



# Constraint Solving

René Thiemann      and      Fabian Mitterwallner

based on a previous course by Aart Middeldorp

# Outline

- 1. Summary of Previous Lecture**
- 2. Difference Logic**
- 3. Simplex Algorithm**
- 4. Support of Strict Inequalities**
- 5. Further Reading**

## Definition (Equality Logic)

terms are restricted to variables, no quantifiers:

$$\varphi ::= \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \varphi \rightarrow \varphi \mid \neg \varphi \mid t = t \qquad t ::= x$$

T-solver for conjunction  $\varphi$  of equality logic literals over set of variables  $V$

## Definition

**equality graph** is undirected graph  $G_=(\varphi) = (V, E_=(\varphi), E_\neq(\varphi))$  with

- $E_=(\varphi)$  edges corresponding to positive (equality) literals in  $\varphi$
- $E_\neq(\varphi)$  edges corresponding to negative (inequality) literals in  $\varphi$

## Lemma

$\varphi$  is satisfiable  $\iff$   
 $G_=(\varphi)$  contains no simple contradictory cycles (with exactly one  $E_\neq$  edge)

## Quantifier-Free Fragment of Equality Logic with Uninterpreted Functions (EUF)

$$\varphi ::= \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \varphi \rightarrow \varphi \mid \neg \varphi \mid t = t$$

$$t ::= f(t, \dots, t)$$

### T-Solver for Conjunctive Fragment via Congruence Closure

input: set  $E$  of ground equations and ground equation  $s \approx t$

output: **valid** ( $E \models_T s = t$ ) or **invalid** ( $E \not\models_T s = t$ )

1 build congruence classes

(a) put different subterms of terms in  $E \cup \{s = t\}$  in separate sets

(b) merge sets  $\{\dots, t_1, \dots\}$  and  $\{\dots, t_2, \dots\}$  for all  $t_1 = t_2$  in  $E$

(c) repeatedly merge sets

$$\{\dots, f(s_1, \dots, s_n), \dots\} \text{ and } \{\dots, f(t_1, \dots, t_n), \dots\}$$

if  $s_i$  and  $t_i$  belong to same set for all  $1 \leq i \leq n$

2 if  $s$  and  $t$  belong to same set then return **valid** else return **invalid**

# Outline

1. Summary of Previous Lecture
- 2. Difference Logic**
3. Simplex Algorithm
4. Support of Strict Inequalities
5. Further Reading

## Difference Logic

atoms are constraints of the form

- $x - y \leq c$
- $x - y < c$

where  $x$  and  $y$  are variables and  $c$  is some constant of  $\mathbb{Z}$  or  $\mathbb{Q}$

## Difference Logic

atoms are constraints of the form

- $x - y \leq c$
- $x - y < c$

where  $x$  and  $y$  are variables and  $c$  is some constant of  $\mathbb{Z}$  or  $\mathbb{Q}$

## Remarks

- difference logic is fragment of linear arithmetic; advantage: faster decision procedure

## Difference Logic

atoms are constraints of the form

- $x - y \leq c$
- $x - y < c$

where  $x$  and  $y$  are variables and  $c$  is some constant of  $\mathbb{Z}$  or  $\mathbb{Q}$

## Remarks

- difference logic is fragment of linear arithmetic; advantage: faster decision procedure
- domains:  $\mathbb{Z}$  or  $\mathbb{Q}$  (T-solver in polynomial time)



## Difference Logic

atoms are constraints of the form

- $x - y \leq c$
- $x - y < c$

where  $x$  and  $y$  are variables and  $c$  is some constant of  $\mathbb{Z}$  or  $\mathbb{Q}$

## Remarks

- difference logic is fragment of linear arithmetic; advantage: faster decision procedure
- domains:  $\mathbb{Z}$  or  $\mathbb{Q}$  (T-solver in polynomial time)
- $x - y = c \iff x - y \leq c \wedge y - x \leq -c$

## Difference Logic

atoms are constraints of the form

- $x - y \leq c$
- $x - y < c$

where  $x$  and  $y$  are variables and  $c$  is some constant of  $\mathbb{Z}$  or  $\mathbb{Q}$

## Remarks

- difference logic is fragment of linear arithmetic; advantage: faster decision procedure
- domains:  $\mathbb{Z}$  or  $\mathbb{Q}$  (T-solver in polynomial time)
- $x - y = c \iff x - y \leq c \wedge y - x \leq -c$
- $x - y \geq c \iff y - x \leq -c$
- $x - y > c \iff y - x < -c$

## Difference Logic

atoms are constraints of the form

- $x - y \leq c$
- $x - y < c$

where  $x$  and  $y$  are variables and  $c$  is some constant of  $\mathbb{Z}$  or  $\mathbb{Q}$

## Remarks

- difference logic is fragment of linear arithmetic; advantage: faster decision procedure
- domains:  $\mathbb{Z}$  or  $\mathbb{Q}$  ( $T$ -solver in polynomial time)
- $x - y = c \iff x - y \leq c \wedge y - x \leq -c$
- $x - y \geq c \iff y - x \leq -c$
- $x - y > c \iff y - x < -c$
- $x < c \iff x - x_0 < c$  where  $x_0$  is fresh variable that must be assigned 0

## Example Job-Shop Scheduling

- $m$  machines ( $M_1, \dots, M_m$ )

## Example Job-Shop Scheduling

- $m$  machines ( $M_1, \dots, M_m$ ) and  $n$  jobs ( $J_1, \dots, J_n$ )

## Example Job-Shop Scheduling

- $m$  machines  $(M_1, \dots, M_m)$  and  $n$  jobs  $(J_1, \dots, J_n)$
- each job  $J_i$  is sequence  $(M_1^i, d_1^i), \dots, (M_{n_i}^i, d_{n_i}^i)$  of operations consisting of machine and duration (rational number;  $\tau(M, d) = d$ )

## Example Job-Shop Scheduling

- $m$  machines  $(M_1, \dots, M_m)$  and  $n$  jobs  $(J_1, \dots, J_n)$
- each job  $J_i$  is sequence  $(M_1^i, d_1^i), \dots, (M_{n_i}^i, d_{n_i}^i)$  of operations consisting of machine and duration (rational number;  $\tau(M, d) = d$ )
- $O$  is multiset of all operations from all jobs

## Example Job-Shop Scheduling

- $m$  machines  $(M_1, \dots, M_m)$  and  $n$  jobs  $(J_1, \dots, J_n)$
- each job  $J_i$  is sequence  $(M_1^i, d_1^i), \dots, (M_{n_i}^i, d_{n_i}^i)$  of operations consisting of machine and duration (rational number;  $\tau(M, d) = d$ )
- $O$  is multiset of all operations from all jobs
- **schedule** is function  $S$  that defines for each operation  $v \in O$  its starting time  $S(v)$  on machine specified by  $v$



## Example Job-Shop Scheduling

- $m$  machines  $(M_1, \dots, M_m)$  and  $n$  jobs  $(J_1, \dots, J_n)$
- each job  $J_i$  is sequence  $(M_1^i, d_1^i), \dots, (M_{n_i}^i, d_{n_i}^i)$  of operations consisting of machine and duration (rational number;  $\tau(M, d) = d$ )
- $O$  is multiset of all operations from all jobs
- schedule is function  $S$  that defines for each operation  $v \in O$  its starting time  $S(v)$  on machine specified by  $v$
- schedule  $S$  is **feasible** if

$$S(v) \geq 0$$

for all  $v \in O$

## Example Job-Shop Scheduling

- $m$  machines  $(M_1, \dots, M_m)$  and  $n$  jobs  $(J_1, \dots, J_n)$
- each job  $J_i$  is sequence  $(M_1^i, d_1^i), \dots, (M_{n_i}^i, d_{n_i}^i)$  of operations consisting of machine and duration (rational number;  $\tau(M, d) = d$ )
- $O$  is multiset of all operations from all jobs
- schedule is function  $S$  that defines for each operation  $v \in O$  its starting time  $S(v)$  on machine specified by  $v$
- schedule  $S$  is **feasible** if

$$S(v) \geq 0 \quad \text{for all } v \in O$$

$$S(v_i) + \tau(v_i) \leq S(v_j) \quad \text{for all consecutive } v_i, v_j \text{ in same job}$$

## Example Job-Shop Scheduling

- $m$  machines  $(M_1, \dots, M_m)$  and  $n$  jobs  $(J_1, \dots, J_n)$
- each job  $J_i$  is sequence  $(M_1^i, d_1^i), \dots, (M_{n_i}^i, d_{n_i}^i)$  of operations consisting of machine and duration (rational number;  $\tau(M, d) = d$ )
- $O$  is multiset of all operations from all jobs
- schedule is function  $S$  that defines for each operation  $v \in O$  its starting time  $S(v)$  on machine specified by  $v$
- schedule  $S$  is **feasible** if

$$S(v) \geq 0 \quad \text{for all } v \in O$$

$$S(v_i) + \tau(v_i) \leq S(v_j) \quad \text{for all consecutive } v_i, v_j \text{ in same job}$$

$$S(v_i) + \tau(v_i) \leq S(v_j) \vee S(v_j) + \tau(v_j) \leq S(v_i)$$

for every pair of different operations  $v_i, v_j$  scheduled on same machine

## Example Job-Shop Scheduling

- $m$  machines  $(M_1, \dots, M_m)$  and  $n$  jobs  $(J_1, \dots, J_n)$
- each job  $J_i$  is sequence  $(M_1^i, d_1^i), \dots, (M_{n_i}^i, d_{n_i}^i)$  of operations consisting of machine and duration (rational number;  $\tau(M, d) = d$ )
- $O$  is multiset of all operations from all jobs
- schedule is function  $S$  that defines for each operation  $v \in O$  its starting time  $S(v)$  on machine specified by  $v$
- schedule  $S$  is feasible if

$$S(v) \geq 0 \quad \text{for all } v \in O$$

$$S(v_i) + \tau(v_i) \leq S(v_j) \quad \text{for all consecutive } v_i, v_j \text{ in same job}$$

$$S(v_i) + \tau(v_i) \leq S(v_j) \vee S(v_j) + \tau(v_j) \leq S(v_i)$$

for every pair of different operations  $v_i, v_j$  scheduled on same machine

- aim: minimize global duration  $gd$ ; add constraints  $S(v) + \tau(v) \leq gd$  for each  $v \in O$

## Definition Inequality Graph

conjunction  $\varphi$  of nonstrict difference constraints

- **inequality graph** of  $\varphi$  contains edge from  $x$  to  $y$  with weight  $c$  for every constraint  $x - y \leq c$  in  $\varphi$

## Definition Inequality Graph

conjunction  $\varphi$  of nonstrict difference constraints

- inequality graph of  $\varphi$  contains edge from  $x$  to  $y$  with weight  $c$  for every constraint  $x - y \leq c$  in  $\varphi$

## Theorem

conjunction  $\varphi$  of nonstrict difference constraints is satisfiable

inequality graph of  $\varphi$  has no negative cycle



## Definition Inequality Graph

conjunction  $\varphi$  of nonstrict difference constraints

- inequality graph of  $\varphi$  contains edge from  $x$  to  $y$  with weight  $c$  for every constraint  $x - y \leq c$  in  $\varphi$

## Theorem

conjunction  $\varphi$  of nonstrict difference constraints is satisfiable  $\iff$

inequality graph of  $\varphi$  has no negative cycle

## Example

$$x - y \leq 2 \quad x \xrightarrow{2} y$$

## Definition Inequality Graph

conjunction  $\varphi$  of nonstrict difference constraints

- inequality graph of  $\varphi$  contains edge from  $x$  to  $y$  with weight  $c$  for every constraint  $x - y \leq c$  in  $\varphi$

## Theorem

conjunction  $\varphi$  of nonstrict difference constraints is satisfiable  $\iff$

inequality graph of  $\varphi$  has no negative cycle

## Example

$$x - y \leq 2$$

$$y - z \leq -3$$

$$x \xrightarrow{2} y \xrightarrow{-3} z$$



## Definition Inequality Graph

conjunction  $\varphi$  of nonstrict difference constraints

- inequality graph of  $\varphi$  contains edge from  $x$  to  $y$  with weight  $c$  for every constraint  $x - y \leq c$  in  $\varphi$

## Theorem

conjunction  $\varphi$  of nonstrict difference constraints is satisfiable  $\iff$

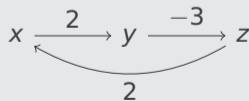
inequality graph of  $\varphi$  has no negative cycle

## Example

$$x - y \leq 2$$

$$y - z \leq -3$$

$$z - x \leq 2$$



## Definition Inequality Graph

conjunction  $\varphi$  of nonstrict difference constraints

- inequality graph of  $\varphi$  contains edge from  $x$  to  $y$  with weight  $c$  for every constraint  $x - y \leq c$  in  $\varphi$

