Satz von Mahaney

Dünne Sprachen sind nicht NP-hart, es sei denn $\mathrm{P} = \mathrm{NP}$

Sebastian Kuhnert

07.12.2004

Gliederung

Gliederung

Dünne Sprachen *(sparse sets)*

- Dünne Sprachen enthalten höchstens polynomiell viele Wörter pro Wortlänge.
- Für eine Sprache L bezeichne $C_L(n) := |L \cap \Sigma^n|$ die Anzahl der Wörter der Länge n (von engl. *census*, Bevölkerungszählung).

Definition (Dünne Sprache)

Eine Sprache $S \subseteq \Sigma^*$ heißt dünn,
wenn ein Polynom $p_S(n)$ existiert, sodass $\forall n \in \mathbb{N} : C_S(n) \leq p_S(n)$.

- Damit ist auch $\sum_{k=0}^{n} C_S(k)$ polynomiell beschränkt.
- Für eine allgemeine Sprache $L \subseteq \Sigma^*$ gilt nur $C_L(n) \leq |\Sigma|^n$.
- Tally-Sprachen sind dünn mit $p(n) = 1$.

Dünne Sprachen *(sparse sets)*

- Dünne Sprachen enthalten höchstens polynomiell viele Wörter pro Wortlänge.
- Für eine Sprache L bezeichne $C_L(n) := |L \cap \Sigma^n|$ die Anzahl der Wörter der Länge n (von engl. *census*, Bevölkerungszählung).

Definition (Dünne Sprache)

Eine Sprache $S \subseteq \Sigma^*$ heißt dünn,
wenn ein Polynom $p_S(n)$ existiert, sodass $\forall n \in \mathbb{N} : C_S(n) \leq p_S(n)$.

- Damit ist auch $\sum_{k=0}^{n} C_S(k)$ polynomiell beschränkt.
- Für eine allgemeine Sprache $L \subseteq \Sigma^*$ gilt nur $C_L(n) \leq |\Sigma|^n$.
- Tally-Sprachen sind dünn mit $p(n) = 1$.

Dünne Sprachen *(sparse sets)*

- Dünne Sprachen enthalten höchstens polynomiell viele Wörter pro Wortlänge.
- Für eine Sprache L bezeichne $C_L(n) := |L \cap \Sigma^n|$ die Anzahl der Wörter der Länge n (von engl. *census*, Bevölkerungszählung).

Definition (Dünne Sprache)

Eine Sprache $S \subseteq \Sigma^*$ heißt dünn,
wenn ein Polynom $p_S(n)$ existiert, sodass $\forall n \in \mathbb{N} : C_S(n) \leq p_S(n)$.

- Damit ist auch $\sum_{k=0}^{n} C_S(k)$ polynomiell beschränkt.
- Für eine allgemeine Sprache $L \subseteq \Sigma^*$ gilt nur $C_L(n) \leq |\Sigma|^n$.
- Tally-Sprachen sind dünn mit $p(n) = 1$.

Dünne Sprachen *(sparse sets)*

- Dünne Sprachen enthalten höchstens polynomiell viele Wörter pro Wortlänge.
- Für eine Sprache L bezeichne $C_L(n) := |L \cap \Sigma^n|$ die Anzahl der Wörter der Länge n (von engl. *census*, Bevölkerungszählung).

Definition (Dünne Sprache)

Eine Sprache $S \subseteq \Sigma^*$ heißt dünn,
wenn ein Polynom $p_S(n)$ existiert, sodass $\forall n \in \mathbb{N} : C_S(n) \leq p_S(n)$.

- Damit ist auch $\sum_{k=0}^{n} C_S(k)$ polynomiell beschränkt.
- Für eine allgemeine Sprache $L \subseteq \Sigma^*$ gilt nur $C_L(n) \leq |\Sigma|^n$.
- Tally-Sprachen sind dünn mit $p(n) = 1$.

Dünne Sprachen *(sparse sets)*

- Dünne Sprachen enthalten höchstens polynomiell viele Wörter pro Wortlänge.
- Für eine Sprache L bezeichne $C_L(n) := |L \cap \Sigma^n|$ die Anzahl der Wörter der Länge n (von engl. *census*, Bevölkerungszählung).

Definition (Dünne Sprache)

Eine Sprache $S \subseteq \Sigma^*$ heißt dünn,
wenn ein Polynom $p_S(n)$ existiert, sodass $\forall n \in \mathbb{N} : C_S(n) \leq p_S(n)$.

- Damit ist auch $\sum_{k=0}^{n} C_S(k)$ polynomiell beschränkt.
- Für eine allgemeine Sprache $L \subseteq \Sigma^*$ gilt nur $C_L(n) \leq |\Sigma|^n$.
- Tally-Sprachen sind dünn mit $p(n) = 1$.

Dünne Sprachen *(sparse sets)*

- Dünne Sprachen enthalten höchstens polynomiell viele Wörter pro Wortlänge.
- Für eine Sprache L bezeichne $C_L(n) := |L \cap \Sigma^n|$ die Anzahl der Wörter der Länge n (von engl. *census*, Bevölkerungszählung).

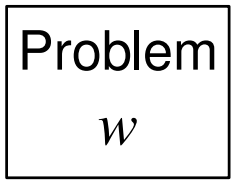
Definition (Dünne Sprache)

Eine Sprache $S \subseteq \Sigma^*$ heißt dünn,
wenn ein Polynom $p_S(n)$ existiert, sodass $\forall n \in \mathbb{N} : C_S(n) \leq p_S(n)$.

- Damit ist auch $\sum_{k=0}^{n} C_S(k)$ polynomiell beschränkt.
- Für eine allgemeine Sprache $L \subseteq \Sigma^*$ gilt nur $C_L(n) \leq |\Sigma|^n$.
- Tally-Sprachen sind dünn mit $p(n) = 1$.

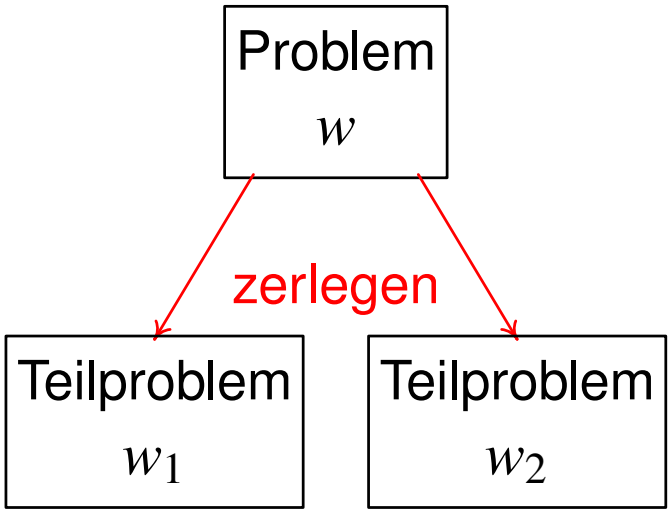
Gliederung

Selbstreduktionsbäume *(self-reducing-trees)*



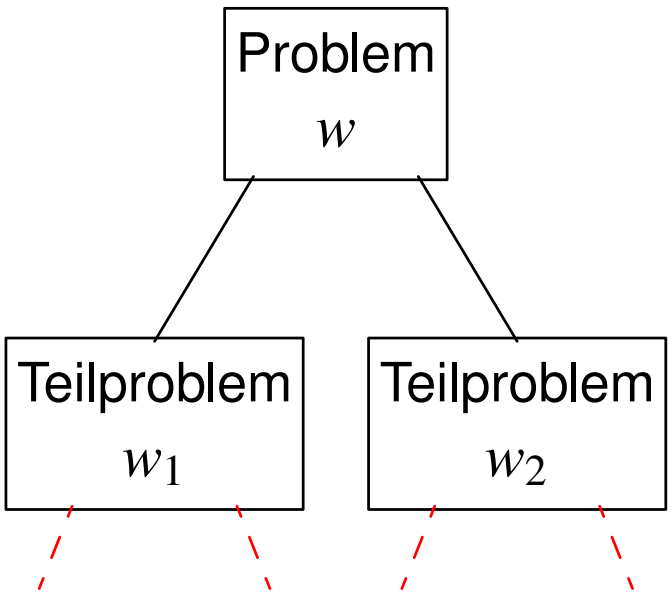
- Ein Wort w einer selbstreduzierbaren Sprache L soll entschieden werden.
- Es werden Wörter $w_1 \dots w_k \in L$ konstruiert, die leichter als w entscheidbar sind.
- Diese werden bis zu einem geeigneten Rekursionsschluss weiterzerlegt.
- Die Ergebnisse werden durch eine logische Funktion zusammengeführt.
- Wird hierfür die Disjunktion verwendet, spricht man auch von einem *d*-Selbstreduktionsbaum.

Selbstreduktionsbäume *(self-reducing-trees)*



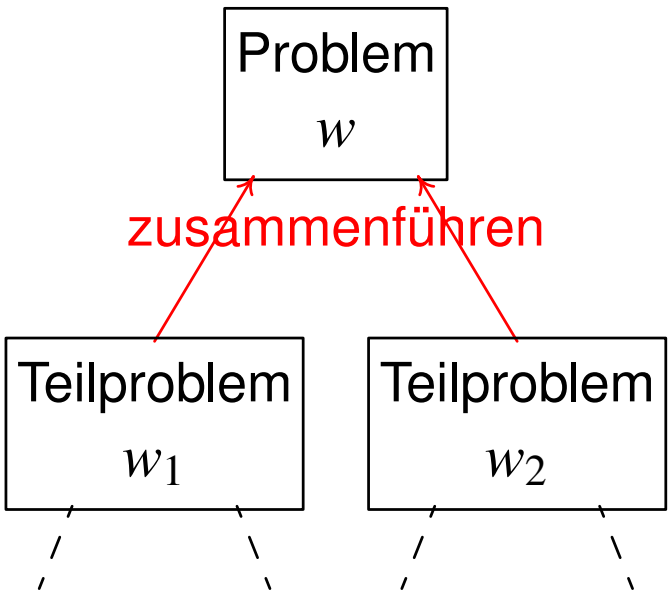
- Ein Wort w einer selbstreduzierbaren Sprache L soll entschieden werden.
- Es werden Wörter $w_1 \dots w_k \in L$ konstruiert, die leichter als w entscheidbar sind.
- Diese werden bis zu einem geeigneten Rekursionsschluss weiterzerlegt.
- Die Ergebnisse werden durch eine logische Funktion zusammengeführt.
- Wird hierfür die Disjunktion verwendet, spricht man auch von einem *d*-Selbstreduktionsbaum.

Selbstreduktionsbäume *(self-reducing-trees)*



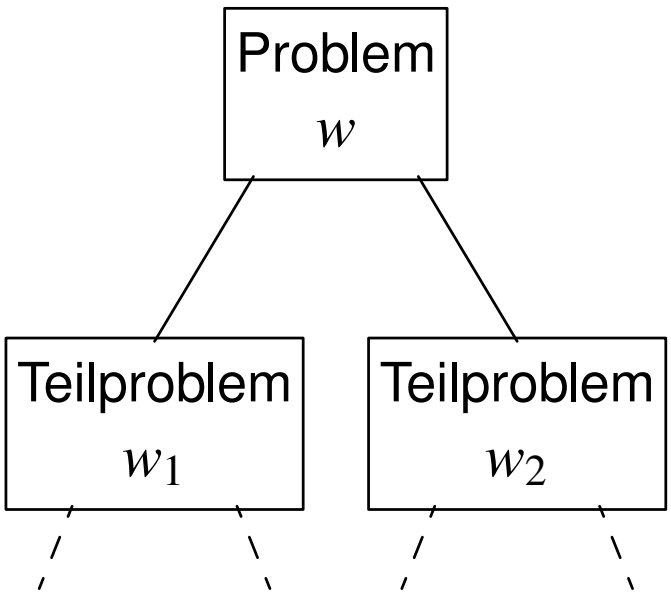
- Ein Wort w einer selbstreduzierbaren Sprache L soll entschieden werden.
- Es werden Wörter $w_1 \dots w_k \in L$ konstruiert, die leichter als w entscheidbar sind.
- Diese werden bis zu einem geeigneten Rekursionsschluss weiterzerlegt.
- Die Ergebnisse werden durch eine logische Funktion zusammengeführt.
- Wird hierfür die Disjunktion verwendet, spricht man auch von einem *d*-Selbstreduktionsbaum.

Selbstreduktionsbäume *(self-reducing-trees)*



- Ein Wort w einer selbstreduzierbaren Sprache L soll entschieden werden.
- Es werden Wörter $w_1 \dots w_k \in L$ konstruiert, die leichter als w entscheidbar sind.
- Diese werden bis zu einem geeigneten Rekursionsschluss weiterzerlegt.
- Die Ergebnisse werden durch eine logische Funktion zusammengeführt.
- Wird hierfür die Disjunktion verwendet, spricht man auch von einem *d*-Selbstreduktionsbaum.

Selbstreduktionsbäume *(self-reducing-trees)*



- Ein Wort w einer selbstreduzierbaren Sprache L soll entschieden werden.
- Es werden Wörter $w_1 \ldots w_k \in L$ konstruiert, die leichter als w entscheidbar sind.
- Diese werden bis zu einem geeigneten Rekursionsschluss weiterzerlegt.
- Die Ergebnisse werden durch eine logische Funktion zusammengeführt.
- Wird hierfür die Disjunktion verwendet, spricht man auch von einem *d*-Selbstreduktionsbaum.

Gliederung

Beweisüberblick

Satz von Mahaney *(Mahaney's Theorem)*

Dünne Sprachen sind nicht NP-hart, es sei denn $\mathrm{P} = \mathrm{NP}$.

- Widerspruchsbeweis mit Annahme $\mathrm{SAT} \leq_m^{\mathrm{P}} S$ für eine beliebige dünne Sprache S mit $\Sigma_{k=0}^{n} C_S(k) \leq p_S(n)$.
- Wir konstruieren eine Sprache L mit $L \leq_m^{\mathrm{P}} \mathrm{SAT}$.
- Damit ist $L \leq_m^{\mathrm{P}} S$ via einer Reduktionsfunktion $f \in \mathrm{FP}$.
- Wir zeigen, dass SAT unter Kenntnis von f in deterministischer Polynomialzeit entscheidbar ist.
- Unter der Annahme wäre also $\mathrm{SAT} \in \mathrm{P}$.
- Wegen der NP-Vollständigkeit von SAT würde auch $\mathrm{P} = \mathrm{NP}$ gelten.

