Znalostní agenti I.
(7. přednáška)

Znalostní agenti

- svět je často částečně pozorovatelný

Znalostní agenti

- svět je často částečně pozorovatelný
- inteligentní agenti se rozhodují na základě dostupných znalostí

Znalostní agenti

- svět je často částečně pozorovatelný
- inteligentní agenti se rozhodují na základě dostupných znalostí
- inteligentní agenti by si měli umět tyto znalosti rozšiřovat

Znalostní agenti

- svět je často částečně pozorovatelný
- inteligentní agenti se rozhodují na základě dostupných znalostí
- inteligentní agenti by si měli umět tyto znalosti rozšiřovat
  - pozorováním světa

Znalostní agenti

- svět je často částečně pozorovatelný
- inteligentní agenti se rozhodují na základě dostupných znalostí
- inteligentní agenti by si měli umět tyto znalosti rozšiřovat
  - pozorováním světa
  - odvozováním z již získaných znalostí

Dračí doupě

Cíl: Projít jeskyní, najít poklad.

Dračí doupě

Cíl: Projít jeskyní, najít poklad.
Překážky:

Dračí doupě

Cíl: Projít jeskyní, najít poklad.

Překážky:

- zapáchající příšera (**Wumpus**), která sežere každého, kdo jí přijde do cesty.

Dračí doupě

Cíl: Projít jeskyní, najít poklad.

Překážky:

- zapáchající příšera (**Wumpus**), která sežere každého, kdo jí přijde do cesty.
- hluboké jámy, do kterých lze spadnout a hrdina se nedostane ven.

Dračí doupě

Cíl: Projít jeskyní, najít poklad.

Překážky:

- zapáchající příšera (**Wumpus**), která sežere každého, kdo jí přijde do cesty.
- hluboké jámy, do kterých lze spadnout a hrdina se nedostane ven.

Pravidla:

- v sousedství jámy to fouká

Dračí doupě

Cíl: Projít jeskyní, najít poklad.

Překážky:

- zapáchající příšera (**Wumpus**), která sežere každého, kdo jí přijde do cesty.
- hluboké jámy, do kterých lze spadnout a hrdina se nedostane ven.

Pravidla:

- v sousedství jámy to fouká
- v sousedství příšery to zapáchá

Dračí doupě

Cíl: Projít jeskyní, najít poklad.

Překážky:

- zapáchající příšera (**Wumpus**), která sežere každého, kdo jí přijde do cesty.
- hluboké jámy, do kterých lze spadnout a hrdina se nedostane ven.

Pravidla:

- v sousedství jámy to fouká
- v sousedství příšery to zapáchá
- poklad se třpytí
- příšera, která umře, vydá skřek slyšitelný po celé jeskyni

Dračí doupě

Cíl: Projít jeskyní, najít poklad.

Překážky:

- zapáchající příšera (**Wumpus**), která sežere každého, kdo jí přijde do cesty.
- hluboké jámy, do kterých lze spadnout a hrdina se nedostane ven.

Pravidla:

- v sousedství jámy to fouká
- v sousedství příšery to zapáchá
- poklad se třpytí
- příšera, která umře, vydá skřek slyšitelný po celé jeskyni
- je možno jít dopředu nebo se otočit o 90 stupňů

Dračí doupě

Cíl: Projít jeskyní, najít poklad.

Překážky:

- zapáchající příšera (**Wumpus**), která sežere každého, kdo jí přijde do cesty.
- hluboké jámy, do kterých lze spadnout a hrdina se nedostane ven.

Pravidla:

- v sousedství jámy to fouká
- v sousedství příšery to zapáchá
- poklad se třpytí
- příšera, která umře, vydá skřek slyšitelný po celé jeskyni
- je možno jít dopředu nebo se otočit o 90 stupňů
- není možno chodit skrz stěny, náraz do zdi pořádně bolí

Dračí doupě

Cíl: Projít jeskyní, najít poklad.

Překážky:

- zapáchající příšera (**Wumpus**), která sežere každého, kdo jí přijde do cesty.
- hluboké jámy, do kterých lze spadnout a hrdina se nedostane ven.

