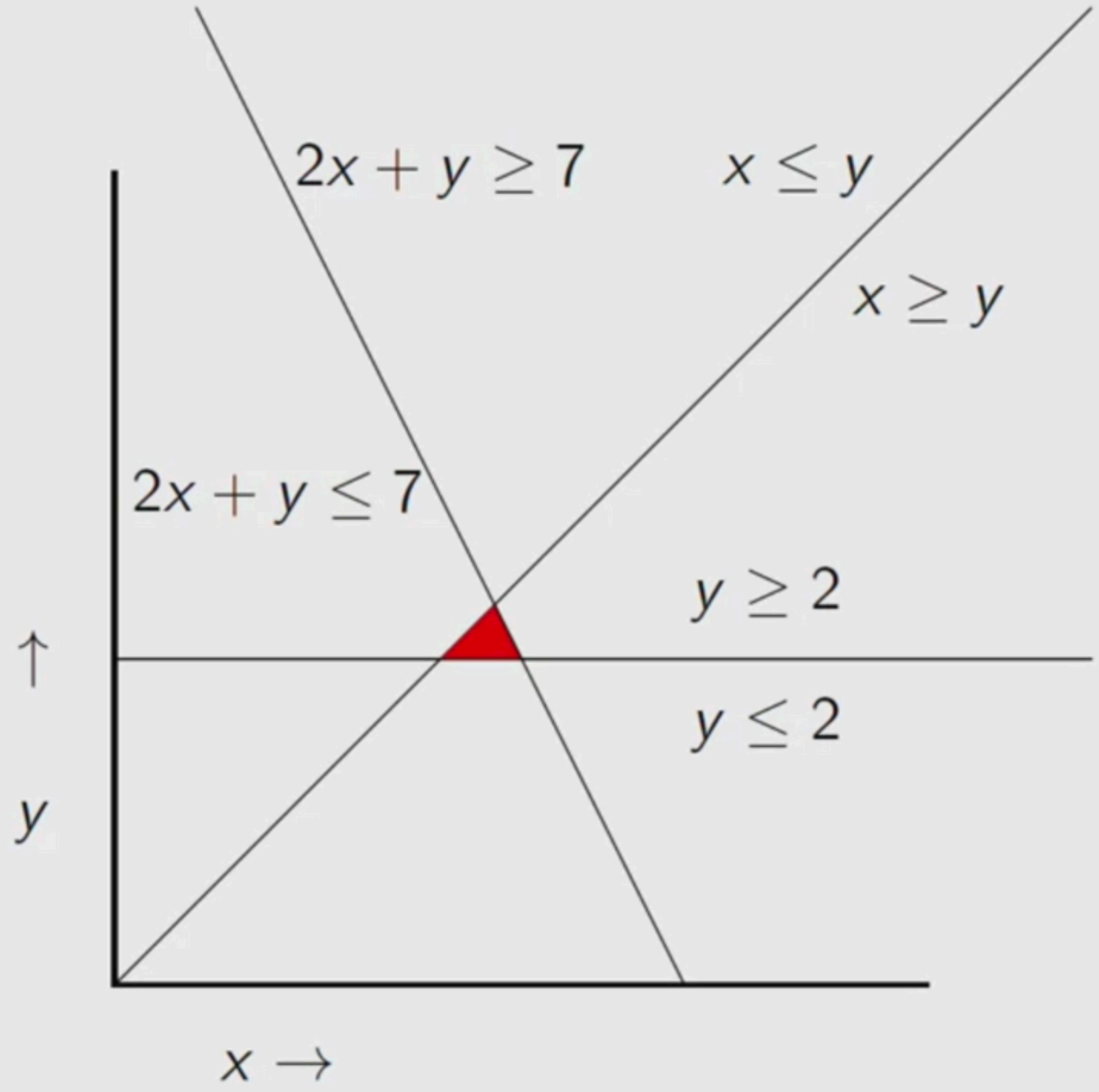
SMT Solver’s Algorithm : Simplex to Reluplex

Konkuk University
Department of Computer Science & Engineering
Kunha Kim

Deal with Inequalities



So indeed the **red** part describes the values x, y satisfying

$$x \geq y$$

$$y \geq 2$$

$$2x + y \leq 7$$

The Simplex method

- But if we have 100, 1000, … variables to check, how can we solve this problem?
- For SMT the underlying approach is the simplex method for linear optimization.
- Among all real values $x_1, \cdots, x_n \geq 0$ find the maximal value of a linear goal function $v + c_1 x_1 + \cdots + c_n x_n$ satisfying k linear constraints $a_{i1} x_1 + a_{i2} x_2 + \cdots + a_{in} x_n \leq b_i$ for $i = 1,2,\cdots,k$.
- Here v, a_{ij}, c_i and b_i are given real values, satisfying $b_i \geq 0$ for $i = 1,2,\cdots,k$
- Trivially if we choose 0 for all x_i, it can be a solution.
- To use Simplex, we have to encode inequalities to $slack\ form$.
- The left side variable is called $basic$ or $slack\ variable$, and the others called $non-basic\ variable$.

Example 7.A Consider the formula F, which we will use as our running example:

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

For clarity, we will drop the conjunction operator and simply list the inequalities. We convert F into a formula F_s in Simplex form:

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

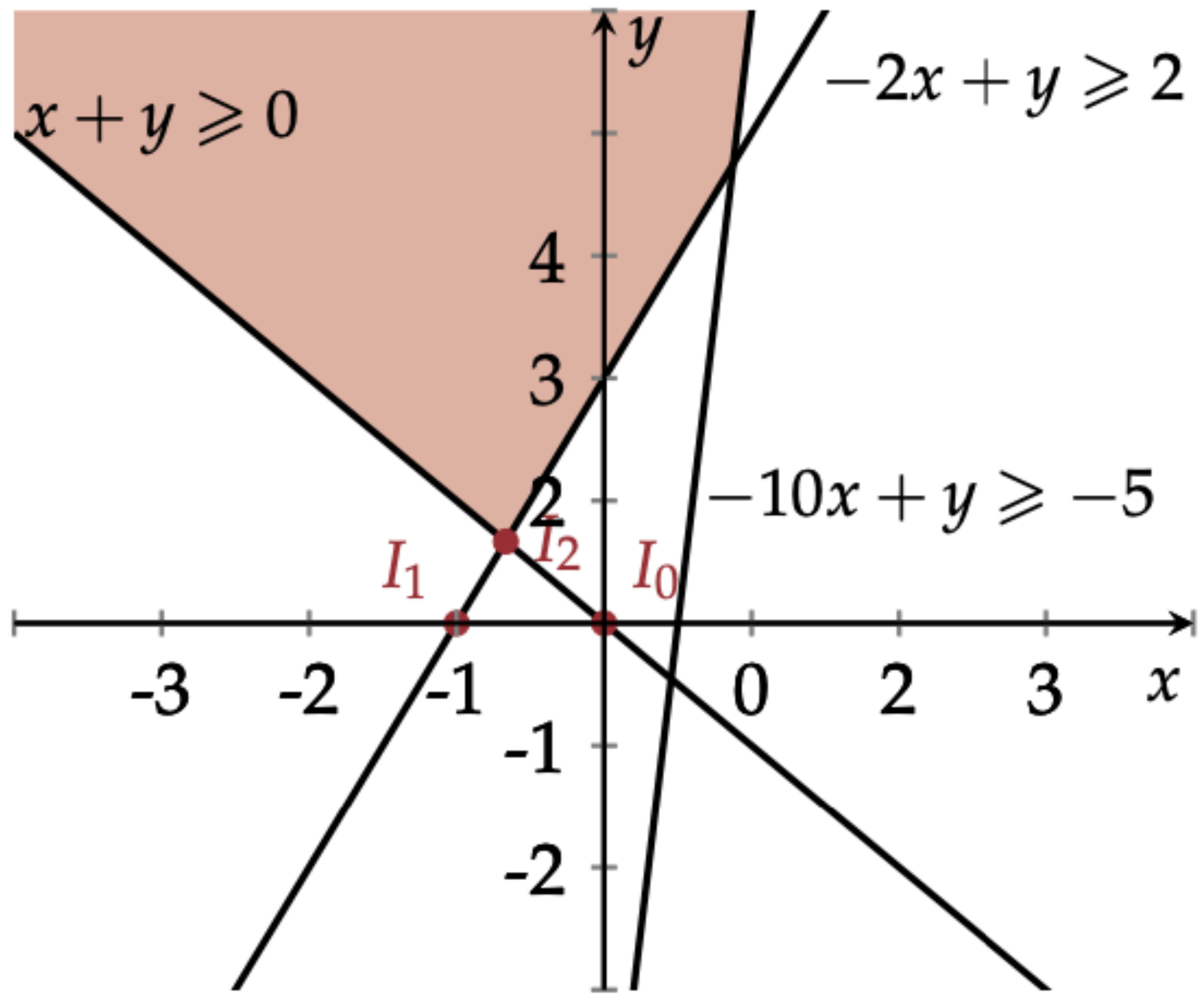


Figure 7.1 Simplex example

Simplex in Detail

- We're now equipped to present the Simplex algorithm, shown in Algorithm 3. The algorithm maintains the following two invariants:

    - The interpretation I always satisfies the equalities, so only the bounds may be violated. This is initially true, as I assigns all variables to 0.

    - The bounds of non-basic variables are all satisfied. This is initially true, as non-basic variables have no bounds.

- In every iteration of the while loop, Simplex looks for a basic variable whose bounds are not satisfied by the current interpretation, and attempts to fix the interpretation.

- There are two symmetric cases, encoded as two branches of the if statement, $x_i < l_i$ or $x_i > u_i$.

Simplex in Detail (Cont.)

- Let's consider $x_i < l_i$. We need to increase x_i in I.
- We don't directly fix x_i, we choose some x_j s.t. $c_{ij} \neq 0$.
- Assume we have found x_j. We increase its current interpretation by $\dfrac{l_i - I(x_i)}{c_{ij}}$; this makes $I(x_i) = l_i$.
- After we have updated the interpretation of x_j, there is a chance that we have violated one of the bounds of x_j.
- We swap x_i with x_j, and replace x_j with $x_j = -\dfrac{x_i}{c_{ij}} + \sum_{kinN\setminus\{j\}} \dfrac{cij}{c_{ik}} x_k$.
- This process is called $pivot$.

Algorithm 3: Simplex

Data: A formula F in Simplex form
Result: $I \models F$ or UNSAT

Let I be the interpretation that sets all variables $fv(F)$ to 0
while *true* **do**
  if $I \models F$ **then return** I
  Let x_i be the first basic variable s.t. $I(x_i) < l_i$ or $I(x_i) > u_i$
  if $I(x_i) < l_i$ **then**
    Let x_j be the first non-basic variable s.t.

$$(I(x_j) < u_j \text{ and } c_{ij} > 0) \text{ or } (I(x_j) > l_j \text{ and } c_{ij} < 0)$$

    if *If no such* x_j *exists* **then return** UNSAT
    $I(x_j) \leftarrow I(x_j) + \frac{l_i - I(x_i)}{c_{ij}}$
  else
    Let x_j be the first non-basic variable s.t.

$$(I(x_j) > l_j \text{ and } c_{ij} > 0) \text{ or } (I(x_j) < u_j \text{ and } c_{ij} < 0)$$

    if *If no such* x_j *exists* **then return** UNSAT
    $I(x_j) \leftarrow I(x_j) + \frac{u_i - I(x_i)}{c_{ij}}$
  Pivot x_i and x_j

Example

Example 7.D Let's now work through our running example in detail. Recall that our formula is:

$$
\begin{aligned}
s_1 &= x + y \\
s_2 &= -2x + y \\
s_3 &= -10x + y \\
s_1 &\geqslant 0 \\
s_2 &\geqslant 2 \\
s_3 &\geqslant -5
\end{aligned}
$$

Say the variables are ordered as follows:

$$x, y, s_1, s_2, s_3$$

$$
\begin{aligned}
x &= 0.5y - 0.5s_2 \\
s_1 &= 1.5y - 0.5s_2 \\
s_3 &= -4y + 5s_2
\end{aligned}
$$

First iteration In the first iteration, we pick the variable x to fix the bounds of s_2, as it is the first one in our ordering. Note that x is unbounded (i.e., its bounds are $-\infty$ and ∞), so it easily satisfies the conditions. To increase the interpretation of s_2 to 2, and satisfy its lower bound, we can decrease $I_0(x)$ to -1, resulting in the following satisfying assignment:

$$I_1 = \{x \mapsto -1,\ y \mapsto 0,\ s_1 \mapsto -1,\ s_2 \mapsto 2,\ s_3 \mapsto 10\}$$

Second iteration The only basic variable not satisfying its bounds is now s_1, since $I_1(s_1) = -1 < 0$. The first non-basic variable that we can tweak is y. We can increase the value of $I(y)$ by $1/1.5 = 2/3$, resulting in the following interpretation:

$$I_2 = \{x \mapsto -2/3,\ y \mapsto 2/3,\ s_1 \mapsto 0,\ s_2 \mapsto 2,\ s_3 \mapsto 7/3\}$$

Formal Definition of LRA

- We denote real arithmetic as $\mathcal{T}_{\mathbb{R}}$. $\mathcal{T}_{\mathbb{R}}$ consists of the signature containing all rational number constants and the symbols $\{+, -, \cdot, \leq, \geq\}$, paired with the standard model of the real numbers.

