Lezione 22

Ragionamento con NAFF e
Programmi normali.

Dalla lezione precedente

- comp(P) = completamento di Clark di P

1. Assunzione (CK) con Clark

- (CK) con Clark si formula come segue:
- ***Semantica***
  - conseguenze F considerate: F è $\exists x\ A$ oppure $\neg\exists x\ A$ (A atomica)
  - KB $\models_{CK}$ F sse comp(KB) $\models$ F
- ***Ragionamento:***
  - la procedura top down + NAFF forniscono una proof theory ***valida e completa :***

    KB $\models_{CK} \exists x\ A$ sse il goal A ha una risposta;

    KB $\models_{CK} \neg\exists x\ A$ sse si ha fallimento finito
  - ***Il ragionamento è non monotono***

Esempio di non monotonicità

- KB1 = {studente(luigi).}
- comp(KB1) = {studente(X) $\leftrightarrow$ X=luigi}
  - KB1 $\models_{CK}$ $\neg$studente(mario)
- KB2 = {studente(luigi). studente(mario).}
- comp(KB2) = {studente(X) $\leftrightarrow$ X=luigi $\vee$ X=mario}
  - KB2 $\models_{CK}$ studente(mario)
  - anche se KB2 contiene KB1, non ne conserva le conseguenze negative, anzi, ne contraddice alcune: nuova conoscenza elimina negazioni

(CK) con Clark rispetto a (CK) con CWA

- (CK- CWA): la proof-theory usa NAF
  - NAF non è effettiva, richiede un oracolo, essendo la dimostrabilità in DCL non decidibile in generale
- (CK- Clark): la proof theory usa NAFF
  - NAFF è effettiva, corrisponde al fallimento finito
  - L'insieme dei fatti negativi derivabili è più piccolo di quello ottenibile con CWA

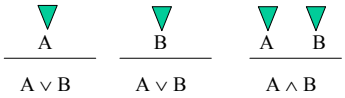
Goals esistenziali positivi.

- Possiamo ampliare in modo quasi immediato la forma delle conseguenze per cui si ha validità e completezza di (CK-Clark) con NAFF, ai goals esistenziali positivi
- Basta aggiungere alle regole per DCL le regole per and e or nella prossima diapositiva
  - implementate anche da Prolog, che usa , come and e ; come or

Goal esistenziali positivi:

? G(X): formula aperta costruita con $\wedge$, $\vee$

Implicitamente: ? $\exists$X G(X)



Con gli alberi di prova così arricchiti si ottiene validità e completezza rispetto ai goal esistenziali positivi, in questa forma:

KB $\models_{clark/CWA}$ $\exists$x G sse il goal G ha una risposta;

KB $\models_{clark/CWA}$ $\neg\exists$x G sse si ha fallimento finito

DCL: l'algoritmo top down esteso

- Ingresso: KB e atomo A? (anche aperto)
- Albero Prova := A?
- Fino a che vi è un F? e nessun F?FAIL
  - ***scegli*** un F?
    - se F è una congiunzione o una disgiunzione:
      - applica la regola ad essa relativa (pag. precedente);
    - se F è un atomo:
      - se esiste una clausola la cui testa unifica con F
      - ***trova*** k: H ← Body tale che H, F unifichino ;
      - ***applica k alla foglia F? selezionata***
    - se nessuno dei casi precedenti si applica, poni F?FAIL

rompe(X,Y) ← urta(X,Y) ∧ fragile(Y) ∧ pesante(X).
fragile(X) ← bicchiere(X) ∨ uovo(X).
pesante(X) ← X=martello ∨ X=pentola.
bicchiere(b).
uovo(u).
urta(martello,u).
urta(martello,pentola).

