EPILOG: The Computational System for Episodic Logic

QUICK REFERENCE GUIDE

Stephanie Schaeffer

Chung Hee Hwang

John de Haan

Lenhart K. Schubert

August 1993

Revised September 2000

Prepared for Boeing Computer Services, Seattle, Washington

Under Purchase Contract W-278258

Contents

Chapter 1

Logical Syntax

To read this syntax summary, note the following: '*' means 0 or more occurrences, '+' means 1 or more occurrences, '|' and ',' mean choices (as do separate lines for multiline definitions) and{} indicates optionality. Items in *italics* are syntax types, in **bold** are actual input (the **()** are included although they don't really look bold), regular print includes syntax instructions, and comments.

wff->(**(**{negation}*quantifier variable*{*wff*}*wff*)
(**(**{*negation*}*term pred term**)
(**(***wff-op wff*)
(**(**{*negation*}*wff logical-conn wff*+)
(**(**{*negation*}*wff episodic-op term*)
(**(**{*negation*} *wff causal-conn wff*)
(**(**{*negation*} *wff* **true**)

wff-op -> *name* examples: **nec, poss, probably, perhaps, past, perf, futr, pres, prog**, ...
(*sentence-modifier pred*)
Note:*pred* is a 1 place predicate

sentence-modifier->*name*examples:**adv-s, adv-e, adv-f, adv-p** , ...

quantifier->**A**, **E**, **the**, **most**, **many**, **some**, **few**, **none**
(*quantifier-modifier quantifier*)

quantifier-modifier-> *name*examples:**nearly**, ...

variable->*name*|*name_sort*

sort->**episode**, **ep**, **event**, **set**, **time**, **number**, **num**, **real**, **integer**, **int**, **string**, **propos**

negation->**not**

logical-conn->**and**, **or**, **implies**, <=>, *number-pred*
(*number-pred variable+*) controlled variables

episodic-op-> ******, *****, **@**, _

causal-conn-> **because**

name-> lisp symbol name, consisting of a string of characters and numbers,
starting with a character

number-pred-> real number <= 1

term -> constant
(*pred-nominalization-op pred*)
(*sentence-nominalization-op wff*)
(*function term+*)
quasi-quoted-expression
record
quoted-expression

*pred-nominalization-op->name*examples:**K, K1, Ka, To,**...

*sentence-nominalization-op-> name*examples:**that, whether, Ke, YN-q**, ...

record->**($ ´sort constant+)**

quoted-expression-> quoted list, contents unspecified [1]

quasi-quoted-expressionR ->**(qquote***wff***)|(qq***wff***)**
(qquote*term***)|(qq***term***)**

constant->*name*|*name_sort*| number | string

function->*name*examples:**set-of, date, cardinality-of, start-of, pair, fst, rst,** ...

pred->*name*examples:**kill**, **love**, **eat**, **pretty**, ...
(*pred-modifier pred*)
(*multi-pred-modifier pred+*)
lambda-pred

lambda-pred->*lambda-expr*
(*lambda-pred term**)

lambda-expr->**(L***variable wff***)|(L***variable pred***)**

pred-modifier->*number*|*name*examples:**very, plur, coll, almost, sort-of, former, in-manner, ly** ...
(*modifier-forming-op pred*)

modifier-forming-op->*name*examples:**coll-of, attr, adv-a, nn, na, adv-q**, ...

*multi-pred-modifier-> name*examples:**rel, mos,** ...

In addition, each subpart may be named using the symbol ! and a name just before the closing bracket. These names may then be used instead of the whole expression wherever the expression is legal. For example, *(A x (x wolf ! p1) (x grey) ! p2)*would mean *p2* could now be used for *(Ax (x wolf) (x grey))*, and *p1* could be used for *(x wolf)* - e.g. *(A x p1 (x fierce))*.

The thesis description of episodic logic has the negation operator acting on a sentence argument, rather than inside the sentence. EPILOG will accept this form as well and move the negation inside.

Notes: Some operators are stored but not used for inference yet. Infix position is used for wffs themselves, prefix for all other constructions. Wherever a syntax type is defined as *type -> name* , the user may add his own names there using the *add-predicate, add-operator,* etc functions. Where there is a fixed set, no new ones may be added (logical

[1] Note that no meta variables (variables over wffs, predicates, etc) may be quantified within a quasi-quoted expression. Any quantification outside a quasi-quoted expression in a meaning postulate or simplification schema must be over a meta variable or a sorted variable.

connectives, quantifiers, etc). Controlled variables are stored but not currently involved in inference.

This syntax is really somewhat more permissive than intended. For example, the syntax ignores predicate adicity, so it permits [John gives], (very gives), and other oddities. For more details on particular constructions, see the "Logical Syntax Details" section in the User Manual.

Some examples (each followed by the system's attempt at English generation):

*(A x (x wolf) (A y (y human) (A z_ep ((x meet y) * z) ((y in-danger) @ z))))*

If someone is met by a wolf, he is in some danger.

(not wolf1 friendly)

WOLF1 is not friendly.

(nec (A x (x wolf) (x fierce)))

Necessarily wolves are fierce.

((lrrh pretty) and (lrrh friendly))

Little Red Riding Hood is friendly and pretty.

*((wolf1 meet lrrh) ** ep1_episode)*

WOLF1 met Little Red Riding Hood.