Pravidla:

- v sousedství jámy to fouká
- v sousedství příšery to zapáchá
- poklad se třpytí
- příšera, která umře, vydá skřek slyšitelný po celé jeskyni
- je možno jít dopředu nebo se otočit o 90 stupňů
- není možno chodit skrz stěny, náraz do zdi pořádně bolí
- je možno sebrat poklad

Dračí doupě

Cíl: Projít jeskyní, najít poklad.

Překážky:

- zapáchající příšera (**Wumpus**), která sežere každého, kdo jí přijde do cesty.
- hluboké jámy, do kterých lze spadnout a hrdina se nedostane ven.

Pravidla:

- v sousedství jámy to fouká
- v sousedství příšery to zapáchá
- poklad se třpytí
- příšera, která umře, vydá skřek slyšitelný po celé jeskyni
- je možno jít dopředu nebo se otočit o 90 stupňů
- není možno chodit skrz stěny, náraz do zdi pořádně bolí
- je možno sebrat poklad
- je možno střelit jeden šíp, který letí daným směrem dokud nezabije příšeru nebo nenarazí na stěnu

Dračí doupě

Cíl: Projít jeskyní, najít poklad.

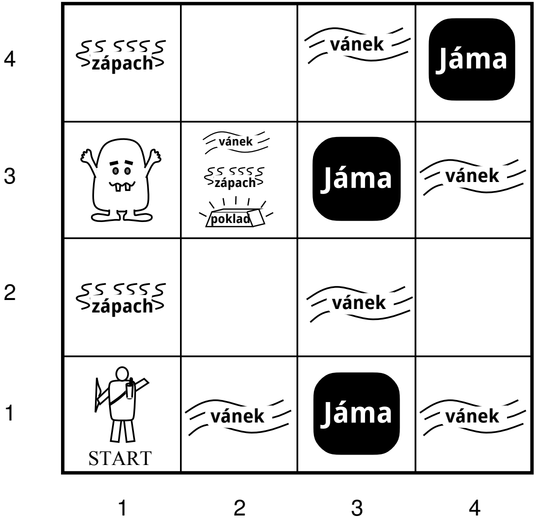
Překážky:

- zapáchající příšera (**Wumpus**), která sežere každého, kdo jí přijde do cesty.
- hluboké jámy, do kterých lze spadnout a hrdina se nedostane ven.

Pravidla:

- v sousedství jámy to fouká
- v sousedství příšery to zapáchá
- poklad se třpytí
- příšera, která umře, vydá skřek slyšitelný po celé jeskyni
- je možno jít dopředu nebo se otočit o 90 stupňů
- není možno chodit skrz stěny, náraz do zdi pořádně bolí
- je možno sebrat poklad
- je možno střelit jeden šíp, který letí daným směrem dokud nezabije příšeru nebo nenarazí na stěnu

Schéma jeskyně



Průchod jeskyní

(1,1) => (2,1) => (1,1) => (1,2) => (2,2) => ... => (2,3) => (2,2) => (1,2) => (1,1)

Znalostní báze — výroková logika

Naše znalostní báze (náš model světa) si bude pro každé políčko uchovávat informaci o tom, zda

- to tam fouká ($F_{i,j}$)

Znalostní báze — výroková logika

Naše znalostní báze (náš model světa) si bude pro každé políčko uchovávat informaci o tom, zda

- to tam fouká ($F_{i,j}$)
- to tam zapáchá ($Z_{i,j}$)

Znalostní báze — výroková logika

Naše znalostní báze (náš model světa) si bude pro každé políčko uchovávat informaci o tom, zda

- to tam fouká ($F_{i,j}$)
- to tam zapáchá ($Z_{i,j}$)
- je tam příšera ($P_{i,j}$)

Znalostní báze — výroková logika

Naše znalostní báze (náš model světa) si bude pro každé políčko uchovávat informaci o tom, zda

- to tam fouká ($F_{i,j}$)
- to tam zapáchá ($Z_{i,j}$)
- je tam příšera ($P_{i,j}$)
- je tam jáma ($J_{i,j}$)

Znalostní báze — výroková logika

Naše znalostní báze (náš model světa) si bude pro každé políčko uchovávat informaci o tom, zda