- We focus on $linear$ formulas : formulas over $\mathcal{T}_{\mathbb{R}}$ with the additional restriction that the multiplication symbol · can only appear if at least one of its operands is a rational constant.

- Linear atoms can always be rewritten into the form $\sum_{x_i \in \mathcal{X}} c_i x_i \star d$, for $\star \in \{=, \leq, \geq\}$, where $\mathcal{X}$ is a set of variables and c_i, d are rational constants.

- Simplex algorithm is efficient decision procedure for determining the $\mathcal{T}_{\mathbb{R}}$ -satisfiability of conjunctions of linear atoms.

Formal Definition of Simplex

- The rules of the calculus operate over data structures we call $configurations$.
- For a given set of variables $\mathcal{X} = \{x_1, \cdots, x_n\}$, a simplex configuration is either one of the distinguished symbols $\{SAT, UNSAT\}$ or a tuple $< B, T, l, u, \alpha >$, where :
- $B \subseteq \mathcal{X}$ is a set of basic variables;
- T, the $tableau$, contains for each $x_i \in B$ an equation $x_i = \sum_{x_j \notin B} c_j x_j$;
- l, u are mappings that assign each variable $x \in \mathcal{X}$ a lower and an upper bound, respectively;
- And α, the $assignment$, maps each variable $x \in \mathcal{X}$ to a real value.
- For $x_i \in B$ and $x_j \notin B$ we denote by $T_{i,j}$ the coefficient c_j of x_j in the equation $x_i = \sum_{x_j \notin B} c_j x_j$.

Derivation rules for Simplex

$$\text{Pivot}_1 \ \frac{x_i \in \mathcal{B}, \quad \alpha(x_i) < l(x_i), \quad x_j \in \text{slack}^+(x_i)}{T := pivot(T, i, j), \quad \mathcal{B} := \mathcal{B} \cup \{x_j\} \setminus \{x_i\}}$$

$$\text{Pivot}_2 \ \frac{x_i \in \mathcal{B}, \quad \alpha(x_i) > u(x_i), \quad x_j \in \text{slack}^-(x_i)}{T := pivot(T, i, j), \quad \mathcal{B} := \mathcal{B} \cup \{x_j\} \setminus \{x_i\}}$$

$$\text{Update} \ \frac{x_j \notin \mathcal{B}, \quad \alpha(x_j) < l(x_j) \vee \alpha(x_j) > u(x_j), \quad l(x_j) \leq \alpha(x_j) + \delta \leq u(x_j)}{\alpha := update(\alpha, x_j, \delta)}$$

$$\text{Failure} \ \frac{x_i \in \mathcal{B}, \quad (\alpha(x_i) < l(x_i) \ \wedge \ \text{slack}^+(x_i) = \emptyset) \vee (\alpha(x_i) > u(x_i) \ \wedge \ \text{slack}^-(x_i) = \emptyset)}{\texttt{UNSAT}}$$

$$\text{Success} \ \frac{\forall x_i \in \mathcal{X}.\ l(x_i) \leq \alpha(x_i) \leq u(x_i)}{\texttt{SAT}}$$

Fig. 3: Derivation rules for the abstract simplex algorithm.

$$\text{slack}^+(x_i) = \{x_j \notin \mathcal{B} \mid (T_{i,j} > 0 \wedge \alpha(x_j) < u(x_j)) \vee (T_{i,j} < 0 \wedge \alpha(x_j) > l(x_j))$$
$$\text{slack}^-(x_i) = \{x_j \notin \mathcal{B} \mid (T_{i,j} < 0 \wedge \alpha(x_j) < u(x_j)) \vee (T_{i,j} > 0 \wedge \alpha(x_j) > l(x_j))$$

Introduction to Reluplex

- Using the Simplex algorithm as the theory solver within $DPLL^T$ allows us to solve formulas in LRA.

- But this approach may inefficient on ReLU function. Because ReLU constraint splits problem into two subproblems, which can be exponential.

- To fix this issue, the work of Katz et al. developed an extension of Simplex, called $Reluplex$, that natively handles ReLU constraints in addition to linear inequalities.

- In worst case it also end up with exponential explosion, but empirically it has been shown to be a promising approach for scaling SMT solving to larger neural networks.

Reluplex in Detail

- Initially, Simplex is invoked on the formula F', which is the original formula F but without the ReLU constraints.
- If Simplex returns $UNSAT$, then we know that F is $UNSAT$. Otherwise, if Simplex returns a model $I \models F'$, it may not be the case that $I \models F$.
- If $I \not\models F$, then we know that one of the ReLU constraints is not satisfied.
- Note that if any of x_i and x_j is a basic variable, we pivot it with a non-basic variable.
- This is because we want to modify the interpretation of one of x_i or x_j, which may affect the interpretation of the other variable if it is a basic variable and $c_{ij} \neq 0$.
- Finally, we modify the interpretation of x_i or x_j, ensuring that $I \models x_i = relu(x_j)$.

