Künstliche Intelligenz - Übung 4

Julian Dobmann, Kai Kruschel

Aufgabe 2

"To prove logical consequence take the axioms and negated conjecture in CNF, and check for unsatisfiability."

Vermutung/Conjecture: I always avoid a kangaroo.

Negierte Vermutung:

```
cnf(avoid_kangaroo,negated_conjecture,
    ( ~ avoided(the_kangaroo) )).
```

Von dieser (negierten) Vermutung ist die Unerfüllbarkeit zu zeigen.

DPLL-Algo

Quelle: [*http://www.cs.miami.edu/~geoff/Courses/CSC648-12S/Content/DPLL.shtml*](http://www.cs.miami.edu/~geoff/Courses/CSC648-12S/Content/DPLL.shtml)

Mit römischen Ziffern sind "Breakpoints" im Algorithmus markiert.

```
def dpll(S)
    # I
    if (S is empty)
        return "satisfiable"
    end

    # II
    while (there is some *unit clause* {p} or some pure literal p)
        # III
        if ({p} and {~p} are in S)
            return "unsatisfiable"
        else
            S = simplify(S,p)
        end
    end

    # IV
    if (S is empty)
        return "satisfiable"
    end

    # V
    # pick some literal q from a shortest clause in S
    q = pick(S)

    # VI
    if (dpll(simplify(S,q)) == "satisfiable")
        return "satisfiable"
    else
        return dpll(simplify(S,~q))
    end
end


def simplify(S, p)
    delete every clause in S containing p
    delete every occurence in S of ~p
    return S
end
```

