IKI 30320: Sistem Cerdas
Kuliah 11: Logical Inference & Wumpus Agent

Ruli Manurung

Fakultas Ilmu Komputer
Universitas Indonesia

24 Oktober 2007

Outline

Jenis-jenis metode pembuktian

Secara umum, ada 2 jenis:

- Pengaplikasian *inference rule*
  - Hasilkan kalimat baru yang sah (*sound*) dari yang lama
  - Bukti (proof): serangkaian pengaplikasian *inference rule* Inference rule sebagai operator $\rightarrow$ algoritma search.
  - Biasanya, kalimat harus diterjemahkan ke dalam sebuah normal form
- Model checking
  - Penjabaran *truth table* (eksponensial dalam n)
  - Backtracking lebih efisien, mis: algoritma DPLL
  - Heuristic search dalam *model space* (*sound* tapi *incomplete*), mis: min-conflicts hill-climbing

Outline

Satisfiability sebagai CSP

- Model checking: mencari model untuk KB = CSP
  - Variable: propositional symbols
  - Domain: {true,false}
  - Constraints: Semua kalimat dalam $KB + \alpha$ harus bernilai *true*
- Algoritma backtracking untuk CSP bisa dipakai.
- Secara teoritis: depth-first search semua kemungkinan model = truth table!
- Ada heuristic yang bisa digunakan untuk pruning

Algoritma Davis-Putnam

- Algoritma DPLL adalah backtracking + heuristic a la CSP
- Sentence harus dalam bentuk CNF (conjunctive normal form) $\rightarrow$ *conjunction of disjunction of literals*.
  Mis: $(A \vee \neg B) \wedge (B \vee \neg C \vee \neg D)$
- Memanfaatkan tiga heuristic
  - Early termination
  - Pure symbol
  - Unit clause

Heuristic: early termination

- DPLL mendeteksi sentence true atau false sebelum model lengkap (semua symbol telah di-assign).
- Sebuah clause bernilai *true* jika *ada* literal yang *true*. Mis.: Jika $A = true$, $(A \vee B) \wedge (A \vee C)$ pasti *true* juga
- Sebuah sentence bernilai *false* jika *ada* clause yang *false*.

Heuristic: pure symbol

- Pure symbol adalah symbol yang muncul dengan "tanda" yang sama dalam semua clause.
- Mis.: $(A \vee \neg B) \wedge (\neg B \vee \neg C) \wedge (C \vee A)$
  A adalah pure symbol karena selalu muncul "positif"
  B adalah pure symbol karena selalu muncul "negatif"
  C bukan pure symbol
- Jika *pure symbol* diberi nilai shg. *literal*-nya *true*, *clause* di mana ia muncul pasti *true*.
- Ketika menentukan apakah symbol itu *pure* atau tidak, abaikan *clause* yang sudah diketahui *true*.
  Mis.: jika $B = false$, maka pada *sentence* di atas $(\neg B \vee \neg C)$ pasti *true*, sehingga C menjadi *pure*.

Heuristic: unit clause

- Unit clause adalah *clause* dengan hanya 1 *literal*, atau *clause* di mana semua *literal* kecuali 1 diberi nilai *false*.
- Mis.: Jika $B = false$, $(B \vee \neg C)$ adalah *unit clause*. Agar true, C harus di-*assign* nilai *false*.
- Heuristic ini memilih meng-*assign* nilai kepada symbol demikian terlebih dahulu.
- Meng-*assign* nilai symbol di dalam *unit clause* dapat berakibat timbul *unit clause* lain $\rightarrow$ unit propagation.
- Mirip forward chaining pada Horn clause.

Backtracking untuk satisfiability

Algoritma DPLL

function DPLL-SATISFIABLE?(*s*) **returns** *true* or *false*
  inputs: *s*, a sentence in propositional logic

  clauses ← the set of clauses in the CNF representation of *s*
  symbols ← a list of the proposition symbols in *s*
  return DPLL(*clauses*, *symbols*, [ ])

---

function DPLL(*clauses*, *symbols*, *model*) **returns** *true* or *false*

  if every clause in *clauses* is true in *model* **then return** *true*
  if some clause in *clauses* is false in *model* **then return** *true*
  P, *value* ← FIND-PURE-SYMBOL(*symbols*, *clauses*, *model*)
  if *P* is non-null **then return** DPLL(*clauses*, *symbols*–*P*, [*P* = *value*|*model*])
  P, *value* ← FIND-UNIT-CLAUSE(*clauses*, *model*)
  if *P* is non-null **then return** DPLL(*clauses*, *symbols*–*P*, [*P* = *value*|*model*])
  P ← FIRST(*symbols*); *rest* ← REST(*symbols*)
  return DPLL(*clauses*, *rest*, [*P* = *true*|*model*]) **or** DPLL(*clauses*, *rest*, [*P* = *false*|*model*])

