Satisfiability modulo theories Part 2 Neural Theory Solvers

Verifying cyberphysical systems

Sayan Mitra

mitras@illinois.edu

Today

- SMT
- Decision procedure for Linear Real Arithmetic

  Simplex Algorithm [Dantzig 1947]
- Next week: Verification of Neural Networks

  Reluplex [Katz et al 2017]

References

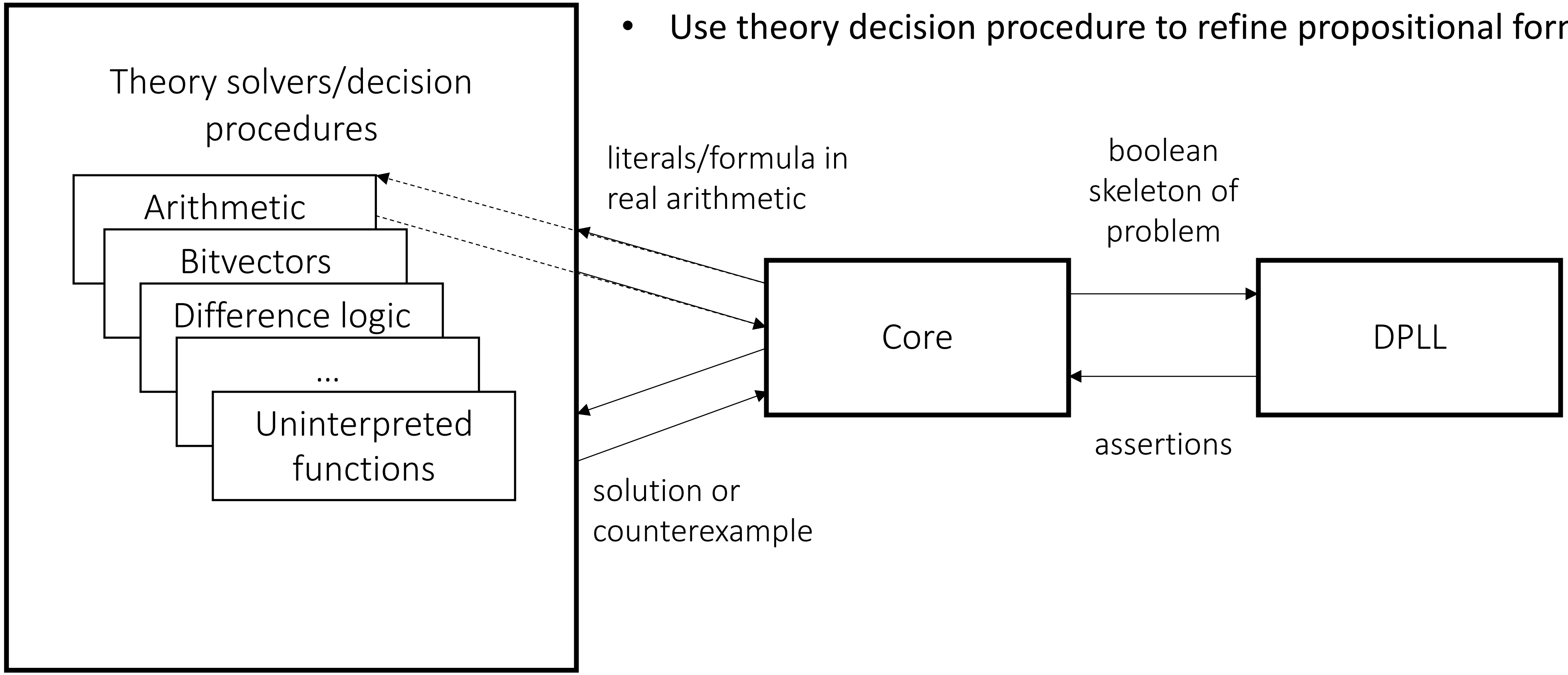
- Lectures on SMT from Clark Barrett
- Book: Introduction to Neural Network Verification by Aws Albarghouthi
- Book: Decision Procedures by Daniel Kroening and Ofer Strichman