*(((lrrh in-danger) @ ep2_ep) because ((wolf1 meet lrrh) * ep1))*

Little Red Riding Hood was in some danger because she was met by WOLF1.

*((wolf1 want (To (L x (E y_ep ((x eat lrrh) * y))))) ** ep3_ep)*

WOLF1 wanted to eat Little Red Riding Hood.

(lrrh (mos pretty girl))

Little Red Riding Hood is the most pretty (prettiest) girl.

((start-of ep2) during ep1)

The start of WOLF1 wanted to eat Little Red Riding Hood while WOLF1 met her.

*((wolf1 ((ly quick) eat) gm) ** ep4_ep)*

WOLF1 quickly ate Grandmother.

Chapter 2

EPILOG Commands

Note: Abbreviations for commands are shown in brackets after the command description. Fuller descriptions of the commands and their arguments are given in the User's Guide. Note that many more queries are handled by the display command - see the display options in Chapter 5.

2.1 Loading knowledge and story

(**KNOWLEDGE** {*formula*}+) (**KN**)
- asserts knowledge in the form of rules

(**GOAL-KNOWLEDGE** {*formula*}+) (**GOAL-KN, GKN**)
- like **kn** , but rules for questions only

(**STORY** {*formula*}+)
- asserts story facts

(**REASSERT** *phi-name*)
- repeats input-driven inferencing for a wff

(**MEANING-POSTULATE** {*formula*}+) (**MP**)
- asserts a meaning-postulate (axiom schema)

(**SIMPLIFICATION-SCHEMA** {*formula*}+) (**SIMP-SCHEMA, SS**)
- asserts a simplfication schema

(**META** {*formula*}+)
- asserts meta knowledge - knowledge about predicates and operators

2.2 Queries

(**DISPLAY** *arguments*)
- displays info desired by arguments

(**RETRIEVE** *concept topic super sub subnets*) (**RET**)
- retrieves specified info from knowledge base

(**GET-EVERYTHING** *subnet print-subnets*)
- gets all wffs in system and returns them in a list

2.3 Question answering

(**QUESTION** *formula number effort*) (**Q**)
- asks the system a yes/no or wh question

(**PROOF-Q** *formula number effort*) (**PQ**)
- asks the system a question and forces it to concentrate only on a YES answer

(**DISPROOF-Q** *formula number effort*) (**DQ**)
- asks the system a question and forces it to concentrate only on a NO answer

2.4 Response Generation

(**DO-SAY** *arguments*)
- "says" the requested info (answer, info about a concept, etc)

(**SAY-IT** *wff-list*)
- "says" the list of wff names given as a sentence

(**SAY-THEM** *list-of-lists*)
- "says" a series of sentences with one list of wff names per sentence

(**ADD-WORD** *word trans*)
- adds translation information for a predicate or operator

(**ADD-LEX** *word type &key present past negative-present negative-past passive prespart plural props trans subject-pronoun object-pronoun*)
- adds lexical entries for particular verbs, nouns, etc.

2.5 Specialists

(**USE-SPEC** *specialists*)
- activates a specialist

2.6 Miscellany

(**TRACE-ITEM** *items*)
- starts tracing given items

(**UNTRACE-ITEM** *items*)
- stops tracing given items

(**TRACEABLE** *item description trace-values*)
- sets up an item for tracing

(**TRACE-ALL**)
- starts tracing on everything

(**UNTRACE-ALL**)
- stops tracing on everything

(**TWEAK** *item value*)
- changes the value of a parameter

(**CHECKPOINT** *item*)
- starts a checkpoint so that info may be retracted to it

(**RETRACT** *item*)
- retracts all assertions back to given checkpoint

(**HELP** *topic*)
- prints help information

(**ESH**)
- starts up the shell interface

(**WRITE-PERM-MEMORY** *file-name*) (**WPM**)
- saves all the current formulas in a file (given and inferred)

(**READ-PERM-MEMORY** *file-name*) (**RPM**)
- reads in a file saved by **write-perm-memory**

2.7 Adding New Syntactic Entities

(**ADD-HIER** *hier-name main-node sub-nodes*)
- adds new hierarchies and hierarchy nodes (predicates)

(**ADD-PART-HIER** *hier-name main-node sub-nodes*)
- adds new part hierarchies and part hierarchy nodes (predicates)

(**ADD-INDICATE** *predicate indicator-list*)
- adds topic indicators to the predicate

(**ADD-TOPIC** *topic predicate-list*)
- adds the topic as an indicator for each predicate

(**SET-HIER-TYPE** *hier-name type*)
- adds hierarchy properties to a hierarchy

(**ADD-INTEREST** *node interest*)
- adds interest levels to topics or predicates

(**ADD-OPERATOR** *operator &key op-type result-type arg-types indicators specialists*)
- adds new operators to the system

(**ADD-OPERATOR-TYPE** *op-type &key result-type arg-types*)
- defines operator types

(**ADD-PREDICATE** *pred &key type part hier subnodes parent hier-type sort indicators specialists package entry-rtn*)
- adds new predicates

(**ADD-FUNCTION** *fnname &key rel-pred specialists*)
- adds new functions to the system

(**ADD-SORT** *sort &key nicknames*)
- adds new sorts to the system

(**ADD-QUANTIFIER** *quant &key prob negation distributive*)
- adds new quantifiers to the system

Chapter 3

EPILOG Trace Values

Tracing of items is turned on using command **trace-item** and turned off using command **untrace-item** . Initially when the system starts up, compound trace items *qa* and *forward* are automatically being traced, which means that input-driven inferences made will be displayed, and when questions are asked the answer(s), number of iterations required, and time required will be displayed. Current items which may be traced:

3.1 Main System Operations

3.1.1 Normalization

lambda - shows lambda conversion

normalize - shows normalization

3.1.2 Application of Simplification Schemas

simplfication-schema - shows replacement of wff by simplification schemas

For more detailed tracing, use the meaning postulate trace values. They also operate during the application of simplification schemas.

