ASKBUDJR: A Primitive Expert System for the Evaluation of the Fire Hazard of A Room

Richard L. Smith

U.S. DEPARTMENT OF COMMERCE
National Bureau of Standards
National Engineering Laboratory
Center for Fire Research
Gaithersburg, MD 20899

March 1986

U.S. DEPARTMENT OF COMMERCE
REAU OF STANDARDS

NBSIR 86-3319

ASKBUDJR: A PRIMITIVE EXPERT SYSTEM FOR THE EVALUATION OF THE FIRE HAZARD OF A ROOM

Richard L. Smith

U.S. DEPARTMENT OF COMMERCE
National Bureau of Standards
National Engineering Laboratory
Center for Fire Research
Gaithersburg, MD 20899

March 1986

U.S. DEPARTMENT OF COMMERCE, Malcolm Baldrige, *Secretary*
NATIONAL BUREAU OF STANDARDS, Ernest Ambler, *Director*

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ASKBUDJR: A PRIMITIVE EXPERT SYSTEM FOR THE EVALUATION OF THE FIRE HAZARD OF A ROOM

Richard L. Smith

Abstract

The Center for Fire Research (CFR) has a long-term project to develop expert systems as a technology transfer mechanism. CFR has as the long-term goal of this project: to develop a computer program which will make an expert estimate of the fire safety of a building based on CFR's deterministic physical models, technical data, and the expert judgment of its staff. The first major program to be developed by this project is based on the expertise of Harold E. (Bud) Nelson. Thus, this program will be called ASKBUD. In this report, the first exploratory steps taken to develop an expert system for fire hazard evaluation are described. Also, the progress made to date, as well as some of the major problems that must be solved, will be discussed. Since the ASKBUD expert system discussed in this report is in its infancy, we call it ASKBUDJr.

1. INTRODUCTION

The Center for Fire Research (CFR) has a long-term project to develop expert systems as a technology transfer mechanism. CFR has as the long-term goal of this project: to develop a computer program which will make an expert estimate of the fire safety of a building based on CFR's deterministic physical models, technical data, and the expert judgment of its staff. The program will have the capability to explain its conclusions it makes. It is important that this program, or one based on it, should be readily usable by CFR's clients. This is a progress report on this project.

An expert system is a computer program that solves real-world problems whose solution would normally require a human expert [1,2]. A number of such computer programs have been built. It is recommended that an expert system first be developed using the expertise of only one expert. This facilitates the evaluation of the computer program and the collection of knowledge. Later, the expertise of other experts can be added to the program. One major benefit of developing an expert system is that one is forced to record the human expert reasoning process explicitly. This facilitates the improvement of the human expert's decision making process. Another major advantage of an expert system is it effectiveness as a tool for the transfer of technology.

Using expert systems to transfer technology will greatly improve the speed, efficiency, and accuracy of transferring the valuable technology developed by CFR to its clients. It could lead to a revolution in fire safety engineering. It is envisioned that eventually every fire safety engineer in this country will have at his disposal an expert consultant in fire safety; one that is always state-of-the-art, always working at top quality, and recognized worldwide as an expert in its field. This consultant will be a computer program that will run on a modestly priced computer, currently comparable to an IBM AT[1]. Its performance will equal or surpass the performance of most human experts. It will tirelessly explain its results to the user so it will also act as a teacher. It will not become tired or bored. It will always operate at its best, never having an off day.

---

[1]Certain commercial equipment, instruments, or materials are identified in this report in order to describe adequately the program. Such identification does not imply recommendation or endorsement by the National Bureau of Standards, nor does it imply that the materials or equipment identified are necessarily the best available for the purpose.

This will not come about easily or cheaply. But, we do not need to wait for the completion of the final expert system before we have useful programs. Like a person entering a new field or discipline, a program on its way to becoming an expert program will start as a novice. Thus it becomes competent in its craft and, if it continues its development, it becomes an expert. Just as an apprentice can help the journeyman, programs developed by this project will be of significant use to the fire safety engineer before we have reached our final goal of having an expert system comparable to the best human experts.

Thus we see a series of programs growing out of this project that will be of value to fire safety engineers. Each of these programs should improve the decision-making quality of the human expert that it aids.

The first major program to be developed by this project is based on the fire safety expertise of Harold E. (Bud) Nelson. Thus, this program will be called ASKBUD. In this report, the first exploratory steps taken to develop an expert system for fire hazard evaluation are described. Also, the progress made to date, as well as some of the major problems that must be solved, will be discussed. Since the ASKBUD expert system discussed in this report is in its infancy, we call it ASKBUDJr. It was written in Golden Common Lisp (version 1.0) on a Heath personal computer (model 161) with two floppy disks and 640 K RAM. This hardware and software restricted the size and complexity of ASKBUDJr.

