Satisfiability Checking

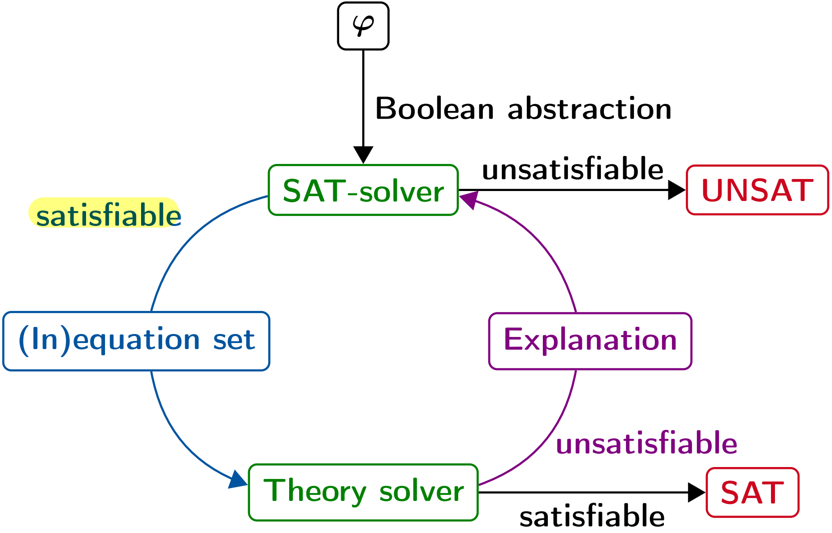
17 Summary II

Prof. Dr. Erika Ábrahám

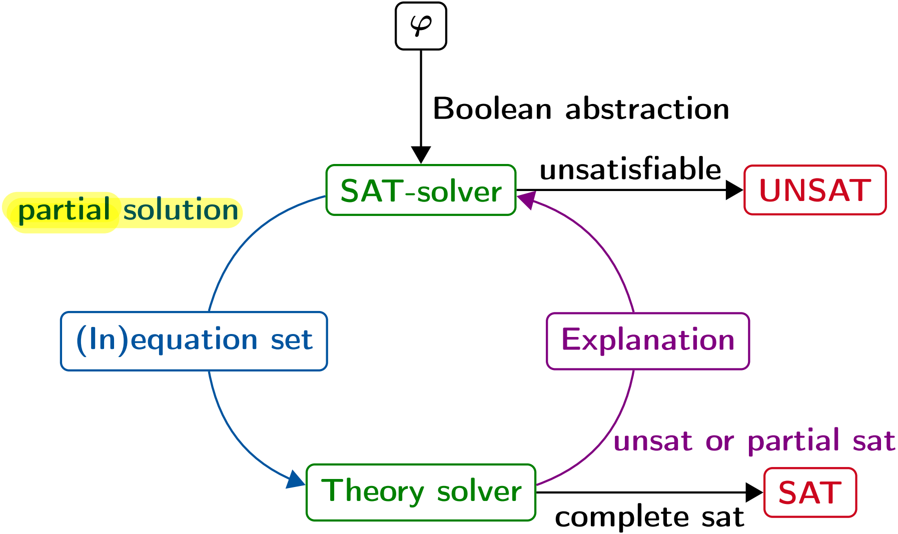
RWTH Aachen University
Informatik 2
LuFG Theory of Hybrid Systems

WS 22/23

Full lazy SMT-solving



Less lazy SMT-solving



Requirements on the theory solver

1. Incrementality: In less lazy solving we incrementally extend a set of constraints, whose satisfiability check should be carried out by the theory solver. The theory solver should make use of the previous satisfiability check for the analysis of the extended set.
2. (Preferably minimal) infeasible subsets: If the constraint set is infeasible then compute a reason for unsatisfaction.
3. Backtracking: The theory solver should be able to remove constraints (in inverse chronological order).

17 Summary II

Conjunctions of equalities: Transitive closure

$$\varphi^E : \quad x_1 = x_2 \wedge x_2 = x_3 \wedge x_4 = x_5 \wedge x_1 \neq x_3$$

Transitive closure:

For each equality, merge the classes of the respective variables!

x_1 x_2 x_3 x_4 x_5

UNSAT

How to achieve incrementality? How to compute infeasible subsets?