Algorithm 4: Reluplex

Data: A formula F in Reluplex form
Result: $I \models F$ or UNSAT

Let I be the interpretation that sets all variables $fv(F)$ to 0
Let F' be the non-ReLU constraints of F
while *true* **do**
  ▷ Calling Simplex (note that we supply Simplex with a reference to the initial interpretation and it can modify it)
  $r \leftarrow \text{Simplex}(F', I)$
  If r is UNSAT **then return** UNSAT
  r is an interpretation I
  if $I \models F$ **then return** I

  ▷ Handle violated ReLU constraint
  Let ReLU constraint $x_i = \text{relu}(x_j)$ be s.t. $I(x_i) \neq \text{relu}(I(x_j))$
  if x_i *is basic* **then**
    pivot x_i with non-basic variable x_k, where $k \neq j$ and $c_{ik} \neq 0$
  if x_j *is basic* **then**
    pivot x_j with non-basic variable x_k, where $k \neq i$ and $c_{jk} \neq 0$
  Perform one of the following operations:

$$I(x_i) \leftarrow \text{relu}(I(x_j)) \quad \textbf{or} \quad I(x_j) \leftarrow I(x_i)$$

  ▷ Case splitting (ensures termination)
  if $u_j > 0$, $l_j < 0$, *and* $x_i = relu(x_j)$ *considered more than* τ *times* **then**
    $r_1 \leftarrow \text{Reluplex}(F \wedge x_j \geqslant 0 \wedge x_i = x_j)$
    $r_2 \leftarrow \text{Reluplex}(F \wedge x_j \leqslant 0 \wedge x_i = 0)$
    if $r_1 = r_2 =$ unsat **then return** UNSAT
    if $r_1 \neq$ unsat **then return** r_1
    return r_2

The Reluplex Algorithm

- We take a different route and extend the theory $\mathcal{T}_{\mathbb{R}}$ to a theory $\mathcal{T}_{\mathbb{R}_R}$ of reals and ReLUs.
- $\mathcal{T}_{\mathbb{R}_R}$ is almost identical to $\mathcal{T}_{\mathbb{R}}$, except that its signature additionally includes the binary predicate ReLU with the interpretation : ReLU(x, y) iff $y = max(0,\ x)$.
- The main idea is to encode a single ReLU node v as a $pair$ of variables, v^b and v^f, and then assert ReLU$(v^b,\ v^f)$.
- v^b, the $backward - facing$ variable, is used to express the connection of v to nodes from the preceding layer.
- v^f, the $foward - facing$ variable, is used for the connections of x to the following layer.

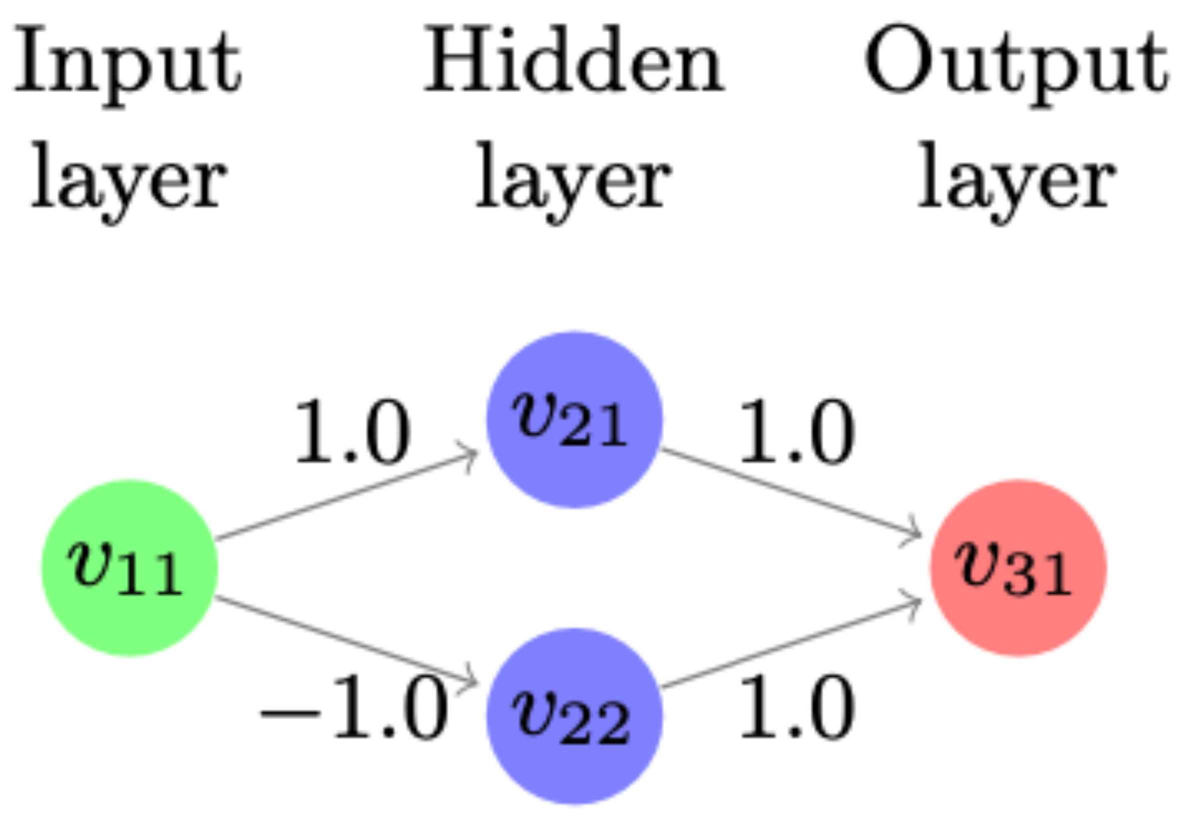
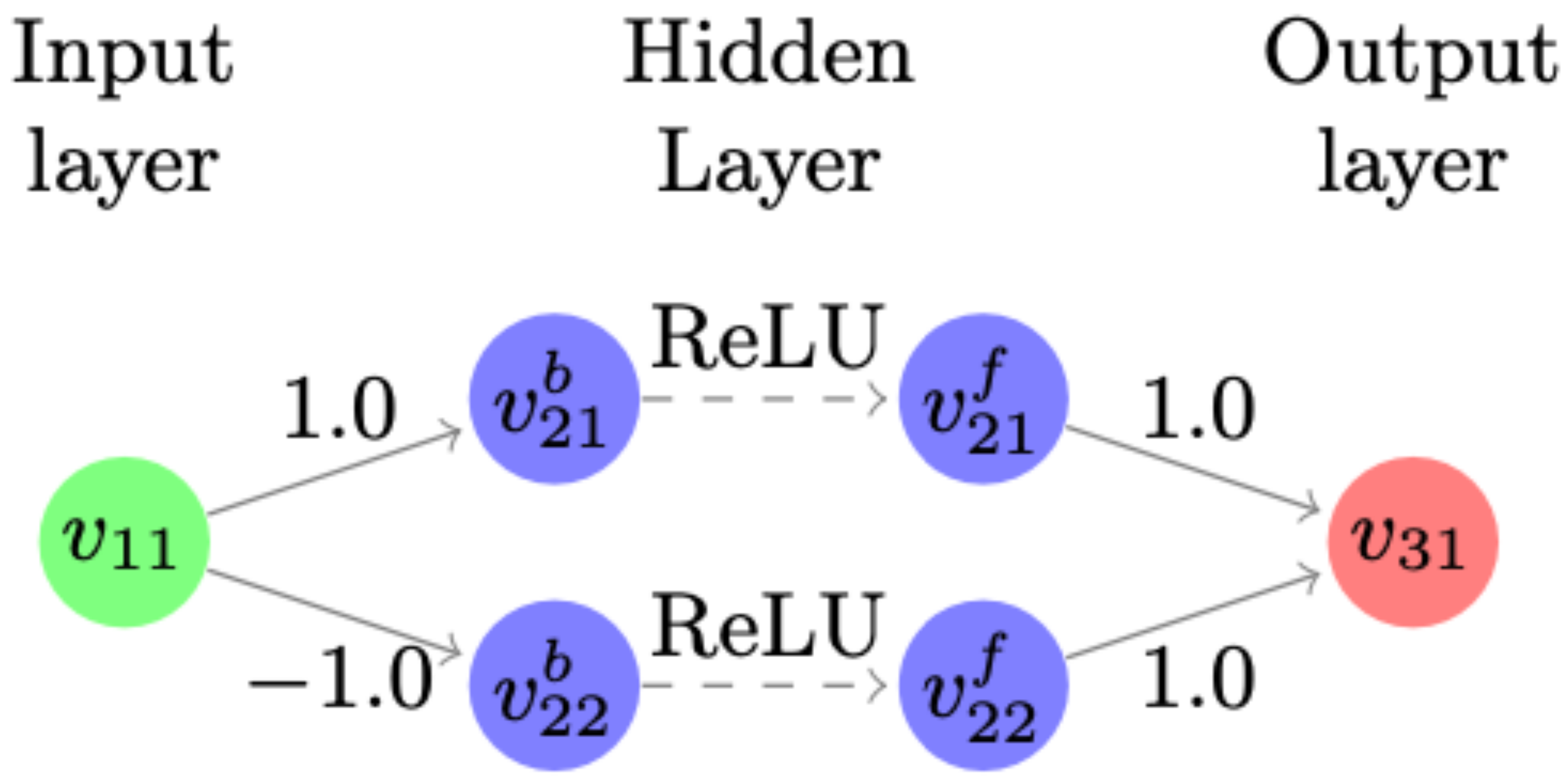


