

Entscheidungsverfahren mit Anwendungen in der Softwareverifikation

VI: SMT, DPLL(T)

Carsten Sinz
Institut für Theoretische Informatik

21.11.2018

- Theorien
- SMT-LIB
- DPLL(T): Idee
- Abstraktes DPLL(T)

- **Wunsch:** Logik, die Arithmetik beherrscht
- n-Damen Problem benötigt dann (fast) keine weitere Kodierung
 - $X_{i,j} = 0 \vee X_{i,j} = 1$ für alle $X_{i,j}$
- **Wunsch:** Unterstützung weiterer eingebauter Datentypen
- **Lösung:** Benutze SMT! (Satisfiability Modulo Theories)
- **Beobachtung:** SMT ist wie Haskell / OCaml / Scala / Swift / ...

Definition: Eine Theorie T besteht aus

- einer **Signatur** Σ bestehend aus Konstanten-, Funktions- und Prädikatensymbolen
 - einer Menge Ax von **Axiomen** (Sätze der Prädikatenlogik)
-
- Die Symbole in Σ haben **keine** festgelegte Bedeutung
 - Die Bedeutung wird erst durch Ax festgelegt

- **Beispiel (T_{UF}):**

- $\Sigma = \{ =, a, b, c, \dots, f, g, h, \dots \}$

- Axiome Ax :

1. $\forall x . x = x$ (Reflexivität)

2. $\forall x, y . x = y \rightarrow y = x$ (Symmetrie)

3. $\forall x, y, z . x = y \wedge y = z \rightarrow x = z$ (Transitivität)

4. Für jedes n-stellige Funktionssymbol $f \in \Sigma$:

$$\forall x_1, \dots, x_n, y_1, \dots, y_n . \bigwedge_{i=1}^n x_i = y_i \rightarrow f(x_1, \dots, x_n) = f(y_1, \dots, y_n)$$

(Kongruenz)

- (Kongruenz) ist ein **Axiomenschema**, das eine (auch unendliche) Menge von Axiomen repräsentiert

- **Beispiel (Peano Arithmetik):**

- $\Sigma = \{ =, 0, 1, +, * \}$

- Axiome Ax:

1. Axiome der Theorie der uninterpretierten Funktionen

2. $\forall x . \neg(x + 1 = 0)$ (Null)

3. $\forall x, y . x + 1 = y + 1 \rightarrow x = y$ (Nachfolger)

4. $F[0] \wedge (\forall x . F[x] \rightarrow F[x + 1]) \rightarrow \forall x . F[x]$ (Induktion)

5. $\forall x . x + 0 = x$ (Plus Null)

6. $\forall x, y . x + (y + 1) = (x + y) + 1$ (Plus Nachfolger)

7. $\forall x . x * 0 = 0$ (Multiplikation Null)

8. $\forall x, y . x * (y + 1) = (x * y) + x$ (Multiplikation Nachfolger)

- (Induktion) ist ein (unendliches) Axiomenschema

- **Beispiel (Presburger Arithmetik):**

- $\Sigma = \{ =, 0, 1, + \}$ (keine Multiplikation!)

- Axiome Ax:

1. Axiome der Theorie der uninterpretierten Funktionen

2. $\forall x . \neg(x + 1 = 0)$ (Null)

3. $\forall x, y . x + 1 = y + 1 \rightarrow x = y$ (Nachfolger)

4. $F[0] \wedge (\forall x . F[x] \rightarrow F[x + 1]) \rightarrow \forall x . F[x]$ (Induktion)

5. $\forall x . x + 0 = x$ (Plus Null)

6. $\forall x, y . x + (y + 1) = (x + y) + 1$ (Plus Nachfolger)

- (Induktion) ist ein (unendliches) Axiomenschema

- In der Presburger Arithmetik sind Konzepte wie Teilbarkeit oder Primzahlen nicht formalisierter; ansonsten ähnlich der Peano-Arithmetik

- **Beispiel (Lineare Arithmetik der ganzen Zahlen):**

- $\Sigma = \{ =, >, +, -, \dots, -2, -1, 0, 1, 2, \dots \}$ (keine Multiplikation!)