- to tam fouká ($F_{i,j}$)
- to tam zapáchá ($Z_{i,j}$)
- je tam příšera ($P_{i,j}$)
- je tam jáma ($J_{i,j}$)
- je tam poklad ($Pk_{i,j}$)

Znalostní báze — výroková logika

Naše znalostní báze (náš model světa) si bude pro každé políčko uchovávat informaci o tom, zda

- to tam fouká ($F_{i,j}$)
- to tam zapáchá ($Z_{i,j}$)
- je tam příšera ($P_{i,j}$)
- je tam jáma ($J_{i,j}$)
- je tam poklad ($Pk_{i,j}$)
- tím políček letěl šíp ($S_{i,j}$)

Znalostní báze — výroková logika

Naše znalostní báze (náš model světa) si bude pro každé políčko uchovávat informaci o tom, zda

- to tam fouká ($F_{i,j}$)
- to tam zapáchá ($Z_{i,j}$)
- je tam příšera ($P_{i,j}$)
- je tam jáma ($J_{i,j}$)
- je tam poklad ($Pk_{i,j}$)
- tím políček letěl šíp ($S_{i,j}$)
- zda jsem políčko navštívil ($N_{i,j}$)

Znalostní báze — výroková logika

Naše znalostní báze (náš model světa) si bude pro každé políčko uchovávat informaci o tom, zda

- to tam fouká ($F_{i,j}$)
- to tam zapáchá ($Z_{i,j}$)
- je tam příšera ($P_{i,j}$)
- je tam jáma ($J_{i,j}$)
- je tam poklad ($Pk_{i,j}$)
- tím políček letěl šíp ($S_{i,j}$)
- zda jsem políčko navštívil ($N_{i,j}$)

Dále si bude pamatovat

Znalostní báze — výroková logika

Naše znalostní báze (náš model světa) si bude pro každé políčko uchovávat informaci o tom, zda

- to tam fouká ($F_{i,j}$)
- to tam zapáchá ($Z_{i,j}$)
- je tam příšera ($P_{i,j}$)
- je tam jáma ($J_{i,j}$)
- je tam poklad ($Pk_{i,j}$)
- tím políček letěl šíp ($S_{i,j}$)
- zda jsem políčko navštívil ($N_{i,j}$)

Dále si bude pamatovat, zda je příšera mrtvá (Sk)

Znalostní báze — výroková logika

Naše znalostní báze (náš model světa) si bude pro každé políčko uchovávat informaci o tom, zda

- to tam fouká ($F_{i,j}$)
- to tam zapáchá ($Z_{i,j}$)
- je tam příšera ($P_{i,j}$)
- je tam jáma ($J_{i,j}$)
- je tam poklad ($Pk_{i,j}$)
- tím políček letěl šíp ($S_{i,j}$)
- zda jsem políčko navštívil ($N_{i,j}$)

Dále si bude pamatovat, zda je příšera mrtvá (Sk), zda jsem vystrelil sip (S)

Znalostní báze — výroková logika

Naše znalostní báze (náš model světa) si bude pro každé políčko uchovávat informaci o tom, zda

- to tam fouká ($F_{i,j}$)
- to tam zapáchá ($Z_{i,j}$)
- je tam příšera ($P_{i,j}$)
- je tam jáma ($J_{i,j}$)
- je tam poklad ($Pk_{i,j}$)
- tím políček letěl šíp ($S_{i,j}$)
- zda jsem políčko navštívil ($N_{i,j}$)

Dále si bude pamatovat, zda je příšera mrtvá (Sk), zda jsem vystrelil sip (S) a zda jsem sebral poklad ($Hura$)

Znalostní báze — výroková logika

Naše znalostní báze (náš model světa) si bude pro každé políčko uchovávat informaci o tom, zda

- to tam fouká ($F_{i,j}$)
- to tam zapáchá ($Z_{i,j}$)
- je tam příšera ($P_{i,j}$)
- je tam jáma ($J_{i,j}$)
- je tam poklad ($Pk_{i,j}$)
- tím políček letěl šíp ($S_{i,j}$)
- zda jsem políčko navštívil ($N_{i,j}$)

