Examples for the test ... how to solve ...

SATisfyability

Aussagenlogik
Propositional logic:

① $\neg p \wedge (\neg q \vee q \vee \neg q) \wedge (p \vee \neg r) \wedge q \wedge (\neg t \vee r \vee \neg s) \wedge (s \vee s) \wedge t$

↓ false ↓ true ↓ true

Für SAT muss alles true sein.

Als erstes die einzelnen anschauen

Dann den Rest berechnen.

Prädikatenlogik
First Order logic:

Truth table.

equality logic:

$$i = j \wedge k = \ell \wedge k \neq m \wedge \ell \neq i \wedge m = i \wedge f = j$$

① find equivalence classes

$\{i, j\}, \{k, l\}, \{m, i\}, \{f, j\}$

② merge equivalence classes Schrittweise

$\{i, j\}, \{k, l\}, \{m, i\}, \{f, j\}$

$\{i, j, m\}, \{k, l\}, \{f, j\}$

$\{i, j, m, f\}, \{k, l\}$

$(k \neq m) \wedge (l \neq i) \Rightarrow$ SAT

Examples for the test ... how to solve ...

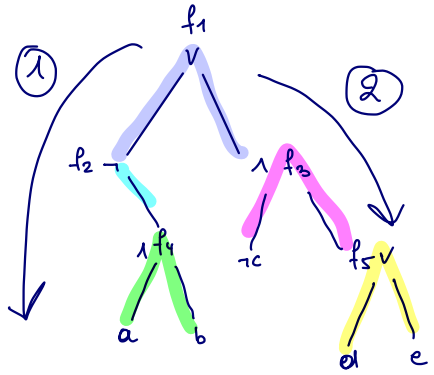
Tseitin Transformation

Tseitin Transformation

$$\underbrace{\neg}_{f_2}(\underbrace{a \wedge b}_{f_4}) \vee (\neg \underbrace{c \wedge (\underbrace{d \vee e}_{f_5})}_{f_3})$$

(f_1 spans the whole formula)

1) Einteilen in Formelteile & Baum zeichnen



$$f_1 \wedge (f_1 \leftrightarrow f_2 \vee f_3) \wedge (\neg f_2 \leftrightarrow f_4) \wedge (f_4 \leftrightarrow a \wedge b) \wedge (f_3 \leftrightarrow \neg c \wedge f_5) \wedge (f_5 \leftrightarrow d \vee e)$$

$a \wedge b \rightarrow (\bar{f} \vee a) \wedge (\bar{f} \vee b) \wedge (\bar{a} \vee \bar{b} \vee f)$

$(a \vee b) \rightarrow (f \vee \bar{a}) \wedge (f \vee \bar{b}) \wedge (a \vee b \vee \bar{f})$

$\bar{a} \rightarrow (\bar{f} \vee \bar{a}) \wedge (f \vee a)$

"Three Steps"

3 Pfeile, 4 Zustände

2) Schrittweise mit Baum umschreiben

- $f_1 \Leftrightarrow f_2 \vee f_3 \equiv (f_1 \vee \bar{f_2}) \wedge (f_1 \vee \bar{f_3}) \wedge (\bar{f_1} \vee f_2 \vee f_3)$
- $f_2 \Leftrightarrow \neg f_4 \equiv (\bar{f_2} \vee \bar{f_4}) \wedge (f_2 \vee f_4)$
- $f_3 \Leftrightarrow \neg c \wedge f_5 \equiv (\bar{f_3} \vee \bar{c}) \wedge (\bar{f_3} \vee f_5) \wedge (f_3 \vee \bar{c} \vee \bar{f_5})$
- $f_4 \Leftrightarrow a \wedge b \equiv (\bar{f_4} \vee a) \wedge (\bar{f_4} \vee b) \wedge (f_4 \vee \bar{a} \vee \bar{b})$
- $f_5 \Leftrightarrow d \vee e \equiv (f_5 \vee \bar{d}) \wedge (f_5 \vee \bar{e}) \wedge (\bar{f_5} \vee e \vee d)$

Examples for the test ... how to solve ...