## Theorem

conjunction  $\varphi$  of nonstrict difference constraints is satisfiable  $\iff$

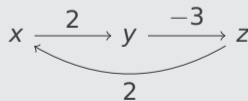
inequality graph of  $\varphi$  has no negative cycle

## Example

$$x - y \leq 2$$

$$y - z \leq -3$$

$$z - x \leq 2$$



satisfiable

## Theorem

conjunction  $\varphi$  of nonstrict difference constraints is satisfiable  
inequality graph of  $\varphi$  has no negative cycle



## Proof

cycle

$$x_1 \xrightarrow{k_1} x_2 \xrightarrow{k_2} x_3 \longrightarrow \cdots \longrightarrow x_n \xrightarrow{k_n} x_1$$

in inequality graph of  $\varphi$

## Theorem

conjunction  $\varphi$  of nonstrict difference constraints is satisfiable  
inequality graph of  $\varphi$  has no negative cycle



## Proof

cycle

$$x_1 \xrightarrow{k_1} x_2 \xrightarrow{k_2} x_3 \longrightarrow \cdots \longrightarrow x_n \xrightarrow{k_n} x_1$$

in inequality graph of  $\varphi$  corresponds to conjunction

$$x_1 - x_2 \leq k_1 \wedge x_2 - x_3 \leq k_2 \wedge \cdots \wedge x_n - x_1 \leq k_n$$

## Theorem

conjunction  $\varphi$  of nonstrict difference constraints is satisfiable  
inequality graph of  $\varphi$  has no negative cycle



## Proof

cycle

$$x_1 \xrightarrow{k_1} x_2 \xrightarrow{k_2} x_3 \longrightarrow \cdots \longrightarrow x_n \xrightarrow{k_n} x_1$$

in inequality graph of  $\varphi$  corresponds to conjunction

$$x_1 - x_2 \leq k_1 \wedge x_2 - x_3 \leq k_2 \wedge \cdots \wedge x_n - x_1 \leq k_n$$

adding these literals gives

$$0 \leq k_1 + k_2 + \cdots + k_n$$

## Theorem

conjunction  $\varphi$  of nonstrict difference constraints is satisfiable  
inequality graph of  $\varphi$  has no negative cycle



## Proof

⇒ negative cycle

$$x_1 \xrightarrow{k_1} x_2 \xrightarrow{k_2} x_3 \longrightarrow \cdots \longrightarrow x_n \xrightarrow{k_n} x_1$$

in inequality graph of  $\varphi$  corresponds to conjunction

$$x_1 - x_2 \leq k_1 \wedge x_2 - x_3 \leq k_2 \wedge \cdots \wedge x_n - x_1 \leq k_n$$

adding these literals gives

$$0 \leq k_1 + k_2 + \cdots + k_n$$

with  $k_1 + k_2 + \cdots + k_n < 0$



## Theorem

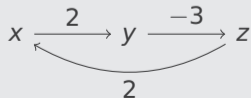
conjunction  $\varphi$  of nonstrict difference constraints is satisfiable  
inequality graph of  $\varphi$  has no negative cycle



## Proof

⇐ assume inequality graph of  $\varphi$  has no negative cycle  
construct satisfying assignment for  $\varphi$  as follows

## Example



## Theorem

conjunction  $\varphi$  of nonstrict difference constraints is satisfiable  
inequality graph of  $\varphi$  has no negative cycle



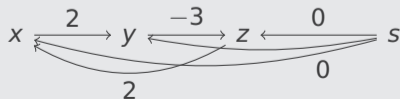
## Proof

⇐ assume inequality graph of  $\varphi$  has no negative cycle

construct satisfying assignment for  $\varphi$  as follows

- add additional starting node  $s$  in graph, add edges  $s \rightarrow x$  with weight 0 for all variables  $x$

## Example





## Theorem

conjunction  $\varphi$  of nonstrict difference constraints is satisfiable  
inequality graph of  $\varphi$  has no negative cycle



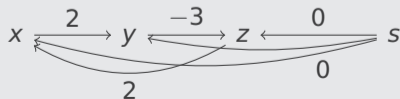
## Proof

⇐ assume inequality graph of  $\varphi$  has no negative cycle

construct satisfying assignment for  $\varphi$  as follows

- add additional starting node  $s$  in graph, add edges  $s \rightarrow x$  with weight 0 for all variables  $x$
- define  $v(x) = -\text{distance}(s, x)$ ; well-defined, since there are no negative cycles

## Example



$$v(x) = 1, v(y) = 0, v(z) = 3$$

## Theorem

conjunction  $\varphi$  of nonstrict difference constraints is satisfiable  
inequality graph of  $\varphi$  has no negative cycle



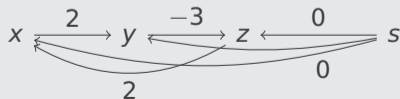
## Proof

⇐ assume inequality graph of  $\varphi$  has no negative cycle

construct satisfying assignment for  $\varphi$  as follows

- add additional starting node  $s$  in graph, add edges  $s \rightarrow x$  with weight 0 for all variables  $x$
- define  $v(x) = -\text{distance}(s, x)$ ; well-defined, since there are no negative cycles
- $v$  satisfies  $\varphi$

## Example



$$v(x) = 1, v(y) = 0, v(z) = 3$$

- Dijkstra
  - computes distances from a single source to all other nodes
  - complexity:  $\mathcal{O}(|V| \cdot \log(|V|) + |E|)$
  - restriction: **no negative cycles allowed**

## Algorithms for Distance Computation and Negative Cycle Detection

- Dijkstra
  - computes distances from a single source to all other nodes
  - complexity:  $\mathcal{O}(|V| \cdot \log(|V|) + |E|)$
  - restriction: **no negative cycles allowed**
- Bellman–Ford
  - computes distances from a single source to all other nodes
  - complexity:  $\mathcal{O}(|V| \cdot |E|)$
  - can also detect negative cycles

## Algorithms for Distance Computation and Negative Cycle Detection

- Dijkstra
  - computes distances from a single source to all other nodes
  - complexity:  $\mathcal{O}(|V| \cdot \log(|V|) + |E|)$
  - restriction: **no negative cycles allowed**
- Bellman–Ford
  - computes distances from a single source to all other nodes
  - complexity:  $\mathcal{O}(|V| \cdot |E|)$
  - can also detect negative cycles
- Floyd–Warshall
  - computes distances between all nodes
  - complexity:  $\mathcal{O}(|V|^3)$
  - can also detect negative cycles

## Algorithms for Distance Computation and Negative Cycle Detection

- Dijkstra
  - computes distances from a single source to all other nodes
  - complexity:  $\mathcal{O}(|V| \cdot \log(|V|) + |E|)$
  - restriction: **no negative cycles allowed**
- Bellman–Ford
  - computes distances from a single source to all other nodes
  - complexity:  $\mathcal{O}(|V| \cdot |E|)$
  - can also detect negative cycles
- Floyd–Warshall
  - computes distances between all nodes
  - complexity:  $\mathcal{O}(|V|^3)$
  - can also detect negative cycles

⇒ use Bellman–Ford algorithm for difference logic

## Bellman–Ford Algorithm for Inequality Graphs

**Input** inequality graph  $(V, E, w)$  with fresh starting node  $s$

**Output** “ $\exists$  negative cycle” or distances to node  $s$

## Bellman–Ford Algorithm for Inequality Graphs

**Input** inequality graph  $(V, E, w)$  with fresh starting node  $s$

**Output** “ $\exists$  negative cycle” or distances to node  $s$

- 1 distance[ $v$ ] := 0 for all nodes  $v \in V$  (this step is special for inequality graphs)



## Bellman–Ford Algorithm for Inequality Graphs

**Input** inequality graph  $(V, E, w)$  with fresh starting node  $s$

**Output** “ $\exists$  negative cycle” or distances to node  $s$

- 1 distance[ $v$ ] := 0 for all nodes  $v \in V$  (this step is special for inequality graphs)
- 2 repeat  $|V| - 1$  times  
for all  $(u, v) \in E$  do

## Bellman–Ford Algorithm for Inequality Graphs

**Input** inequality graph  $(V, E, w)$  with fresh starting node  $s$

**Output** “ $\exists$  negative cycle” or distances to node  $s$

- 1 distance[ $v$ ] := 0 for all nodes  $v \in V$  (this step is special for inequality graphs)
- 2 repeat  $|V| - 1$  times  
for all  $(u, v) \in E$  do  
if distance[ $v$ ] > distance[ $u$ ] +  $w(u, v)$  then

## Bellman–Ford Algorithm for Inequality Graphs

**Input** inequality graph  $(V, E, w)$  with fresh starting node  $s$

**Output** “ $\exists$  negative cycle” or distances to node  $s$

- 1 distance[ $v$ ] := 0 for all nodes  $v \in V$  (this step is special for inequality graphs)
- 2 repeat  $|V| - 1$  times  
for all  $(u, v) \in E$  do  
if distance[ $v$ ] > distance[ $u$ ] +  $w(u, v)$  then  
distance[ $v$ ] := distance[ $u$ ] +  $w(u, v)$   
predecessor[ $v$ ] :=  $u$

## Bellman–Ford Algorithm for Inequality Graphs

**Input** inequality graph  $(V, E, w)$  with fresh starting node  $s$

**Output** “ $\exists$  negative cycle” or distances to node  $s$

- 1 distance[ $v$ ] := 0 for all nodes  $v \in V$  (this step is special for inequality graphs)
- 2 repeat  $|V| - 1$  times  
for all  $(u, v) \in E$  do  
if distance[ $v$ ] > distance[ $u$ ] +  $w(u, v)$  then  
distance[ $v$ ] := distance[ $u$ ] +  $w(u, v)$   
predecessor[ $v$ ] :=  $u$
- 3 for all  $(u, v) \in E$  do  
if distance[ $v$ ] > distance[ $u$ ] +  $w(u, v)$  then  
return “ $\exists$  negative cycle”

## Bellman–Ford Algorithm for Inequality Graphs

**Input** inequality graph  $(V, E, w)$  with fresh starting node  $s$

**Output** “ $\exists$  negative cycle” or distances to node  $s$

- 1 distance[ $v$ ] := 0 for all nodes  $v \in V$  (this step is special for inequality graphs)
- 2 repeat  $|V| - 1$  times  
for all  $(u, v) \in E$  do  
if distance[ $v$ ] > distance[ $u$ ] +  $w(u, v)$  then  
distance[ $v$ ] := distance[ $u$ ] +  $w(u, v)$   
predecessor[ $v$ ] :=  $u$
- 3 for all  $(u, v) \in E$  do  
if distance[ $v$ ] > distance[ $u$ ] +  $w(u, v)$  then  
return “ $\exists$  negative cycle”  
which can be reconstructed using predecessor array

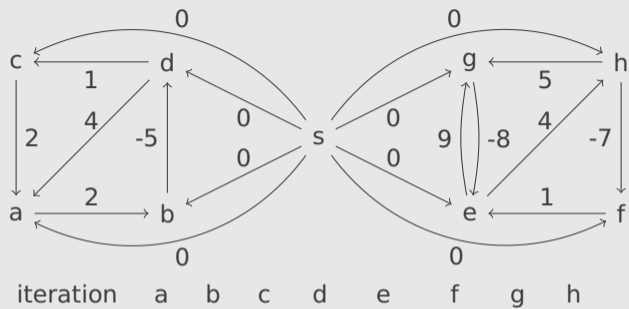
## Bellman–Ford Algorithm for Inequality Graphs

**Input** inequality graph  $(V, E, w)$  with fresh starting node  $s$

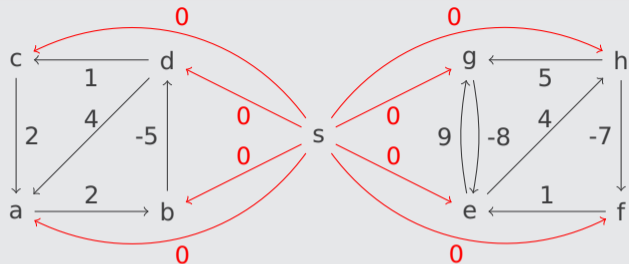
**Output** “ $\exists$  negative cycle” or distances to node  $s$

- 1 distance[ $v$ ] := 0 for all nodes  $v \in V$  (this step is special for inequality graphs)
- 2 repeat  $|V| - 1$  times  
for all  $(u, v) \in E$  do  
if distance[ $v$ ] > distance[ $u$ ] +  $w(u, v)$  then  
distance[ $v$ ] := distance[ $u$ ] +  $w(u, v)$   
predecessor[ $v$ ] :=  $u$
- 3 for all  $(u, v) \in E$  do  
if distance[ $v$ ] > distance[ $u$ ] +  $w(u, v)$  then  
return “ $\exists$  negative cycle”  
which can be reconstructed using predecessor array
- 4 return distance array, shortest paths available via predecessor array