- Axiome Ax:

1. Axiome der Theorie der uninterpretierten Funktionen
2. Weitere Axiome ähnlich Presburger Arithmetik

- **Beispiel (\mathcal{T}_A):**

- $\Sigma = \{ =, \text{read}, \text{write} \}$

- Axiome Ax:

1. (Reflexivität), (Symmetrie), (Transitivität) wie in der Theorie der uninterpretierten Funktionen
2. $\forall a, i, j . i = j \rightarrow \text{read}(a, i) = \text{read}(a, j)$ (Array Kongruenz)
3. $\forall a, v, i, j . i = j \rightarrow \text{read}(\text{write}(a, i, v), j) = v$ (Read-Over-Write 1)
4. $\forall a, v, i, j . i \neq j \rightarrow \text{read}(\text{write}(a, i, v), j) = \text{read}(a, j)$
(Read-Over-Write 2)
5. $\forall a, b . a = b \leftrightarrow (\forall i . \text{read}(a, i) = \text{read}(b, i))$ (Extensionalität)

- **Definition (T-erfüllbar):**

Eine Σ -Formel F ist *T-erfüllbar* gdw. $Ax \cup \{ F \}$ erfüllbar ist.

- **Definition (T-gültig):**

Eine Σ -Formel F ist *T-gültig* gdw. $Ax \models F$.

- **Definition (Entscheidbarkeit):**

Eine Theorie T ist *entscheidbar* gdw. die T-Erfüllbarkeit für jede Σ -Formel entschieden werden kann.

- **Definition (Fragment):**

Ein Fragment einer Theorie T ist eine syntaktisch eingeschränkte Teilmenge aller Σ -Formeln.

- **Beispiel:** quantorenfreies Fragment (QFF)

- **Definition (Fragment):**

Ein Fragment einer Theorie T ist eine syntaktisch eingeschränkte Teilmenge aller Σ -Formeln.

- **Beispiel:** quantorenfreies Fragment (QFF)

- **Definition (Entscheidbarkeit Fragment):**

Ein Fragment einer Theorie T ist entscheidbar gdw. die T -Erfüllbarkeit für jede Σ -Formel des Fragments entschieden werden kann.

Theorie	Entscheidbar?	QFF entscheidbar?
Uninterpretierte Funktionen	✗	✓
Peano Arithmetik	✗	✗
Presburger Arithmetik	✓	✓
Lineare Arithmetik ganzer Zahlen	✓	✓
Arrays	✗	✓

- Im folgenden: nur quantoren-freie Σ -Formeln
- Benutze DPLL um **Konjunktionen von T-Literalen** zu erzeugen
 - diese T-Literale erfüllen das **Boolesche Skelett** der Formel
- Prüfe, ob die Konjunktion **konsistent** in T ist
 - Falls ja, extrahiere ein T-Modell
 - Falls nein, **schließe** diesen „Kandidaten“ **aus**

Lineare Arithmetik ganzer Zahlen:

$$y \geq 1 \wedge (x < 0 \vee y < 1) \wedge (x \geq 0 \vee y < 0)$$

Boolesches Skelett:

$$A \wedge (B \vee C) \wedge (D \vee E)$$

Modell für das Boolesche Skelett:

$$A, C, E$$

Entsprechende T -Literale:

$$y \geq 1, y < 1, y < 0$$

Inkonsistente Konjunktion von T -Literalen:

$$y \geq 1 \wedge y < 1 \wedge y < 0$$

Schließe diesen "Kandidaten" aus:

$$\neg(y \geq 1) \vee \neg(y < 1)$$

Lineare Arithmetik ganzer Zahlen:

$$y \geq 1 \wedge (x < 0 \vee y < 1) \wedge (x \geq 0 \vee y < 0) \wedge [\neg(y \geq 1) \vee \neg(y < 1)]$$