3.1.3 Classification and Storage

class - traces start and results of classification

entry-class - traces classifications used to store each wff input to or inferred by the system.

keys - traces selection of trigger keys for rules

memory - traces expansion of input array

3.1.4 Forward Inference

forward - traces forward inferences as they are made

forward-details - shows parent formulas when wff printed

infer-details - shows details of inference decision (for debugging)

forward-eval - traces evaluation of literals during forward inferencing

forward-qa - traces min-question attempts during forward inference

forward-test - shows comparisons between keys during forward inference

forward-access - shows accesses of formulas during forward inference

forward-class - shows classifications checked during forward inference

forward-int - shows interesting parts of forward inference (everything except classification)

forward-min - shows the inferences made, and evaluations during forward inferencing

forward-all - shows everything to do with forward inferencing, including *forward, forward-access, forward-test, forward-eval, forward-qa* and *forward-class* .

3.1.5 Meaning Postulate Inference

mp-eval - traces evaluation of literals during meaning postulate inferencing

mp-test - shows comparisons between keys during meaning postulate inference

mp-access - shows accesses of formulas during meaning postulate inference

mp-class - shows classifications checked during meaning postulate inference

mp-all - shows everything during meaning postulate inference

mp-int - shows interesting parts of meaning postulate inference (everything except classification)

mp-min - shows the inferences made, and evaluations during meaning postulate inferencing

3.1.6 Entry and Assertion in General

entry-eval - traces verifications done during input

entry-time - keeps track of how long it takes to enter a story formula, including all the forward inferences.

entry - shows interesting parts of story entry, including *forward* and *entry-time* .

entry-all - shows everything to do with an input assertion, including all forward and mp inference stuff.

entry-int - shows interesting stuff for input assertions, including *forward-int* and *mp-int* .

entry-min - shows minimum stuff to see what happens with input assertions, including *forward-min* and *mp-min* .

3.1.7 Question Answering

subgoal -shows splitting of subgoals

qa-test - shows tests for (in)compatibility between set-of support clauses and accessed clauses

qa-success -shows successful subgoal actions during the qa process

qa-access -shows access actions done during the qa process

qa-eval -shows evaluations and simplifications that take place during the qa process

qa-time - Keeps track of how much time each question takes

qa-iterations - Keeps track of how many iterations each question takes.

qa-answer - Displays the answers to the question

qa - Traces the minimum essentials for question answering - the time and iterations, and the answer

qa-all - Traces everything during qa, including *qa-time, qa-success, qa-access, qa-eval, qa-test, subgoal*

qa-int - Traces the minimum to see how the question was answered, including *qa-success, qa-answer, subgoal, qa-time, qa-iterations* .

qa-after - Prints out the successful subgoal AFTER a question has been answered.

3.2 Specialists

3.2.1 Specialist Interface

spec-test - Traces comparisons between literals handed to the interface.

spec-entry - shows literals handed to the interface for entry into

spec-eval - show literals handed to the interface for specialists to evaluate

spec-assert - show assertions made by specialists back to EPILOG

interested-party - Traces addition to interested party lists, and reassertion of literals from there.

function-eval - Traces evaluation of functional arguments by the specialists.

spec-int - traces all aspects of the specialist interface, including *spec-test, spec-entry, spec-eval, spec-assert, interested-party,* and *function-eval* .

3.2.2 Type Specialist

type-test -shows comparisons between type predicates

3.2.3 Episode Specialist

episode-test -shows comparisons between episodic predicates

3.2.4 Predicate Hierarchy Specialist

hier-test -shows comparisons between predicates on the hierarchy

3.2.5 Part Specialist

part-entry - Traces input of part-of relations

part-eval - Traces evaluation of part-of relations

part-assert - Traces assertions made by the part specialist

part-test - Traces literal comparisons made by the part specialist

part-min - Traces interesting things about parts, including *part-entry, part-eval, part-assert, and part-test*

part-all - Traces everything about parts, including *part-entry, part-eval, part-assert, and part-test*

3.2.6 Time Specialist

time-entry - Traces input of temporal relations

time-point-entry - Traces input of point relations

time-eval - Traces evaluation of temporal relations

time-function-eval - Traces evaluation of temporal functions

time-point-eval - Traces evaluation of point relations

abs-time-entry - Traces entry of dates

abs-time-eval - Traces evaluation of date relationships

time-duration-entry - Traces entry of durations

time-duration-eval - Traces evaluation of durations

time-search - Traces metagraph searching

time-test - Traces comparisons of literals

time-all - Traces all time operations, including *time-entry, time-eval, time-point-entry, time-point-eval, abs-time-entry, abs-time-eval, time-duration-entry, time-duration-eval, time-search* , and *time-test* .

time - Traces interesting time stuff, including *time-entry, time-eval, time-function-eval, time-point-entry, time-point-eval, abs-time-entry, abs-time-eval, time-duration-entry, time-duration-eval, and time-test* .

time-min - Traces basic time stuff, including *time-entry, time-eval, time-function-eval,* and *time-test* .