2. ASKBUDJR

No attempt will be make to defend the technical approach that the expert (H. Nelson) used because by definition the expert is correct for purposes of evaluating an expert system based on one expert. Furthermore, the underlying fire technology of ASKBUDJr is not the subject of this report. The organization, control, and structure of the computer program ASKBUDJr are the main subjects.

Furthermore, we were severely limited in the techniques that ASKBUDJr could use because of the limitations imposed by the software and hardware. Thus, our human expert had to choose or invent techniques that were consistent with these limitations. It should not be inferred that the techniques used in ASKBUDJr are recommended or that the human expert would use them if he was not constrained as he was by the limitation of the hardware and software. Therefore, the technical approach will be described but not defended or recommended. However, we are willing to explain and defend the transformation of this approach into a computer program.

The problem that ASKBUDJr will address involves a single room, e.g., a bedroom, a motel room, or a hospital room. It is assumed that a fire starts in the room and that we have only one occupant. The structure of the building does not become involved in the fire. The room has one window and one door. Also, the room will have the following furnishings: a bed, a chest, a chair, a table, a wastebasket, and a set of curtains or drapes.

The degree of hazard the occupant is exposed to depends upon how fast the hazard from the fire builds up versus how fast he can evacuate the room. Of all the many things that could influence these times, we are limiting ourself to the considerations of only a few. However, there will be enough meat on these bare bones to identify some interesting problems.

The user of ASKBUDJr will be asked to enter the following for each item of the room's furnishings:

- the peak burning rate in kW
- the effective heat of combustion, DHc, in kJ/g
- LC50, the amount of burned material per unit volume that will kill 50% of a sample of rats in 30 minutes using a standard test procedure in mg/L or g/m^3.
- the growth rate of the burning rate as one of four t-squared curves, i.e., slow, moderate, fast, or very fast (or zero if it doesn't burn)
- what other items will be ignited if the one in question burns

For our t-squared burning curves:

a **slow** growing curve leads to 1 MW in **600** sec
a **moderate** growing curve leads to 1 MW in **300** sec
a **fast** growing curve leads to 1 MW in **150** sec
a **very fast** growing curve leads to 1 MW in **75** sec.

The user will also enter the sizes of the room (length, width, and height), the size of the door and window openings, and whether there is a fire detector/alarm or not.

Finally the user inputs whether the occupant is: awake, asleep, drunk, mobile, or nonmobile; the time, in seconds, the occupant would take to move out of the room after becoming aware of the fire; and whether the occupant has a heart or lung condition. Also the user will input whether there is a person outside the room who can aid the occupant and how long, in seconds, it would take this person to come in and remove the occupant after becoming aware of the fire.

The principal output of ASKBUDJr is a statement of the level of risk the occupant is exposed to and the margin-of-safety-time, the length of time from the start of the fire until hazardous conditions exist less the length of time for the occupant to get out of the room. The time interval to hazardous conditions is either the length of time to flashover or the length of time to reach a dangerous level of toxicity, both measured from the time of ignition.

To the user, ASKBUDJr consists of the introductory message (Figure 1) and a request for user input (Figure 2) and the program's final output (Figure 3). However, there is more to ASKBUDJr. It consists of the programming modules and their principal procedures as shown in Table 1.

We will now discuss each of these modules in turn. The module "askbudjr" contains the introductory message, the procedure that loads all the required files, and the procedure that runs all the procedures in the right order.

The "toolkit" contains needed procedures that are not provided in Golden Common Lisp, e.g., a procedure that computes the square root of a number.

To first order, the match procedure [3] can be said to compare two symbolic expressions to see if they are identical. However, "match" is more general than this simple statement would indicate. For example, "match" would say for the following two expressions that the first matches the second.

1. The ? is greater than the ? +.

2. The time-to-flashover is greater than the time-to-toxicity except when the door is closed.

The match procedure is used in the forward-chain procedure [3] as is the expression "rules". The rulejr module contains all the rules for the system (see Table 2 and appendix A3)[2]. The rules have the general form:

---

[2]Instead of the approach we chose, we could, in principle, have used the following approach. For each possible combination of the parameters for the room and occupant we could have a rule that states whether the associated level of risk is acceptable or not. For this case, let us estimate the number of rules we would have to write.