Beweisüberblick

Satz von Mahaney *(Mahaney's Theorem)*

Dünne Sprachen sind nicht NP-hart, es sei denn $\mathrm{P} = \mathrm{NP}$.

- Widerspruchsbeweis mit Annahme $\mathrm{SAT} \leq_m^{\mathrm{P}} S$
  für eine beliebige dünne Sprache S mit $\Sigma_{k=0}^{n} C_S(k) \leq p_S(n)$.
- Wir konstruieren eine Sprache L mit $L \leq_m^{\mathrm{P}} \mathrm{SAT}$.
- Damit ist $L \leq_m^{\mathrm{P}} S$ via einer Reduktionsfunktion $f \in \mathrm{FP}$.
- Wir zeigen, dass SAT unter Kenntnis von f in deterministischer Polynomialzeit entscheidbar ist.
- Unter der Annahme wäre also $\mathrm{SAT} \in \mathrm{P}$.
- Wegen der NP-Vollständigkeit von SAT würde auch $\mathrm{P} = \mathrm{NP}$ gelten.

Beweisüberblick

Satz von Mahaney *(Mahaney's Theorem)*

Dünne Sprachen sind nicht NP-hart, es sei denn $\mathrm{P} = \mathrm{NP}$.

- Widerspruchsbeweis mit Annahme $\mathrm{SAT} \leq_m^{\mathrm{P}} S$ für eine beliebige dünne Sprache S mit $\Sigma_{k=0}^{n} C_S(k) \leq p_S(n)$.
- Wir konstruieren eine Sprache L mit $L \leq_m^{\mathrm{P}} \mathrm{SAT}$.
- Damit ist $L \leq_m^{\mathrm{P}} S$ via einer Reduktionsfunktion $f \in \mathrm{FP}$.
- Wir zeigen, dass SAT unter Kenntnis von f in deterministischer Polynomialzeit entscheidbar ist.
- Unter der Annahme wäre also $\mathrm{SAT} \in \mathrm{P}$.
- Wegen der NP-Vollständigkeit von SAT würde auch $\mathrm{P} = \mathrm{NP}$ gelten.

Beweisüberblick

Satz von Mahaney *(Mahaney's Theorem)*

Dünne Sprachen sind nicht NP-hart, es sei denn $\mathrm{P} = \mathrm{NP}$.

- Widerspruchsbeweis mit Annahme $\mathrm{SAT} \leq_m^{\mathrm{P}} S$ für eine beliebige dünne Sprache S mit $\Sigma_{k=0}^{n} C_S(k) \leq p_S(n)$.
- Wir konstruieren eine Sprache L mit $L \leq_m^{\mathrm{P}} \mathrm{SAT}$.
- Damit ist $L \leq_m^{\mathrm{P}} S$ via einer Reduktionsfunktion $f \in \mathrm{FP}$.
- Wir zeigen, dass SAT unter Kenntnis von f in deterministischer Polynomialzeit entscheidbar ist.
- Unter der Annahme wäre also $\mathrm{SAT} \in \mathrm{P}$.
- Wegen der NP-Vollständigkeit von SAT würde auch $\mathrm{P} = \mathrm{NP}$ gelten.

Beweisüberblick

Satz von Mahaney *(Mahaney's Theorem)*

Dünne Sprachen sind nicht NP-hart, es sei denn $\mathrm{P} = \mathrm{NP}$.

- Widerspruchsbeweis mit Annahme $\mathrm{SAT} \leq_m^{\mathrm{P}} S$ für eine beliebige dünne Sprache S mit $\Sigma_{k=0}^{n} C_S(k) \leq p_S(n)$.
- Wir konstruieren eine Sprache L mit $L \leq_m^{\mathrm{P}} \mathrm{SAT}$.
- Damit ist $L \leq_m^{\mathrm{P}} S$ via einer Reduktionsfunktion $f \in \mathrm{FP}$.
- Wir zeigen, dass SAT unter Kenntnis von f in deterministischer Polynomialzeit entscheidbar ist.
- Unter der Annahme wäre also $\mathrm{SAT} \in \mathrm{P}$.
- Wegen der NP-Vollständigkeit von SAT würde auch $\mathrm{P} = \mathrm{NP}$ gelten.

Beweisüberblick

Satz von Mahaney *(Mahaney's Theorem)*

Dünne Sprachen sind nicht NP-hart, es sei denn $\mathrm{P} = \mathrm{NP}$.

- Widerspruchsbeweis mit Annahme $\mathrm{SAT} \leq_m^{\mathrm{P}} S$ für eine beliebige dünne Sprache S mit $\Sigma_{k=0}^{n} C_S(k) \leq p_S(n)$.
- Wir konstruieren eine Sprache L mit $L \leq_m^{\mathrm{P}} \mathrm{SAT}$.
- Damit ist $L \leq_m^{\mathrm{P}} S$ via einer Reduktionsfunktion $f \in \mathrm{FP}$.
- Wir zeigen, dass SAT unter Kenntnis von f in deterministischer Polynomialzeit entscheidbar ist.
- Unter der Annahme wäre also $\mathrm{SAT} \in \mathrm{P}$.
- Wegen der NP-Vollständigkeit von SAT würde auch $\mathrm{P} = \mathrm{NP}$ gelten.

Beweisüberblick

Satz von Mahaney *(Mahaney's Theorem)*

Dünne Sprachen sind nicht NP-hart, es sei denn $\mathrm{P} = \mathrm{NP}$.

- Widerspruchsbeweis mit Annahme $\mathrm{SAT} \leq_m^{\mathrm{P}} S$
  für eine beliebige dünne Sprache S mit $\Sigma_{k=0}^{n} C_S(k) \leq p_S(n)$.
- Wir konstruieren eine Sprache L mit $L \leq_m^{\mathrm{P}} \mathrm{SAT}$.
- Damit ist $L \leq_m^{\mathrm{P}} S$ via einer Reduktionsfunktion $f \in \mathrm{FP}$.
- Wir zeigen, dass SAT unter Kenntnis von f in deterministischer Polynomialzeit entscheidbar ist.
- Unter der Annahme wäre also $\mathrm{SAT} \in \mathrm{P}$.
- Wegen der NP-Vollständigkeit von SAT
  würde auch $\mathrm{P} = \mathrm{NP}$ gelten.

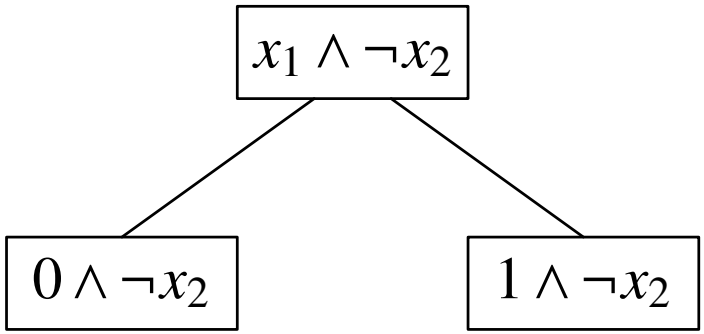
Gliederung

Selbstreduktionsbaum für eine Formel

$$\boxed{x_1 \wedge \neg x_2}$$

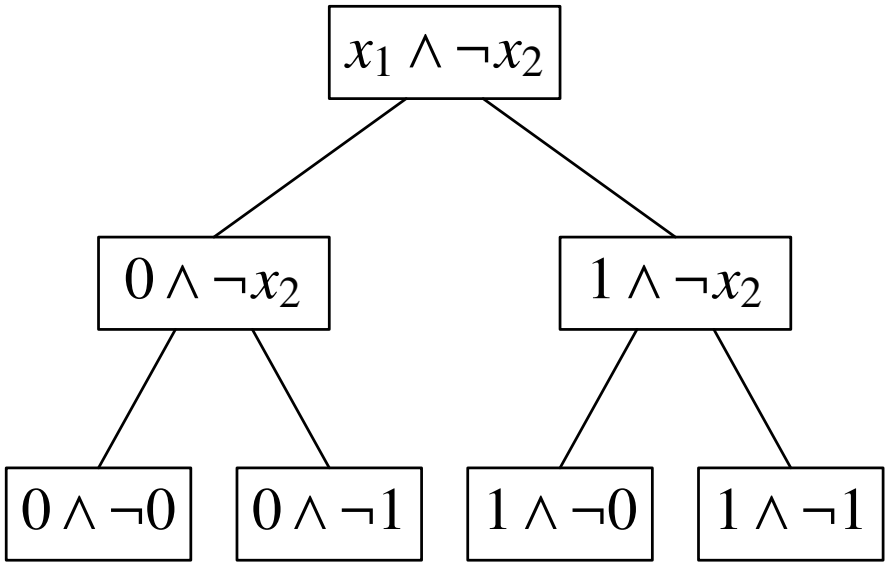
- Frage: Erfüllbarkeit von $\varphi = x_1 \wedge \neg x_2$
- Zerlegung durch sukzessive Belegung der einzelnen Variablen
- Erfüllbarkeit einer Formel mit $n-1$ Variablen ist einfacher zu prüfen als die einer Formel mit n Variablen
- Überprüfen der Blätter
- Zusammenführen durch logisches Oder

Selbstreduktionsbaum für eine Formel



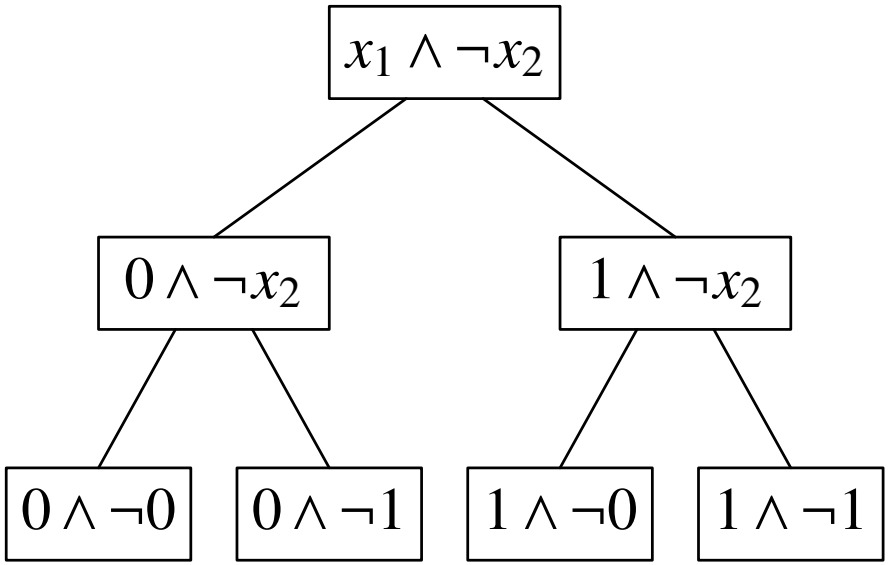
- Frage: Erfüllbarkeit von $\varphi = x_1 \wedge \neg x_2$
- Zerlegung durch sukzessive Belegung der einzelnen Variablen
- Erfüllbarkeit einer Formel mit $n-1$ Variablen ist einfacher zu prüfen als die einer Formel mit n Variablen
- Überprüfen der Blätter
- Zusammenführen durch logisches Oder

Selbstreduktionsbaum für eine Formel



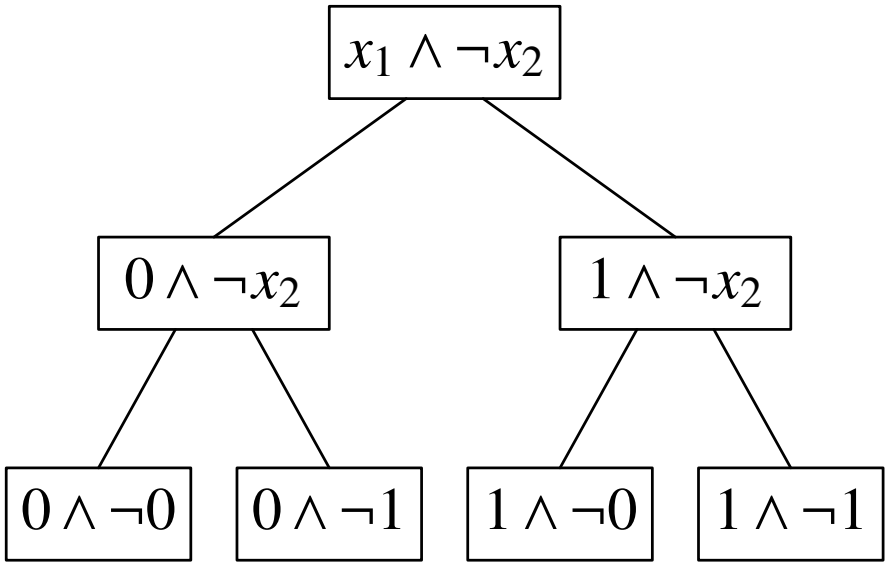
- Frage: Erfüllbarkeit von $\varphi = x_1 \wedge \neg x_2$
- Zerlegung durch sukzessive Belegung der einzelnen Variablen
- Erfüllbarkeit einer Formel mit $n-1$ Variablen ist einfacher zu prüfen als die einer Formel mit n Variablen
- Überprüfen der Blätter
- Zusammenführen durch logisches Oder