SMT

$$\phi \equiv g(a) = c \wedge f\big(g(a)\big) \neq f(c) \vee g(a) = d \wedge c \neq d$$

Several approaches, lazy approach:

- Abstract ϕ to propositional form
- Feed to DPLL
- Use theory decision procedure to refine propositional formula a guide SAT



DPLL^{T}: DPLL modulo theories

How can we extend DPLL to handle formulas over other theories like

- Difference Logic (DL)
- Linear Real Arithmetic (LRA)
- Uninterpreted functions (UF)

Idea: Start with a *Boolean abstraction* (or skeleton) and
incrementally add more *theory* information
until we can conclusively say SAT or UNSAT

Example: DPLL^{LRA}

$F \equiv (x \leq 0 \vee x \leq 10) \wedge (\neg x \leq 0)$

Boolean abstraction: replace every unique linear inequality with a Boolean variable
$F^B \equiv (p \vee q) \wedge (\neg p)$

where p *abstracts* $x \leq 0$ and q *abstracts* $x \leq 10$

Abstraction because information is lost

The relationship $x > 10 \Rightarrow x > 0$, i.e., $\neg q \Rightarrow \neg p$ is lost in F_B

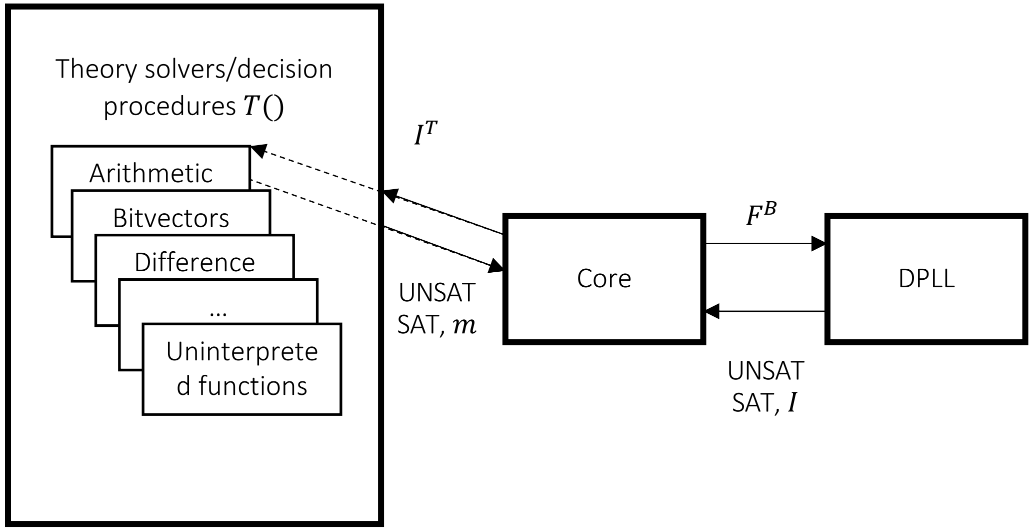
Notation. $(F^B)^T$ maps F^B back to theory T, i.e., $(F^B)^T = F$.

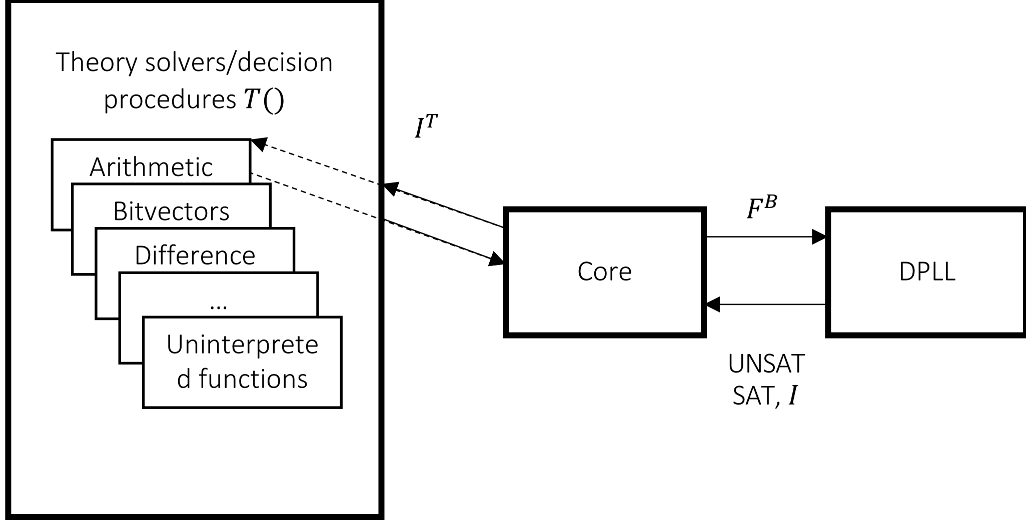
Proposition. If F^B is UNSAT then F is UNSAT, but the converse does not hold, i.e., F^B is SAT does not mean that F is SAT.

