Difference Logic

Satisfiability Checking Seminar

Alex Ryndin
Supervisor: Gereon Kremer

WS 2016/2017

Outline

- Main Literature
- Difference Logic
- Example Problem: Job Scheduling
- SAT Checking of Propositional Logic
- SAT Checking of Difference Logic
- Constraint Graph
- Negative Cycles in a Constraint Graph
- How to Find a Negative Cycle in a Graph
- Goldberg-Radzik Heuristic
- SAT Checking Algorithm for Difference Logic (Sketch)
- Conclusion

Main Literature

- [Cotton et al. 2004] Scott Cotton, Eugene Asarin, Oded Maler and Peter Niebert. **“Some progress in satisfiability checking for difference logic“**. In Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems, pages 263–276. Springer, 2004.
- [Goldberg+Radzik 1993] Andrew V. Goldberg and Tomasz Radzik. **“A heuristic improvement of the Bellman-Ford algorithm“**. Applied Mathematics Letters, 6(3):3–6, 1993.
- [Cormen et al. 2009] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein. **“Introduction to algorithms“**. MIT press, third edition, 2009.
  Note: the chapter 24 **“Single-Source Shortest Paths“** is relevant for the topic.

Difference Logic

- Difference logic (DL) is a special case of linear arithmetic (LA) logic.
- It is a Propositional Logic (PL) enhanced with constraints of the following form:
$$x - y \prec c \tag{1}$$
where x, y are variables, c is a constant and $\prec \in \{<, \leq\}$ is a comparison operator.
- x, y, c can be defined either over Integers $\mathbb{Z}$ or over Reals $\mathbb{R}$.

Difference Logic

- A couple of examples:

$$\phi_1 = (p \vee q) \wedge (p \rightarrow (u - v < 3.3)) \wedge (q \rightarrow (v - w < -5.15)) \quad (2)$$

$$\begin{aligned} \phi_2 = {} & (u - v < 1) \wedge (v - w < 5) \\ & \wedge (w - x \leq -3) \wedge (x - y < 1) \\ & \wedge (y - z \leq -5) \wedge (y - v \leq 0) \end{aligned} \quad (3)$$

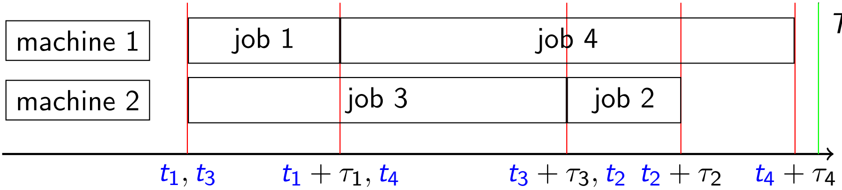
$$\begin{aligned} \phi_3 = {} & (u - v < 1) \wedge (v - w < 5) \\ & \wedge (w - x \leq -3) \wedge (x - y < -3) \\ & \wedge (y - z \leq -5) \wedge (y - w < 4) \end{aligned} \quad (4)$$

Difference Logic. Special cases

- $x < c \quad \Leftrightarrow \quad x - 0 < c \quad \Leftrightarrow \quad x - zero < c$ where $zero$ is a special variable which, however, does not change the algorithm for deciding SAT of DL formula.
- $x \geq c \quad \Leftrightarrow \quad -x \leq -c \quad \Leftrightarrow \quad 0 - x \leq -c \quad \Leftrightarrow \quad zero - x \leq -c$
- $x \neq c \quad \Leftrightarrow \quad ((x < c) \vee (x > c))$ and then see above
- $x = c \quad \Leftrightarrow \quad \neg((x < c) \vee (x > c))$ and then see above
- An example:

$$
\begin{aligned}
&(v = -3) \\
&(\neg((v < -3) \vee (v > -3))) \\
&(\neg((v < -3) \vee (-v < 3))) \\
&(\neg((v - 0 < -3) \vee (0 - v < 3))) \\
&(\neg((v - zero < -3) \vee (zero - v < 3)))
\end{aligned}
$$