Selbstreduktionsbaum für eine Formel



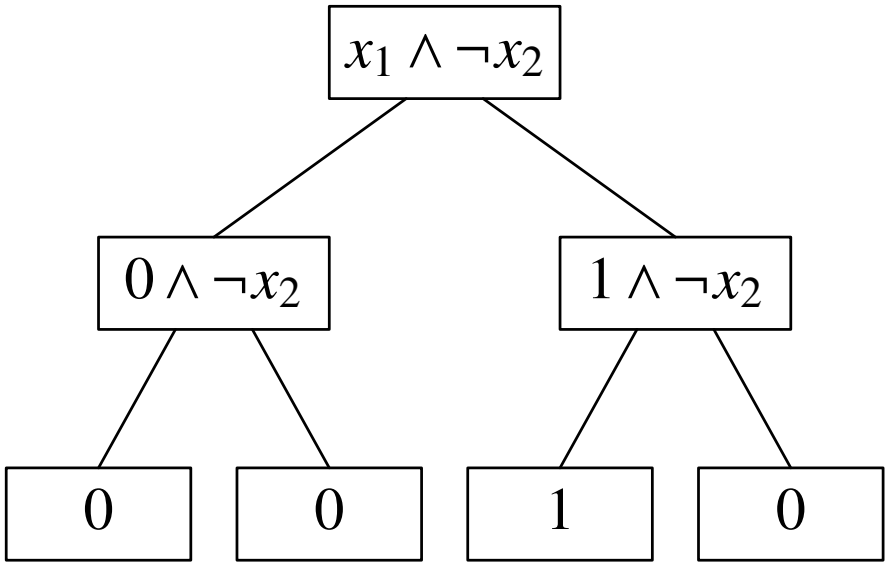
- Frage: Erfüllbarkeit von $\varphi = x_1 \wedge \neg x_2$
- Zerlegung durch sukzessive Belegung der einzelnen Variablen
- Erfüllbarkeit einer Formel mit $n-1$ Variablen ist einfacher zu prüfen als die einer Formel mit n Variablen
- Überprüfen der Blätter
- Zusammenführen durch logisches Oder

Selbstreduktionsbaum für eine Formel



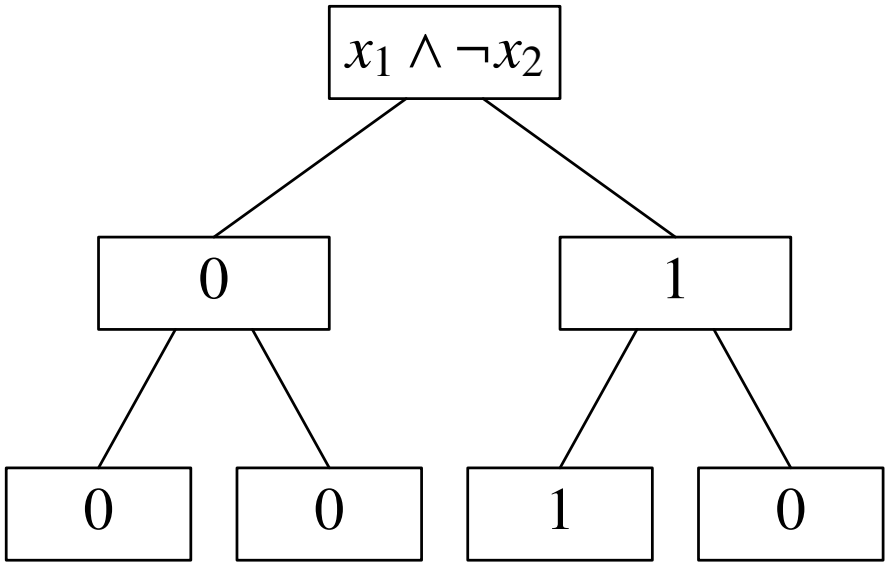
- Frage: Erfüllbarkeit von $\varphi = x_1 \wedge \neg x_2$
- Zerlegung durch sukzessive Belegung der einzelnen Variablen
- Erfüllbarkeit einer Formel mit $n-1$ Variablen ist einfacher zu prüfen als die einer Formel mit n Variablen
- Überprüfen der Blätter
- Zusammenführen durch logisches Oder

Selbstreduktionsbaum für eine Formel



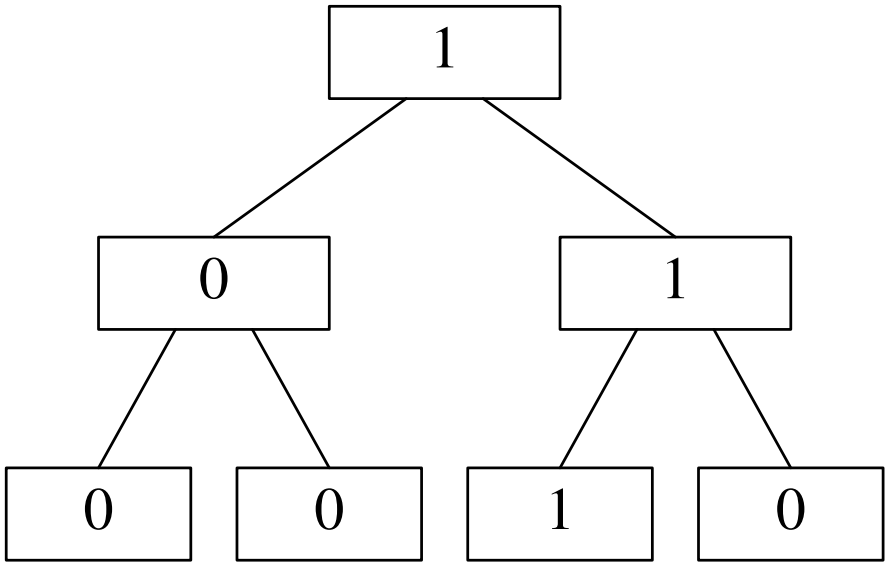
- Frage: Erfüllbarkeit von $\varphi = x_1 \wedge \neg x_2$
- Zerlegung durch sukzessive Belegung der einzelnen Variablen
- Erfüllbarkeit einer Formel mit $n-1$ Variablen ist einfacher zu prüfen als die einer Formel mit n Variablen
- Überprüfen der Blätter
- Zusammenführen durch logisches Oder

Selbstreduktionsbaum für eine Formel



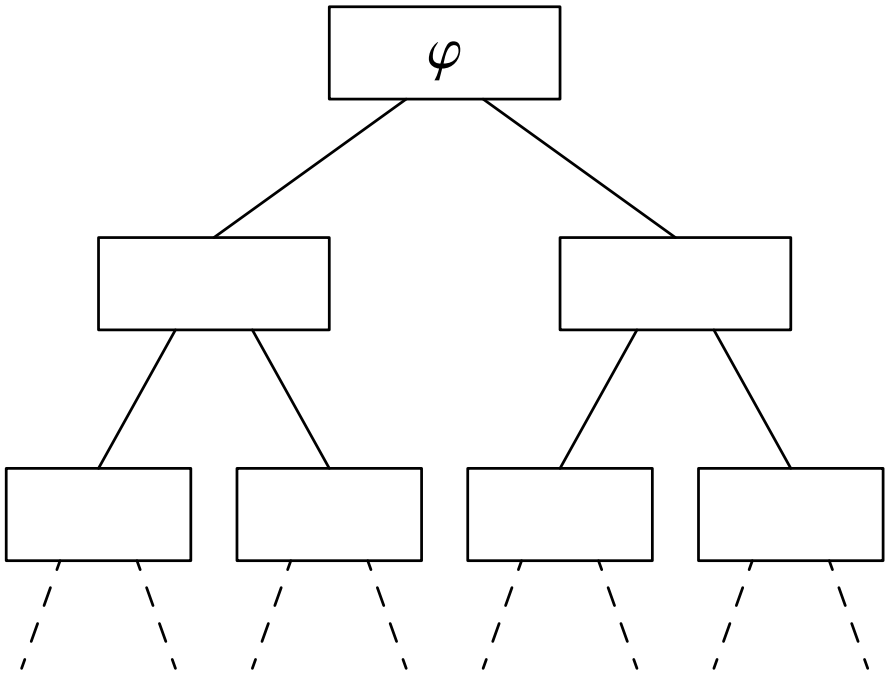
- Frage: Erfüllbarkeit von $\varphi = x_1 \wedge \neg x_2$
- Zerlegung durch sukzessive Belegung der einzelnen Variablen
- Erfüllbarkeit einer Formel mit $n-1$ Variablen ist einfacher zu prüfen als die einer Formel mit n Variablen
- Überprüfen der Blätter
- Zusammenführen durch logisches Oder

Selbstreduktionsbaum für eine Formel



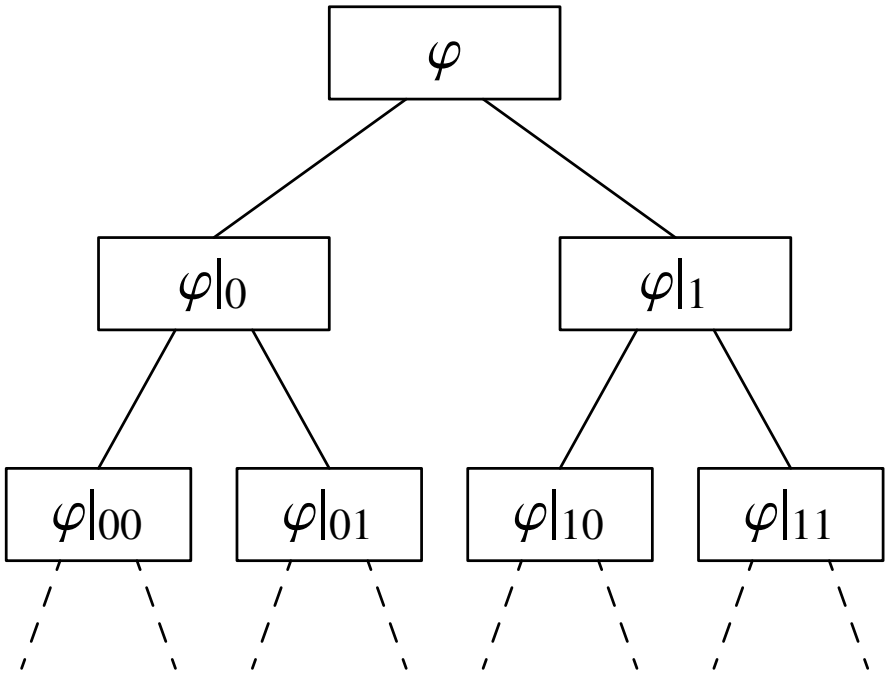
- Frage: Erfüllbarkeit von $\varphi = x_1 \wedge \neg x_2$
- Zerlegung durch sukzessive Belegung der einzelnen Variablen
- Erfüllbarkeit einer Formel mit $n-1$ Variablen ist einfacher zu prüfen als die einer Formel mit n Variablen
- Überprüfen der Blätter
- Zusammenführen durch logisches Oder

Selbstreduktionsbaum für beliebige Formeln



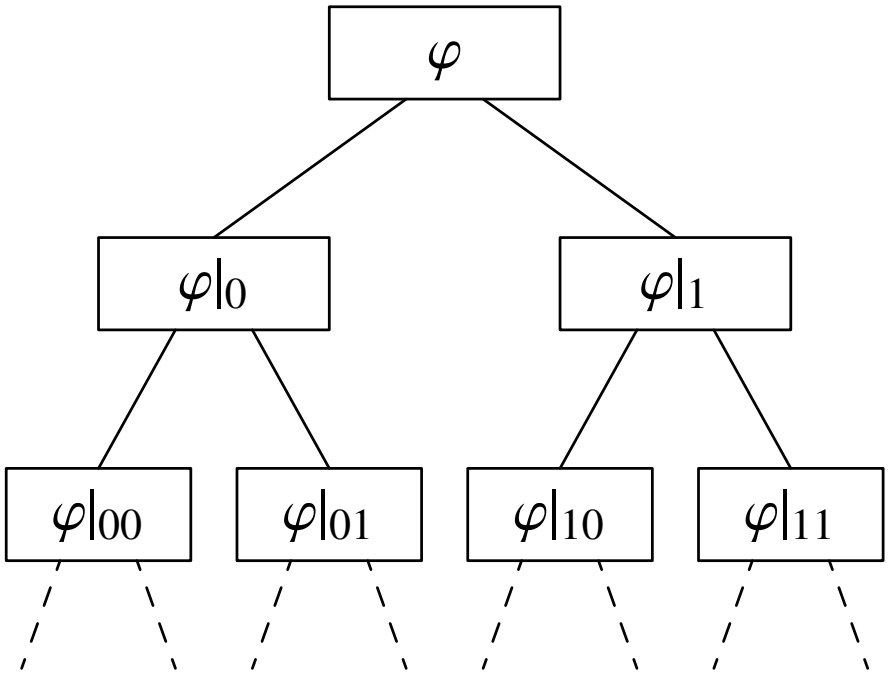
- Verallgemeinerung auf beliebige Formel φ
- Schreibweise: $\varphi|_w$ für Belegung der ersten n Variablen von φ mit $w = b_1 b_2 \cdots b_n$
- Ziel: Einheitliche Länge für w, um eine *Selbst*reduktion zu ermöglichen
- Lösung: Padding mit 0 (Hier hat φ drei Variablen)
- Schreibweise: w' für gepaddete Belegung w

Selbstreduktionsbaum für beliebige Formeln



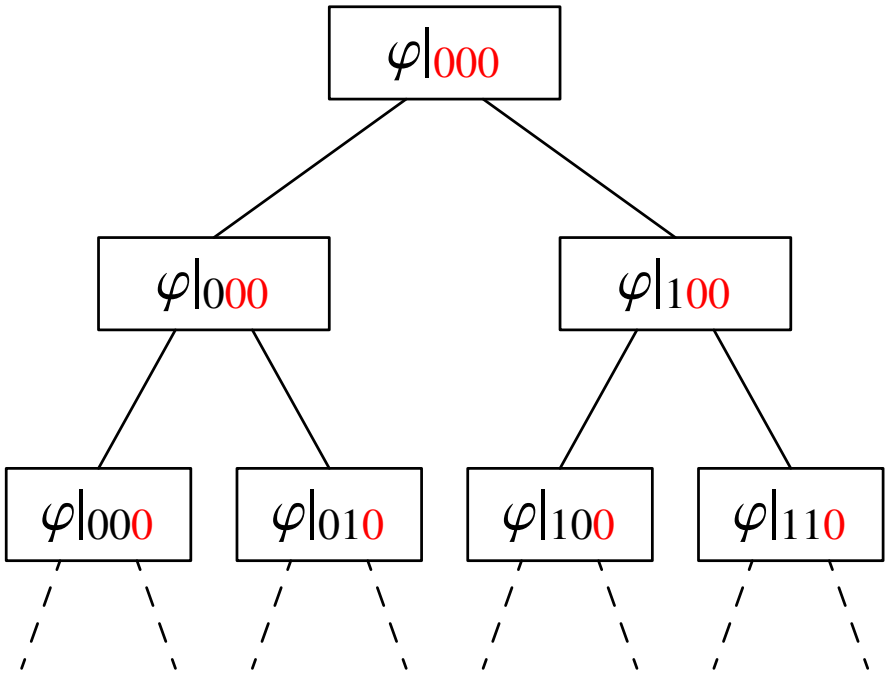
- Verallgemeinerung auf beliebige Formel φ
- Schreibweise: $\varphi|_w$ für Belegung der ersten n Variablen von φ mit $w = b_1 b_2 \cdots b_n$
- Ziel: Einheitliche Länge für w, um eine *Selbst*reduktion zu ermöglichen
- Lösung: Padding mit 0 (Hier hat φ drei Variablen)
- Schreibweise: w' für gepaddete Belegung w