3.2.7 Number Specialist

number-entry - Traces entry of number relations

number-prop - Traces propagation of number relations

number-eval - Traces evaluation of number relations

number-function-eval - Traces evaluation of number functions

number-search - Traces searching through the number graph

number-compare - Traces comparison of literals using the number specialist

number-all - Traces all number operations, including *number-entry, number-eval, number-prop, number-compare number-function-eval* , and *number-search* .

number-min - Traces basic number operations, including *number-entry, number-compare, number-function-eval,* and *number-eval* .

3.2.8 Color Specialist

color-test - Traces comparison of predicates using the color specialist

color-details - Traces details of comparison of predicates using the color specialist

color-all - Traces all color specialist operations, including *color-test* and *color-details* .

3.2.9 Equality Specialist

equality-entry - Traces entry of set membership and equality relationships, and cardinality information.

equality-eval - Traces evaluation of equality relationships,

equality-function-eval - Traces evaluation of equality functions

equality-test - Traces comparison of literals within the equality specialist.

equality-min - Traces interesting things in the equality specialist, including *equality-eval, equality-entry, equality-function-eval* and *equality-test* .

equality-all - Traces everything about the equality specialist, including *equality-eval, equality-entry, equality-function-eval* and *equality-test* .

3.2.10 Set Specialist

set-entry - Traces entry of set membership and equality relationships, and cardinality information.

set-eval - Traces evaluation of set membership and equality relationships,

set-function-eval - Traces evaluation of set functions

set-test - Traces comparison of literals within the set specialist.

set-assert - Traces assertions made by the set specialist.

set-min - Traces interesting things in the set specialist, including *set-eval, set-entry, set-assert, set-function-eval,* and *set-test* .

set-all - Traces everything about the set specialist, including *set-eval, set-entry, set-assert, set-function-eval,* and *set-test* .

3.2.11 String Specialist

string-eval - Traces evaluation of literals involving string relation

string-function-eval - Traces evaluation of functions involving strings

string-all - Traces all string specialist operations, including *string-eval* and *string-function-eval* .

3.2.12 Belief Specialist

belief-enter - Traces entry of literals into the belief specialist

belief-evaluate - Traces evaluation of literals by the belief specialist

belief-details - Traces details of belief specialist operation.

belief-all - Equivalent to *belief-enter, belief-evaluate, belief-details*

3.2.13 Meta Specialist

meta-eval - Traces evaluation of literals involving meta level objects

meta-function-eval - Traces evaluation of functions involving meta level objects

meta-entry - Traces entry of information into meta-specialist

meta-all - Traces all meta operations, including *meta-eval, meta-function-eval,* and *meta-entry* .

3.2.14 Other Specialist

other-eval - Traces evaluation of literals involving external routines

other-entry - Traces entry of information using external routines

other-all - Traces everything to do with the "other" specialist, including *other-eval* and *other-entry* .

3.3 Response Generation

filtration -shows which formulas were filtered out and why.

filtration-details -shows the estimation of the likelihood of a clause being known.

organization - traces organization of clauses - currently little organization is done so nothing is traced.

response - Displays the wffs being input to the response generator (after filtration).

verbalization - shows the set of wffs used for each sentence, and the sentence fragments resulting from that set.

fragment - shows the creation of fragments for clauses and literals.

combine - shows how the fragments are combined.

combine-details - shows more detailed information on all combining attempts.

retrieval - shows any wffs taken from the knowledge base to be used in filling in a fragment.

trans-details - shows the details of translating into fragments (for debugging)

response-all - traces everything to do with response generation (all the above trace values).

response-int - traces interesting stuff during response generation, including *filtration, organization, verbalization, response, fragment* , and *retrieval* .

response-int - traces the minimum amount of stuff needed to see how the response generation works, including *filtration, verbalization, response,* and *retrieval* .

Chapter 4

EPILOG Tweakable Parameters

Tweakable parameters may have their values changed using the command **tweak** . Current parameters which may be tweaked:

print
: When set to *nil* , no regular printing is done - regardless of trace values or response parameters. The default is *t* - printing is done.

warn
: When set to *t* , warnings are printed, otherwise they are supressed. The default is t - warnings are printed.

4.1 Main System Operations

4.1.1 Normalization

stop-if-error
: indicates whether the system should pause and wait for the user to press return when an error is detected in the syntax of an input formula. The default for this is t - stop and wait. If lisp is not being run interactively, the system will determine this and set the flag off when it starts up.

quoted-indicators
: a list of symbols which indicate that what follows is a quoted list for EPILOG. This is initially set to be just the lisp quote symbol *(')* .

name-symbol
: the symbol which indicates that the next item following is to be used as a name for the preceeding structure. Initially this is set to *!* .

expand-names
: indicates whether or not named symbols (using the symbol above) should be expanded when printed. This is initially set to t. Note, if you change this, it will not affect the printing of formulas already input, only new ones.

split-episodic
: indicates whether or not top level episodic formulas with embedded conjuntions should be automatically split. The default is t - split them. If this is turned off, a meaning postulate should be entered to make the split inference.