Example Problem: Job Scheduling



- p_{mj} = *True* if job j is scheduled on machine m:
  e.g. $p_{11} = p_{14} = p_{23} = p_{22} =$ *True* for the figure above
- job i starts at t_i and lasts τ_i
- a machine cannot process two or more jobs simultaneously:
  $(p_{mi} \wedge p_{mj}) \rightarrow ((t_i + \tau_i \leq t_j) \vee (t_j + \tau_j \leq t_i)) \quad \Leftrightarrow$
  $(p_{mi} \wedge p_{mj}) \rightarrow ((t_i - t_j \leq -\tau_i) \vee (t_j - t_i \leq -\tau_j))$
- the overall processing time should not exceed T:
  $t_i + \tau_i \leq T \quad \Leftrightarrow \quad t_i - 0 \leq T - \tau_i$

Example Problem: Job Scheduling

$$\phi = \bigwedge_{j=1}^{4} (p_{1j} \vee p_{2j}) \quad \wedge$$

each task is executed on at least one machine

$$\bigwedge_{j=1}^{4} ((p_{1j} \rightarrow \neg p_{2j}) \wedge (p_{2j} \rightarrow \neg p_{1j})) \quad \wedge$$

each task can be scheduled on one machine only

$$\bigwedge_{j=1}^{4} (t_j \geq 0) \;\wedge\; \bigwedge_{j=1}^{4} (t_j \leq T - \tau_j) \quad \wedge$$

general time constraints

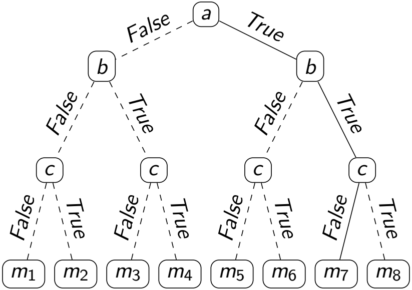
$$\bigwedge_{m=1}^{2} \bigwedge_{i=1}^{3} \bigwedge_{j=i+1}^{4} ((p_{mi} \wedge p_{mj}) \rightarrow ((t_i - t_j \leq -\tau_i) \vee (t_j - t_i \leq -\tau_j)))$$

no time overlap rule

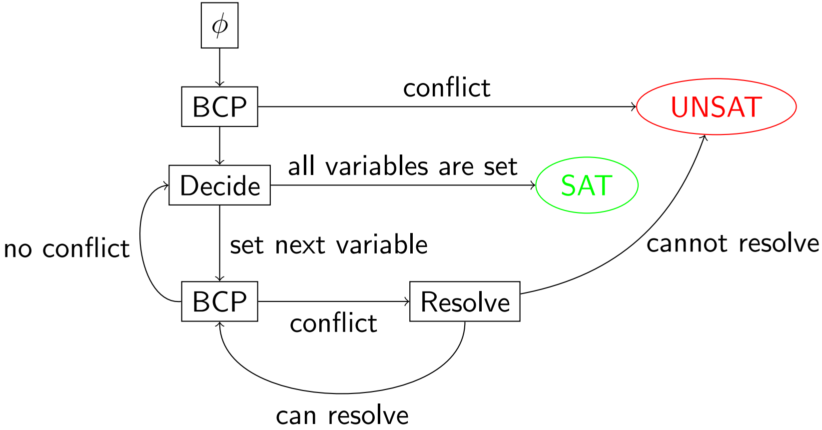
SAT Checking of Propositional Logic

$\phi = (a \vee b) \wedge (b \vee c) \wedge (c \vee a) \wedge \ldots$

SAT checking = intelligent search in the model space. The model space can be represented as a tree.