Hoare Logic

5 Rules:

- "Skip" rule: Condition holds afterwards (Genauso wie vorher) raufschreiben
- "assignment" rule: (reversed: Starting with PostCondition – What must be satisfied)
- "composition" rule: With two hoare triples and Precondition triple 1 = Postcondition triple 2, (patch / merge) these together
- "conditional" rule: replace precoditions, reasoning about branches individually; P, Q must be perfect match!
- "Consequence" rule: Combining FOL with Hoare — weaken precondition / strengthen post condition; strengthen pre-condition / weaken post condition
- "while loops" rule: Statement doesn't change P, P holds throughout loop (always require / inductive invariant)

```
{true}

x := n;

y := m;

if (x > y) {
```

$(x > y) \wedge (x-y) = (n-m) \supset (x-y) = |n-m|$

$(x-y) = (n-m)$

```
  t := x;
```

$(t-y) = (n-m)$

```
  x := y;
```

$(t-x) = (n-m)$

```
  y := t;
```

$(y-x) = (n-m)$

```
} else {
```

Bedingung $(x \leq y) \wedge (y-x) = (n-m) \supset (x-y) = (n-m)$

```
  skip;
```

$(y-x) = (n-m)$

```
}
```

$(y-x) = (n-m)$

```
while (x != 0) {
```

$|(y-1) - (x-1)| = |n-m|$

```
  x := x - 1;
```

$|(y-1) - x| = |n-m|$

```
  y := y - 1;
```

$|y-x| = |n-m|$

```
}
```

loop Bedingung $(x == 0) \wedge |x-y| = |n-m|$

$\{(y = |n-m|)\}$

Examples for the test ... how to solve ...

Invariants

non-inductive invariant: feste unveränderbare Variable

- $(r \geq -1)$

  before $r=-1$ $i=-2$ $n=4$

```
int r = -1;
int i =-2;
while ((i <= 4) && (r == -1)) {
  if (i * i == 4)
    r = i;
  else
    i = i + 1;
}
```

  after $r=-2$ ↯ $i=-2$ $n=4$

Begründen mit
A) Counterexample for induction
B) Explain "invariant"

inductive invariant:

n number of loop iterations
holds in every loop iteration
if it holds for one it holds for all

} Begründung mit
A) Base case
B) Induction (STEP)

- $(r^2=n) \vee (r<0)$

  if branch: $i^2=n$ holds and $r=i \Rightarrow r^2=n$

  else branch: holds

```
int r = -1;
int i = 0;
while ((i <= n) && (r == -1)) {
  if (i * i == n)   holds
    r = i;
  else
    i = i + 1;
}
```

- $(r \leq 1) \vee (r \neq n)$

  if branch: $i^2=n=r^2$ holds &
  $(r \neq n) \Leftrightarrow (r \leq 1)$ ✓

  else branch: holds

neither – Begründung: When is it violated?

- $(i^2 \leq n)$

  $n=2$ holds before

i	i^2
0	0
1	1
2	2 → $i^2 > n$ ↯

```
i=0
while ((i <= 2) && (r == -1)) {
  if (i * i == 2)
    r = i;
  else
    i = i + 1;
}
```

Examples for the test ... how to solve ...

Control Flow (Coverage)

path coverage: (every path = equivalence class)

statement coverage: every statement once (branches, ...)

branch coverage: geht ins if egal ob T/L

decision coverage: boolean expression

Branch coverage implies decision coverage
- if "decision" means Boolean expressions at branching points only

Decision coverage is stronger than branch coverage
- if "branch" doesn't include unconditional jumps
- if "decision" refers to *all* Boolean expressions

Often branch and decision outcome are used synonymously

condition | coverage: every subexpression, condition, atom

man testet die "Härtefälle"

(A||B) &&C

0 0 0
1 1 1

condition|decision coverage: all conditions | decisions fulfilled

man testet genau die Werte, die die boolean decision verändern

(A||B) &&C

0 1 1
1 0 1
1 1 0
0 0 1

modified condition|decision coverage: all conditions | decisions affect each other

man testet alle

```
bool range_check (unsigned m, unsigned n) {
  if (m > n) {
    unsigned t = m;
    m = n;
    n = t;
  }
  bool result = false;
  bool tmp = true;
  unsigned i = m;
  while ((i > 0) && (i < n)) {
    i = i + 1;
    if (i % m == 0)
        result = result || tmp;
  }
  return result;
}
```

Inputs		Outputs
m	n	result
3	7	true
1	0	false
2	5	true

equivalence class: condition evaluates to same value.