*(A x_wff (A y_wff (A z_ep ('((x and y) * z) true) ('((x * z) and (y * z)) true))))*

check-pred-parts

when a new predicate is input which can be split into several parts (between -), this flag indicates that both parts must exist as predicates on their own to make this a "compound" predicate, in which case it will inherit properties from the last part. Otherwise only the last part is checked (the default - nil).

memory-load-specs

indicates whether formulas re-loaded using the permanent memory feature should also be sent to the specialists to rebuild their domains. The default is *t* . If set to *nil* , loading is much faster, but the specialists will not be able to assist in evaluation.

4.1.2 Classification

default-indicators

A list of indicators to use for new predicates which do not specify indicators of their own. This is initially set to *(tp.unknown)* , but may be changed to something more meaningful to your own system if you will be entering a large number of predicates which belong to a particular category.

key-threshold

The threshold of interest beyond which all keys will be included in the list of trigger keys for a rule. These trigger keys are then used to classify the rule, as well as for matching against input formulas.

literal-complexity

The base complexity for each literal in the complexity calculation. This is initially set to 2, but may be changed using the *tweak* command.

variable-complexity

The complexity for a variable in the complexity calculation. This is initially set to 0, but may be changed using the *tweak* command.

constant-complexity

The complexity for a constant in the complexity calculation. This is initially set to 1, but may be changed using the *tweak* command.

function-complexity

The complexity for a functional term in the complexity calculation. This is initially set to 2, but may be changed using the *tweak* command.

subnet-topics

Topics that indicate separate subnet storage (*tp.mental-attitude*). Note: it may be dangerous to change this.

useful-nonepisodic-topics

a list of topics which are considered "important" enough to ensure that literals involving predicates which indicate these topics should be included in the trigger keys for rules. This is initially set to *(tp.causal-relationship tp.kinship tp.happening tp.location)* .

4.1.3 Assertion

input-array-expansion-size

indicates the increment by which the input array is expanded. This is initially set to 100. Changing it should not be necessary unless it is known that a huge number of story sentences are to be input.

consistency-effort
Indicates how hard to try to determine if an assertion is inconsistent with already known facts. Levels are 0 (no consistency testing), 1 (lookup only), 2 (verification with unit probabilities only), 3 (verification with anything), and 7 (full blown question attempt).

consistency-action
Indicates what to do if inconsistent assertions are made. Actions are 0 (enter anyway), 1 (print a warning and enter anyway), 2 (reject), 3 (print a warning and reject), 4 (try to combine).

4.1.3.1 Forward Inference

forward-full
Indicates whether or not to allow partial rule instantiation. If this flag is set to t , all variables must be matched in a rule before it is instantiated. The default is to allow partial rule instantiation.

forward-effort
Indicates how hard to try to verify rule antecedents during input-driven inferencing. Initially set to 0, if it is tweaked to something greater than 0, a simple backchaining effort will be attempted to verify the antecedent instead of just lookup or specialist evaluation.

interest-threshold
The minimum interest level (product of interest and probability) an inferred wff can have that will still enable more input-driven inferencing. This is initially set to 6, but may be changed using the *tweak* command.

rule-forward
Indicates whether or not input-driven inferencing should be attempted on conditionals. Initially set on, but may be turned off using *tweak* .

story-forward
Indicates whether or not input-driven inferencing should be attempted on story facts. Initially set on, but may be turned off using *tweak* .

4.1.4 Interestingness

maximum-interest
Indicates the maximum interest value any object may have. This is initially set to 1-0.5.

minimum-interest
Indicates the minimum interest value any object may have. This is initially set to 1.

deep-thought
a multiplier used on the interest during calculations to see whether to continue a particular inference line. Low numbers (< 1) indicate shallow thinking (i.e. prefer short inference paths), while high numbers indicate that longer inference paths are desirable. This is initially set to 1 (i.e. the interest alone makes the decision with no preference for inference path length either way).

back-up-interest
tells whether or not to back interestingness up causal paths. An interesting conclusion would then affect its causal precondition by increasing its interest. This is initially set off (nil), but may be set on for testing. There is still work to be done on determining exactly what constitutes a causal chain.

initial-charge
this is the amount to start the inherited interest component of new story sentences with. It may be necessary to give a new formula an extra boost to get it over the initial hump. However, our testing so far has not had this problem, so it is initially set at 0.

inherit-amount
this is the amount to decrease the inherited interest by at each step in an inference path. It is initially set to 5.

check-inherit
this flag indicates that forward inference should be allowed to continue based only on inherited interest values if the interest level of the particular formula is too low by itself. This is initially set to t (allow continuance).

wff-component
a multiplier used to indicate how much of a formula's interest should be used to update the interest value of its arguments. This is initially set to 0.1. Making the number higher makes the interest levels of new formulas involving existing entities increase more rapidly.

arg-component
a weighting used to indicate how much the interest of an argument affects the interest value of the entire formula. This is initially set to 1. Specific predicates may have weighting factors of their own for specific argument positions - this is an overal weighting to use after that.

operator-component
a weighting used to indicate how much the interest of an operator, predicate, or function affects the interest value of the entire formula. This is initially set to 1.

4.1.5 Unification and Comparison

unify-sorts
Indicates whether sorts should be considered when unifying terms. Initially set on, but may be turned off using *tweak* .