Conjunction of equalities: Algorithm

Input: A conjunction φ of equalities and disequalities without UF

Algorithm

1. Define an equivalence class for each variable in φ.
2. For each equality $x = y$ in φ: merge the equivalence classes of x and y.
3. For each disequality $x \neq y$ in φ:
   if x is in the same class as y, return 'UNSAT'.
4. Return 'SAT'.

Uninterpreted functions: Congruence closure

$$\varphi^E : \quad x_1 = x_2 \wedge x_2 = x_3 \wedge x_4 = x_5 \wedge x_1 \neq x_5 \wedge F(x_1) \neq F(x_3)$$

Congruence closure:

If all the arguments of two function applications are in the same class, merge the classes of the applications!

$x_1 \quad x_2 \quad x_3 \qquad x_4 \quad x_5$

UNSAT

$F(x_1) \quad F(x_3)$

How to achieve incrementality? How to compute infeasible subsets?

Conjunction of equalities with UF: Algorithm

Input: A conjunction φ of equalities and disequalities with UFs
Output: Satisfiability of φ

Algorithm

1. $\mathcal{C} := \{\{t\} \mid t$ occurs as subexpression in an (in)equation in $\varphi\}$;
2. for each equality $t = t'$ in φ with $[t] \neq [t']$

   $\mathcal{C} := (\mathcal{C} \setminus \{[t], [t']\}) \cup \{[t] \cup [t']\}$;

   while exists $F(t), F(t')$ in φ with $[t] = [t']$ and $[F(t)] \neq [F(t')]$

   $\mathcal{C} := (\mathcal{C} \setminus \{[F(t)], [F(t')]\}) \ \cup \ \{[F(t)] \cup [F(t')]\}$;
3. for each inequality $t \neq t'$ in φ

   if $[t] = [t']$ return "UNSAT";
4. return "SAT";

($[t] \in \mathcal{C}$ denotes the unique set in $\mathcal{C}$ that contains t)

Gaussian and Fourier-Motzkin variable elimination

- Goal: decide satisfiability of conjunctions of linear constraints over the reals (ℓ equations, k inequations, n variables)

$$\bigwedge_{1\leq i\leq \ell} \sum_{1\leq j\leq n} c_{ij}x_j = d_i \wedge \bigwedge_{1\leq i\leq k} \sum_{1\leq j\leq n} a_{ij}x_j \leq b_i$$

- Eliminate variable x_n:

  Gauss: If there exists an equation $\sum_{1\leq j\leq n} c_{ij}x_j = d_i$ with some $c_{in} \neq 0$ then remove this equation and replace x_n by $\frac{d_i}{c_{in}} - \sum_{j=1}^{n-1} \frac{c_{ij}}{c_{in}} x_j$ in all remaining constraints.

  Fourier-Motzkin: Otherwise partition the inequations according to the coefficients a_{in}:

  - $a_{in} = 0$: no bound on x_n
  - $a_{in} > 0$: upper bound $\beta_i = \frac{b_i}{a_{in}} - \sum_{j=1}^{n-1} \frac{a_{ij}}{a_{in}} x_j$ on x_n
  - $a_{in} < 0$: lower bound $\beta_i = \frac{b_i}{a_{in}} - \sum_{j=1}^{n-1} \frac{a_{ij}}{a_{in}} x_j$ on x_n

  Remove all inequalitites defining a bound on x_n and add for each pair of a lower bound β_ℓ and upper bound β_u the constraint $\beta_\ell \leq \beta_u$.

Gaussian and Fourier-Motzkin variable elimination

How to achieve incrementality? 每次新加的 lower bound / upper bound，和其他 upper bound / lower bound 比较

How to compute infeasible subsets?

Satisfiability with simplex

- Problem: Check the satisfiability of a linear inequation system

$$\bigwedge_{i=1}^{m} \sum_{j=1}^{n} a_{ij} x_j \bowtie_i b_i \qquad \bowtie_i \in \{\leq, \geq\}$$

  in the real domain

- General form:

$$A\vec{x} = \vec{s} \quad \wedge \quad \bigwedge_{i=1}^{m} s_i \bowtie_i b_i$$

  $s_1, \ldots, s_m$: additional/auxiliary/slack variables

- Tableau:

$$\begin{array}{rcc} & & \begin{array}{ccc} x_1 & \ldots & x_n \end{array} \leftarrow \text{Non-basic variables } \mathcal{N} \\ \text{Basic variables } \mathcal{B} \rightarrow & \begin{array}{c} s_1 \\ \ldots \\ s_m \end{array} & \begin{pmatrix} a_{11} & \ldots & a_{1n} \\ & \ldots & \\ a_{m1} & \ldots & a_{mn} \end{pmatrix} \end{array}$$

Data structures

- Simplex maintains:
  - The tableau,
  - an assignment α to all (original and additional) variables.

