W.l.o.g., we only consider strict in-equations:

\[ 6 \cdot x_1 \leq 18 + 2x_2 \]
\[ 9 - x_2 \leq 4 \cdot x_1 \]

... where we always divide by gcds:

\[ 3 \cdot x_1 < 9 + x_2 \]
\[ 8 - x_2 < 4 \cdot x_1 \]

This implies:

\[ 3 \cdot (8 - x_2) < 4 \cdot (9 + x_2) \]
We thereby obtain:

- If one derived in-equation is unsatisfiable, then also the overall system.
- If all derived in-equations are satisfiable, then there is a solution which, however, need not be integer.
- An integer solution is guaranteed to exist if there is sufficient separation between lower and upper bound ...

Assume \( \alpha < a \cdot x \quad b \cdot x < \beta \).

Then it should hold that:

\[ b \cdot \alpha < a \cdot \beta \]

and moreover:

\[ a \cdot b < a \cdot \beta - b \cdot \alpha \]

W.l.o.g., we only consider strict in-equations:

\[
6 \cdot x_1 < 18 + 2x_2 \\
8 - x_2 < 4 \cdot x_1
\]

... where we always divide by gcdds:

\[
3 \cdot x_1 < 9 + x_2 \\
8 - x_2 < 4 \cdot x_1
\]

This implies:

\[
3 \cdot (8 - x_2) < 4 \cdot (9 + x_2)
\]

... in the Example:

\[
12 < 4 \cdot (9 + x_2) - 3 \cdot (8 - x_2)
\]

or:

\[
12 < 12 + 7x_2
\]

or:

\[
0 < x_2
\]

In the example, also these strengthened in-equations are satisfiable

\[ \Rightarrow \quad \text{the system has a solution over } \mathbb{Z}. \]
Discussion

- If the strengthened in-equations are satisfiable, then also the original system. The reverse implication may be wrong!
- In the case where upper and lower bound are not sufficiently separated, we have:
\[
a \cdot \beta \leq b \cdot \alpha + [a \cdot b]
\]
or:
\[
x \cdot \alpha < 0 \rightarrow x < b \cdot \alpha + [a \cdot b]
\]
Division with \( b \) yields:
\[
\alpha < a \cdot x < \alpha + [a]
\]
\[\implies \alpha + i = a \cdot x \quad \text{for some} \quad i \in \{1, \ldots, a - 1\} \]

4. Generalization to a Logic

Disjunction:
\[
(x - 2y = 15 \land x + y = 7) \lor (x + y = 6 \land 3x + z = -8)
\]

Quantors:
\[
\exists x: z - 2x = 42 \land z + x = 19
\]

Discussion (cont.)

- Fourier-Motzkin Elimination is not the best method for rational systems of in-equations.
- The Omega test is necessarily exponential.
  If the system is solvable, the test generally terminates rapidly.
  It may have problems with unsolvable systems.
- Also for ILP, there are other/smarter algorithms ...
- For programming language problems, however, it seems to behave quite well.

4. Generalization to a Logic

Disjunction:
\[
(x - 2y = 15 \land x + y = 7) \lor (x + y = 6 \land 3x + z = -8)
\]

Quantors:
\[
\exists x: z - 2x = 42 \land z + x = 19
\]
\[\implies \quad \text{Presburger Arithmetic}\]
Presburger Arithmetic = full arithmetic without multiplication

Arithmetic : highly undecidable even incomplete

\[ \phi ::= x + y = z \mid x = n \mid \phi_1 \land \phi_2 \mid \neg \phi \mid \exists x : \phi \]

\[ \exists \cdot \cdot \cdot \bar{z} . \ y + \bar{z} = x \]

\[ \Rightarrow \exists \cdot \cdot \cdot \bar{z} . \ y + \bar{z} = x \]
Idea: Code the values of the variables as Words

<table>
<thead>
<tr>
<th></th>
<th>t</th>
<th>z</th>
<th>y</th>
<th>x</th>
</tr>
</thead>
<tbody>
<tr>
<td>213</td>
<td>1 0 1 0 1 0 1 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>42</td>
<td>0 1 0 1 0 1 0 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>89</td>
<td>1 0 0 1 1 0 1 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>1 0 0 0 1 0 0 0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Presburger Formulas over $\mathbb{N}$:

\[
\phi \ ::= \ x + y = z \mid x = n \mid \\
\phi_1 \land \phi_2 \mid \neg \phi \mid \\
\exists x : \phi
\]

