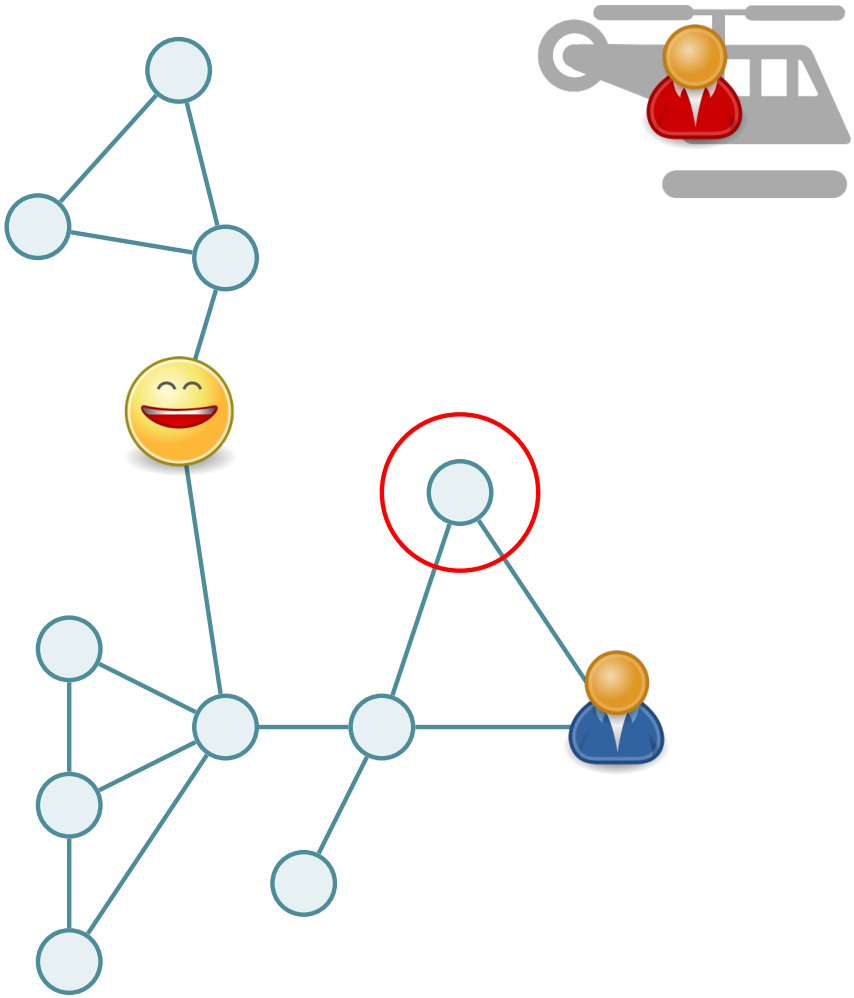
Das Räuber-und-Gendarmen-Spiel
Zwei Gendarmen reichen nicht.



Das Räuber-und-Gendarmen-Spiel
Zwei Gendarmen reichen nicht.

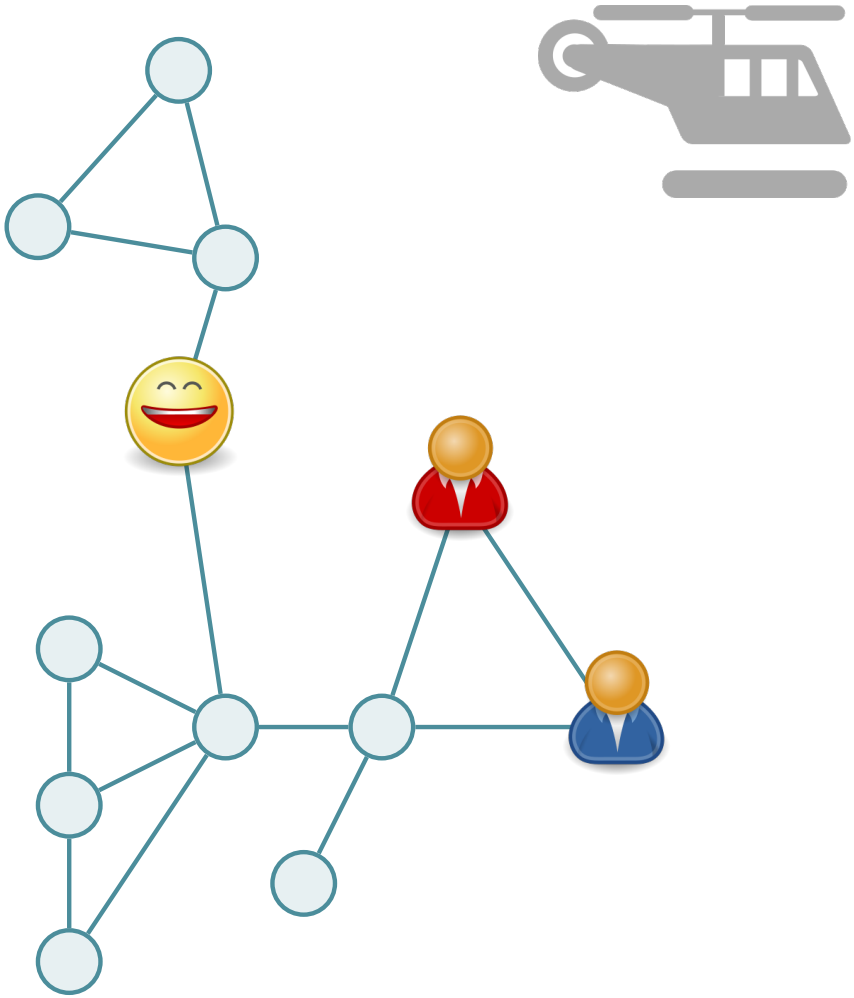


Das Räuber-und-Gendarmen-Spiel
Zwei Gendarmen reichen nicht.

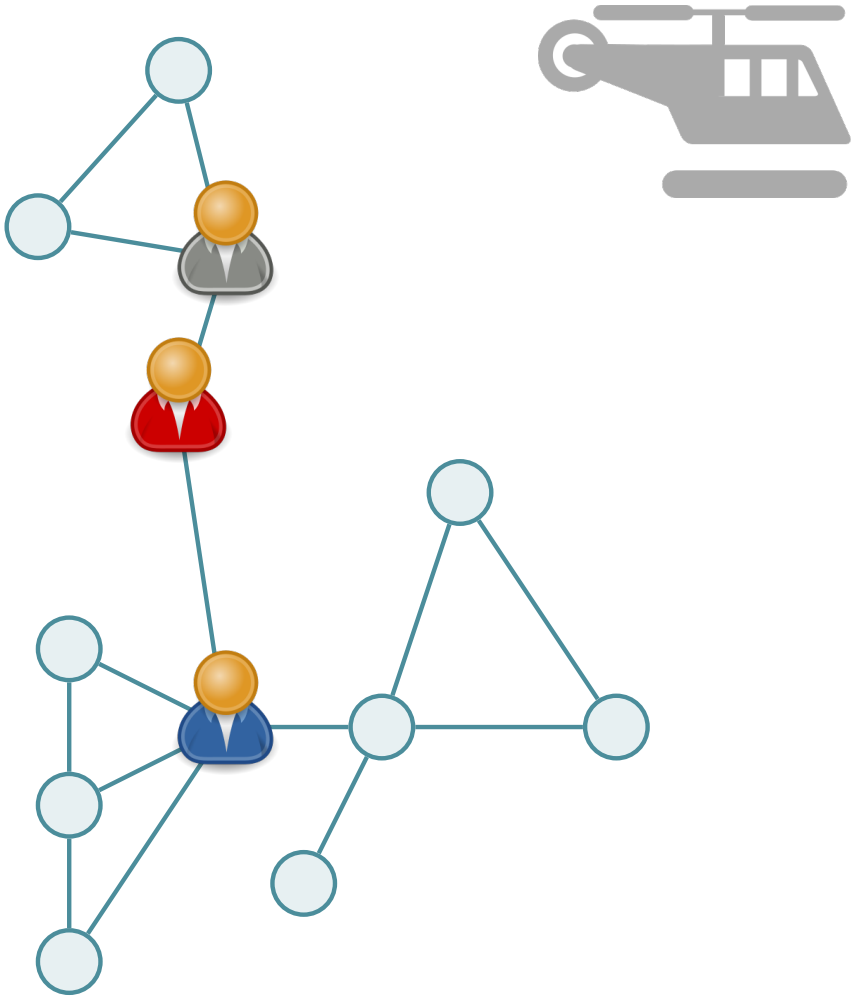


Das Räuber-und-Gendarmen-Spiel

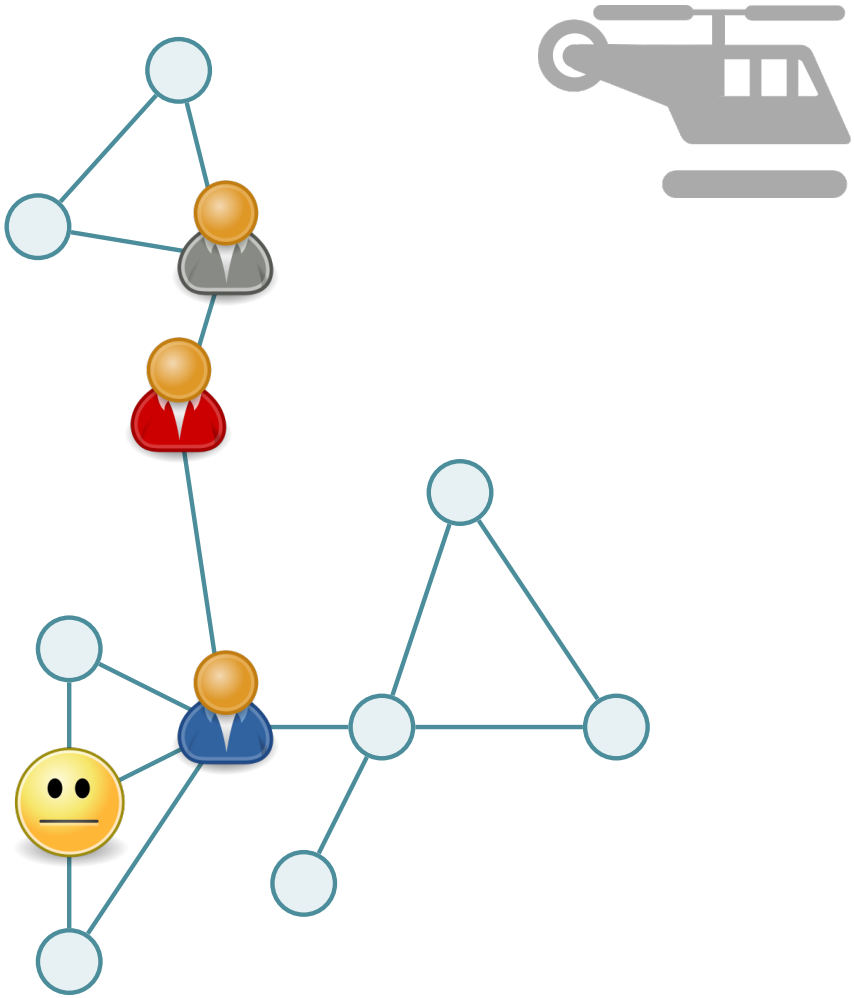
Zwei Gendarmen reichen nicht.



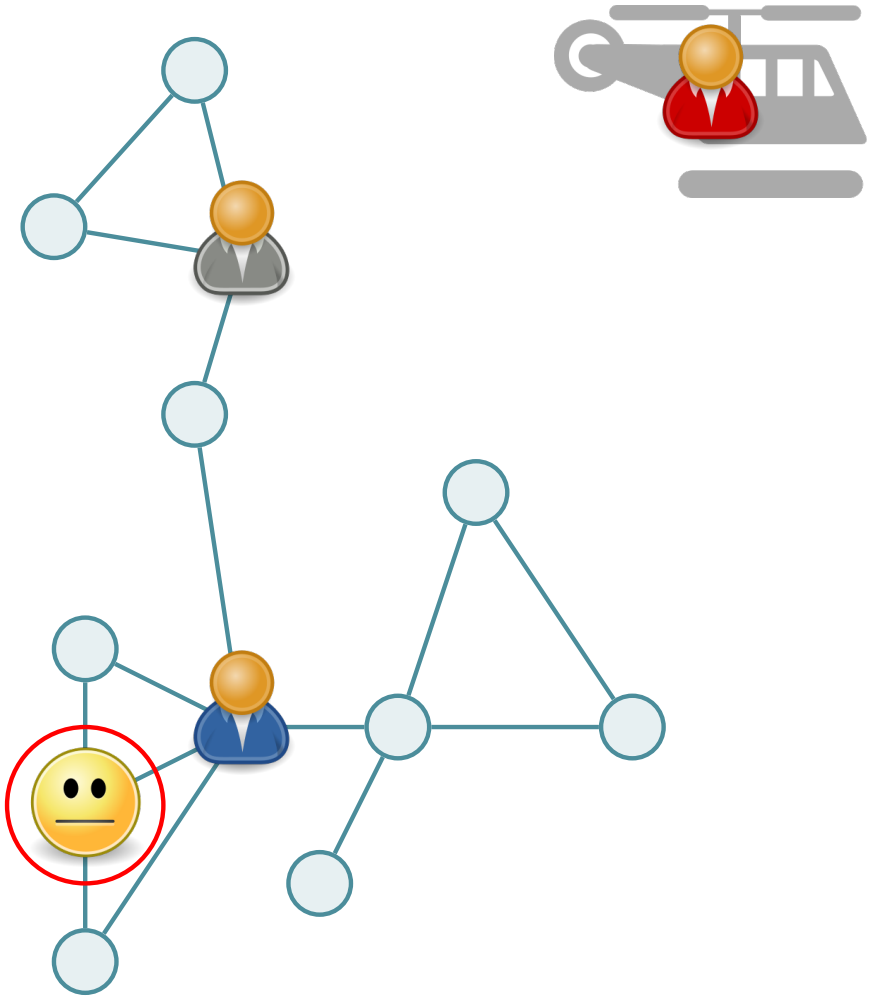
Das Räuber-und-Gendarmen-Spiel
Drei Gendarmen reichen.



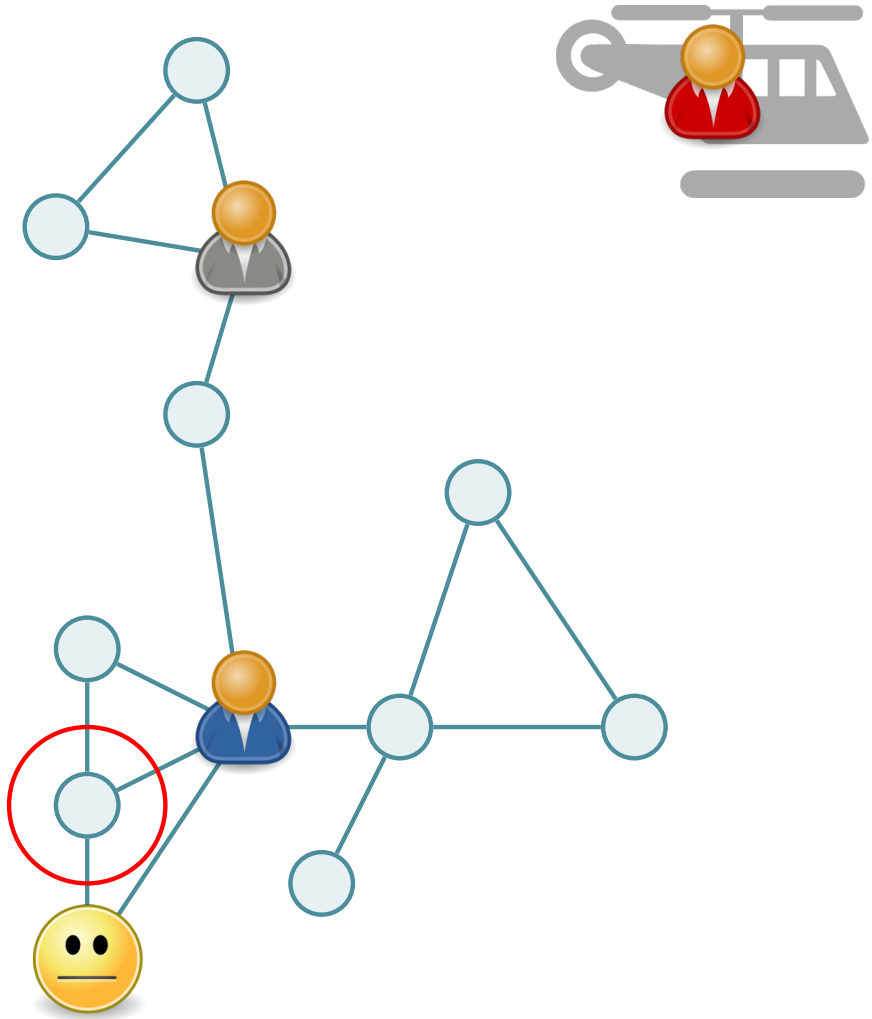
Das Räuber-und-Gendarmen-Spiel
Drei Gendarmen reichen.



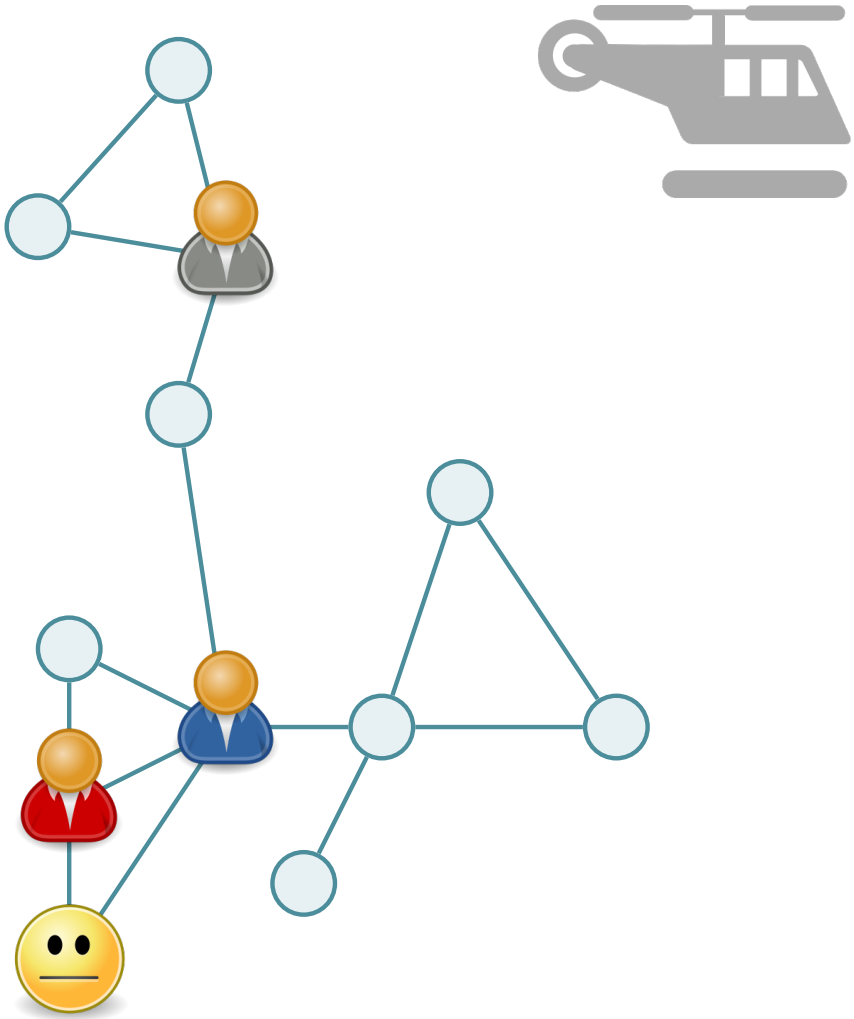
Das Räuber-und-Gendarmen-Spiel
Drei Gendarmen reichen.



