Script generated by TTT

Title: Seidl: Programmoptimierung (14.01.2013)

Date: Mon Jan 14 14:00:22 CET 2013

Duration: 93:05 min

Pages: 53

4. Generalization to a Logic

Disjunction:

$$\neg(x - 2y \neq 15 \quad \cancel{\wedge}\,\vee \quad x + y \neq 7) \quad \vee$$
$$(x + y = 6 \quad \wedge \quad 3x + z = -8)$$

Quantors:

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

4. Generalization to a Logic

Disjunction:

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

Quantors:

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

$\Longrightarrow$ Presburger Arithmetic

Mojzesz Presburger, 1904–1943 (?)

Presburger Arithmetic $=$ full arithmetic without multiplication

Presburger Arithmetic $=$ full arithmetic without multiplication

Arithmetic : highly undecidable :-(
even incomplete :-((

Presburger Arithmetic $=$ full arithmetic without multiplication

Arithmetic : highly undecidable :-(
even incomplete :-((

$\Longrightarrow$ Hilbert's 10th Problem
$\Longrightarrow$ Gödel's Theorem

Presburger Arithmetic $=$ full arithmetic without multiplication

Arithmetic : highly undecidable :-(
even incomplete :-((

$\Longrightarrow$ Hilbert's 10th Problem
$\Longrightarrow$ Gödel's Theorem

723

Presburger Formulas over $\mathbb{N}$:

$$\phi \quad ::= \quad x + y = z \quad | \quad x = n \quad | \quad \phi_1 \wedge \phi_2 \quad | \quad \neg\phi \quad | \quad \exists\, x : \; \phi$$

$3x + y = 5 \quad \leftrightarrow$

$\exists x_1, x_2, z.\; x + x = x_1 \wedge x_1 + x = x_2 \wedge x_2 + y = z \wedge z = 5)$

Presburger Formulas over $\mathbb{N}$:

$$\phi \quad ::= \quad x + y = z \quad | \quad x = n \quad | \quad \phi_1 \wedge \phi_2 \quad | \quad \neg\phi \quad | \quad \exists\, x : \; \phi$$

Goal: PSAT

Find values for the free variables in $\mathbb{N}$ such that ϕ holds ...

$\cancel{\forall}\exists x.\, x + y = 5$

Presburger Formulas over $\mathbb{N}$:

$$\phi \quad ::= \quad x + y = z \quad | \quad x = n \quad | \quad \phi_1 \wedge \phi_2 \quad | \quad \neg\phi \quad | \quad \exists\, x : \; \phi$$

Goal: PSAT

Find values for the free variables in $\mathbb{N}$ such that ϕ holds ...

Presburger Formulas over $\mathbb{N}$:

$$\phi \quad ::= \quad x + y = z \quad | \quad x = n \quad | \quad \phi_1 \wedge \phi_2 \quad | \quad \neg\phi \quad | \quad \exists\, x : \; \phi$$

Goal: PSAT

Find values for the free variables in $\mathbb{N}$ such that ϕ holds ...

Idea: Code the values of the variables as Words :-)

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

2^0 2^1 2^2 2^3 2^4 2^6

Idea: Code the values of the variables as Words :-)

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

Idea: Code the values of the variables as Words :-)

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

Idea: Code the values of the variables as Words :-)

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

Idea: Code the values of the variables as Words :-)

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

Observation:

The set of satisfying variable assignments is regular :-))

Observation:

The set of satisfying variable assignments is regular :-))

$\phi_1 \wedge \phi_2$	$\Longrightarrow$	$\mathcal{L}(\phi_1) \cap \mathcal{L}(\phi_2)$	(Intersection)
$\neg\phi$	$\Longrightarrow$	$\overline{\mathcal{L}(\phi)}$	(Complement)
$\exists\, x : \quad \phi$	$\Longrightarrow$	$\pi_x(\mathcal{L}(\phi))$	(Projection)

Projecting away the x-component:

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

$w\underline{0}^2 \in L \leftrightarrow$

$w\underline{0}^* \subseteq L$

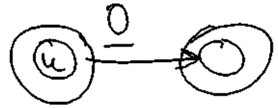
Warning:

- Our representation of numbers is not unique: 011101 should be accepted iff every word from $011101 \cdot 0^*$ is accepted!
- This property is preserved by union, intersection and complement :-)
- It is lost by projection !!!