(pesante(X) ∧ rompe(X,Y)) ∨ fragile(X)

rompe(X,Y) ← urta(X,Y) ∧ fragile(Y) ∧ pesante(X).
fragile(X) ← bicchiere(X) ∨ uovo(X).
pesante(X) ← X=martello ∨ X=pentola.
bicchiere(b).
uovo(u).
urta(martello,u).
urta(martello,pentola).

pesante(X) ∧ rompe(X,Y)

(pesante(X) ∧ rompe(X,Y)) ∨ fragile(X)

rompe(X,Y) ← urta(X,Y) ∧ fragile(Y) ∧ pesante(X).
fragile(X) ← bicchiere(X) ∨ uovo(X).
pesante(X) ← X=martello ∨ X=pentola.
bicchiere(b).
uovo(u).
urta(martello,u).
urta(martello,pentola).

pesante(X) rompe(X,Y)

pesante(X) ∧ rompe(X,Y)

(pesante(X) ∧ rompe(X,Y)) ∨ fragile(X)

rompe(X,Y) ← urta(X,Y) ∧ fragile(Y) ∧ pesante(X).
fragile(X) ← bicchiere(X) ∨ uovo(X).
pesante(X) ← X=martello ∨ X=pentola.
bicchiere(b).
uovo(u).
urta(martello,u).
urta(martello,pentola).

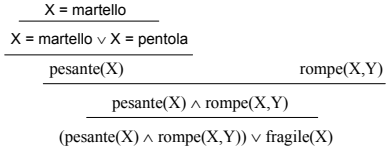
X = martello ∨ X = pentola

pesante(X) rompe(X,Y)

pesante(X) ∧ rompe(X,Y)

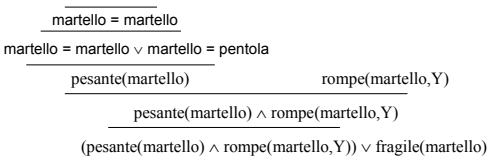
(pesante(X) ∧ rompe(X,Y)) ∨ fragile(X)

rompe(X,Y) ← urta(X,Y) ∧ fragile(Y) ∧ pesante(X).
fragile(X) ← bicchiere(X) ∨ uovo(X).
pesante(X) ← X=martello ∨ X=pentola.
bicchiere(b).
uovo(u).
urta(martello,u).
urta(martello,pentola).



rompe(X,Y) ← urta(X,Y) ∧ fragile(Y) ∧ pesante(X).
fragile(X) ← bicchiere(X) ∨ uovo(X).
pesante(X) ← X=martello ∨ X=pentola.
bicchiere(b).
uovo(u).
urta(martello,u).
urta(martello,pentola).

ESERCIZIO:
A) Andare avanti
B) Trovare tutti gli alberi di prova



Estensione problematica: Cosa succede con programmi definiti se il goal contiene atomi negativi?

- L'uso del mgu non garantisce più la completezza e la correttezza della procedura top-down non deterministica
- occorre ***un oracolo (non determinismo trova)***
  - nella scelta dell'unificatore da usare quando si incontra un letterale negativo (non più il mgu)
  - Come conseguenza, la scelta dell'atomo diventa problematica; l'unificazione con atomi positivi può aiutare a fare le scelte dell'unificatore

ESEMPIO

- Vediamo un esempio in cui, selezionando prima un goal non negativo e poi quello negativo istanziato si ha successo, mentre si ha fallimento se si seleziona subito il goal negativo; la seconda risposta è logicamente errata:
  - per avere la risposta corretta ci vuole l'oracolo, come vedremo con l'algoritmo top down completo

rompe(X,Y) ← urta(X,Y) ∧ fragile(Y).
pesante(X) ← X=martello ∨ X=pentola.
bicchiere(b).
fragile(u).
urta(martello,u).
urta(martello,pentola).

In blu
la formula selezionata

rompe(X,Y) ∨ (fragile(X) ∧ ¬rompe(X,Y))

rompe(X,Y) ← urta(X,Y) ∧ fragile(Y).
pesante(X) ← X=martello ∨ X=pentola.
bicchiere(b).
fragile(u).
urta(martello,u).
urta(martello,pentola).

fragile(X) ∧ ¬rompe(X,Y)

rompe(X,Y) ∨ (fragile(X) ∧ ¬rompe(X,Y))

rompe(X,Y) ← urta(X,Y) ∧ fragile(Y).
pesante(X) ← X=martello ∨ X=pentola.
bicchiere(b).
fragile(u).
urta(martello,u).
urta(martello,pentola).