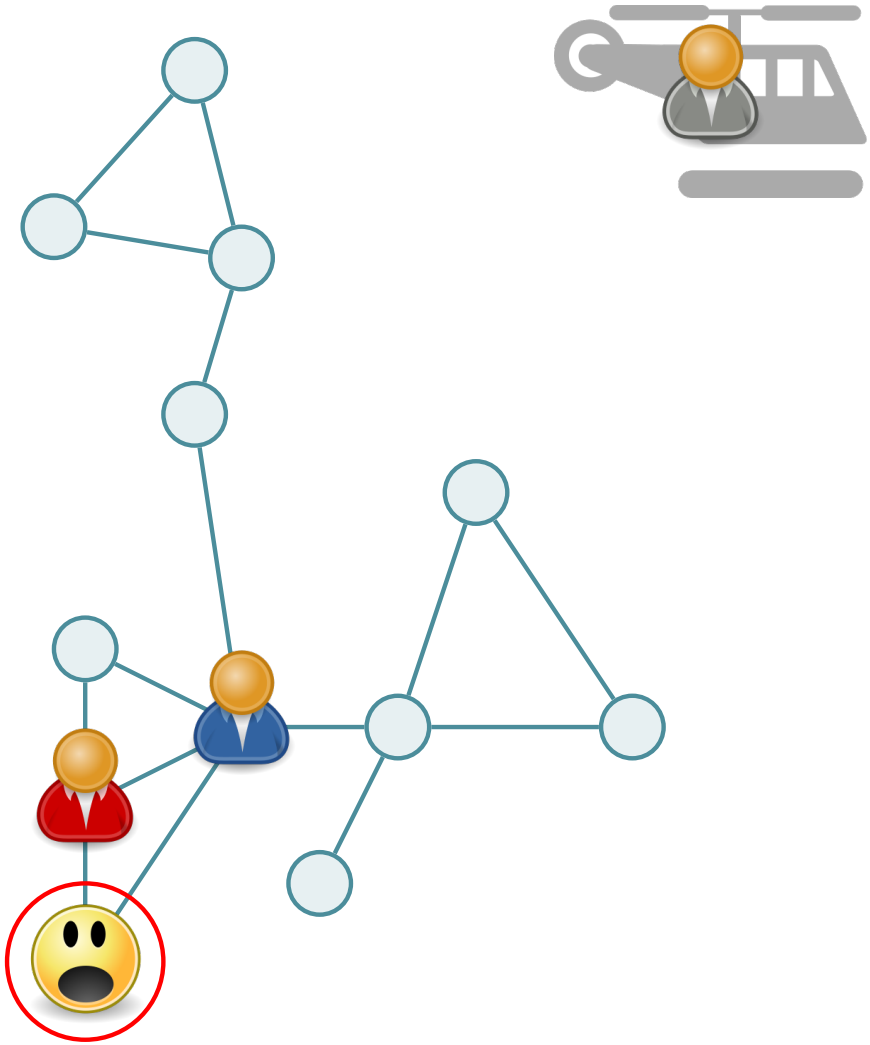
Das Räuber-und-Gendarmen-Spiel
Drei Gendarmen reichen.



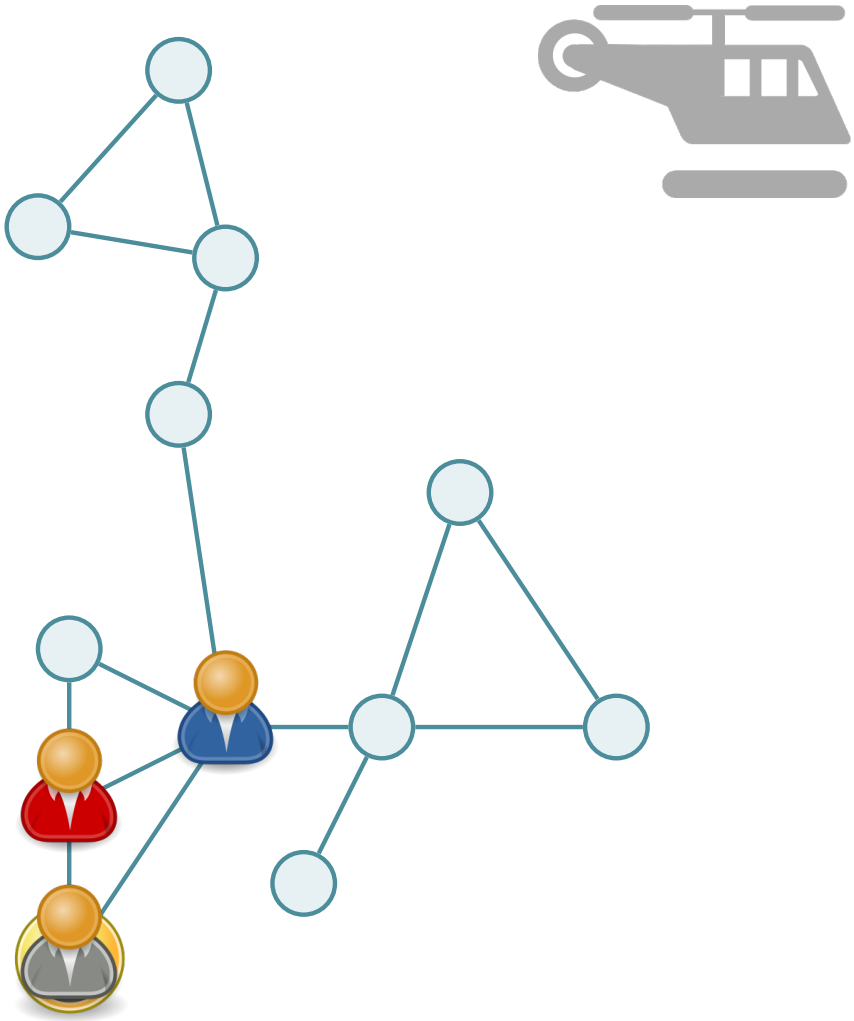
Das Räuber-und-Gendarmen-Spiel
Drei Gendarmen reichen.



Das Räuber-und-Gendarmen-Spiel
Drei Gendarmen reichen.



Das Räuber-und-Gendarmen-Spiel
Drei Gendarmen reichen.

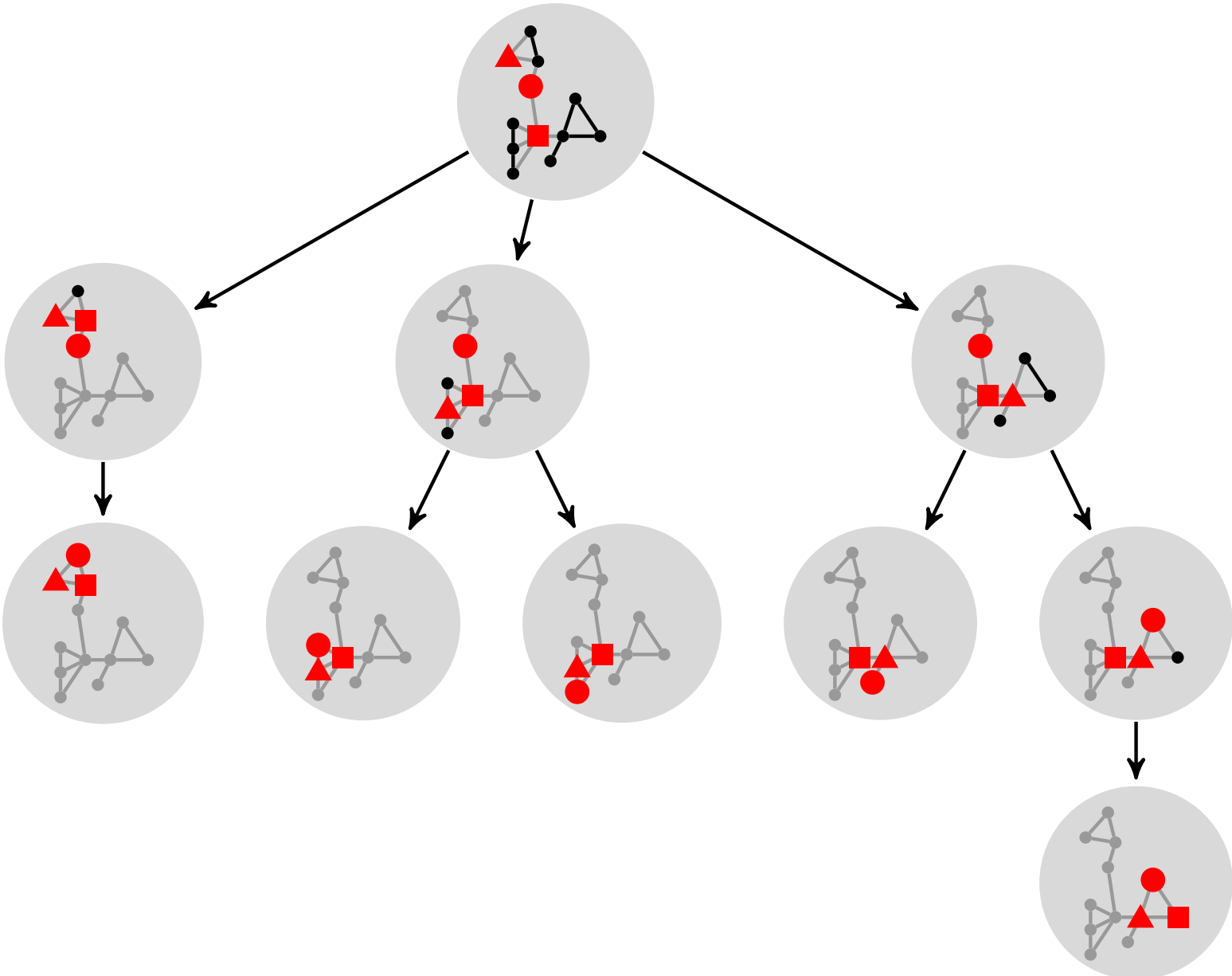


Strategie der Gendarmen = Baumzerlegung

Eine *Gewinnstrategie der Gendarmen* lässt sich als *Baum* darstellen:

- Jeder Baumknoten ist eine Menge von Knotenpositionen, an denen sich Gendarmen befinden.
- Die *Wurzel* ist die Knotenmenge der Startpositionen.
- Die *Kinder* eines Baumknoten entsprechen den *Teilgraphen, in denen sich der Räuber befinden könnte*.

Beispiel einer Baumzerlegung der Weite 2



Wieso helfen Baumzerlegungen beim 3-Färbbarkeitsproblem?

Satz

Für einen Graphen G sei eine Baumzerlegung der Weite k mit n Knoten gegeben. Dann lässt sich in Zeit

$$O(3^k n)$$

entscheiden, ob der Graph 3-färbbar ist.

Wieso helfen Baumzerlegungen beim 3-Färbbarkeitsproblem?

Satz

Für einen Graphen G sei eine Baumzerlegung der Weite k mit n Knoten gegeben. Dann lässt sich in Zeit

$$O(3^k n)$$

entscheiden, ob der Graph 3-färbbar ist.

Satz

Für einen Graphen G sei eine Baumzerlegung der Weite k mit n Knoten gegeben. Dann lässt sich in Zeit

$$O(2^k n)$$

die größte unabhängige Menge von G berechnen.

Gliederung

Worum es geht

- Klassische algorithmische Metatheoreme . . .
- . . . versus neue Varianten für die Komplexitätstheorie

Neue Varianten 1: Platzkomplexität

- Algorithmische Metatheoreme für Platzklassen
- Anwendung: Zyklen gerader Länge
- Anwendung: Pseudopolynomialzeit-Algorithmen

Neue Varianten 2: Schaltkreise

- Algorithmische Metatheoreme für Schaltkreisklassen
- Anwendung: Visible-Pushdown-Sprachen

Warum wir *Meta* theoreme brauchen.

Theorem 4.4
Each of the following problems is in NC, when restricted to graphs with treewidth $\leq K$, for constant K: vertex cover [GT1], dominating set [GT2], domatic number [GT3], chromatic number [GT4], monochromatic triangle [GT5], feedback vertex set [GT7], feedback arc set [GT8], partial feedback edge set [GT9], minimum maximal matching [GT10], partition into triangles [GT11], partition into isomorphic subgraphs for fixed H [GT12], partition into Hamiltonian subgraphs [GT13], partition into forests [GT14], partition into cliques [GT15], partition into perfect matchings [GT16], clique [GT19], independent set [GT20], induced subgraph with property P (for monadic second order properties P) [GT21], induced connected subgraph with property P (for monadic second order properties P) [GT22], induced path [GT23], balanced complete bipartite subgraph [GT24], bipartite subgraph [GT25], degree bounded connected subgraph for fixed d [GT26], planar subgraph [GT27], transitive subgraph [GT29], uniconnected subgraph [GT30], minimum k-connected subgraph for fixed k [GT31], cubic subgraph [GT32], minimum equivalent digraph [GT33], Hamiltonian completion [GT34], Hamiltonian circuit [GT37], directed Hamiltonian circuit [GT38], Hamiltonian path (and directed Hamiltonian path) [GT39], subgraph isomorphism for fixed H, subgraph isomorphism for connected H with bounded valence [GT 48], graph contractability for fixed H [GT51], graph homomorphism for fixed H [GT52], path with forbidden pairs for fixed n [GT54], multiple choice matching for fixed J [GT55], graph grundy numbering for graphs with bounded valence [GT56], kernel [GT57], k-closure [GT58], path distinguishers [GT60], degree constrained spanning tree [ND1], maximum leaf spanning tree [ND2], bounded diameter spanning tree [ND3], k'th best spanning tree for fixed k [ND9], bounded component spanning forest for fixed k [ND10], multiple choice branching for fixed m [ND11], Steiner tree in graphs [ND12], max cut [ND16], minimum cut into bounded sets [ND17], rural postman [ND27], longest circuit [ND28], longest path [ND29], shortest weight-constrained path [ND30], k'th shortest path for fixed k [ND31], disjoint connecting paths for fixed k [ND40], maximum length-bounded disjoint paths for fixed J [ND41], maximum fixed-length disjoint paths for fixed J [ND42], chordal graph completion for fixed k, chromatix index, spanning tree parity problem, distance d chromatic number for fixed d and k, thickness $\leq k$ for fixed k, membership for each class C of graphs, which is closed unded minor taking.

Was haben all diese Probleme gemeinsam?

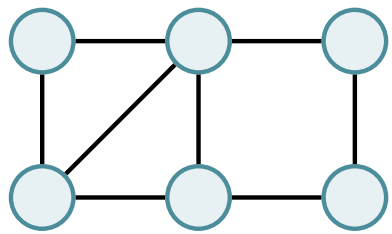
Courcelle erkannte 1990, dass all diese Probleme sich in *monadischer Logik zweiter Stufe* (MSO-Logik) beschreiben lassen.

Die Beschreibung von Grapheigenschaften mittels Logik.

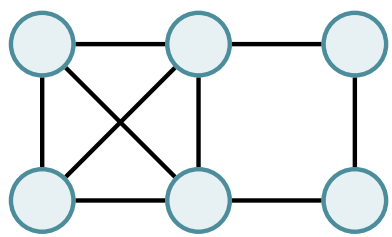
Eine typische *monadische zweitstufige* Formel ist $\varphi_{\text{3-colorable}} =$

$$\exists R\, \exists G\, \exists B\, \forall x (R(x) \vee G(x) \vee B(x)) \wedge$$
$$\forall x \forall y \big(E(x,y) \rightarrow$$
$$\neg(R(x) \wedge R(y)) \wedge \neg(G(x) \wedge G(y)) \wedge \neg(B(x) \wedge B(y))\big)$$

Sie »beschreibt« 3-Färbbarkeit:



$\models \varphi_{\text{3-colorable}}$



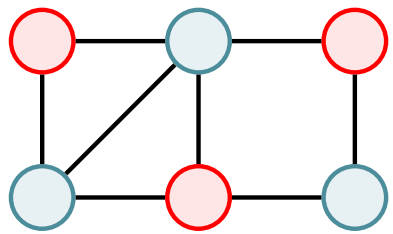
$\not\models \varphi_{\text{3-colorable}}$

Die Beschreibung von Grapheigenschaften mittels Logik.

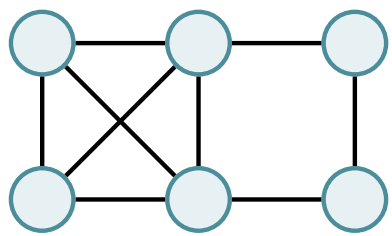
Eine typische *monadische zweitstufige* Formel ist $\varphi_{\text{3-colorable}} =$

$$\exists R \, \exists G \, \exists B \, \forall x (R(x) \vee G(x) \vee B(x)) \wedge$$
$$\forall x \forall y \big(E(x,y) \rightarrow$$
$$\neg(R(x) \wedge R(y)) \wedge \neg(G(x) \wedge G(y)) \wedge \neg(B(x) \wedge B(y))\big)$$

Sie »beschreibt« 3-Färbbarkeit:



$\models \varphi_{\text{3-colorable}}$



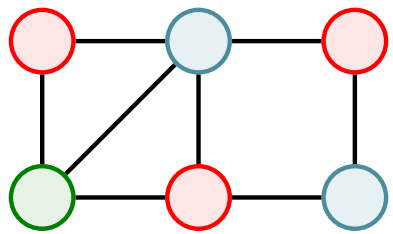
$\not\models \varphi_{\text{3-colorable}}$

Die Beschreibung von Grapheigenschaften mittels Logik.

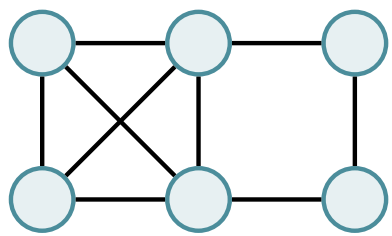
Eine typische *monadische zweitstufige* Formel ist $\varphi_{\text{3-colorable}} =$

$$\exists R\, \exists G\, \exists B\, \forall x (R(x) \vee G(x) \vee B(x)) \wedge$$
$$\forall x \forall y \big(E(x,y) \rightarrow$$
$$\neg(R(x) \wedge R(y)) \wedge \neg(G(x) \wedge G(y)) \wedge \neg(B(x) \wedge B(y))\big)$$

Sie »beschreibt« 3-Färbbarkeit:



$\models \varphi_{\text{3-colorable}}$



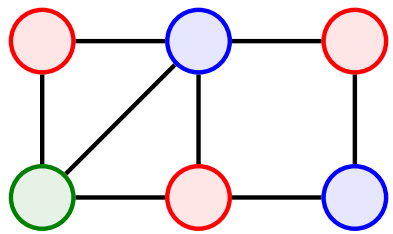
$\not\models \varphi_{\text{3-colorable}}$

Die Beschreibung von Grapheigenschaften mittels Logik.

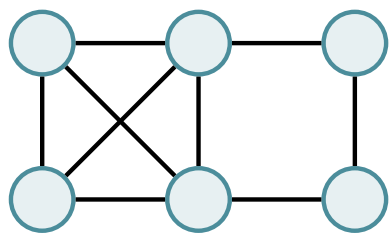
Eine typische *monadische zweitstufige* Formel ist $\varphi_{\text{3-colorable}} =$

$$\exists R\, \exists G\, \exists B\, \forall x (R(x) \vee G(x) \vee B(x)) \wedge$$
$$\forall x \forall y \big(E(x,y) \rightarrow$$
$$\neg(R(x) \wedge R(y)) \wedge \neg(G(x) \wedge G(y)) \wedge \neg(B(x) \wedge B(y))\big)$$

Sie »beschreibt« 3-Färbbarkeit:



$\models \varphi_{\text{3-colorable}}$



$\not\models \varphi_{\text{3-colorable}}$

Das erste algorithmische Metatheorem.

Satz (Courcelle)

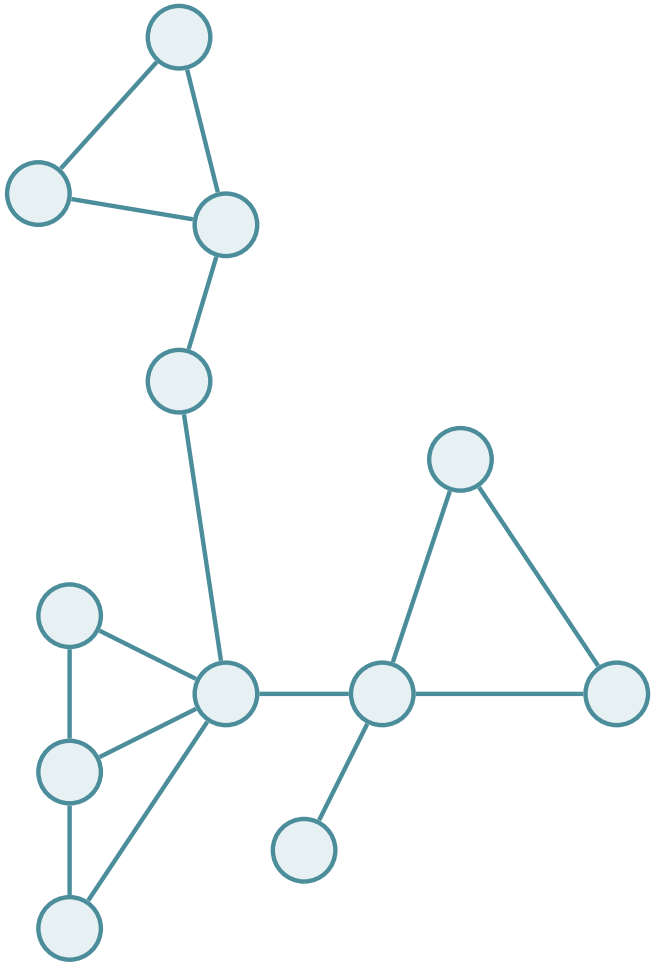
Sei φ eine MSO-Formel und k eine Zahl. Dann ist

$$\{G \mid G \models \varphi \text{ und } G \text{ hat Baumweite höchstens } k\}$$

in linearer Zeit entscheidbar.