  Initially, $\alpha(x_i) = 0$ for $i \in \{1, ..., n+m\}$
- Two invariants are maintained throughout:
  - $A\vec{x} = \vec{s}$
  - All non-basic variables satisfy their bounds

1. If the bounds of all basic variables are satisfied by α, return "satisfiable".
2. Otherwise, find a basic variable x_i that violates its bounds.
3. Find a suitable non-basic variable x_j such that
   - $(\alpha(x_i) < l_i \land a_{ij} > 0) \rightarrow \alpha(x_j) < u_j$
   - $(\alpha(x_i) < l_i \land a_{ij} < 0) \rightarrow \alpha(x_j) > l_j$
   - $(\alpha(x_i) > u_i \land a_{ij} > 0) \rightarrow \alpha(x_j) > l_j$
   - $(\alpha(x_i) > u_i \land a_{ij} < 0) \rightarrow \alpha(x_j) < u_j$
4. If there is no suitable variable, return "unsatisfiable".
5. Otherwise, pivot.

Pivoting

$$
\begin{array}{c}
\\
y_1 \\ \cdots \\ y_i \\ \cdots \\ y_m
\end{array}
\begin{array}{c}
\begin{array}{ccccc} y'_1 & \cdots & y'_j & \cdots & y'_n \end{array} \\
\begin{pmatrix}
a_{11} & \cdots & a_{1j} & \cdots & a_{1n} \\
 & & \cdots & & \\
a_{i1} & \cdots & a_{ij} & \cdots & a_{in} \\
 & & \cdots & & \\
a_{m1} & \cdots & a_{mj} & \cdots & a_{mn}
\end{pmatrix}
\end{array}
$$

$$\Downarrow$$

$$
\begin{array}{c}
\\
y_1 \\ \cdots \\ y'_j \\ \cdots \\ y_m
\end{array}
\begin{array}{c}
\begin{array}{ccccc} y'_1 & \cdots & y_i & \cdots & y'_n \end{array} \\
\begin{pmatrix}
a_{11} - \frac{a_{1j}a_{i1}}{a_{ij}} & \cdots & \frac{a_{1j}}{a_{ij}} & \cdots & a_{1n} - \frac{a_{1j}a_{in}}{a_{ij}} \\
 & & \cdots & & \\
-\frac{a_{i1}}{a_{ij}} & \cdots & \frac{1}{a_{ij}} & \cdots & -\frac{a_{in}}{a_{ij}} \\
 & & \cdots & & \\
a_{m1} - \frac{a_{mj}a_{i1}}{a_{ij}} & \cdots & \frac{a_{mj}}{a_{ij}} & \cdots & a_{mn} - \frac{a_{mj}a_{in}}{a_{ij}}
\end{pmatrix}
\end{array}
$$

1. $\alpha(y_i) = l_i$ 2. $\alpha(y'_k)$ unchanged for $k \neq j$ 3. rest according to the new matrix

Termination

To achieve completeness, we use Bland's rule:

1. Determine a total order on the variables
2. Choose the first basic variable that violates its bounds, and the first non-basic suitable variable for pivoting.

General simplex with Bland's rule

1. Transform the system into the general form

$$A\vec{x} = \vec{s} \quad \text{and} \quad \bigwedge_{i=1}^{m} l_i \leq s_i \leq u_i \; .$$

2. Construct the tableau with the initial assignment.
3. Determine a fixed order on the variables.
4. If there is no basic variable that violates its bounds, return "satisfiable". Otherwise, let y_i be the first basic variable in the order that violates its bounds.
5. Search for the first suitable non-basic variable y'_j in the order for pivoting with y_i. If there is no such variable, return "unsatisfiable".
6. Perform the pivot operation on y_i and y'_j.
7. Go to step 4.

Requirements on the theory solver

- (Minimal) infeasible subsets (to explain infeasibility)
- Incrementality (to add constraints stepwise)
- Backtracking (to mimic backtracking in the SAT solver)

Minimal infeasible subsets in simplex:

- The constraints corresponding to the basic variable of the contradictory row and all non-basic variables with non-zero coefficients in this row are together unsatisfiable.