# Example Bellman-Ford in Action



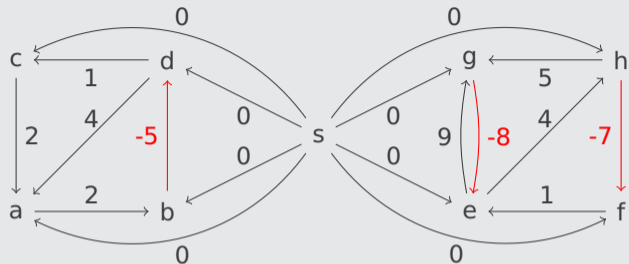
# Example Bellman-Ford in Action



iteration	a	b	c	d	e	f	g	h
0	0	0	0	0	0	0	0	0

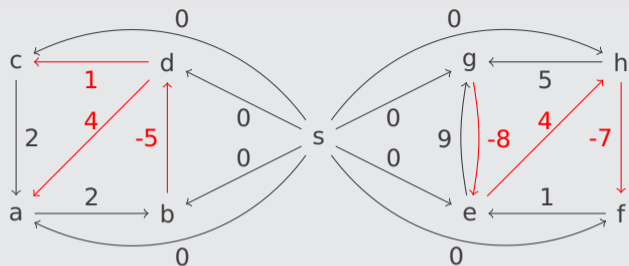


# Example Bellman-Ford in Action



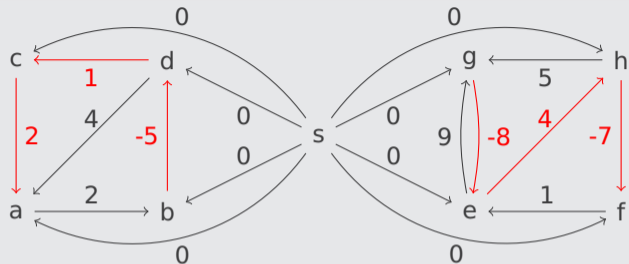
iteration	a	b	c	d	e	f	g	h
0	0	0	0	0	0	0	0	0
1	0	0	0	-5	-8	-7	0	0

## Example Bellman-Ford in Action



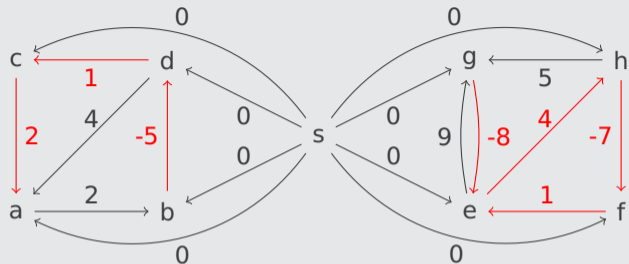
iteration	a	b	c	d	e	f	g	h
0	0	0	0	0	0	0	0	0
1	0	0	0	-5	-8	-7	0	0
2	-1	0	-4	-5	-8	-7	0	-4

# Example Bellman-Ford in Action



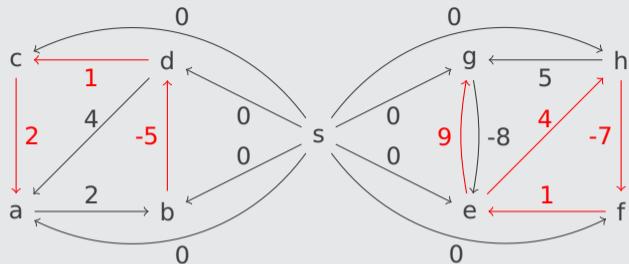
iteration	a	b	c	d	e	f	g	h
0	0	0	0	0	0	0	0	0
1	0	0	0	-5	-8	-7	0	0
2	-1	0	-4	-5	-8	-7	0	-4
3	-2	0	-4	-5	-8	-11	0	-4

# Example Bellman-Ford in Action



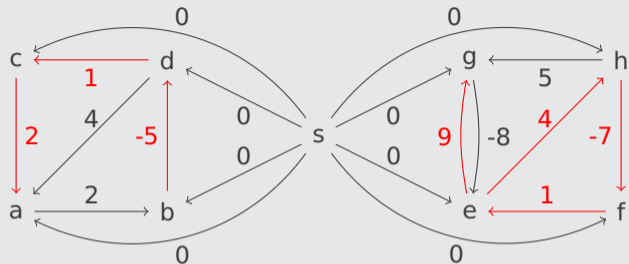
iteration	a	b	c	d	e	f	g	h
0	0	0	0	0	0	0	0	0
1	0	0	0	-5	-8	-7	0	0
2	-1	0	-4	-5	-8	-7	0	-4
3	-2	0	-4	-5	-8	-11	0	-4
4	-2	0	-4	-5	-10	-11	0	-4

# Example Bellman-Ford in Action



iteration	a	b	c	d	e	f	g	h
0	0	0	0	0	0	0	0	0
1	0	0	0	-5	-8	-7	0	0
2	-1	0	-4	-5	-8	-7	0	-4
3	-2	0	-4	-5	-8	-11	0	-4
4	-2	0	-4	-5	-10	-11	0	-4
5	-2	0	-4	-5	-10	-11	-1	-6

## Example Bellman-Ford in Action



iteration	a	b	c	d	e	f	g	h
0	0	0	0	0	0	0	0	0
1	0	0	0	-5	-8	-7	0	0
2	-1	0	-4	-5	-8	-7	0	-4
3	-2	0	-4	-5	-8	-11	0	-4
4	-2	0	-4	-5	-10	-11	0	-4
5	-2	0	-4	-5	-10	-11	-1	-6

... 2 more iterations, then negative cycle is detected

# Outline

1. Summary of Previous Lecture
2. Difference Logic
- 3. Simplex Algorithm**
4. Support of Strict Inequalities
5. Further Reading

## Definition (Theory of Linear Arithmetic over $D$ )

- for variables  $x_1, \dots, x_n$ , built quantifier-free formulas according to grammar

$$\varphi ::= \neg\varphi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid t < t \mid t \leq t$$

$$t ::= a_1x_1 + \dots + a_nx_n + b$$

$$s = t \text{ is encoded as } s \leq t \wedge t \leq s$$

$$\text{for } a_1, \dots, a_n, b \in \text{in domain } D$$

- solution** assigns values in  $D$  to  $x_1, \dots, x_n$



## Definition (Theory of Linear Arithmetic over $D$ )

- for variables  $x_1, \dots, x_n$ , built quantifier-free formulas according to grammar

$$\varphi ::= \neg\varphi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid t < t \mid t \leq t$$

$$t ::= a_1x_1 + \dots + a_nx_n + b$$

$$s = t \text{ is encoded as } s \leq t \wedge t \leq s$$

$$\text{for } a_1, \dots, a_n, b \in \text{in domain } D$$

- solution** assigns values in  $D$  to  $x_1, \dots, x_n$

## Definitions

- Linear Real Arithmetic (LRA)** uses domain  $D = \mathbb{R}$
- Linear Integer Arithmetic (LIA)** uses domain  $D = \mathbb{Z}$

## Definition (Theory of Linear Arithmetic over $D$ )

- for variables  $x_1, \dots, x_n$ , built quantifier-free formulas according to grammar

$$\varphi ::= \neg\varphi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid t < t \mid t \leq t$$

$$t ::= a_1x_1 + \dots + a_nx_n + b$$

$$s = t \text{ is encoded as } s \leq t \wedge t \leq s$$

$$\text{for } a_1, \dots, a_n, b \in \text{in domain } D$$

- solution** assigns values in  $D$  to  $x_1, \dots, x_n$

## Definitions

- Linear Real Arithmetic (LRA)** uses domain  $D = \mathbb{R}$
- Linear Integer Arithmetic (LIA)** uses domain  $D = \mathbb{Z}$

## Definition (Theory of Linear Arithmetic over $D$ )

- for variables  $x_1, \dots, x_n$ , built quantifier-free formulas according to grammar

$$\varphi ::= \neg\varphi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid t < t \mid t \leq t$$

$$t ::= a_1x_1 + \dots + a_nx_n + b$$

$$s = t \text{ is encoded as } s \leq t \wedge t \leq s$$

$$\text{for } a_1, \dots, a_n, b \in \text{in domain } D$$

- solution** assigns values in  $D$  to  $x_1, \dots, x_n$

## Definitions

- Linear Real Arithmetic (LRA)** uses domain  $D = \mathbb{R}$
- Linear Integer Arithmetic (LIA)** uses domain  $D = \mathbb{Z}$

## Example

- $x + y + z = 2 \wedge z > y \wedge y > -1$

## Definition (Theory of Linear Arithmetic over $D$ )

- for variables  $x_1, \dots, x_n$ , built quantifier-free formulas according to grammar

$$\varphi ::= \neg\varphi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid t < t \mid t \leq t \quad s = t \text{ is encoded as } s \leq t \wedge t \leq s$$
$$t ::= a_1x_1 + \dots + a_nx_n + b \quad \text{for } a_1, \dots, a_n, b \in \text{in domain } D$$

- solution** assigns values in  $D$  to  $x_1, \dots, x_n$

## Definitions

- Linear Real Arithmetic (LRA)** uses domain  $D = \mathbb{R}$
- Linear Integer Arithmetic (LIA)** uses domain  $D = \mathbb{Z}$

## Example

- $x + y + z = 2 \wedge z > y \wedge y > -1$   
is satisfiable in LRA and LIA, e.g. with  $v(x) = v(y) = 0$  and  $v(z) = 2$

## Definition (Theory of Linear Arithmetic over $D$ )

- for variables  $x_1, \dots, x_n$ , built quantifier-free formulas according to grammar

$$\varphi ::= \neg\varphi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid t < t \mid t \leq t \quad s = t \text{ is encoded as } s \leq t \wedge t \leq s$$
$$t ::= a_1x_1 + \dots + a_nx_n + b \quad \text{for } a_1, \dots, a_n, b \in \text{in domain } D$$

- solution** assigns values in  $D$  to  $x_1, \dots, x_n$

## Definitions

- Linear Real Arithmetic (LRA)** uses domain  $D = \mathbb{R}$
- Linear Integer Arithmetic (LIA)** uses domain  $D = \mathbb{Z}$

## Example

- $x + y + z = 2 \wedge z > y \wedge y > -1$   
is satisfiable in LRA and LIA, e.g. with  $v(x) = v(y) = 0$  and  $v(z) = 2$
- $x < 3 \wedge 2x > 4$

## Definition (Theory of Linear Arithmetic over $D$ )

- for variables  $x_1, \dots, x_n$ , built quantifier-free formulas according to grammar

$$\varphi ::= \neg\varphi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid t < t \mid t \leq t \quad s = t \text{ is encoded as } s \leq t \wedge t \leq s$$
$$t ::= a_1x_1 + \dots + a_nx_n + b \quad \text{for } a_1, \dots, a_n, b \in \text{in domain } D$$

- **solution** assigns values in  $D$  to  $x_1, \dots, x_n$

## Definitions

- **Linear Real Arithmetic (LRA)** uses domain  $D = \mathbb{R}$
- **Linear Integer Arithmetic (LIA)** uses domain  $D = \mathbb{Z}$

## Example

- $x + y + z = 2 \wedge z > y \wedge y > -1$   
is satisfiable in LRA and LIA, e.g. with  $v(x) = v(y) = 0$  and  $v(z) = 2$
- $x < 3 \wedge 2x > 4$   
is unsatisfiable in LIA but satisfiable in LRA, e.g. with  $v(x) = 2.5$

## Relevance of Linear Arithmetic

LRA and LIA admit more natural and succinct encodings of

- everything with cardinality constraints:  $n$ -queens, Sudoku, Minesweeper, ...
- planning problems
- scheduling problems
- component configuration problems
- ...