	Number of Possible Values	Cummulative Number of Rules
awake or asleep	2	2
drunk or sober	2	4
mobile or nonmobile	2	8
heart or lung condition	2	16
initial slope of burning	5	80
detector in room	2	160
availability of external aid	2	320
door open or closed	2	1280
other parameters	$> 2^{39}$	$> 10^{13}$

Thus one sees why we don't build a huge truth table one square at a time. Organizing the knowledge so we avoid a combinatorial explosion and have a system that works is a significant portion of the knowledge representation problem.

If the sky is blue, then the sun is shining.

or, in general,

If A, B, and C, then D.

The forward-chain procedure takes the initial assertions (statements of facts that define the problem) and uses "match" to see if any of the rules apply. If they do, it adds the rule's conclusion, the "then" part of the expression, to the list of assertions. "Forward-chain" continues until there are no more new conclusions to be made. It then quits.

Knowledge in this system is contained in assertions, rules, frames, and the methods of calculating various parameters such as time to hazard. A frame is a data structure which allows values to be assigned to properties of the room, occupant, or furnishings. It allows various types of values such as explicit values, default values, if-needed values, and inheritances. The frame module contains the procedures for the creation and manipulation of frames. If one requests a value from a frame, first the procedure in the frame's module looks to see if there is an explicit value for the parameter of interest. If none, it looks to see if there is a default value. If none, it looks to see if there is an if-needed or demon procedure that will compute the needed value. There is also the option that a value can be inherited from a parent frame. However, this feature is not used in ASKBUDJr.

The inputjr1 module contains the default and if-needed values for all the parameters used in ASKBUDJr.

In the frame for each item of furnishings, there is a will-ignite property. This is a list of all furnishings that a given item will ignite. From this will-ignite information, the fuel-packages procedure will compute all the fuel packages in the room. For example, if the wastebasket will ignite the bed and table and the table will ignite the curtains-drapes and we assume the other items will not ignite anything, then the fuel packages are: (1) chair, (2) chest, (3) curtains-drapes, (4) bed, (5) table, curtains-drapes, (6) wastebasket, bed, table, curtains-drapes.

The time-to-flashover procedure takes the list of fuel packages and sees which one will lead to flashover in the shortest time. The peak burning rate of a fuel package is taken to be the sum of its members' peak-burning-rates. ASKBUDJr uses the Thomas equation to predict the power level required for flashover. The Thomas equation estimates the minimum power in kilowatts that will flashover a room. It is written as

$$(dQ/dt)fo = 278\ Av\ (hv)^{1/2} + 7.8\ Aw$$

(dQ/dt)fo is the estimated minimum flashover power in kW,
Av is the vent area in square meters (e.g., area of open windows and doors),
hv is the vent height in meters (the actual vertical length of the vent), and
Aw is the total surface area of the room in square meters.

If the sum of the peak burning rate of the items in any fuel package exceeds the minimum flashover power, we assume flashover occurs.

The time-to-flashover procedure determines the shortest time to flashover due to any fuel package. To do this, we assume that an entire fuel package burns as a single unit with its growth rate determined by the fastest burning member of the package that has an individual peak burning rate greater than 200 kW. Thus the equation for the energy emitted by the burning of a fuel package, q, is

$$dq/dt = Kt^2$$

for $0 < t < [(\text{sum of } (dq/dt)\text{max of items in fuel package})/(\text{growth rate of fastest growing element with } (dq/dt)\text{max} > 200 \text{ kW})]^{1/2}$

where K is the growth rate of the fastest growing element which has a peak burning rate greater than 200 kW.

Besides the hazard due to flashover, we also consider toxic gases hazard. To compute the time to a hazardous condition due to toxic gases we proceed as follows in the "toxic" module.

For a one item fuel package for which m is the weight loss of the burning object and q is the energy released, we have

$$dq/dt = (dm/dt)\ DHc$$

where DHc is the effective heat of combustion. Thus it follows that

$$dm/dt = (dq/dt)/DHc.$$

A hazardous condition is reached whenever

$$m/Vc = r\ LC50$$

where Vc = one half the volume of the room; r is a constant that is taken to be 3, except for an occupant who is drunk or has a heart or lung condition, then "r" is taken to be 1; LC50 is the value for the material burning in units of mg/l or g/m^3. The selection of the value for Vc is consistent with the worst case approach. It is assumed that the toxic gases are all in the upper half of the room.

