APPENDIX TWO
Creating a Database

The use of the discrimination net to store formulas requires that the most informative part of the formula occur before any arguments. Thus we want to write (hemoglobin-level Jones 1.5). But we would like to use pretty-formulas like "(Jones has a hemoglobin-level of 1.5)". This is accomplished by using the *reform-list*, which is a list of triples like

```
`((,patient has a hemoglobin-level of ,degree)
      (hemoglobin-level ,patient ,degree) (,patient ,degree))
```

These triples are constructed by first deciding on a pretty-formula, then apply reform to it to obtain the first member of the triple, and then letting the second member be the formula it should reform into. To avoid possible variable conflicts, the variables *patient* and *degree* are taken to be gensyms. Thus we add triples to the *reform-list* using code like the following:

```
(let ((patient (gensym "patient")) (heart-rate (gensym "heart-rate")) (drug (gensym "drug"))
      (P (gensym "P")) (degree (gensym "degree")) (Q (gensym "Q")) (x (gensym "x"))
      (y (gensym "y")) (A (gensym "A")) (r (gensym "r")))
 (let
   ((pairs
     (list
       `((,patient has a severe headache to degree ,degree on a scale of 1-5)
         (headache ,patient ,degree) (,patient ,degree))
       `((,patient has ingested cocaine) (ingested-cocaine ,patient) (,patient))
       `((,patient has ingested amphetamines) (ingested-amphetamines ,patient) (,patient))
       `((,patient has ingested antihistamines) (ingested-antihistamines ,patient) (,patient))
       )))
   (setf pairs (subset #'(lambda (x) (not (member x *reform-list* :test 's-equal))) pairs))
   (dolist (pair pairs) (push pair *reform-list*))))
```

Standing plans are constructed using the following macro:

(def-standing-plan *name*
:strength *number*
:defeasible? *T or F*
:variables *parameter-variables*
:forwards-conditions
:preconditions *identification of parameters and conditions on parameters*
:goal-type *string*
:goal-type-variables *variables*
:description *string*
:plan-generator *optional function that can be entered in place of the following five slots*
:expected-value *expression that evaluates to a number*
:plan-message *string*
:user-question *string*
:user-question-type *y/n or m/c or nil*
:user-question-choices
:completion-question *string*
:completion-question-type *y/n or m/c or nil*
:completion-question-choices
:completion-formulas *optionally, two strings, the first,continaining the variable 'x' and*

possibly containing parameter-variables and the second possibly containing parameter-variables. The first string is the form of a formula which records the information given in the answer to the completion-question and the second string reports that the plan has been executed.
)

Here are two examples:

```
(def-standing-plan First-plan-to-eliminate-tachycardia
   :strength .98
   :defeasible? t
   :variables  patient heart-rate
   :preconditions
   "(patient is experiencing tachycardia with a heart rate of heart-rate)"
   "~(patient has subnormal blood pressure)"
   :goal-type "(Bring patient out of tachycardia)"
   :goal-type-variables  patient
   :description
   "If a patient is experiencing tachycardia with a heart rate greater than 100, and does not have
 subnormal blood pressure, it is desirable to administer thiamine."
   :expected-value  (if (> heart-rate 100) (goal-strength goal) 0)
   :plan-goals (list goal)
   :plan-message "Administer thiamine to patient."
   :completion-question
   "Once thiamine has been administered, is patient still experiencing tachycardia?"
   :completion-question-type y/n
   :completion-formulas
   "(patient is experiencing tachycardia with a heart rate of heart-rate)"
   "(thiamine has been administered to patient)")

(def-standing-plan test-for-hemoglobin-level
   :strength .98
   :defeasible? t
   :variables  patient
   :forwards-conditions
   :preconditions
   :goal-type  "(I know that (patient has a hemoglobin-level of level))"
   :goal-type-variables  patient level
   :description
   :plan-generator
   :expected-value  (goal-strength goal)
   :plan-message  "Perform the text for hemoglobin-level."
   :user-question  "What is the patient's hemoglobin-level?"
   :user-question-type
   :user-question-choices
   :completion-question  "What does the hemoglobin test reveal the patient's hemoglobin-level to be?"
   :completion-question-type
   :completion-question-choices
   :completion-formulas
   "(patient has a hemoglobin-level of x)"
   "(patient has been tested to determine his/her hemoglobin-level)" )
```

If there is no completion-question, there can still be a single completion-formula reporting information that will be encoded as a percept when the plan has been completed. Note that the free variable in the first completion-formula must be 'x'. It is bound to the answer in any question other than a y/n question.