Dále si bude pamatovat, zda je příšera mrtvá (Sk), zda jsem vystrelil sip (S) a zda jsem sebral poklad ($Hura$). Naše znalosti o světě pak můžeme zapsat pomocí axiomů

Znalostní báze — výroková logika

Naše znalostní báze (náš model světa) si bude pro každé políčko uchovávat informaci o tom, zda

- to tam fouká ($F_{i,j}$)
- to tam zapáchá ($Z_{i,j}$)
- je tam příšera ($P_{i,j}$)
- je tam jáma ($J_{i,j}$)
- je tam poklad ($Pk_{i,j}$)
- tím políček letěl šíp ($S_{i,j}$)
- zda jsem políčko navštívil ($N_{i,j}$)

Dále si bude pamatovat, zda je příšera mrtvá (Sk), zda jsem vystrelil sip (S) a zda jsem sebral poklad ($Hura$). Naše znalosti o světě pak můžeme zapsat pomocí axiomů (formulí výrokové logiky nad proměnnými F, Z, P, J, Pk):

Znalostní báze — výroková logika

Naše znalostní báze (náš model světa) si bude pro každé políčko uchovávat informaci o tom, zda

- to tam fouká ($F_{i,j}$)
- to tam zapáchá ($Z_{i,j}$)
- je tam příšera ($P_{i,j}$)
- je tam jáma ($J_{i,j}$)
- je tam poklad ($Pk_{i,j}$)
- tím políček letěl šíp ($S_{i,j}$)
- zda jsem políčko navštívil ($N_{i,j}$)

Dále si bude pamatovat, zda je příšera mrtvá (Sk), zda jsem vystrelil sip (S) a zda jsem sebral poklad ($Hura$). Naše znalosti o světě pak můžeme zapsat pomocí axiomů (formulí výrokové logiky nad proměnnými F, Z, P, J, Pk):

- $P_{i,j} \rightarrow (Z_{i-1,j} \wedge Z_{i+1,j} \wedge Z_{i,j-1} \wedge Z_{i,j+1})$

Znalostní báze — výroková logika

Naše znalostní báze (náš model světa) si bude pro každé políčko uchovávat informaci o tom, zda

- to tam fouká ($F_{i,j}$)
- to tam zapáchá ($Z_{i,j}$)
- je tam příšera ($P_{i,j}$)
- je tam jáma ($J_{i,j}$)
- je tam poklad ($Pk_{i,j}$)
- tím políček letěl šíp ($S_{i,j}$)
- zda jsem políčko navštívil ($N_{i,j}$)

Dále si bude pamatovat, zda je příšera mrtvá (Sk), zda jsem vystrelil sip (S) a zda jsem sebral poklad ($Hura$). Naše znalosti o světě pak můžeme zapsat pomocí axiomů (formulí výrokové logiky nad proměnnými F, Z, P, J, Pk):

- $P_{i,j} \rightarrow (Z_{i-1,j} \wedge Z_{i+1,j} \wedge Z_{i,j-1} \wedge Z_{i,j+1})$
- $J_{i,j} \rightarrow (F_{i-1,j} \wedge F_{i+1,j} \wedge F_{i,j-1} \wedge F_{i,j+1})$

Znalostní báze — výroková logika

Naše znalostní báze (náš model světa) si bude pro každé políčko uchovávat informaci o tom, zda

- to tam fouká ($F_{i,j}$)
- to tam zapáchá ($Z_{i,j}$)
- je tam příšera ($P_{i,j}$)
- je tam jáma ($J_{i,j}$)
- je tam poklad ($Pk_{i,j}$)
- tím políček letěl šíp ($S_{i,j}$)
- zda jsem políčko navštívil ($N_{i,j}$)

Dále si bude pamatovat, zda je příšera mrtvá (Sk), zda jsem vystrelil sip (S) a zda jsem sebral poklad ($Hura$). Naše znalosti o světě pak můžeme zapsat pomocí axiomů (formulí výrokové logiky nad proměnnými F, Z, P, J, Pk):