DATA Flow (defs, p, c ...)

Examples for the test ... how to solve ...

predicate is a statement that may be true or false depending on the values of its variables

computation is the calculation of the values of these variables

all defs: all definitions are used $(dpu(l,x) \cup dcu(l,x))$

int a = 5, a kommt später noch vor

all c uses: all comp uses affected by each def are executed $l' \in dcu(l,x)$ (if emty, do none → bad, so)

berechnete Werte werden verwendet

all p uses: all desicions affected by each def are executed $l' \in dpu(l,x)$

berechnete boolean Werte werden verwendet

all c uses / some p uses: all defs & affect comp then these comps are executed $dcu(l,x)$ & $dcu(l,x) \neq \emptyset \Rightarrow dpu(l,x)$

all p uses / some c uses: all defs & affect desicions then these des are executed (only predicates are used.) $dpu(l,x)$ & $dpu(l,x) \neq \emptyset \Rightarrow dcu(l,x)$

all uses: every use has to be covered

all du-paths: all possible ways of reaching the

Examples for the test ... how to solve ...

Equivalence Classes

alle Punkte bei Funktion einzeln aufschreiben

eg: priority triage(countries travel, symptoms sympt, int age)

Bedingungen für travel einzeln als Gründe angeben

sympt

age

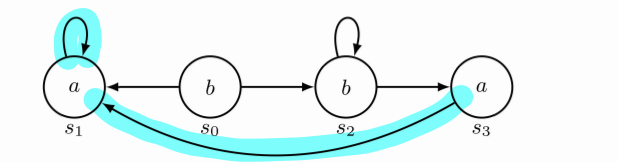
↳ Boundary Testing

Find values that cover equivalence classes (valid and invalid)

Kripke Structure

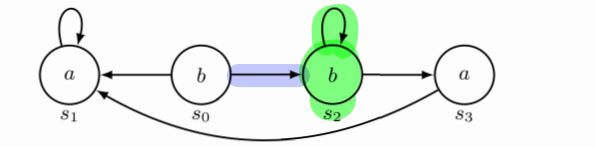
Examples for the test ... how to solve ...

a ∧ X a



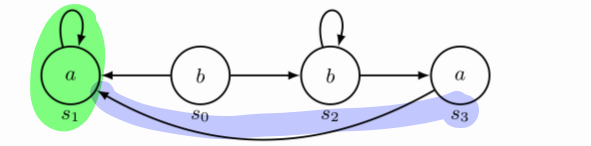
s_1, s_3

EF(EG b)



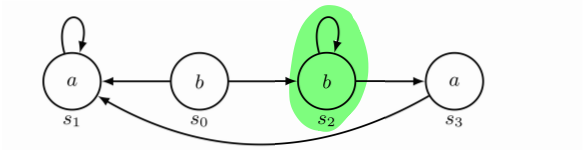
s_0, s_2

FG a



s_1, s_3

EG b



s_0, s_2

LTL Reasons about entire paths

CTL

Nützliche Kombinationen [edit]

- **AG**q: q gilt auf allen (vom Startzustand aus erreichbaren) Pfaden immer
- **GF**q: q gilt unendlich oft
- **FG**q: ab einem Zeitpunkt gilt q immer
- **AF**q: q gilt schlussendlich, also irgendwann
- **EF**q: es ist möglich, dass q gelten wird

- **X**q: Im nächsten Zustand gilt q (*d.h. ne**X**t*)
- **F**q: Nach beliebig vielen Schritten gibt es einen Zustand in dem q gilt *(d.h. irgendwann in der Zukunft - **F**uture)*
- **G**q: q gilt in jedem (folgenden) Zustand (*d.h. immer - **G**lobally*)
- q **U** r: q gilt bis r gilt, wobei r irgendwann gelten muss, möglicherweise auch im ersten Zustand (*d.h. **U**ntil*)

Die Operatoren beziehen sich immer nur auf genau einen Pfad => *immer* bedeutet *immer auf diesem Pfad*