Überlegung:

- Sind die verschärften Ungleichungen erfüllbar, dann auch das ursprüngliche System. Die Umkehrung gilt i.A. nicht :-(
- In dem Fall ist bei einem Paar Ungleichungen weniger Platz:

$$a \cdot \beta \leq b \cdot \alpha + \boxed{a \cdot b}$$

oder:

$$b \cdot \alpha < ab \cdot x < b \cdot \alpha + \boxed{a \cdot b}$$

Kürzen durch b liefert:

$$\alpha < a \cdot x < \alpha + \boxed{a}$$

$\Longrightarrow$ $\boxed{\alpha + i = a \cdot x}$ für ein $i \in \{1, \ldots, a-1\}$!!!

Diskussion:

→ Fourier-Motzkin-Elimination ist nicht das beste Verfahren für rationale Ungleichungssysteme.

→ Der Omega-Test ist notwendig exponentiell :-)

Wenn das System lösbar ist, terminiert der Test i.a. schnell.

Mit unlösbaren Systemen tut er sich schwerer :-(

→ Auch für ILP gibt es andere/intelligentere Verfahren ...

→ Für Probleme bei Programmiersprachen funktioniert er wohl ganz gut :-)

4. Verallgemeinerung zu einer Logik

Disjunktion:

$$
\begin{array}{l}
(x - 2y = 15 \quad \wedge \quad x + y = 7) \qquad \vee \\
(x + y = 6 \qquad \wedge \quad 3x + z = -8)
\end{array}
$$

Quantoren:

$$\exists\, x : \quad z - 2x = 42 \quad \wedge \quad z + x = 19$$

4. Verallgemeinerung zu einer Logik

Disjunktion:

$$(x - 2y = 15 \quad \wedge \quad x + y = 7) \qquad \vee$$
$$(x + y = 6 \qquad \wedge \quad 3x + z = -8)$$

Quantoren:

$$\exists\, x : \quad z - 2x = 42 \quad \wedge \quad z + x = 19$$

$\Longrightarrow$ Presburger Arithmetik

Mojzesz Presburger, 1904–1943 (?)

Presburger Arithmetik = normale Arithmetik ohne Multiplikation

Presburger Arithmetik $\quad = \quad$ normale Arithmetik
ohne Multiplikation

Arithmetik $\quad : \quad$ hochgradig unentscheidbar :-(
sogar sogar unvollständig :-((

Presburger Arithmetik	$==$	normale Arithmetik ohne Multiplikation

Arithmetik	:	hochgradig unentscheidbar :-(sogar sogar unvollständig :-((

$\Longrightarrow$ Hilberts 10tes Problem

$\Longrightarrow$ Gödels Theorem

Presburger Formeln:

$$
\begin{array}{lll}
\phi & ::= & x + y = z \quad | \quad x = n \quad | \\
& & \phi_1 \wedge \phi_2 \quad | \quad \neg\phi \quad | \\
& & \exists\, x : \quad \phi
\end{array}
$$

Presburger Formeln:

$$
\begin{array}{lll}
\phi & ::= & x + y = z \quad | \quad x = n \quad | \\
& & \phi_1 \wedge \phi_2 \quad | \quad \neg\phi \quad | \\
& & \exists\, x : \quad \phi
\end{array}
$$

Ziel: PSAT

Finde Werte in $\mathbb{N}$ für die freien Variablen, so dass ϕ gilt ...

Idee: Codiere die Werte der Variablen als Worte :-)

213	t	1	0	1	0	1	0	1	1
42	z	0	1	0	1	0	1	0	0
89	y	1	0	0	1	1	0	1	0
17	x	1	0	0	0	1	0	0	0

Idee: Codiere die Werte der Variablen als Worte :-)

213	t	1	0	1	0	1	0	1	1
42	z	0	1	0	1	0	1	0	0
89	y	1	0	0	1	1	0	1	0
17	x	1	0	0	0	1	0	0	0

Idee: Codiere die Werte der Variablen als Worte :-)

213	t	1	0	1	0	1	0	1	1
42	z	0	1	0	1	0	1	0	0
89	y	1	0	0	1	1	0	1	0
17	x	1	0	0	0	1	0	0	0

Idee: Codiere die Werte der Variablen als Worte :-)