Goal: PSAT

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

Idea: Code the values of the variables as Words

<table>
<thead>
<tr>
<th></th>
<th>t</th>
<th>z</th>
<th>y</th>
<th>x</th>
</tr>
</thead>
<tbody>
<tr>
<td>213</td>
<td>1 0 1 0 1 0 1 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>42</td>
<td>0 1 0 1 0 1 0 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>89</td>
<td>1 0 0 1 1 0 1 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>1 0 0 0 1 0 0 0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Observation

The set of satisfying variable assignments is regular!
Observation

The set of satisfying variable assignments is regular!

\[\phi_1 \land \phi_2 \implies \mathcal{L}(\phi_1) \cap \mathcal{L}(\phi_2)\]  
\[\neg \phi \implies \overline{\mathcal{L}(\phi)}\]  
\[\exists x : \phi \implies \pi_x(\mathcal{L}(\phi))\]

Projecting away the \( x \)-component:

<table>
<thead>
<tr>
<th>213</th>
<th>t</th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>42</td>
<td>z</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>89</td>
<td>y</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Caveat

- 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 !!

\[\implies \text{The automaton for projection must be enriched such that the property is re-established}!!\]
Automata for Basic Predicates

\[ x + y = z \]

\[ \exists x \exists y \exists z \quad x + x = x \land x + y = y \land x = 5 \]
3.3 Improving the Memory Layout

Goal

- Better utilization of caches
  - reduction of the number of cache misses
- Reduction of allocation/de-allocation costs
  - replacing heap allocation by stack allocation
  - support to free superfluous heap objects
- Reduction of access costs
  - short-circuiting indirection chains (Unboxing)

Results

Ferrante, Rackoff, 1973 : \( \text{PSAT} \leq \text{DSpace}(2^{2^m}) \)

Fischer, Rabin, 1974 : \( \text{PSAT} \geq \text{NTime}(2^{2^m}) \)
1. Cache Optimization

Idea: local memory access

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

Possible Solutions

→ Reorganize the data accesses !
→ Reorganize the data !

Such optimizations can be made fully automatic only for arrays.

Example

\[
\begin{align*}
&\text{for } (j = 1; j < n; j++) \\
&\quad \text{for } (i = 1; i < m; i++) \\
&\quad \; a[i][j] = a[i - 1][j - 1] + a[i][j];
\end{align*}
\]
At first, always iterate over the rows!

Exchange the ordering of the iterations:

\[
\text{for } (i = 1; i < m; i++)
\]

\[
\text{for } (j = 1; j < n; j++)
\]

\[
a[i][j] = a[i - 1][j - 1] + a[i][j];
\]

When is this permitted??
Iteration Scheme: allowed dependencies:

\[ \begin{align*}
(i_1, j_1) &= (i_2 - 1, j_2 - 1) \\
 1_s^1 &\leq i_2 \\
 j_2 &\leq j_1 \\
(i_1, j_1) &= (i_2 - 1, j_2 - 1) \\
 i_2 &\leq i_1 \\
 j_1 &\leq j_2
\end{align*} \]

The first implies: \( j_2 \leq j_2 - 1 \) Hurra!
The second implies: \( i_2 \leq i_2 - 1 \) Hurra!

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

\[
\begin{array}{|c|c|}
\hline
\text{Write} & \text{Read} \\
\hline
(i_1, j_1) &= (i_2 - 1, j_2 - 1) \\
 1_s^1 &\leq i_2 \\
 j_2 &\leq j_1 \\
(i_1, j_1) &= (i_2 - 1, j_2 - 1) \\
 i_2 &\leq i_1 \\
 j_1 &\leq j_2 \\
\hline
\end{array}
\]

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++)
            \( c[i][j] = c[i][j] + a[i][k] \cdot b[k][j] \);

Over \( b[k] \) the iteration is columnwise.