- $P_{i,j} \rightarrow (Z_{i-1,j} \wedge Z_{i+1,j} \wedge Z_{i,j-1} \wedge Z_{i,j+1})$
- $J_{i,j} \rightarrow (F_{i-1,j} \wedge F_{i+1,j} \wedge F_{i,j-1} \wedge F_{i,j+1})$
- $P_{i,j} \wedge S_{i,j} \rightarrow Sk$

Agent Hrdina

```python
def agentHrdina(akce, vjem):
  if vjem == 'zapach':
    self.baze.update('Z'+self.pos, True)
  ...
  if len(self.plan) > 0:
    return self.plan.pop()

  # Pokud jsem nasel poklad tak ho seberu
  # a vydam se zpet
  if vjem == 'poklad':
    plan = self.route((1,1))
    return 'seber_poklad'
```

Agent Hrdina, pokračování . . .

```
# Zkusim najit prokazatelne bezpecne
# nenavstivene policko a vydat se tam
for pos in jeskyne:
  if self.baze.ask('not J'+pos+' and not P'+
     pos):
    if self.baze.ask('not N'+pos):
      self.plan = self.route(pos)
      if self.plan:
        return self.plan.pop()

# Pokud jsem jeste nestrilel, tak to zkusim
if self.baze.ask('not S'):
  return 'vystrel'
```

Agent Hrdina, pokračování ...

```python
# Zkusim najit alespon policka, ktera nejsou
# prokazatelne nebezpecna a vydat se tam
for pos in jeskyne:
  if not self.baze.ask(['J'+pos+' or P'+pos]):
    if self.baze.ask('not N'+pos):
      self.plan = self.route(pos)
      if self.plan:
        return self.plan.pop()

# Vzdej to a vylez z jeskyne
plan = self.route((1,1))
```

`baze.ask`: Sémantické vs Syntaktické odvozování

Jsou v zásadě dvě možnosti, jak zjišťovat platnost formulí:

`baze.ask`: Sémantické vs Syntaktické odvozování

Jsou v zásadě dvě možnosti, jak zjišťovat platnost formulí:

Ověřování modelů (model checking)

`baze.ask`: Sémantické vs Syntaktické odvozování

Jsou v zásadě dvě možnosti, jak zjišťovat platnost formulí:

Ověřování modelů (model checking)

- procházení všech modelů (enumerace pravdivostní tabulky)

`baze.ask`: Sémantické vs Syntaktické odvozování

Jsou v zásadě dvě možnosti, jak zjišťovat platnost formulí:

Ověřování modelů (model checking)

- procházení všech modelů (enumerace pravdivostní tabulky)
- Davis-Putnam-Logemann-Loveland algoritmus (DPLL)

`baze.ask`: Sémantické vs Syntaktické odvozování

Jsou v zásadě dvě možnosti, jak zjišťovat platnost formulí:

Ověřování modelů (model checking)

- procházení všech modelů (enumerace pravdivostní tabulky)
- Davis-Putnam-Logemann-Loveland algoritmus (DPLL)
- lokální prohledávání (minimalizace konfliktů), WalkSAT

`baze.ask`: Sémantické vs Syntaktické odvozování

Jsou v zásadě dvě možnosti, jak zjišťovat platnost formulí:

Ověřování modelů (model checking)

- procházení všech modelů (enumerace pravdivostní tabulky)
- Davis-Putnam-Logemann-Loveland algoritmus (DPLL)
- lokální prohledávání (minimalizace konfliktů), WalkSAT

Dokazování

`baze.ask`: Sémantické vs Syntaktické odvozování

Jsou v zásadě dvě možnosti, jak zjišťovat platnost formulí:

Ověřování modelů (model checking)

- procházení všech modelů (enumerace pravdivostní tabulky)
- Davis-Putnam-Logemann-Loveland algoritmus (DPLL)
- lokální prohledávání (minimalizace konfliktů), WalkSAT

Dokazování

- hledání důkazů aplikací odvozovacích pravidel

`baze.ask`: Sémantické vs Syntaktické odvozování

Jsou v zásadě dvě možnosti, jak zjišťovat platnost formulí:

Ověřování modelů (model checking)

- procházení všech modelů (enumerace pravdivostní tabulky)
- Davis-Putnam-Logemann-Loveland algoritmus (DPLL)
- lokální prohledávání (minimalizace konfliktů), WalkSAT