fragile(X) ¬rompe(X,Y)

fragile(X) ∧ ¬rompe(X,Y)

rompe(X,Y) ∨ (fragile(X) ∧ ¬rompe(X,Y))

rompe(X,Y) ← urta(X,Y) ∧ fragile(Y).
pesante(X) ← X=martello ∨ X=pentola.
bicchiere(b).
fragile(u).
urta(martello,u).
urta(martello,pentola).

fragile(u) **¬rompe(u,Y)**

fragile(u) ∧ ¬rompe(u,Y)

rompe(u,Y) ∨ (fragile(u) ∧ ¬rompe(u,Y))

rompe(X,Y) ← urta(X,Y) ∧ fragile(Y).
pesante(X) ← X=martello ∨ X=pentola.
bicchiere(b).
fragile(u).
urta(martello,u).
urta(martello,pentola).

rompe(u,Y)

fragile(u) ¬rompe(u,Y)

fragile(u) ∧ ¬rompe(u,Y)

rompe(u,Y) ∨ (fragile(u) ∧ ¬rompe(u,Y))

rompe(X,Y) ← urta(X,Y) ∧ fragile(Y).
pesante(X) ← X=martello ∨ X=pentola.
bicchiere(b).
fragile(u).
urta(martello,u).
urta(martello,pentola).

rompe(u,Y)

urta(u,Y) fragile(Y)

rompe(u,Y)

fragile(u) ¬rompe(u,Y)

fragile(u) ∧ ¬rompe(u,Y)

rompe(u,Y) ∨ (fragile(u) ∧ ¬rompe(u,Y))

rompe(X,Y) ← urta(X,Y) ∧ fragile(Y).
pesante(X) ← X=martello ∨ X=pentola.
bicchiere(b).
fragile(u).
urta(martello,u).
urta(martello,pentola).

rompe(u,Y)

FAIL:urta(u,Y) fragile(Y)

rompe(u,Y)

fail

fragile(u) ¬rompe(u,Y)

fragile(u) ∧ ¬rompe(u,Y)

rompe(u,Y) ∨ (fragile(u) ∧ ¬rompe(u,Y))

rompe(X,Y) ← urta(X,Y) ∧ fragile(Y).
pesante(X) ← X=martello ∨ X=pentola.
bicchiere(b).
fragile(u).
urta(martello,u).
urta(martello,pentola).

rompe(u,Y)

FAIL:urta(u,Y) fragile(Y)

rompe(u,Y)

fail

fragile(u) ¬rompe(u,Y) **successo**

fragile(u) ∧ ¬rompe(u,Y)

rompe(u,Y) ∨ (fragile(u) ∧ ¬rompe(u,Y))

rompe(X,Y) ← urta(X,Y) ∧ fragile(Y).
pesante(X) ← X=martello ∨ X=pentola.
bicchiere(b).
fragile(u).
urta(martello,u).
urta(martello,pentola).

MA SE AVESSIMO SCELTO:

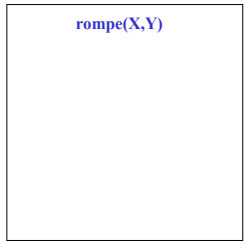
fragile(X) ¬rompe(X,Y)

fragile(X) ∧ ¬rompe(X,Y)

rompe(X,Y) ∨ (fragile(X) ∧ ¬rompe(X,Y))

---

rompe(X,Y) ← urta(X,Y) ∧ fragile(Y).
pesante(X) ← X=martello ∨ X=pentola.
bicchiere(b).
fragile(u).
urta(martello,u).
urta(martello,pentola).



fragile(X) ¬rompe(X,Y)

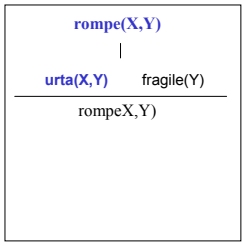
fragile(X) ∧ ¬rompe(X,Y)

rompe(X,Y) ∨ (fragile(X) ∧ ¬rompe(X,Y))

---

rompe(X,Y) ← urta(X,Y) ∧ fragile(Y).
pesante(X) ← X=martello ∨ X=pentola.
bicchiere(b).
fragile(u).
urta(martello,u).
urta(martello,pentola).