## T-Solver for Conjunctions of Linear Arithmetic Inequalities

- **integers** (LIA): **NP-complete**



## T-Solver for Conjunctions of Linear Arithmetic Inequalities

- integers (LIA): **NP-complete**
- **reals** or rationals (LRA): **polynomial** **Simplex algorithm**

## T-Solver for Conjunctions of Linear Arithmetic Inequalities

- integers (LIA): **NP-complete**
- reals or rationals (LRA): **polynomial** **Simplex algorithm**

## Some History

**1947 Danzig** proposed Simplex algorithm to solve **optimization problem**:

$$\text{maximize } c(\vec{x}) \quad \text{such that} \quad A\vec{x} \leq b \text{ and } \vec{x} \geq 0$$

for linear objective function  $c$ , matrix  $A$ , vector  $b$ , and vector of variables  $\vec{x}$

- ▶ also known as **linear programming**

## T-Solver for Conjunctions of Linear Arithmetic Inequalities

- integers (LIA): **NP-complete**
- reals or rationals (LRA): **polynomial** **Simplex algorithm**

**exponential worst-case complexity**

## Some History

**1947 Danzig** proposed Simplex algorithm to solve optimization problem:

$$\text{maximize } c(\vec{x}) \quad \text{such that} \quad A\vec{x} \leq b \text{ and } \vec{x} \geq 0$$

for linear objective function  $c$ , matrix  $A$ , vector  $b$ , and vector of variables  $\vec{x}$

- ▶ also known as linear programming

## T-Solver for Conjunctions of Linear Arithmetic Inequalities

- integers (LIA): **NP-complete**
- reals or rationals (LRA): **polynomial** **Simplex algorithm**

exponential worst-case complexity

### Some History

**1947 Danzig** proposed Simplex algorithm to solve optimization problem:

$$\text{maximize } c(\vec{x}) \quad \text{such that} \quad A\vec{x} \leq b \text{ and } \vec{x} \geq 0$$

for linear objective function  $c$ , matrix  $A$ , vector  $b$ , and vector of variables  $\vec{x}$

- ▶ also known as linear programming

**1979 Khachiyan** proposed **polynomial** version based on ellipsoid method

## T-Solver for Conjunctions of Linear Arithmetic Inequalities

- integers (LIA): **NP-complete**
- reals or rationals (LRA): **polynomial** **Simplex algorithm**

exponential worst-case complexity

## Some History

**1947 Danzig** proposed Simplex algorithm to solve optimization problem:

$$\text{maximize } c(\vec{x}) \quad \text{such that} \quad A\vec{x} \leq b \text{ and } \vec{x} \geq 0$$

for linear objective function  $c$ , matrix  $A$ , vector  $b$ , and vector of variables  $\vec{x}$

- ▶ also known as linear programming

**1979 Khachiyan** proposed polynomial version based on ellipsoid method

**1984 Karmakar** proposed **polynomial** version based on interior points method

## T-Solver for Conjunctions of Linear Arithmetic Inequalities

- integers (LIA): **NP-complete**
- reals or rationals (LRA): **polynomial** **Simplex algorithm**

exponential worst-case complexity

### Some History

**1947 Danzig** proposed Simplex algorithm to solve optimization problem:

$$\text{maximize } c(\vec{x}) \quad \text{such that} \quad A\vec{x} \leq b \text{ and } \vec{x} \geq 0$$

for linear objective function  $c$ , matrix  $A$ , vector  $b$ , and vector of variables  $\vec{x}$

- ▶ also known as linear programming

**1979 Khachiyan** proposed polynomial version based on ellipsoid method

**1984 Karmakar** proposed polynomial version based on interior points method

**2000-** T-solver version of simplex algorithm to solve **satisfiability problem**

$$A\vec{x} \leq b$$

## Problem Input (General Form)

- $m$  equalities

$$a_1x_1 + \dots + a_nx_n = 0$$

## Problem Input (General Form)

- $m$  equalities
- (optional) lower and upper bounds on variables

$$a_1x_1 + \dots + a_nx_n = 0$$

$$l_i \leq x_i \leq u_i$$



## Problem Input (General Form)

- $m$  equalities
- (optional) lower and upper bounds on variables

$$a_1x_1 + \dots + a_nx_n = 0$$

$$l_i \leq x_i \leq u_i$$

## Lemma

*any conjunctive LRA problem without  $<$  can be turned into equisatisfiable general form*

## Problem Input (General Form)

- $m$  equalities
- (optional) lower and upper bounds on variables

$$a_1x_1 + \dots + a_nx_n = 0$$

$$l_i \leq x_i \leq u_i$$

## Lemma

any conjunctive LRA problem without  $<$  can be turned into equisatisfiable general form

## Example

$$\begin{array}{l} x - y \geq -1 \\ y \leq 4 \\ x + y \geq 6 \\ 3x - y \leq 7 \end{array} \quad \implies \quad \begin{array}{l} -x + y - s_1 = 0 \quad s_1 \leq 1 \\ y - s_2 = 0 \quad s_2 \leq 4 \\ -x - y - s_3 = 0 \quad s_3 \leq -6 \\ 3x - y - s_4 = 0 \quad s_4 \leq 7 \end{array}$$

## Problem Input (General Form)

- $m$  equalities
- (optional) lower and upper bounds on variables

$$a_1x_1 + \dots + a_nx_n = 0$$

$$l_i \leq x_i \leq u_i$$

## Lemma

any conjunctive LRA problem without  $<$  can be turned into equisatisfiable general form

## Example

$$\begin{array}{lcl} x - y \geq -1 & \implies & -x + y - s_1 = 0 \quad s_1 \leq 1 \\ y \leq 4 & & y - s_2 = 0 \quad s_2 \leq 4 \\ x + y \geq 6 & & -x - y - s_3 = 0 \quad s_3 \leq -6 \\ 3x - y \leq 7 & & 3x - y - s_4 = 0 \quad s_4 \leq 7 \end{array}$$

- $s_1, s_2, s_3, s_4$  are **slack variables**

## Problem Input (General Form)

- $m$  equalities
- (optional) lower and upper bounds on variables

$$a_1x_1 + \dots + a_nx_n = 0$$

$$l_i \leq x_i \leq u_i$$

## Lemma

*any conjunctive LRA problem without  $<$  can be turned into equisatisfiable general form*

## Example

$$\begin{array}{lcl} x - y \geq -1 & \implies & -x + y - s_1 = 0 \quad s_1 \leq 1 \\ y \leq 4 & & y - s_2 = 0 \quad s_2 \leq 4 \\ x + y \geq 6 & & -x - y - s_3 = 0 \quad s_3 \leq -6 \\ 3x - y \leq 7 & & 3x - y - s_4 = 0 \quad s_4 \leq 7 \end{array}$$

- $s_1, s_2, s_3, s_4$  are slack variables
- $x, y$  are **problem variables**

## Representation

- represent equalities

$$-x + y - s_1 = 0$$

$$y - s_2 = 0$$

$$-x - y - s_3 = 0$$

$$3x - y - s_4 = 0$$

## Representation

- represent equalities

$$-x + y - s_1 = 0$$

$$y - s_2 = 0$$

$$-x - y - s_3 = 0$$

$$3x - y - s_4 = 0$$

via  $m \times n$  matrix presentation

$$\begin{array}{c} s_1 \\ s_2 \\ s_3 \\ s_4 \end{array} \begin{array}{cc} x & y \\ \left( \begin{array}{cc} -1 & 1 \\ 0 & 1 \\ -1 & -1 \\ 3 & -1 \end{array} \right) \end{array}$$

## Representation

- represent equalities

$$-x + y - s_1 = 0$$

$$y - s_2 = 0$$

$$-x - y - s_3 = 0$$

$$3x - y - s_4 = 0$$

via  $m \times n$  matrix presentation

**basic** variables  $\rightarrow$

$$\begin{matrix} s_1 \\ s_2 \\ s_3 \\ s_4 \end{matrix} \begin{pmatrix} x & y \\ -1 & 1 \\ 0 & 1 \\ -1 & -1 \\ 3 & -1 \end{pmatrix}$$

## Representation

- represent equalities

$$-x + y - s_1 = 0$$

$$y - s_2 = 0$$

$$-x - y - s_3 = 0$$

$$3x - y - s_4 = 0$$

via  $m \times n$  matrix presentation

**basic** variables  $\rightarrow$

$$\begin{matrix} & x & y \\ s_1 & \begin{pmatrix} -1 & 1 \end{pmatrix} \\ s_2 & \begin{pmatrix} 0 & 1 \end{pmatrix} \\ s_3 & \begin{pmatrix} -1 & -1 \end{pmatrix} \\ s_4 & \begin{pmatrix} 3 & -1 \end{pmatrix} \end{matrix} \quad \leftarrow \text{nonbasic variables}$$



## Representation

- represent equalities

$$-x + y - s_1 = 0$$

$$y - s_2 = 0$$

$$-x - y - s_3 = 0$$

$$3x - y - s_4 = 0$$

via  $m \times n$  matrix presentation

$$\begin{array}{l} \text{basic variables} \rightarrow \\ s_1 \\ s_2 \\ s_3 \\ s_4 \end{array} \begin{array}{cc} & \begin{array}{cc} x & y \end{array} \\ \begin{pmatrix} -1 & 1 \\ 0 & 1 \\ -1 & -1 \\ 3 & -1 \end{pmatrix} & \leftarrow \text{nonbasic variables} \end{array}$$

## Notation

- matrix is **tableau**
- $B$  is set of **basic variables** (in tableau listed vertically)
- $N$  is set of **nonbasic variables** (in tableau listed horizontally)

## Simplex Algorithm as $T$ -solver

**Input:** conjunction of LRA atoms  $\varphi$  without  $<$   
**Output:** satisfiable assignment or unsatisfiable

## Simplex Algorithm as $T$ -solver

**Input:** conjunction of LRA atoms  $\varphi$  without  $<$

**Output:** satisfiable assignment or unsatisfiable

1 transform  $\varphi$  into **general form** and construct **tableau**

## Simplex Algorithm as $T$ -solver

**Input:** conjunction of LRA atoms  $\varphi$  without  $<$

**Output:** satisfiable assignment or unsatisfiable

1 transform  $\varphi$  into general form and construct tableau

2 fix order on variables and **assign 0 to each variable**

## Simplex Algorithm as $T$ -solver

**Input:** conjunction of LRA atoms  $\varphi$  without  $<$

**Output:** satisfiable assignment or unsatisfiable

- 1 transform  $\varphi$  into general form and construct tableau
- 2 fix order on variables and assign 0 to each variable
- 3 if all **basic variables satisfy** their **bounds** then return **current (satisfying) assignment**

## Simplex Algorithm as $T$ -solver

**Input:** conjunction of LRA atoms  $\varphi$  without  $<$

**Output:** satisfiable assignment or unsatisfiable

- 1 transform  $\varphi$  into general form and construct tableau
- 2 fix order on variables and assign 0 to each variable
- 3 if all basic variables satisfy their bounds then return **current (satisfying) assignment**
- 4 let  $x_i \in B$  be variable that **violates its bounds**

## Simplex Algorithm as $T$ -solver

**Input:** conjunction of LRA atoms  $\varphi$  without  $<$

**Output:** satisfiable assignment or unsatisfiable

- 1 transform  $\varphi$  into general form and construct tableau
- 2 fix order on variables and assign 0 to each variable
- 3 if all basic variables satisfy their bounds then return **current (satisfying) assignment**
- 4 let  $x_i \in B$  be variable that violates its bounds
- 5 search for suitable variable  $x_j \in N$  for **pivoting** with  $x_i$

## Simplex Algorithm as $T$ -solver

**Input:** conjunction of LRA atoms  $\varphi$  without  $<$

**Output:** satisfiable assignment or unsatisfiable

- 1 transform  $\varphi$  into general form and construct tableau
- 2 fix order on variables and assign 0 to each variable
- 3 if all basic variables satisfy their bounds then return **current (satisfying) assignment**
- 4 let  $x_i \in B$  be variable that violates its bounds
- 5 search for suitable variable  $x_j \in N$  for pivoting with  $x_i$
- 6 return **unsatisfiable** if search unsuccessful



## Simplex Algorithm as $T$ -solver

**Input:** conjunction of LRA atoms  $\varphi$  without  $<$

**Output:** satisfiable assignment or unsatisfiable

- 1 transform  $\varphi$  into general form and construct tableau
- 2 fix order on variables and assign 0 to each variable
- 3 if all basic variables satisfy their bounds then return **current (satisfying) assignment**
- 4 let  $x_i \in B$  be variable that violates its bounds
- 5 search for suitable variable  $x_j \in N$  for pivoting with  $x_i$
- 6 return **unsatisfiable** if search unsuccessful
- 7 perform **pivot** operation on  $x_i$  and  $x_j$

## Simplex Algorithm as $T$ -solver

**Input:** conjunction of LRA atoms  $\varphi$  without  $<$

**Output:** satisfiable assignment or unsatisfiable

- 1 transform  $\varphi$  into general form and construct tableau
- 2 fix order on variables and assign 0 to each variable
- 3 if all basic variables satisfy their bounds then return **current (satisfying) assignment**
- 4 let  $x_i \in B$  be variable that violates its bounds
- 5 search for suitable variable  $x_j \in N$  for pivoting with  $x_i$
- 6 return **unsatisfiable** if search unsuccessful
- 7 perform **pivot** operation on  $x_i$  and  $x_j$
- 9 update assignment

