Diskrete Mathematik

Arne Dür Kurt Girstmair Simon Legner
Georg Moser Harald Zankl

Fakultät für Mathematik, Informatik und Physik @ UIBK
Sommersemester 2011

Zusammenfassung der letzten LV

Table-Filling Algorithmus

1. wenn p akzeptierend und q nicht dann markiere $\{p, q\}$ als unterscheidbar
2. sei a ein Eingabezeichen und
   - sei $\delta(p, a) = r$, $\delta(q, a) = s$
   - sodass $\{r, s\}$ markiert

   dann markiere $\{p, q\}$ als unterscheidbar

Fakten

- der Table-Filling Algorithmus ist korrekt
- das heißt, wenn das Paar $\{p, q\}$ nicht markiert werden kann, dann sind p und q äquivalent

Definition

DEA $A = (Q, \Sigma, \delta, q_0, F)$
konstruiere $B = (Q_B, \Sigma, \delta_B, q_B, F_B)$:

1. unerreichbare Zustände eliminieren
2. partitioniere die Zustände in äquivalente Blöcke
3. Q_B sind die verschiedenen Blöcke
4. $\forall$ S ein Block, $\forall$ a ein Eingabesymbol:
   $\exists$ Block T, sodass für alle $q \in S$ gilt $\delta(q, a) \in T$
   setze $\delta_B(S, a) = T$
5. q_B ist der Block, der q_0 enthält
6. F_B ist die Menge der Blöcke, die Zustände aus F enthalten

Satz

der Minimierungsalgorithmus ist optimal und eindeutig

Übersicht

Endliche Automaten

Automaten, reguläre Sprachen und Grammatiken, (nicht)-deterministische endliche Automaten, Teilmengenkonstruktion, ϵ-NEAs, Umwandlung endlicher Automaten in reguläre Ausdrücke, Pumpinglemma, Minimierung

Berechenbarkeitstheorie

Einführung in die Berechenbarkeitstheorie, Turingmaschinen, Entscheidungsprobleme, Universelle Maschinen und Diagonalisierung

Komplexitätstheorie

Einführung in die Komplexitätstheorie, die Klassen P und NP, logarithmisch platzbeschränkte Reduktionen, Speicherplatzkomplexität

Übersicht

Endliche Automaten

Automaten, reguläre Sprachen und Grammatiken, (nicht)-deterministische endliche Automaten, Teilmengenkonstruktion, ϵ-NEAs, Umwandlung endlicher Automaten in reguläre Ausdrücke, Pumpinglemma, Minimierung

Berechenbarkeitstheorie

Einführung in die Berechenbarkeitstheorie, Turingmaschinen, Entscheidungsprobleme, Universelle Maschinen und Diagonalisierung

Komplexitätstheorie

Einführung in die Komplexitätstheorie, die Klassen P und NP, logarithmisch platzbeschränkte Reduktionen, Speicherplatzkomplexität

Einführung in die Berechenbarkeitstheorie

Frage

Ist jedes Problem algorithmisch lösbar?

Einführung in die Berechenbarkeitstheorie

Frage

Ist jedes Problem algorithmisch lösbar?

Einführung in die Berechenbarkeitstheorie

Frage

Ist jedes Problem algorithmisch lösbar?

Antwort

Nein

Einführung in die Berechenbarkeitstheorie

Frage

Ist jedes Problem algorithmisch lösbar?

Antwort

Nein

Ein einfaches Programm

```
main()
{
        printf(‘‘hello, world\n’’);
}
```

Beispiel

betrachte Programm F

```
main()
{
  int n, summe = 3, x, y, z;
  scanf(``%d'', &n);
  while (1) {
    for (x=1; x <= summe-2; x++)
    for (y=1; y <= summe-x-1; y++) {
      z = summe - x - y;
      if (pow(x,n) + pow(y,n) == pow(z,n))
      printf(``hello, world\n'');
    } summe++;
  }
}
```