4.1.6 Question Answering

General Question Parameters:

qa-iterations
The maximum number of actions to be done from the agenda during a question answering attempt. If the attempt stops due to reaching this maximum, it can be restarted by just doing a *(q)* command with no formula. The default is 10, but it may be changed using the *tweak* command.

question-effort
The default effort level which will be used for all questions. The values are 0 -lookup/specialist evaluation only, 1 - allow subgoal splitting, 2 - allow inference, 3 - allow assumptions. The default is 3 - the maximum effort level.

question-threshold
The probability threshold above which answers are accepted. Any answer obtained with a probability lower than this is rejected (but is saved on the *other-answers* list). This threshold is initially 0.4, but may be reset using *tweak* .

minimum-effort
The minimum number of iterations to use in finding answers to a question. If an answer with probability less than 1 is found, and this many iterations have not been used yet, the system will try again to find another answer. The default is 5.

max-wh-difference
The maximum difference allowed between the depth of the first answer obtained for a wh-question, and any subsequent answer. If set to nil, all possibilities are tried to exhaustion. The default is 0, which gives the most efficient result, although may not give all the desirable answers in all cases.

Parameters which affect position in the question-answering agenda:

qa-access-weight
This is the penalty used for access actions when ranking them to put them on the agenda. It is initially 40, which ensures that subgoals get preference over access actions. This may be changed using *tweak* as well.

contra-weight
This is the penalty for contrapositive subgoals, and for accesses which will lead to contrapositive subgoals. It is initially 10. This may be changed using *tweak* as well.

rank-importance
This is the weight of the rank when calculating agenda position for actions. It determines how much higher ranked actions are preferred over lower ranked actions. It is initially set to 100, which makes it the most important features for ranking agenda items, but may be reset with *tweak* .

qa-depth
This is the "maximum" depth that will be considered in the agenda positioning calculation. When a subgoal's depth gets to half this amount, the amount it contributes to the agenda position gradually tapers off until at the maximum, there is no contribution. This makes the subgoal or access action quite undesirable, unless it is the only one available. This is initially set to 20, which means that the rank importance starts tapering off at 10.

prob-importance
This is the weight of probability when calculating agenda position for actions. It determines how much actions with higher probability are preferred over those with lower probability. It is initially set to 100, which makes it the second most important feature forranking agenda items (probability numbers are less than 1, rank numbers are usually greater - this is why probability becomes second most important feature even though the "importance" numbers are the same), but may be reset with *tweak* .

initial-prob
This is the probability to use in calculating the agenda position of initial (depth 0) subgoals. It is set to .75 initially, and should be less than 1 to prevent the system from preferring the initial accesses to later ones which have actually retrieved information from the knowledge base. Note: this does NOT affect the probability of the answer, only the agenda positioning of initial subgoals.

interest-importance
This is the weight of the subgoal's interest to complexity ratio when calculating agenda position for accesses or subgoals. It determines how much actions with higher interest measures are preferred to those with lower measures. It is initially set to 10.

difficulty-importance
This is the weight of subgoal difficulty when calculating agenda position for accesses or subgoals. It determines how much actions with lower difficulty measures are preferred over those with higher measures. It is initially set to 50.

quantified-difficulty
This is the difficulty value added to subgoals which are quantified but are not conditionals (e.g. existentially quantified). It is set to 20, which penalizes existentially quantified subgoals moreso than simpler subgoals, but less than the more difficult subgoals requiring assumption.

conditional-difficulty
This is the difficulty value added to conditional subgoals which are solved by assuming the antecedent and trying to prove the consequent. It is initially set to 30 (quite high).

split-difficulty
This is the difficulty value added to subgoals which are split into several simpler subgoals, without any assumptions. It is initially set to 10 (low).

assume-difficulty
This is the difficulty value added to disjunctive subgoals which are solved by assuming the negation of one of the disjuncts and trying to prove the rest. It is initially set to 30 (quite high).

mp-weight
This indicates how much to penalize access actions which are looking for mp's to infer with. It is initially set to 1-0.50, a very large penalty, as the mp's are rarely helpful in question answering (but can be, so they must be available).

residue-penalty
This indicates how much to penalize subgoal actions which have a residue involved in the comparison. The residues are rarely helpful and so the penalty is set quite large (1-0.50).

use-inherit
This flag indicates whether or not to consider "inherited" interest in positioning agenda items. It is set to *t* (use) by default.

qa-inherit-amount
This flag controls how much the inherited interest descreases from parent subgoal to child subgoal. It is initially set at 20.

favor-interest
This is the amount to "prime" an agenda item with when it is moved to the top of the agenda after too much time has been spent on the other proof/disproof attempt. It is initially set to 1-0.5. Note that ***use-inherit*** must be t for this to be considered, as the interest is added as an inherited interest value.

favor-position
This is the amount above the highest agenda item to place an agenda item when it is moved to the top of the agenda after too much time has been spent on the other proof/disproof attempt. It is initially set to 100.

Parameters which affect ACCESS actions:

max-wffs
The maximum number of retrieved formulas to test in an access action. Any formulas not tested

are put back on the agenda, so nothing is lost. This is used to control how long actions can take. It is initially set to 10, but may be changed using *tweak* . Changing it to a higher value may make some accesses take longer because more wffs are tested, but the matching wff may be found sooner. There is a trade-off - if this access action does not contain the wff we need, having a low *max-wffs* allows us to get on to the next action more quickly.

max-class
This contains the maximum number of classifications to retrieve wffs for in an access action. Additional classifications are put back on the agenda for a later time. This is initally set to 5, and may be reset using *tweak* . If this parameter is set to 1, and *max-wffs* set to a very large number, the access actions are more similar to **ECoNet** 's.