Incrementality in simplex:

- Add all constraints but without bounds on non-active constraints.
- If a constraint becomes true, activate its bound.

Backtracking in simplex:

- Remove bounds of unassigned constraints

17 Summary II

1. Full and less lazy SMT solving
2. Equality logic with uninterpreted functions
3. Gaussian and Fourier-Motzkin variable elimination
4. The simplex method
5. Branch and bound

Integer linear systems

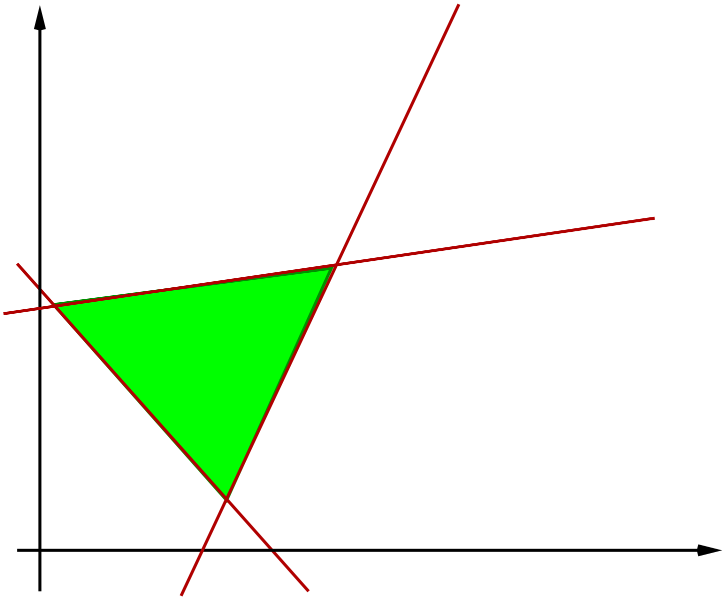
Definition

An integer linear system S is a linear system $Ax = 0$, $\bigwedge_{i=1}^{m} l_i \leq s_i \leq u_i$, with the additional integrality requirement that all variables are of type integer.

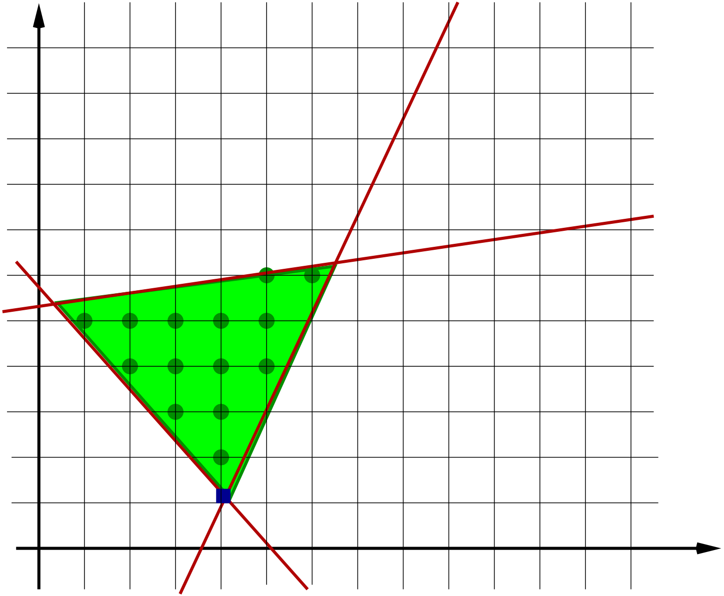
Definition (relaxed system)

Given an integer linear system S, its relaxation relaxed(S) is S without the integrality requirement.