Boolesches Skelett:

$$A \wedge (B \vee C) \wedge (D \vee E) \wedge (\neg A \vee \neg C)$$

Modell für das Boolesche Skelett:

$$A, \neg C, B, D$$

Entsprechende T -Literale:

$$y \geq 1, \neg(y < 1), x < 0, x \geq 0$$

Inkonsistente Konjunktion von T -Literalen:

$$y \geq 1 \wedge \neg(y < 1) \wedge x < 0 \wedge x \geq 0$$

SchlieÙe diesen "Kandidaten" aus:

$$\neg(x < 0) \vee \neg(x \geq 0)$$

Lineare Arithmetik ganzer Zahlen:

$$y \geq 1 \wedge (x < 0 \vee y < 1) \wedge (x \geq 0 \vee y < 0) \wedge [\neg(y \geq 1) \vee \neg(y < 1)] \wedge [\neg(x < 0) \vee \neg(x \geq 0)]$$

Boolesches Skelett:

$$A \wedge (B \vee C) \wedge (D \vee E) \wedge (\neg A \vee \neg C) \wedge (\neg B \vee \neg D)$$

Modell für das Boolesche Skelett:

$$A, \neg C, B, \neg D, E$$

Entsprechende T -Literale:

$$y \geq 1, \neg(y < 1), x < 0, \neg(x \geq 0), y < 0$$

Inkonsistente Konjunktion von T -Literalen:

$$y \geq 1 \wedge \neg(y < 1) \wedge x < 0 \wedge \neg(x \geq 0) \wedge y < 0$$

SchlieÙe diesen "Kandidaten" aus:

$$\neg(y \geq 1) \vee \neg(y < 0)$$

Lineare Arithmetik ganzer Zahlen:

$$y \geq 1 \wedge (x < 0 \vee y < 1) \wedge (x \geq 0 \vee y < 0) \wedge [\neg(y \geq 1) \vee \neg(y < 1)] \wedge [\neg(x < 0) \vee \neg(x \geq 0)] \wedge [\neg(y \geq 1) \vee \neg(y < 0)]$$

Boolesches Skelett:

$$A \wedge (B \vee C) \wedge (D \vee E) \wedge (\neg A \vee \neg C) \wedge (\neg B \vee \neg D) \wedge (\neg A \vee \neg E)$$

Boolesches Skelett hat kein Modell

\implies Ursprüngliche Formel hat kein T -Modell

- Erweiterung von abstraktem DPLL
- **Parametrisiert** durch eine Theorie T
- Benötigt einen T-Solver, der die **Konsistenz** einer Konjunktion von T-Literalen entscheiden kann
- **Zustände:**
 - `Fail`
 - $M \parallel F$
 - F: Quantorenfreie Σ -Formel in CNF
 - M: Liste von T-Literalen
 - Konsistenz von M wird durch T-Solver überprüft
- **Regeln:**
 - $M \parallel F \Rightarrow_T M' \parallel F'$ oder $M \parallel F \Rightarrow_T \text{Fail}$

- **UnitPropagate:**

$$M \parallel F, C \vee l \implies_T Ml \parallel F, C \vee l \quad \text{falls} \begin{cases} M \models \neg C \\ l \text{ undefiniert in } M \end{cases}$$

- **Decide:**

$$M \parallel F \implies_T Ml^d \parallel F \quad \text{falls} \begin{cases} l \text{ oder } \neg l \text{ kommt in } F \text{ vor} \\ l \text{ undefiniert in } M \end{cases}$$

- **Backtrack:**

$$Ml^d N \parallel F, C \implies_T M\neg l \parallel F, C \quad \text{falls} \begin{cases} Ml^d N \models \neg C \\ N \text{ enthält kein Literal} \\ \text{der Form } k^d \end{cases}$$

- **Fail:**

$$M \parallel F, C \implies_T \text{Fail} \quad \text{falls} \begin{cases} M \models \neg C \\ M \text{ enthält kein Literal der Form } l^d \end{cases}$$

- **Theory-Inconsistent:**

$$M \parallel F \implies_T \emptyset \parallel F, \neg l_1 \vee \dots \vee l_n \quad \text{falls} \begin{cases} \{l_1, \dots, l_n\} \subseteq M \\ \neg l_1 \vee \dots \vee \neg l_n \text{ T-g\"ultig} \end{cases}$$

- **Theory-Propagate:**

$$M \parallel F \implies_T Ml \parallel F \quad \text{falls} \begin{cases} M \vDash_T l \\ l \text{ oder } \neg l \text{ kommt in } F \text{ vor} \\ l \text{ undefiniert in } M \end{cases}$$

- **Satz:**

Alle Ableitungen $\emptyset \parallel F \Longrightarrow_T S_1 \Longrightarrow_T \dots$ sind endlich.