213	t	1	0	1	0	1	0	1	1
42	z	0	1	0	1	0	1	0	0
89	y	1	0	0	1	1	0	1	0
17	x	1	0	0	0	1	0	0	0

Idee: Codiere die Werte der Variablen als Worte :-)

213	t	1	0	1	0	1	0	1	1
42	z	0	1	0	1	0	1	0	0
89	y	1	0	0	1	1	0	1	0
17	x	1	0	0	0	1	0	0	0

Idee: Codiere die Werte der Variablen als Worte :-)

213	t	1	0	1	0	1	0	1	1
42	z	0	1	0	1	0	1	0	0
89	y	1	0	0	1	1	0	1	0
17	x	1	0	0	0	1	0	0	0

Idee: Codiere die Werte der Variablen als Worte :-)

213	t	1	0	1	0	1	0	1	1
42	z	0	1	0	1	0	1	0	0
89	y	1	0	0	1	1	0	1	0
17	x	1	0	0	0	1	0	0	0

Idee: Codiere die Werte der Variablen als Worte :-)

213	t	1	0	1	0	1	0	1	1
42	z	0	1	0	1	0	1	0	0
89	y	1	0	0	1	1	0	1	0
17	x	1	0	0	0	1	0	0	0

Idee: Codiere die Werte der Variablen als Worte :-)

213	t	1	0	1	0	1	0	1	1
42	z	0	1	0	1	0	1	0	0
89	y	1	0	0	1	1	0	1	0
17	x	1	0	0	0	1	0	0	0

Idee: Codiere die Werte der Variablen als Worte :-)

213	t	1	0	1	0	1	0	1	1
42	z	0	1	0	1	0	1	0	0
89	y	1	0	0	1	1	0	1	0
17	x	1	0	0	0	1	0	0	0

Beobachtung:

Die Menge der erfüllenden Variablenbelegungen ist regulär :-))

Beobachtung:

Die Menge der erfüllenden Variablenbelegungen ist regulär :-))

$$
\begin{array}{lcll}
\phi_1 \wedge \phi_2 & \Longrightarrow & \mathcal{L}(\phi_1) \cap \mathcal{L}(\phi_2) & \text{(Durchschnitt)} \\
\neg \phi & \Longrightarrow & \overline{\mathcal{L}(\phi)} & \text{(Komplement)} \\
\exists\, x : \quad \phi & \Longrightarrow & \pi_x(\mathcal{L}(\phi)) & \text{(Projektion)}
\end{array}
$$

Weg-Projizierung der x-Komponente:

213	t	1	0	1	0	1	0	1	1
42	z	0	1	0	1	0	1	0	0
89	y	1	0	0	1	1	0	1	0
17	x	1	0	0	0	1	0	0	0

Weg-Projizierung der x-Komponente:

213	t	1	0	1	0	1	0	1	1
42	z	0	1	0	1	0	1	0	0
89	y	1	0	0	1	1	0	1	0

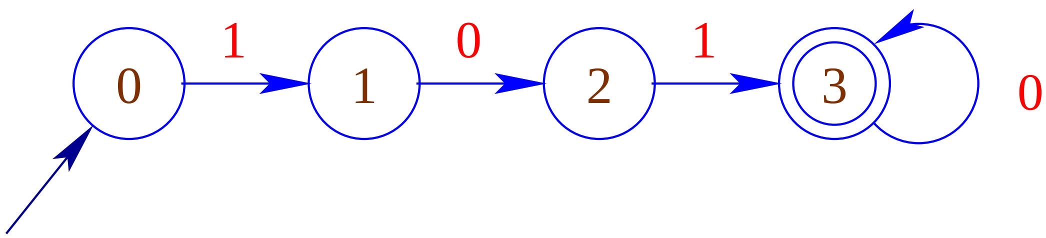
Achtung:

- unsere Zahldarstellung ist nicht eindeutig: 011101 sollte genau dann akzeptiert werden, wenn jedes Wort aus $011101 \cdot 0^*$ akzeptiert wird!
- Diese Eigenschaft bleibt bei Vereinigung, Durchschnitt und Komplement erhalten :-)
- Bei Projektion geht sie u.U. verloren !!!

$\Longrightarrow$ Der Automat für Projektion muss so angereichert werden, dass er die Eigenschaft wieder herstellt.