Fig. 2: A small neural network.



Formal Definition of Reluplex

- For a given set of variable $\mathcal{X} = \{x_1, \cdots, x_n\}$, a Reluplex configuration is either one of the distinguished symbols $\{SAT, UNSAT\}$ or a tuple $< B, T, l, u, \alpha, R >$ where B, T, l, u and α are as before, $R \subset \mathcal{X} \times \mathcal{X}$ is the set of ReLU connections.

- The initial configuration for a conjunction of atoms is also obtained as before except that $< x, y > \in R$ iff ReLU(x, y) is an atom.

- The naive approach that we mentioned before only used when Reluplex's approach exceeds some threshold.

- Intuitively, this is likely to limit splits to "problematic" ReLU pairs, while still guaranteeing termination (see Section III of the appendix).

$$\mathsf{Update}_b \quad \frac{x_i \notin \mathcal{B}, \quad \langle x_i, x_j \rangle \in R, \quad \alpha(x_j) \neq \max(0, \alpha(x_i)), \quad \alpha(x_j) \geq 0}{\alpha := \mathit{update}(\alpha, x_i, \alpha(x_j) - \alpha(x_i))}$$

$$\mathsf{Update}_f \quad \frac{x_j \notin \mathcal{B}, \quad \langle x_i, x_j \rangle \in R, \quad \alpha(x_j) \neq \max(0, \alpha(x_i))}{\alpha := \mathit{update}(\alpha, x_j, \max(0, \alpha(x_i)) - \alpha(x_j))}$$

$$\mathsf{PivotForRelu} \quad \frac{x_i \in \mathcal{B}, \quad \exists x_l.\ \langle x_i, x_l \rangle \in R \vee \langle x_l, x_i \rangle \in R, \quad x_j \notin \mathcal{B}, \quad T_{i,j} \neq 0}{T := \mathit{pivot}(T, i, j), \quad \mathcal{B} := \mathcal{B} \cup \{x_j\} \setminus \{x_i\}}$$

$$\mathsf{ReluSplit} \quad \frac{\langle x_i, x_j \rangle \in R, \quad l(x_i) < 0, \quad u(x_i) > 0}{u(x_i) := 0 \qquad l(x_i) := 0}$$

$$\mathsf{ReluSuccess} \quad \frac{\forall x \in \mathcal{X}.\ l(x) \leq \alpha(x) \leq u(x), \quad \forall \langle x^b, x^f \rangle \in R.\ \alpha(x^f) = \max(0, \alpha(x^b))}{\mathtt{SAT}}$$

Fig. 5: Additional derivation rules for the abstract Reluplex algorithm.

Example

- Consider Figure 4. Assume we want to check whether it is possible to satisfy $v_{11} \in [0,1]$ and $v_{31} \in [0.5,1]$. Intuitively, it is true.

- The initial Reluplex configuration is obtained by introducing new basic variables a_1, a_2, a_3, and encoding the network with equations $a_1 = -v_{11} + v_{21}^b$, $a_2 = v_{11} + v_{22}$, $a_3 = -v_{21}^f - v_{22}^f + v_{31}$.

- The equations above form the initial tableau T_0, and the initial set of basic variables is $B = \{a_1, a_2, a_3\}$. The set of ReLU connections is $R = \{ < v_{21}^b, v_{21}^f > , < v_{22}^b, v_{22}^f > \}$.

- At the next slide, we skip details how to derive final tableau for simplicity.

variable	v_{11}	v_{21}^b	v_{21}^f	v_{22}^b	v_{22}^f	v_{31}	a_1	a_2	a_3
lower bound	0	$-\infty$	0	$-\infty$	0	0.5	0	0	0
assignment	0	0	0	0	0	0	0	0	0
upper bound	1	∞	∞	∞	∞	1	0	0	0

$$v_{11} = v_{21}^b - a_1$$
$$v_{22}^b = -v_{21}^b + a_1 + a_2$$
$$v_{21}^f = -v_{22}^f + v_{31} - a_3$$

variable	v_{11}	v_{21}^b	v_{21}^f	v_{22}^b	v_{22}^f	v_{31}	a_1	a_2	a_3
lower bound	0	$-\infty$	0	$-\infty$	0	0.5	0	0	0
assignment	0.5	0.5	0.5	-0.5	0	0.5	0	0	0
upper bound	1	∞	∞	∞	∞	1	0	0	0

Efficiently Implementing Reluplex

- There are some techniques to boost the performance of Reluplex : tighter bound derivation, conflict analysis, floating point arithmetic and under approximation.