Example. $F_1 \equiv (x \leq 0 \wedge x \geq 10)$ is clearly UNSAT, however $F_1^B \equiv p \wedge q$ is SAT.

Lazy DPLL^{T} Algorithm using a Decision Procedure $T()$

Input: A formula F in CNF form over theory T
Output: $I \vDash F$ or UNSAT
Let F^B be the abstraction of F
while true **do**
  if $\mathrm{DPLL}(F^B)$ is unsat then **return** UNSAT
else
  Let I be the model returned by $DPLL$
  Assume I is represented as a formula
  if $T(I^T)$ is sat **then return** SAT and the model returned by $T()$
  else $F^B := F^B \wedge \neg I$





- $\phi \equiv \underbrace{g(a) = c}_{1} \wedge \underbrace{f\big(g(a)\big) \neq f(c)}_{\bar{2}} \vee \underbrace{g(a) = d}_{3} \wedge \underbrace{c \neq d}_{\bar{4}}$
- send $\phi^B \equiv \{1, \bar{2} \vee 3, \bar{4}\}$ to DPLL
- DPLL returns SAT with model I:$\{1, \bar{2}, \bar{4}\}$
- UF solver concretizes $I^{UF} \equiv g(a) = c, f\big(g(a)\big) \neq f(c), c \neq d$
- UF checks I^{UF} as UNSAT
- send $\phi^B \wedge \neg I$: $\{1, \bar{2} \vee 3, \bar{4}, \bar{1} \vee 2 \vee 4\}$ to DPLL; this is a new fact learned by DPLL
- DPLL returns model I': $\{1, 2, 3, \bar{4}\}$
- UF solver concretizes I'^{UF} and finds this to be UNSAT
- send $\phi^B \wedge \neg I \wedge \neg I'$: $\{1, \bar{2} \vee 3, \bar{4}, \bar{1} \vee 2 \vee 4, \bar{1} \vee \bar{2} \vee \bar{3} \vee 4\}$ to DPLL; another fact
- returns UNSAT

Linear Real Arithmetic

Reference : Introduction to Neural Network Verification by Aws Albarghouthi

Decision Procedure for Linear Real Arithmetic

Input: $F \equiv \bigwedge_{i=1}^{n} \sum_{j=1}^{m} c_{ij} x_j \leq b_i$ where $c_{ij}, b_i \in \mathbb{R}$

Output: $\exists \boldsymbol{x} \in \mathbb{R}^m$ such that $\boldsymbol{x} \vDash F$?

Solution based on Simplex Algorithm [Dantzig 1947]

Simplex solves

$\max \sum_{j=1}^{m} a_j x_j$ subject to

$\bigwedge_{i=1}^{n} \sum_{j=1}^{m} c_{ij} x_j \leq b_i$

Our focus will be on finding any solution $\boldsymbol{x} \in \mathbb{R}^m$ that satisfies F

Decision Procedure for Linear Real Arithmetic

Input: $F \equiv \wedge_{i=1}^{n} \Sigma_{j=1}^{m} c_{ij} x_j \leq b_i$ where $c_{ij}, b_i \in \mathbb{R}$

Output: $\exists$ a model $\boldsymbol{x} \in \mathbb{R}^m$ such that $\boldsymbol{x} \vDash F$?