Selbstreduktionsbaum für beliebige Formeln



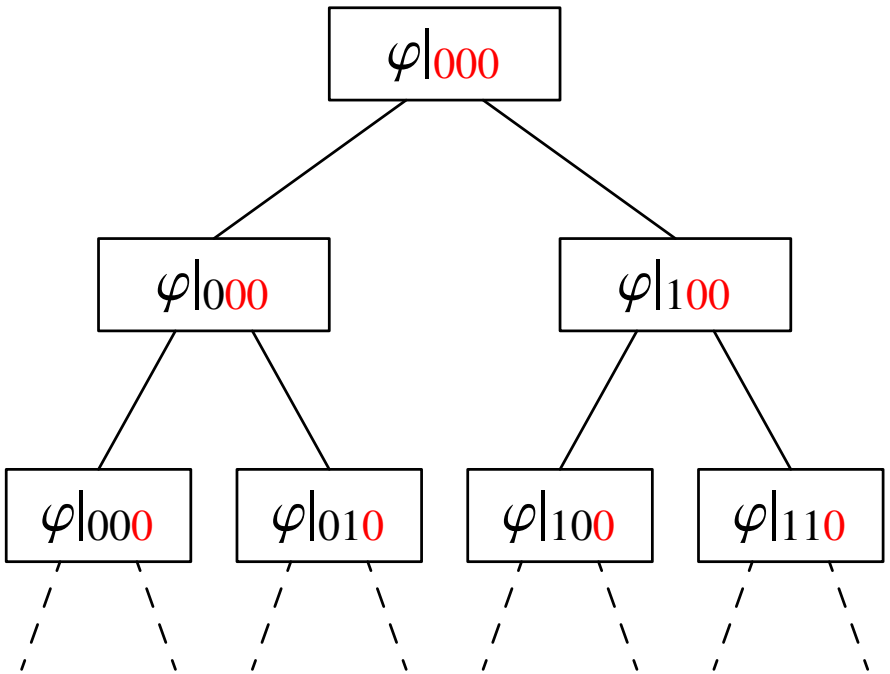
- Verallgemeinerung auf beliebige Formel φ
- Schreibweise: $\varphi|_w$ für Belegung der ersten n Variablen von φ mit $w = b_1 b_2 \cdots b_n$
- Ziel: Einheitliche Länge für w, um eine *Selbst*reduktion zu ermöglichen
- Lösung: Padding mit 0 (Hier hat φ drei Variablen)
- Schreibweise: w' für gepaddete Belegung w

Selbstreduktionsbaum für beliebige Formeln



- Verallgemeinerung auf beliebige Formel φ
- Schreibweise: $\varphi|_w$ für Belegung der ersten n Variablen von φ mit $w = b_1 b_2 \cdots b_n$
- Ziel: Einheitliche Länge für w, um eine *Selbst*reduktion zu ermöglichen
- Lösung: Padding mit 0 (Hier hat φ drei Variablen)
- Schreibweise: w' für gepaddete Belegung w

Selbstreduktionsbaum für beliebige Formeln



- Verallgemeinerung auf beliebige Formel φ
- Schreibweise: $\varphi|_w$ für Belegung der ersten n Variablen von φ mit $w = b_1 b_2 \cdots b_n$
- Ziel: Einheitliche Länge für w, um eine *Selbst*reduktion zu ermöglichen
- Lösung: Padding mit 0
  (Hier hat φ drei Variablen)
- Schreibweise: w' für gepaddete Belegung w

Konstruktion der Sprache L

- Sei $|\mathrm{Var}(\varphi)|$ die Anzahl der Variablen in φ, die lexikographische Ordnung für Bitstrings sei $\preceq$.
- L enthält Tupel $\langle \varphi | w \rangle$ aus einer Formel φ und einer Belegung w, die φ wahr macht.
- Problem: $\varphi|_{01}$ kann wahr sein, ohne dass $\varphi|_{00}$ wahr ist. Im Selbstreduktionsbaum ist $\varphi|_{01}$ Kind von $\varphi|_{00}$.
- Alle $u \preceq w$ müssen also auch in L liegen. Diese Konstruktion heißt *leftset*.

Die Sprache L

$$L = \left\{ \langle \varphi | w \rangle \;\middle|\; |w| = |\mathrm{Var}(\varphi)|, \left(\exists u \in \{0,1\}^{|\mathrm{Var}(\varphi)|} : w \preceq u, \varphi|_u = 1 \right) \right\}$$

Konstruktion der Sprache L

- Sei $|\mathrm{Var}(\varphi)|$ die Anzahl der Variablen in φ, die lexikographische Ordnung für Bitstrings sei $\preceq$.
- L enthält Tupel $\langle \varphi | w \rangle$ aus einer Formel φ und einer Belegung w, die φ wahr macht.
- Problem: $\varphi|_{01}$ kann wahr sein, ohne dass $\varphi|_{00}$ wahr ist. Im Selbstreduktionsbaum ist $\varphi|_{01}$ Kind von $\varphi|_{00}$.
- Alle $u \preceq w$ müssen also auch in L liegen. Diese Konstruktion heißt *leftset*.

Die Sprache L

$$L = \left\{ \langle \varphi | w \rangle \;\middle|\; |w| = |\mathrm{Var}(\varphi)|, \left(\exists u \in \{0,1\}^{|\mathrm{Var}(\varphi)|} : w \preceq u, \varphi|_u = 1 \right) \right\}$$

Konstruktion der Sprache L

- Sei $|\mathrm{Var}(\varphi)|$ die Anzahl der Variablen in φ,
  die lexikographische Ordnung für Bitstrings sei $\preceq$.
- L enthält Tupel $\langle \varphi | w \rangle$ aus einer Formel φ
  und einer Belegung w, die φ wahr macht.
- Problem: $\varphi|_{01}$ kann wahr sein, ohne dass $\varphi|_{00}$ wahr ist.
  Im Selbstreduktionsbaum ist $\varphi|_{01}$ Kind von $\varphi|_{00}$.
- Alle $u \preceq w$ müssen also auch in L liegen.
  Diese Konstruktion heißt *leftset*.

Die Sprache L

$$L = \left\{ \langle \varphi | w \rangle \;\middle|\; |w| = |\mathrm{Var}(\varphi)|, \left(\exists u \in \{0,1\}^{|\mathrm{Var}(\varphi)|} : w \preceq u, \varphi|_u = 1 \right) \right\}$$

Konstruktion der Sprache L

- Sei $|\mathrm{Var}(\varphi)|$ die Anzahl der Variablen in φ,
  die lexikographische Ordnung für Bitstrings sei $\preceq$.
- L enthält Tupel $\langle \varphi | w \rangle$ aus einer Formel φ
  und einer Belegung w, die φ wahr macht.
- Problem: $\varphi|_{01}$ kann wahr sein, ohne dass $\varphi|_{00}$ wahr ist.
  Im Selbstreduktionsbaum ist $\varphi|_{01}$ Kind von $\varphi|_{00}$.
- Alle $u \preceq w$ müssen also auch in L liegen.
  Diese Konstruktion heißt *leftset*.

Die Sprache L

$$L = \left\{ \langle \varphi | w \rangle \;\middle|\; |w| = |\mathrm{Var}(\varphi)|, \left(\exists u \in \{0,1\}^{|\mathrm{Var}(\varphi)|} : w \preceq u, \varphi|_u = 1 \right) \right\}$$

Konstruktion der Sprache L

- Sei $|\mathrm{Var}(\varphi)|$ die Anzahl der Variablen in φ,
  die lexikographische Ordnung für Bitstrings sei $\preceq$.
- L enthält Tupel $\langle \varphi | w \rangle$ aus einer Formel φ
  und einer Belegung w, die φ wahr macht.
- Problem: $\varphi|_{01}$ kann wahr sein, ohne dass $\varphi|_{00}$ wahr ist.
  Im Selbstreduktionsbaum ist $\varphi|_{01}$ Kind von $\varphi|_{00}$.
- Alle $u \preceq w$ müssen also auch in L liegen.
  Diese Konstruktion heißt *leftset*.

Die Sprache L

$$L = \left\{ \langle \varphi | w \rangle \;\middle|\; |w| = |\mathrm{Var}(\varphi)|, \left(\exists u \in \{0,1\}^{|\mathrm{Var}(\varphi)|} : w \preceq u, \varphi|_u = 1 \right) \right\}$$

Eigenschaften der Sprache L

- Es ist klar, dass $L \in \mathrm{NP}$
  (Für Eingabe $\langle \varphi, w \rangle$ rate ein $u \succeq w$ und prüfe, ob $\varphi|_u = 1$)
- Deswegen gilt $L \leq_m^{\mathrm{P}} \mathrm{SAT}$
- Gemäß unserer Annahme, dass es eine NP-harte dünne Sprache S gibt, gilt auch $L \leq_m^{\mathrm{P}} S$ via einer Funktion $f \in \mathrm{FP}$.
- Die Laufzeit von $f(\langle \varphi | w \rangle)$ sei für $|w| = |Var(\varphi)|$ durch $p_f(|\varphi|)$ beschränkt.
- Polynomialzeit ist in diesem Beweis ausreichend,
  es gilt natürlich auch $L \leq_m^{\mathrm{L}} \mathrm{SAT}$ und $L \leq_m^{\mathrm{L}} S$.

Eigenschaften der Sprache L

- Es ist klar, dass $L \in \mathrm{NP}$
  (Für Eingabe $\langle \varphi, w \rangle$ rate ein $u \succeq w$ und prüfe, ob $\varphi|_u = 1$)
- Deswegen gilt $L \leq_m^{\mathrm{P}} \mathrm{SAT}$
- Gemäß unserer Annahme, dass es eine NP-harte dünne Sprache S gibt, gilt auch $L \leq_m^{\mathrm{P}} S$ via einer Funktion $f \in \mathrm{FP}$.
- Die Laufzeit von $f(\langle \varphi | w \rangle)$ sei für $|w| = |Var(\varphi)|$ durch $p_f(|\varphi|)$ beschränkt.
- Polynomialzeit ist in diesem Beweis ausreichend,
  es gilt natürlich auch $L \leq_m^{\mathrm{L}} \mathrm{SAT}$ und $L \leq_m^{\mathrm{L}} S$.

Eigenschaften der Sprache L

- Es ist klar, dass $L \in \mathrm{NP}$
  (Für Eingabe $\langle \varphi, w \rangle$ rate ein $u \succeq w$ und prüfe, ob $\varphi|_u = 1$)
- Deswegen gilt $L \leq_m^{\mathrm{P}} \mathrm{SAT}$
- Gemäß unserer Annahme, dass es eine NP-harte dünne Sprache S gibt, gilt auch $L \leq_m^{\mathrm{P}} S$ via einer Funktion $f \in \mathrm{FP}$.
- Die Laufzeit von $f(\langle \varphi | w \rangle)$ sei für $|w| = |Var(\varphi)|$ durch $p_f(|\varphi|)$ beschränkt.
- Polynomialzeit ist in diesem Beweis ausreichend,
  es gilt natürlich auch $L \leq_m^{\mathrm{L}} \mathrm{SAT}$ und $L \leq_m^{\mathrm{L}} S$.

Eigenschaften der Sprache L

- Es ist klar, dass $L \in \mathrm{NP}$
  (Für Eingabe $\langle \varphi, w \rangle$ rate ein $u \succeq w$ und prüfe, ob $\varphi|_u = 1$)
- Deswegen gilt $L \leq_m^{\mathrm{P}} \mathrm{SAT}$
- Gemäß unserer Annahme, dass es eine NP-harte dünne Sprache S gibt, gilt auch $L \leq_m^{\mathrm{P}} S$ via einer Funktion $f \in \mathrm{FP}$.
- Die Laufzeit von $f(\langle \varphi | w \rangle)$ sei für $|w| = |Var(\varphi)|$ durch $p_f(|\varphi|)$ beschränkt.
- Polynomialzeit ist in diesem Beweis ausreichend, es gilt natürlich auch $L \leq_m^{\mathrm{L}} \mathrm{SAT}$ und $L \leq_m^{\mathrm{L}} S$.

Eigenschaften der Sprache L

- Es ist klar, dass $L \in \mathrm{NP}$
  (Für Eingabe $\langle \varphi, w \rangle$ rate ein $u \succeq w$ und prüfe, ob $\varphi|_u = 1$)
- Deswegen gilt $L \leq_m^{\mathrm{P}} \mathrm{SAT}$
- Gemäß unserer Annahme, dass es eine NP-harte dünne Sprache S gibt, gilt auch $L \leq_m^{\mathrm{P}} S$ via einer Funktion $f \in \mathrm{FP}$.
- Die Laufzeit von $f(\langle \varphi | w \rangle)$ sei für $|w| = |Var(\varphi)|$ durch $p_f(|\varphi|)$ beschränkt.
- Polynomialzeit ist in diesem Beweis ausreichend,
  es gilt natürlich auch $L \leq_m^{\mathrm{L}} \mathrm{SAT}$ und $L \leq_m^{\mathrm{L}} S$.

Gliederung

Algorithmenidee

- Ziel: Algorithmus, der SAT mit f in Polynomialzeit entscheidet
- Für Eingabe φ bauen wir ebenenweise den Selbstreduktionsbaum für φ auf. (Breitensuche)
- Immer wenn eine Ebene aufgebaut ist, werfen wir so viele Knoten weg, dass nur polynomiell viele übrigbleiben.
- Wir müssen sicherstellen, dass wir dadurch das Akzeptanzverhalten nicht verändern.

Algorithmenidee

- Ziel: Algorithmus, der SAT mit f in Polynomialzeit entscheidet
- Für Eingabe φ bauen wir ebenenweise den Selbstreduktionsbaum für φ auf. (Breitensuche)
- Immer wenn eine Ebene aufgebaut ist, werfen wir so viele Knoten weg, dass nur polynomiell viele übrigbleiben.
- Wir müssen sicherstellen, dass wir dadurch das Akzeptanzverhalten nicht verändern.