## Simplex Algorithm as $T$ -solver

**Input:** conjunction of LRA atoms  $\varphi$  without  $<$

**Output:** satisfiable assignment or unsatisfiable

- 1 transform  $\varphi$  into general form and construct tableau
- 2 fix order on variables and assign 0 to each variable
- 3 if all basic variables satisfy their bounds then return **current (satisfying) assignment**
- 4 let  $x_i \in B$  be variable that violates its bounds
- 5 search for suitable variable  $x_j \in N$  for pivoting with  $x_i$
- 6 return **unsatisfiable** if search unsuccessful
- 7 perform **pivot** operation on  $x_i$  and  $x_j$
- 9 update assignment
- 10 go to step 3

## Simplex Algorithm as $T$ -solver

**Input:** conjunction of LRA atoms  $\varphi$  without  $<$

**Output:** satisfiable assignment or unsatisfiable

- 1 transform  $\varphi$  into general form and construct tableau
- 2 fix order on variables and assign 0 to each variable
- 3 if all basic variables satisfy their bounds then return **current (satisfying) assignment**
- 4 let  $x_i \in B$  be variable that violates its bounds
- 5 search for **suitable variable**  $x_j \in N$  for pivoting with  $x_i$
- 6 return **unsatisfiable** if search unsuccessful
- 7 perform **pivot** operation on  $x_i$  and  $x_j$
- 9 **update** assignment
- 10 go to step 3

# Simplex, Visually

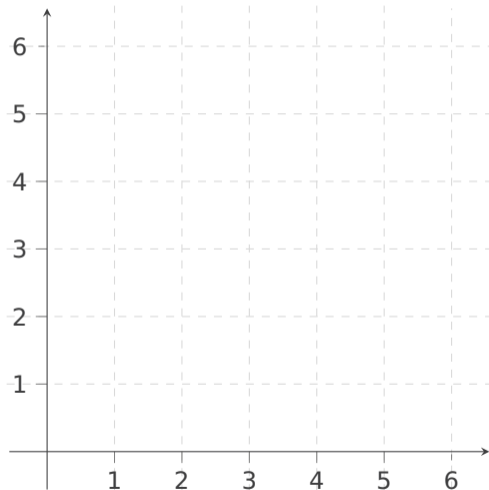
- constraints

$$x - y \geq -1$$

$$y \leq 4$$

$$x + y \geq 6$$

$$3x - y \leq 7$$



# Simplex, Visually

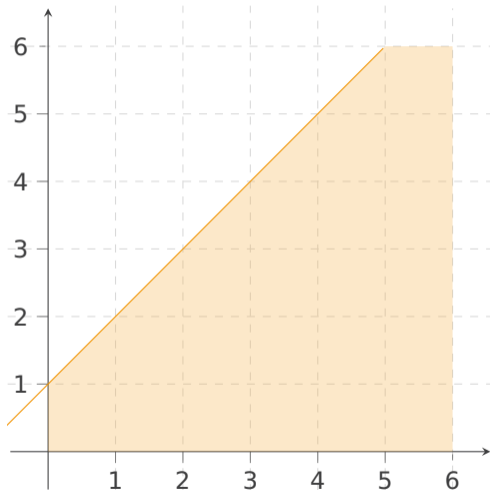
- constraints

$$x - y \geq -1$$

$$y \leq 4$$

$$x + y \geq 6$$

$$3x - y \leq 7$$



# Simplex, Visually

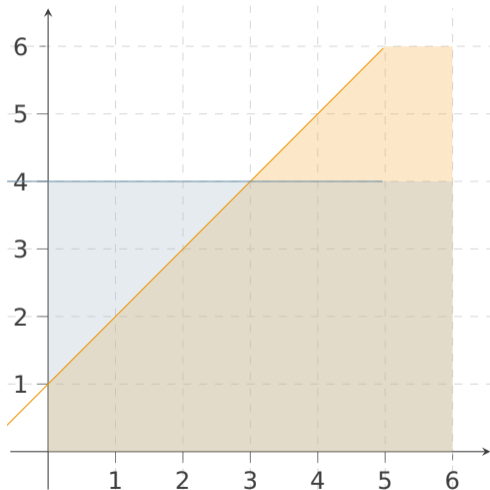
- constraints

$$x - y \geq -1$$

$$y \leq 4$$

$$x + y \geq 6$$

$$3x - y \leq 7$$



# Simplex, Visually

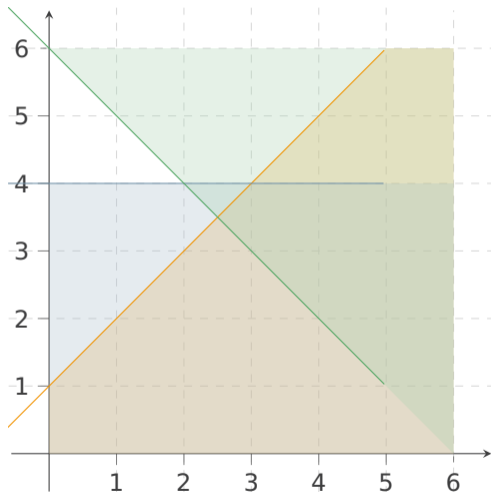
- constraints

$$x - y \geq -1$$

$$y \leq 4$$

$$x + y \geq 6$$

$$3x - y \leq 7$$





# Simplex, Visually

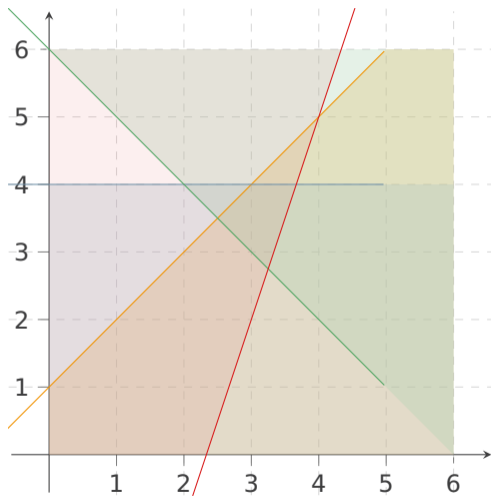
- constraints

$$x - y \geq -1$$

$$y \leq 4$$

$$x + y \geq 6$$

$$3x - y \leq 7$$



# Simplex, Visually

- constraints

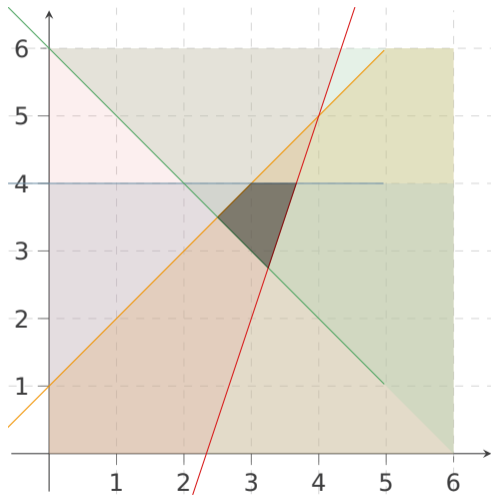
$$x - y \geq -1$$

$$y \leq 4$$

$$x + y \geq 6$$

$$3x - y \leq 7$$

- solution space



# Simplex, Visually

- constraints

$$x - y \geq -1$$

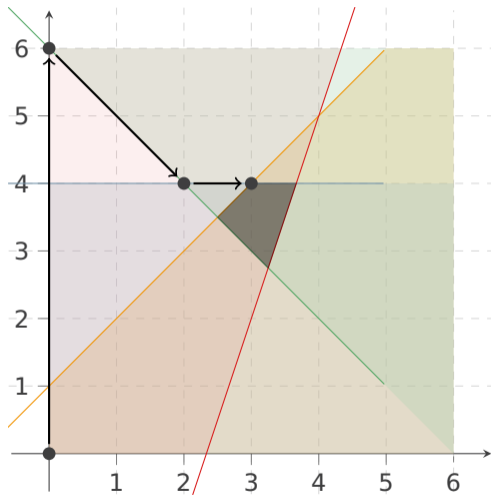
$$y \leq 4$$

$$x + y \geq 6$$

$$3x - y \leq 7$$

- solution space

- Simplex solution search



## Example

	tableau		bounds
	$x$	$y$	
$s_1$	$\begin{pmatrix} -1 & 1 \\ 0 & 1 \\ -1 & -1 \\ 3 & -1 \end{pmatrix}$		$s_1 \leq 1$
$s_2$			$s_2 \leq 4$
$s_3$			$s_3 \leq -6$
$s_4$			$s_4 \leq 7$

## Example

	tableau	bounds	assignment								
	$x$	$y$									
$s_1$	$\begin{pmatrix} -1 & 1 \\ 0 & 1 \\ -1 & -1 \\ 3 & -1 \end{pmatrix}$		$s_1 \leq 1$	$x$	$y$	$s_1$	$s_2$	$s_3$	$s_4$		
$s_2$			$s_2 \leq 4$	0	0	0	0	0	0		
$s_3$			$s_3 \leq -6$								
$s_4$			$s_4 \leq 7$								

## Example

	tableau		bounds
	$x$	$y$	
$s_1$	$\begin{pmatrix} -1 & 1 \\ 0 & 1 \\ -1 & -1 \\ 3 & -1 \end{pmatrix}$		$s_1 \leq 1$
$s_2$			$s_2 \leq 4$
$s_3$			$s_3 \leq -6$
$s_4$			$s_4 \leq 7$

assignment					
$x$	$y$	$s_1$	$s_2$	$s_3$	$s_4$
0	0	0	0	0	0

### 1 Iteration 1

- $s_3$  violates its bounds

## Example

	tableau	bounds	assignment
	$x \quad y$		
$s_1$	$\left( \begin{array}{cc} -1 & 1 \\ 0 & 1 \\ -1 & -1 \\ 3 & -1 \end{array} \right)$	$s_1 \leq 1$	$\begin{array}{cccccc} x & y & s_1 & s_2 & s_3 & s_4 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 \end{array}$
$s_2$		$s_2 \leq 4$	
$s_3$		$s_3 \leq -6$	
$s_4$		$s_4 \leq 7$	

### 1 Iteration 1

- $s_3$  violates its bounds
- decreasing  $s_3$  requires to increase  $x$  or  $y$  (both **suitable** since they have no upper bound)

## Example

	tableau	bounds	assignment
	$x$ $y$		
$s_1$	$\begin{pmatrix} -1 & 1 \\ 0 & 1 \\ -1 & -1 \\ 3 & -1 \end{pmatrix}$	$s_1 \leq 1$	$\begin{array}{cccccc} x & y & s_1 & s_2 & s_3 & s_4 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 \end{array}$
$s_2$		$s_2 \leq 4$	
$s_3$		$s_3 \leq -6$	
$s_4$		$s_4 \leq 7$	

### 1 Iteration 1

- $s_3$  violates its bounds
- decreasing  $s_3$  requires to increase  $x$  or  $y$  (both suitable since they have no upper bound)

- pivot  $s_3$  with  $y$ :

$$s_3 = -x - y \quad \Leftrightarrow \quad y = -x - s_3$$

$$s_1 = -x + y = -x + (-x - s_3) = -2x - s_3$$

$$s_2 = -x - s_3 \qquad s_4 = 4x + s_3$$



## Example

	tableau	bounds	assignment
	$x \quad s_3$		
$s_1$	$\begin{pmatrix} -2 & -1 \\ -1 & -1 \\ -1 & -1 \\ 4 & 1 \end{pmatrix}$	$s_1 \leq 1$	$\begin{array}{cccccc} x & y & s_1 & s_2 & s_3 & s_4 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 \end{array}$
$s_2$		$s_2 \leq 4$	
$y$		$s_3 \leq -6$	
$s_4$		$s_4 \leq 7$	

### 1 Iteration 1

- $s_3$  violates its bounds
- decreasing  $s_3$  requires to increase  $x$  or  $y$  (both suitable since they have no upper bound)

- pivot  $s_3$  with  $y$ :

$$s_3 = -x - y \quad \Leftrightarrow \quad y = -x - s_3$$

$$s_1 = -x + y = -x + (-x - s_3) = -2x - s_3$$

$$s_2 = -x - s_3 \qquad s_4 = 4x + s_3$$

## Example

	tableau	bounds	assignment							
	$x$ $s_3$									
$s_1$	$\begin{pmatrix} -2 & -1 \\ -1 & -1 \\ -1 & -1 \\ 4 & 1 \end{pmatrix}$	$s_1 \leq 1$	$x$	$y$	$s_1$	$s_2$	$s_3$	$s_4$	$0$	$0$
$s_2$		$s_2 \leq 4$	$0$	$0$	$0$	$0$	$0$	$0$	$0$	$0$
$y$		$s_3 \leq -6$								
$s_4$		$s_4 \leq 7$								

### 1 Iteration 1

- $s_3$  violates its bounds
- decreasing  $s_3$  requires to increase  $x$  or  $y$  (both suitable since they have no upper bound)