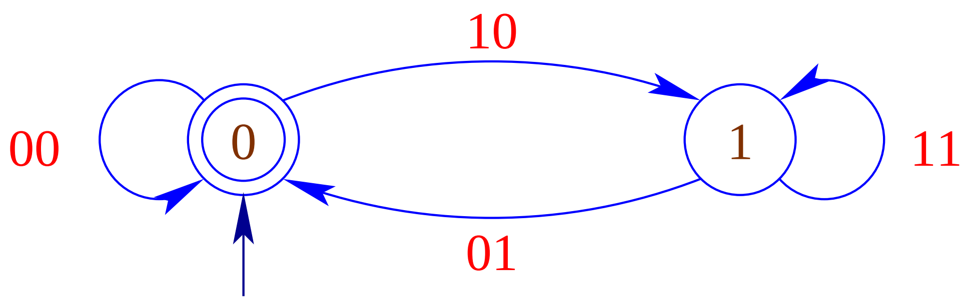
Automaten für Basis-Prädikate:

$$x = 5$$



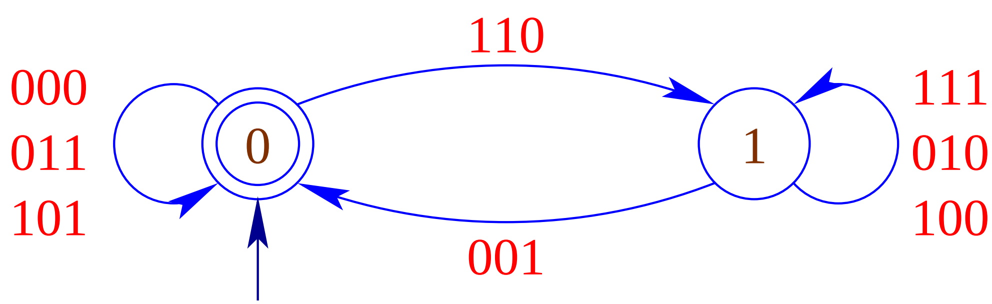
Automaten für Basis-Prädikate:

$x+x = y$



Automaten für Basis-Prädikate:

$x+y = z$



Ergebnisse:

Ferrante, Rackoff,1973 : $\mathrm{PSAT} \leq \mathrm{DSPACE}(2^{2^{c \cdot n}})$

Ergebnisse:

Ferrante, Rackoff,1973 : $\text{PSAT} \leq \text{DSPACE}(2^{2^{c \cdot n}})$

Fischer, Rabin,1974 : $\text{PSAT} \geq \text{NTIME}(2^{2^{c \cdot n}})$

3.4 Verbesserung der Speicher-Organisation

Ziel:

- Ausnutzung von Caches

  $\Longrightarrow$ Verringerung der Anzahl der Cache-Misses

- Verringerung der Allokations / Deallokations-Kosten

  $\Longrightarrow$ Ersetzung von Heap-Allokation durch Stack-Allokation

  $\Longrightarrow$ Unterstützung der Freigabe überflüssiger Heap-Objekte

- Verringerung der Zugriffskosten

  $\Longrightarrow$ Verkürzung der Indirektionsketten (Unboxing)

1. Cache-Optimierung:

Idee: lokale Speicherzugriffe

- Laden aus dem Speicher lädt nicht nur ein Byte, sondern füllt eine ganze Cache-Zeile.
- Zugriff auf benachbarte Zellen werden billiger.
- Passen alle Daten einer inneren Schleife in den Cache, wird die Iteration extrem speicher-effizient ...

Mögliche Lösungen:

$\longrightarrow$ Organisiere Zugriffe auf die vorhanden Daten um !

$\longrightarrow$ Organisiere die Daten um !

