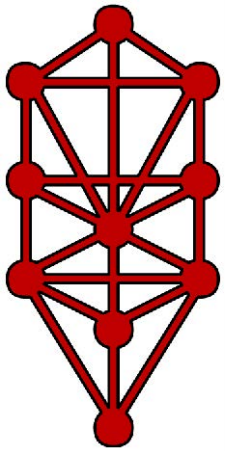
This page is intentionally left blank.

Kazimir Majorinc

DISKURZIVNI PREGLED PLANNERA

Povijest Lispa 43.



Razmjena vještina
Hacklab u mami
14. prosinca 2013.

Treće poglavlje u **Hewittovoj** doktorskoj radnji iz **1972**. Discursive - talking or writing about many different things in a way that is not highly organized.

Najjednostavniji način za razumijevanje PLANNERA je da se pogleda kako planner radi. Hewitt pokazuje nekoliko primjera koji nisu zamišljeni kao „toy problemi,“

Primjer 1. Iz sudova

Turing je čovjek.

Svi ljudi su griješni.

proizlazi

Turing je griješan.

U PLANNERu se može upisati.

```
<ASSERT [HUMAN TURING]>
<ASSERT <DEFINE THEOREM1
                <CONSEQUENT [Y]
                            [FALLIBLE ?Y]>
                            <GOAL [HUMAN ?Y]>>>>
```

Dokaz da je Turing griješan može se generirati upisom

```
<GOAL [FALLIBLE TURING]>
```

Primjer pokazuje važne osobine PLANNERA:

(1) postoje bar dvije različite vrste informacija spremljenih u bazi podataka, deklarativne i imperativne.

(2) PLANNER izračunava naredbe, proizvodeći vrijednost i side-effect.

```
<ASSERT [HUMAN TURING]>
```

ASSERT je funkcija koja unosi argument u bazu tvrdnji (data base of assertions).

```
<ASSERT <DEFINE THEOREM1
                <CONSEQUENT [Y]
                            [FALLIBLE ?Y]>
                            <GOAL [HUMAN ?Y]>>>>
```

U ovom primjeru, u bazu podataka se unosi i teorem tipa **CONSEQUENT**.

Teorem tvrdi ako želimo pokazati da **[FALLIBLE ?Y]** to možemo učiniti postavljanjem cilja **[HUMAN ?Y]** gdje je **Y** identifikator. Čudan znak **?** je dio PLANNEROVOG pattern-matching jezika.

Ako se od PLANNERa traži da dokaže `[A Y]`, ne postoji način na koji se može znati jesu li `A` i `Y` konstante (kao `HUMAN` i `TURING`) ili varijable (Hewitt koristi riječ identifikatori). LISP riješava taj problem korištenjem funkcije `QUOTE` koja označava konstantu. Ova je mogućnost nepogodna za „pattern matching“ (Hewitt ne kaže zašto) i zato PLANNER radi obratno - dodavanjem `?` se pokazuje da je izraz varijabla. U `THEOREM1` postoji samo jedan identifikator - `Y`.

Natrag na primjer:

```
<ASSERT [HUMAN TURING]>
<ASSERT <DEFINE THEOREM1
                <CONSEQUENT [Y]
                            [FALLIBLE ?Y]>
                            <GOAL [HUMAN ?Y]>>>>
<GOAL [FALLIBLE TURING]>
```

Da smo pitali `<GOAL [HUMAN TURING]>`, PLANNER bi potražio rješenje u bazi podataka i odmah ga pronašao. Ali, `[FALLIBLE TURING]` nije upisano nigdje u bazi podataka, pa ga treba dokazati.

Naredba `GOAL` može dati niz savjeta kako izvesti teoreme.

Natrag na primjer:

```
<ASSERT [HUMAN TURING]>
<ASSERT <DEFINE THEOREM1
                <CONSEQUENT [Y]
                            [FALLIBLE ?Y]>
                            <GOAL [HUMAN ?Y]>>>>
<GOAL [FALLIBLE TURING]>
```

Kako se izvršava program? Prvo se pokušava „postići cilj“

```
<GOAL [FALLIBLE TURING]>
```

`[FALLIBLE TURING]` nije u bazi podataka

poziva se `THEOREM1` i supstituira `TURING` umjesto `Y`.

`<GOAL [HUMAN TURING]>` je zadovoljen

return `[FALLIBLE TURING]`.

Primjer 2:

```
<ASSERT [HUMAN TURING]>
<ASSERT <DEFINE THEOREM1
                <CONSEQUENT [Y]
                            [FALLIBLE ?Y]>
                            <GOAL [HUMAN ?Y]>>>>
```

Želimo saznati `[EXISTS X [FALLIBLE X]]` .

Taj se izraz u PLANNERu prevodi sa

```
<PROG [X] <GOAL [FALLIBLE ?X]>>
```

PLANNER prvo traži `[FALLIBLE ?X]` u bazi podataka; ako ne nađe, onda traži u `CONSEQUENTU` teorema,

Zatim pokušava rješiti `[HUMAN ?Y]`.

i pronalazi `[HUMAN TURING]`. Iz toga izvodi prema teoremu `[FALLIBLE TURING]`.