Beispiel

betrachte Programm F

```
main()
{
  int n, summe = 3, x, y, z;
  scanf(``%d'', &n);
  while (1) {
    for (x=1; x <= summe-2; x++)
    for (y=1; y <= summe-x-1; y++) {
      z = summe - x - y;
      if (pow(x,n) + pow(y,n) == pow(z,n))
      printf(``hello, world\n'');
    } summe++;
  }
}
```

Program F ist kein "hello, world"-Programm

Beispiel

betrachte Programm G

```
main()
{
  int n, x, y, z, test, summe=4;
  while (1) {
    test = 0;
    for (x=2; x <= summe; x++) {
      y = summe - x;
      if (is_prime(x) && is_prime(y)) test = 1;
    }
    if (!test) printf(‘‘hello, world\n’’);
    summe = summe + 2;
  }
}
```

Beispiel

betrachte Programm G

```
main()
{
  int n, x, y, z, test, summe=4;
  while (1) {
    test = 0;
    for (x=2; x <= summe; x++) {
      y = summe - x;
      if (is_prime(x) && is_prime(y)) test = 1;
    }
    if (!test) printf(‘‘hello, world\n’’);
    summe = summe + 2;
  }
}
```

Goldbach'sche Vermutung: *Jede gerade Zahl größer als 2 kann als Summe zweier Primzahlen geschrieben werden*

Beispiel

betrachte Programm G

```
main()
{
  int n, x, y, z, test, summe=4;
  while (1) {
    test = 0;
    for (x=2; x <= summe; x++) {
      y = summe - x;
      if (is_prime(x) && is_prime(y)) test = 1;
    }
    if (!test) printf(``hello, world\n'');
    summe = summe + 2;
  }
}
```

Program G ist wahrscheinlich kein "hello, world"-Programm

Satz

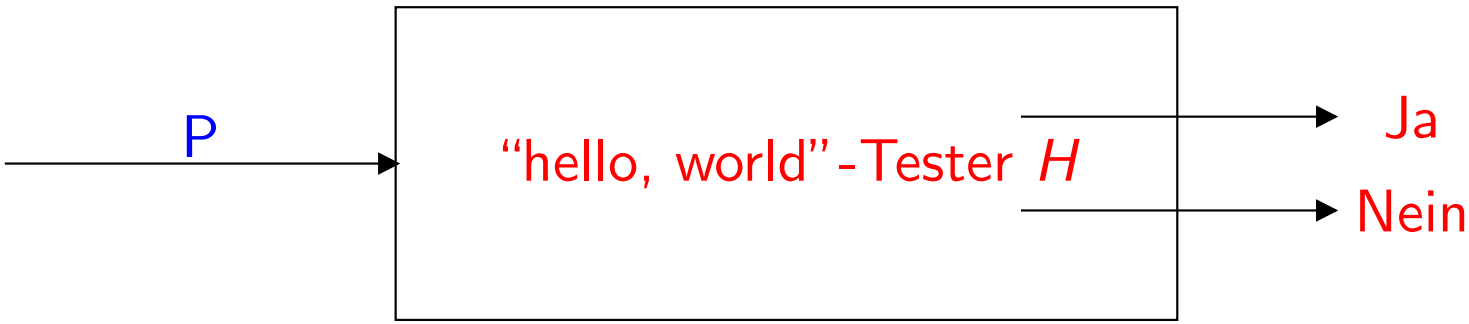
es kann kein Testprogramm für "hello, world"-Programme geben

Satz

es kann kein Testprogramm für "hello, world"-Programme geben

Beweisskizze.

