4. Generalization to a Logic

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

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

\[\implies\] Presburger Arithmetic

Mojzesz Presburger, 1904–1943 (?)
Presburger Arithmetic = full arithmetic without multiplication

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

⇒ Hilbert’s 10th Problem
⇒ Gödel’s Theorem
Presburger Formulas over \( \mathbb{N} \):

\[
\phi ::= x + y = z \quad | \quad x = n \\
(\phi_1 \land \phi_2) \quad | \quad \neg \phi \\
\exists x : \phi
\]

\[3x + y = 5 \quad \Leftarrow \]

\[\exists x, y, z : x + x = x \land x + x = x \land 2y = 2 \land z = 5\]

Goal: PSAT

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

---

Presburger Formulas over \( \mathbb{N} \):

\[
\phi ::= x + y = z \quad | \quad x = n \\
(\phi_1 \land \phi_2) \quad | \quad \neg \phi \\
\exists x : \phi
\]

\[\forall j : x \cdot x + y = 5 \]

Goal: PSAT

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

---

Presburger Formulas over \( \mathbb{N} \):

\[
\phi ::= x + y = z \quad | \quad x = n \\
(\phi_1 \land \phi_2) \quad | \quad \neg \phi \\
\exists x : \phi
\]

Goal: PSAT

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

---

Presburger Formulas over \( \mathbb{N} \):

\[
\phi ::= x + y = z \quad | \quad x = n \\
(\phi_1 \land \phi_2) \quad | \quad \neg \phi \\
\exists x : \phi
\]

Goal: PSAT

Find values for the free variables in \( \mathbb{N} \) such that \( \phi \) holds ...
Idea: Code the values of the variables as Words :-)

\[
\begin{array}{c|cccc}
213 & t & 1 & 0 & 1 \\
42  & z & 0 & 1 & 0 \\
89  & y & 1 & 0 & 0 \\
17  & x & 1 & 0 & 0 \\
\end{array}
\]
**Idea:** Code the values of the variables as Words :)

<table>
<thead>
<tr>
<th></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>
<th>φ</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>
<td>φ</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>
<td>φ</td>
</tr>
<tr>
<td>17</td>
<td>x</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>φ</td>
</tr>
</tbody>
</table>

**Observation:**

The set of satisfying variable assignments is regular :))

\[
\begin{align*}
\phi_1 \land \phi_2 & \quad \implies \quad \mathcal{L}(\phi_1) \cap \mathcal{L}(\phi_2) \quad \text{(Intersection)} \\
\neg \phi & \quad \implies \quad \mathcal{L}(\neg \phi) \quad \text{(Complement)} \\
\exists x : \phi & \quad \implies \quad \pi_x(\mathcal{L}(\phi)) \quad \text{(Projection)}
\end{align*}
\]

**Observation:**

The set of satisfying variable assignments is regular :))

<table>
<thead>
<tr>
<th></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>
<tr>
<td>17</td>
<td>x</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Warning:

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

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

---

Projecting away the \( x \)-component:

\[
\begin{array}{c|cccccccc}
213 & t & & & & & & & 1 & 0 & 1 & 0 & 1 & 1 & 1 \\
42  & z & & & & & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\
89  & y & & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 &  &  &  &  \\
\end{array}
\]

Warning:

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

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

\[
\begin{align*}
\chi_\Lambda + \chi_x &< \chi_\Xi \\
\chi_\Lambda & = \subseteq
\end{align*}
\]
Automata for Basic Predicates:

\[ x + y = z \]

Results:

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

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

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

Fischer, Rabin.1974 : \( \text{PSAT} \geq \text{NTIME}(2^{2^n}) \)
3.3 Improving the Memory Layout

Goal:
- Better utilization of caches
  \[ \implies \] reduction of the number of cache misses
- Reduction of allocation/de-allocation costs
  \[ \implies \] replacing heap allocation by stack allocation
- Support to free superfluous heap objects
  \[ \implies \] short-circuiting indirection chains (Unboxing)

Possible Solutions:
- Reorganize the data accesses!
- Reorganize the data!

Such optimizations can be made fully automatic only for arrays.

Example:

\[
\begin{align*}
  & (j = 1; j < n; j++) \\
  & (i = 1; i < m; i++) \\
  & a[i][j] = a[i-1][j-1] + a[i][j];
\end{align*}
\]

1. Cache Optimization:

Idea: local memory access

- Loading from memory fetches not just one byte but fills a complete cache line.
- Access to neighborhood 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:

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

Exchange the ordering of the iterations:

\[
\text{for } (i = 1; i < m; i++)
\text{ for } (j = 1; j < n; j++)
\quad a[i][j] = a[i - 1][j - 1] + a[i][j];
\]

When is this permitted???

Possible Solutions:

→ Reorganize the data accesses!
→ Reorganize the data!

Such optimizations can be made fully automatic only for arrays.

Example:

\[
\text{for } (j = 1; j < n; j++)
\text{ for } (i = 1; i < m; i++)
\quad a[i][j] = a[i - 1][j - 1] + a[i][j];
\]

Iteration Scheme: before:
Iteration Scheme: allowed dependencies:

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

<table>
<thead>
<tr>
<th>Write</th>
<th>Read</th>
</tr>
</thead>
<tbody>
<tr>
<td>((i_1, j_1))</td>
<td>((i_2 - 1, j_2 - 1))</td>
</tr>
<tr>
<td>(i_2 \leq i_1)</td>
<td>(i_2 \leq i_1)</td>
</tr>
<tr>
<td>(j_2 \leq j_1)</td>
<td>(j_2 \leq j_1)</td>
</tr>
</tbody>
</table>

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

Example: Matrix-Matrix Multiplication

\[
\begin{align*}
\text{for } (i = 0; i < N; i++) \\
\quad \text{for } (j = 0; j < M; j++) \\
\quad \quad \text{for } (k = 0; k < K; k++) \\
\quad \quad \quad c[i][j] &= c[i][j] + a[i][k] \times b[k][j];
\end{align*}
\]

Over \(b[][]\) the iteration is columnwise. :-(
Exchange the two inner loops:

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

Is this permitted???
Exchange the two inner loops:

\[
\begin{align*}
\text{for } & (i = 0; i < N; i++) \\
\text{for } & (k = 0; k < K; k++) \\
\text{for } & (j = 0; j < M; j++) \\
& c[i][j] = c[i][j] + a[i][k] \cdot b[k][j];
\end{align*}
\]

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 ...
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];

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

for (i = 0; i < N; i++)
    for (j = 0; j < M; j++)
        c[i][j] = 0;
    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];

Correctness:

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

\[\rightarrow\] The ordering of the write accesses to a memory cell may not be changed :-(