Das erste algorithmische Metatheorem.

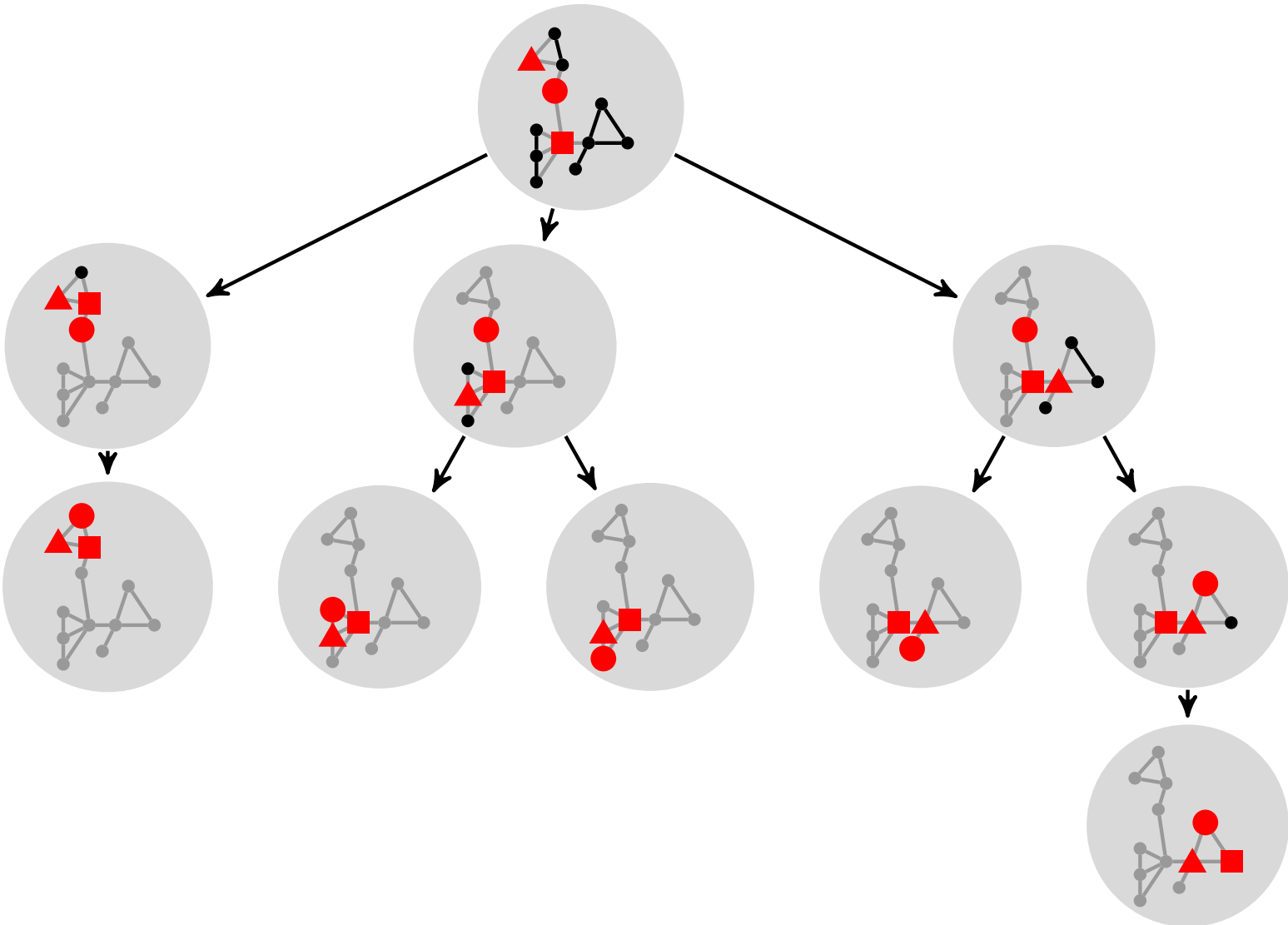
Beweisschritt 1: Die Ausgangsfrage.



$\models \varphi_{\text{3-colorable}}$?

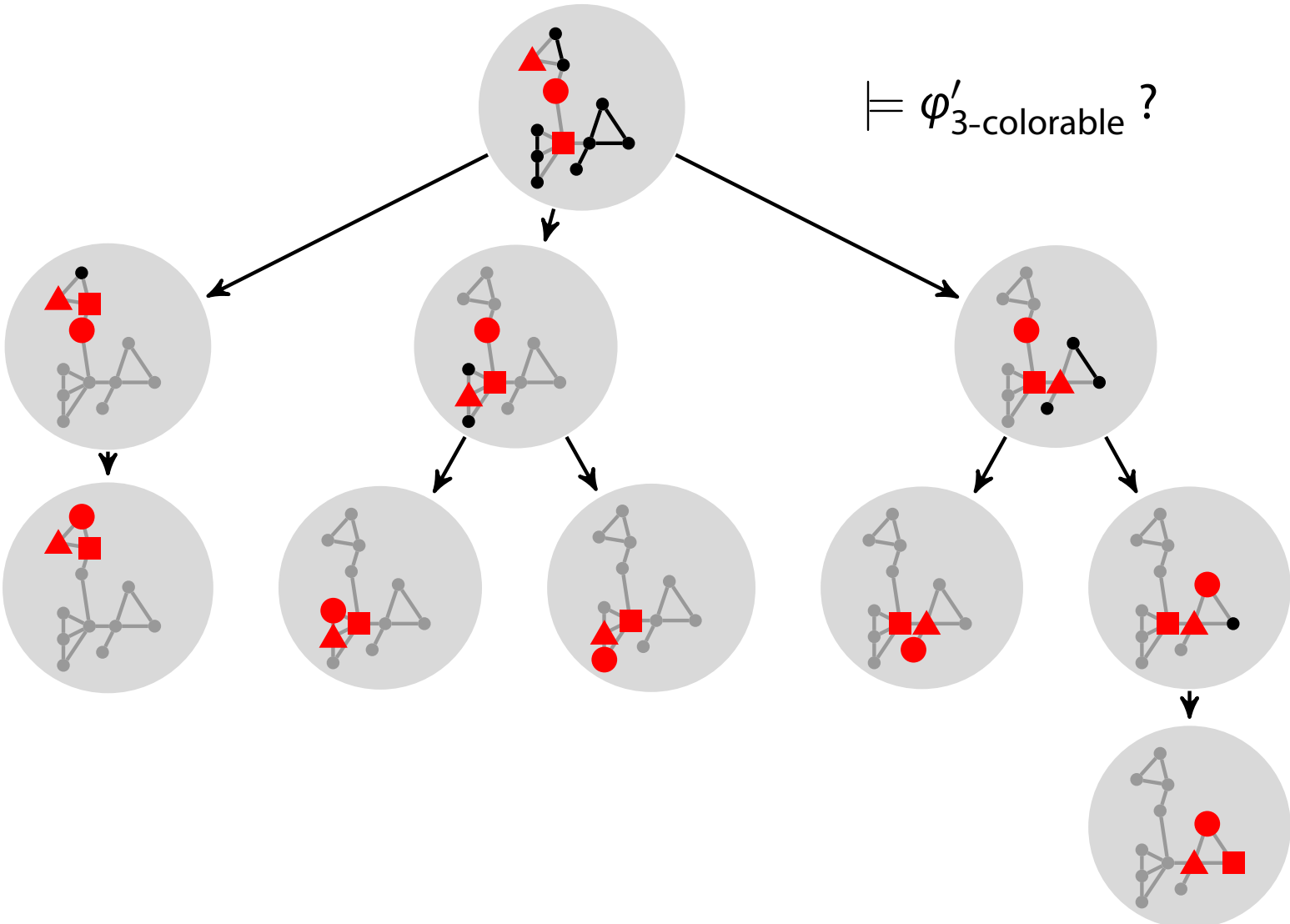
Das erste algorithmische Metatheorem.

Beweisschritt 2: Berechnung einer Baumzerlegung in linearer Zeit.



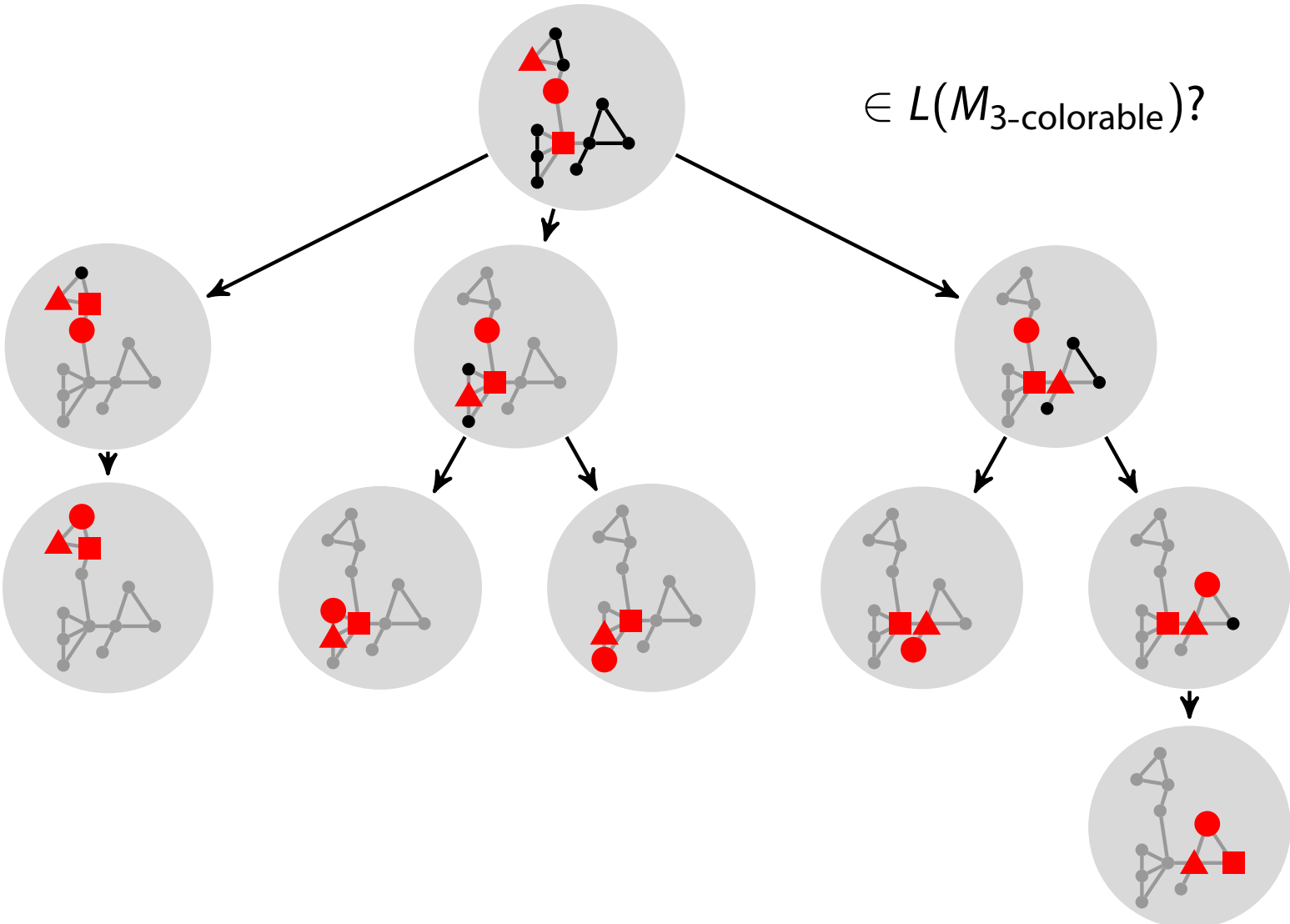
Das erste algorithmische Metatheorem.

Beweisschritt 3: Übertragung der Formel auf den Baum:



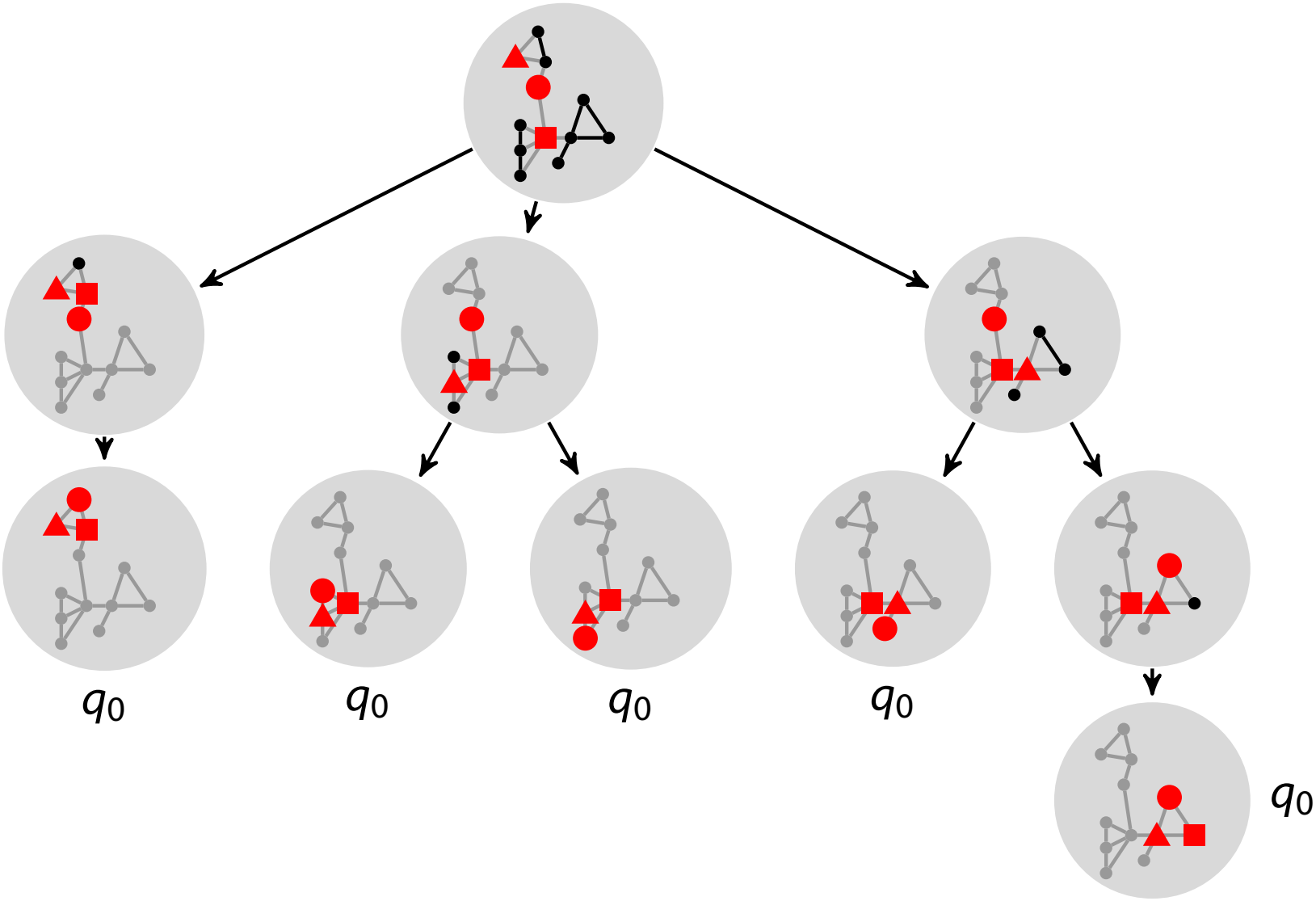
Das erste algorithmische Metatheorem.

Beweisschritt 4: Umwandlung der Formel in einen Baumautomaten:



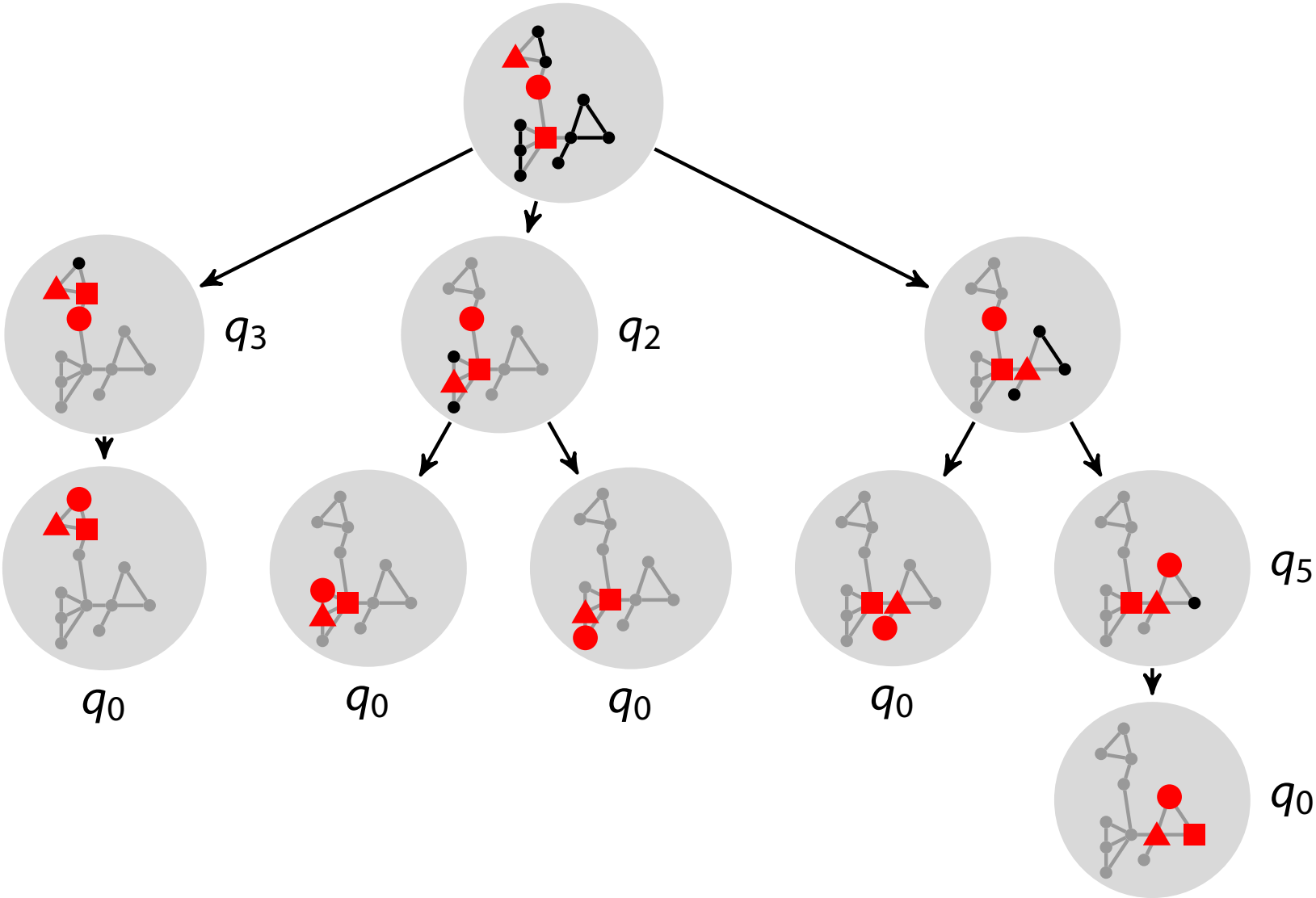
Das erste algorithmische Metatheorem.

Beweisschritt 5: Auswertung des Automaten in linearer Zeit.