SAT Checking of Propositional Logic



SAT Checking of Difference Logic

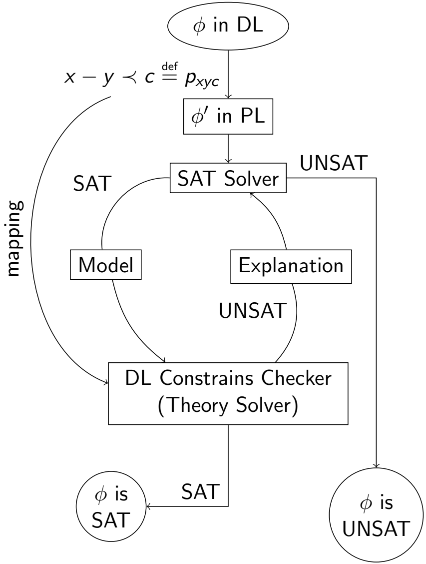


Figure: Lazy approach

SAT Checking of Difference Logic

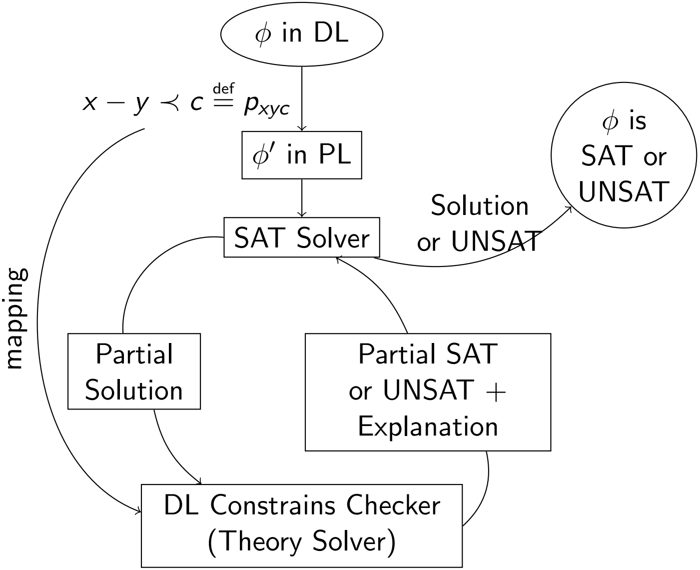
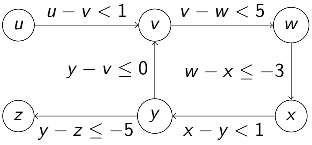
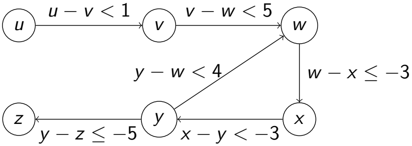


Figure: Incremental approach

Constraint Graph





$$\begin{aligned}\phi_2 = &(u - v < 1) \\ &\wedge (v - w < 5) \\ &\wedge (w - x \leq -3) \\ &\wedge (x - y < 1) \\ &\wedge (y - z \leq -5) \\ &\wedge (y - v \leq 0)\end{aligned}$$

$$\begin{aligned}\phi_3 = &(u - v < 1) \\ &\wedge (v - w < 5) \\ &\wedge (w - x \leq -3) \\ &\wedge (x - y < -3) \\ &\wedge (y - z \leq -5) \\ &\wedge (y - w < 4)\end{aligned}$$

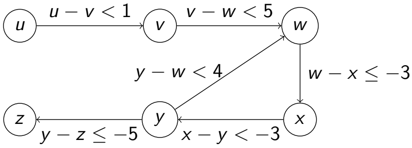
$$\Gamma = (V, E, weight, op)$$

Negative Cycles in a Constraint Graph

Negative cycle
$x \to y \to w \to x$

$$\begin{array}{r}
x - y < -3 \\
y - w < 4 \\
\underline{w - x \leq -3} \\
0 < -2
\end{array}$$

A conflict!

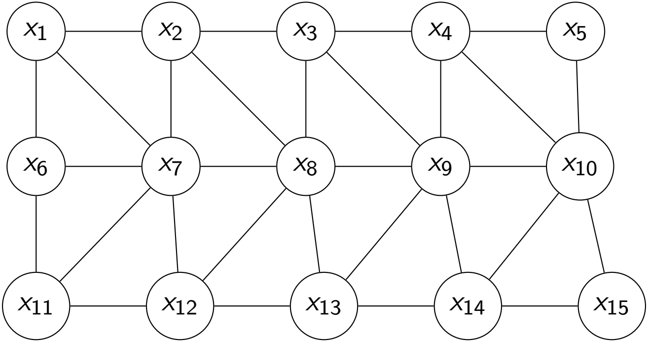


$$\begin{aligned}
\phi_3 = {} & (u - v < 1) \\
& \wedge (v - w < 5) \\
& \wedge (w - x \leq -3) \\
& \wedge (x - y < -3) \\
& \wedge (y - z \leq -5) \\
& \wedge (y - w < 4)
\end{aligned}$$

How to Find a Negative Cycle in a Graph

- First idea: enumerate all cycles
  - and check if they are negative
    ($0 \prec c$ conflict clause where $c < 0$ and $\prec \in \{<, \leq\}$)
  - or they have zero weight and an edge with a strict inequality
    ($0 < 0$ conflict clause)
- Any problems with this approach?