We wish to determine when we will reach this toxic condition. This will happen when

$$m(t) = r\ LC50\ Vc.$$

Since $dq/dt = Kt^2$, we have

$$dm/dt = Kt^2/DHc$$

or

$$m(t) = (Kt^3)/3\ DHc.$$

Assuming we do not run out of material, we get for the time to toxic hazard, tx,

$$tx = (3\ DHc\ r\ LC50\ Vc/K)^{1/3}.$$

For two or more items in a fuel package we compute a weighted average of the LC50 and DHc for all the items in the package. The relative weight, ri, assigned to each item is:

Item	Relative Weight
bed	1.0
chest	1.0
table	0.5
chair	0.3
wastebasket	0.05
curtains-drapes	0.05

The average of DHc, (DHc)a, is computed using the following expression

$$(DHc)a = (r1\ (DHc)1 + r2\ (DHc)2 + \ldots)/(r1 + r2 + \ldots)$$

where ri is the relative weight of the item with heat of combustion (DHc)i.

The average of the effective LC50, (LC50)a, is computed using the following expression

$$(1/(LC50)a) = (r1/(r1 + r2 + \ldots))/(LC50)1 + (r2/(r1 + r2 + \ldots))/(LC50)2 + \ldots$$

where (LC50)i is the LC50 of the i-th item.

To take into account the impact on the time to toxic hazard due to having the window or door open, we make the following six approximations:

1. If the area of the window opening is larger than 16 sq. ft., then the time to toxic hazard is infinite (10^5 seconds), i.e., a toxic hazard is never reached.

2. If the room is not in a building or is in a big building and the total open area of the door and window is larger than 16 sq. ft., then the time to toxic hazard is infinite.

3. If the door is closed and the area of the window opening is less than 16 sq. ft., then the time to toxic hazard, th, is

$$th = tx/(1 - Wa/16)$$

where Wa = the area of the window in square feet.

4. If the room is in a large building or no building and the open area of the door and the window is less than 16 sq. ft., then the time to toxicity is

$$th = tx/(1 - (Wa + Da)/16)$$

where Da = the area of the door opening in square feet.

5. If the room is in a small building, the open area of the door is greater than 16 sq. ft., and the open area of the window is less than 16 sq. ft., then the time to toxicity is

$$th = 1.44225\ tx/(1 - Wa/16).$$

6. If the room is in a small building, the open area of the door is less than 16 sq. ft., and the open area of the window is less than 16 sq. ft., then the time to toxicity is

$$th = tx\ (1 + (Da/16)(.44225))/(1 - Wa/16).$$

If the occupant is capable of escape, the time for the occupant to escape the burning room is the time it takes him to become aware of the fire plus the time it takes for him to move out of the room. We assume the awake, sober occupant becomes aware of the fire when it reaches a power level of 25 kW. The sober, asleep occupant becomes aware of the fire when and if the detector sets off an alarm. In all other cases the occupant will have to rely on external aid to escape. It there is no person to aid the escape, then the rescue time is infinite (10^5 s). With no detector and the occupant asleep, we assume the rescuer takes 5 minutes to become aware of the fire and to get the occupant out. In all other cases, the rescuer learns of the fire when the occupant does or when the alarm goes off. In this case, the rescue time is the time to awareness plus the time to come in and take the occupant out, which we assume to be 30 seconds unless the user enters a different duration.

The key parameter, margin-of-safety-time, is defined as the difference between the time to hazard and the lesser of the escape or rescue time.

The "inputjr2" module contains the procedure that clears out all the old explicit values in the frames (reset-all) so that the procedure request-values

can ask the program user for new values. The framass procedure takes the information in the frames as explicit, default, or if-needed values and makes assertions out of them.

The status procedure reports the values that will be used by the program for all the variables of the problem, such as room length, occupant awake, etc. A status report is given in Figure 2.

Finally, the whyjr procedure gives the "if" part of any rule that was used. An example of its use is shown in Figure 3.

3. SUMMARY AND THE NEXT STEP

In developing ASKBUDJr we had to take into account software and hardware limitations. For example, the version of Golden Common Lisp we used lacked most numerical functions such as square roots, exponents, transcendental functions, etc. The generic software tools were limited to procedures in the modules match, forchain, frames, and whyjr. These factors had a strong influence on the problem ASKBUDJr could consider and the approaches that could be followed to a solution. However, ASKBUDJr gave reasonable answers, considering its constraints, for a number of situations.

The knowledge of the expert is spread rather widely throughout ASKBUDJr. It is in the frames in the default values and if-needed values. It is in the rules. It is also in the procedures used to calculate the various quantities of interest. From the point of view of getting the conclusion only, it doesn't matter where the knowledge resides. However, the problem of providing

an explanation for the conclusion that traces back to the problem's initial point needs further investigation if we use a similar structure for the knowledge in future programs.