¬∧∨→←↔∀∃



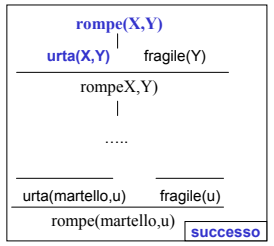
fragile(X) ¬rompe(X,Y)

fragile(X) ∧ ¬rompe(X,Y)

rompe(X,Y) ∨ (fragile(X) ∧ ¬rompe(X,Y))

---

rompe(X,Y) ← urta(X,Y) ∧ fragile(Y).
pesante(X) ← X=martello ∨ X=pentola.
bicchiere(b).
fragile(u).
urta(martello,u).
urta(martello,pentola).



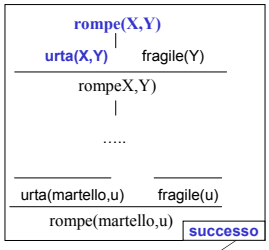
fragile(X) ¬rompe(X,Y)

fragile(X) ∧ ¬rompe(X,Y)

rompe(X,Y) ∨ (fragile(X) ∧ ¬rompe(X,Y))

---

rompe(X,Y) ← urta(X,Y) ∧ fragile(Y).
pesante(X) ← X=martello ∨ X=pentola.
bicchiere(b).
fragile(u).
urta(martello,u).
urta(martello,pentola).



FAIL: ¬rompe(X,Y)

fragile(X)

fragile(X) ∧ ¬rompe(X,Y)

rompe(X,Y) ∨ (fragile(X) ∧ ¬rompe(X,Y))

LOGICAMENTE ERRATO!

Estensione completa della procedura top down

- Ingresso: KB e formula F? (anche aperto)
- Albero Prova := A?
- Fino a che (vi è un F? e nessun F?FAIL) ***scegli*** un F?
  - se F è una congiunzione o una disgiunzione:
    applica la regola ad essa relativa ;
  - se F è un letterale negativo:
    trova un'istanza di F e ***applica NAFF***
  - se F è un atomo:
    - se esiste una clausola la cui testa unifica con F
      trova k: H ← Body tale che H, F unifichino ;
      applica k alla foglia F? selezionata
    - se nessuno dei casi precedenti si applica, ***poni*** **F?FAIL**

rompe(X,Y) ← urta(X,Y) ∧ fragile(Y).
pesante(X) ← X=martello ∨ X=pentola.
bicchiere(b).
fragile(u).
urta(martello,u).
urta(martello,pentola).

CON LA PROCEDURA COMPLETA:
Trovo X=u

fragile(X) **¬rompe(X,Y)**

fragile(X) ∧ ¬rompe(X,Y)

rompe(X,Y) ∨ (fragile(X) ∧ ¬rompe(X,Y))

rompe(X,Y) ← urta(X,Y) ∧ fragile(Y).
pesante(X) ← X=martello ∨ X=pentola.
bicchiere(b).
fragile(u).
urta(martello,u).
urta(martello,pentola).

ECC: FINO A SUCCESSO.
Ottengo una delle soluzioni possibili;
le altre con backtraking sulla sostituzione

fragile(u) **¬rompe(u,Y)**

fragile(u) ∧ ¬rompe(u,Y)

rompe(u,Y) ∨ (fragile(u) ∧ ¬rompe(u,Y))

- NOTA: ***trova*** : non-determinismo don't know; si può dare un algoritmo che decide se in un albero NAFF è stata applicata in modo logicamente corretto;
- si potrebbe dunque usare backtraking;
- ma siccome si possono avere infinite sostituzioni, si può avere ramificazione infinita;
  ciò rende la ricerca con backtraking non praticabile
- Una soluzione fattibile ma incompleta:

Estensione dei goal con atomi negati e regola di selezione di letterali negativi ground

