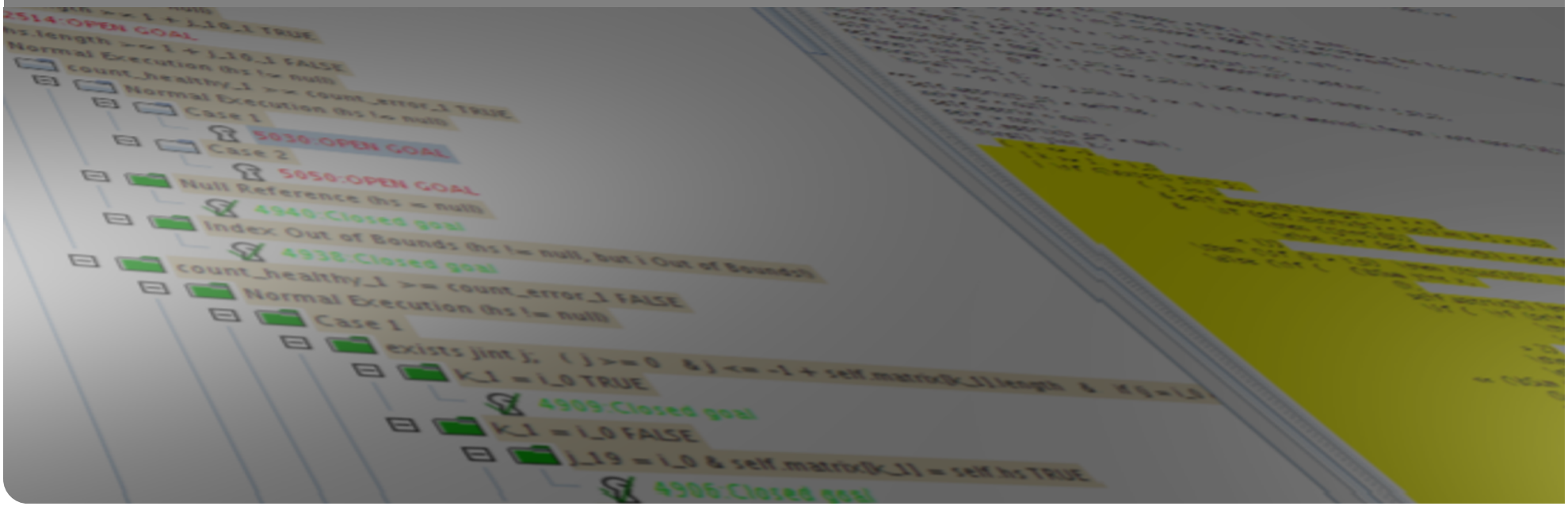
# Formale Systeme

## Prof. Dr. Bernhard Beckert, WS 2018/2019

SMT-LIB

KIT – Institut für Theoretische Informatik



www.kit.edu

# SMT-LIB

- Standardisiertes Schnittstelle für SMT Solvers.
- Aktuelle Version: SMT-LIB v2.6 

## Solvers

Alt-Ergo, AProVE, Boolector, CVC4, MathSAT 5, OpenSMT 2, raSAT, SMTInterpol, SMT-RAT, STP, veriT, Yices 2, Z3
**Empfehlung: Z3**

- Online-Tool: `https://rise4fun.com/z3`
- Source: `https://github.com/Z3Prover/z3`
- Binaries: `https://github.com/Z3Prover/z3/releases`

# SMT-LIB

- Standardisiertes Schnittstelle für SMT Solvers.
- Aktuelle Version: SMT-LIB v2.6
- Basiert auf S-Exp (Lisp-Syntax)