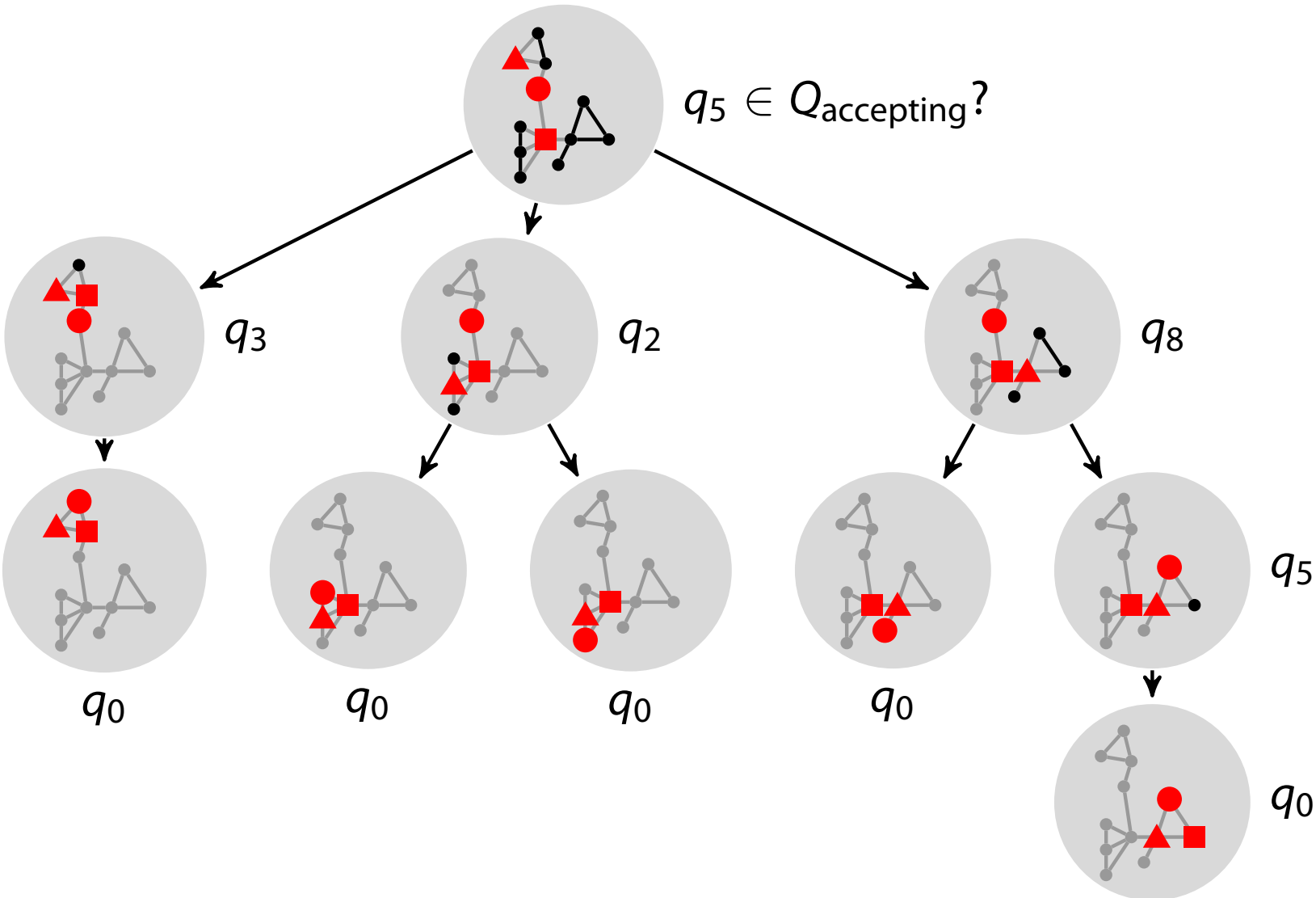
Das erste algorithmische Metatheorem.

Beweisschritt 5: Auswertung des Automaten in linearer Zeit.



Das erste algorithmische Metatheorem.

Beweisschritt 5: Auswertung des Automaten in linearer Zeit.



Die allgemeine Bauart von algorithmischen Metatheoremen.

Satz (Allgemeine Bauart von Metatheoremen)

Wenn

- *die Problemstellung in einer bestimmten Logik beschreibbar ist*
- *und die Eingaben eine bestimmte Zerlegungseigenschaft haben,*

dann

- *gibt es eine bestimmte Art von Algorithmus.*

Die allgemeine Bauart von algorithmischen Metatheoremen.

Satz (Courcelle)

Wenn

- *die Problemstellung MSO-beschreibbar ist*
- *und die Eingaben Baumweite k haben,*

dann

- *gibt es einen Linearzeitalgorithmus.*

Die allgemeine Bauart von algorithmischen Metatheoremen.

Satz (Bodlaender, Courcelle)

Wenn

- *die Problemstellung MSO-beschreibbar ist*
- *und die Eingaben Baumweite k haben,*

dann

- *gibt es einen parallelen Algorithmus mit Laufzeit $O(\log n)$.*

Die allgemeine Bauart von algorithmischen Metatheoremen.

Satz (Courcelle, Makowsky, Rotics)

Wenn

- *die Problemstellung MSO-beschreibbar ist*
- *und die Eingaben Cliqueweite k haben,*

dann

- *gibt es einen Polynomialzeitalgorithmus.*

Die allgemeine Bauart von algorithmischen Metatheoremen.

Satz (Frick, Grohe)

Wenn

- *die Problemstellung FO-beschreibbar ist*
- *und die Eingaben planar sind,*

dann

- *gibt es einen Linearzeitalgorithmus.*

Die allgemeine Bauart von algorithmischen Metatheoremen.

Satz (Flum, Grohe)

Wenn

- *die Problemstellung FO-beschreibbar ist*
- *und die Eingaben enthalten einen bestimmten Minor nicht,*

dann

- *gibt es einen Polynomialzeitalgorithmus.*

Gliederung

Worum es geht

- Klassische algorithmische Metatheoreme . . .
- . . . versus neue Varianten für die Komplexitätstheorie

Neue Varianten 1: Platzkomplexität

- Algorithmische Metatheoreme für Platzklassen
- Anwendung: Zyklen gerader Länge
- Anwendung: Pseudopolynomialzeit-Algorithmen

Neue Varianten 2: Schaltkreise

- Algorithmische Metatheoreme für Schaltkreisklassen
- Anwendung: Visible-Pushdown-Sprachen

Die komplexitätstheoretische Sicht der Dinge

Leitfragen

- Wie *schwierig* ist 3-Färbbarkeit auf Graphen beschränkter Baumweite?
- Ist das Problem für eine Klasse *vollständig*?
- Wenn ja, für welche?

Was wir schon wissen

- Die algorithmischen Metatheoreme liefern *scharfe obere Schranken* für die Zeitkomplexität. . .
- . . . aber man konnte *nicht* Vollständigkeit für diese Klassen zeigen.

Algorithmische Metatheoreme 2.0

Seit 2010 konnten wir *neue* algorithmischen Metatheoreme zeigen:

1. Sie machen Aussagen über *neue Ressourcen* wie *Speicherplatz* oder *Schaltkreistiefe*.
2. Mit ihnen lassen sich die Klassen identifizieren, für die die Problem *vollständig* sind.
3. Mit ihnen lassen sich alte Resultate neu elegant und kurz beweisen.
4. Sie haben Anwendungen, die weit über Graphen mit beschränkter Baumweite hinausgehen.

Gliederung

Worum es geht

- Klassische algorithmische Metatheoreme . . .
- . . . versus neue Varianten für die Komplexitätstheorie

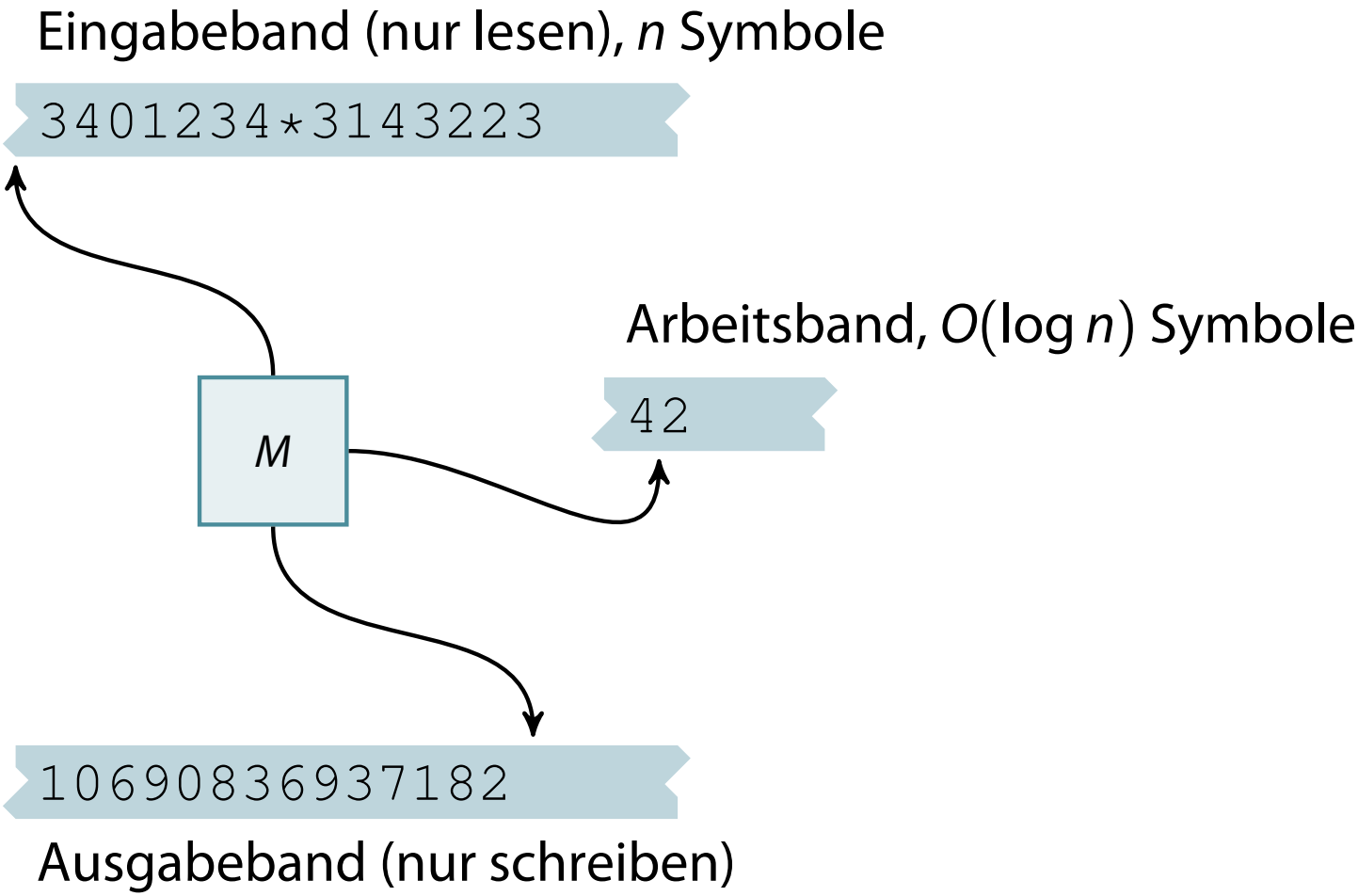
Neue Varianten 1: Platzkomplexität

- Algorithmische Metatheoreme für Platzklassen
- Anwendung: Zyklen gerader Länge
- Anwendung: Pseudopolynomialzeit-Algorithmen

Neue Varianten 2: Schaltkreise

- Algorithmische Metatheoreme für Schaltkreisklassen
- Anwendung: Visible-Pushdown-Sprachen

Die Bänder einer Logspace-Maschine.



Die Logspace-Version des Satzes von Courcelle.

Satz (Elberfeld, T, Jakoby, 2010)

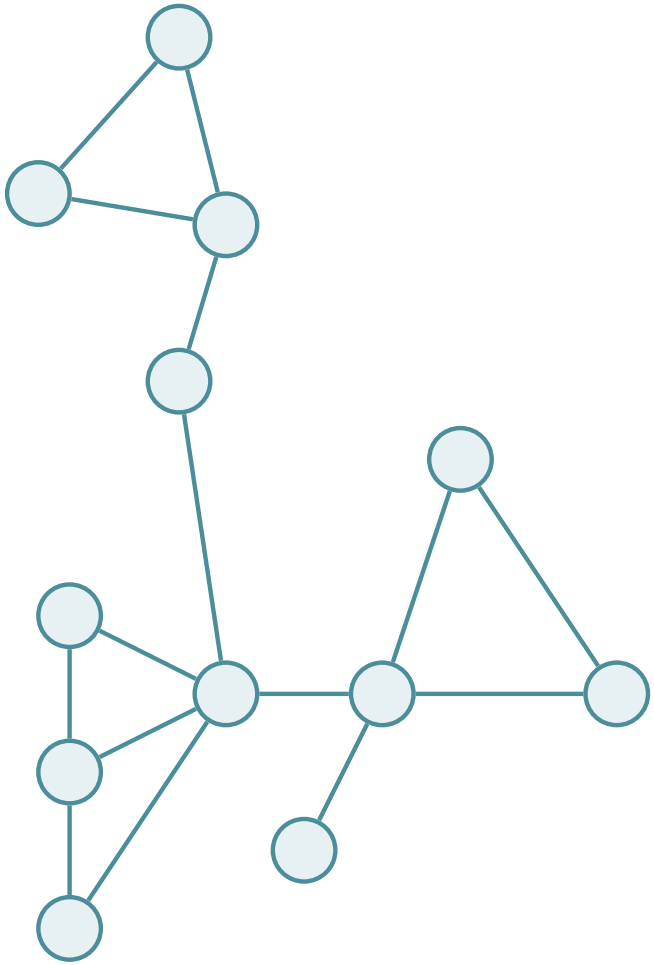
Sei φ eine MSO-Formel und k eine Zahl. Dann ist

$$\{G \mid G \models \varphi \text{ und } G \text{ hat Baumweite höchstens } k\}$$

in logarithmischem Platz entscheidbar.

Die Logspace-Version des Satzes von Courcelle.

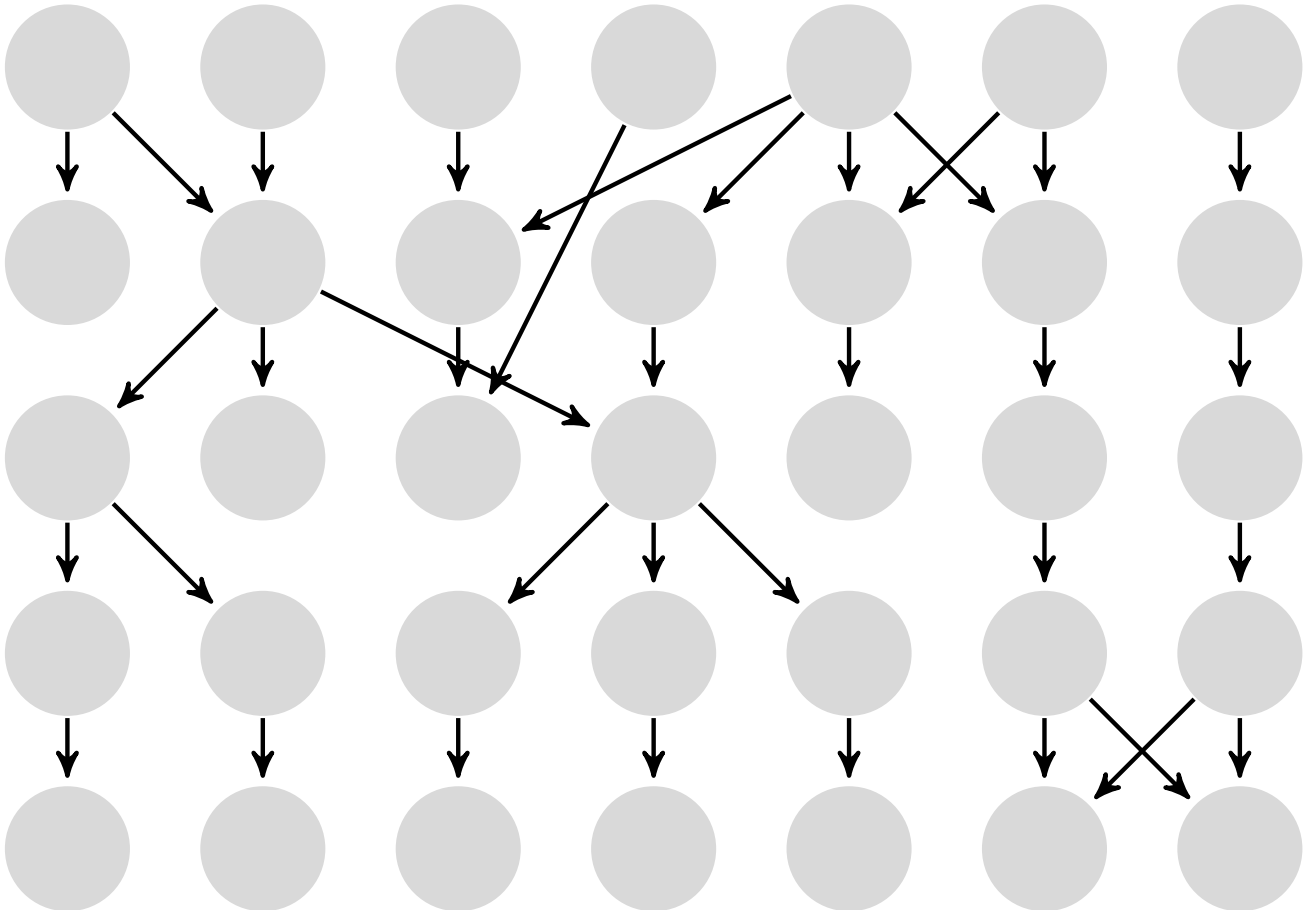
Beweisschritt 1: Die Ausgangsfrage.



$\models \varphi_{\text{3-colorable}}$?

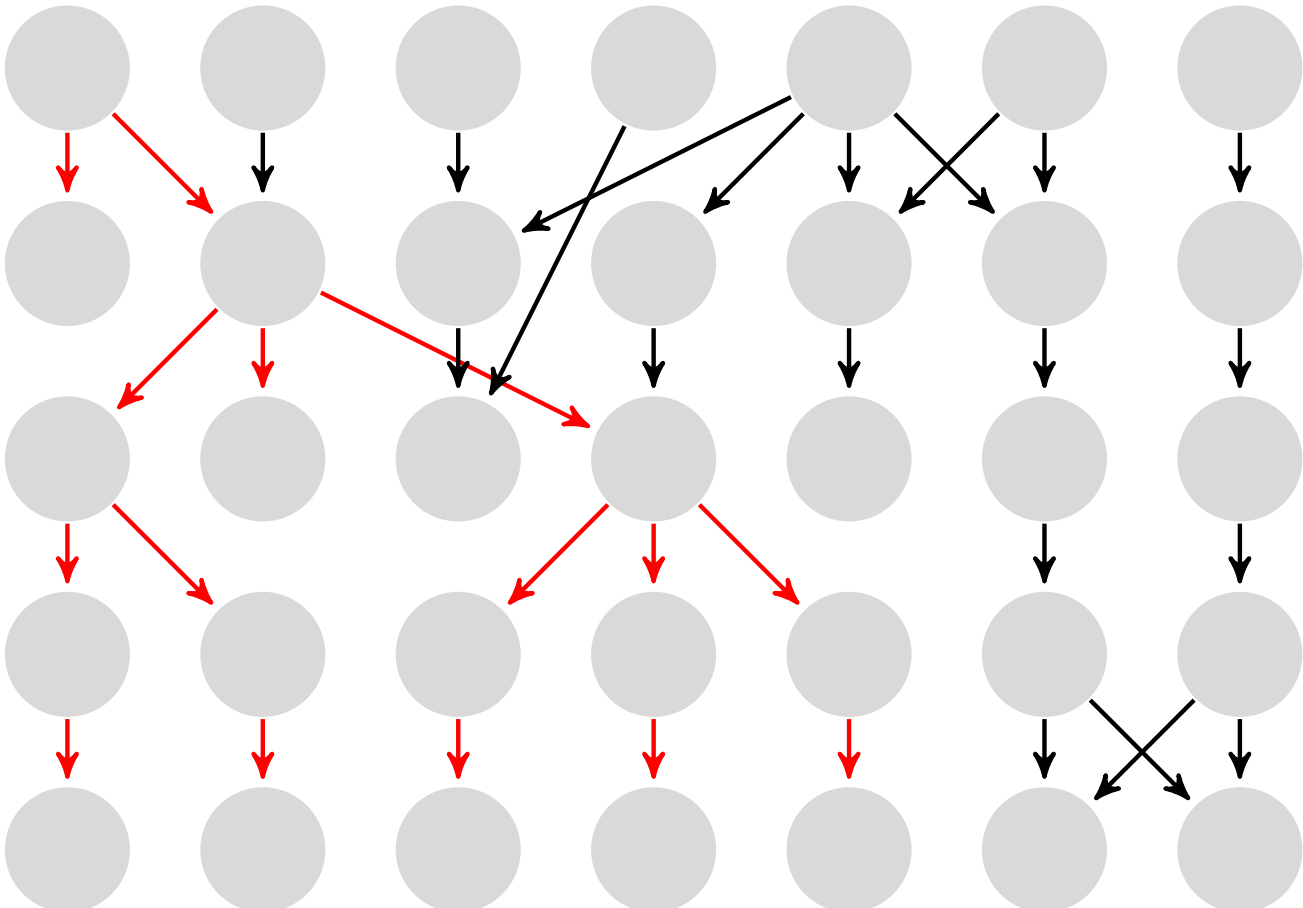
Die Logspace-Version des Satzes von Courcelle.

Beweisschritt 2: Berechnung einer Baumzerlegung in logarithmischem Platz.