$\Longrightarrow$ The automaton for projection must be enriched such that the property is re-established !!

Projecting away the x-component:

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



Warning:

- Our representation of numbers is not unique: 011101 should be accepted iff every word from $011101 \cdot 0^*$ is accepted!
- This property is preserved by union, intersection and complement :-)
- It is lost by projection !!!

$\Longrightarrow$ The automaton for projection must be enriched such that the property is re-established !!

$x_1 + x_2 = x_3$

$x_1 = 5$

Warning:

- Our representation of numbers is not unique: 011101 should be accepted iff every word from $011101 \cdot 0^*$ is accepted!
- This property is preserved by union, intersection and complement :-)
- It is lost by projection !!!

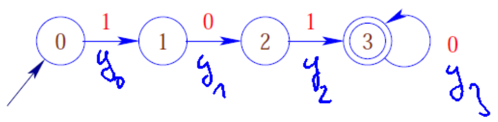
$\Longrightarrow$ The automaton for projection must be enriched such that the property is re-established !!

Automata for Basic Predicates:

x_1

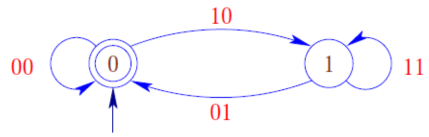
y

$x = 5$



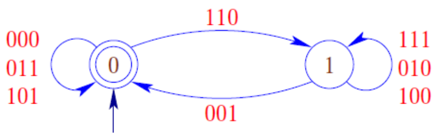
Automata for Basic Predicates:

$x+x = y$



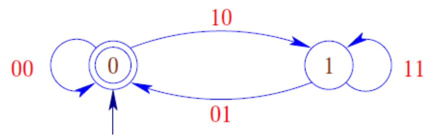
Automata for Basic Predicates:

$x+y = z$



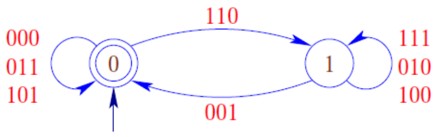
Automata for Basic Predicates:

$x+x = y$



Automata for Basic Predicates:

$x{+}y = z$



Results:

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

Results:

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

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

Results:

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

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

3.3 Improving the Memory Layout

Goal:

- Better utilization of caches

  $\Longrightarrow$ reduction of the number of cache misses

- Reduction of allocation/de-allocation costs

  $\Longrightarrow$ replacing heap allocation by stack allocation

  $\Longrightarrow$ support to free superfluous heap objects

- Reduction of access costs

  $\Longrightarrow$ short-circuiting indirection chains (Unboxing)

1. Cache Optimization:

Idea: local memory access

- Loading from memory fetches not just one byte but fills a complete cache line.
- Access to neighbored cells become cheaper.
- If all data of an inner loop fits into the cache, the iteration becomes maximally memory-efficient ...

Possible Solutions:

$\rightarrow$ Reorganize the data accesses !

$\rightarrow$ Reorganize the data !

Such optimizations can be made fully automatic only for arrays :-(

Example:

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

a_{11}

Possible Solutions:

a_{1n-1}

$\rightarrow$ Reorganize the data accesses ! a_{i1}

$\rightarrow$ Reorganize the data !

$a_{i,n-1}$

Such optimizations can be made fully automatic only for arrays :-(

Example:

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

$\Longrightarrow$ At first, always iterate over the rows!

$\Longrightarrow$ Exchange the ordering of the iterations:

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

When is this permitted???

a_{11}

a_{12}

a_{13}

a_{14}

$\Longrightarrow$ At first, always iterate over the rows!

$\Longrightarrow$ Exchange the ordering of the iterations:

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

When is this permitted???

Possible Solutions:

$\rightarrow$ Reorganize the data accesses !

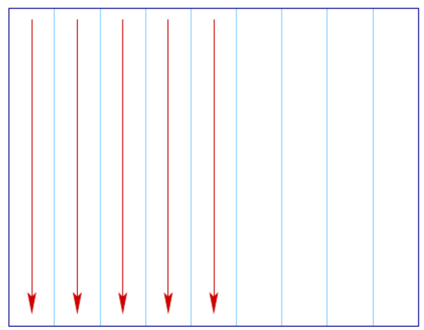
$\rightarrow$ Reorganize the data !