Simplex expects F to be expressed in the Simplex form, which is a conjunction of

- Linear equalities: $\Sigma_{i=1}^{m} c_i x_i = 0$
- Bounds: $\mathrm{l_i} \leq x_i \leq u_i$

Transforming to Simplex Form

Consider the i^{th} inequality in F: $\Sigma_{j=1}^{m} c_{ij} x_j \leq b_i$

Rewrite this as:

$s_i = \Sigma_{j=1}^{m} c_{ij} x_j \wedge$
$s_i \leq b_i$

s_i is called a *slack variable*

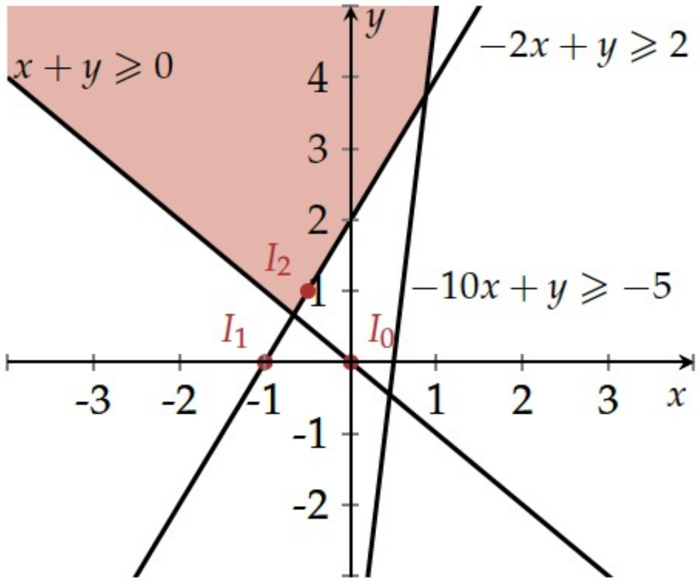
Putting together all the rewritten conjuncts we get F_S

Proposition.

1. Any model of F_S is a model of F, disregarding the assignments to the slack variables.

2. If F_S is UNSAT then F is UNSAT.

Simplex (Informal)



Idea. Simultaneously try to find a model or a proof of UNSAT

Start with some *model (or valuation)* that satisfies all linear equalities (say, $x_i = 0, \forall i$)

In each iteration, pick a bound that is not satisfied and modify the model to satisfy the bound OR discover that the formula is UNSAT

$$\boldsymbol{x_0} = \langle x \mapsto 0, y \mapsto 0 \rangle$$
$$\boldsymbol{x_0}\backslash\text{unsat} - 2x + y \geq 2$$
$$\boldsymbol{x_1} = \langle x \mapsto -1, y \mapsto 0 \rangle$$
$$\boldsymbol{x_1}\backslash\text{unsat}\, x + y \geq 0$$
$$\boldsymbol{x_2} = \left\langle x \mapsto -\frac{1}{2}, y \mapsto 1 \right\rangle \vDash F$$

Variable naming and ordering for Simplex

The input formula F_S (after rewriting) has two types of variables

- **Basic variables** appear on the LHS of an equality; initially these are the *slack variables*
- **Non-basic variables** all others

In each iteration, some basic variable becomes non-basic

We fix an *arbitrary total ordering* on variables $x_1, \ldots, x_n$

For a basic variable x_i and non-basic variable x_j we denote by c_{ij} the coefficient of x_j in the definition of x_i, i.e.,

$x_i = \ldots + c_{ij}\, x_j + \ldots$

The upper and lower bounds of x_i are called u_i and l_i (possibly $\infty, -\infty$)

Simplex (Formal) 1

The algorithm maintains two invariants

1. The model $\boldsymbol{x}$ always satisfies the equalities; bounds may be violated.

    Why is this invariant satisfied by our initialization of all 0s?

2. The bounds of all non-basic variables are all satisfied.

    Why is this invariant satisfied by our initialization?