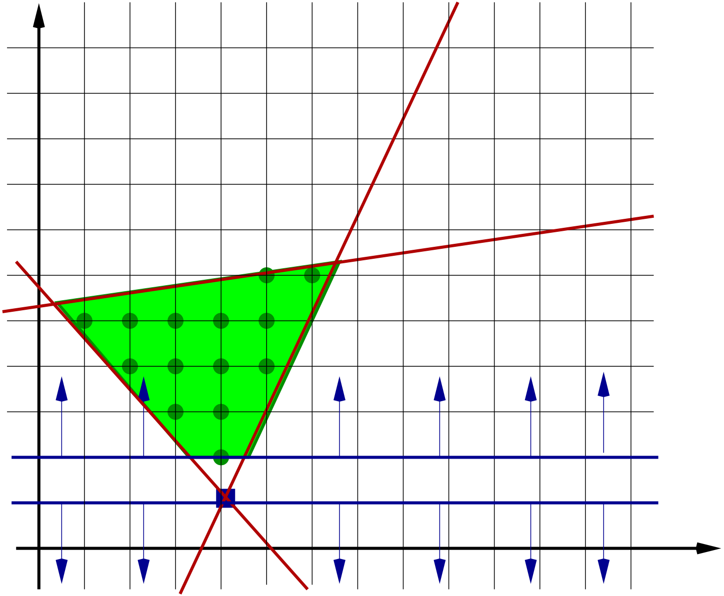
Geometric interpretation



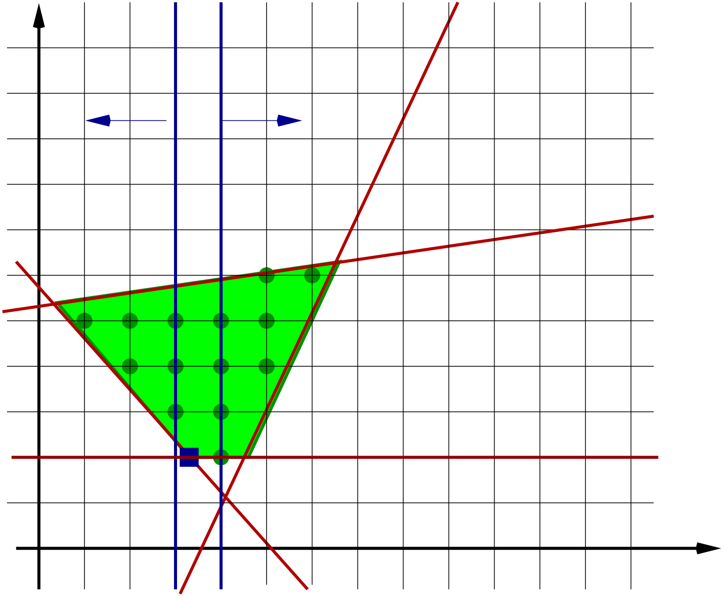
Geometric interpretation



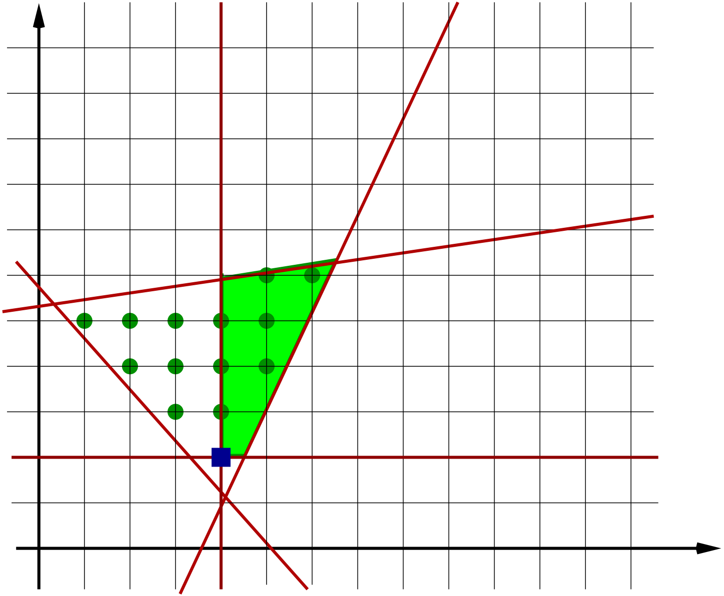
Geometric interpretation



Geometric interpretation



Geometric interpretation



Branch and bound algorithm

Input: An integer linear system S
Output: SAT if S is satisfiable, UNSAT otherwise

```
procedure Branch-and-Bound(S) {
  res = LP(relaxed(S));
  if (res==UNSAT) return UNSAT;
  else if (res is integral) return SAT;
  else {
    Select a variable v that is assigned a non-integral value r;
    if (Branch-and-Bound(S ∪ {v ≤ ⌊r⌋})==SAT) return SAT;
    else if (Branch-and-Bound(S ∪ {v ≥ ⌈r⌉})==SAT) return SAT;
    else return UNSAT;
  }
}
```