- wir betrachten ein hypothetisches Programm H, das wir "hello, world"-Tester nennen

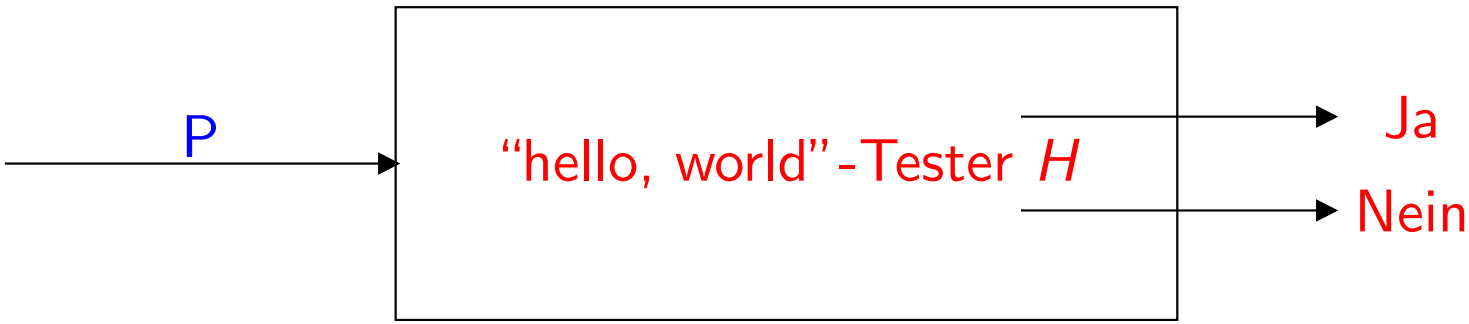


Satz

es kann kein Testprogramm für "hello, world"-Programme geben

Beweisskizze.

- wir betrachten ein hypothetisches Programm H, das wir "hello, world"-Tester nennen



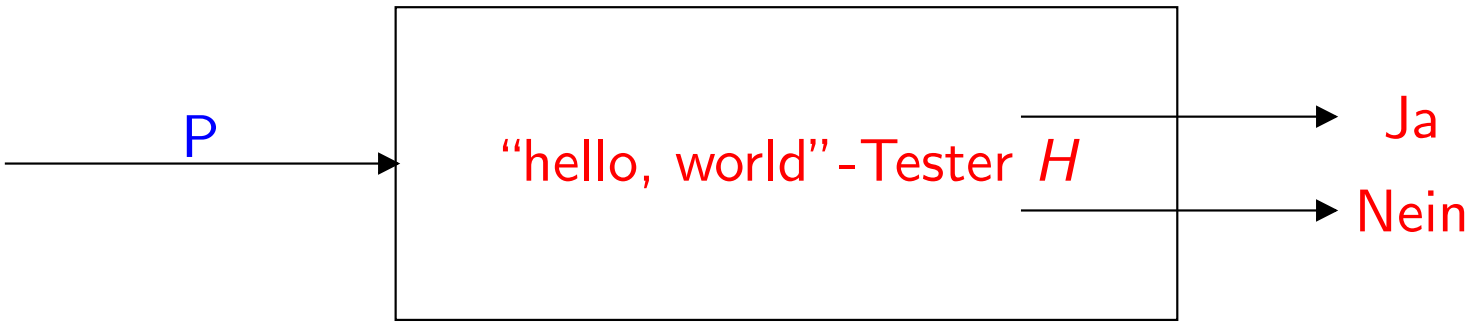
- man kann zeigen, dass ein "hello, world"-Tester nicht existieren kann
  Beweismethode: Diagonalisierung

Satz

es kann kein Testprogramm für "hello, world"-Programme geben

Beweisskizze.

- wir betrachten ein hypothetisches Programm H, das wir "hello, world"-Tester nennen