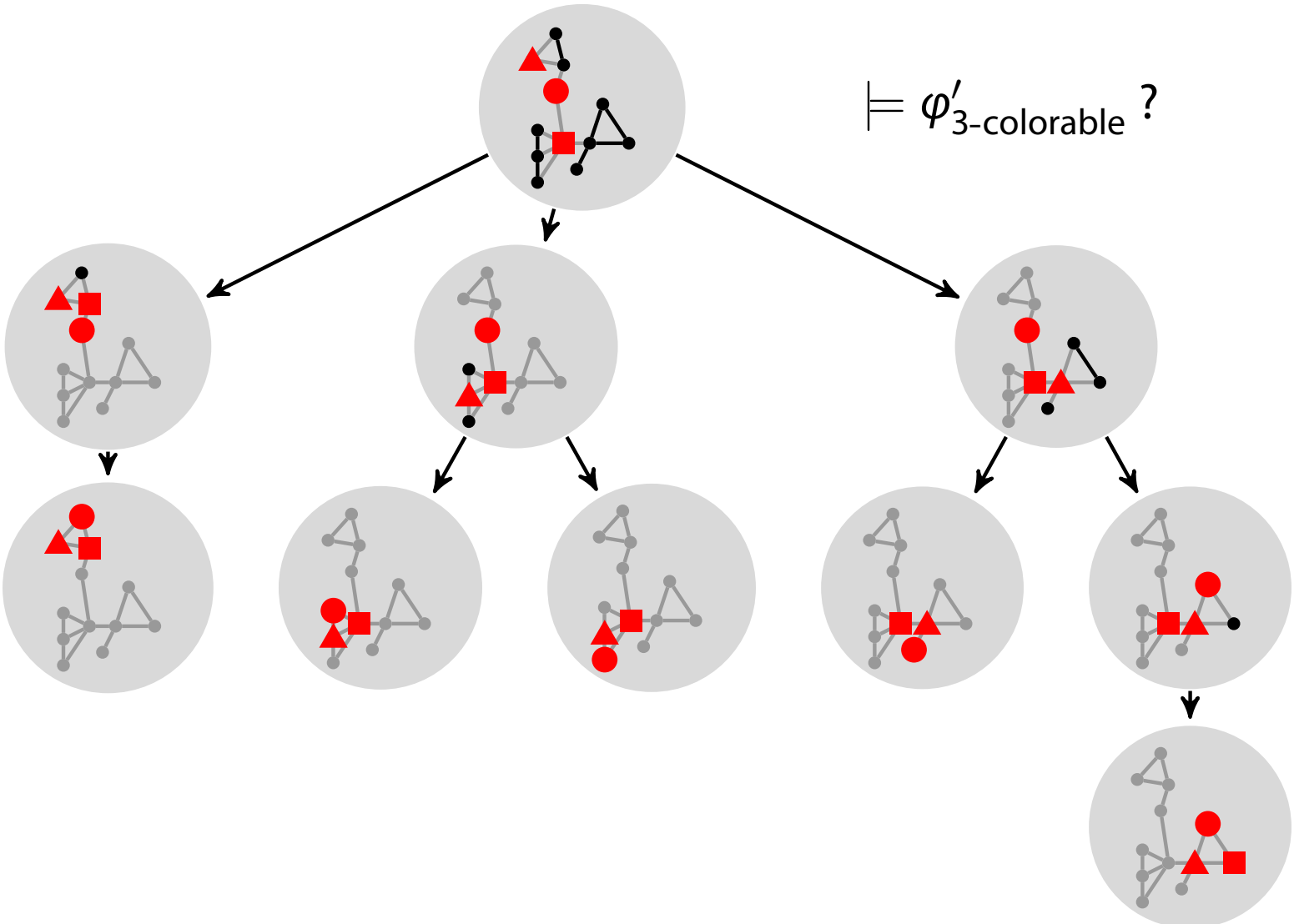
Die Logspace-Version des Satzes von Courcelle.

Beweisschritt 2: Berechnung einer Baumzerlegung in logarithmischem Platz.



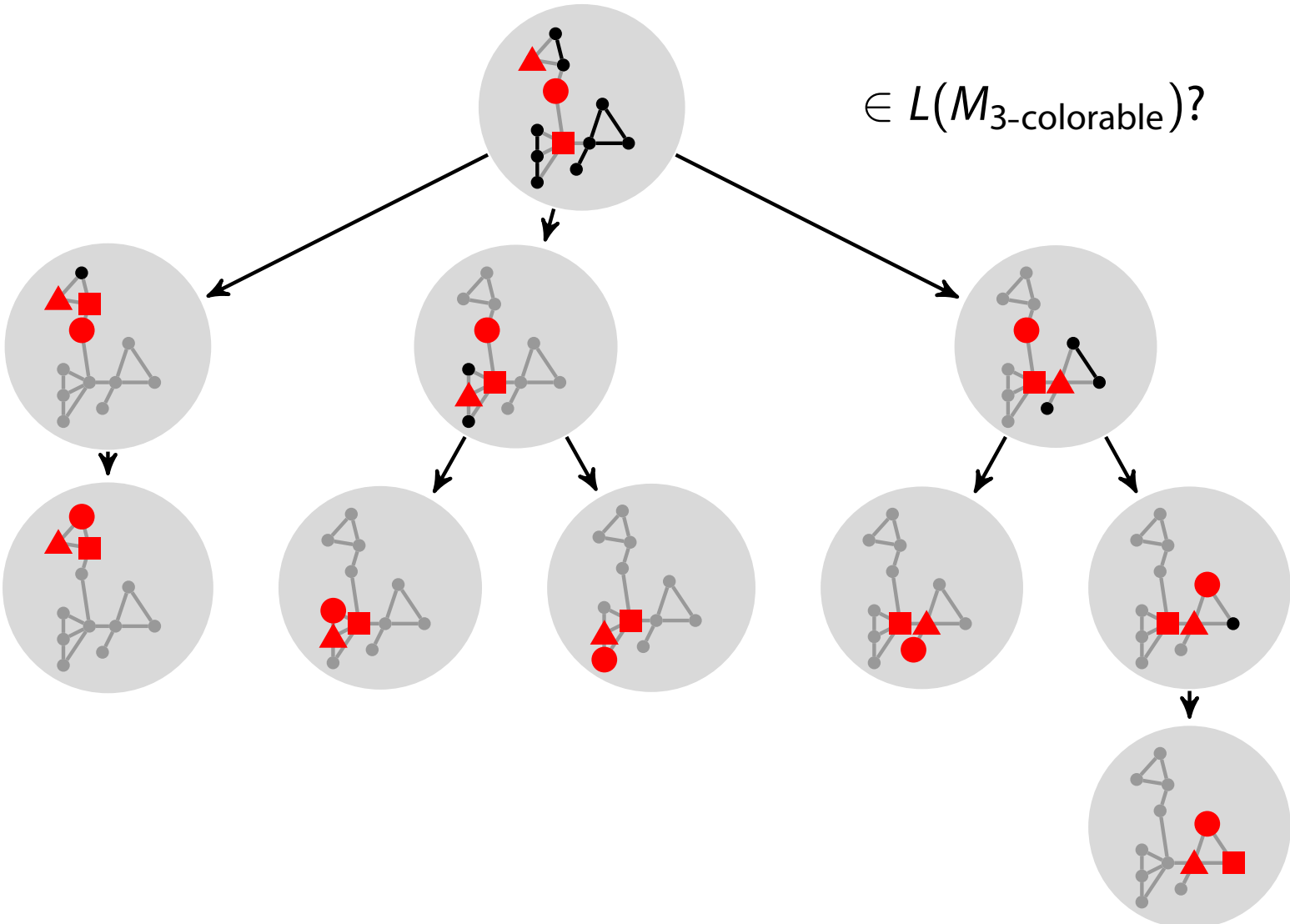
Die Logspace-Version des Satzes von Courcelle.

Beweisschritt 3: Übertragung der Formel auf den Baum:



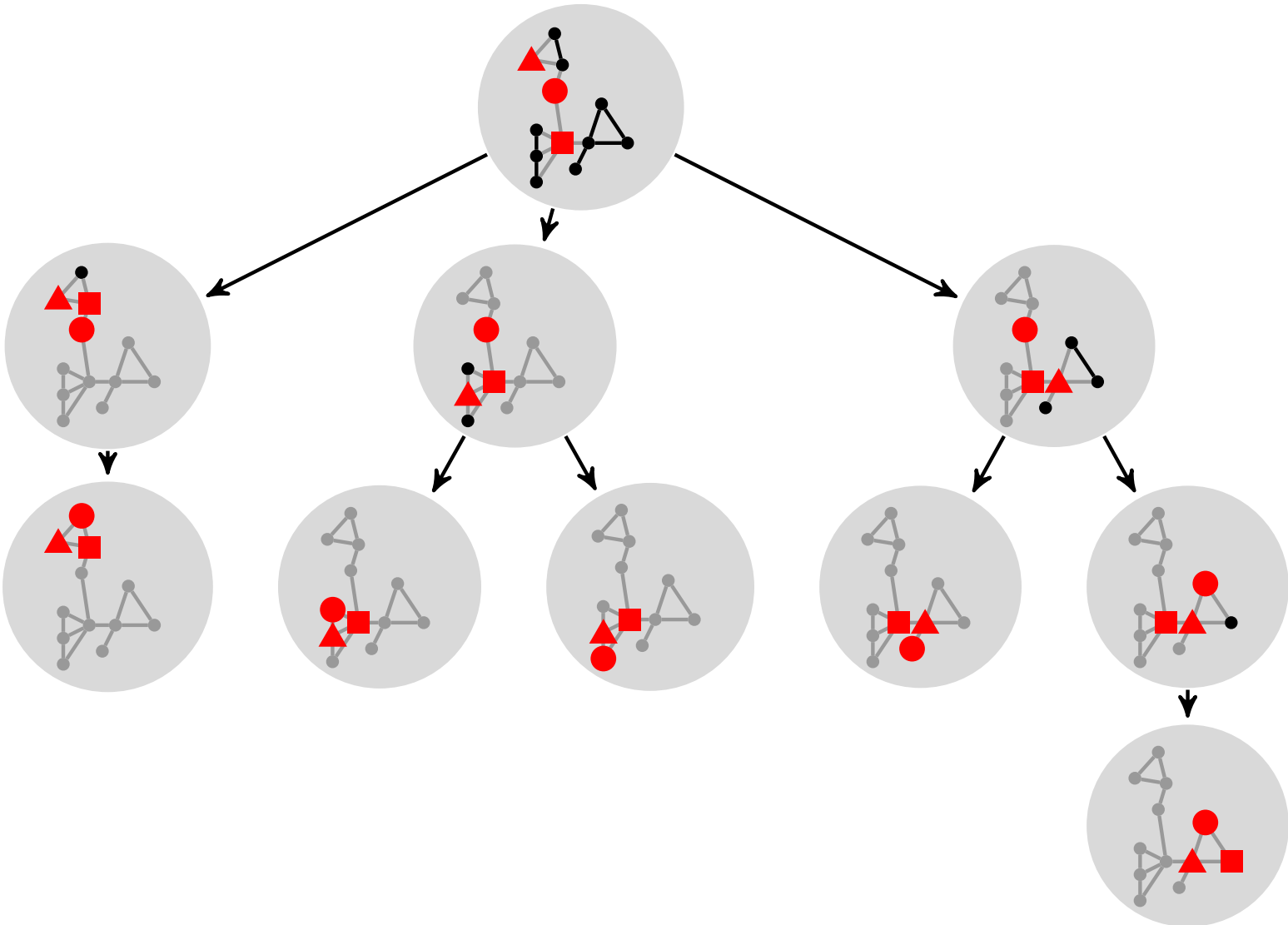
Die Logspace-Version des Satzes von Courcelle.

Beweisschritt 4: Umwandlung der Formel in einen Baumautomaten:



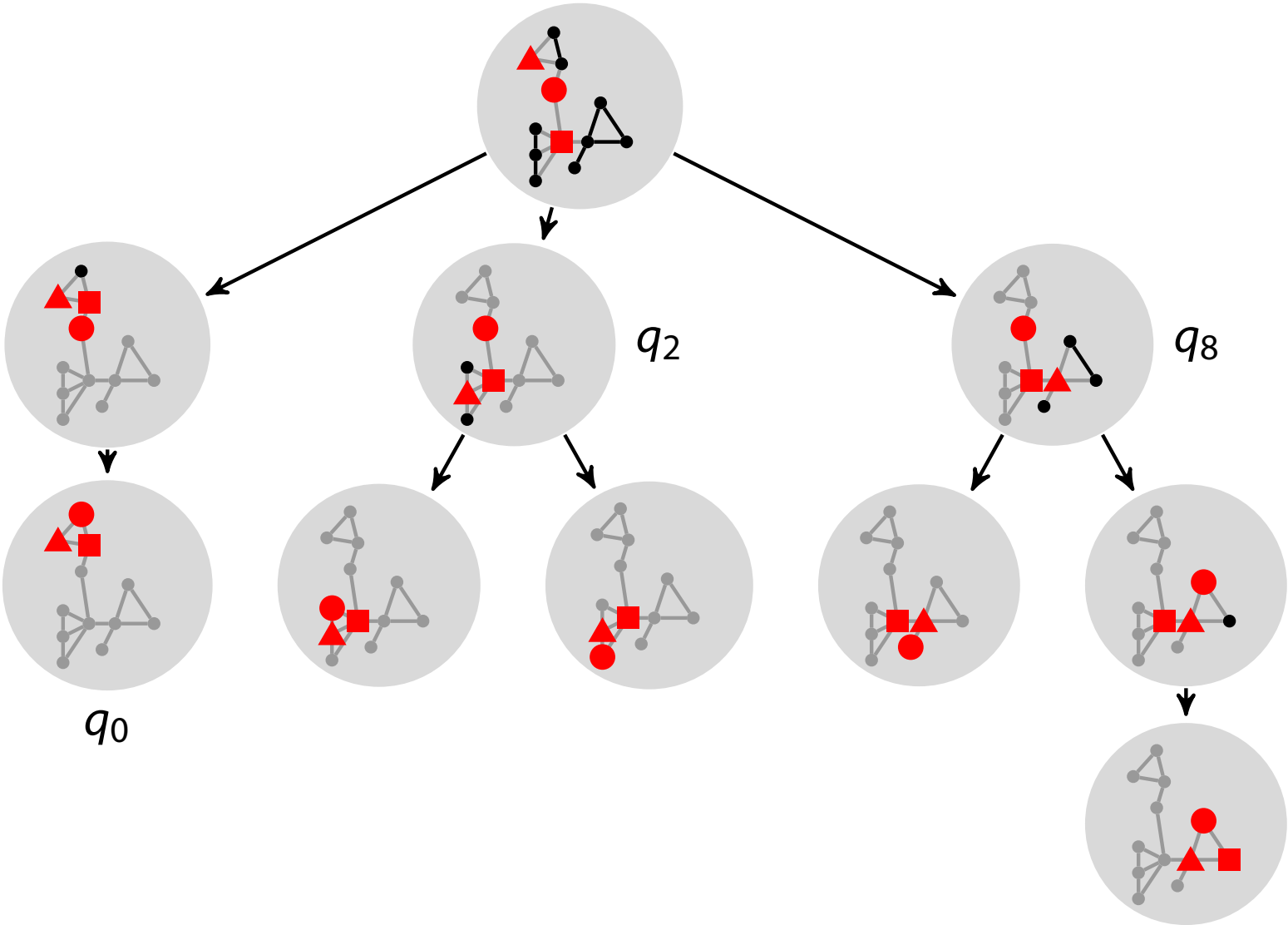
Die Logspace-Version des Satzes von Courcelle.

Beweisschritt 5: Auswertung des Automaten in logarithmischem Platz.



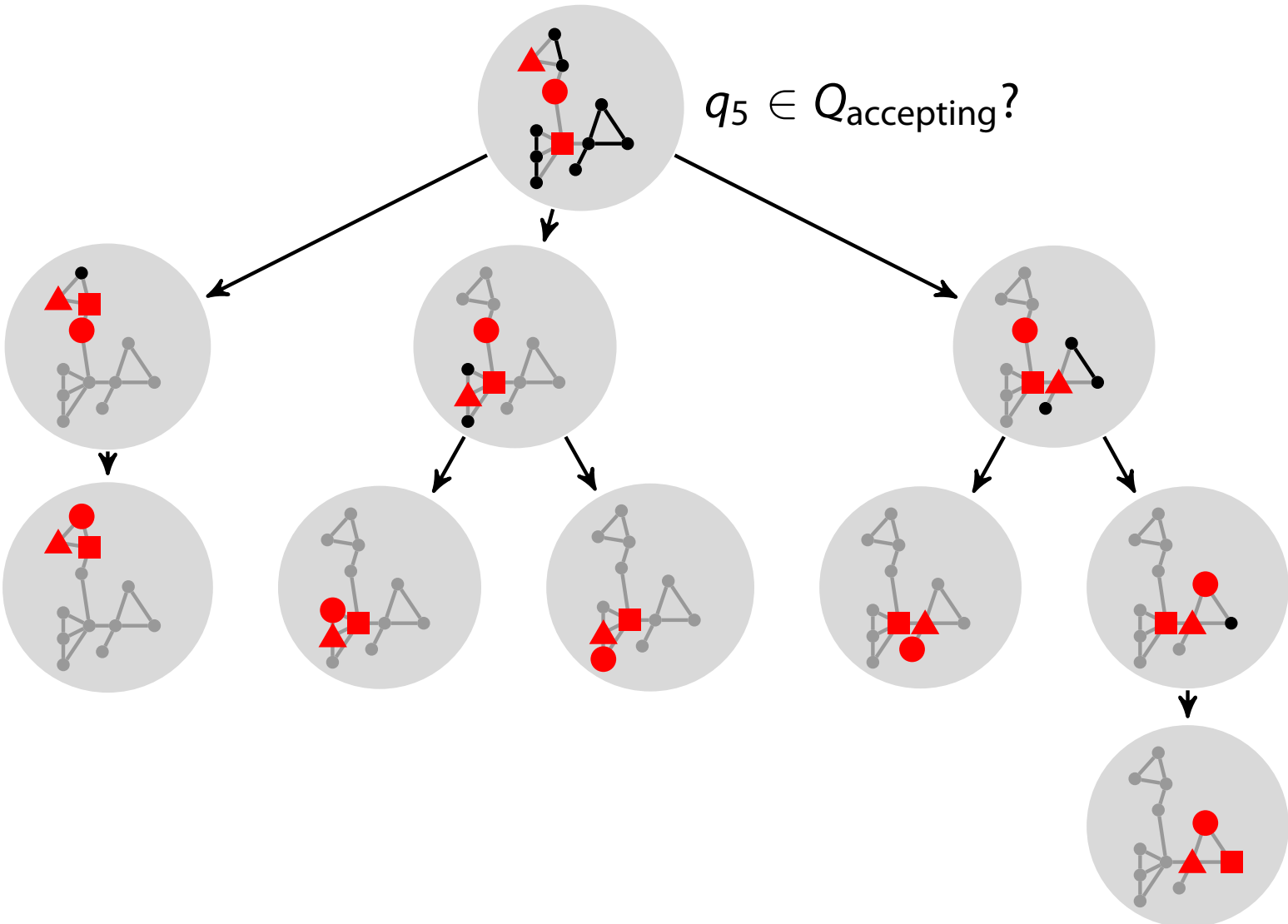
Die Logspace-Version des Satzes von Courcelle.

Beweisschritt 5: Auswertung des Automaten in logarithmischem Platz.



Die Logspace-Version des Satzes von Courcelle.

Beweisschritt 5: Auswertung des Automaten in logarithmischem Platz.



Niedrig hängenden Früchte.

Folgerung

Für jedes k gilt $\{G \mid G \text{ ist 3-färbbar und hat Baumweite höchstens } k\} \in \mathsf{L}$.

Folgerung

Für jedes k gilt

$$\{G \mid G \text{ hat ein perfektes Matching und Baumweite höchstens } k\} \in \mathsf{L}.$$

Niedrig hängenden Früchte.

Folgerung

Für jedes k gilt

$$\{(G, s, t) \mid \textit{es gibt einen Weg von } s \textit{ nach } t \textit{ und } G \textit{ hat Baumweite höchstens } k\} \in \mathsf{L}.$$

Folgerung

Für jedes k gilt

$$\{G \mid G \textit{ enthält einen Zyklus gerader Länge und } G \textit{ hat Baumweite höchstens } k\} \in \mathsf{L}.$$

. . .

Gliederung

Höher hängende Früchte.

Algorithmische Metatheoreme benötigen, dass der Graph *zerlegbar* ist.