Goals are constructed using the following macro:

(def-treatment-goal *name*
:description *description-string*
:goal-formula *string*
:goal-strength *expression that evaluates to a number*
:observations *observation-schemas*
:goal-generator *optional goal-generator-function, replaces goal-formula and goal-strength*
:strength *strength-of-the-reason*
:variables *parameter-variables*)

Here are two examples:

```
(def-treatment-goal to-alleviate-tachycardia
   :description "If the patient is experiencing tachycardia, treat him for it."
   :goal-formula "(Bring patient out of tachycardia)"
   :goal-strength  (/ heart-rate 200)
   :observations "(patient is experiencing tachycardia with a heart rate of heart-rate)"
   :variables  patient heart-rate
   :strength 1.0
   :diagnosis "(patient is experiencing tachycardia with a heart rate of heart-rate)")

(def-treatment-goal  to-alleviate-cocaine-ingestion
   :description  "If the patient has ingested cocaine, treat him for it."
   :goal-formula "(treat patient for cocaine ingestion)"
   :goal-strength  (* cocaine-prob 2)
   :observations
     "(the PROB of (patient has ingested cocaine) is cocaine-prob)"
   :goal-generator
   :strength  .95
   :variables  patient cocaine-prob
   :diagnosis "(patient has ingested cocaine)")
```

For entry purposes, I recommend cutting and pasting from a template, and using boldface as above to facilitate reading the resulting code.

Probability relations are entered into *background-knowledge* using the following macro:

(def-prob-rule
:consequent *a string*
:reference-formulas *a list of strings*
:schema-variables *the list of variables bound by the probability operator*
:premise-variables *the list of variables standing for parameters*
:value *a number, string with arithmetical operators in infix notation, or a lisp expression the evaluates to a number given the values of the parameters in it.*)

Here are two examples:

```
(def-prob-rule antihistamine-ingenstion3
   :consequent  "(x has ingested antihistamines)"
   :reference-formulas
  "(x has dry skin to degree y on a scale of 1-5 )"
  "(x has a severe headache to degree z on a scale of 1-5)"
  "~(x is suicidal)"
   :schema-variables  x
   :premise-variables  y z
   :value  .4)

(def-prob-rule antihistamine-ingenstion4
```

```
   :consequent  "(x has ingested antihistamines)"
   :reference-formulas
  "(x has dry skin to degree y on a scale of 1-5 )"
  "(x has a severe headache to degree z on a scale of 1-5)"
  "(x is suicidal)"
  "(x has a hemoglobin-level of h)"
   :schema-variables  x
   :premise-variables  y z h
   :value  (if (> h 1) .9 .7))
```

The first produces the probability:

(∀y)(∀z)(the probability (x) of (x has ingested antihistamines)
given
((x has dry skin to degree y on a scale of 1-5)
& ((x has a severe headache to degree z on a scale of 1-5)
& ~(x is suicidal)))
is 0.4)

The second produces the probability

(∀y)(∀z)(∀h)(the probability (x) of (x has ingested antihistamines)
given ((x has dry skin to degree y on a scale of 1-5)
& ((x has a severe headache to degree z on a scale of 1-5)
& (~(x is suicidal)
& (x has a hemoglobin-level of h))))
is (IF (> h 1) 0.4 0.3))

Note that you cannot use 'v' as a variable. It is reserved for use in symbolizing disjunction.
A sample database is contained in Sudafed/Cocaine_3.13.