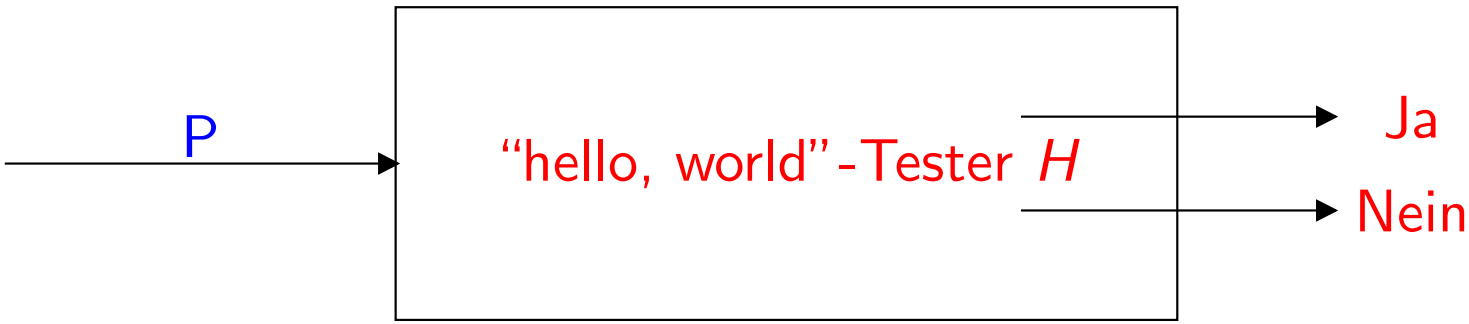
- man kann zeigen, dass ein "hello, world"-Tester nicht existieren kann
  Beweismethode: Diagonalisierung
- es ist algorithmisch nicht feststellbar, ob ein Programm P ein "hello, world"-Programm ist

Satz

es kann kein Testprogramm für "hello, world"-Programme geben

Beweisskizze.

- wir betrachten ein hypothetisches Programm H, das wir "hello, world"-Tester nennen



- man kann zeigen, dass ein "hello, world"-Tester nicht existieren kann
  Beweismethode: Diagonalisierung
- es ist algorithmisch nicht feststellbar, ob ein Programm P ein "hello, world"-Programm ist

Definition (informell)

ein Problem, das nicht algorithmisch lösbar ist, wird unentscheidbar genannt

Definition (informell)

ein Problem, das nicht algorithmisch lösbar ist, wird unentscheidbar genannt

Definition

als Halteproblem bezeichnen wir das Problem, ob ein beliebiges Programm auf seiner Eingabe hält

Definition (informell)

ein Problem, das nicht algorithmisch lösbar ist, wird unentscheidbar genannt

Definition

als Halteproblem bezeichnen wir das Problem, ob ein beliebiges Programm auf seiner Eingabe hält

Definition

Postsches Korrespondenzproblem: Gegeben zwei Listen von Strings der gleichen Länge $w_1, w_2, \ldots, w_n$ und $x_1, x_2, \ldots, x_n$. Gesucht sind Indizes $i_1, i_2, \ldots, i_m$, sodass

$$w_{i_1} w_{i_2} \ldots w_{i_m} = x_{i_1} x_{i_2} \ldots x_{i_m}$$

Satz

die folgenden Probleme sind unentscheidbar:

Satz

die folgenden Probleme sind unentscheidbar:

1. *das Halteproblem*

Satz

die folgenden Probleme sind unentscheidbar:

1. *das Halteproblem*
2. *das Postsche Korrespondenzproblem*

Satz

die folgenden Probleme sind unentscheidbar:

1. *das Halteproblem*
2. *das Postsche Korrespondenzproblem*
3. *ist eine beliebige Sprache regulär?*

Satz

die folgenden Probleme sind unentscheidbar:

1. *das Halteproblem*
2. *das Postsche Korrespondenzproblem*
3. *ist eine beliebige Sprache regulär?*
4. *. . .*

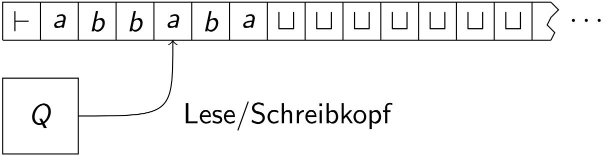
Satz

die folgenden Probleme sind unentscheidbar:

1. *das Halteproblem*
2. *das Postsche Korrespondenzproblem*
3. *ist eine beliebige Sprache regulär?*

Definition (informell)

deterministische, einbändige Turingmaschine:



Definition (formal)

eine deterministische, einbändige Turingmaschine M ist ein 9-Tupel

$$M = (Q, \Sigma, \Gamma, \vdash, \sqcup, \delta, s, t, r)$$

sodass

Definition (formal)

eine deterministische, einbändige Turingmaschine M ist ein 9-Tupel

$$M = (Q, \Sigma, \Gamma, \vdash, \sqcup, \delta, s, t, r)$$

sodass

1. Q eine endliche Menge von Zuständen,

Definition (formal)

eine deterministische, einbändige Turingmaschine M ist ein 9-Tupel

$$M = (Q, \Sigma, \Gamma, \vdash, \sqcup, \delta, s, t, r)$$

sodass

1. Q eine endliche Menge von Zuständen,
2. Σ eine endliche Menge von Eingabesymbolen,

Definition (formal)

eine deterministische, einbändige Turingmaschine M ist ein 9-Tupel

$$M = (Q, \Sigma, \Gamma, \vdash, \sqcup, \delta, s, t, r)$$

sodass

1. Q eine endliche Menge von Zuständen,
2. Σ eine endliche Menge von Eingabesymbolen,
3. Γ eine endliche Menge von Bandsymbolen, mit $\Sigma \subseteq \Gamma$,

Definition (formal)

eine deterministische, einbändige Turingmaschine M ist ein 9-Tupel

$$M = (Q, \Sigma, \Gamma, \vdash, \sqcup, \delta, s, t, r)$$

sodass

1. Q eine endliche Menge von Zuständen,
2. Σ eine endliche Menge von Eingabesymbolen,
3. Γ eine endliche Menge von Bandsymbolen, mit $\Sigma \subseteq \Gamma$,
4. $\vdash \in \Gamma \setminus \Sigma$, der linke Endmarker,

Definition (formal)

eine deterministische, einbändige Turingmaschine M ist ein 9-Tupel

$$M = (Q, \Sigma, \Gamma, \vdash, \sqcup, \delta, s, t, r)$$

sodass

1. Q eine endliche Menge von Zuständen,
2. Σ eine endliche Menge von Eingabesymbolen,
3. Γ eine endliche Menge von Bandsymbolen, mit $\Sigma \subseteq \Gamma$,
4. $\vdash \in \Gamma \setminus \Sigma$, der linke Endmarker,
5. $\sqcup \in \Gamma \setminus \Sigma$, das Blanksymbol,

Definition (formal)

eine deterministische, einbändige Turingmaschine M ist ein 9-Tupel

$$M = (Q, \Sigma, \Gamma, \vdash, \sqcup, \delta, s, t, r)$$

sodass

1. Q eine endliche Menge von Zuständen,
2. Σ eine endliche Menge von Eingabesymbolen,
3. Γ eine endliche Menge von Bandsymbolen, mit $\Sigma \subseteq \Gamma$,
4. $\vdash \in \Gamma \setminus \Sigma$, der linke Endmarker,
5. $\sqcup \in \Gamma \setminus \Sigma$, das Blanksymbol,
6. $\delta \colon Q \times \Gamma \to Q \times \Gamma \times \{\mathrm{L}, \mathrm{R}\}$ die Übergangsfunktion,

Definition (formal)

eine deterministische, einbändige Turingmaschine M ist ein 9-Tupel

$$M = (Q, \Sigma, \Gamma, \vdash, \sqcup, \delta, s, t, r)$$

sodass

1. Q eine endliche Menge von Zuständen,
2. Σ eine endliche Menge von Eingabesymbolen,
3. Γ eine endliche Menge von Bandsymbolen, mit $\Sigma \subseteq \Gamma$,
4. $\vdash \in \Gamma \setminus \Sigma$, der linke Endmarker,
5. $\sqcup \in \Gamma \setminus \Sigma$, das Blanksymbol,
6. $\delta\colon Q \times \Gamma \to Q \times \Gamma \times \{\mathrm{L}, \mathrm{R}\}$ die Übergangsfunktion,
7. $s \in Q$, der Startzustand,