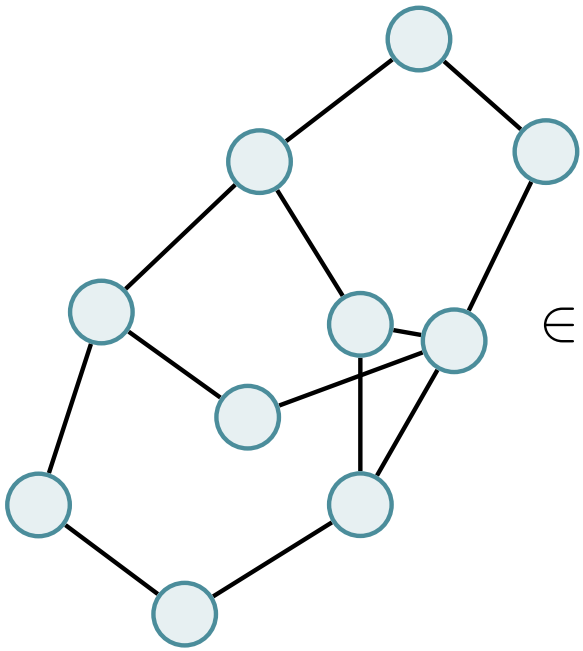
Ist dies nicht der Fall, kann man wie folgt vorgehen:

- *Hat* der Graph eine Baumzerlegung,
  wende das algorithmische Metatheorem an.
- *Anderenfalls* muss der Graph *viele Verbindungen haben*.
  Nutze dies algorithmisch.

Man kann gerade Zyklen in logarithmischem Platz finden.

Definition (Even-Cycle-Problem)

Die Sprache EVEN-CYCLE enthält alle ungerichteten Graphen G, die einen Zyklus gerader Länge enthalten.



$\in$ EVEN-CYCLE

Man kann gerade Zyklen in logarithmischem Platz finden.

Satz

EVEN-CYCLE $\in$ L.

Beweis.

1. Überprüfe, ob der Graph eine niedrige Baumweite hat.
2. Falls ja, so benutze das Metatheorem.
3. Falls nein, so gibt »ja« aus.
   Thomassen hat gezeigt, dass jeder Graph mit sehr großer Baumweite einen Zyklus gerader Länge enthält.

□

Gliederung

Ein stärkerer Satz: Berechnung der Anzahlen von Lösungen.

Satz (Elberfeld, T, Jakoby, 2010)

Sei $\varphi(X)$ eine MSO-Formel mit einer freien Variable und k eine Zahl. Dann gibt es eine Logspace-Turingmaschine M, die bei Eingabe

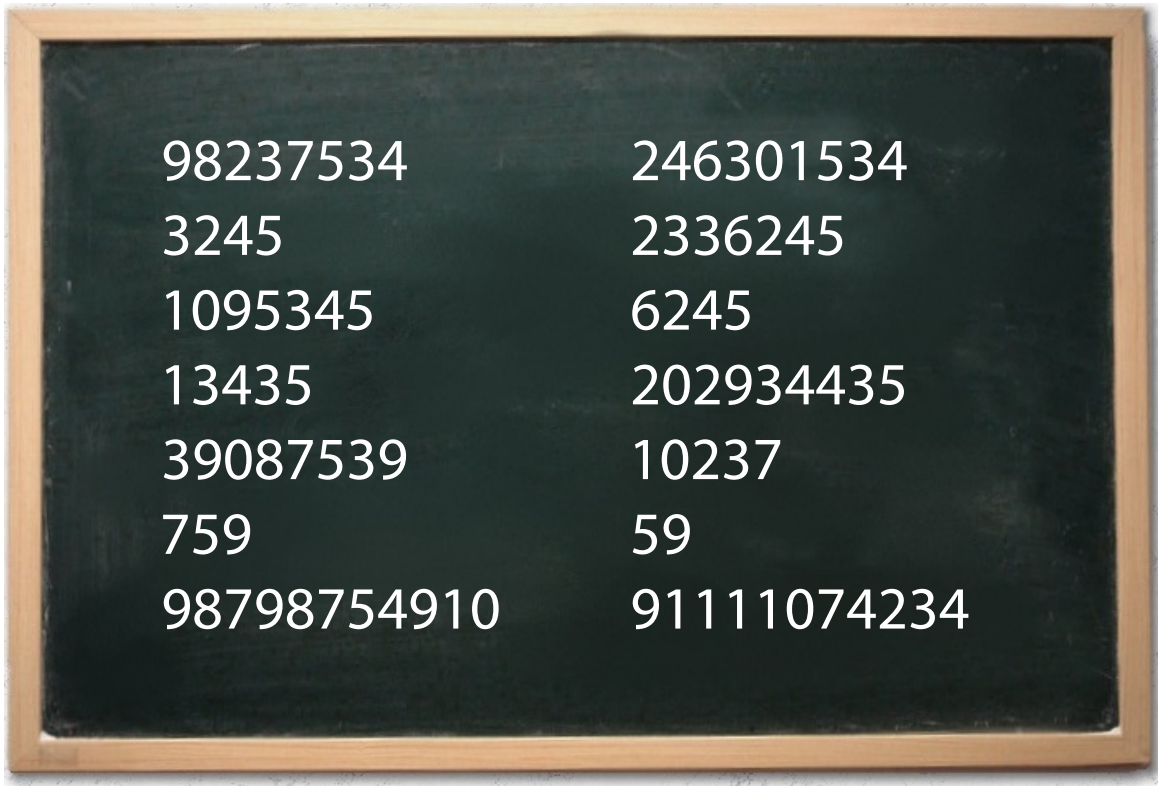
1. *eines Graphen G mit Baumweite höchstens k und*
2. *einer Zahl s*

die Anzahl an Knotenmengen S berechnet mit

1. $|S| = s$ *und*
2. $G \models \varphi(S)$.

- Beispielsweise drückt $\varphi(X) = \forall u \forall v \left[E(u,v) \rightarrow \neg(X(u) \wedge X(v)) \right]$ aus, dass X eine unabhängige Menge ist.
- M berechnet also, *wie viele unabhängige Mengen der Größe s* ein Graph hat. Dies können *exponentiell viele* sein.

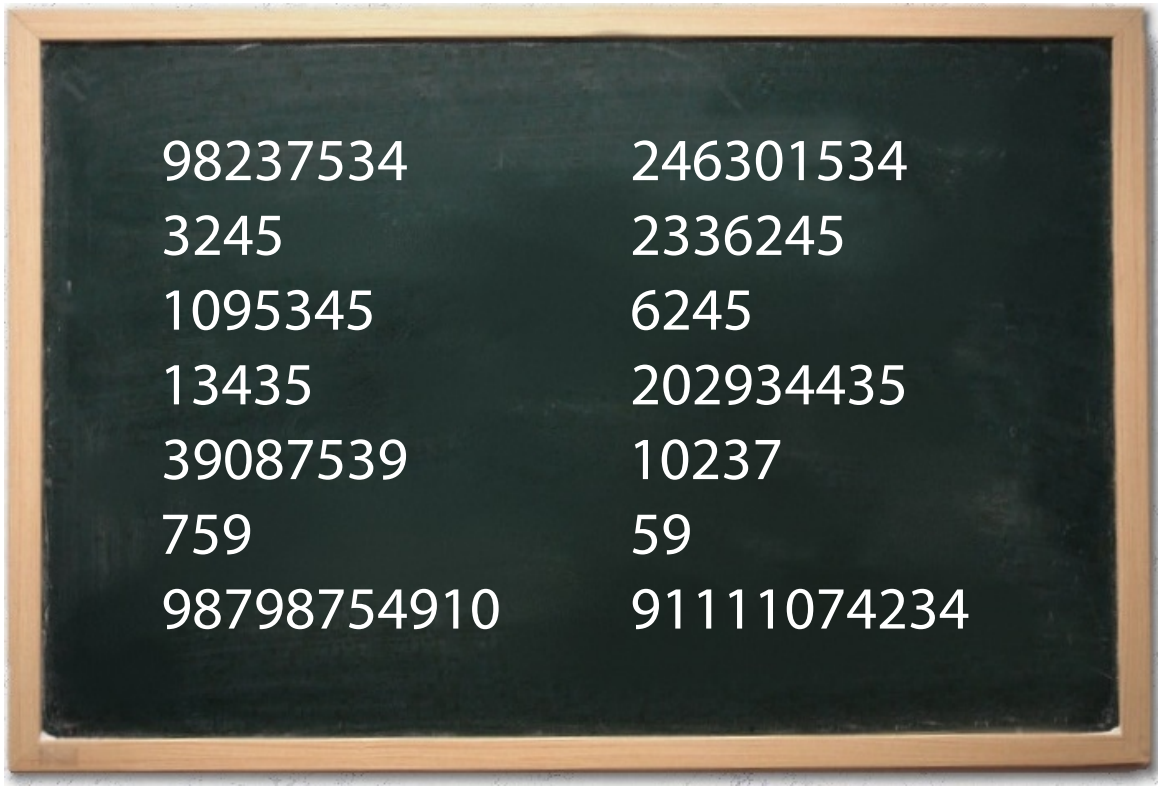
Das Subset-Sum-Problem



Das Subset-Sum-Problem

Kreise einige Zahlen ein, so dass ihre Summe genau 1000000000 ist.

Das Subset-Sum-Problem



Das Subset-Sum-Problem

Kreise einige Zahlen ein, so dass ihre Summe genau 1000000000 ist.

Dies ist ein bekanntes NP-vollständiges Problem.

Das Subset-Sum-Problem

Das unäre Subset-Sum-Problem

Kreise einige Zahlen ein, so dass ihre Summe genau 𝍸 𝍸 𝍸 𝍸 𝍸 𝍸 𝍸 𝍸 𝍸 𝍸 𝍸 ist.

Wie schwierig ist Subset-Sum-Problem?

Seien $a_1, \ldots, a_n$ die Eingabezahlen und s die Zielsumme.

Satz

Mit einem dynamischen Programm lässt sich Subset-Sum leicht lösen in

- *Zeit* $O(ns)$ *und*
- *Platz* $O(s)$.

Für große s ist Platz der *Flaschenhals*.

- Bei 1GHz Taktfrequenz, sind 4GB Speicher in *einer Sekunde* gefüllt.
- Bei $s = 10^{12}$ bräuchte man *1TB Speicher*, aber nur *6 Minuten*.

Wie schwierig ist Subset-Sum-Problem?

Seien $a_1, \ldots, a_n$ die Eingabezahlen und s die Zielsumme.

Satz

Mit einem dynamischen Programm lässt sich Subset-Sum leicht lösen in

- *Zeit $O(ns)$ und*
- *Platz $O(s)$.*

Für große s ist Platz der *Flaschenhals*.

Satz (Lokshtanov, Nederlof, 2010)

Es gibt eine algebraische Methode, die Subset-Sum in

- *Zeit $n^{O(1)}s^{O(1)}$ und*
- *Platz $n^{O(1)}$ löst.*

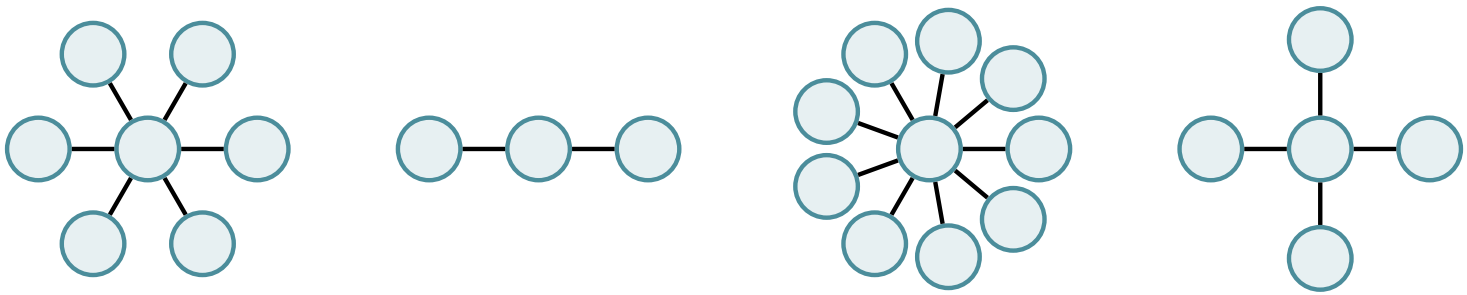
Ein platzeffizienter Pseudopolynomialzeit-Algorithmus.
Teil 1: Das unäre Problem

Satz

UNARY-SUBSET-SUM $\in$ L.

Beweis.

Für eine Eingabe wie 𝍸 II, III, 𝍸 𝍸, 𝍸 betrachte den Wald



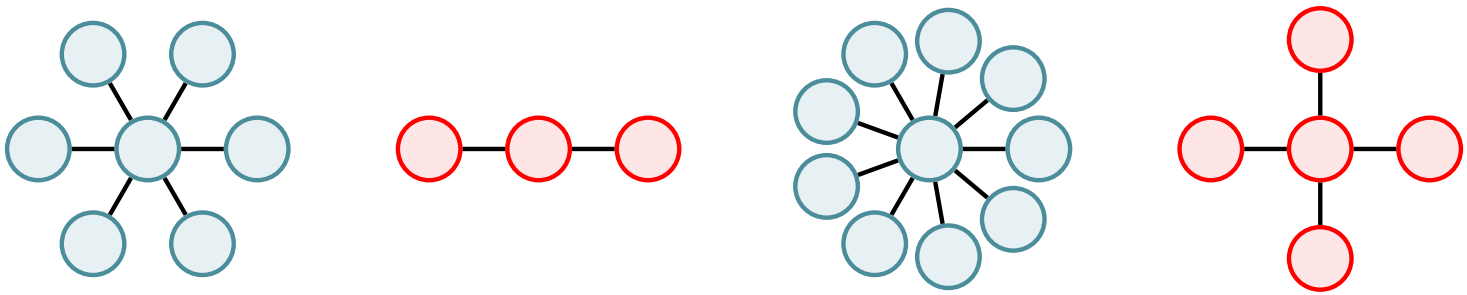
Ein platzeffizienter Pseudopolynomialzeit-Algorithmus.
Teil 1: Das unäre Problem

Satz

UNARY-SUBSET-SUM $\in$ L.

Beweis.

Für eine Eingabe wie 𝍸 II, III, 𝍸 𝍸, 𝍸 betrachte den Wald



1. Die Größe jeder Menge *S*, die unter Erreichbarkeit abgeschlossen ist, entspricht einer Teilsumme.

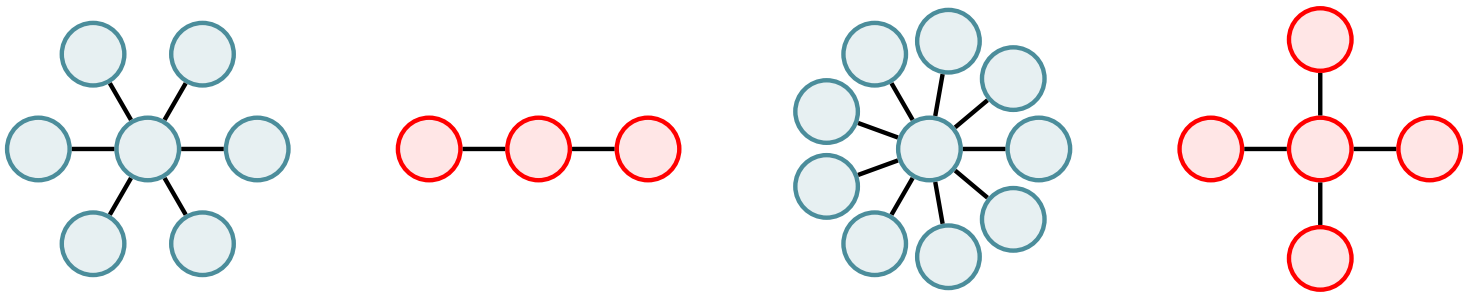
Ein platzeffizienter Pseudopolynomialzeit-Algorithmus.
Teil 1: Das unäre Problem

Satz

UNARY-SUBSET-SUM $\in$ L.

Beweis.

Für eine Eingabe wie 𝍸 II, III, 𝍸 𝍸, 𝍸 betrachte den Wald



1. Die Größe jeder Menge S, die unter Erreichbarkeit abgeschlossen ist, entspricht einer Teilsumme.
2. Das Metatheorem angewendet auf die MSO-Formel $\varphi_{\text{closed}}(X) = \forall u \forall v \left[(X(u) \wedge E(u,v)) \rightarrow X(v) \right]$ liefert die Behauptung.

□

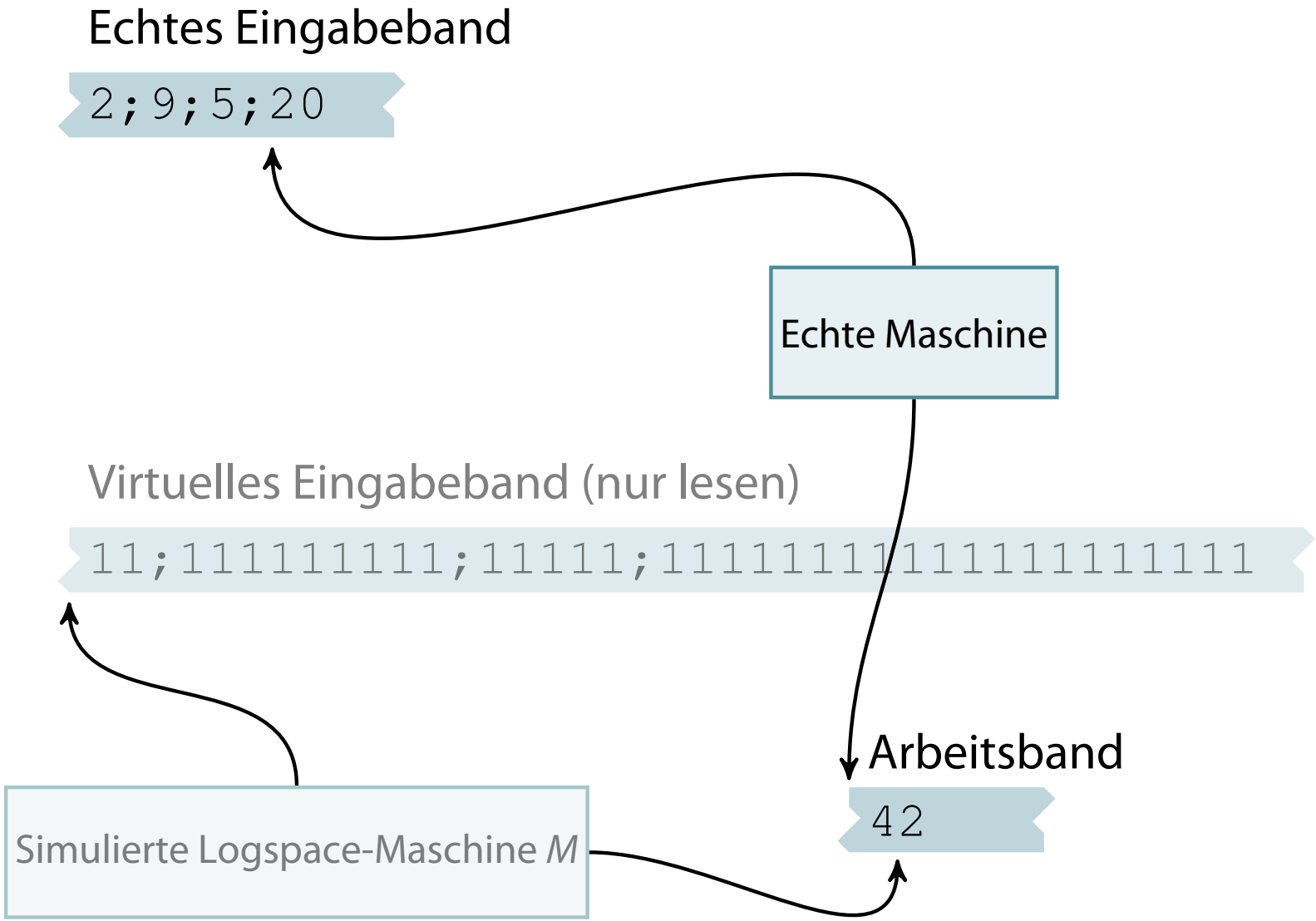
Ein platzeffizienter Pseudopolynomialzeit-Algorithmus.
Teil 2: Das binäre Problem

Folgerung

Subset-Sum lässt sich lösen in

- *Zeit* $n^{O(1)}s^{O(1)}$ *und*
- *Platz* $n^{O(1)}$.

Ein platzeffizienter Pseudopolynomialzeit-Algorithmus.
Teil 2: Das binäre Problem



Ein platzeffizienter Pseudopolynomialzeit-Algorithmus.
Teil 2: Das binäre Problem

Genauso lassen sich platzeffiziente Pseudopolynomialzeit-Algorithmen entwickeln für:

- Rucksack-Problem,
- Varianten von Bin-Packing,
- Varianten von Scheduling,
- Ganzzahlige Optimierung bei einer festen Anzahl an Ungleichungen.

Gliederung

Steckbrief der Klasse NC^1

- $REG \subseteq NC^1 \subseteq L$.
- Das Auswerten boolescher Formeln ist NC^1-vollständig.
- Baumzerlegungen können nicht in NC^1 berechnet werden, es sei denn, $NC^1 = L$.