- pivot  $s_3$  with  $y$ :

$$s_3 = -x - y \quad \Leftrightarrow \quad y = -x - s_3$$

$$s_1 = -x + y = -x + (-x - s_3) = -2x - s_3$$

$$s_2 = -x - s_3 \qquad s_4 = 4x + s_3$$

- update assignment

$$s_3 := -6 \qquad x = 0$$

$$s_1 := -2x - s_3 = 6 \qquad s_2 := 6 \qquad y := 6 \qquad s_4 := -6$$

## Example

	tableau	bounds	assignment							
	$x$ $s_3$									
$s_1$	$\begin{pmatrix} -2 & -1 \\ -1 & -1 \\ -1 & -1 \\ 4 & 1 \end{pmatrix}$	$s_1 \leq 1$	$x$	$y$	$s_1$	$s_2$	$s_3$	$s_4$		
$s_2$		$s_2 \leq 4$	0	6	6	6	-6	-6		
$y$		$s_3 \leq -6$								
$s_4$		$s_4 \leq 7$								

### 1 Iteration 1

- $s_3$  violates its bounds
- decreasing  $s_3$  requires to increase  $x$  or  $y$  (both suitable since they have no upper bound)

- pivot  $s_3$  with  $y$ :

$$s_3 = -x - y \quad \Leftrightarrow \quad y = -x - s_3$$

$$s_1 = -x + y = -x + (-x - s_3) = -2x - s_3$$

$$s_2 = -x - s_3 \qquad s_4 = 4x + s_3$$

- update assignment

$$s_3 := -6 \qquad x = 0$$

$$s_1 := -2x - s_3 = 6 \qquad s_2 := 6 \qquad y := 6 \qquad s_4 := -6$$

## Example

	tableau	bounds
	$x \quad s_3$	
$s_1$	$\left( \begin{array}{cc} -2 & -1 \\ -1 & -1 \\ -1 & -1 \\ 4 & 1 \end{array} \right)$	$s_1 \leq 1$
$s_2$		$s_2 \leq 4$
$y$		$s_3 \leq -6$
$s_4$		$s_4 \leq 7$

assignment					
$x$	$y$	$s_1$	$s_2$	$s_3$	$s_4$
0	6	6	6	-6	-6

2 Iteration 2

## Example

	tableau	bounds
	$x \quad s_3$	
$s_1$	$\left( \begin{array}{cc} -2 & -1 \\ -1 & -1 \\ -1 & -1 \\ 4 & 1 \end{array} \right)$	$s_1 \leq 1$
$s_2$		$s_2 \leq 4$
$y$		$s_3 \leq -6$
$s_4$		$s_4 \leq 7$

assignment					
$x$	$y$	$s_1$	$s_2$	$s_3$	$s_4$
0	6	6	6	-6	-6

### 2 Iteration 2

- $s_2$  violates its bounds

## Example

	tableau	bounds	assignment
	$x \quad s_3$		
$s_1$	$\left( \begin{array}{cc} -2 & -1 \\ -1 & -1 \\ -1 & -1 \\ 4 & 1 \end{array} \right)$	$s_1 \leq 1$	$\begin{array}{cccccc} x & y & s_1 & s_2 & s_3 & s_4 \\ \hline 0 & 6 & 6 & 6 & -6 & -6 \end{array}$
$s_2$		$s_2 \leq 4$	
$y$		$s_3 \leq -6$	
$s_4$		$s_4 \leq 7$	

### 2 Iteration 2

- $s_2$  violates its bounds
- decreasing  $s_2$  requires increase of  $x$  or  $s_3$ :  $x$  is **suitable**, but  $s_3$  is not

## Example

	tableau	bounds	assignment							
	$x$	$s_3$								
$s_1$	$\begin{pmatrix} -2 & -1 \\ -1 & -1 \\ -1 & -1 \\ 4 & 1 \end{pmatrix}$	$s_1 \leq 1$	$x$	$y$	$s_1$	$s_2$	$s_3$	$s_4$		
$s_2$		$s_2 \leq 4$	0	6	6	6	-6	-6		
$y$		$s_3 \leq -6$								
$s_4$		$s_4 \leq 7$								

### 2 Iteration 2

- $s_2$  violates its bounds
- decreasing  $s_2$  requires increase of  $x$  or  $s_3$ :  $x$  is suitable, but  $s_3$  is not
- pivot  $s_2$  with  $x$ :

$$\begin{aligned} x &= -s_2 - s_3 & s_1 &= -2x - s_3 = 2s_2 + s_3 \\ y &= -x - s_3 = s_2 & s_4 &= 4x + s_3 = -4s_2 - 3s_3 \end{aligned}$$

## Example

	tableau	bounds	assignment
	$s_2 \quad s_3$		
$s_1$	$\begin{pmatrix} 2 & 1 \\ -1 & -1 \\ 1 & 0 \\ -4 & -3 \end{pmatrix}$	$s_1 \leq 1$	$x \quad y \quad s_1 \quad s_2 \quad s_3 \quad s_4$
$x$		$s_2 \leq 4$	<hr/>
$y$		$s_3 \leq -6$	$0 \quad 6 \quad 6 \quad 6 \quad -6 \quad -6$
$s_4$		$s_4 \leq 7$	

### 2 Iteration 2

- $s_2$  violates its bounds
- decreasing  $s_2$  requires increase of  $x$  or  $s_3$ :  $x$  is suitable, but  $s_3$  is not

- pivot  $s_2$  with  $x$ :

$$\begin{aligned} x &= -s_2 - s_3 & s_1 &= -2x - s_3 = 2s_2 + s_3 \\ y &= -x - s_3 = s_2 & s_4 &= 4x + s_3 = -4s_2 - 3s_3 \end{aligned}$$



## Example

	tableau	bounds	assignment
	$s_2 \quad s_3$		
$s_1$	$\begin{pmatrix} 2 & 1 \\ -1 & -1 \\ 1 & 0 \\ -4 & -3 \end{pmatrix}$	$s_1 \leq 1$	$x \quad y \quad s_1 \quad s_2 \quad s_3 \quad s_4$
$x$		$s_2 \leq 4$	$0 \quad 6 \quad 6 \quad 6 \quad -6 \quad -6$
$y$		$s_3 \leq -6$	
$s_4$		$s_4 \leq 7$	

### 2 Iteration 2

- $s_2$  violates its bounds
- decreasing  $s_2$  requires increase of  $x$  or  $s_3$ :  $x$  is suitable, but  $s_3$  is not

- pivot  $s_2$  with  $x$ :

$$\begin{aligned} x &= -s_2 - s_3 & s_1 &= -2x - s_3 = 2s_2 + s_3 \\ y &= -x - s_3 = s_2 & s_4 &= 4x + s_3 = -4s_2 - 3s_3 \end{aligned}$$

- update assignment

$$\begin{aligned} s_2 &:= 4 & s_3 &= -6 \\ s_1 &:= 2 & x &:= 2 & y &:= 4 & s_4 &:= 2 \end{aligned}$$

## Example

	tableau	bounds	assignment							
	$s_2$ $s_3$									
$s_1$	$\left( \begin{array}{cc} 2 & 1 \\ -1 & -1 \\ 1 & 0 \\ -4 & -3 \end{array} \right)$	$s_1 \leq 1$	$x$	$y$	$s_1$	$s_2$	$s_3$	$s_4$		
$x$		$s_2 \leq 4$	2	4	2	4	-6	2		
$y$		$s_3 \leq -6$								
$s_4$		$s_4 \leq 7$								

### 2 Iteration 2

- $s_2$  violates its bounds
- decreasing  $s_2$  requires increase of  $x$  or  $s_3$ :  $x$  is suitable, but  $s_3$  is not

- pivot  $s_2$  with  $x$ :

$$\begin{aligned}
 x &= -s_2 - s_3 & s_1 &= -2x - s_3 = 2s_2 + s_3 \\
 y &= -x - s_3 = s_2 & s_4 &= 4x + s_3 = -4s_2 - 3s_3
 \end{aligned}$$

- update assignment

$$\begin{aligned}
 s_2 &:= 4 & s_3 &= -6 \\
 s_1 &:= 2 & x &:= 2 & y &:= 4 & s_4 &:= 2
 \end{aligned}$$

## Example

	tableau	bounds	assignment
	$s_2 \quad s_3$		
$s_1$	$\begin{pmatrix} 2 & 1 \\ -1 & -1 \\ 1 & 0 \\ -4 & -3 \end{pmatrix}$	$s_1 \leq 1$	$x \quad y \quad s_1 \quad s_2 \quad s_3 \quad s_4$
$x$		$s_2 \leq 4$	<hr/>
$y$		$s_3 \leq -6$	2    4    2    4    -6    2
$s_4$		$s_4 \leq 7$	

### 3 Iteration 3

- $s_1$  violates its bounds

## Example

	tableau	bounds	assignment
	$s_2 \quad s_3$		
$s_1$	$\begin{pmatrix} 2 & 1 \\ -1 & -1 \\ 1 & 0 \\ -4 & -3 \end{pmatrix}$	$s_1 \leq 1$	$x \quad y \quad s_1 \quad s_2 \quad s_3 \quad s_4$
$x$		$s_2 \leq 4$	<hr/>
$y$		$s_3 \leq -6$	2    4    2    4    -6    2
$s_4$		$s_4 \leq 7$	

### 3 Iteration 3

- $s_1$  violates its bounds
- decreasing  $s_1$  requires to decrease  $s_2$  or  $s_3$  (both suitable since they have no lower bound)

## Example

	tableau	bounds	assignment
	$s_2 \quad s_3$		
$s_1$	$\begin{pmatrix} 2 & 1 \\ -1 & -1 \\ 1 & 0 \\ -4 & -3 \end{pmatrix}$	$s_1 \leq 1$	$x \quad y \quad s_1 \quad s_2 \quad s_3 \quad s_4$
$x$		$s_2 \leq 4$	<hr/>
$y$		$s_3 \leq -6$	2    4    2    4    -6    2
$s_4$		$s_4 \leq 7$	

### 3 Iteration 3

- $s_1$  violates its bounds
- decreasing  $s_1$  requires to decrease  $s_2$  or  $s_3$  (both suitable since they have no lower bound)
- pivot  $s_1$  with  $s_3$ :

$$s_3 = s_1 - 2s_2$$

$$y = s_2$$

$$x = -s_1 + s_2$$

$$s_4 = -3s_1 + 2s_2$$

## Example

	tableau	bounds	assignment
	$s_2 \quad s_1$		
$s_3$	$\begin{pmatrix} -2 & 1 \\ 1 & -1 \\ 1 & 0 \\ 2 & -3 \end{pmatrix}$	$s_1 \leq 1$	$\begin{array}{cccccc} x & y & s_1 & s_2 & s_3 & s_4 \\ \hline 2 & 4 & 2 & 4 & -6 & 2 \end{array}$
$x$		$s_2 \leq 4$	
$y$		$s_3 \leq -6$	
$s_4$		$s_4 \leq 7$	

### 3 Iteration 3

- $s_1$  violates its bounds
- decreasing  $s_1$  requires to decrease  $s_2$  or  $s_3$  (both suitable since they have no lower bound)
- pivot  $s_1$  with  $s_3$ :

$$s_3 = s_1 - 2s_2$$

$$y = s_2$$

$$x = -s_1 + s_2$$

$$s_4 = -3s_1 + 2s_2$$

## Example

	tableau	bounds	assignment
	$  \begin{array}{c}  s_2 \quad s_1 \\  s_3 \left( \begin{array}{cc} -2 & 1 \\ 1 & -1 \\ 1 & 0 \\ 2 & -3 \end{array} \right)  \end{array}  $	$  \begin{array}{l}  s_1 \leq 1 \\  s_2 \leq 4 \\  s_3 \leq -6 \\  s_4 \leq 7  \end{array}  $	$  \begin{array}{cccccc}  x & y & s_1 & s_2 & s_3 & s_4 \\  \hline  2 & 4 & 2 & 4 & -6 & 2  \end{array}  $

### 3 Iteration 3

- $s_1$  violates its bounds
- decreasing  $s_1$  requires to decrease  $s_2$  or  $s_3$  (both suitable since they have no lower bound)

- pivot  $s_1$  with  $s_3$ :

$$s_3 = s_1 - 2s_2$$

$$x = -s_1 + s_2$$

$$y = s_2$$

$$s_4 = -3s_1 + 2s_2$$

- update assignment

$$s_1 := 1$$

$$s_2 = 4$$

$$s_3 := -7$$

$$x := 3$$

$$y := 4$$

$$s_4 := 5$$

## Example

	tableau	bounds	assignment							
	$s_2$ $s_1$									
$s_3$	$\begin{pmatrix} -2 & 1 \\ 1 & -1 \\ 1 & 0 \\ 2 & -3 \end{pmatrix}$	$s_1 \leq 1$	$x$	$y$	$s_1$	$s_2$	$s_3$	$s_4$		
$x$		$s_2 \leq 4$	3	4	1	4	-7	5		
$y$		$s_3 \leq -6$								
$s_4$		$s_4 \leq 7$								