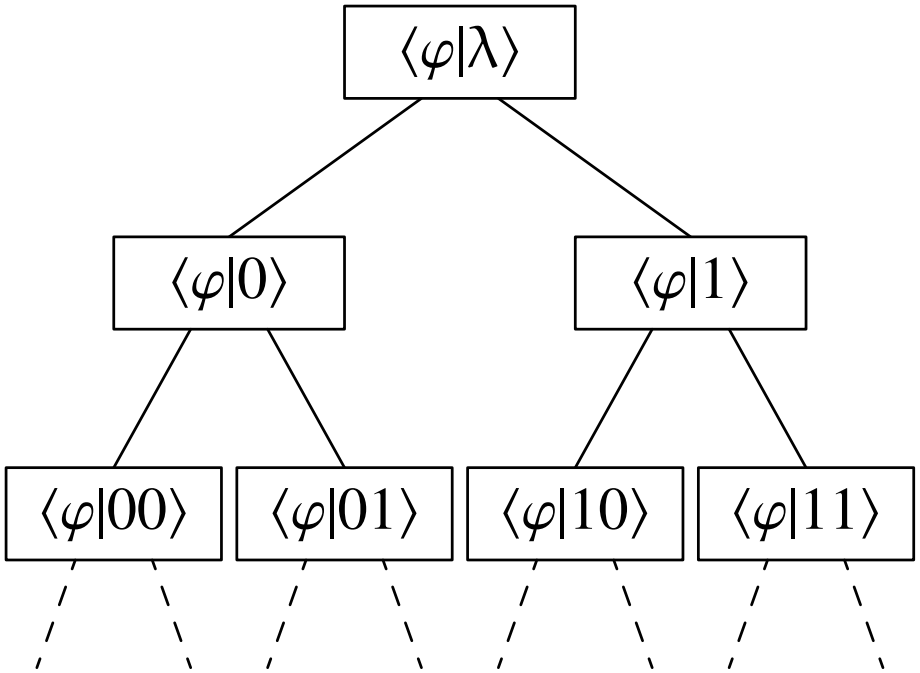
Algorithmenidee

- Ziel: Algorithmus, der SAT mit f in Polynomialzeit entscheidet
- Für Eingabe φ bauen wir ebenenweise den Selbstreduktionsbaum für φ auf. (Breitensuche)
- Immer wenn eine Ebene aufgebaut ist, werfen wir so viele Knoten weg, dass nur polynomiell viele übrigbleiben.
- Wir müssen sicherstellen, dass wir dadurch das Akzeptanzverhalten nicht verändern.

Algorithmenidee

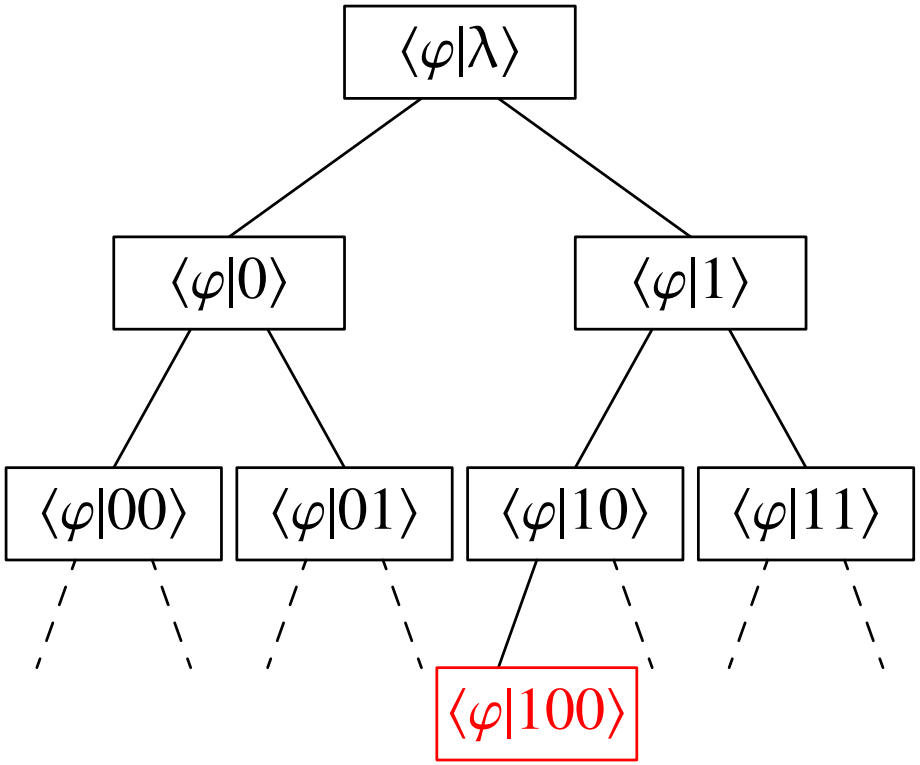
- Ziel: Algorithmus, der SAT mit f in Polynomialzeit entscheidet
- Für Eingabe φ bauen wir ebenenweise den Selbstreduktionsbaum für φ auf. (Breitensuche)
- Immer wenn eine Ebene aufgebaut ist, werfen wir so viele Knoten weg, dass nur polynomiell viele übrigbleiben.
- Wir müssen sicherstellen, dass wir dadurch das Akzeptanzverhalten nicht verändern.

Der Selbstreduktionsbaum



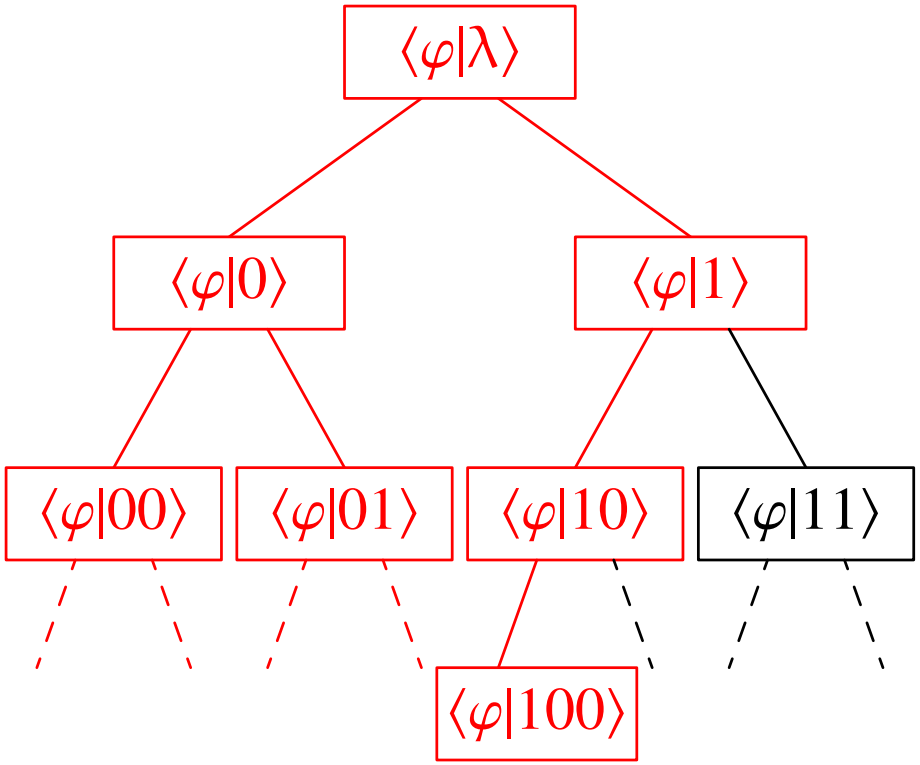
- Die Nachfolger aller Knoten sind jeweils nach der Größe von w bezüglich $\preceq$ geordnet.
- Wenn φ erfüllbar, existiert ein größtes Blatt $\langle \varphi | w_0 \rangle$ mit $\varphi|_{w_0} = 1$
- Genau alle Knoten auf und links von dem Pfad von der Wurzel zu $\langle \varphi | w_0 \rangle$ sind in L.
- Diese Eigenschaft ist wichtig für die Konstruktion des Algorithmus.

Der Selbstreduktionsbaum



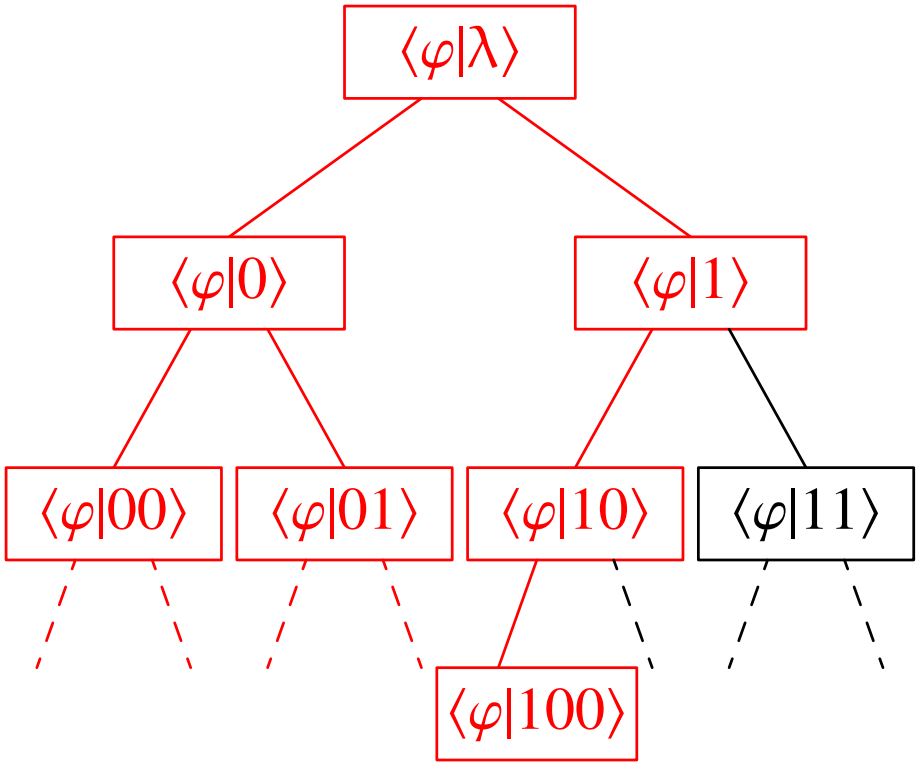
- Die Nachfolger aller Knoten sind jeweils nach der Größe von w bezüglich $\preceq$ geordnet.
- Wenn φ erfüllbar, existiert ein größtes Blatt $\langle \varphi | w_0 \rangle$ mit $\varphi|_{w_0} = 1$
- Genau alle Knoten auf und links von dem Pfad von der Wurzel zu $\langle \varphi | w_0 \rangle$ sind in L.
- Diese Eigenschaft ist wichtig für die Konstruktion des Algorithmus.

Der Selbstreduktionsbaum



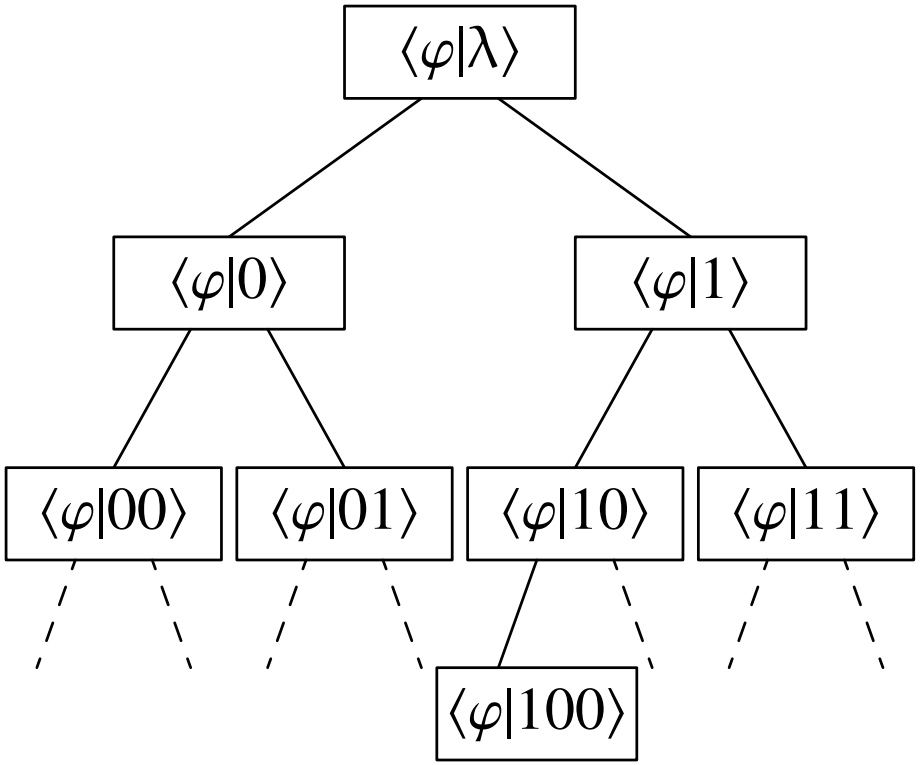
- Die Nachfolger aller Knoten sind jeweils nach der Größe von w bezüglich $\preceq$ geordnet.
- Wenn φ erfüllbar, existiert ein größtes Blatt $\langle \varphi | w_0 \rangle$ mit $\varphi|_{w_0} = 1$
- Genau alle Knoten auf und links von dem Pfad von der Wurzel zu $\langle \varphi | w_0 \rangle$ sind in L.
- Diese Eigenschaft ist wichtig für die Konstruktion des Algorithmus.