Die NC^1-Version des Satzes von Courcelle.

Satz (Elberfeld, T, Jakoby, 2012)

Sei φ eine MSO-Formel und k eine Zahl. Dann gibt es einen NC^1-Schaltkreis, der bei

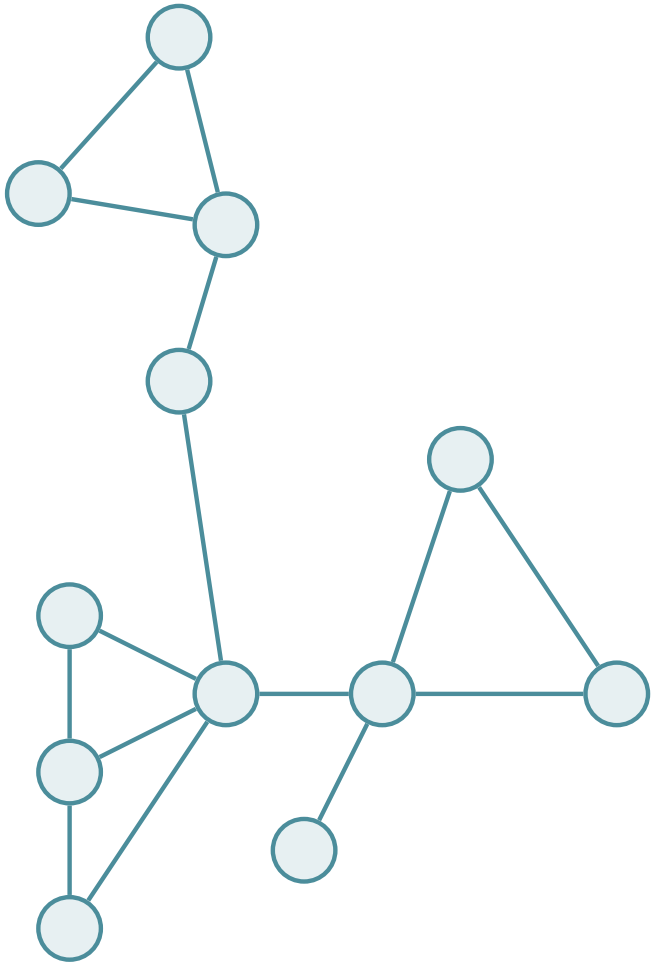
1. *Eingabe eines Graphen G*
2. *zusammen mit einer Baumzerlegung von G der Weite k*

entscheidet, ob $G \models \varphi$.

Weiterhin gilt auch die entsprechende »Zählversion« dieses Satzes.

Die NC^1-Version des Satzes von Courcelle.

Beweisschritt 1: Die Ausgangsfrage.

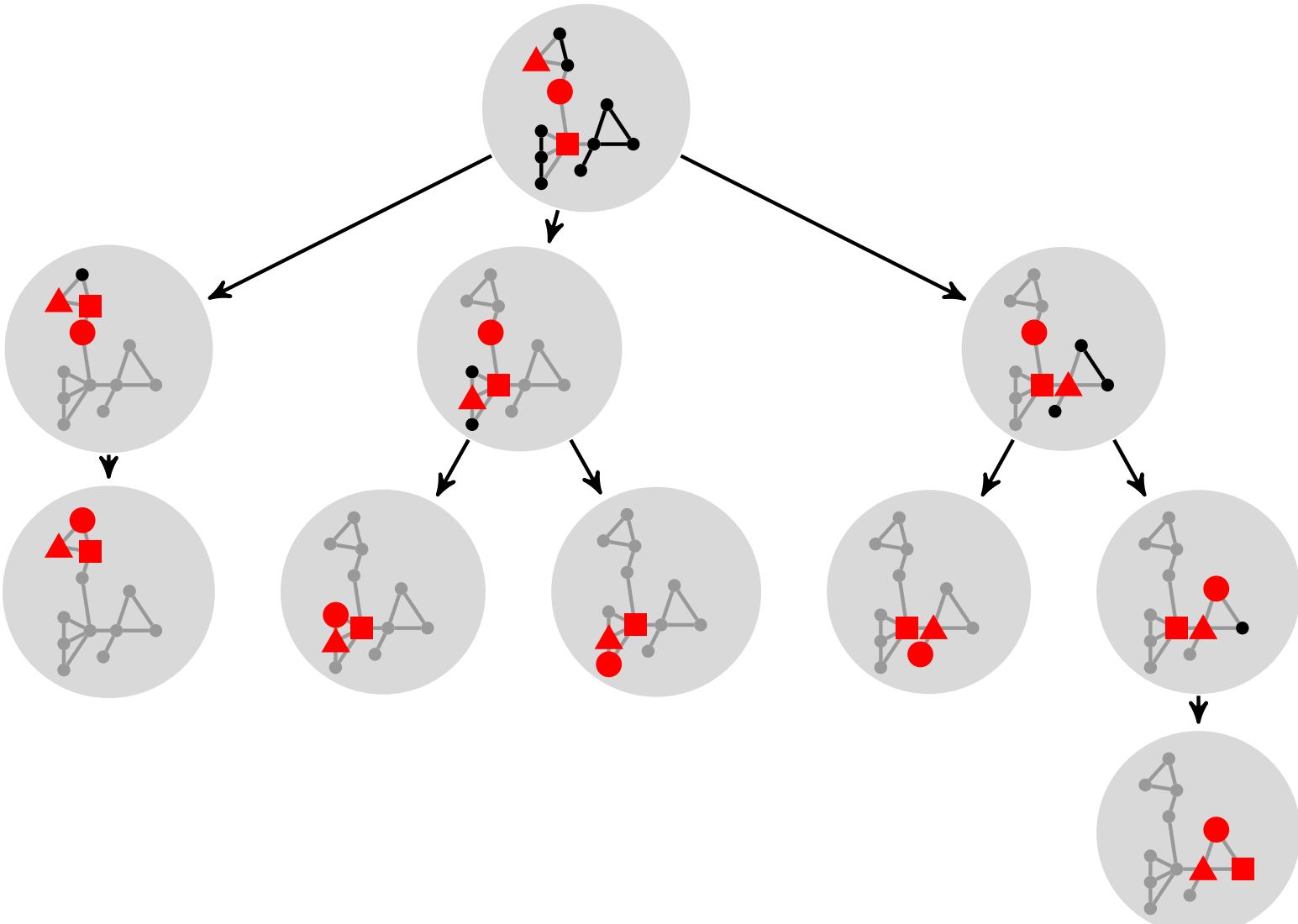


$\models \varphi_{\text{3-colorable}}$?

Die NC^1-Version des Satzes von Courcelle.

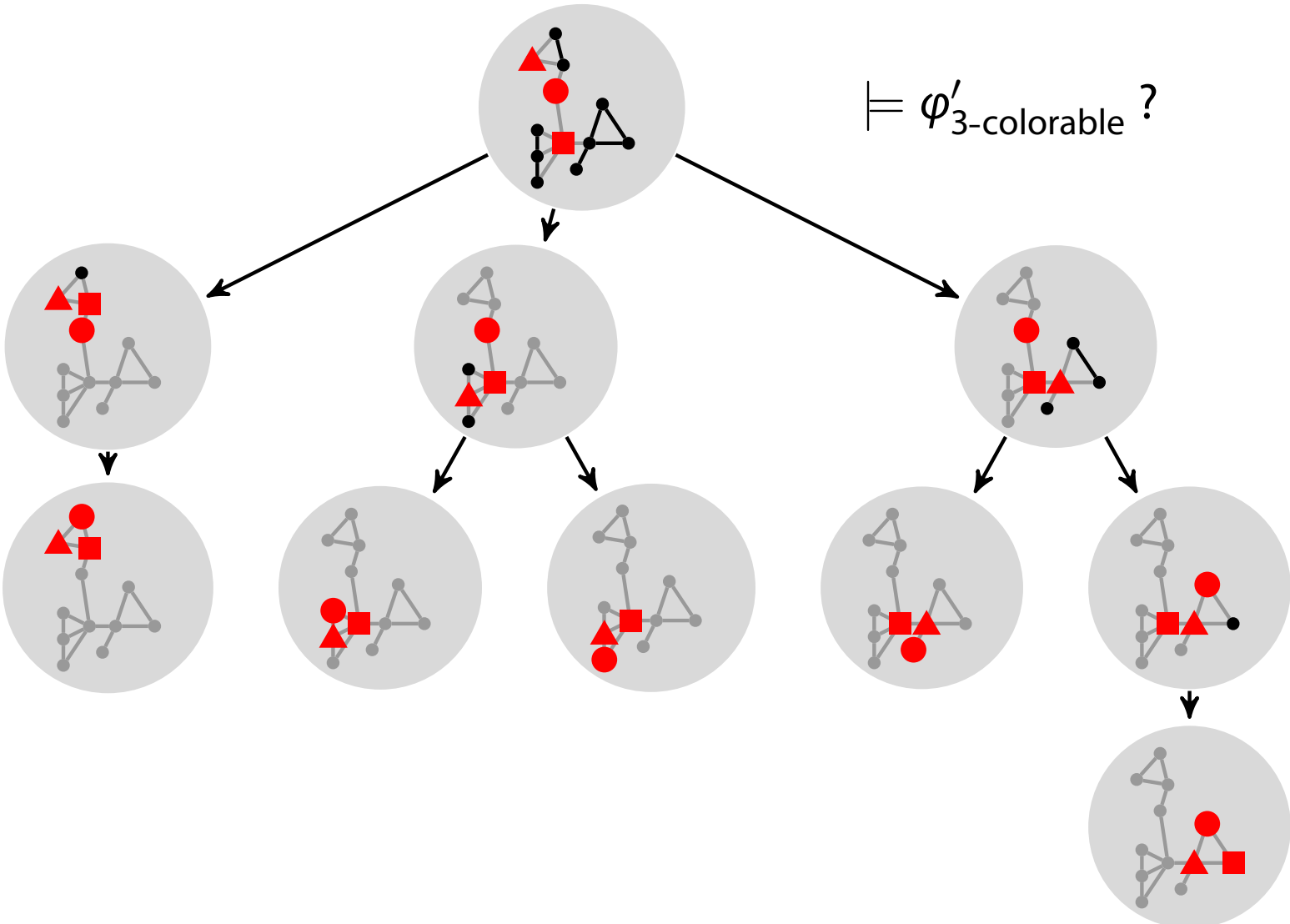
Beweisschritt 2: Berechnung einer Baumzerlegung.

Dies ist trivial, denn sie ist Teil der Eingabe.



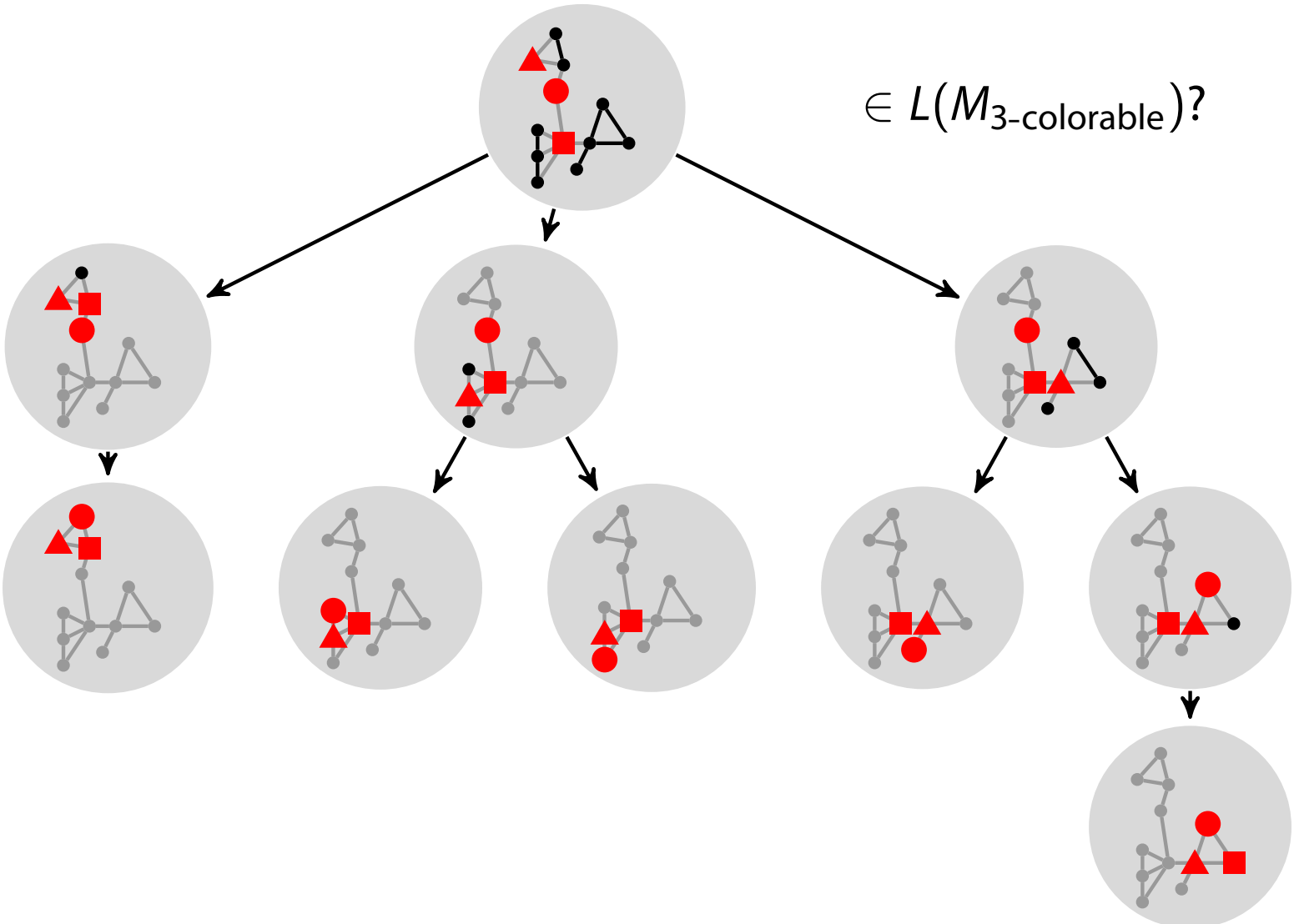
Die NC^1-Version des Satzes von Courcelle.

Beweisschritt 3: Übertragung der Formel auf den Baum:



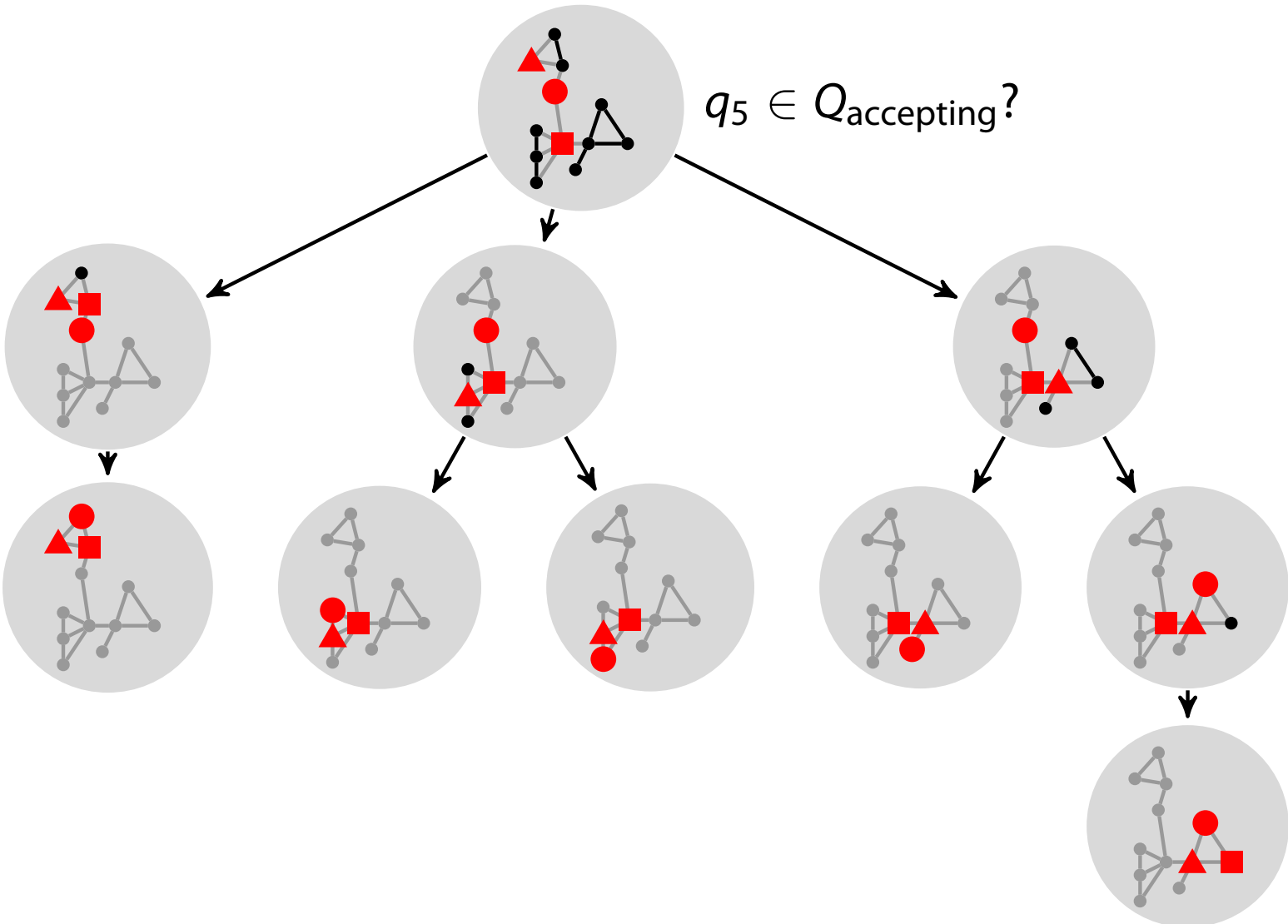
Die NC^1-Version des Satzes von Courcelle.

Beweisschritt 4: Umwandlung der Formel in einen Baumautomaten:



Die NC^1-Version des Satzes von Courcelle.

Beweisschritt 5: Auswertung des Automaten in NC^1.



Gliederung

Eine Klasse sehr einfach zu parsender Sprachen

Visible-Pushdown-Sprachen

Dies sind *einfach zu parsende* kontextfreie Sprachen:

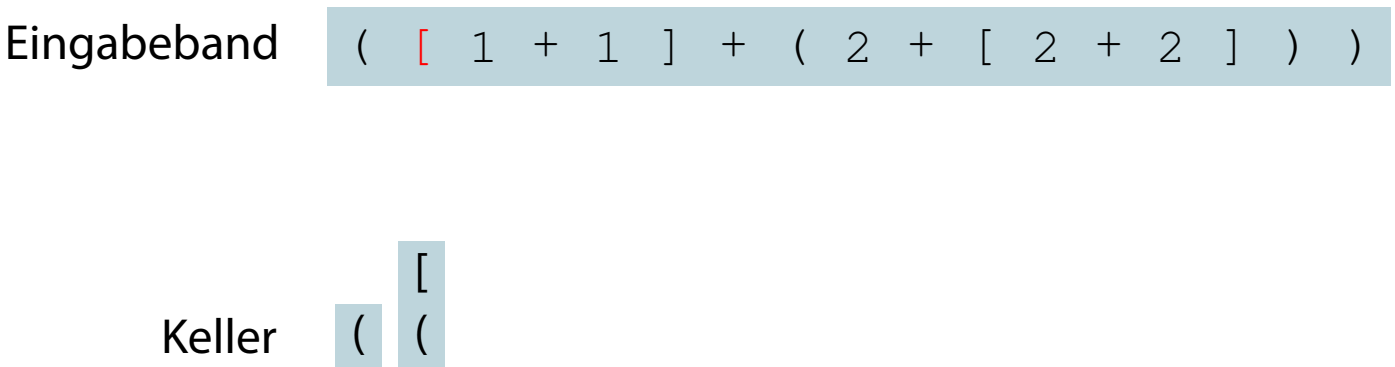
1. Die Eingabe wird von einem Kellerautomaten verarbeitet.
2. Ob er eine Push- oder Pop-Operation durchführt, *hängt nur vom gelesenen Symbol ab* und *nicht* vom aktuellen Zustand.

Das Parsen von korrekt geklammerten Ausdrücken.

Eingabeband `( [ 1 + 1 ] + ( 2 + [ 2 + 2 ] ) )`

Keller `(`

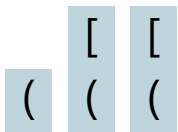
Das Parsen von korrekt geklammerten Ausdrücken.



Das Parsen von korrekt geklammerten Ausdrücken.