- La parte scegli dell'algoritmo diventa:
  Scegli un F che non sia un letterale negativo non ground
- ***Si ha la correttezza, ma si perde la completezza della procedura top down;***
  - ovvero esistono prove che la procedura non trova perché si potrebbe arrivare ad un goal con un unico letterale negativo aperto e quindi floundering
- ***Floundering:*** viene scelto un letterale negativo aperto
  - Nell'esempio considerato prima si ha floundering, anche se la procedura non deterministica trova delle soluzioni

Clausole e Programmi Normali

- Le clausole normali sono clausole in cui è ammessa la negazione nel corpo
- I programmi normali usano clausole normali
- Si applica l'algoritmo top-down incompleto appena visto.
- Teorema di correttezza:
  - Se G ha successo con sostituzione di risposta σ in una computazione senza floundering, allora
    KB |= Gσ
- NOTA: Ora G può contenere letterali negativi

Cenni sul problema della insoddisfacibilità

- DEF. Una programma normale KB è ***insoddisfacibile*** sse comp(KB) non ha modelli.
- Se KB è un programma definito, comp(KB) non è mai insoddisfacibile
- Vi sono programmi normali con comp(KB) insoddisfacibile
  - Esempio: comp(p ← ¬p) = p ↔ ¬p

Cenno sul problema della completezza

- Con programmi normali quanto detto sulla completezza nel caso DCL salta
- La completezza si può ottenere con programmi di forma particolare
  - DCL come caso particolare estremo (non si usa negazione)
  - Programmi stratificati e altre classi di programmi

(HK) e vincoli

- Vi sono altri usi della negazione, utilizzata in modo diverso dal fallimento.
- Cominciamo con la negazione nel caso delle clausole di Horn
  - Da cui (HK), Horn Knowledge, nel libro di testo
- Le clausole di Horn estendono DCL con l'aggiunta di clausole del tipo
  - false ← Body

- Anche in questo caso si possono scrivere KB insoddisfacibili, cioè prive di modelli, ad es.:

  false ← p.

  p.
- Proof Theory: quella di DCL
- Def. KB ***inconsistente*** sse esiste un albero di prova di false
- **Teorema:** KB è inconsistente sse è insoddisfacibile

Applicazioni di (HK): vincoli di integrità

- In una KB, distinguiamo fra atomi assumibili o aperti, e atomi definiti o chiusi:
  - **Assumibili:** ci riteniamo liberi di interpretarli, cioè di assumerli come veri o falsi in varie circostanze
  - **Definiti:** non siamo liberi di interpretarli, perché sono completamente definiti dalle clausole, in funzione di quanto assumiamo sugli assumibili.
- **Vincoli:** clausole di KB della forma
  - false ← Body.
  - Vincolano l'insieme delle assunzioni possibili a quelle consistenti con KB

Applicazione 1. Vincoli in basi dati

- Considerare VDB (Vincoli su base dati) che ha come assumibili le relazioni dello schema di base dati e fornisce dei vincoli di integrità su di essa
- Ogni specifica istanza dello schema fornisce una interpretazione specifica degli assumibili
- Le interpretazioni che verificano i vincoli di integrità sono quelle che non derivano false
- Conviene error(X) per spiegare i motivi delle violazioni individuate: X può essere l'albero di prova di false o altra informazione più succinta

Esempio

- error(sesso(X)) :- maschio(X), femmina(X).
- Se la base dati contiene

  maschio(gigi).

  femmina(gigi).
- L'interrogazione error(Errore) fornisce:

  Errore = sesso(gigi).

Applicazione 2. Diagnosi Guasti

- Mentre i vincoli di integrità in basi dati usano la ***deduzione*** di false (o di error(..)), nella diagnosi si applica ***l'abduzione;***
- Gli assumibili sono i predicati relativi al corretto funzionamento delle parti che si possono guastare; tipicamente, conviene usare
  - ok(k) la parte k non è guasta
  - e usare ok come assumibile
- Nel caso di un guasto, una ***diagnosi*** sarà:

  $\neg ok(k1) \vee \ldots \vee \neg ok(kn)$

  Cioè: uno o più dei dispositivi k1, ..,kn si sono guastati