In the next step, we move to a more powerful computer, to more powerful software tools, and to a more demanding problem. With this enhanced capability and challenge, we must resolve the general structure of how to represent the expert's knowledge in ASKBUD and what type of explanation facility will be required.

At one extreme we could try to put all the knowledge into rules. Since there is some transformation from frames and procedures to rules [4], we could convert ASKBUDJr to a pure rule system. However, speed or economy of size may argue against this.

On the other hand, we can put our expert's knowledge into rules and procedures (or models) with the rules selecting which procedure to use and analyze the procedures output. Only future investigation will resolve this question.

Another question that ASKBUDJr brings up is that of logical consistency. It is possible to enter logically inconsistent information into ASKBUDJr and it will accept them. The question arises whether in future programs there should be a logical consistency check built into the program so the user cannot enter logically inconsistent data.

Finally, we need to resolve the question of the form for our output. Is a statement saying the risk is low, moderate, etc., of use to potential uses of such an expert system? If it is, do these terms need to be more clearly defined? These and other questions may be resolved as we develop future programs.

4. REFERENCES

[1] Weiss, S. and Kulikowski, C., A practical guide to designing expert systems, Rowman & Allanheld Publishers, New Jersey, 1984, 174 p.

[2] Goodall, A., The guide to expert systems, Learned Information, New Jersey, 1985, 220 p.

[3] Winston, H. and Horn, B., LISP second edition, Addison-Wesley Publishing Co., Massachusetts, 1984, 434 p.

[4] Rich, E., Artificial Intelligence, McGraw-Hill Book Co., New York, 1983, 436 p.

Table 1. Programming Modules of ASKBUDJr

Modules	Procedures
askbudjr	intro-message superload runprograms
toolkit	cube-root square-root
match	match (chapter 17, ref. 3)
rulesjr	rules
forchain	forward-chain (chapter 18, ref. 3)
frames	fget-v-d-p fput (chapter 22, ref. 3)
inputjrl	(default and if-needed values for frames)
fuelpack	fuel-packages
flashovr	time-to-flashover
toxic	time-to-toxic time-to-hazard margin-of-safety
inputjr2	framass reset-all request-values
status	status
whyjr	whyjr

Table 2. Rules Used in ASKBUDJr

Rule Number	Statement of Rules
1.	If the occupant is drunk, then the occupant is nonmobile.
2.	If the occupant is drunk, then the occupant reacts to smoke as if he had a heart or lung condition.
3.	If the time to toxic hazard is greater than the time to flashover, then the time to hazard is equal to the time to flashover, o$o seconds (o$o is the number of seconds to flashover).
4.	If the time to flashover is greater than the time to toxic hazard, then the time to hazard is equal to the time to toxic hazard, t$t seconds (t$t is the number of seconds to toxic hazard).
5.	If the ratio of the margin-of-safety-time to the escape-time is greater than one, then the occupant will have a safety margin of m$m seconds which is (m$m/e$e) times his physical escape time. He must recognize the need to evacuate and start his escape within m$m seconds from his moment of awareness to avoid eliminating his margin of safety (m$m is the margin of safety time in seconds and e$e is the escape time).
6.	If the margin of safety is negative, then in case of fire the risk to the person is very high.
7.	If the margin of safety is less than one minute and the ratio of the margin-of-safety-time to the escape-time is less than one, then the risk is high.
8.	If the margin of safety is less than one minute and the ratio of the margin-of-safety-time to the escape-time is greater than one and less than two, then the safety is marginal.
9.	If the margin of safety is less than one minute and the ratio of the margin-of-safety-time to the escape-time is greater than two, then the safety is moderate.
10.	If the margin of safety is greater than one minute and less than five minutes and the ratio of the margin-of-safety-time to the escape-time is greater than two, then the risk is low.
11.	If the margin of safety is greater than five minutes, then the risk is low.
12.	If the ratio of the margin-of-safety-time to the escape-time is less than one and the margin of safety is greater than one minute and less than five minutes, then the risk is marginal.
13.	If the ratio of the margin-of-safety-time to the escape-time is greater than one and less than two and the margin of safety is greater than one minute and less than five minutes, then the risk is moderate.
14.	If the room never becomes hazardous, then the risk is low.