Eingabeband

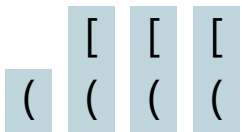
Keller



Das Parsen von korrekt geklammerten Ausdrücken.

Eingabeband

Keller

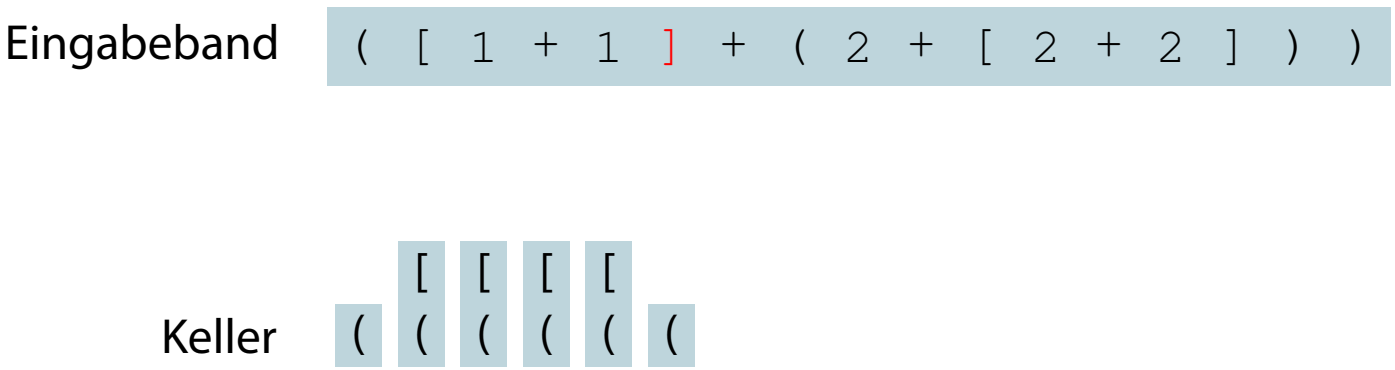


Das Parsen von korrekt geklammerten Ausdrücken.

Eingabeband

Keller

Das Parsen von korrekt geklammerten Ausdrücken.



Das Parsen von korrekt geklammerten Ausdrücken.

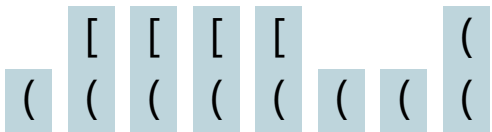
Eingabeband

Keller

Das Parsen von korrekt geklammerten Ausdrücken.

Eingabeband

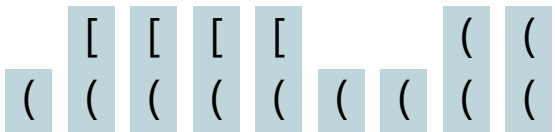
Keller



Das Parsen von korrekt geklammerten Ausdrücken.

Eingabeband

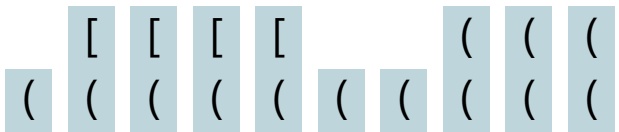
Keller



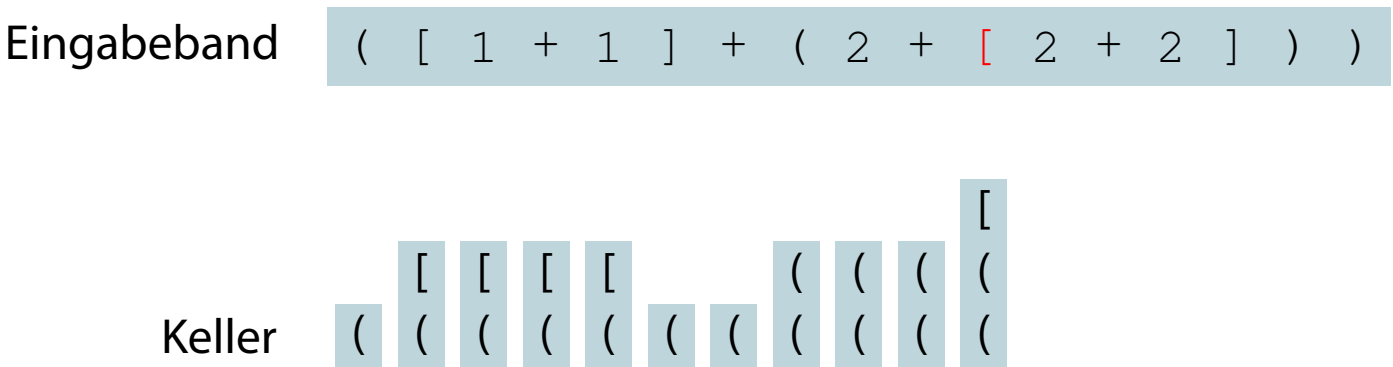
Das Parsen von korrekt geklammerten Ausdrücken.

Eingabeband

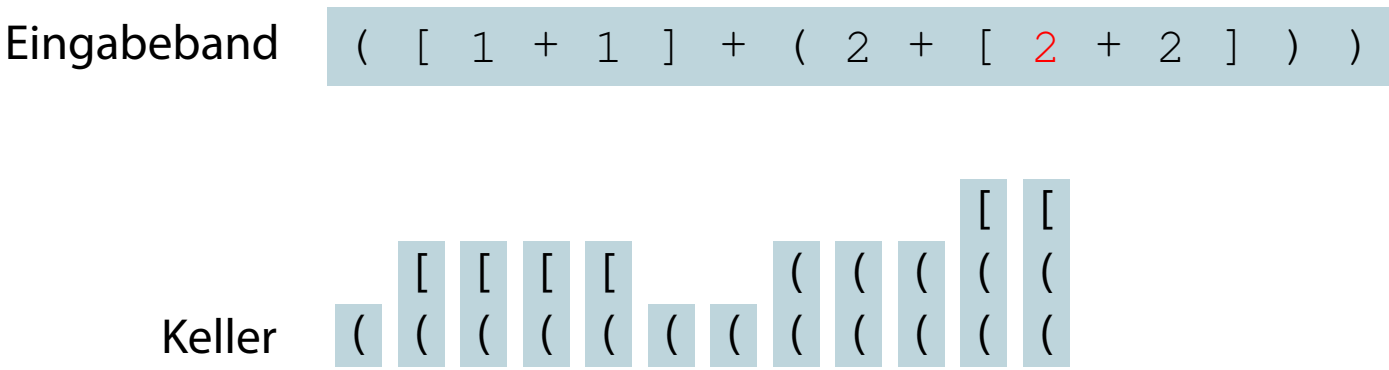
Keller



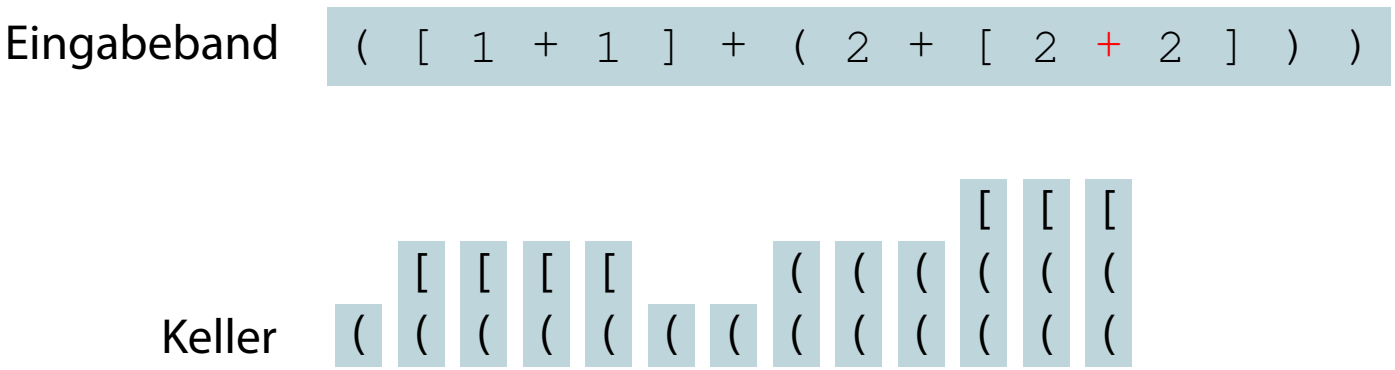
Das Parsen von korrekt geklammerten Ausdrücken.



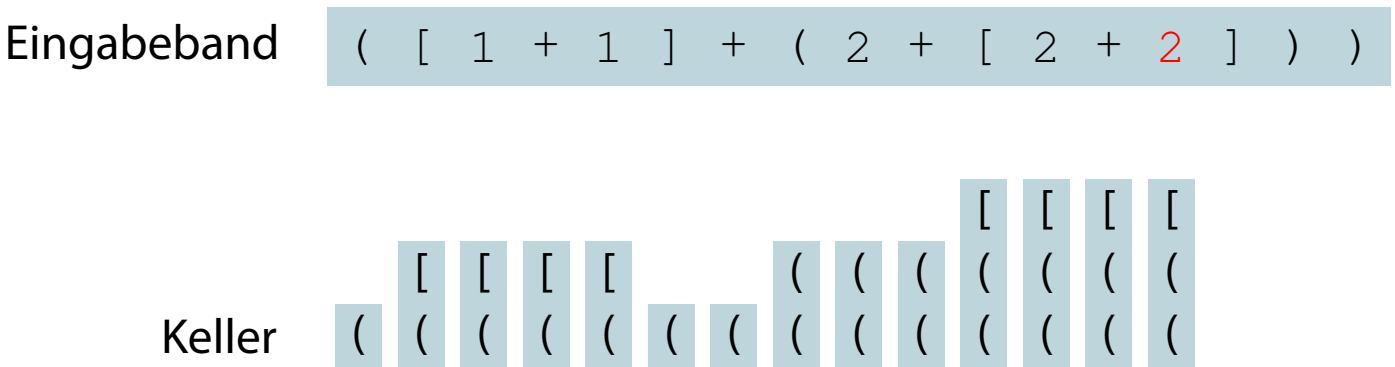
Das Parsen von korrekt geklammerten Ausdrücken.



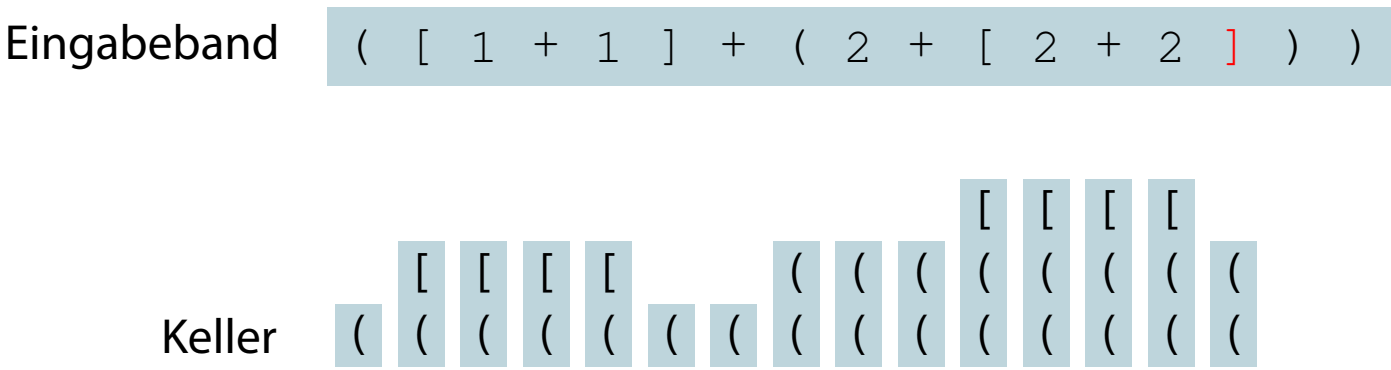
Das Parsen von korrekt geklammerten Ausdrücken.



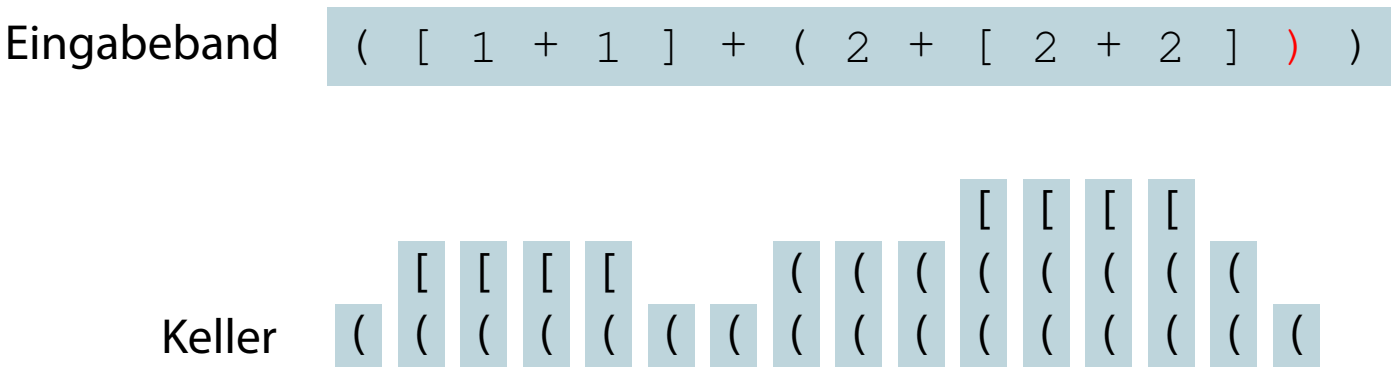
Das Parsen von korrekt geklammerten Ausdrücken.



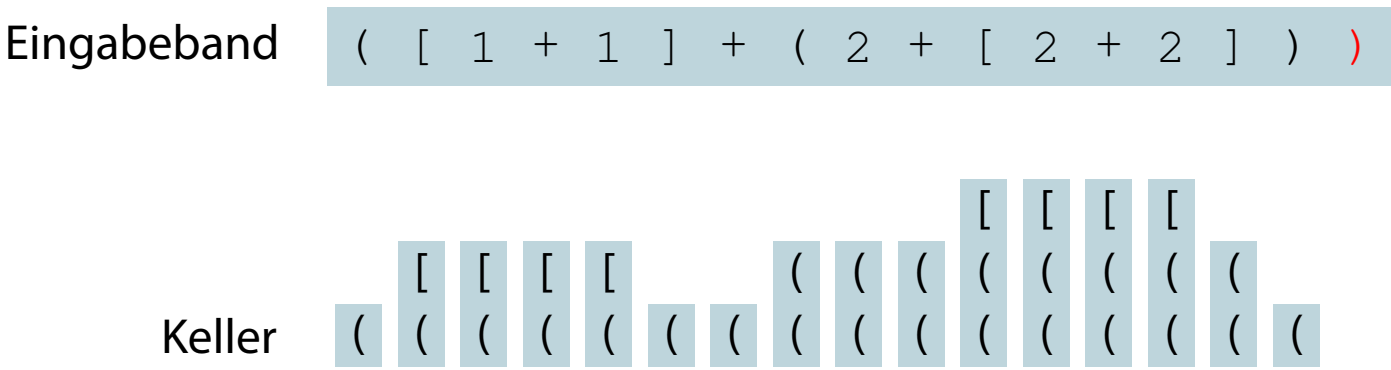
Das Parsen von korrekt geklammerten Ausdrücken.



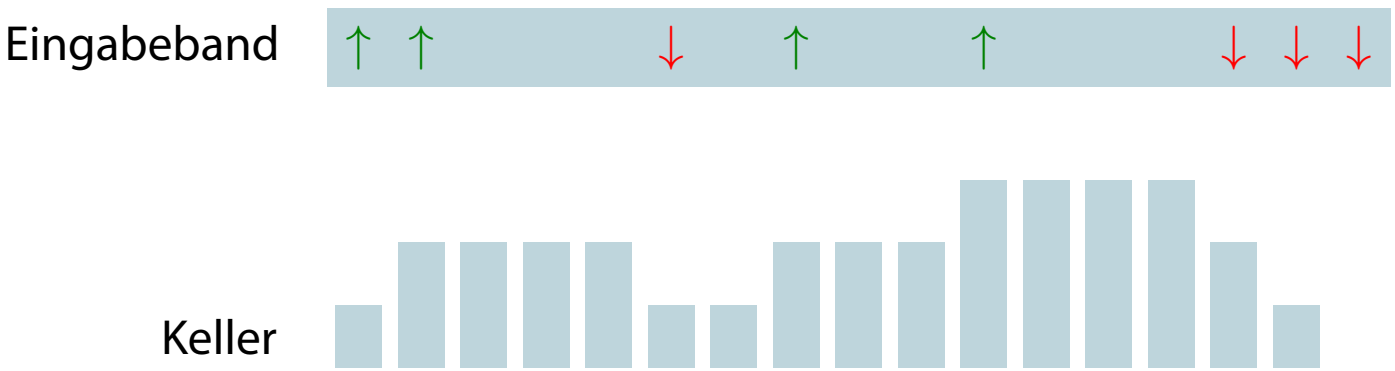
Das Parsen von korrekt geklammerten Ausdrücken.



Das Parsen von korrekt geklammerten Ausdrücken.



Man kann die »Form« des Stacks vorherberechnen.



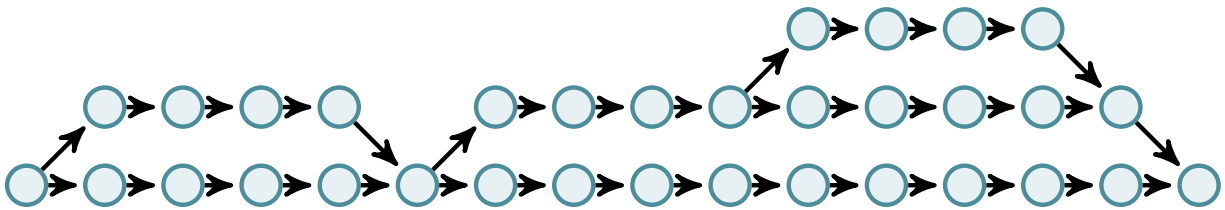
Die Komplexität von Visible-Pushdown-Sprachen

Satz

Jede Visible-Pushdown-Sprache lässt sich in NC^1 *entscheiden.*

Beweis.

Zu einer Eingabe wie $([1 + 1] + (2 + [2 + 2]))$ kann man durch »einfaches Zählen« folgenden Graphen zusammen mit einer Baumzerlegung berechnen:



□

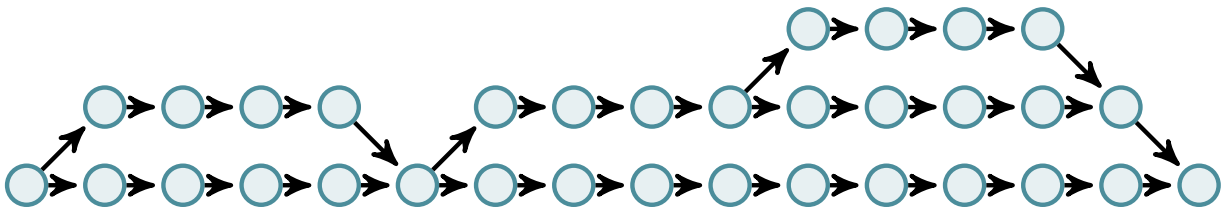
Die Komplexität von Visible-Pushdown-Sprachen

Satz

Jede Visible-Pushdown-Sprache lässt sich in NC^1 *entscheiden.*

Beweis.

Zu einer Eingabe wie $([1 + 1] + (2 + [2 + 2]))$ kann man durch »einfaches Zählen« folgenden Graphen zusammen mit einer Baumzerlegung berechnen:



Wende nun das algorithmische Metatheorem für NC^1 auf ein φ an, das besagt *»es gibt eine Belegung der Knoten mit Stacksymbolen und Automatenzuständen, die lokal konsistent ist«*. □

Zusammenfassung

Algorithmische Metatheoreme sind Sätze der Bauart

> *Wenn ein Problem in einer bestimmten Logik beschreibbar ist und die Eingabegraph eine bestimmte Zerlegung zulassen, dann gibt es eine bestimmte Art Algorithmus für das Problem.«*

- Es gibt algorithmische Metatheoreme für logarithmischen Platz.
- Es gibt algorithmische Metatheoreme für NC^1.
- Algorithmische Metatheoreme haben Anwendungen, die *weit über baumzerlegbare Graphen hinausgehen*.

Ausblick

Was weiß man noch?

- Es gibt algorithmische Metatheoreme für Schaltkreise konstanter Tiefe (AC^0 und TC^0).
- Die Beweistechniken lassen sich auch innerhalb der *reinen Logik* anwenden, um die *Ausdruckstärke von monadischer Logik zweiter Stufe* zu charakterisieren.

Offene Probleme

- Anwendungen, Anwendungen, Anwendung