### 3 Iteration 3

- $s_1$  violates its bounds
- decreasing  $s_1$  requires to decrease  $s_2$  or  $s_3$  (both suitable since they have no lower bound)

- pivot  $s_1$  with  $s_3$ :

$$s_3 = s_1 - 2s_2$$

$$x = -s_1 + s_2$$

$$y = s_2$$

$$s_4 = -3s_1 + 2s_2$$

- update assignment

$$s_1 := 1$$

$$s_2 = 4$$

$$s_3 := -7$$

$$x := 3$$

$$y := 4$$

$$s_4 := 5$$



## Example

$$\begin{array}{l} s_3 \\ x \\ y \\ s_4 \end{array} \begin{array}{cc} \text{tableau} & \\ s_2 & s_1 \\ \left( \begin{array}{cc} -2 & 1 \\ 1 & -1 \\ 1 & 0 \\ 2 & -3 \end{array} \right) \end{array} \begin{array}{l} \text{bounds} \\ s_1 \leq 1 \\ s_2 \leq 4 \\ s_3 \leq -6 \\ s_4 \leq 7 \end{array}$$

$$\begin{array}{cccccc} \text{assignment} & & & & & \\ x & y & s_1 & s_2 & s_3 & s_4 \\ \hline 3 & 4 & 1 & 4 & -7 & 5 \end{array}$$

### 4 Iteration 4

- all variables satisfy their bounds

$$A\vec{x}_N = \vec{x}_B \quad (1)$$

$$-\infty \leq l_j \leq x_j \leq u_j \leq +\infty \quad (2)$$

### Invariant

- (1) is satisfied and (2) holds for all nonbasic variables

# Simplex Algorithm as $T$ -solver

$$A\vec{x}_N = \vec{x}_B \quad (1)$$

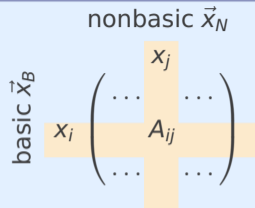
$$-\infty \leq l_i \leq x_i \leq u_i \leq +\infty \quad (2)$$

## Invariant

- (1) is satisfied and (2) holds for all nonbasic variables

## Pivoting

- **swap** basic  $x_i$  and nonbasic  $x_j$ , so  $i \in B$  and  $j \in N$



(\*)

# Simplex Algorithm as $T$ -solver

$$A\vec{x}_N = \vec{x}_B \quad (1)$$

$$-\infty \leq l_i \leq x_i \leq u_i \leq +\infty \quad (2)$$

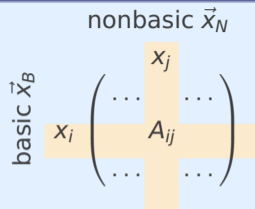
## Invariant

- (1) is satisfied and (2) holds for all nonbasic variables

## Pivoting

- **swap** basic  $x_i$  and nonbasic  $x_j$ , so  $i \in B$  and  $j \in N$

$$x_i = \sum_{k \in N} A_{ik} x_k \quad (*)$$



# Simplex Algorithm as $T$ -solver

$$A\vec{x}_N = \vec{x}_B \quad (1)$$

$$-\infty \leq l_i \leq x_i \leq u_i \leq +\infty \quad (2)$$

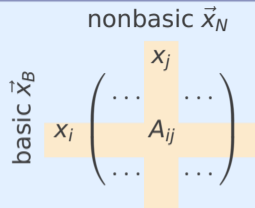
## Invariant

- (1) is satisfied and (2) holds for all nonbasic variables

## Pivoting

- **swap** basic  $x_i$  and nonbasic  $x_j$ , so  $i \in B$  and  $j \in N$

$$x_i = \sum_{k \in N} A_{ik} x_k \quad \implies \quad x_j = \frac{1}{A_{ij}} \left( x_i - \sum_{k \in N - \{j\}} A_{ik} x_k \right) \quad (*)$$



# Simplex Algorithm as $T$ -solver

$$A\vec{x}_N = \vec{x}_B \quad (1)$$

$$-\infty \leq l_i \leq x_i \leq u_i \leq +\infty \quad (2)$$

## Invariant

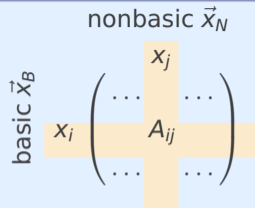
- (1) is satisfied and (2) holds for all nonbasic variables

## Pivoting

- swap basic  $x_i$  and nonbasic  $x_j$ , so  $i \in B$  and  $j \in N$

$$x_i = \sum_{k \in N} A_{ik} x_k \quad \implies \quad x_j = \frac{1}{A_{ij}} \left( x_i - \sum_{k \in N - \{j\}} A_{ik} x_k \right) \quad (*)$$

- new tableau  $A'$  consists of (\*) and  $A_{B-\{i\}} \vec{x}_N = \vec{x}_{B-\{i\}}$  with (\*) substituted



# Simplex Algorithm as $T$ -solver

$$A\vec{x}_N = \vec{x}_B \quad (1)$$

$$-\infty \leq l_i \leq x_i \leq u_i \leq +\infty \quad (2)$$

## Invariant

- (1) is satisfied and (2) holds for all nonbasic variables

## Pivoting

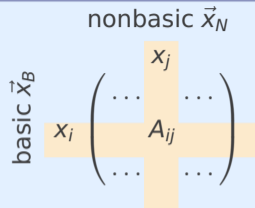
- swap basic  $x_i$  and nonbasic  $x_j$ , so  $i \in B$  and  $j \in N$

$$x_i = \sum_{k \in N} A_{ik} x_k \quad \implies \quad x_j = \frac{1}{A_{ij}} \left( x_i - \sum_{k \in N - \{j\}} A_{ik} x_k \right) \quad (*)$$

- new tableau  $A'$  consists of (\*) and  $A_{B-\{i\}} \vec{x}_N = \vec{x}_{B-\{i\}}$  with (\*) substituted

## Update

- assignment of  $x_i$  is updated to previously violated bound  $l_i$  or  $u_i$ ,



# Simplex Algorithm as $T$ -solver

$$A\vec{x}_N = \vec{x}_B \quad (1)$$

$$-\infty \leq l_i \leq x_i \leq u_i \leq +\infty \quad (2)$$

## Invariant

- (1) is satisfied and (2) holds for all nonbasic variables

## Pivoting

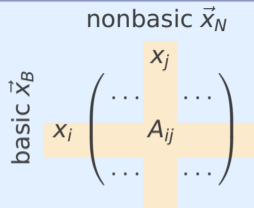
- swap basic  $x_i$  and nonbasic  $x_j$ , so  $i \in B$  and  $j \in N$

$$x_i = \sum_{k \in N} A_{ik} x_k \quad \implies \quad x_j = \frac{1}{A_{ij}} \left( x_i - \sum_{k \in N - \{j\}} A_{ik} x_k \right) \quad (*)$$

- new tableau  $A'$  consists of (\*) and  $A_{B-\{i\}} \vec{x}_N = \vec{x}_{B-\{i\}}$  with (\*) substituted

## Update

- assignment of  $x_i$  is updated to previously violated bound  $l_i$  or  $u_i$ ,
- assignment of  $x_k$  is recomputed using  $A'$  for all  $k \in B - \{i\} \cup \{j\}$





## Simplex Algorithm as $T$ -solver

$$A\vec{x}_N = \vec{x}_B \quad (1)$$

$$\forall i \in N. -\infty \leq l_i \leq x_i \leq u_i \leq +\infty \quad (2)$$

### Suitability

- basic variable  $x_i$  violates lower or upper bound

## Simplex Algorithm as $T$ -solver

$$A\vec{x}_N = \vec{x}_B \quad (1)$$

$$\forall i \in N. -\infty \leq l_i \leq x_i \leq u_i \leq +\infty \quad (2)$$

### Suitability

- basic variable  $x_i$  violates lower or upper bound
- pick nonbasic variable  $x_j$  such that
  - if  $u_j < l_j$ : problem is trivially unsatisfiable and no suitable  $x_j$  exists

## Simplex Algorithm as $T$ -solver

$$A\vec{x}_N = \vec{x}_B \quad (1)$$

$$\forall i \in N. -\infty \leq l_i \leq x_i \leq u_i \leq +\infty \quad (2)$$

### Suitability

- basic variable  $x_i$  violates lower or upper bound
- pick nonbasic variable  $x_j$  such that
  - if  $u_i < l_i$ : problem is trivially unsatisfiable and no suitable  $x_j$  exists
  - if  $x_i < l_i$ :  $A_{ij} > 0$  and  $x_j < u_j$  or  $A_{ij} < 0$  and  $x_j > l_j$

## Simplex Algorithm as $T$ -solver

$$A\vec{x}_N = \vec{x}_B \quad (1)$$

$$\forall i \in N. -\infty \leq l_i \leq x_i \leq u_i \leq +\infty \quad (2)$$

### Suitability

- basic variable  $x_i$  violates lower or upper bound
- pick nonbasic variable  $x_j$  such that
  - if  $u_i < l_i$ : problem is trivially unsatisfiable and no suitable  $x_j$  exists
  - if  $x_i < l_i$ :  $A_{ij} > 0$  and  $x_j < u_j$  or  $A_{ij} < 0$  and  $x_j > l_j$
  - if  $x_i > u_i$ :  $A_{ij} > 0$  and  $x_j > l_j$  or  $A_{ij} < 0$  and  $x_j < u_j$

## Simplex Algorithm as $T$ -solver

$$A\vec{x}_N = \vec{x}_B \quad (1)$$

$$\forall i \in N. -\infty \leq l_i \leq x_i \leq u_i \leq +\infty \quad (2)$$

### Suitability

- basic variable  $x_i$  violates lower or upper bound
- pick nonbasic variable  $x_j$  such that
  - if  $u_i < l_i$ : problem is trivially unsatisfiable and no suitable  $x_j$  exists
  - if  $x_i < l_i$ :  $A_{ij} > 0$  and  $x_j < u_j$  or  $A_{ij} < 0$  and  $x_j > l_j$
  - if  $x_i > u_i$ :  $A_{ij} > 0$  and  $x_j > l_j$  or  $A_{ij} < 0$  and  $x_j < u_j$

### Observation

- problem is unsatisfiable if no suitable pivot exists

# Outline

1. Summary of Previous Lecture
2. Difference Logic
3. Simplex Algorithm
- 4. Support of Strict Inequalities**
5. Further Reading

## Motivation

strict inequalities naturally arise, e.g., as negated non-strict inequalities in DPLL( $T$ )

$$\neg(x + 3 \leq 5y) \leftrightarrow x + 3 > 5y$$

## Motivation

strict inequalities naturally arise, e.g., as negated non-strict inequalities in DPLL( $T$ )

$$\neg(x + 3 \leq 5y) \leftrightarrow x + 3 > 5y$$

## How to Treat Strict Inequalities

replace in conjunction of inequalities  $C$  every strict inequality

$$a_1x_1 + \dots + a_nx_n > b$$

$$a_1x_1 + \dots + a_nx_n < b$$

by non-strict inequality

$$a_1x_1 + \dots + a_nx_n \geq b + \delta$$

$$a_1x_1 + \dots + a_nx_n \leq b - \delta$$

to obtain constraints  $C_\delta$  in LRA without  $>$  and  $<$ , and treat  $\delta$  symbolically during simplex algorithm  
( $\delta$  represents small positive rational number)



## Motivation

strict inequalities naturally arise, e.g., as negated non-strict inequalities in DPLL( $T$ )

$$\neg(x + 3 \leq 5y) \quad \leftrightarrow \quad x + 3 > 5y$$

## How to Treat Strict Inequalities

replace in conjunction of inequalities  $C$  every strict inequality

$$a_1x_1 + \cdots + a_nx_n > b$$

$$a_1x_1 + \cdots + a_nx_n < b$$

by non-strict inequality

$$a_1x_1 + \cdots + a_nx_n \geq b + \delta$$

$$a_1x_1 + \cdots + a_nx_n \leq b - \delta$$

to obtain constraints  $C_\delta$  in LRA without  $>$  and  $<$ , and treat  $\delta$  symbolically during simplex algorithm  
( $\delta$  represents small positive rational number)

## Lemma

$C$  is satisfiable  $\iff \exists$  rational number  $\delta > 0$  such that  $C_\delta$  is satisfiable

## Symbolical computation with $\delta$

- $\delta$  represents small positive rational number, i.e., smaller than every concrete rational number that occurs during the computations of the simplex algorithm