Figure 1. Introductory Message

ASKBUDJr - version 1.1 - is a demonstration of a simple expert system for the evaluation of the fire hazard of a room. It was written by Richard L. Smith in Golden Common Lisp. It is based on the expertise of Harold E. Nelson.

CENTER FOR FIRE RESEARCH
National Bureau of Standards
September 24, 1985

ASKBUDJr deals with a fire in one room with one occupant. In the room is a bed, chest, table, chair, wastebasket, and curtains-drapes. You will be asked to supply information about the room furnishings, and occupant. If you don't have the information, or don't want to enter it, type "p" for pass.

Type any letter and "return" to continue.

Figure 2. Request for User Input

The present status of the room is:

furnishing	A peak	B h.-of-comb.	C LC50	D growth-rate	E will-ignite
1. Bed	1000	35	20	FAST	NIL
2. Chair	500	20	20	MODERATE	(TABLE CURTAINS-DRAPES)
3. Table	500	18	60	SLOW	(CHAIR)
4. Chest	1000	25	40	SLOW	NIL
5. Wastebasket	50	20	60	VERY-FAST	(BED)
6. Curtains	100	25	15	VERY-FAST	(CHAIR)

8. ROOM - dimensions in feet

A detector	B length	C width	D height	E door-height	F door-width	G fraction-open
N	15.0	12.0013	7.99869	7.00131	2.5	1

H window-width	I window-height	J in-big-bldg	K dim-m-f
2.5	2.00131	Y	F

7. Occupant

A awake	B mobile	C drunk	D heart-lung	E external-air	F time-to-es	G rescue-time
N	Y	N	N	N	8.23	1.0F+05

(DO YOU WANT TO INPUT DATA INSTEAD OF USING DEFAULT DATA? Y OR N)

Figure 3. ASKBUDJr Program's Output

```
((THE ROOM WILL FLASHOVER WITH THE FUEL PACKAGE (WASTEBASKET BED) BURNING IN 120.319
SECONDS))(THE ROOM WILL BE TOXIC WITH THE FUEL PACKAGE (CHAIR TABLE CURTAIN
S-DRAPES) BURNING IN 99.9019 SECONDS)

; Reading file B:/RULESJR.LSP

(RULE IDENTIFY4 SAYS THE TIME TO HAZARD IS EQUAL TO THE TIME TO TOXIC HAZARD 99.9019
SECONDS)
(RULE IDENTIFY7  SAYS THE RISK IS HIGH)
T
* (whyjr 'identify7)
((THE MARGIN OF SAFETY IS LESS THAN ONE MINUTE)(THE RATIO OF THE MARGIN-OF-SAFETY
TIME TO THE ESCAPE-TIME IS LESS THAN ONE))
*
```

APPENDIX A. OPERATING INSTRUCTIONS

To run ASKBUDJr, one must first load Golden Common Lisp. Then, with the ASKBUDJr disk in drive B, type

```
(load "askbudjr.lsp").
```

Then one types

```
(askbudjr)
```

and follow the instructions. If after a run, you would like to run the program again with some changes, type

```
(do-again)
```

and follow instructions.