- **Satz:**

Sei $\emptyset \parallel F \Longrightarrow_T S_1 \Longrightarrow_T \dots \Longrightarrow_T S$ eine maximale Ableitung. Dann gilt:

- F ist unerfüllbar gdw. $s = \text{Fail}$
- Falls S die Form $M \parallel F$ hat, dann hat M ein T-Modell und jedes T-Modell von M ist auch ein T-Modell von F

- Bereits gesehen:
 - T-Solver für lineare Arithmetik ganzer Zahlen
- In den nächsten Vorlesungsstunden: T-Solver für
 - Uninterpretierte Funktionen
 - Bit-Vektoren
 - Arrays
- Kombination von T-Solvern

- **Mehrsortige Logik (many-sorted logic)**

- Es gibt eine Menge $S = \{ T_1, T_2, \dots \}$ von (disjunkten) Sorten („Datentypen“, Wertebereiche)
 - z.B. `Int`, `Real`, `BitVector`, `Array(T1, T2)`
- Variablen, Relationssymbole und Funktionssymbole bekommen neben der Stelligkeit noch eine Typ-Signatur
 - z.B. `+`: `Int x Int -> Int`
`read`: `Array(Int, Real) x Int -> Real`
- Dabei sind auch Sorten-Konstruktoren (wie z.B. `Array`) möglich
 - Polymorphe Typen
- Terme und Formeln müssen dann Typ-konform sein
- Notation: „`t : T`“ steht für „Term `t` besitzt Typ `T`“

- Definiert Standard-Format für SMT-Formeln und Befehle für SMT-Solver
- Aktuelle Version: SMT-LIB 2.6 (<http://smtlib.cs.uiowa.edu>)



Language

The SMT-LIB Standard: Version 2.6, by Clark Barrett, Pascal Fontaine, and Cesare Tinelli.

Latest official release of Version 2.6 of the SMT-LIB standard. [[pdf](#) | [bib](#)]

The SMT-LIB Standard: Version 2.5, by Clark Barrett, Pascal Fontaine, and Cesare Tinelli.

Latest official release of Version 2.5 of the SMT-LIB standard. [[pdf](#) | [bib](#)]

Previous releases: [2015-05-28](#);

The SMT-LIB v2 Language and Tools: A Tutorial, by David R. Cok.

A tutorial on Version 2.0 of the language and on a number of SMT-LIB tools developed by the author. [[pdf](#)]

[Home](#)

[About](#)

[News](#)

[Standard](#)

[Language](#)

[Theories](#)

[Logics](#)

[Examples](#)