- Consider a basic variable $x_i \in B$ and let pos(x_i) = $\{x_j \notin B \mid T_{i,j} > 0\}$ and neg(x_i) = $\{x_j \notin B \mid T_{i,j} < 0\}$.

- Tighter bound derivation is critical for the cost of Reluplex, because it can eliminate the case splitting.

- The actual amount of bound tightening to perform can be determined heuristically; we describe the heuristic that we used in Section 6.

Conflict Analysis

- Conflict analysis is a standard technique in SAT / SMT Solver.
- Bound derivation can lead to situations where we learn that $l(x) > u(x)$ for some variable x.
- Such contradictions allow Reluplex to immediately undo a previous split(or UNSAT if no previous split exist).
- However, in many cases more than just the previous split can be undone.
- You can check an example here : https://en.wikipedia.org/wiki/Conflict-driven_clause_learning or https://ieeexplore.ieee.org/document/769433

Floating Point Arithmetic

- SMT solvers typically use precise (as opposed to floating point) arithmetic to avoid roundoff errors and guarantee soundness.
- Unfortunately, precise computation is usually at least an order of magnitude slower than its floating point equivalent.
- To provide acceptable range of roundoff error, we also added some safeguards.
- (i) After a certain number of Pivot steps we would measure the accumulated roundoff error;
- (ii) If the error exceeded a threshold M, we would restore the coefficients of the current tableau T using the initial tableau T_0.

Floating Point Arithmetic(Cont.)

- Cumulative roundoff error defined as $\sum_{x_i in B_0} |\alpha(x_i) - \sum_{x_j \notin B_0} T_{0_{i,j}} \times \alpha(x_j)|$.

- T is restored by starting from T_0 and performing a short series of Pivot steps that result in the same set of basic variables as in T.

- In general, the shortest sequence of pivot steps to transform T_0 to T is much shorter than the series of steps that was followed by Reluplex.

- Hence, although it is also performed using floating point arithmetic, it incurs a smaller roundoff error.

- The tableau restoration technique serves to increase our confidence in the algorithm's results when using floating point arithmetic, but it does not guarantee soundness.

ACAS Xu System

- Airborne collision avoidance systems are critical for ensuring the safe operation of aircraft.
- The unmanned variant of ACAS X, known as ACAS Xu, produces horizontal maneuver advisories.
- So far, development of ACAS Xu has focused on using a large lookup table that maps sensor measurements to advisories. However, this table requires over 2GB of memory.
- To overcome this challenge, a DNN representation was explored as a potential replacement for the table.
- Initial results show a dramatic reduction in memory requirements without compromising safety.
- In fact, due to its continuous nature, the DNN approach can sometimes outperform the discrete lookup table.

ACAS Xu System(Cont.)

- A DNN implementation of ACAS Xu presents new certification challenges.
- Proving that a set of inputs cannot produce an erroneous alert is paramount for certifying the system for use in safety-critical settings.
- Previous certification methodologies included exhaustively testing the system in 1.5 million simulated encounters, but this is insufficient for proving that faulty behaviors do not exist within the continuous DNNs.
- This highlights the need for verifying DNNs and makes the ACAS Xu DNNs prime candidates on which to apply Reluplex.

Network Functionality

- The ACAS Xu system maps input variables to action advisories. Each advisory is assigned a score, with the lowest score corresponding to the best action.
- The input state is composed of seven dimensions:
    - ρ : distance from ownship to intruder
    - θ : Angle to intruder relative to ownship heading direction
    - ψ : Heading angle of intruder relative to ownship heading direction
    - v_{own} : Speed of ownship
    - v_{int} : Speed of intruder
    - τ : Time until loss of vertical separation
    - a_{prev} : Previous advisory