Definition (formal)

eine deterministische, einbändige Turingmaschine M ist ein 9-Tupel

$$M = (Q, \Sigma, \Gamma, \vdash, \sqcup, \delta, s, t, r)$$

sodass

1. Q eine endliche Menge von Zuständen,
2. Σ eine endliche Menge von Eingabesymbolen,
3. Γ eine endliche Menge von Bandsymbolen, mit $\Sigma \subseteq \Gamma$,
4. $\vdash \in \Gamma \setminus \Sigma$, der linke Endmarker,
5. $\sqcup \in \Gamma \setminus \Sigma$, das Blanksymbol,
6. $\delta\colon Q \times \Gamma \to Q \times \Gamma \times \{\mathrm{L}, \mathrm{R}\}$ die Übergangsfunktion,
7. $s \in Q$, der Startzustand,
8. $t \in Q$, der akzeptierende Zustand und

Definition (formal)

eine deterministische, einbändige Turingmaschine M ist ein 9-Tupel

$$M = (Q, \Sigma, \Gamma, \vdash, \sqcup, \delta, s, t, r)$$

sodass

1. Q eine endliche Menge von Zuständen,
2. Σ eine endliche Menge von Eingabesymbolen,
3. Γ eine endliche Menge von Bandsymbolen, mit $\Sigma \subseteq \Gamma$,
4. $\vdash \in \Gamma \setminus \Sigma$, der linke Endmarker,
5. $\sqcup \in \Gamma \setminus \Sigma$, das Blanksymbol,
6. $\delta\colon Q \times \Gamma \to Q \times \Gamma \times \{\mathrm{L}, \mathrm{R}\}$ die Übergangsfunktion,
7. $s \in Q$, der Startzustand,
8. $t \in Q$, der akzeptierende Zustand und
9. $r \in Q$, der verwerfende Zustand mit $t \neq r$.

Beispiel: Binärer Nachfolger

Beispiel

betrachte

$$M = (\{s, t, r, p\}, \{0, 1\}, \{\vdash, \sqcup, 0, 1\}, \vdash, \sqcup, \delta, s, t, r)$$

mit δ:

$p \in Q$	$a \in \Gamma$	$\delta(p, a)$
s	0	$(s, 0, \mathrm{R})$
s	1	$(s, 1, \mathrm{R})$
s	$\sqcup$	$(p, \sqcup, \mathrm{L})$
s	$\vdash$	$(s, \vdash, \mathrm{R})$
p	0	$(t, 1, \mathrm{L})$
p	1	$(p, 0, \mathrm{L})$
p	$\vdash$	$(t, \vdash, \mathrm{R})$

Zusatzbedingungen

- $\forall\ p \in Q, \exists\ q \in Q$ sodass:

$$\delta(p, \vdash) = (q, \vdash, \mathrm{R})$$

Zusatzbedingungen

- $\forall\ p \in Q$, $\exists\ q \in Q$ sodass:
$$\delta(p, \vdash) = (q, \vdash, \mathrm{R})$$
- $\forall\ b \in \Gamma$, $\exists\ c, c' \in \Gamma$ und $d, d' \in \{\mathrm{L}, \mathrm{R}\}$:
$$\delta(t, b) = (t, c, d)$$
$$\delta(r, b) = (r, c', d')$$

Zusatzbedingungen

- $\forall\ p \in Q, \exists\ q \in Q$ sodass:
$$\delta(p, \vdash) = (q, \vdash, \mathrm{R})$$
- $\forall\ b \in \Gamma, \exists\ c, c' \in \Gamma$ und $d, d' \in \{\mathrm{L}, \mathrm{R}\}$:
$$\delta(t, b) = (t, c, d)$$
$$\delta(r, b) = (r, c', d')$$

Beispiel (Fortsetzung)