## Symbolical computation with $\delta$

- $\delta$  represents small positive rational number, i.e., smaller than every concrete rational number that occurs during the computations of the simplex algorithm
- treat  $\delta$  symbolically:  $\mathbb{Q}_\delta = \{(c, k) \mid c, k \in \mathbb{Q}\}$  with  $(c, k)$  representing  $c + k\delta$

## Symbolical computation with $\delta$

- $\delta$  represents small positive rational number, i.e., smaller than every concrete rational number that occurs during the computations of the simplex algorithm
- treat  $\delta$  symbolically:  $\mathbb{Q}_\delta = \{(c, k) \mid c, k \in \mathbb{Q}\}$  with  $(c, k)$  representing  $c + k\delta$
- operations for all  $a, c_1, k_1, c_2, k_2 \in \mathbb{Q}$

## Symbolical computation with $\delta$

- $\delta$  represents small positive rational number, i.e., smaller than every concrete rational number that occurs during the computations of the simplex algorithm
- treat  $\delta$  symbolically:  $\mathbb{Q}_\delta = \{(c, k) \mid c, k \in \mathbb{Q}\}$  with  $(c, k)$  representing  $c + k\delta$
- operations for all  $a, c_1, k_1, c_2, k_2 \in \mathbb{Q}$ 
  - addition:  $(c_1, k_1) + (c_2, k_2) = (c_1 + c_2, k_1 + k_2)$

## Symbolical computation with $\delta$

- $\delta$  represents small positive rational number, i.e., smaller than every concrete rational number that occurs during the computations of the simplex algorithm
- treat  $\delta$  symbolically:  $\mathbb{Q}_\delta = \{(c, k) \mid c, k \in \mathbb{Q}\}$  with  $(c, k)$  representing  $c + k\delta$
- operations for all  $a, c_1, k_1, c_2, k_2 \in \mathbb{Q}$ 
  - addition:  $(c_1, k_1) + (c_2, k_2) = (c_1 + c_2, k_1 + k_2)$
  - multiplication:  $a \cdot (c_1, k_1) = (ac_1, ak_1)$

## Symbolical computation with $\delta$

- $\delta$  represents small positive rational number, i.e., smaller than every concrete rational number that occurs during the computations of the simplex algorithm
- treat  $\delta$  symbolically:  $\mathbb{Q}_\delta = \{(c, k) \mid c, k \in \mathbb{Q}\}$  with  $(c, k)$  representing  $c + k\delta$
- operations for all  $a, c_1, k_1, c_2, k_2 \in \mathbb{Q}$ 
  - addition:  $(c_1, k_1) + (c_2, k_2) = (c_1 + c_2, k_1 + k_2)$
  - multiplication:  $a \cdot (c_1, k_1) = (ac_1, ak_1)$
  - equality:  $(c_1, k_1) = (c_2, k_2) \leftrightarrow c_1 = c_2 \wedge k_1 = k_2$

## Symbolical computation with $\delta$

- $\delta$  represents small positive rational number, i.e., smaller than every concrete rational number that occurs during the computations of the simplex algorithm
- treat  $\delta$  symbolically:  $\mathbb{Q}_\delta = \{(c, k) \mid c, k \in \mathbb{Q}\}$  with  $(c, k)$  representing  $c + k\delta$
- operations for all  $a, c_1, k_1, c_2, k_2 \in \mathbb{Q}$ 
  - addition:  $(c_1, k_1) + (c_2, k_2) = (c_1 + c_2, k_1 + k_2)$
  - multiplication:  $a \cdot (c_1, k_1) = (ac_1, ak_1)$
  - equality:  $(c_1, k_1) = (c_2, k_2) \Leftrightarrow c_1 = c_2 \wedge k_1 = k_2$
  - comparison:  $(c_1, k_1) < (c_2, k_2) \Leftrightarrow c_1 < c_2 \vee c_1 = c_2 \wedge k_1 < k_2$   
 $(c_1, k_1) \leq (c_2, k_2) \Leftrightarrow c_1 < c_2 \vee c_1 = c_2 \wedge k_1 \leq k_2$



## Symbolical computation with $\delta$

- $\delta$  represents small positive rational number, i.e., smaller than every concrete rational number that occurs during the computations of the simplex algorithm
- treat  $\delta$  symbolically:  $\mathbb{Q}_\delta = \{(c, k) \mid c, k \in \mathbb{Q}\}$  with  $(c, k)$  representing  $c + k\delta$
- operations for all  $a, c_1, k_1, c_2, k_2 \in \mathbb{Q}$ 
  - addition:  $(c_1, k_1) + (c_2, k_2) = (c_1 + c_2, k_1 + k_2)$
  - multiplication:  $a \cdot (c_1, k_1) = (ac_1, ak_1)$
  - equality:  $(c_1, k_1) = (c_2, k_2) \Leftrightarrow c_1 = c_2 \wedge k_1 = k_2$
  - comparison:  $(c_1, k_1) < (c_2, k_2) \Leftrightarrow c_1 < c_2 \vee c_1 = c_2 \wedge k_1 < k_2$   
 $(c_1, k_1) \leq (c_2, k_2) \Leftrightarrow c_1 < c_2 \vee c_1 = c_2 \wedge k_1 \leq k_2$
- multiplication of two  $\mathbb{Q}_\delta$ -numbers is not defined, but also not required for the simplex algorithm

## Symbolical computation with $\delta$

- $\delta$  represents small positive rational number, i.e., smaller than every concrete rational number that occurs during the computations of the simplex algorithm
- treat  $\delta$  symbolically:  $\mathbb{Q}_\delta = \{(c, k) \mid c, k \in \mathbb{Q}\}$  with  $(c, k)$  representing  $c + k\delta$
- operations for all  $a, c_1, k_1, c_2, k_2 \in \mathbb{Q}$ 
  - addition:  $(c_1, k_1) + (c_2, k_2) = (c_1 + c_2, k_1 + k_2)$
  - multiplication:  $a \cdot (c_1, k_1) = (ac_1, ak_1)$
  - equality:  $(c_1, k_1) = (c_2, k_2) \Leftrightarrow c_1 = c_2 \wedge k_1 = k_2$
  - comparison:  $(c_1, k_1) < (c_2, k_2) \Leftrightarrow c_1 < c_2 \vee c_1 = c_2 \wedge k_1 < k_2$   
 $(c_1, k_1) \leq (c_2, k_2) \Leftrightarrow c_1 < c_2 \vee c_1 = c_2 \wedge k_1 \leq k_2$
- multiplication of two  $\mathbb{Q}_\delta$ -numbers is not defined, but also not required for the simplex algorithm
  - coefficients in the tableau stay in  $\mathbb{Q}$

## Symbolical computation with $\delta$

- $\delta$  represents small positive rational number, i.e., smaller than every concrete rational number that occurs during the computations of the simplex algorithm
- treat  $\delta$  symbolically:  $\mathbb{Q}_\delta = \{(c, k) \mid c, k \in \mathbb{Q}\}$  with  $(c, k)$  representing  $c + k\delta$
- operations for all  $a, c_1, k_1, c_2, k_2 \in \mathbb{Q}$ 
  - addition:  $(c_1, k_1) + (c_2, k_2) = (c_1 + c_2, k_1 + k_2)$
  - multiplication:  $a \cdot (c_1, k_1) = (ac_1, ak_1)$
  - equality:  $(c_1, k_1) = (c_2, k_2) \Leftrightarrow c_1 = c_2 \wedge k_1 = k_2$
  - comparison:  $(c_1, k_1) < (c_2, k_2) \Leftrightarrow c_1 < c_2 \vee c_1 = c_2 \wedge k_1 < k_2$   
 $(c_1, k_1) \leq (c_2, k_2) \Leftrightarrow c_1 < c_2 \vee c_1 = c_2 \wedge k_1 \leq k_2$
- multiplication of two  $\mathbb{Q}_\delta$ -numbers is not defined, but also not required for the simplex algorithm
  - coefficients in the tableau stay in  $\mathbb{Q}$
  - only bounds and assignment require  $\mathbb{Q}_\delta$

## Example

	tableau	constraints	assignment				
	$x$ $y$		$x$	$y$	$s_1$	$s_2$	$s_3$
$s_1$	$\left( \begin{array}{cc} 1 & 1 \\ 2 & -1 \\ -1 & 2 \end{array} \right)$	$2 < s_1$	<hr/>	<hr/>	<hr/>	<hr/>	<hr/>
$s_2$		$0 \leq s_2$	0	0	0	0	0
$s_3$		$1 \leq s_3$					

## Example

	tableau	constraints	assignment				
	$x$ $y$		$x$	$y$	$s_1$	$s_2$	$s_3$
$s_1$	$\left( \begin{array}{cc} 1 & 1 \\ 2 & -1 \\ -1 & 2 \end{array} \right)$	$(2, 1) \leq s_1$	<hr/>	<hr/>	<hr/>	<hr/>	<hr/>
$s_2$		$(0, 0) \leq s_2$	$(0, 0)$	$(0, 0)$	$(0, 0)$	$(0, 0)$	$(0, 0)$
$s_3$		$(1, 0) \leq s_3$					

## Example

	tableau	constraints	assignment				
	$x$ $y$		$x$	$y$	$s_1$	$s_2$	$s_3$
$s_1$	$\left( \begin{array}{cc} 1 & 1 \end{array} \right)$	$(2, 1) \leq s_1$					
$s_2$	$\left( \begin{array}{cc} 2 & -1 \end{array} \right)$	$(0, 0) \leq s_2$	$(0, 0)$	$(0, 0)$	$(0, 0)$	$(0, 0)$	$(0, 0)$
$s_3$	$\left( \begin{array}{cc} -1 & 2 \end{array} \right)$	$(1, 0) \leq s_3$					

- pivot  $s_1$  with  $x$

## Example

	tableau	constraints	assignment				
	$s_1$ $y$		$x$	$y$	$s_1$	$s_2$	$s_3$
$x$	$\begin{pmatrix} 1 & -1 \\ 2 & -3 \\ -1 & 3 \end{pmatrix}$	$(2, 1) \leq s_1$					
$s_2$		$(0, 0) \leq s_2$	$(0, 0)$	$(0, 0)$	$(0, 0)$	$(0, 0)$	$(0, 0)$
$s_3$		$(1, 0) \leq s_3$					

- pivot  $s_1$  with  $x \implies$   
 $x = s_1 - y$   
 $s_2 = 2(s_1 - y) - y = 2s_1 - 3y$   
 $s_3 = -(s_1 - y) + 2y = -s_1 + 3y$

## Example

	tableau	constraints	assignment				
	$s_1$ $y$		$x$	$y$	$s_1$	$s_2$	$s_3$
$x$	$\begin{pmatrix} 1 & -1 \\ 2 & -3 \\ -1 & 3 \end{pmatrix}$	$(2, 1) \leq s_1$					
$s_2$		$(0, 0) \leq s_2$	$(0, 0)$	$(0, 0)$	$(2, 1)$	$(0, 0)$	$(0, 0)$
$s_3$		$(1, 0) \leq s_3$					

- pivot  $s_1$  with  $x \implies$   
 $x = s_1 - y$   
 $s_2 = 2(s_1 - y) - y = 2s_1 - 3y$   
 $s_3 = -(s_1 - y) + 2y = -s_1 + 3y$



## Example

	tableau	constraints	assignment				
	$s_1$ $y$		$x$	$y$	$s_1$	$s_2$	$s_3$
$x$	$\begin{pmatrix} 1 & -1 \\ 2 & -3 \\ -1 & 3 \end{pmatrix}$	$(2, 1) \leq s_1$					
$s_2$		$(0, 0) \leq s_2$	$(2, 1)$	$(0, 0)$	$(2, 1)$	$(4, 2)$	$(-2, -1)$
$s_3$		$(1, 0) \leq s_3$					

- pivot  $s_1$  with  $x \implies$   
 $x = s_1 - y$   
 $s_2 = 2(s_1 - y) - y = 2s_1 - 3y$   
 $s_3 = -(s_1 - y) + 2y = -s_1 + 3y$

# Outline

1. Summary of Previous Lecture
2. Difference Logic
3. Simplex Algorithm
4. Support of Strict Inequalities
- 5. Further Reading**

## Kröning and Strichmann

- Sections 5.1, 5.2 and 5.7

## Further Reading



Bruno Dutertre and Leonardo de Moura.

A Fast Linear-Arithmetic Solver for DPLL(T)

In Proc. of International Conference on Computer Aided Verification, pp. 81–94, 2006.

## Important Concepts

- basic and nonbasic variables
- Bellman–Ford algorithm
- difference logic
- linear arithmetic (LRA and LIA)
- negative cycles
- pivoting
- $\mathbb{Q}_\delta$
- simplex algorithm
- suitable pair of variables
- tableau