How to Find a Negative Cycle in a Graph



- Problem: a graph can have an enormous number of cycles
- E.g. extreme case: fully connected graph
  - Number of cycles $= \sum_{i=2}^{n} \binom{i}{n} \frac{i!}{2}$
  - $i!$ is divided over 2 because e.g. permutations $(1, 2, \ldots, i-1, i)$ and $(i, i-1, \ldots, 2, 1)$ correspond to the same cycle

How to Find a Negative Cycle in a Graph

- Use Bellman-Ford algorithm for this task [Cormen et al. 2009]
  - $O(|V| \cdot |E|)$ time complexity

BELLMAN-FORD(graph $\Gamma = (V, E, weight)$, source vertex $s \in V$)

```
for each vertex x ∈ V
do d(x) = ∞
d(s) = 0
for i = 1 to |V| − 1
do for each edge (x, y) ∈ E
      do if d(x) + weight(x, y) < d(y)
            then d(y) = d(x) + weight(x, y)
for each edge (x, y) ∈ E
do if d(x) + weight(x, y) < d(y)
         then return False
return True
```

How to Find a Negative Cycle in a Graph

- [Cotton et al. 2004] uses the admissible graph Γ_d to find a negative or zero weight cycle in the original constraint graph Γ
- Terminology:
  - Reduced cost function $r(x, y) = weight(x, y) + d(x) - d(y)$
  - Admissible edge: $r_d(x, y) \leq 0$
  - Admissible graph Γ_d – a graph consisting of admissible edges
- Implications:
  - Γ_d is dynamic because it depends on d which changes during the execution of the algorithm
  - If $r(x, y) < 0$ then the edge (x, y) can be "relaxed" i.e. used to improve $d(y)$
  - Γ_d consists of edges which might potentially be used to improve d
  - Intuition: if Γ_d has a cycle then this cycle might be used to update d infinitely

How to Find a Negative Cycle in a Graph

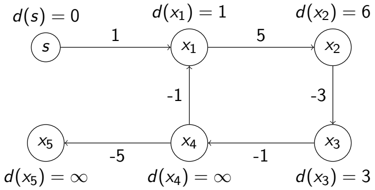


Figure: Γ

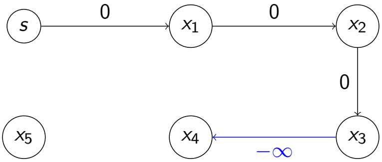


Figure: Γ_d

$$r(x, y) = weight(x, y) + d(x) - d(y)$$

$$\begin{array}{ccc} r(s, x_1) = 0 & r(x_1, x_2) = 0 & r(x_2, x_3) = 0 \\ r(x_3, x_4) = -\infty & r(x_4, x_1) = \infty & r(x_4, x_5) = \varnothing \end{array}$$

Γ_d has no cycles. Let us relax the edge (x_3, x_4)

How to Find a Negative Cycle in a Graph

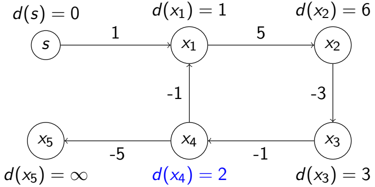


Figure: Γ

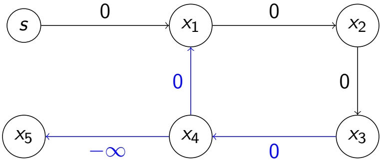


Figure: Γ_d

$$r(x,y) = weight(x,y) + d(x) - d(y)$$

$$r(s,x_1) = 0 \quad r(x_1,x_2) = 0 \quad r(x_2,x_3) = 0$$
$$r(x_3,x_4) = 0 \quad r(x_4,x_1) = 0 \quad r(x_4,x_5) = -\infty$$

Now Γ_d has a cycle: $x_1 \to x_2 \to x_3 \to x_4 \to x_1$. This cycle in Γ has indeed zero weight.