Primjer 3:

Dodajmo u bazu podataka dvije tvrdnje:

```
<ASSERT [HUMAN SOCRATES]>
<ASSERT [GREEK SOCRATES]>
```

Nakon toga u bazi podataka imamo tvrdnje

```
[HUMAN TURING]
[HUMAN SOCRATES]
[GREEK SOCRATES]
```

i `THEOREM1`

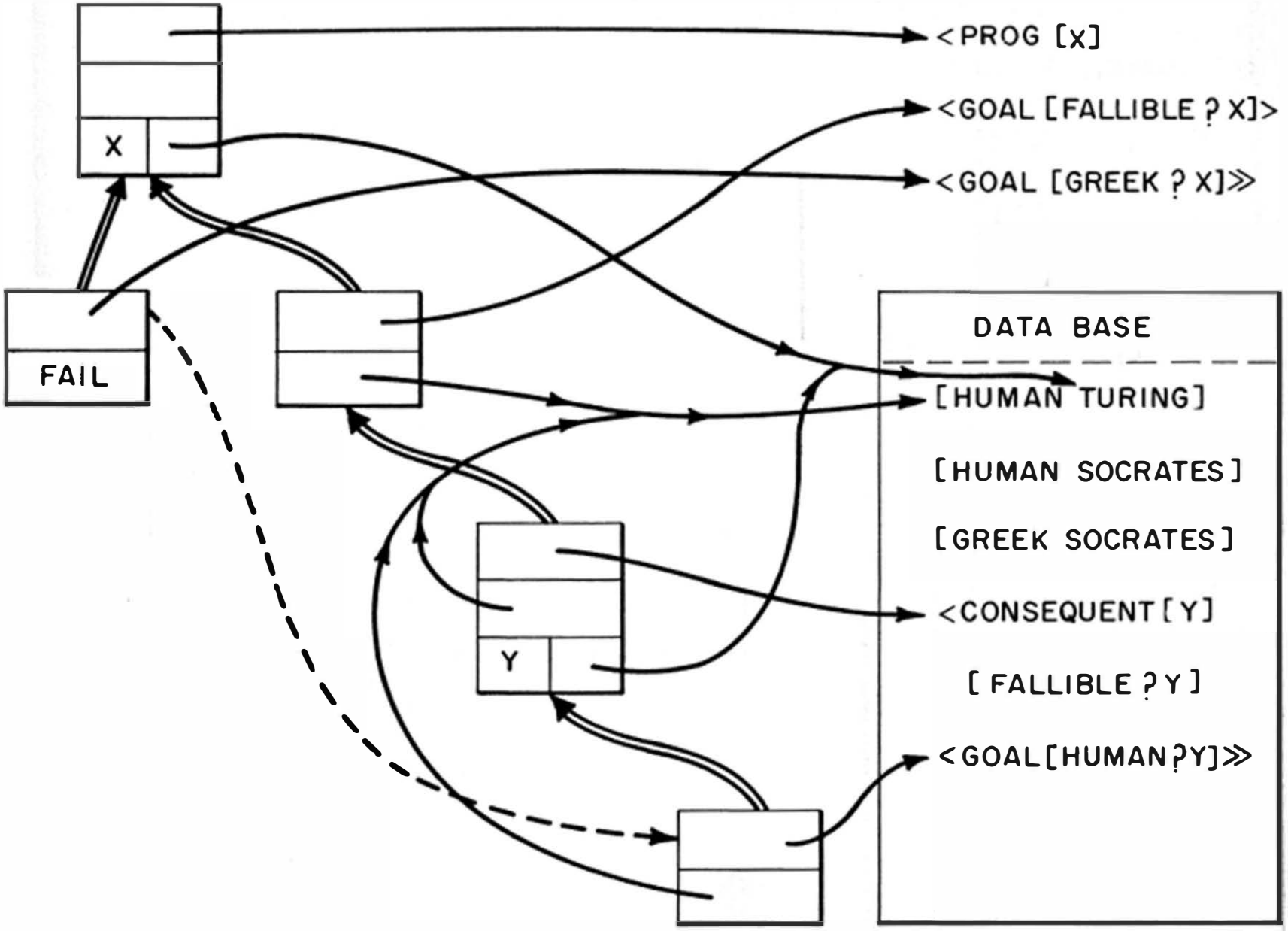
```
<CONSEQUENT [Y]
            [FALLIBLE ?Y]>
            <GOAL [HUMAN ?Y]>>
```

Želimo pitati PLANNERA „postoji li griješni Grk“ .(??)

```
<PROG [X]
      <GOAL [FALLIBLE ?X]>
      <GOAL [GREEK ?X]>>
```

Ako PLANNER dođe do greške računajući izraz, tada se vraća natrag. U ovom slučaju, prvi cilj **[FALLIBLE ?X]** može biti zadovoljen isto kao prije. No, ako PLANNER ostvari prvi cilj pomoću **[FALLIBLE TURING]** onda će PLANNER pokušati izračunati **[GREEK TURING]**, tu će propasti (zato što to nije tvrđeno). U LISP-u bi to bio problem, jer je izračunavanje prvog GOAL-a već završilo, ali PLANNER se vraća i pokušava zadovoljiti **<GOAL [FALLIBLE ?X]>**, na drugi način. Uz pomoć teorema nalazi **[FALLIBLE SOCRATES]**, provjerava **[GREEK SOCRATES]** što uspijeva, i završava sa rezultatom **[GREEK SOCRATES]**.

SNAPSHOT NO. 2



kraj