Solche Optimierungen funktionieren i.a. automatisch nur für Felder :-(

Beispiel:

$$
\begin{array}{l}
\textsf{for}\ \ (j = 1; j < n; j{+}{+}) \\
\quad\quad \textsf{for}\ \ (i = 1; i < m; i{+}{+}) \\
\quad\quad\quad\quad a[i][j] = a[i-1][j-1] + a[i][j];
\end{array}
$$

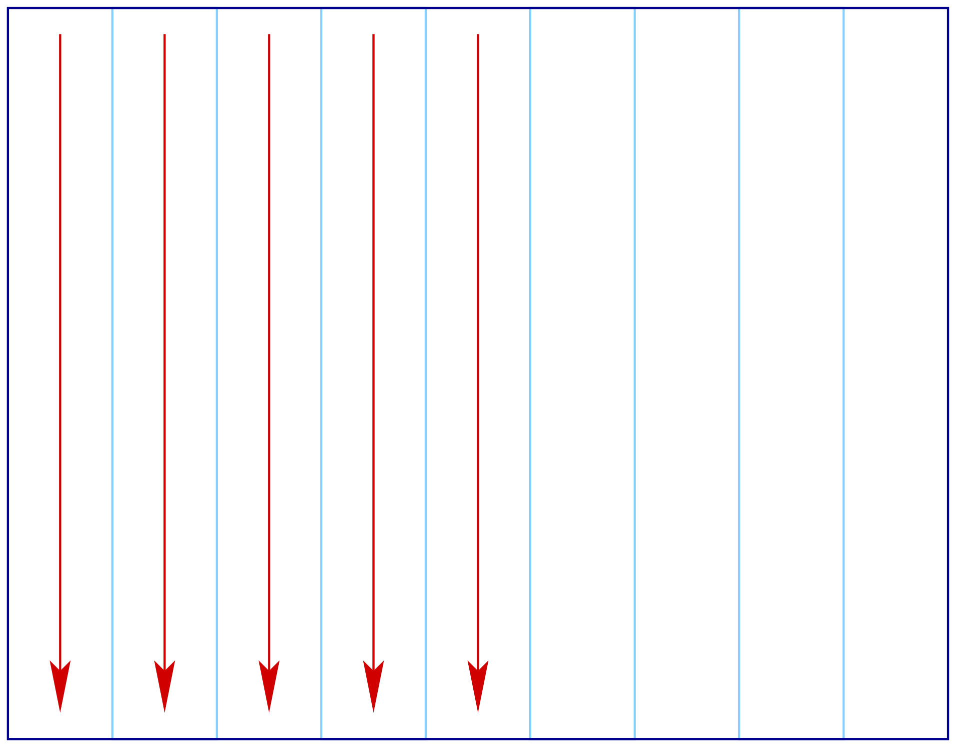
$\Longrightarrow$ Iteriere stets erst über die Zeilen!

$\Longrightarrow$ Vertausche die Reihenfolge der Iterationen:

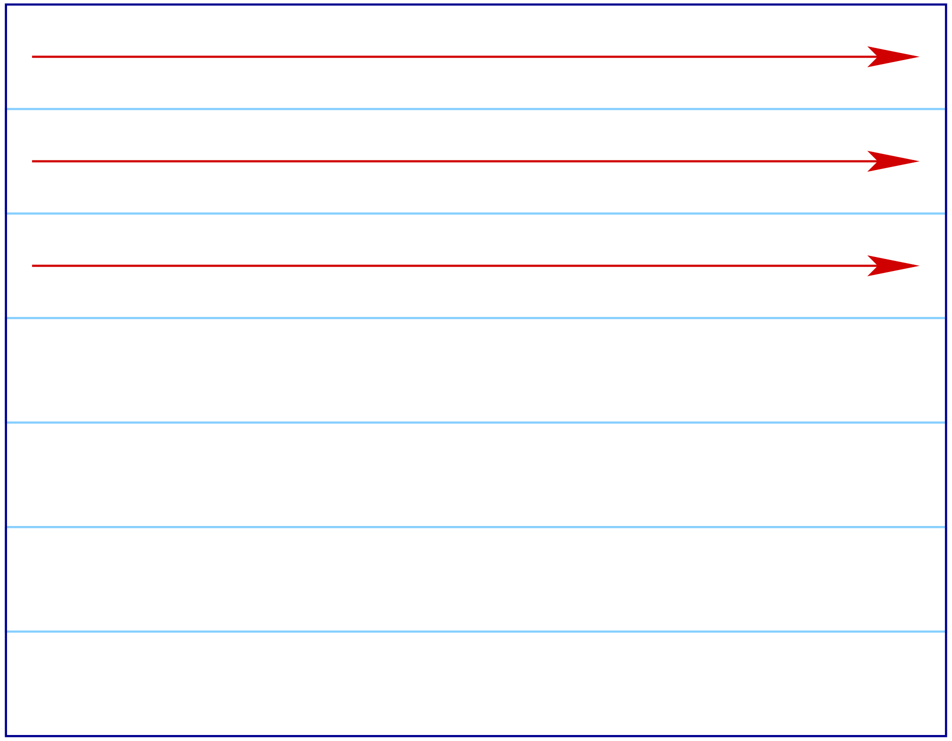
$$
\begin{array}{l}
\textsf{for}\ \ (i = 1; i < m; i{+}{+}) \\
\qquad \textsf{for}\ \ (j = 1; j < n; j{+}{+}) \\
\qquad\qquad a[i][j] = a[i-1][j-1] + a[i][j];
\end{array}
$$