$p \in Q$	$a \in \Gamma$	$\delta(p, a)$
t	$*$	$*$
r	$*$	$*$

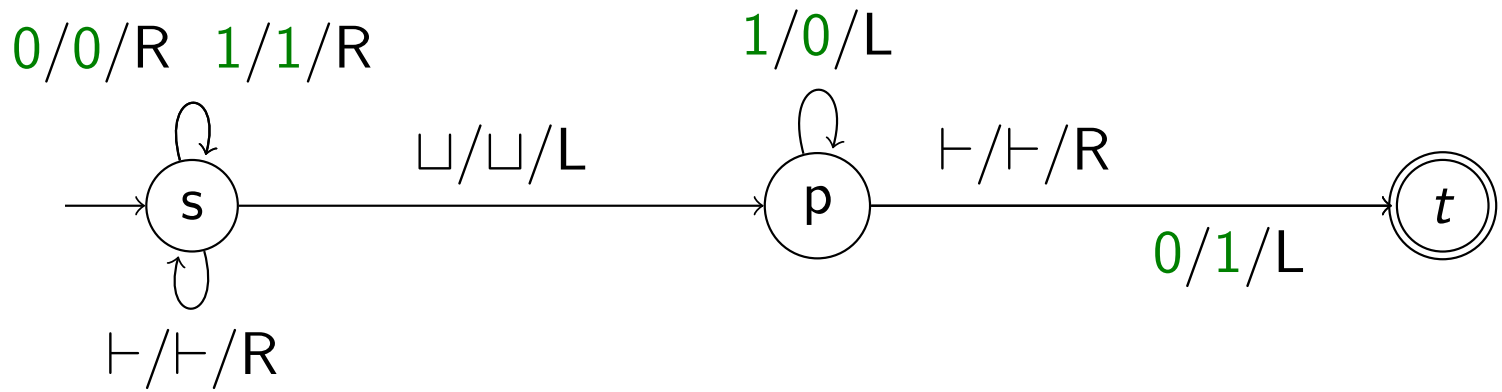
sodass Zusatzbedingungen erfüllt

Beispiel (Fortsetzung)

betrachte

$$M = (\{s, t, r, p\}, \{0, 1\}, \{\vdash, \sqcup, 0, 1\}, \vdash, \sqcup, \delta, s, t, r)$$

mit δ:

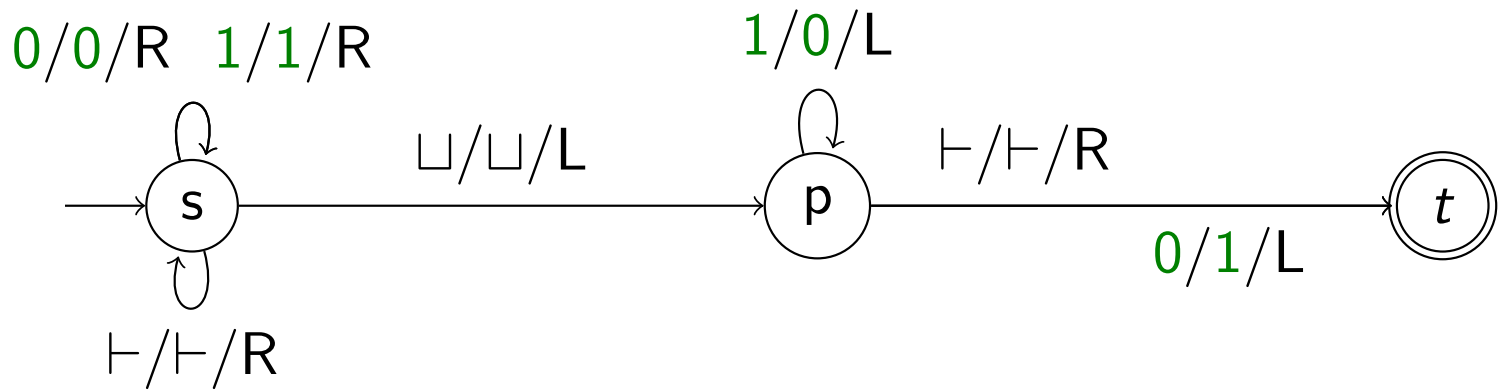


Beispiel (Fortsetzung)

betrachte

$$M = (\{s, t, r, p\}, \{0, 1\}, \{\vdash, \sqcup, 0, 1\}, \vdash, \sqcup, \delta, s, t, r)$$

mit δ:



Church-Turing-These

jedes algorithmisch lösbare Problem ist mit einer Turingmaschine lösbar