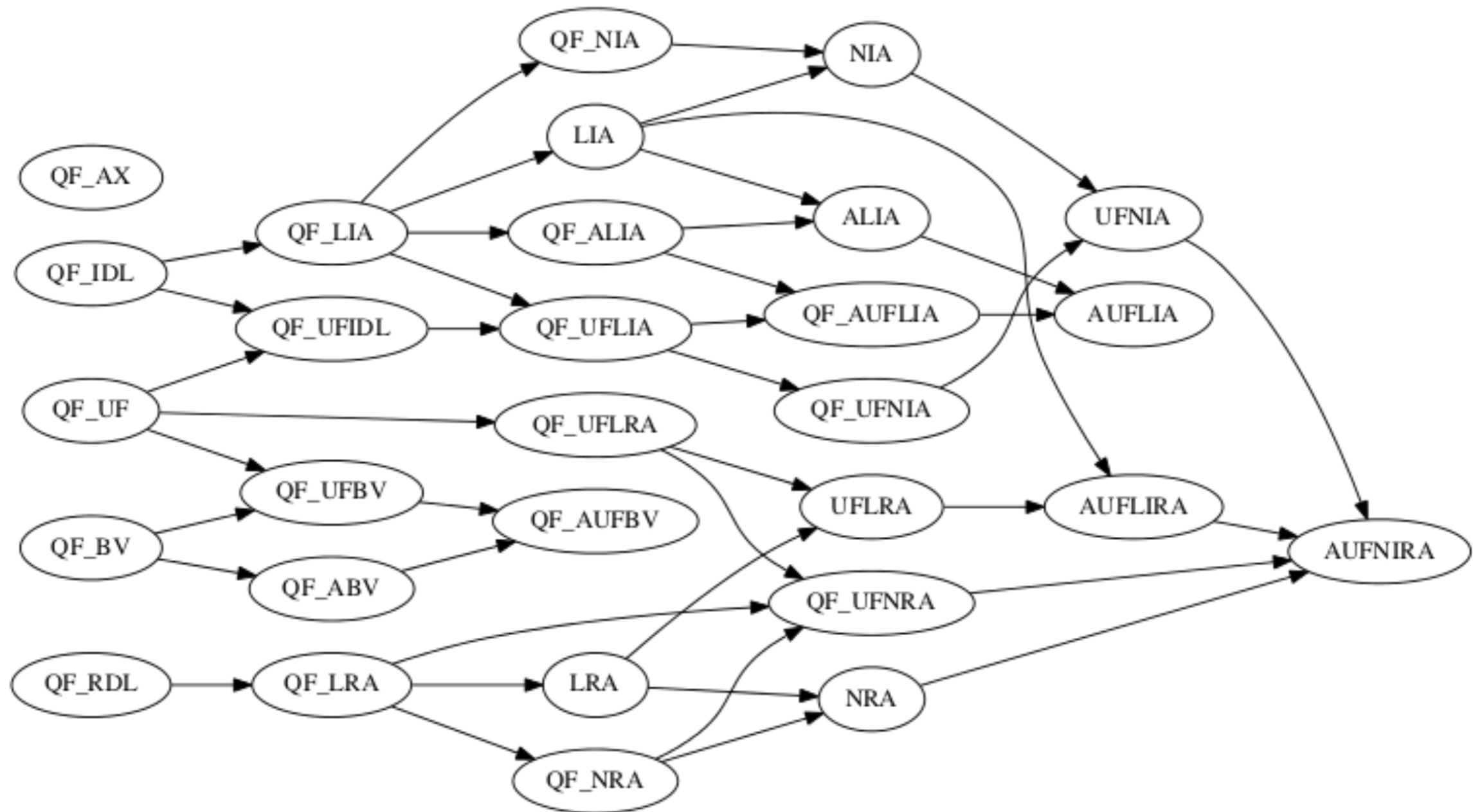
[Benchmarks](#)

- **SMT-LIB definiert:**
 - **Syntax für SMT-Formeln**
 - Lisp-ähnlich
 - **Logiken**
 - Entsprechen unseren Theorie-Fragmenten
 - z.B. QF_LIA für die quantorenfreie Logik der linearen ganzzahligen Arithmetik
 - **Theorien**
 - Entsprechen grob unserer Definition von Theorien, mehrsortige Logik
 - **Kommandos für SMT-Solver, z.B.:**

```
(set-logic QF_LIA)
(check-sat)
(get-model)
```

SMT-LIB: QF_LIA Beispiel

```
(set-option :produce-models true)
(set-logic QF_LIA)
(declare-const x1 Int)
(declare-const x2 Int)
(declare-const x3 Int)
(declare-const x4 Int)
(declare-const x5 Int)
(declare-const x6 Int)
(declare-const x7 Int)
(declare-const x8 Int)
(declare-const x9 Int)
(declare-const x10 Int)
(assert
  (and
    (= (+ (* -68 x6) (* 31 x7) (* 44 x8)) -3793)
    (= (+ (* -28 x1) (* -20 x3) (* 20 x4) (* -63 x6) (* -72 x8) (* 75 x9) (* 77 x10))
      -13264)
    (= (+ (* -67 x7) (* -63 x9) (* -30 x10)) 4340)
    (= (+ (* 48 x2) (* 40 x8) (* -52 x10)) -2940)
    (= (+ (* 59 x1) (* -53 x2) (* -72 x4) (* 72 x5) (* -26 x6) (* -97 x7) (* 50 x8)
      (* 22 x9) (* -73 x10))
      557)
  ))
(check-sat)
(get-model)
```



A → B bedeutet, dass jede Formel von A auch eine Formeln von B ist