Anwendung des DPLL-Algorithmus: `dpll(S)`.

```
S = {
  ~ in_house(Cat) | cat(Cat),
  ~ gazer(Gazer) | suitable_pet(Gazer),
  ~ detested(Detested) | avoided(Detested),
  ~ carnivore(Carnivore) | prowler(Carnivore),
```

```
  ~ cat(Cat) | mouse_killer(Cat),
  ~ take_to_me(Taken_animal) | in_house(Taken_animal),
  ~ kangaroo(Kangaroo) | ~ suitable_pet(Kangaroo),
  ~ mouse_killer(Killer) | carnivore(Killer),
  takes_to_me(Animal) | detested(Animal),
  ~ prowler(Prowler) | gazer(Prowler),
  kangaroo(the_kangaroo),
  ~ avoided(the_kangaroo)
}
@I: S is not empty
@II: S contains a unit clause: kangaroo(the_kangaroo)
@III: ~ kangaroo(the_kangaroo) is not in S
    ->  S = simplify(S, kangaroo(the_kangaroo))
        S = {
          ~ in_house(Cat) | cat(Cat),
          ~ gazer(Gazer) | suitable_pet(Gazer),
          ~ detested(Detested) | avoided(Detested),
          ~ carnivore(Carnivore) | prowler(Carnivore),
          ~ cat(Cat) | mouse_killer(Cat),
          ~ take_to_me(Taken_animal) | in_house(Taken_animal),
          ~ suitable_pet(Kangaroo),
          ~ mouse_killer(Killer) | carnivore(Killer),
          takes_to_me(Animal) | detested(Animal),
          ~ prowler(Prowler) | gazer(Prowler),
          ~ avoided(the_kangaroo)
        }
@II: S contains a unit clause: ~ avoided(the_kangaroo)
@III: avoided(the_kangaroo) is not in S
    ->  S = simplify(S, ~ avoided(the_kangaroo))
        S = {
          ~ in_house(Cat) | cat(Cat),
          ~ gazer(Gazer) | suitable_pet(Gazer),
          ~ detested(Detested),
          ~ carnivore(Carnivore) | prowler(Carnivore),
          ~ cat(Cat) | mouse_killer(Cat),
          ~ take_to_me(Taken_animal) | in_house(Taken_animal),
          ~ suitable_pet(Kangaroo),
          ~ mouse_killer(Killer) | carnivore(Killer),
          takes_to_me(Animal) | detested(Animal),
          ~ prowler(Prowler) | gazer(Prowler)
        }
@II: S contains a unit clause: ~ suitable_pet(Kangaroo)
@III: suitable_pet(Kangaroo) is not in S
    ->  S = simplify(S, ~ suitable_pet(Kangaroo))
        S = {
          ~ in_house(Cat) | cat(Cat),
          ~ gazer(Gazer),
          ~ detested(Detested),
```

```
          ~ carnivore(Carnivore) | prowler(Carnivore),
          ~ cat(Cat) | mouse_killer(Cat),
          ~ take_to_me(Taken_animal) | in_house(Taken_animal),
          ~ mouse_killer(Killer) | carnivore(Killer),
          takes_to_me(Animal) | detested(Animal),
          ~ prowler(Prowler) | gazer(Prowler)
        }
@II: S contains a unit clause: ~ gazer(Gazer)
@III: gazer(Gazer) is not in S
    ->  S = simplify(S, ~ gazer(Gazer))
        S = {
          ~ in_house(Cat) | cat(Cat),
          ~ detested(Detested),
          ~ carnivore(Carnivore) | prowler(Carnivore),
          ~ cat(Cat) | mouse_killer(Cat),
          ~ take_to_me(Taken_animal) | in_house(Taken_animal),
          ~ mouse_killer(Killer) | carnivore(Killer),
          takes_to_me(Animal) | detested(Animal),
          ~ prowler(Prowler)
        }
@II: S contains a unit clause: ~ detested(Detested)
@III: detested(Detested) is not in S
    ->  S = simplify(S, ~ detested(Detested))
        S = {
          ~ in_house(Cat) | cat(Cat),
          ~ carnivore(Carnivore) | prowler(Carnivore),
          ~ cat(Cat) | mouse_killer(Cat),
          ~ take_to_me(Taken_animal) | in_house(Taken_animal),
          ~ mouse_killer(Killer) | carnivore(Killer),
          takes_to_me(Animal),
          ~ prowler(Prowler)
        }
@II: S contains a unit clause: ~ prowler(Prowler)
@III: prowler(Prowler) is not in S
    ->  S = simplify(S, ~ prowler(Prowler))
        S = {
          ~ in_house(Cat) | cat(Cat),
          ~ carnivore(Carnivore),
          ~ cat(Cat) | mouse_killer(Cat),
          ~ take_to_me(Taken_animal) | in_house(Taken_animal),
          ~ mouse_killer(Killer) | carnivore(Killer),
          takes_to_me(Animal)
        }
@II: S contains a unit clause: ~ carnivore(Carnivore)
@III: carnivore(Carnivore) is not in S
    ->  S = simplify(S, ~ carnivore(Carnivore))
        S = {
          ~ in_house(Cat) | cat(Cat),
```

```
          ~ cat(Cat) | mouse_killer(Cat),
          ~ take_to_me(Taken_animal) | in_house(Taken_animal),
          ~ mouse_killer(Killer),
          takes_to_me(Animal)
        }
@II: S contains a unit clause: ~ mouse_killer(Killer)
@III: mouse_killer(Killer) is not in S
    ->  S = simplify(S, ~ mouse_killer(Killer))
        S = {
          ~ in_house(Cat) | cat(Cat),
          ~ cat(Cat),
          ~ take_to_me(Taken_animal) | in_house(Taken_animal),
          takes_to_me(Animal)
        }
@II: S contains a unit clause: ~ cat(Cat)
@III: cat(Cat) is not in S
    ->  S = simplify(S, ~ cat(Cat))
        S = {
          ~ in_house(Cat),
          ~ take_to_me(Taken_animal) | in_house(Taken_animal),
          takes_to_me(Animal)
        }
@II: S contains a unit clause: ~ in_house(Cat)
@III: in_house(Cat) is not in S
    ->  S = simplify(S, ~ in_house(Cat))
        S = {
          ~ take_to_me(Taken_animal),
          takes_to_me(Animal)
        }
@II: S contains a unit clause: ~ take_to_me(Taken_animal)
@III: take_to_me(Taken_animal) is in S
    ->  return "unsatisfiable"
```

Das Ergebnis "unsatisfiable" bedeutet, dass die ursprüngliche (nicht negierte) Vermutung erfüllbar ist, also eine logische Konsequenz aus den Axiomen ist.