Wann ist das erlaubt ???

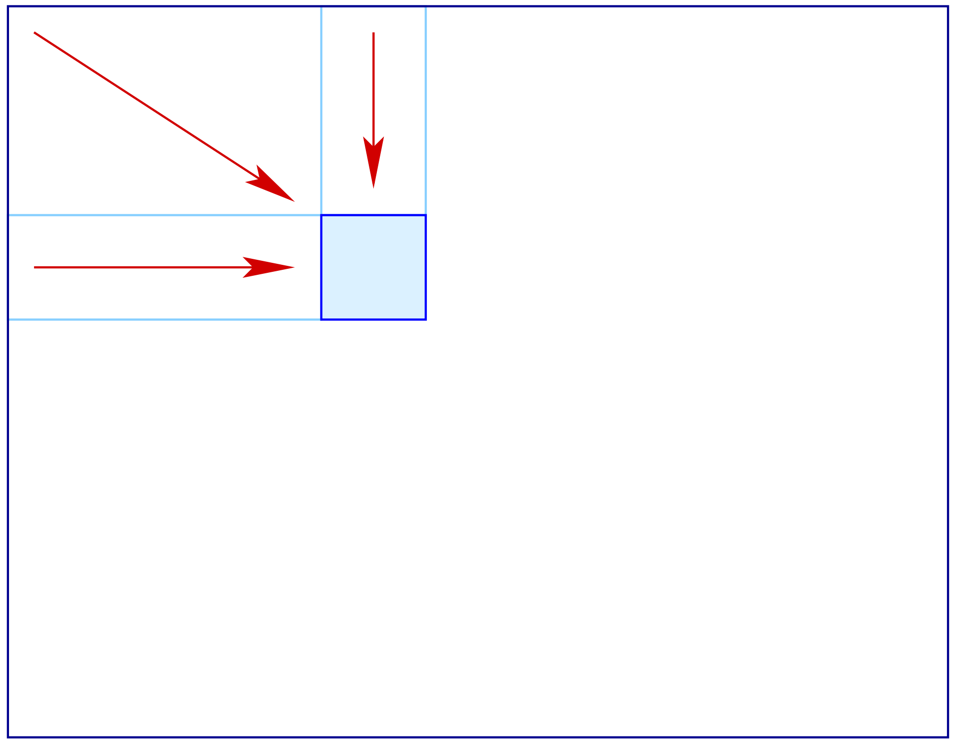
Iterations-Schema: vorher:



Iterations-Schema: nachher:



Iterations-Schema: erlaubte Abhängigkeiten:



In unserem Fall müssen wir überprüfen, dass die folgenden Gleichungs-Systeme keine Lösung haben:

Schreiben		Lesen
(i_1, j_1)	$=$	$(i_2 - 1, j_2 - 1)$
i_1	$\leq$	i_2
j_2	$\leq$	j_1
(i_1, j_1)	$=$	$(i_2 - 1, j_2 - 1)$
i_2	$\leq$	i_1
j_1	$\leq$	j_2