Der Selbstreduktionsbaum



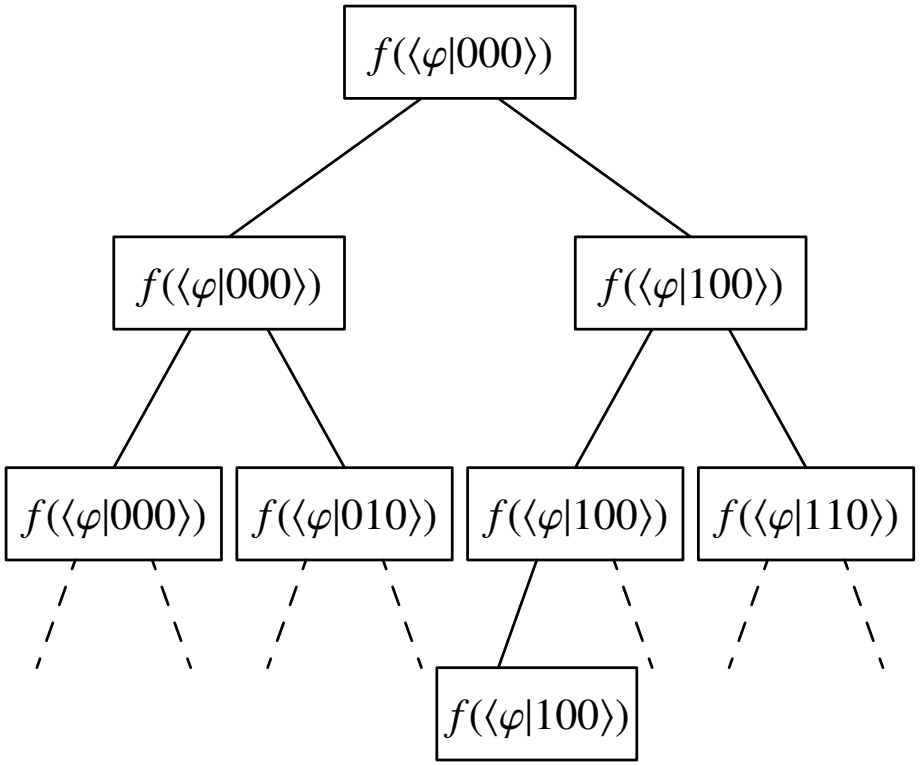
- Die Nachfolger aller Knoten sind jeweils nach der Größe von w bezüglich $\preceq$ geordnet.
- Wenn φ erfüllbar, existiert ein größtes Blatt $\langle \varphi | w_0 \rangle$ mit $\varphi|_{w_0} = 1$
- Genau alle Knoten auf und links von dem Pfad von der Wurzel zu $\langle \varphi | w_0 \rangle$ sind in L.
- Diese Eigenschaft ist wichtig für die Konstruktion des Algorithmus.

Label am Selbstreduktionsbaum



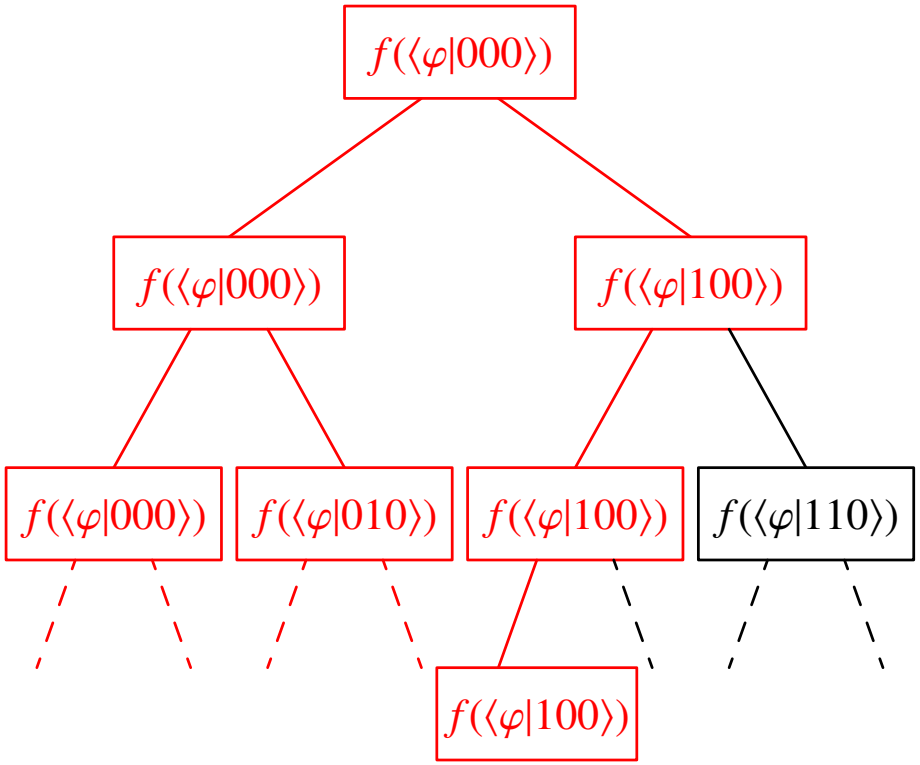
- Jedem Knoten $\langle \varphi | w \rangle$ wird ein Label $f(\langle \varphi | w' \rangle)$ zugewiesen.
- Genau die Label sind in S, deren Knoten in L sind.
- Es gibt nur polynomiell viele unterschiedliche Label, die in S liegen.
- Das können wir beim angestrebten Abschneiden von Teilbäumen ausnutzen.

Label am Selbstreduktionsbaum



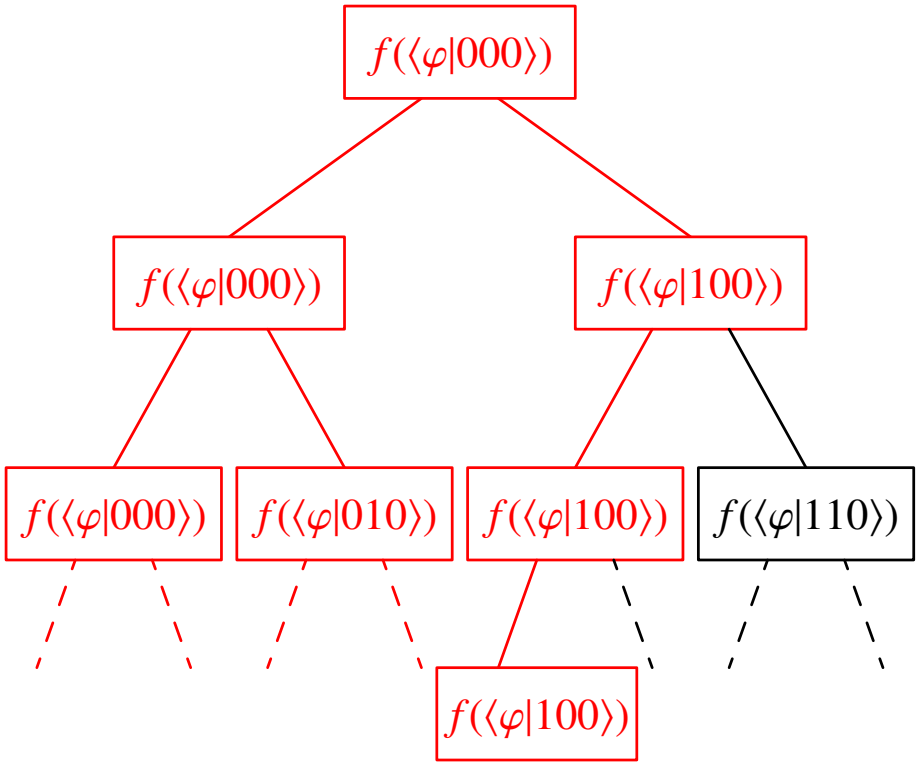
- Jedem Knoten $\langle\varphi|w\rangle$ wird ein Label $f(\langle\varphi|w'\rangle)$ zugewiesen.
- Genau die Label sind in S, deren Knoten in L sind.
- Es gibt nur polynomiell viele unterschiedliche Label, die in S liegen.
- Das können wir beim angestrebten Abschneiden von Teilbäumen ausnutzen.

Label am Selbstreduktionsbaum



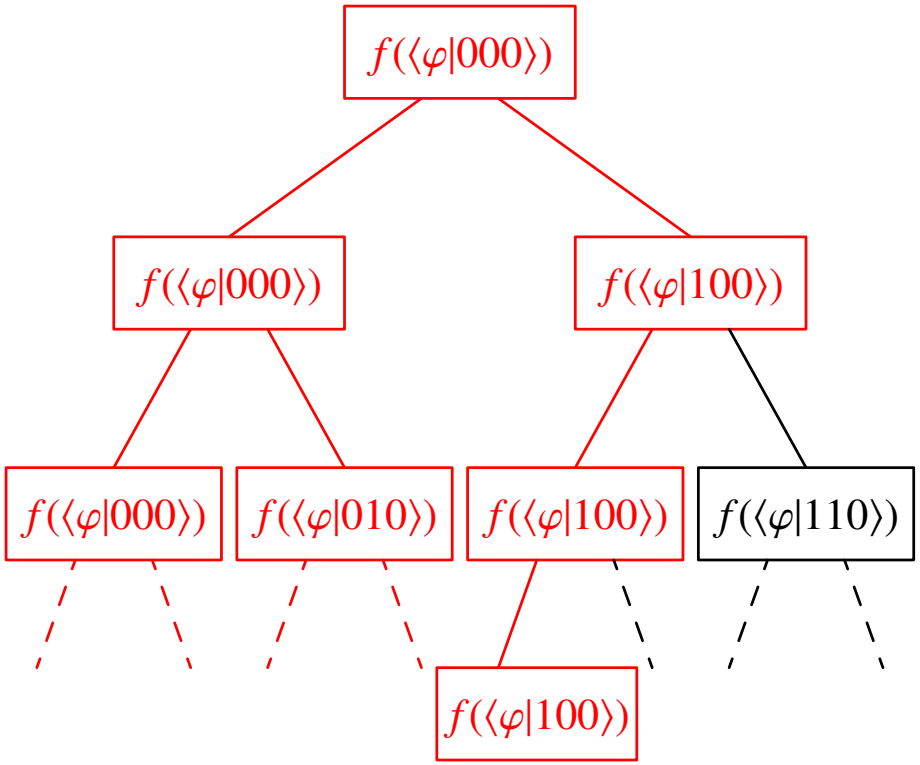
- Jedem Knoten $\langle\varphi|w\rangle$ wird ein Label $f(\langle\varphi|w'\rangle)$ zugewiesen.
- Genau die Label sind in S, deren Knoten in L sind.
- Es gibt nur polynomiell viele unterschiedliche Label, die in S liegen.
- Das können wir beim angestrebten Abschneiden von Teilbäumen ausnutzen.

Label am Selbstreduktionsbaum



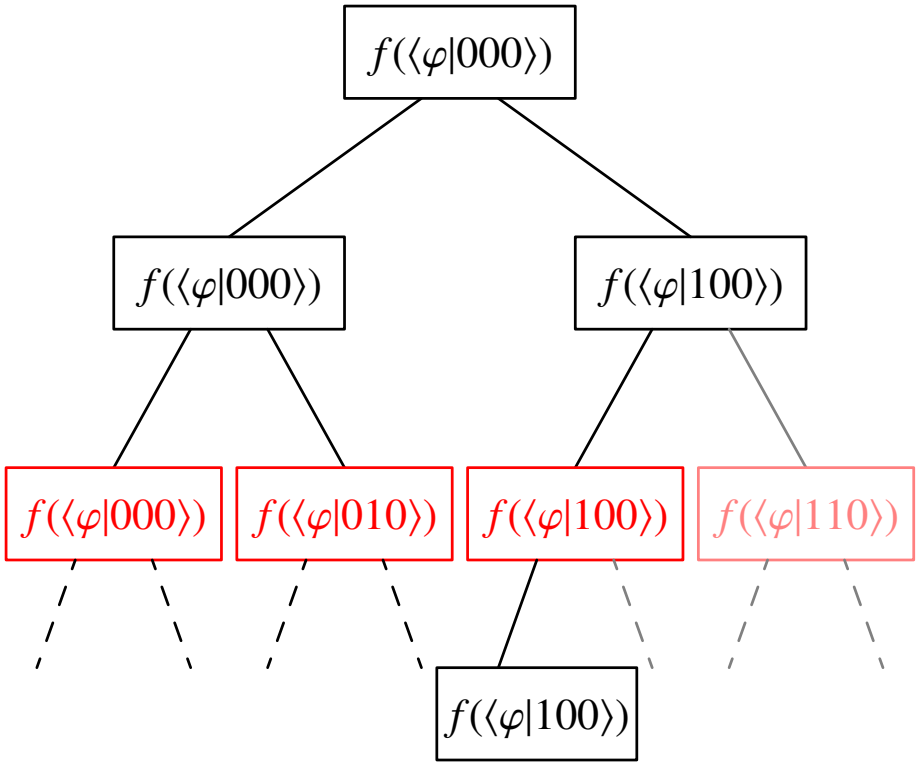
- Jedem Knoten $\langle\varphi|w\rangle$ wird ein Label $f(\langle\varphi|w'\rangle)$ zugewiesen.
- Genau die Label sind in S, deren Knoten in L sind.
- Es gibt nur polynomiell viele unterschiedliche Label, die in S liegen.
- Das können wir beim angestrebten Abschneiden von Teilbäumen ausnutzen.