How to Find a Negative Cycle in a Graph

Theorem

Given a constraint graph Γ and a series of distance estimating functions $(d_0, d_1, d_2, d_3, \dots)$, Γ has a negative or zero cycle if and only if Γ_d has a cycle under some distance estimate d_k.

Proof.

Use "proof-by-contradiction" approach.

$\Rightarrow$ Use the following fact inferred from [Cormen et al. 2009]. When Γ has a negative cycle then the series $(d_0, d_1, d_2, d_3, \dots)$ will never converge.

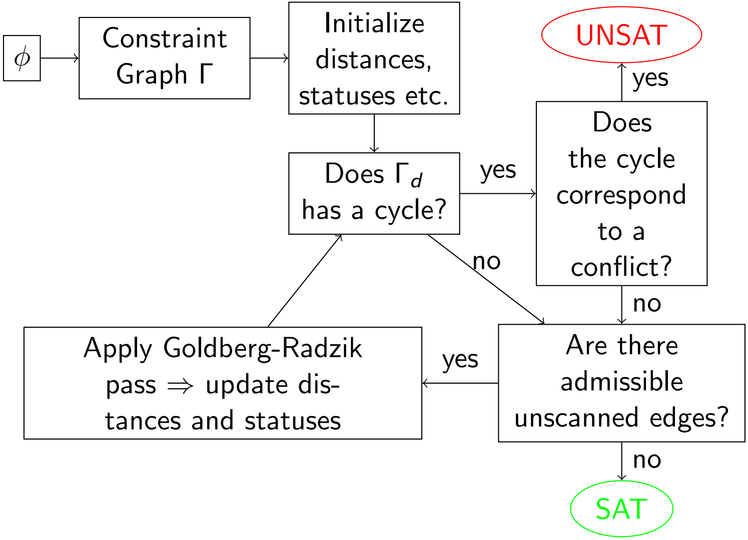
$\Leftarrow$ Cycle is in Γ_d therefore all its edges are admissible and therefore $d(x_i) + weight(x_i, x_{i+1}) \leq d(x_{i+1})$. Sum the latter inequality along all the edges of the cycle and show that the cycle's weight will be non-positive: $\sum_{i=0}^{n-1} weight(x_i, x_{i+1}) \leq 0$

For the full proof please see my seminar paper or [Cotton et al. 2004]. $\square$

Goldberg-Radzik Heuristic

- [Goldberg+Radzik 1993] suggests a heuristic to speed up Bellman-Ford algorithm in practical cases.
- The theoretical upper bound stays the same: $O(|V| \cdot |E|)$
- Idea:
  - Mark vertices as "unreached", "labeled" and "scanned" (vertex status)
  - In the beginning of each pass take vertices that have at least one outgoing admissible edge – set B
  - Also, mark those vertices that have no outgoing admissible edges as "scanned"
  - Calculate set A – unexplored vertices (i.e. "unreached") which are reachable from B in Γ_d
  - Sort A topologically using Γ_d as the input graph
  - Execute a pass:
    - For each vertex in A relax all outgoing admissible edges (of course, if they can be relaxed i.e. if $r(x, y) < 0$)
  - Execute passes until all the vertices are scanned

SAT Checking Algorithm for Difference Logic (Sketch)



Conclusion

- Many timing problems (logistics, planning, scheduling, circuits checking) can be expressed in DL. Therefore it is important to have an efficient algorithm for checking SAT of a DL formula.
- Conjunction of DL constraints can be represented by a constraint graph Γ.
  - A negative cycle corresponds to a conflict $0 \prec c$ where $c < 0$ and $\prec \in \{<, \leq\}$.
  - A zero weight cycle with a strict inequality edge corresponds to a conflict $0 < 0$.
- There is no need to enumerate all cycles in Γ. Bellman-Ford algorithm can be used to detect a negative cycle in $O(|V| \cdot |E|)$ operations.
- A cycle in admissible graph Γ_d corresponds to a negative or zero weight cycle in the corresponding constraint graph Γ.

Thank you

Thank you for your attention