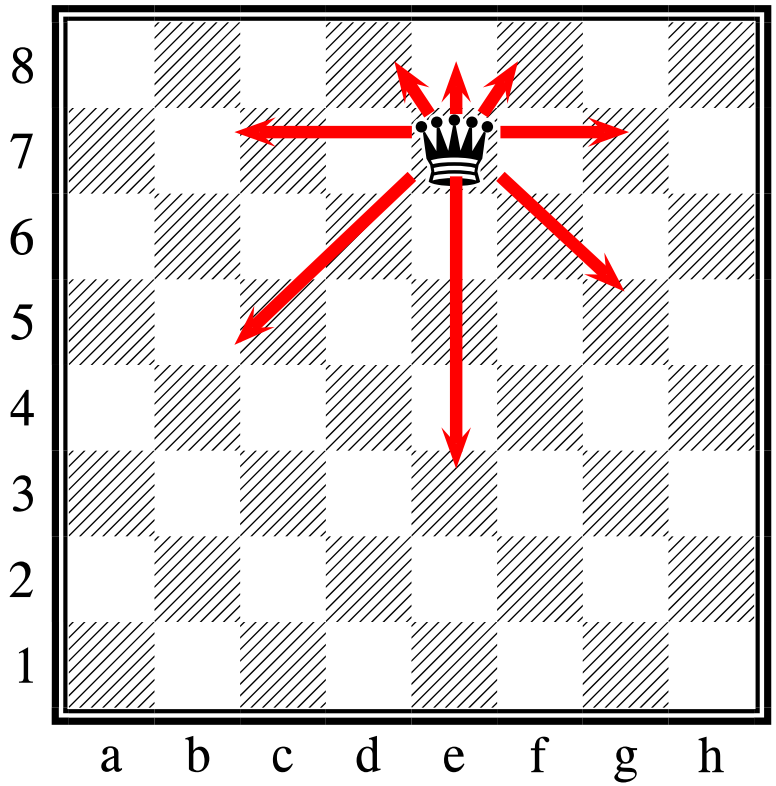
```
1  ; This example illustrates basic arithmetic and
2  ; uninterpreted functions
3
4  (declare-fun x () Int)
5  (declare-fun y () Int)
6  (declare-fun z () Int)
7  (assert (>= (* 2 x) (+ y z)))
8  (declare-fun f (Int) Int)
9  (declare-fun g (Int Int) Int)
10 (assert (< (f x) (g x x)))
11 (assert (> (f y) (g x x)))
12 (check-sat)
13 (get-model)
```

# $n$-Dame Probleme

Man plaziere $n$ Damen so auf einem $n \times n$ Schachbrett, dass sie sich gegenseitig nicht bedrohen.

# $n$-Dame Probleme

Man plaziere $n$ Damen so auf einem $n \times n$ Schachbrett, dass sie sich gegenseitig nicht bedrohen.



# $n$-Dame Probleme in SMT-LIB

## 1. Schritt: Signatur

- Konstante: $N \in \mathbb{Z}$
- Schachbrett: $v : \mathbb{Z} \times \mathbb{Z} \to \mathbb{B}$

# $n$-Dame Probleme in SMT-LIB

## 1. Schritt: Signatur

- Konstante: $N \in \mathbb{Z}$
- Schachbrett: $v : \mathbb{Z} \times \mathbb{Z} \to \mathbb{B}$

### SMT

```
1 (define-fun N () Int 5)
2 (declare-fun v (Int Int) Bool)
```

# $n$-Dame Probleme in SMT-LIB

## 1. Schritt: Signatur

- Konstante: $N \in \mathbb{Z}$
- Schachbrett: $v : \mathbb{Z} \times \mathbb{Z} \to \mathbb{B}$

### SMT

```
1 (define-fun N () Int 5)
2 (declare-fun v (Int Int) Bool)
```

### Syntax: `define/declare-fun`

$$(\texttt{declare-fun}\ \textit{name}\ (\sigma_1 \ldots \sigma_n)\ \sigma)$$

$$(\texttt{define-fun}\ \textit{name}\ ((x_1\ \sigma_1) \ldots (x_n\ \sigma_n))\ \sigma\ t)$$

# $n$-Dame Probleme in SMT-LIB

## 2. Schritt: Formalisierung

**Jede Spalte hat mind. eine Königin:**

$$\forall c \in \mathbb{Z}.\ (1 \leq c \leq N \rightarrow \exists r \in \mathbb{Z}.\ (1 \leq r \leq N \wedge v(r, c)))$$

# $n$-Dame Probleme in SMT-LIB

## 2. Schritt: Formalisierung

### Jede Spalte hat mind. eine Königin:

$$\forall c \in \mathbb{Z}.\ (1 \leq c \leq N \rightarrow \exists r \in \mathbb{Z}.\ (1 \leq r \leq N \wedge v(r, c)))$$

### SMT

```
1 (assert
2   (forall ((col Int))
3       (=> (and (>= col 1) (<= col N))
4            (exists ((row Int))
5               (and (>= row 1) (<= row N)
6                    (v row col))))))
```

```
1  (assert
2      (forall ((r Int) (c Int))
3          (=> (v r c) (and
4              ;; One queen per row
5              (forall ((s Int))
6                  (=> (distinct r s) (not (v s c))))
7
8              ;; One queen per column
9              (forall ((d Int))
10                 (=> (distinct c d) (not (v r d))))
11
12             ;; Diagonal conflicts
13             (forall ((s Int) (d Int))
14                 (=> (and (distinct r s) (distinct
                       c d)
15                           (= (- r s) (- c d)))
16                     (not (v s d)))))))))
```