Label am Selbstreduktionsbaum



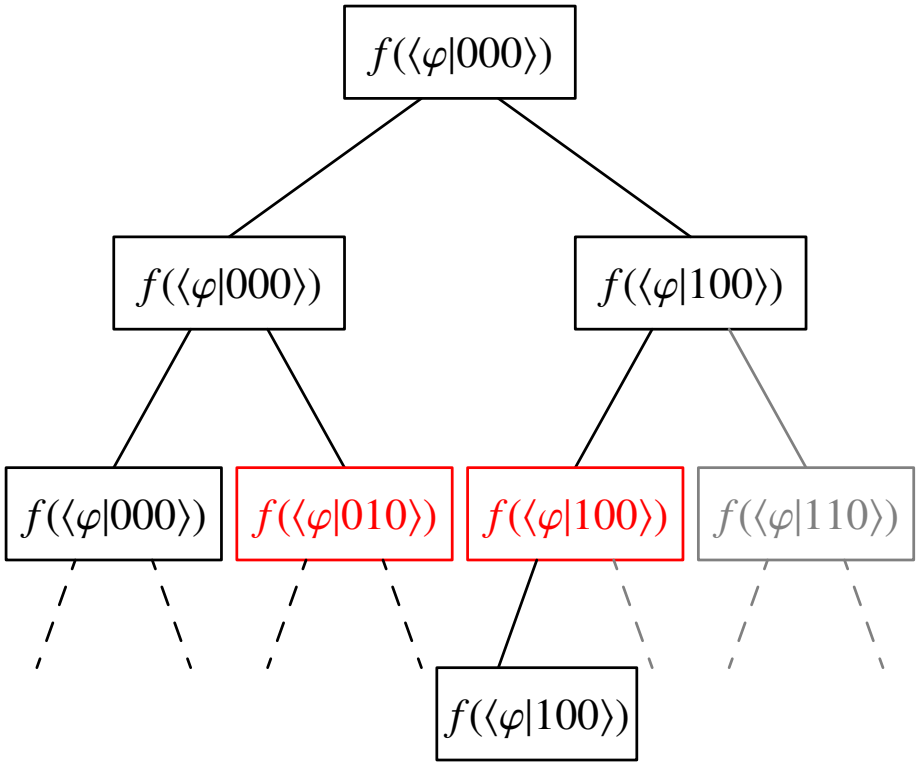
- Jedem Knoten $\langle\varphi|w\rangle$ wird ein Label $f(\langle\varphi|w'\rangle)$ zugewiesen.
- Genau die Label sind in S, deren Knoten in L sind.
- Es gibt nur polynomiell viele unterschiedliche Label, die in S liegen.
- Das können wir beim angestrebten Abschneiden von Teilbäumen ausnutzen.

Abschneiden *(prune)* durch gleiche Label



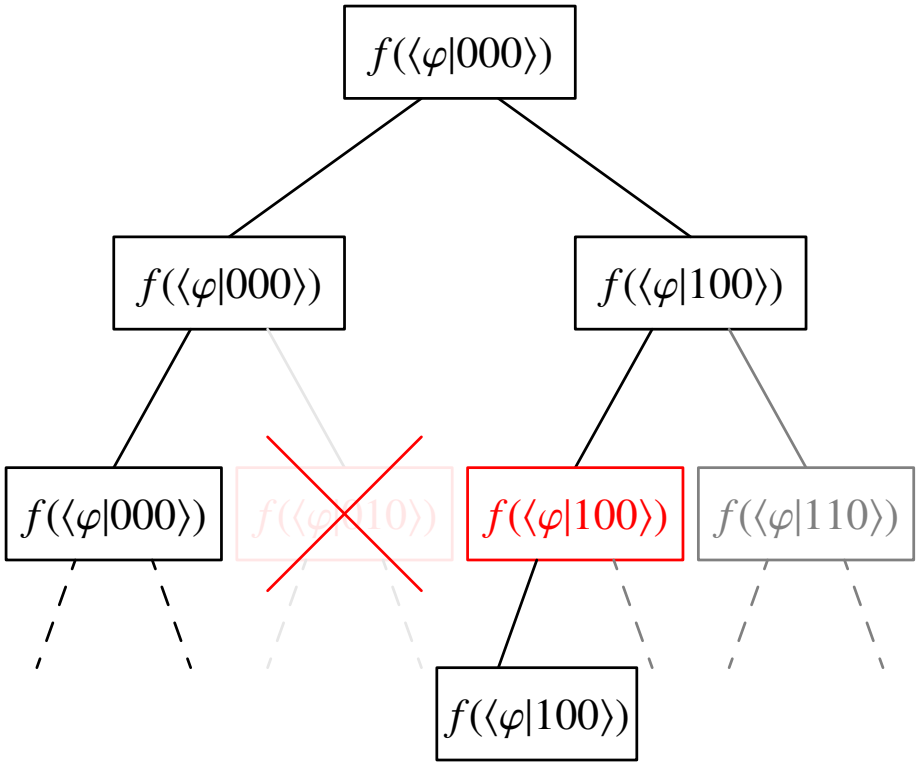
- Nach dem Erzeugen einer neuen Ebene sollen nun Teilbäume abgeschnitten werden.
- $f(\langle\varphi|w_1'\rangle) = f(\langle\varphi|w_2'\rangle) \Rightarrow (\langle\varphi|w_1'\rangle \in L \Leftrightarrow \langle\varphi|w_2'\rangle \in L)$
- Wenn zwei Knoten das gleiche Label haben, kann der kleinere weggelassen werden.
- Das größte Blatt in L kann nicht an ihm hängen.

Abschneiden *(prune)* durch gleiche Label



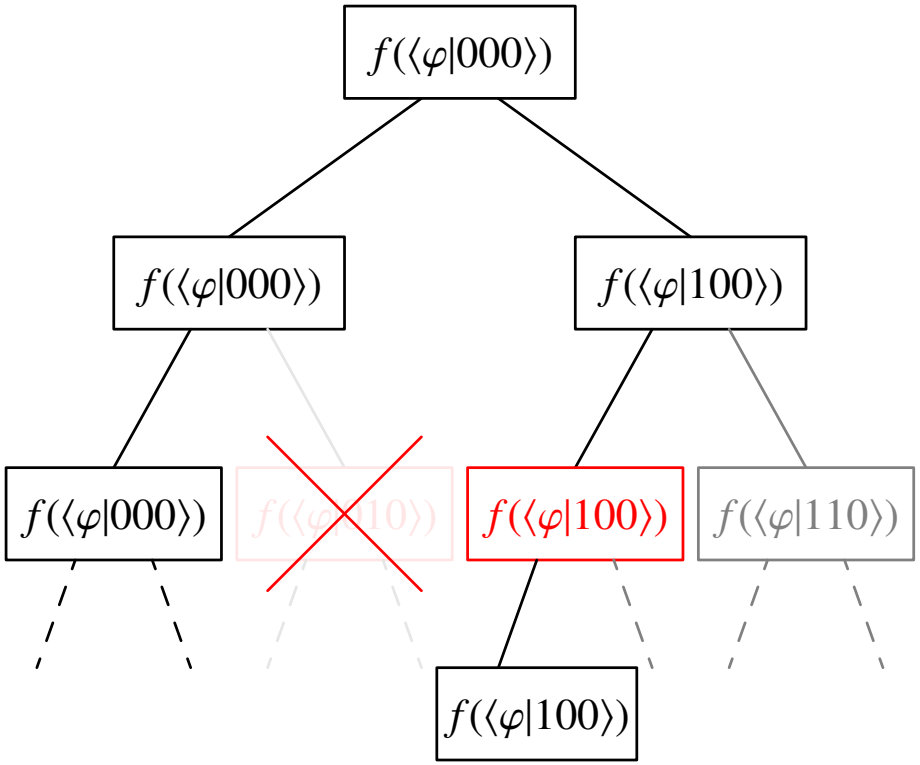
- Nach dem Erzeugen einer neuen Ebene sollen nun Teilbäume abgeschnitten werden.
- $f(\langle\varphi|w_1'\rangle) = f(\langle\varphi|w_2'\rangle)$
  $\Rightarrow (\langle\varphi|w_1'\rangle \in L \Leftrightarrow \langle\varphi|w_2'\rangle \in L)$
- Wenn zwei Knoten das gleiche Label haben, kann der kleinere weggelassen werden.
- Das größte Blatt in L kann nicht an ihm hängen.

Abschneiden *(prune)* durch gleiche Label



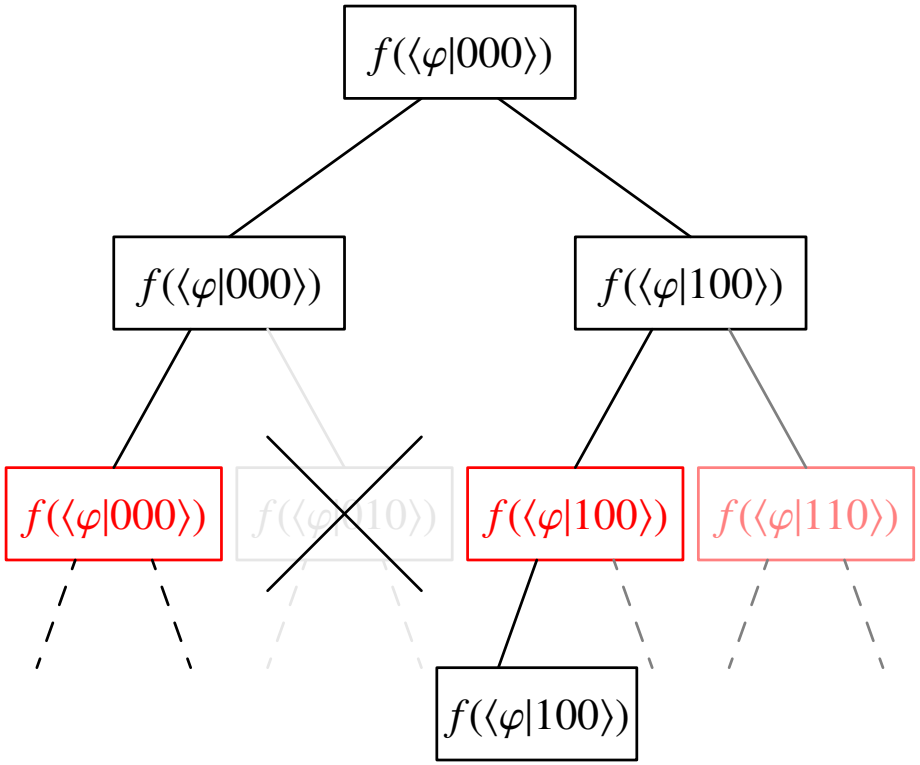
- Nach dem Erzeugen einer neuen Ebene sollen nun Teilbäume abgeschnitten werden.
- $f(\langle\varphi|w_1'\rangle) = f(\langle\varphi|w_2'\rangle)$
  $\Rightarrow (\langle\varphi|w_1'\rangle \in L \Leftrightarrow \langle\varphi|w_2'\rangle \in L)$
- Wenn zwei Knoten das gleiche Label haben, kann der kleinere weggelassen werden.
- Das größte Blatt in L kann nicht an ihm hängen.

Abschneiden *(prune)* durch gleiche Label



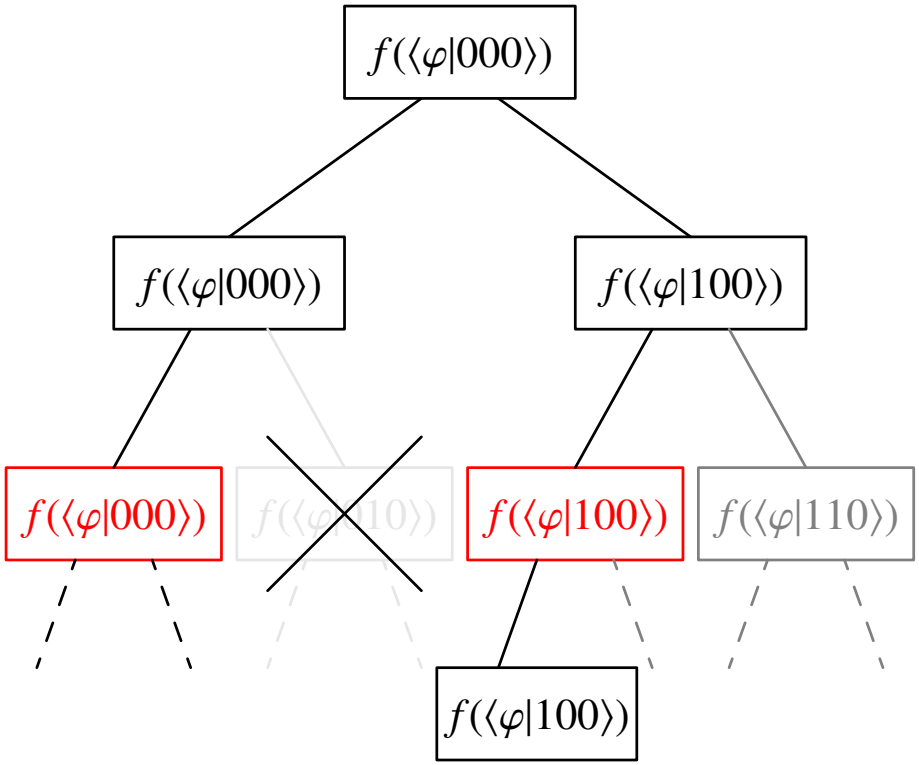
- Nach dem Erzeugen einer neuen Ebene sollen nun Teilbäume abgeschnitten werden.
- $f(\langle\varphi|w_1'\rangle) = f(\langle\varphi|w_2'\rangle)$
  $\Rightarrow (\langle\varphi|w_1'\rangle \in L \Leftrightarrow \langle\varphi|w_2'\rangle \in L)$
- Wenn zwei Knoten das gleiche Label haben, kann der kleinere weggelassen werden.
- Das größte Blatt in L kann nicht an ihm hängen.

Abschneiden *(prune)* durch Dünnheit



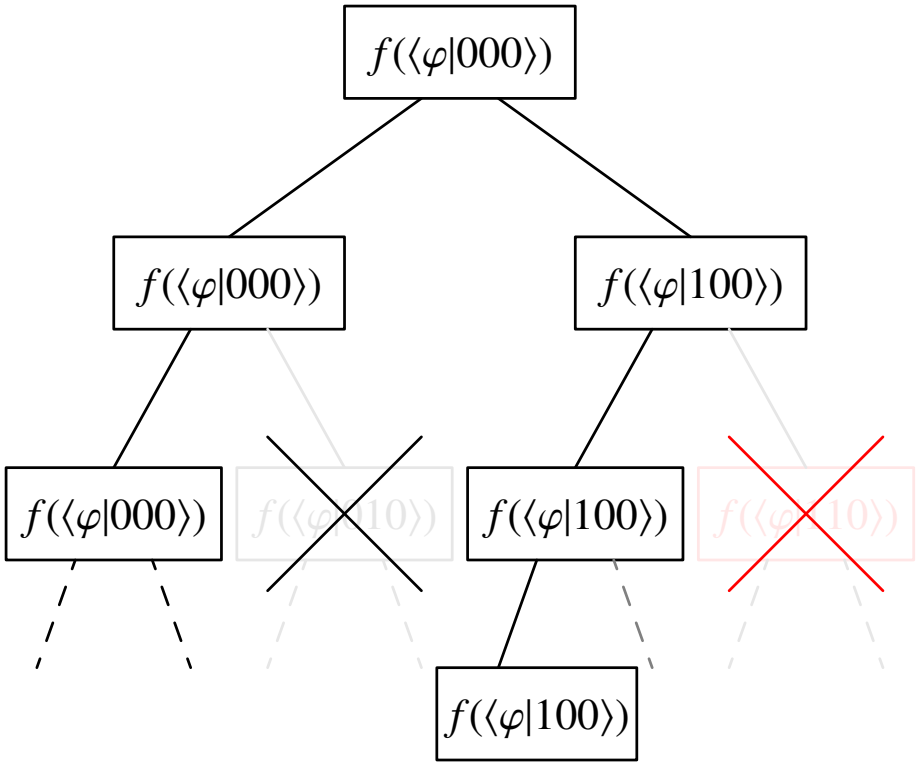
- Es kann immer noch mehr als polynomiell viele Knoten in der Ebene geben.
- Nur die ersten $C_S(|\langle\varphi|0^{|\mathrm{Var}(\varphi)|}\rangle|)$ Label können in S liegen.
- Alle größeren Knoten können weggelassen werden, weil das größte Blatt in L nicht an ihnen hängen kann.
- Es bleiben nur polynomiell viele Knoten in der Ebene übrig.