Dokazování

- hledání důkazů aplikací odvozovacích pravidel
- rezoluce

Sémantické odvozování — enumerace

```
def enumSAT( clauses, unassigned_vars,
    partial_model):
    if len(unassigned_vars) == 0:
        return sat(clauses, partial_model) == '
            yes'
    else:
        var = unassigned_vars.pop()
        partial_model[var] = True
        if enumSAT( clauses, unassigned_vars,
            partial_model): return partial_model
        partial_model[var] = False
        if enumSAT( clauses, unassigned_vars,
            partial_model): return partial_model
        unassigned_vars.append(var)
        del partial_model[var]
```

Sémantické odvozování — DPLL

```
def dpll(clauses,unassigned_vars,partial_model):
  status = SAT(clauses, partial_model)

  # Pokud castecny model splnuje vsechny formule
  # mame vyhrano
  if status == 'all_sat'
    return partial_model

  # Pokud castecny model nejakou formuli
  # nesplnuje, muzeme ho rovnou zahodit
  elif status == 'some_fail':
    return None
```

DPLL, pokračování . . .

```
# Pokud je nejaka promenna ryzi, vim,
# jakou musi mit hodnotu
var,val = extract_pure( clauses,
    unassigned_vars )
if var:
     partial_model[var] = val
     if dpll( clauses, unassigned_vars,
         partial_model ): return partial_model
     else:
          del partial_model[var]
          unassigned_vars.append(var)
          return None
```

DPLL, pokračování . . .

```
# Pokud je v nejake klauzuli pouze jedna
# promenna, vim, jakou musi mit hodnotu
var, val = extract_unit_var(clauses,
    unassigned_vars)
if var:
     partial_model[var] = val
     if dpll( clauses, unassigned_vars,
         partial_model): return partial_model
     else:
          del partial_model[var]
          unassigned_vars.append(var)
          return None
```

DPLL, pokračování . . .

```
# Zvolme jednu neohodnocenou promennou,
    ohodnotme ji
# a zkusme rekurzivne postavit model
var = unassigned_vars.pop()
partial_model[var] = True
if dpll( clauses, unassigned_vars,
    partial_model): return partial_model
partial_model[var] = False
if dpll( clauses, unassigned_vars,
    partial_model): return partial_model
unassigned_vars.append(var)
del partial_model[var]
return None
```

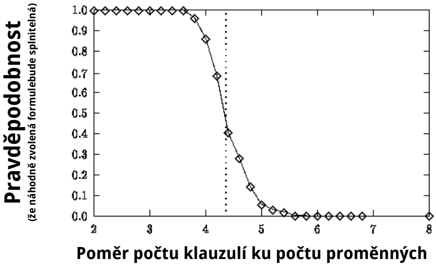
Sémantické odvozování — WalkSAT

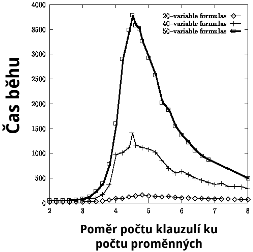
```
def walkSAT(clauses, p, maxflips):
  vars  = extract_vars(klauzule)
  model = random_assignment(vars,[True,False])

  # Zkousej prehazovat promenne
  while maxflips > 0:
    maxflip -= 1

    # Pokud model splnuje klauzule, vyhrali jsme
    if sat(model, clauses): return model

    # Zvol si nahodnou klauzuli
    clause = random.choose(clauses)
    cvars = extract_vars(clause)
```

WalkSAT, pokračování ...

```
    if rand.random() < p:
      # S pravdepodobnosti p prehod nahodne
      # zvolenou promennou
      var = random.choose(cvars)
      model[var] = not model[var]
    else:
      # Prehod promennou, ktera maximalizuje
      # pocet splnenych formuli
      max_sat = 0
      vmax = cvars[0]
      for var in cvars:
        model[var] = not model[var]
        if countSAT(model, clauses) > max_sat:
          vmax = var
          max_sat = countSAT(model,clauses)
        model[var] = not model[var]
      model[vmax] = not model[vmax]
  return None
```

SAT, Fázový přechod





(Zdroj: `http://www.compphys.uni-oldenburg.de/en/download/talks/pekka_orponen.pdf`)