# Ausdrücke: Terme und Formeln

Quantifier/Binder

$$(\texttt{forall}\ (\ (x_1\ \sigma_1)\ldots(x_n\ \sigma_n)\ )\ t)$$
$$(\texttt{exists}\ (\ (x_1\ \sigma_1)\ldots(x_n\ \sigma_n)\ )\ t)$$
$$(\texttt{let}\ (\ (x_1\ t_1)\ldots(x_n\ t_n)\ )\ t)$$

Operatoren und Funktionen

$$(op\ t_1\ \ldots\ t_n)$$

- Bool'sche Operationen:
  $op \in \{\texttt{not}, \texttt{and}, \texttt{or}, =, \texttt{distinct}, =>\}$
- Arithmetische Operatoren (Int, Real):
  $op \in \{+, *, /, -, \texttt{distinct}, =, \ldots\}$
- Arithmetische Operatoren (Bitvektor):
  $op \in \{\texttt{bvnot}, \texttt{bvand}, \texttt{bvmul}, \ldots\}$

  `http://smtlib.cs.uiowa.edu/logics-all.shtml#QF_BV`
- Fallunterscheidung (`ite` *cond then else*)

# $n$-Dame Probleme in SMT-LIB

## 3. Schritt: Erfüllbarkeit überprüfen

```
1 (check-sat)
2 (get-model)
```

### Befehle

- (`assert` $t$) – Hinzufügen der Bedingung $t$
- (`check-sat`) – Erfüllbarkeit überprüfen
- (`get-model`) – Ausgabe des Modells
- (`get-value` ($t_1$ ... $t_n$) ) – Werte berechnen
- (`push`)/(`pop`) – Inkrementelles Lösen
- `(set-logic QF_LIA)` – Logik/Fragment
- `(set-option :option value)` – Einstellungen setzen

# *n*-Dame Probleme in SMT-LIB

## 4. Schritt: Ausführen

```
$ z3 -st queens.smt2
sat
(model
  (define-fun k!513 ((x!0 Int)) Int
    (let ((a!1 (ite (>= x!0 14) (ite (>= x!0 22) (ite (>= x!0 24) 24 22) 14) 11)))
    (let ((a!2 (ite (>= x!0 9) (ite (>= x!0 10) (ite (>= x!0 11) a!1 10) 9) 8)))
    (let ((a!3 (ite (>= x!0 6) (ite (>= x!0 7) (ite (>= x!0 8) a!2 7) 6) 5)))
    (let ((a!4 (ite (>= x!0 3) (ite (>= x!0 4) (ite (>= x!0 5) a!3 4) 3) 2)))
    (let ((a!5 (ite (>= x!0 0) (ite (>= x!0 1) (ite (>= x!0 2) a!4 1) 0) (- 5))))
      (ite (>= x!0 (- 7)) (ite (>= x!0 (- 5)) a!5 (- 7)) (- 14)))))))))
  (define-fun v!515 ((x!0 Int) (x!1 Int)) Bool
    (ite (and (= x!0 8) (= x!1 2)) true
    (ite (and (= x!0 3) (= x!1 7)) true
    (ite (and (= x!0 7) (= x!1 9)) true
    (ite (and (= x!0 10) (= x!1 5)) true
    (ite (and (= x!0 1) (= x!1 10)) true
    (ite (and (= x!0 2) (= x!1 8)) true
    (ite (and (= x!0 9) (= x!1 6)) true
    (ite (and (= x!0 5) (= x!1 4)) true
    (ite (and (= x!0 6) (= x!1 1)) true
    (ite (and (= x!0 4) (= x!1 3)) true
      false)))))))))))
  (define-fun v ((x!0 Int) (x!1 Int)) Bool
    (v!515 (k!513 x!0) (k!513 x!1)))
  (define-fun row!0!516 ((x!0 Int)) Int
    ...)
)
```

# *n*-Dame Probleme in SMT-LIB

## 4. Schritt: Ausführen

```
(get-value ((v 1 1) (v 2 1) (v 3 1) (v 4 1)
            (v 5 1) (v 6 1) (v 7 1) (v 8 1)
            (v 9 1) (v 10 1)))
```

```
(((v 1 1) false)
 ((v 2 1) false)
 ((v 3 1) false)
 ((v 4 1) false)
 ((v 5 1) false)
 ((v 6 1) true)
 ((v 7 1) false)
 ((v 8 1) false)
 ((v 9 1) false)
 ((v 10 1) false))
```