Aufgabe 1

a) $\forall Z \exists Y \forall X (f(X,Y)) \Leftrightarrow (f(X,Z) \wedge \neg f(X,X))$

Äquivalenz in zwei Implikationen umformen:

$$\forall Z \exists Y \forall X ((f(X,Y) \Rightarrow (f(X,Z) \wedge \neg f(X,X)))$$
$$\wedge (f(X,Z) \wedge \neg f(X,X)) \Rightarrow f(X,Y))$$

Implikationen in Disjunktionen umformen:

$$\forall Z \exists Y \forall X (\neg f(X,Y) \vee (f(X,Z) \wedge \neg f(X,X)))$$
$$\wedge \neg (f(X,Z) \wedge \neg f(X,X)) \vee f(X,Y))$$

Negation in der zweiten Zeile auflösen:

$$\forall Z \exists Y \forall X (\neg f(X,Y) \vee (f(X,Z) \wedge \neg f(X,X))$$
$$\wedge (\neg f(X,Z) \vee f(X,X) \vee f(X,Y)))$$

erste Zeile aufteilen in Konjunktion von zwei Disjunktionen:

$$\forall Z \exists Y \forall X ((\neg f(X,Y) \vee (f(X,Z))$$
$$\wedge (\neg f(X,Y) \vee \neg f(X,X)))$$
$$\wedge (\neg f(X,Z) \vee f(X,X) \vee f(X,Y)))$$

Skolemisieren, so dass $\exists Y \equiv skY(Z)$:

$$(\neg f(X, skY(Z)) \vee f(X,Z))$$
$$\wedge (\neg f(X, skY(Z)) \vee \neg f(X,X))$$
$$\wedge (\neg f(X,Z) \vee f(X,X) \vee f(X, skY(Z)))$$

schließlich bringen wir das ganze in die Mengenschreibweise der KNF:
$\{\neg f(X, skY(Z)) \vee f(X,Z),\ \neg f(X, skY(Z)) \vee \neg f(X,X),\ \neg f(X,Z) \vee f(X,X) \vee f(X, skY(Z))\}$

b)$\forall X \forall Y (q(X,Y)) \Leftrightarrow \forall Z (f(Z,X) \Leftrightarrow f(Z,Y))$

aufteilen:

$$\forall X \forall Y (q(X,Y)) \Leftrightarrow \forall Z (f(Z,X))$$
$$\forall Z (f(Z,X)) \Leftrightarrow f(Z,Y)$$

c) $\forall X \exists Y ((p(X,Y) \Leftarrow \forall X \exists T q(Y,X,T)) \Rightarrow r(Y)$

aufteilen:

$$\forall X \exists Y (\forall X \exists T q(Y,X,T)) \Rightarrow r(Y)$$
$$\wedge (\forall X \exists T q(Y,X,T) \Rightarrow p(X,Y))$$

in Disjunktionen umwandeln:

$$\forall X \exists Y (\neg (\forall X \exists T q(Y,X,T))) \vee r(Y)$$
$$\wedge (\neg (\forall X \exists T q(Y,X,T)) \vee p(X,Y))$$

Quantoren nach außen ziehen:

$$\forall X \exists Y((\exists X \forall T(\neg q(Y,X,T) \lor r(Y))$$
$$\land(\exists X \forall T(\neg q(Y,X,T) \lor p(X,Y)))$$

d)

Aufgabe 3

gegeben

$$V = \{X, Y\}$$
$$F = \{vater\_von/1, mutter\_von/, max/0\}$$
$$P = \{verheiratet/2\}$$

gesucht

Herbrand Universum

$$\begin{aligned}
&max,\\
&vater_von(max),\\
&vater_von(vater_von(max))\\
&...\\
&mutter_von(max),\\
&mutter_von(mutter_von(max))\\
&...\\
&vater_von(mutter_von(max)),\\
&vater_von(vater_von(mutter_von(max))),\\
&mutter_von(vater_von(max)),\\
&mutter_von(vater_von(vater_von(max))),\\
&...\\
&mutter_von(vater_von(mutter_von(vater_von(mutter_von(...)))))
\end{aligned}$$

Herbrand Basis

$$\begin{aligned}
&verheiratet(max, vater_von(max)),\\
&verheiratet(max, mutter_von(max)),\\
&verheiratet(vater_von(max), mutter_von(max)),\\
&verheiratet(vater_von(vater_von(max)), mutter_von(vater_von(max))),\\
&...
\end{aligned}$$

Herbrand Interpretation

D

D ist das Herbrand Universum

F

Die Identitätsfunktion:

$$\begin{aligned} max &\rightarrow max, \\ vater\_von(max) &\rightarrow vater\_von(max), \\ &... \end{aligned}$$

R

weist den Elementen der Herbrand Basis Wahrheitswerte zu:

$$\begin{aligned} verheiratet(max, vater\_von(max)) &\rightarrow FALSE, \\ verheiratet(vater\_von(max), mutter\_von(max)) &\rightarrow TRUE, \\ verheiratet(vater\_von(vater\_von(max)), mutter\_von(vater\_von(max))) &\rightarrow TRUE, \\ &... \end{aligned}$$