APPENDIX B. LISP CODE FOR ASKBUDJR

```
 file : askbudjr 1.1

(defun askbudjr ()
  (superload)
  (runprograms))

(defun superload ()          ; must load file with f$f before calls
                             ; to are made
  (setq *print-level* nil)   ; this & nexts line are needed so long
                             ;  list are
  (setq *print-length* nil) ; printed without abbreviations
  (setq f$f '(bed chair chest table wastebasket curtains-drapes))
  (load "toolkit.lsp")
  (load "match.lsp")
  (load "forchain.lsp")
  (load "frames.lsp")
  (load "inputjr1.lsp")
  (load "fuelpack.lsp")
  (load "flashovr.lsp")
  (load "toxic.lsp")
  (load "inputjr2.lsp")
  (load "whyjr.lsp")
  (load "status.lsp"))


(defun runprograms ()
  (reset-all)        ; sets all the "values" in the frames to nil
  (intro-message)
  (input)            ; input the values for the parameters that
                     ; describes the room and occupant
  (do-again1))

(defun do-again ()
  (request-values2)
  (do-again1))

(defun do-again1 ()
  (setq assertions nil) ; resets the gobal variable that contains
                        ; the assertions that characterizies the
                        ;  room and occupant
  (thomas-flashover-package f$f) ; add an assertion about
                                 ; flashover
  (time-to-toxic-assertion f$f)  ; add an assertion about toxicity
  (framass)                 ; converts info in frames to assertions
  (margin-of-safety-ass f$f)    ; add an assertion about margin of
                                ; safety
  (margin-saf-evacuate-ratio-ass f$f)
  (flash-or-toxic-ass f$f)
  (load "rulesjr.lsp") ; we moved this from superload because it
                       ; would calculate expression before the
                       ; assertions had been made
  (forward-chain))  ; with rules & assertion we reason

(defun intro-message ()
```

```
  (terpri)
  (terpri)
  (terpri)
  (Princ '!ASKBUDJR -version 1.1 -is a demonstration of a simple
expert!)
  (terpri)
  (princ '!system for the evaluation of the fire hazard of a room.
It was !)
  (terpri)
  (princ '!written by Richard L. Smith in Golden Common Lisp. It
is based  !)
  (terpri)
  (princ '!on the expertise of Harold E. Nelson. !)
  (terpri)
  (terpri)
  (princ '!    CENTER FOR FIRE RESEARCH!)
  (terpri)
  (princ '!    National Bureau of Standards!)
  (terpri)
  (princ '!    September 24, 1985!)
  (terpri)
  (terpri)
  (terpri)
  (princ '! ASKBUDJR deals with a fire in one room with one
occupant. In the!)
  (terpri)
  (princ '! room is a bed, chest, table, chair, wastebasket, and!)
  (terpri)
  (princ '! curtains-drapes. You will be asked to supply
information about!)
  (terpri)
  (princ '! the room, furnishings, and occupant. If you don't have
the!)
  (terpri)
  (princ '! information, or don't want to enter it, type "p" for
pass.!)
  (terpri)
  (terpri)
  (terpri))
```

```
; file: toolkit

(defun our-cube-root (y)
  (cond ((< y 0) 'negative-number) ; we don't need the cube root
                                   ; of negative numbers
     (t (do ((x 1))
            ((or (< (abs (- y (* x x x)))
                    (cond ((> y 10000) (/ y 1000000.0)
                       (t .001)))
                 (= (- y (* x x x))
                    (cond ((> y 10000) (/ y 1000000.0))
                       (t .001)))) x)
        (setf x (* .5 (+ x (/ y (* x x)))))))))

(defun our-sqrt (y) ; works for large numbers
  (cond ((< y 0) 'negative-number)
     (t (do ((x 1))
            ((or (< (abs (- y (* x x)))
                    (cond ((> y 8000) (/ y 1000000.0)); for large no.
                       (t .001)))
                 (= (- y (* x x))
                    (cond ((> y 8000) (/ y 1000000.0))
                       (t .001)))) x)
        (setf x (* .5 (+ x (/ y x)))))))

(defun our-union (X Y)
  (cond ((null x) y)
     ((member (car x) y) (our-union (cdr x) y))
     (t (our-union (cdr x) (cons (car x) y)))))

(defun our-intersection (X Y)
  (cond ((null x) nil)
     ((member (car x) y)(cons (car x) (our-intersection (cdr x)
y)))
     (t (our-intersection (cdr x) y))))

(defun make-a-set (Y)
  (cond ((null y) nil)
     ((member (car y) (cdr y))
      (make-a-set (cdr y)))
     (t (cons (car y) (make-a-set (cdr y))))))

(defun squash (x) ; problem 4-9
  (cond ((null x) nil)
     ((atom x) (list x))
     (t (append (squash (car x)) (squash (cdr x))))))

(defun our-set-difference (in out) ; problem 4-13
  (cond ((null in) nil)
     ((member (car in) out) (our-set-difference (cdr in) out))
     (t (cons (car in) (our-set-difference (cdr in) out)))))

(defun samesetp (X Y) ; problem 4-15
  (cond ((and (null (our-set-difference x y))
```

```
            (null (our-set-difference y x)))
      t)))

;*****  "make-a-list-super"   takes a list of sets and remove any
; duplications

(defun make-a-list-super1 (Y l); if l=nil, removes dulication of
                               ; first element
  (cond ((null y) y)
     ((equal (length y) 1) (append y l)) ; if there is only one
                                         ; element, return it.
     ((samesetp (car y) (cadr y)); 1st & 2nd elements =, drop 1st
                                 ; element
      (make-a-list-super1 (cdr y) l)); and repeat
     (t (setq l (cons (cadr y) l)) ; if not =, save the second
        (setq y (cons (car y) (cddr y))) ; and look at 3th
        (make-a-list-super1 y l))))

(defun make-a-list-super2 (Y) ; have one agrument instead of 2
  (make-a-list-super1 Y nil))

(defun make-a-list-super3 (Y r) ; this is our non-duplicative list
  (cond ((null y) y)
     ((equal (length (make-a-list-super2 Y)) 1); if list is 1 long
                               ; or 2 long but same sets, return