Such optimizations can be made fully automatic only for arrays :-(

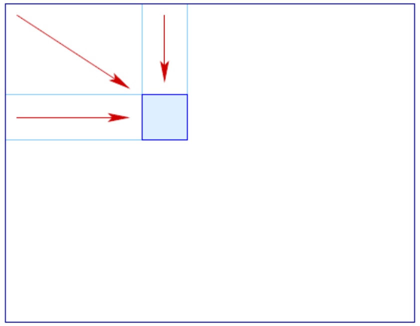
Example:

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

Iteration Scheme: before:



Iteration Scheme: allowed dependencies:



In our case, we must check that the following equation systems have no solution:

Write		Read
(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

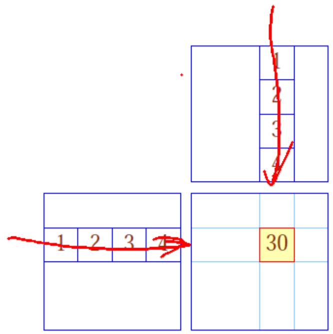
The first implies: $j_2 \leq j_2 - 1$ Hurra!

The second implies: $i_2 \leq i_2 - 1$ Hurra!

Example: Matrix-Matrix Multiplication

```
for (i = 0; i < N; i++)
    for (j = 0; j < M; j++)
        for (k = 0; k < K; k++)
            c[i][j] = c[i][j] + a[i][k] · b[k][j];
```

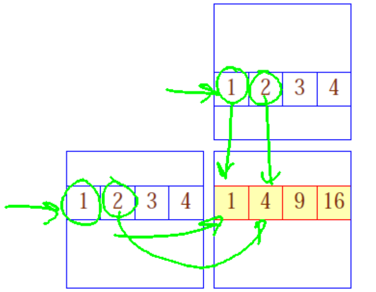
Over $b[][]$ the iteration is columnwise :-(

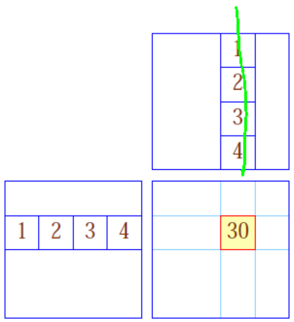


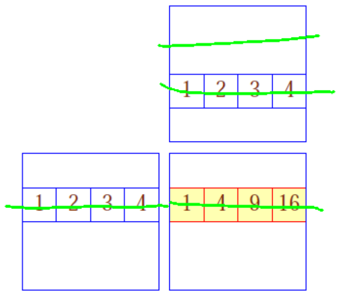
Exchange the two inner loops:

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

Is this permitted ???



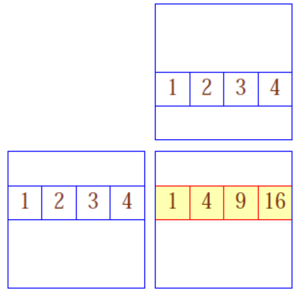




Exchange the two inner loops:

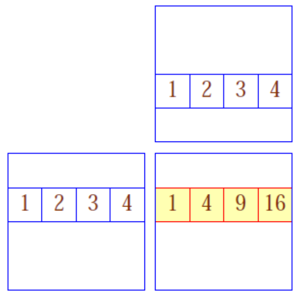
$$
\begin{array}{l}
\textsf{for}\ \ (i = 0; i < N; i{+}{+}) \\
\quad \textsf{for}\ \ (k = 0; k < K; k{+}{+}) \\
\quad\quad \textsf{for}\ \ (j = 0; j < M; j{+}{+}) \\
\quad\quad\quad c[i][j] = c[i][j] + a[i][k] \cdot b[k][j];
\end{array}
$$

Is this permitted ???



Discussion:

- Correctness follows as before :-)
- A similar idea can also be used for the implementation of multiplication for row compressed matrices :-))
- Sometimes, the program must be massaged such that the transformation becomes applicable :-(
- Matrix-matrix multiplication perhaps requires initialization of the result matrix first ...



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

- Now, the two iterations can no longer be exchanged :-(
- The iteration over j, however, can be duplicated ...

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

Correctness:

$\Longrightarrow$ The read entries (here: no) may not be modified in the remaining body of the loop !!!

$\Longrightarrow$ The ordering of the write accesses to a memory cell may not be changed :-)