Abschneiden *(prune)* durch Dünnheit



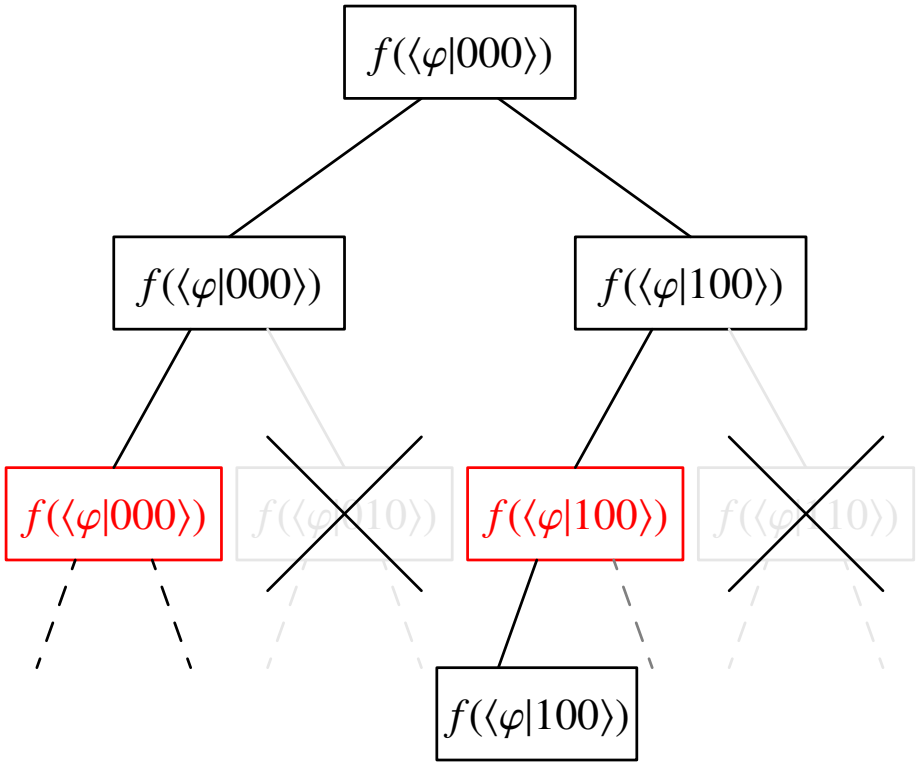
- Es kann immer noch mehr als polynomiell viele Knoten in der Ebene geben.
- Nur die ersten $C_S(|\langle\varphi|0^{|\mathrm{Var}(\varphi)|}\rangle|)$ Label können in S liegen.
- Alle größeren Knoten können weggelassen werden, weil das größte Blatt in L nicht an ihnen hängen kann.
- Es bleiben nur polynomiell viele Knoten in der Ebene übrig.

Abschneiden *(prune)* durch Dünnheit



- Es kann immer noch mehr als polynomiell viele Knoten in der Ebene geben.
- Nur die ersten $C_S(|\langle\varphi|0^{|\mathrm{Var}(\varphi)|}\rangle|)$ Label können in S liegen.
- Alle größeren Knoten können weggelassen werden, weil das größte Blatt in L nicht an ihnen hängen kann.
- Es bleiben nur polynomiell viele Knoten in der Ebene übrig.

Abschneiden *(prune)* durch Dünnheit



- Es kann immer noch mehr als polynomiell viele Knoten in der Ebene geben.
- Nur die ersten $C_S(|\langle\varphi|0^{|\mathrm{Var}(\varphi)|}\rangle|)$ Label können in S liegen.
- Alle größeren Knoten können weggelassen werden, weil das größte Blatt in L nicht an ihnen hängen kann.
- Es bleiben nur polynomiell viele Knoten in der Ebene übrig.

Der Algorithmus

Algorithmus für SAT unter Kenntnis von f

Eingabe φ.

1. Setze $W_0 := \{\langle\varphi|\lambda\rangle\}$
2. Für $i = 1$ bis $|Var(\varphi)| - 1$:
   1. Setze $W_i := \{\langle\varphi|u0\rangle, \langle\varphi|u1\rangle \mid \langle\varphi|u\rangle \in W_{i-1}\}$
   2. Für jede Kombination $\langle\varphi|u\rangle, \langle\varphi|v\rangle \in W_i$ mit gleichem Label lösche das kleinere Element aus W_i
   3. Solange $|W_i| > p_S(p_f(|\varphi|))$, lösche das größte Element aus W_i
3. Setze $W_{|Var(\varphi)|} := \{\langle\varphi|u0\rangle, \langle\varphi|u1\rangle \mid \langle\varphi|u\rangle \in W_{Var-1}\}$
4. Wenn es ein $\langle\varphi|u\rangle \in W_{|Var(\varphi)|}$ mit $\varphi|_u = 1$ gibt, akzeptiere, sonst sonst verwerfe.

Polynomielle Schranken: $p_f(|\varphi|)$ für $f(\langle\varphi|u\rangle)$ und $p_S(n)$ für $\Sigma_{k=0}^{n} C_S(k)$

Laufzeit und Korrektheit des Algorithmus

- Die Laufzeit ist durch ein Polynom beschränkt, weil die W_i in der Größe durch $p_S(p_f(|\varphi|))$ beschränkt sind.
- Wenn der Algorithmus akzeptiert, liegt φ offensichtlich in SAT (es wurde eine gültigmachende Variablenbelegung gefunden).
- Nun müssen wir noch noch zeigen, dass alle $\varphi \in \mathrm{SAT}$ akzeptiert werden.
  - Wenn $\varphi \in \mathrm{SAT}$, dann existiert ein größtes Blatt $\langle \varphi | u \rangle$ mit $\varphi|_u = 1$.
  - Wir haben bereits argumentiert, dass die Knoten auf dem Weg zu diesem Blatt nicht aus den W_i entfernt werden.
  - Also liegt mit $\langle \varphi | u \rangle$ ein Element in $W_{|\mathrm{Var}(\varphi)|}$, das zu 1 ausgewertet wird.
  - Damit akzeptiert der Algorithmus. □

Laufzeit und Korrektheit des Algorithmus

- Die Laufzeit ist durch ein Polynom beschränkt,
  weil die W_i in der Größe durch $p_S(p_f(|\varphi|))$ beschränkt sind.
- Wenn der Algorithmus akzeptiert, liegt φ offensichtlich in SAT (es wurde eine gültigmachende Variablenbelegung gefunden).
- Nun müssen wir noch noch zeigen,
  dass alle $\varphi \in \mathrm{SAT}$ akzeptiert werden.
  - Wenn $\varphi \in \mathrm{SAT}$, dann existiert ein größtes Blatt $\langle \varphi | u \rangle$ mit $\varphi|_u = 1$.
  - Wir haben bereits argumentiert, dass die Knoten auf dem Weg zu diesem Blatt nicht aus den W_i entfernt werden.
  - Also liegt mit $\langle \varphi | u \rangle$ ein Element in $W_{|\mathrm{Var}(\varphi)|}$, das zu 1 ausgewertet wird.
  - Damit akzeptiert der Algorithmus. □

Laufzeit und Korrektheit des Algorithmus

- Die Laufzeit ist durch ein Polynom beschränkt,
  weil die W_i in der Größe durch $p_S(p_f(|\varphi|))$ beschränkt sind.
- Wenn der Algorithmus akzeptiert, liegt φ offensichtlich in SAT (es wurde eine gültigmachende Variablenbelegung gefunden).
- Nun müssen wir noch noch zeigen,
  dass alle $\varphi \in \mathrm{SAT}$ akzeptiert werden.
  - Wenn $\varphi \in \mathrm{SAT}$, dann existiert ein größtes Blatt $\langle \varphi | u \rangle$ mit $\varphi|_u = 1$.
  - Wir haben bereits argumentiert, dass die Knoten auf dem Weg zu diesem Blatt nicht aus den W_i entfernt werden.
  - Also liegt mit $\langle \varphi | u \rangle$ ein Element in $W_{|\mathrm{Var}(\varphi)|}$, das zu 1 ausgewertet wird.
  - Damit akzeptiert der Algorithmus. □

Laufzeit und Korrektheit des Algorithmus

- Die Laufzeit ist durch ein Polynom beschränkt,
  weil die W_i in der Größe durch $p_S(p_f(|\varphi|))$ beschränkt sind.
- Wenn der Algorithmus akzeptiert, liegt φ offensichtlich in SAT (es wurde eine gültigmachende Variablenbelegung gefunden).
- Nun müssen wir noch noch zeigen,
  dass alle $\varphi \in \mathrm{SAT}$ akzeptiert werden.
  - Wenn $\varphi \in \mathrm{SAT}$, dann existiert ein größtes Blatt $\langle \varphi | u \rangle$ mit $\varphi|_u = 1$.
  - Wir haben bereits argumentiert, dass die Knoten auf dem Weg zu diesem Blatt nicht aus den W_i entfernt werden.
  - Also liegt mit $\langle \varphi | u \rangle$ ein Element in $W_{|\mathrm{Var}(\varphi)|}$, das zu 1 ausgewertet wird.
  - Damit akzeptiert der Algorithmus. □

Laufzeit und Korrektheit des Algorithmus

- Die Laufzeit ist durch ein Polynom beschränkt, weil die W_i in der Größe durch $p_S(p_f(|\varphi|))$ beschränkt sind.
- Wenn der Algorithmus akzeptiert, liegt φ offensichtlich in SAT (es wurde eine gültigmachende Variablenbelegung gefunden).
- Nun müssen wir noch noch zeigen, dass alle $\varphi \in \mathrm{SAT}$ akzeptiert werden.
  - Wenn $\varphi \in \mathrm{SAT}$, dann existiert ein größtes Blatt $\langle \varphi | u \rangle$ mit $\varphi|_u = 1$.
  - Wir haben bereits argumentiert, dass die Knoten auf dem Weg zu diesem Blatt nicht aus den W_i entfernt werden.
  - Also liegt mit $\langle \varphi | u \rangle$ ein Element in $W_{|\mathrm{Var}(\varphi)|}$, das zu 1 ausgewertet wird.
  - Damit akzeptiert der Algorithmus. □

Laufzeit und Korrektheit des Algorithmus

- Die Laufzeit ist durch ein Polynom beschränkt, weil die W_i in der Größe durch $p_S(p_f(|\varphi|))$ beschränkt sind.
- Wenn der Algorithmus akzeptiert, liegt φ offensichtlich in SAT (es wurde eine gültigmachende Variablenbelegung gefunden).
- Nun müssen wir noch noch zeigen, dass alle $\varphi \in \mathrm{SAT}$ akzeptiert werden.
  - Wenn $\varphi \in \mathrm{SAT}$, dann existiert ein größtes Blatt $\langle \varphi | u \rangle$ mit $\varphi|_u = 1$.
  - Wir haben bereits argumentiert, dass die Knoten auf dem Weg zu diesem Blatt nicht aus den W_i entfernt werden.
  - Also liegt mit $\langle \varphi | u \rangle$ ein Element in $W_{|\mathrm{Var}(\varphi)|}$, das zu 1 ausgewertet wird.
  - Damit akzeptiert der Algorithmus. □

Laufzeit und Korrektheit des Algorithmus

- Die Laufzeit ist durch ein Polynom beschränkt, weil die W_i in der Größe durch $p_S(p_f(|\varphi|))$ beschränkt sind.
- Wenn der Algorithmus akzeptiert, liegt φ offensichtlich in SAT (es wurde eine gültigmachende Variablenbelegung gefunden).
- Nun müssen wir noch noch zeigen, dass alle $\varphi \in \mathrm{SAT}$ akzeptiert werden.
  - Wenn $\varphi \in \mathrm{SAT}$, dann existiert ein größtes Blatt $\langle \varphi | u \rangle$ mit $\varphi|_u = 1$.
  - Wir haben bereits argumentiert, dass die Knoten auf dem Weg zu diesem Blatt nicht aus den W_i entfernt werden.
  - Also liegt mit $\langle \varphi | u \rangle$ ein Element in $W_{|\mathrm{Var}(\varphi)|}$, das zu 1 ausgewertet wird.
  - Damit akzeptiert der Algorithmus. □

Zusammenfassung

- Durch Kenntnis einer Reduktion auf eine dünne Sprache lässt sich die Suche nach einer akzeptierenden Berechnung auf polynomiellen Aufwand reduzieren

- Analog lässt sich auch zeigen, dass für beliebige $k \in \mathbb{N}$ dünne Sprachen nicht $\leq^{\mathrm{P}}_{k\text{-}tt}$-hart für NP sein können, es sei denn $\mathrm{P} = \mathrm{NP}$.

Literatur

- Erstveröffentlichung:
  Stephen R. Mahaney: *Sparse complete sets for NP: Solution of a conjecture of Berman and Hartmanis.*
  Journal of Computer and System Sciences, 25(2), 130-142, 1982
- Quelle für den hier vorgestellten Beweis:
  Ding-Zhu Du und Ker-I Ko:
  Theory of Computational Complexity,
  Wiley & Sons, 2000, pp. 261-262
- Alternativer Beweis von Ogihara und Watanabe
  aus dem Skript *Introduction to to Complexity Theory*
  der University of Wisconsin, Madison
  `www.cs.wisc.edu/~jyc/810notes/lecture11.pdf`