Dokazování — rezoluce

Základní krok

Dokazování — rezoluce

Základní krok

$$\frac{a \vee b, -b}{a}$$

Základní krok

$$\frac{a \vee b, -b}{a}$$

Obecně

Dokazování — rezoluce

Základní krok

$$\frac{a \vee b, -b}{a}$$

Obecně

$$\frac{x_0 \vee \cdots x_k \vee \cdots x_n, y_0 \vee \cdots \vee y_l \vee \cdots y_m}{x_0 \vee \cdots \vee x_{k-1} \vee x_{k+1} \vee \cdots \vee x_n \vee y_0 \vee \cdots y_{l-1} \vee y_{l+1} \vee \cdots \vee y_m},$$

kde

$$x_k = \; - y_l.$$

Dokazování — rezoluce

Základní krok

$$\frac{a \vee b, -b}{a}$$

Obecně

$$\frac{x_0 \vee \cdots x_k \vee \cdots x_n, y_0 \vee \cdots \vee y_l \vee \cdots y_m}{x_0 \vee \cdots \vee x_{k-1} \vee x_{k+1} \vee \cdots \vee x_n \vee y_0 \vee \cdots y_{l-1} \vee y_{l+1} \vee \cdots \vee y_m},$$

kde

$$x_k = -y_l.$$

Dokazování — rezoluce

Základní krok

$$\frac{a \vee b, -b}{a}$$

Obecně

$$\frac{x_0 \vee \cdots x_k \vee \cdots x_n, y_0 \vee \cdots \vee y_l \vee \cdots y_m}{x_0 \vee \cdots \vee x_{k-1} \vee x_{k+1} \vee \cdots \vee x_n \vee y_0 \vee \cdots y_{l-1} \vee y_{l+1} \vee \cdots \vee y_m},$$

kde

$$x_k = -y_l.$$

Rezoluce — algoritmus

Algoritmus

Rezoluce — algoritmus

Algoritmus

Chceme dokázat φ z axiomu ψ.

Rezoluce — algoritmus

Algoritmus

Chceme dokázat φ z axiomu ψ.

1. Přepiš formuli $\psi \wedge \neg\varphi$ do CNF, seznam klauzulí budiž $K_0 = K$.

Rezoluce — algoritmus

Algoritmus

Chceme dokázat φ z axiomu ψ.

1. Přepiš formuli $\psi \wedge \neg\varphi$ do CNF, seznam klauzulí budiž $K_0 = K$.
2. Pokud žádné dvě klauzule v K neobsahují komplementární literál, vrať `False`

Rezoluce — algoritmus

Algoritmus

Chceme dokázat φ z axiomu ψ.

1. Přepiš formuli $\psi \wedge \neg\varphi$ do CNF, seznam klauzulí budiž $K_0 = K$.
2. Pokud žádné dvě klauzule v K neobsahují komplementární literál, vrať `False`
3. Odeber z K dvě klauzule s komplementárním literálem, aplikuj na ně základní krok rezoluce

Rezoluce — algoritmus

Algoritmus

Chceme dokázat φ z axiomu ψ.

1. Přepiš formuli $\psi \wedge \neg\varphi$ do CNF, seznam klauzulí budiž $K_0 = K$.
2. Pokud žádné dvě klauzule v K neobsahují komplementární literál, vrať `False`
3. Odeber z K dvě klauzule s komplementárním literálem, aplikuj na ně základní krok rezoluce
4. Pokud je výsledkem prázdná klauzule, vrať `True`

Algoritmus

Chceme dokázat φ z axiomu ψ.

1. Přepiš formuli $\psi \wedge \neg\varphi$ do CNF, seznam klauzulí budiž $K_0 = K$.
2. Pokud žádné dvě klauzule v K neobsahují komplementární literál, vrať `False`
3. Odeber z K dvě klauzule s komplementárním literálem, aplikuj na ně základní krok rezoluce
4. Pokud je výsledkem prázdná klauzule, vrať `True`
5. Jinak vlož výsledek zpět do K a pokračuj krokem 2

Dokazování — Hornovské klauzule

Hornovská klauzule je klauzule tvaru

Dokazování — Hornovské klauzule

Hornovská klauzule je klauzule tvaru

$$x_0 \wedge \cdots \wedge x_n \to x_{n+1}$$

Dokazování — Hornovské klauzule

Hornovská klauzule je klauzule tvaru

$$x_0 \wedge \cdots \wedge x_n \to x_{n+1},$$

t.j. klauzule, která v obsahuje pouze jeden pozitivní literál (x_{n+1}).