Simplex Algorithm: DP for LRA

Input: A formula F_S in Simplex form
Output: $\boldsymbol{x} \vDash F_S$ or UNSAT
$\boldsymbol{x} \coloneqq \langle x_i \mapsto 0 \rangle$
while true **do**
if $\boldsymbol{x} \vDash F_S$ **then return** $\boldsymbol{x}$
Let x_i be the first basic variable s.t. $\boldsymbol{x} \lceil x_i < \mathrm{l_i}$ or $\boldsymbol{x} \lceil x_i > \mathrm{u_i}$
if $\boldsymbol{x} \lceil x_i < \mathrm{l_i}$ **then**

$\boldsymbol{x} \lceil x_j \coloneqq \boldsymbol{x} \lceil x_j + \frac{l_i - \boldsymbol{x}\lceil x_i}{c_{ij}}$

else

$\boldsymbol{x} \lceil x_j \coloneqq \boldsymbol{x} \lceil x_j + \frac{u_i - \boldsymbol{x}\lceil x_i}{c_{ij}}$

Pivot x_i and x_j

$x_i = \Sigma_{k \in N}^{m} c_{ik} x_k, j \in N$
Pivoting x_i and x_j rewrites x_j as basic variable
$x_i = c_{ij} x_j + \Sigma_{k \in N\setminus\{j\}}^{m} c_{ik} x_k$
$x_j = -\frac{x_i}{c_{ij}} + \Sigma_{k \in N\setminus\{j\}}^{m} \frac{c_{ik}}{c_{ij}} x_k$

Simplex Algorithm: DP for LRA

Input: A formula F_S in Simplex form
Output: $\boldsymbol{x} \vDash F_S$ or UNSAT
$\boldsymbol{x} := \langle x_i \mapsto 0 \rangle$
while true **do**
if $\boldsymbol{x} \vDash F_S$ **then return** $\boldsymbol{x}$
Let x_i be the first basic variable s.t. $\boldsymbol{x}\lceil x_i < \mathrm{l_i}$ or $\boldsymbol{x}\lceil x_i > \mathrm{u_i}$
if $\boldsymbol{x}\lceil x_i < \mathrm{l_i}$ **then**
Let x_j be the first non-basic variable s.t.
$(\boldsymbol{x}\lceil x_j < u_j \wedge c_{ij} > 0) \vee (\boldsymbol{x}\lceil x_j > l_j \wedge c_{ij} < 0)$
If no such x_j exists **then return** UNSAT
$\boldsymbol{x}\lceil x_j := \boldsymbol{x}\lceil x_j + \frac{l_i - \boldsymbol{x}\lceil x_i}{c_{ij}}$
else Let x_j be the first non-basic variable s.t.
$(\boldsymbol{x}\lceil x_j > l_j \wedge c_{ij} > 0) \vee (\boldsymbol{x}\lceil x_j < u_j \wedge c_{ij} < 0)$
If no such x_j exists **then return** UNSAT
$\boldsymbol{x}\lceil x_j := \boldsymbol{x}\lceil x_j + \frac{u_i - \boldsymbol{x}\lceil x_i}{c_{ij}}$
Pivot x_i and x_j

$x_i = \Sigma_{k \in N}^{m} c_{ik} x_k, j \in N$
Pivoting x_i and x_j rewrites x_j as basic variable

$$x_i = c_{ij} x_j + \Sigma_{k \in N \setminus \{j\}}^{m} c_{ik} x_k$$

$$x_j = -\frac{x_i}{c_{ij}} + \Sigma_{k \in N \setminus \{j\}}^{m} \frac{c_{ik}}{c_{ij}} x_k$$

Example

$x + y \geq 0$
$-2x + y \geq 2$
$-10x + y \geq -5$

Rewritten in Simplex form

$s_1 = x + y$
$s_2 = -2x + y$
$s_3 = -10x + y$
$s_1 \geq 0$
$s_2 \geq 2$
$s_3 \geq -5$

Example continued

$s_1 = x + y$
$s_2 = -2x + y$
$s_3 = -10x + y$
$s_1 \geq 0$
$s_2 \geq 2$
$s_3 \geq -5$

Variable ordering
x, y, s_1, s_2, s_3

Initialization $\boldsymbol{x_0} = \langle x \mapsto 0, y \mapsto 0, s_1 \mapsto 0, s_2 \mapsto 0, s_3 \mapsto 0 \rangle$