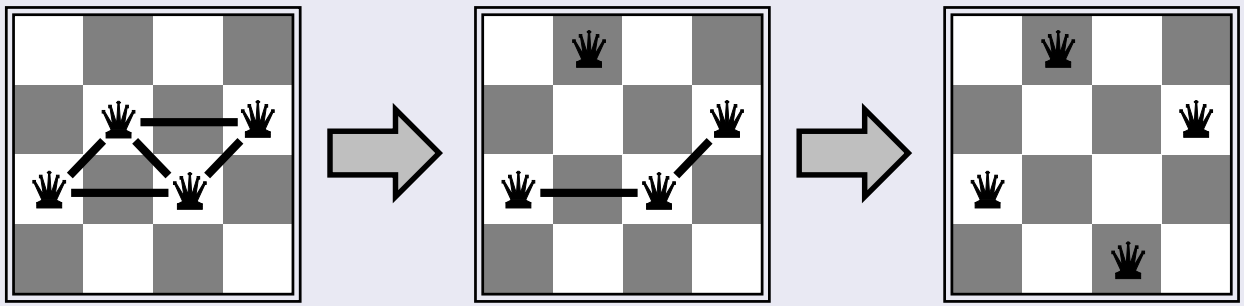
CHAFF, sebuah implementasi DPLL, bisa menyelesaikan *satisfiability* pada masalah verifikasi hardware dengan jutaan variable!

Outline

Pendekatan local search

- Goal dari *satisfiability*: mencari *complete assignment* sehingga setiap *clause* bernilai *true*.
- Ide baru: berikan *assignment* secara acak, lalu coba ganti nilai yang melanggar *constraint* (membuat *clause* bernilai *false*).
- Supaya cepat, gunakanlah evaluation function MIN-CONFLICTS: hitung jumlah *clause* yang *false*.

Ingat local search dengan MIN-CONFLICTS pada *n*-queens?



Algoritma local search

Algoritma WALKSAT

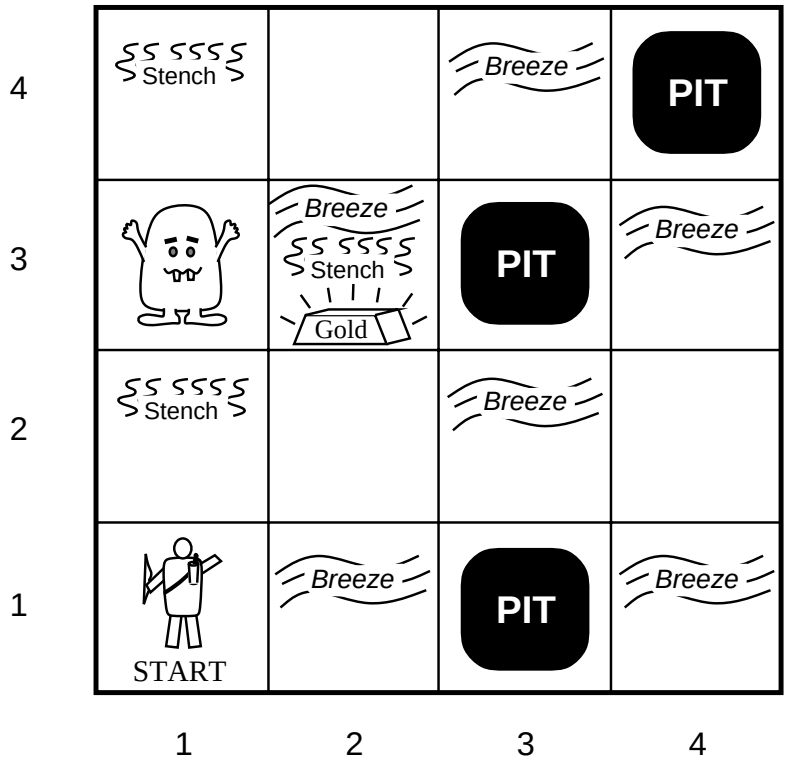
function WALKSAT(*clauses*, *p*, *max-flips*) **returns** a satisfying model or *failure*
inputs: *clauses*, a set of clauses in propositional logic
p, the probability of choosing to do a "random walk" move, typically around 0.5
max-flips, number of flips allowed before giving up

model ← a random assignment of *true*/*false* to the symbols in *clauses*
for $i = 1$ **to** *max-flips* **do**
if *model* satisfies *clauses* **then return** *model*
clause ← a randomly selected clause from *clauses* that is false in *model*
with probability *p* flip the value in *model* of a randomly selected symbol from *clause*
else flip whichever symbol in *clause* maximizes the number of satisfied clauses
return *failure*

- Dalam praktek, WALKSAT lebih cepat dari DPLL!
- Namun, WALKSAT tidak cocok untuk menjamin unsatisfiability.