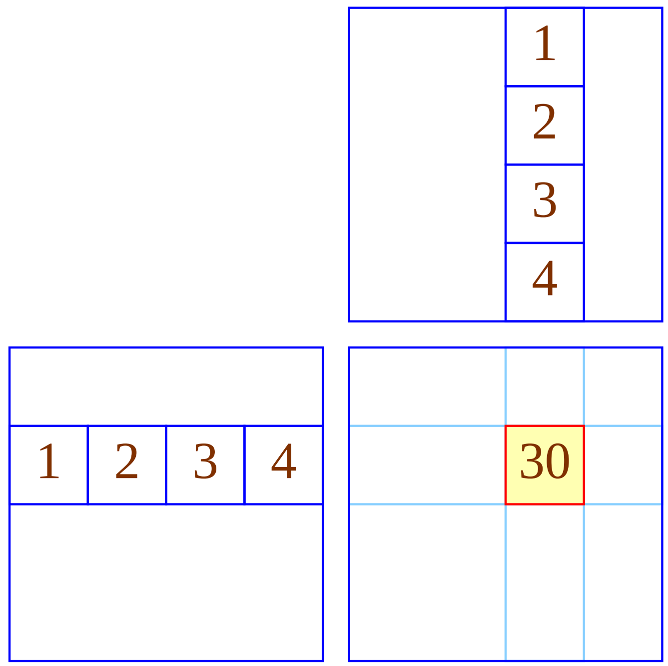
Das erste impliziert: $j_2 \leq j_2 - 1$ Hurra!

Das zweite impliziert: $i_2 \leq i_2 - 1$ Hurra!

Beispiel: Matrix-Matrix-Multiplikation

$$
\begin{array}{l}
\textsf{for}\ \ (i = 0; i < N; i{+}{+}) \\
\quad \textsf{for}\ \ (j = 0; j < M; j{+}{+}) \\
\quad\quad \textsf{for}\ \ (k = 0; k < K; k{+}{+}) \\
\quad\quad\quad c[i][j] = c[i][j] + a[i][k] \cdot b[k][j];
\end{array}
$$

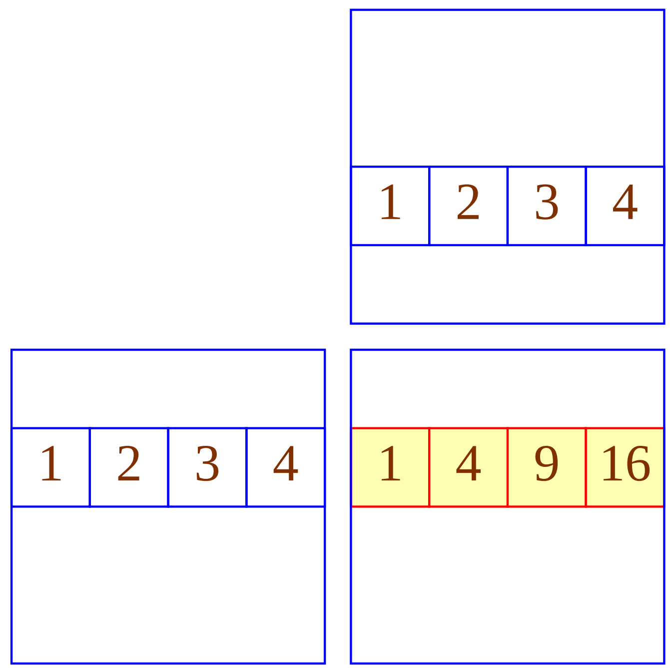
Über $b[][]$ iterieren wir spaltenweise :-(


1
2
3
4
1 2 3 4
30

Vertauschen die beiden inneren Schleifen:

$$
\begin{array}{l}
\textsf{for}\ \ (i = 0; i < N; i{+}{+}) \\
\quad \textsf{for}\ \ (k = 0; k < K; k{+}{+}) \\
\quad\quad \textsf{for}\ \ (j = 0; j < M; j{+}{+}) \\
\quad\quad\quad c[i][j] = c[i][j] + a[i][k] \cdot b[k][j];
\end{array}
$$

Ist das erlaubt ???


1 2 3 4
1 2 3 4
1 4 9 16

Diskussion:

- Die Korrektheit folgt genauso wie eben :-)
- Eine ähnliche Idee lässt sich auch zur Implementierung von Matrix-Multiplikation zeilen-komprimierter Matrizen benutzen :-))
- Möglicherweise muss das Programm erst konditioniert werden, damit die Anwendbarkeit der Transformation erkannt wird :-(
- Matrix-Multiplikation benötigt evt. erst eine Initialisierung der Ergebnis-Matrix ...