$\boldsymbol{x_0}$ satisfies equalities, bounds of s_1 s_3 are satisfied

Pick the first variable x to fix the bound of s_2

Since upper and lower bounds of x are ∞ and $-\infty$ it easily satisfies the blue condition

To increase s_2 to 2 and satisfy its lowerbound we decrease $\boldsymbol{x}[x$ to -1
$\boldsymbol{x_1} = \langle x \mapsto -1, y \mapsto 0, s_1 \mapsto -1, s_2 \mapsto 2, s_3 \mapsto 10 \rangle$

Pivot s_2 with x

$x = -0.5s_2 + 0.5y$
$s_1 = -0.5s_2 + 1.5y$
$s_3 = 5s_2 - 4y$
$s_1 \geq 0$
$s_3 \geq -5$
$-\infty \leq x \leq \infty$

Example continued 2

$$x = -0.5s_2 + 0.5y$$
$$s_1 = -0.5s_2 + 1.5y$$
$$s_3 = 5s_2 - 4y$$
$$s_1 \geq 0$$
$$s_3 \geq -5$$
$$-\infty \leq x \leq \infty$$

$\boldsymbol{x_1} = \langle x \mapsto -1, y \mapsto 0, s_1 \mapsto -1, s_2 \mapsto 2, s_3 \mapsto 10\rangle$

All equalities are still satisfied (invariant)

The only basic variable not satisfying its bounds is now s_1

The first non-basic variable we can tweak is y

Setting y=1 to satisfy the lowerbound of s1 we get
$\boldsymbol{x_2} = \langle x \mapsto -0.5, y \mapsto 1, s_1 \mapsto 0.5, s_2 \mapsto 2, s_3 \mapsto 6\rangle$

Pivot s_1 with y
$\boldsymbol{x_2} \vDash F_S$

$$y = \frac{2}{3}s_1 + \frac{1}{3}s_2$$
$$x = +\frac{1}{3}s_1 - \frac{1}{3}s_2$$
$$s_2 \geq 2$$
$$s_1 \geq 0$$
$$s_3 \geq -5$$
$$-\infty \leq x \leq \infty$$

Why is simplex correct?

- Why does it terminate?

  Because we always looks for the first variable violating the bounds. There is a property (Bland's rule) that ensures that we never revisit the same set of basic and non-basic variables.

- Why does it give the right answer (sound)?
  - If it returns $\boldsymbol{x}$ does it satisfy $\boldsymbol{x} \vDash F$?

    This follows from the condition before **return** $\boldsymbol{x}$
  - If it returns UNSAT is F really unsatisfiable?

Unsatisfiable example

$s_1 = x + y$

$s_2 = -x - 2y$

$s_3 = -x + y$

$s_1 \geq 0$

$s_2 \geq 2$

$s_3 \geq 1$

Consider a Simplex execution in which there are two pivots:

Pivot 1: s_1 with x

$x = s_1 - y$

$s_2 = -s_1 - y$

$s_3 = -s_1 + 2y$

Pivot 2: s_2 with y

$x = 2s_1 + s_2$

$y = -s_1 - s_2$

$s_3 = -3s_1 - 2s_2$

Non-basic variables satisfy their bounds (invariant) and so $s_1 \geq 0, s_2 \geq 2$

If s_2 violates the bound then

$s_3 = -3s_1 - 2s_2 < 1$

We can make s_3 bigger by decreasing s_1 and s_2 but the at most

$s_3 = -3.0 - 2.2 = -4$

which is still less than 1 and Simplex concludes that the formula is UNSAT.

The blue conditions for choosing x_j encodes this condition.

Summary and Takeaways

- Satisfiability modulo theory solvers use theory solvers and DPLL to check satisfiability of formulas in other theories
  - DPLL takes care of disjunctions
  - Theory solvers take care of conjunctions

- Simplex or more generally Linear programming (LP) solvers is a theory solver for linear real arithmetic
  - Simplex algorithm solves LP by incrementally fixing the bounds of basic variables

- Next time Reluplex