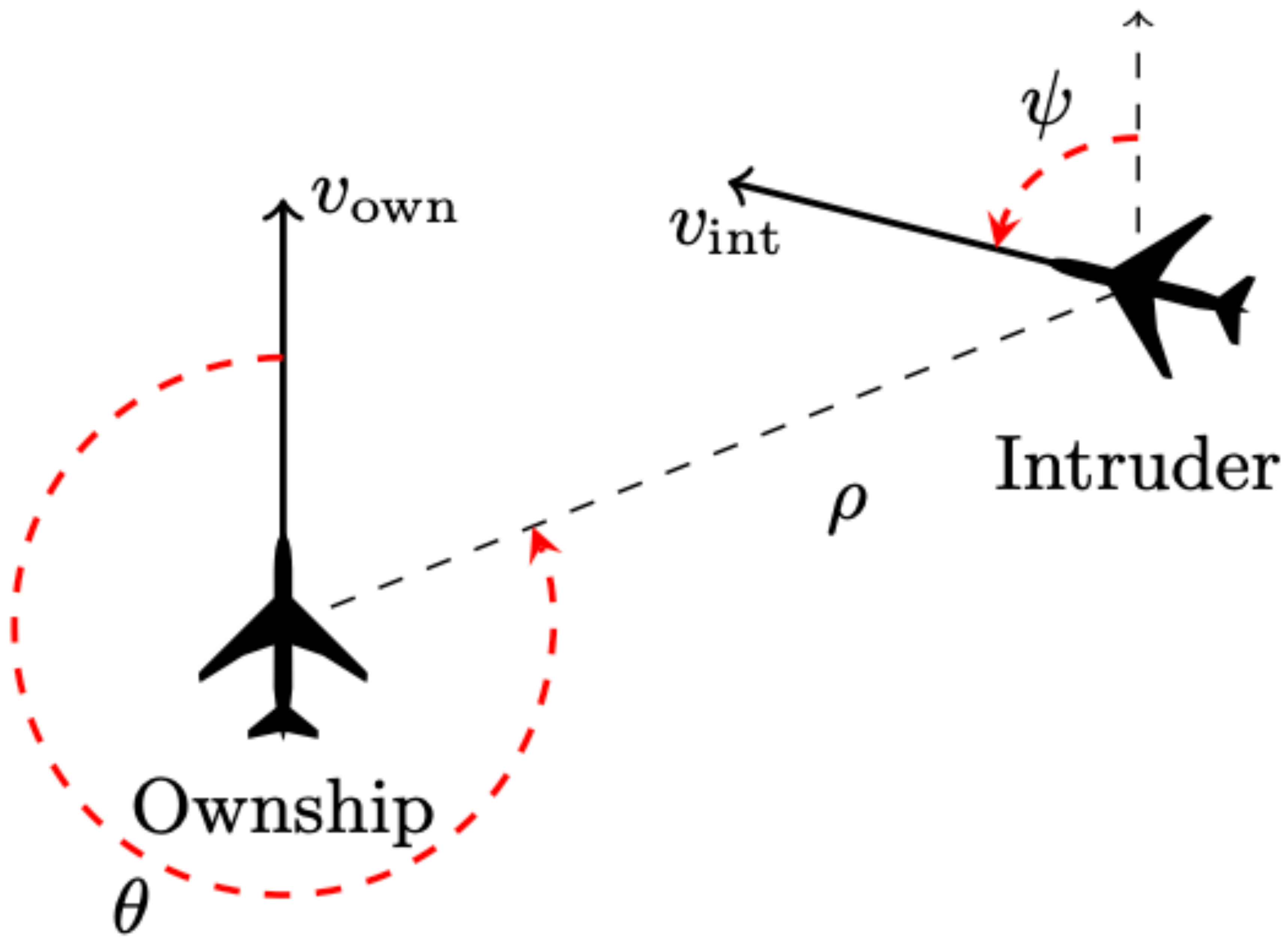


Fig. 6: Geometry for ACAS Xu Horizontal Logic Table

Network Functionality(Cont.)

- There are five outputs which represent the different horizontal advisories that can be given to the ownship:

- Clear-of-Conflict (COC), weak right, strong right, weak left, or strong left. Weak and strong mean heading rates of $1.5°/s$ and $3.0°/s$, respectively.

- The array of 45 DNNs was produced by discretizing τ and a_{prev}, and producing a network for each discretized combination.

- Each of these networks thus has five inputs (one for each of the other dimensions) and five outputs.

- The DNNs are fully connected, use ReLU activation functions, and have 6 hidden layers with a total of 300 ReLU nodes each.

Network Properties

- It is desirable to verify that the ACAS Xu networks assign correct scores to the output advisories in various input domains.
- Fig. 7 illustrates this kind of property by showing a top-down view of a head-on encounter scenario, in which each pixel is colored to represent the best action if the intruder were at that location.
- We used Reluplex to prove properties from the following categories on the DNNs:
- (i) The system does not give unnecessary turning advisories;
- (ii) Alerting regions are uniform and do not contain inconsistent alerts; and
- (iii) Strong alerts do not appear for high τ values.

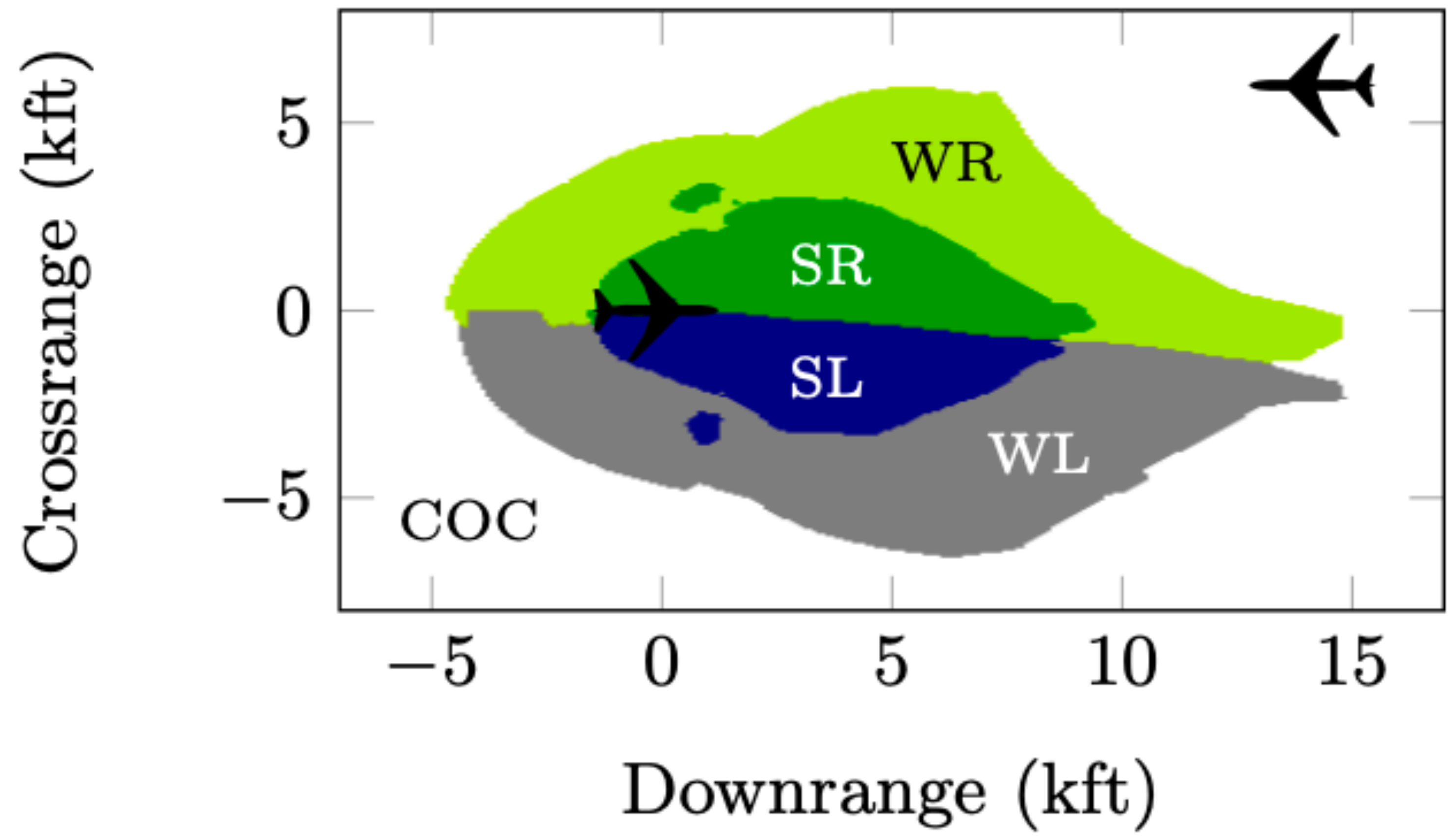


Fig. 7: Advisories for a head-on encounter with $a_{\text{prev}} = \text{COC}, \tau = 0\,\text{s}$.