Parameters which affect SUBGOAL actions:

goal-forward
This flag indicates whether or not input-driven inferencing should be attempted on assumptions made during the proof process (it will only have an effect if input-driven inferencing in general is turned on - using

story-forward and **rule-forward**).
It is initially turned off, but may be turned on using the **tweak** command.

4.1.6.1 Saving Query Results

save-results
indicates whether or not to save question answers obtained, given that some inference was required to get them. This is intially set to nil - do not save.

result-difficulty
indicates how hard the system had to work to get an answer before it will be saved (which only happens if **save-results** is t). This is initially set to 0, which means that any inference used to answer the question - whether from a specialist or application of rules - will result in the answer being saved. Any number higher than 0 indicates how many different rules had to apply before the answer is considered for saving (note that this may be different from the number of rule applications - the same rule may be applied more than once but it is considered to be 1 for this test).

result-forward
indicates whether or not query results which are saved should have input-driven inference done on them. This is initially set to nil - no inference attempted.

4.2 Specialists

4.2.1 Specialist Interface

specialist-entry-effort
The effort level at which internal consistency for a specialist should be checked during entry. This is initially set to the maximum effort level (***maximum-effort***), but may be set to 0 for input which is known to be consistent.

specialist-eval-effort
The effort level specialists use to try to evaluate or compare literals. This is initially set to the maximum effort level (***maximum-effort***), but may be reset using the **tweak** function. Note that an effort level of 0 indicates constant time only operations are to be done, while anything higher indicates that more should be done if necessary (other levels are handled individually by the specialists).

spec-enter
If this flag is set to t, then propositions are checked to see if they are in a specialist domain upon assertion, and given to the specialist to store in its own representation. If nil, no separate storage of information is done. The system initially starts up with this set to t.

spec-evaluate
If this flag is set to t, then propositions are checked to see if they are in a specialist domain and given to the specialist to evaluate. If nil, specialists cannot assist during evaluation. The system initially starts up with this set to t.

spec-eval-fn
indicates whether or not specialists should be used in evaluating functions. This is initially set to t (let the specialists try).

spec-assert
If this flag is set to t, then specialists are allowed to make assertions back to EPILOG. The default is t.

spec-compare-preds
If this flag is set to t, then when predicate comparisons are done, the predicates are checked to see if they belong to the same specialist domain, and sent to the specialist to compare. If nil, specialists cannot assist during predicate comparison. The system initially starts up with this set to t. NOTE: For proper operation of the system (especially with types), this should never be tweaked to nil, except for very short periods of time for testing.

spec-compare-lits
If this flag is set to t, then when literal comparisons are done, the literals are checked to see if they belong to the same specialist domain, and sent to the specialist to compare. If nil, specialists cannot assist during literal comparison. The system initially starts up with this set to t.

fwd-spec-compare
If this flag is set to t, then when literal comparisons during input driven inference are done, the literals are checked to see if they belong to the same specialist domain, and sent to the specialist to compare. If nil, specialists cannot assist during literal comparison. The system initially starts up with this set to nil. This parameter was added to allow more control over when the specialists are applied.

4.2.2 Equality Specialist

unique-names-assumption
indicates whether individual user assigned names (i.e. non-skolem constants) should be considered unique - and not equal. This is initially set to *t* , which says that such names should be considered to be not equal. If set to *nil* , the question of whether the names are equal would be answered *unknown* (in the absense of other information).

4.2.3 Set Specialist

set-assert
Indicates whether or not the set specialist should make assertions back to EPILOG. This is initially set to t.

4.2.4 Part Specialist

part-assert
Indicates whether or not the part specialist should make assertions back to EPILOG. This is initially set to t.

familiar-parts
This is a list of "familiar" parts that would probably be included on a part hierarchy if they existed for a particular type of entity. This includes *head, tail, arm, leg, body, mouth* , etc.

4.2.5 Color Specialist

color-margin
The maximum difference allowed between two range boundaries for them to be considered equal.

color-hedged-operators
This is a list of the operators on predicates that make them into "hedged" predicates. It is initially set to *(sort-of almost)* .

color-intensifying-operators
This is a list of the operators on predicates that intensify them. Currently intensifiers do not have any effect on the color, but they really should "narrow" the boundaries of a particular color, just as the hedging operators widen them. The intensifying operators are initially set to *(very extremely mos)* .

4.2.6 String Specialist

string-separator
This is either nil, or a single character to insert between every pair of strings in the function *string-concat* . Initially it is set to nil.

string-field-divider
This is a list of characters which may be used to separate fields for *string-field* . Initially it is set so that only the character "-" can separate fields (i.e. '(#–)).

4.2.7 Belief Specialist

simulation-effort-level determines how much effort should be expended in trying to answer simulative queries. The value of this parameter is passed to the **question** function as its effort level. This is initially set to 3.

max-simulation-depth determines the maximum allowed depth of simulation nesting. Without this parameter, there would be the possibility of the system running a simulation, and the simulation

running a simulation, and that simulation running another simulation, *ad infinitum.* This is initially set to 3.

4.2.8 Other Specialist

interpret-nil-as-no
Some of the externally defined evaluation routines may return nil meaning *unknown* , while others may intend *no* . The system can handle only one assumption here - if the flag is *t* the assumption is *no* , otherwise *unknown* (the default). To prevent any ambiguity, the routines should return *yes, no* , or *unknown* .