Outline

LogicalAnto^{TM}



Bagaimana kita merancang agent yang menggunakan logic untuk menentukan langkah berikutnya?

TELL *LogicalAnto*^{TM} about WumpusWorld

Di kamar [1, 1] tidak ada *pit* atau *Wumpus*

$\neg P_{1,1}$ dan $\neg W_{1,1}$

Untuk setiap kamar $[x, y]$, TELL Anto hubungan *breeze* dan *pit*

$B_{x,y} \Leftrightarrow (P_{x,y+1} \vee P_{x,y-1} \vee P_{x+1,y} \vee P_{x-1,y})$

Untuk setiap kamar $[x, y]$, TELL Anto hubungan *stench* dan *Wumpus*

$S_{x,y} \Leftrightarrow (W_{x,y+1} \vee W_{x,y-1} \vee W_{x+1,y} \vee W_{x-1,y})$

Hanya ada satu Wumpus

- Harus ada sekurangnya satu Wumpus:
  $W_{1,1} \vee W_{1,2} \vee \ldots \vee W_{4,3} \vee W_{4,4} \vee$
- Tidak ada lebih dari satu Wumpus: $\neg W_{i,j} \vee \neg W_{k,l}$

LogicalAnto^{TM} mulai dengan 155 sentence, 64 propositional symbol.

Cara "kerja" *LogicalAnto*^{TM}

- Setiap kali memasuki ruangan baru $[x, y]$, update KB:
  - TELL(KB,$S_{x,y}$)
  - TELL(KB,$\neg S_{x,y}$)
  - TELL(KB,$B_{x,y}$)
  - TELL(KB,$\neg B_{x,y}$)
- Untuk setiap kamar $[i, j]$ yang bisa dimasuki, buktikan apakah ia aman: $KB \models (\neg P_{i,j} \wedge \neg W_{i,j})$, dkl.: ASK$(\neg P_{i,j} \wedge \neg W_{i,j})$
- Kalau tidak ada yang pasti, cari yang mungkin aman: $KB \not\models (P_{i,j} \vee W_{i,j})$, dkl.: ASK$(\neg(P_{i,j} \vee W_{i,j}))$
- ASK bisa dengan penjabaran truth-table (2^{64} kemungkinan!), DPLL, WALKSAT.

Implementasi *LogicalAnto*^{TM}

Algoritma Agent Wumpus World

function PL-WUMPUS-AGENT(*percept*) **returns** an *action*
  inputs: *percept*, a list, [*stench*,*breeze*,*glitter*]
  static: *KB*, a knowledge base, initially containing the "physics" of the wumpus world
    x, y, orientation, the agent's position (initially [1,1]) and orientation (initially *right*)
    visited, an array indicating which squares have been visited, initially *false*
    action, the agent's most recent action, initially null
    plan, an action sequence, initially empty

  update *x*,*y*,*orientation*, *visited* based on *action*
  if *stench* **then** TELL($KB, S_{x,y}$) **else** TELL($KB, \neg S_{x,y}$)
  if *breeze* **then** TELL($KB, B_{x,y}$) **else** TELL($KB, \neg B_{x,y}$)
  if *glitter* **then** *action* $\leftarrow$ *grab*
  else if *plan* is nonempty then *action* $\leftarrow$ POP(*plan*)
  else if for some fringe square [*i*,*j*], ASK($KB, (\neg P_{i,j} \wedge \neg W_{i,j})$) is *true* **or**
      for some fringe square [*i*,*j*], ASK($KB, (P_{i,j} \vee W_{i,j})$) is *false* **then do**
    plan $\leftarrow$ A*-GRAPH-SEARCH(ROUTE-PROBLEM([*x*,*y*], *orientation*, [*i*,*j*], *visited*))
    action $\leftarrow$ POP(*plan*)
  else *action* $\leftarrow$ a randomly chosen move
  return *action*

Ringkasan

- Satisfiability bisa dimodelkan sebagai CSP, diselesaikan dengan backtracking efisien: DPLL
- Bisa juga dengan local search menggunakan heuristic MIN-CONFLICTS: WALKSAT
- Propositional logic kurang ekspresif, mis.: kalimat *stench* dan *breeze* untuk setiap kamar, repotnya mendefinisikan satu Wumpus, dst.
- Baca bab 7 buku Russell & Norvig