Comparing with other Solvers

- Our implementation consists of three main logical components:
- (i) A simplex engine for providing core functionality such as tableau representation and pivot and update operations;
- (ii) A Reluplex engine for driving the search and performing bound derivation, ReLU pivots and ReLU updates; and
- (iii) A simple SMT core for providing splitting-on-demand services.
- We ran all solvers with a 4 hour timeout on 2 of the ACAS Xu networks (selected arbitrarily), trying to solve for 8 simple satisfiable properties $\varphi_1, \cdots, \varphi_8$, each of the form $x \geq c$ for a fixed output variable x and a constant c.
- The SMT solvers generally performed poorly, with only Yices and MathSat successfully solving two instances each.
- The Gurobi LP Solver is so fast when the problem doesn't need case splitting. But when splitting is needed, it would timeout.

Table 1: Comparison to SMT and LP solvers. Entries indicate solution time (in seconds).

	φ_1	φ_2	φ_3	φ_4	φ_5	φ_6	φ_7	φ_8
CVC4	-	-	-	-	-	-	-	-
Z3	-	-	-	-	-	-	-	-
Yices	1	37	-	-	-	-	-	-
MathSat	2040	9780	-	-	-	-	-	-
Gurobi	1	1	1	-	-	-	-	-
Reluplex	8	2	7	7	93	4	7	9

Verifying Properties

- Next, we used Reluplex to test a set of 10 quantitive properties $\phi_1, \cdots, \phi_{10}$.
- The Stack and Splits columns list the maximal depth of nested case-splits reached (averaged over the tested networks) and the total number of case-splits performed, respectively.
- For each property, we looked for an input that would violate it; thus, an UNSAT result indicates that a property holds, and a SAT result indicates that it does not hold.
- The properties are formally defined in appendix Section VI.
- We observe that for all properties, the maximal depth of nested splits was al- ways well below the total number of ReLU nodes, 300, illustrating the fact that Reluplex did not split on many of them.
- Also, the total number of case-splits indicates that large portions of the search space were pruned.

Property ϕ_1.

- Description: If the intruder is distant and is significantly slower than the ownship, the score of a COC advisory will always be below a certain fixed threshold.
- Tested on: all 45 networks.
- Input constraints: $\rho \geq 55947.691$, $v_{\text{own}} \geq 1145$, $v_{\text{int}} \leq 60$.
- Desired output property: the score for COC is at most 1500.

Property ϕ_2.

- Description: If the intruder is distant and is significantly slower than the ownship, the score of a COC advisory will never be maximal.
- Tested on: $N_{x,y}$ for all $x \geq 2$ and for all y.
- Input constraints: $\rho \geq 55947.691$, $v_{\text{own}} \geq 1145$, $v_{\text{int}} \leq 60$.
- Desired output property: the score for COC is not the maximal score.

Property ϕ_3.

- Description: If the intruder is directly ahead and is moving towards the ownship, the score for COC will not be minimal.
- Tested on: all networks except $N_{1,7}$, $N_{1,8}$, and $N_{1,9}$.
- Input constraints: $1500 \leq \rho \leq 1800$, $-0.06 \leq \theta \leq 0.06$, $\psi \geq 3.10$, $v_{\text{own}} \geq 980$, $v_{\text{int}} \geq 960$.
- Desired output property: the score for COC is not the minimal score.

Table 2: Verifying properties of the ACAS Xu networks.

	Networks	Result	Time	Stack	Splits
ϕ_1	41	UNSAT	394517	47	1522384
	4	TIMEOUT			
ϕ_2	1	UNSAT	463	55	88388
	35	SAT	82419	44	284515
ϕ_3	42	UNSAT	28156	22	52080
ϕ_4	42	UNSAT	12475	21	23940
ϕ_5	1	UNSAT	19355	46	58914
ϕ_6	1	UNSAT	180288	50	548496
ϕ_7	1	TIMEOUT			
ϕ_8	1	SAT	40102	69	116697
ϕ_9	1	UNSAT	99634	48	227002
ϕ_{10}	1	UNSAT	19944	49	88520

Verifying Robustness

- Another class of properties that we tested is $adversarial\ robustness$ properties.
- We say that a network is $\delta - locally - robust$ at input point x if for every x' such that $||x - x'||_\infty \leq \delta$, the network assigns the same label to x and x'.
- In the case of ACAS Xu DNNs, this means that the same output has the lowest score for both x and x'.
- SAT results show that Reluplex found an adversarial input within the prescribed neighborhood, and UNSAT results indicate that no such inputs exist.

Table 3: Local adversarial robustness tests. All times are in seconds.

	$\delta = 0.1$		$\delta = 0.075$		$\delta = 0.05$		$\delta = 0.025$		$\delta = 0.01$		Total
	Result	Time	Result	Time	Result	Time	Result	Time	Result	Time	Time
Point 1	SAT	135	SAT	239	SAT	24	UNSAT	609	UNSAT	57	1064
Point 2	UNSAT	5880	UNSAT	1167	UNSAT	285	UNSAT	57	UNSAT	5	7394
Point 3	UNSAT	863	UNSAT	436	UNSAT	99	UNSAT	53	UNSAT	1	1452
Point 4	SAT	2	SAT	977	SAT	1168	UNSAT	656	UNSAT	7	2810
Point 5	UNSAT	14560	UNSAT	4344	UNSAT	1331	UNSAT	221	UNSAT	6	20462

Global Robustness

- Finally, we mention an additional variant of adversarial robustness which we term *global adversarial robustness*, and which can also be solved by Reluplex.

- Whereas local adversarial robustness is measured for a specific x, global adversarial robustness applies to all inputs simultaneously.

- This is expressed by encoding two side-by-side copies of the DNN in question, N_1 and N_2, operating on separate input variables x_1 and x_2, respectively, such that x_2 represents an adversarial perturbation of x_1.

- We can then check whether $||x_1 - x_2|| \leq \delta$ implies that two copies of DNN produce similar output.

- Formally, we require that if N_1 and N_2 assign output a values p_1 and p_2 respectively, then $|p_1 - p_2| \leq \epsilon$.

- This can be checked for only small networks.