4.2.9 Response Generator

say-answer
Indicates whether to say the answer obtained to a question in English, using the wffs that answered the question. This is initially set to t .

say-knowledge
Indicates whether to say rules input to the system in English. This is initially set to t .

say-mp
Indicates whether to say meaning postulates input to the system in English. This is initially set to nil .

say-story
Indicates whether to say story information input to the system in English. This is initially set to t .

say-meta
Indicates whether to say meta information input to the system in English. This is initially set to nil .

say-infer
Indicates whether to say facts inferred by the system (input-driven inference) in English. This is initially set to t .

say-question
Indicates whether to say the question asked in English. Currently the question is "said" as a statement - a future enhancement is to ask it as a question. This is initially set to nil .

say-immediate
Indicates whether to say lowest level input clauses as they are entered. (The usual method is to gather up the low level wffs and say them as one sentence) This is initially set to nil .

default-lex
indicates whether to let the system figure out what a lexical form should be (t) or prompt the user to ask for it (nil). This is initially set to *t* .

default-trans
indicates whether to let the system figure out what a translation should be (t) or prompt the user to ask for it (nil). This is initially set to *t* . If the system uses a default, the user will be warned.

filter-threshold

the threshold beyond which all wffs are filtered out of a response. This is initially set to 50, but may be modified to any number between 0 (remove everything except the wff with the lowest likelihood of being known) and 100 (remove nothing).

fragment-routines

a list of lisp routines which may be included in translation information, and then called to execute some particular function. Currently only one is available, so the list is *(make-adverb)* .

***max-response-complex*ity**

This may be either nil, which indicates that all formulas should be attempted by the response generator, regardless of complexity, or a number which is the maximum complexity to attempt to say. The default is 40.

prompt-if-too-complex

If this flag is t, when a formula is detected which is too complex to say (using ***max-response-complexity*** , the user will be prompted to see if he wants the system to attempt it anyway. If nil (the default), the formula will just be ignored.

response-warn

This flag indicates whether or not to print response generation warnings. It is initially set to t, so that warnings about possible strange sounding output will be printed.

active-topics

This parameter is used by the response generator to help determine how to say certain predicates when no translation information is available for them. For a predicate with only one argument, it is considered a *noun* is the predicate is a type predicate, a *verb* if one of the indicators for the predicate is on the ***active-topics*** list (or is beneath in the topic hierarchy), and an *adjective* otherwise.

Chapter 5

EPILOG Display Options

Display options may be used with the command **display** to look at certain aspects of the system - the main system and specialist representations. Global display options:

-full, -f - this indicates that full information is desired. Not all display options take this into account however.

-brief, -b - this indicates that a smaller amount of information is desired. Not all display options take this into account however. This is the default if neither *full* or *brief* is included.

-key, -k - this indicates that the other arguments are to be used as input for an apropos to find legal values that look like them. This may be used with the *tweak* , *trace* , and *display* options. If no option is specified, *display* is assumed. The *key* feature is NOT passed to any other display routines.

The current display options are described in the rest of this chapter.

5.1 Main System

checkpoint - This displays the checkpoints available to retract to.

display, options, help - Any of these will display the display options available.

tweakable, tweak - Either of these will display the tweakable parameters.

traceable, trace - Either of these will display the traceable values.

5.1.1 Assertion

node - This displays propositions about a concept. If *-full* is used, subnets for this concept are also printed. If the concept has a sort associated with it, this is also printed. Note that this is also the default if no display option is given.

pred - This displays information about a predicate.

formula, wff - This displays the list formula corresponding to a wff-name.

prob - This displays the lower subjective probability for a wff-name.

wffs - This displays all formulas in the system, based on the main classification.

wffs-by-topic - This displays all formulas in the system, based on the topical classification.

infer - This shows the inference path used to achieve a particular wff-name.

input-array, input - This shows the contents of the input array - i.e. all story sentences entered into the system.

5.1.2 Question Answering

question, q - prints the current question

solutions, solution, answers, answer - repeats the answers to the last question

agenda - prints the question answering agenda

subgoal - prints a subgoal from a question answering attempt

proof - prints the proof subgoal from a question answering attempt

disproof - prints the disproof subgoal from a question answering attempt

5.2 Specialists

spec-effort - Displays the specialist effort values

avail-specs, specs - Displays specialist which may be activated

active-specs - Displays currently active specialists

5.2.1 Type Specialist

hier - Prints hierarchies

5.2.2 Predicate Hierarchy Specialist

hier - Prints hierarchies

5.2.3 Time Specialist

episode-info - Displays information about episodes - all or one named episode

event-info - same as *episode-info*

time-info - Displays information about time points - all or one named point

meta-info - Displays information about links between chains - all or one named chain

chain-info - Displays information about links within chains - all or one named chain

5.2.4 Number Specialist

number-info - Displays the number graph - for all or a single named point

5.2.5 Color Specialist

color-info - Displays color information - purity, dilution and hue, for all known colors or a single named color.

5.2.6 Equality Specialist

equality-info - Displays equivalent and non-equal information for a given item, or displays all known equivalence sets if no item is specified. This may be abbreviated to **equal-info** .

5.2.7 Set Specialist

set-info - Displays information about sets - all or a single named set.

5.3 Response Generator

trans - Displays translation information for predicates given.

lex - Displays lexical information words given.

filter - Displays the current filter threshold.