Dokazování — Hornovské klauzule

Hornovská klauzule je klauzule tvaru

$$x_0 \wedge \cdots \wedge x_n \rightarrow x_{n+1},$$

t.j. klauzule, která v obsahuje pouze jeden pozitivní literál (x_{n+1}).

Hornovská formule je formule, jejíž CNF je konjunkce hornovských klauzulí.

Hornovské klauzule — dopředné řetězení

Chceme zjistit, zda lze ze znalostní báze sestávající z hornovských klauzulí a faktů

Hornovské klauzule — dopředné řetězení

Chceme zjistit, zda lze ze znalostní báze sestávající z hornovských klauzulí a faktů (pozitivních literálů) odvodit fakt x.

Hornovské klauzule — dopředné řetězení

Chceme zjistit, zda lze ze znalostní báze sestávající z hornovských klauzulí a faktů (pozitivních literálů) odvodit fakt x.

Algoritmus začne z faktů ze znalostní bázi

Hornovské klauzule — dopředné řetězení

Chceme zjistit, zda lze ze znalostní báze sestávající z hornovských klauzulí a faktů (pozitivních literálů) odvodit fakt x.

Algoritmus začne z faktů ze znalostní bázi a v každém kroku k nim přidá závěr každé hornovské formule, jejíž premisy už jsou známy.

Hornovské klauzule — dopředné řetězení

Chceme zjistit, zda lze ze znalostní báze sestávající z hornovských klauzulí a faktů (pozitivních literálů) odvodit fakt x.

Algoritmus začne z faktů ze znalostní bázi a v každém kroku k nim přidá závěr každé hornovské formule, jejíž premisy už jsou známy. Pokud tímto postupem dostane fakt x vrátí **True**

Hornovské klauzule — dopředné řetězení

Chceme zjistit, zda lze ze znalostní báze sestávající z hornovských klauzulí a faktů (pozitivních literálů) odvodit fakt x.

Algoritmus začne z faktů ze znalostní bázi a v každém kroku k nim přidá závěr každé hornovské formule, jejíž premisy už jsou známy. Pokud tímto postupem dostane fakt x vrátí **True** jinak vrátí **False**.

Hornovské klauzule — dopředné řetězení

```
def forwardChaining( baze, x ):
  # Spocti kolik ma kazda klauzule v bazi premis
  count = count_premises(baze.clauses)
  odvozeno = []
  work = baze.facts
  while len(work) > 0:
    fact = work.pop()
    if fact == x: return True
    if fact not in odvozeno:
      odvozeno.append(fact)
      for c in baze.clauses:
        if fact in c.premises:
          count[c] -= 1
          if count[c] == 0:
            work.append(c.conclusion)
  return False
```

Hornovské kaluzule — zpětné řetězení

- Dopředné řetězení vychází z faktů a odvozuje nové, dokud nenarazí na x.

Hornovské kaluzule — zpětné řetězení

- Dopředné řetězení vychází z faktů a odvozuje nové, dokud nenarazí na x.
- Odvodí spoustu zbytečností.

Hornovské kaluzule — zpětné řetězení

- Dopředné řetězení vychází z faktů a odvozuje nové, dokud nenarazí na x.
- Odvodí spoustu zbytečností.
- Zpětné odvození odvozuje pouze fakta, která jsou perspektivní, t.j. vedou k x.

Hornovské kaluzule — zpětné řetězení

```
def backwardChaining( baze, x ):
  if x in baze.facts:
    return True
  for c in baze.clauses:
    if x == c.conclusion:
      can_sat_c = True
      for y in c.premises:
        if not backwardChaining( baze, y ):
          can_sat_c = False
          break
      if can_sat_c:
        baze.facts.append(x)
        return True
  return False
```