      (append (make-a-list-super2 Y) r)); this shorten list with r
     (t (setq r (cons (car (make-a-list-super2 Y)) r)); operate on
                                         ; 1st and save it in r
        (setq y (cdr (make-a-list-super2 Y))); set y = to the cdr
        (make-a-list-super3 y r)))); repeat

(defun make-a-list-super (Y); one input variable
  (make-a-list-super3 Y nil))
```

```
;; file : rulesjr

(setq rules `((rule identify1
                (If (The occupant is drunk))
                (then (The occupant is nonmobile)))
              (rule identify2
                (If (The occupant is drunk))
                (then (The occupant reacts to smoke as if he had a
                        heart or lung condition)))
             (rule identify3
                (If (The time to toxic hazard is greater than the
                       time to flashover))
                (then (The time to hazard is equal to the time to
                        flashover ,o$o seconds)))
             (rule identify4
                (If (The time to flashover is greater than the time
                       to toxic hazard))
                (then (The time to hazard is equal to the time to
                        toxic hazard ,t$t seconds)))
             (rule identify5
                (If (The ratio of the margin-of-safety time to the
                      escape-time is greater than one))
                (Then (The occupant will have a safety margin of
                       ,m$m
                       seconds which is
                       ,(/ m$m
                        e$e)
                       times his physical escape time. He must
                       recognize the need to evacuate and start his
                       escape within
                       ,m$m
                       seconds from his moment of awareness to avoid
                       eliminating his margin of safety.)))
             (rule identify6
                (If (The margin of safety is negative))
                (then (In case of fire the risk to the person is
                        very high)))
             (rule identify7
                (If (The margin of safety is less than one minute)
                 (The ratio of the margin-of-safety time to the
                       escape-time is less than one))
                (Then (The risk is high)))
             (rule identify8
                (If (The margin of safety is less than one minute)
                 (The ratio of the margin-of-safety time to the
                       escape-time is greater than one and less than
                       two))
                (then (The safety is marginal)))
             (rule identify9
                (If (The margin of safety is less than one minute)
                 (The ratio of the margin-of-safety time to the
                       escape-time is greater than two))
                (then (The safety is moderate)))
             (rule identify10
```

```
   (If (The margin of safety is greater than one minute
         and less than five minutes)
    (The ratio of the margin-of-safety time to the
         escape-time is greater than two))
   (then (the risk is low)))
(rule identify11
   (If (The margin of safety is greater than five
          minutes))
   (then (the risk is low)))
(rule identify12
   (if (The ratio of the margin-of-safety time to the
         escape-time is less than one)
    (The margin of safety is greater than one minute
         and less than five minutes))
   (then (The risk is marginal)))
(rule identify13
   (if (The ratio of the margin-of-safety time to the
        escape-time is greater than one and less than
          two)
    (The margin of safety is greater than one minute
      and less than five minutes))
   (then (The risk is moderate)))
(rule identify14
   (if (The room never becomes hazardous))
   (then (The risk is low)))))
```

```
;inputjr1

(setf (get 'bed 'frame)
      '(bed (initial-slope-of-burn
                (value)
                (default fast)
                (if-needed))
         (peak-rate-of-burn
                (value)
                (default 1000)
                (if-needed))
         (will-ignite
                (value)
                (default )
                (if-needed))
         (heat-of-combustion
                (value)
                (default 35)
                (if-needed ))
         (relative-weight
                (value)
                (default 1)
                (if-needed))
         (LC50
                (value)
                (default 20)
                (if-needed))))

(setf (get 'chair 'frame)
      '(chair (initial-slope-of-burn
                (value)
                (default moderate)
                (if-needed))
         (peak-rate-of-burn
                (value)
                (default 500)
                (if-needed))
         (will-ignite
                (value)
                (default table curtains-drapes)
                (if-needed))
         (heat-of-combustion
                (value)
                (default 20)
                (if-needed))
         (relative-weight
                (value)
                (default .3)
                (if-needed))
         (LC50
                (value)
                (default 20)
                (if-needed))))
```

```
(setf (get 'table 'frame)
      '(table (initial-slope-of-burn
                (value)
                (default slow)
                (if-needed))
         (peak-rate-of-burn
                (value)
                (default 500)
                (if-needed))
         (will-ignite
                (value)
                (default chair)
                (if-needed))
         (heat-of-combustion
                (value)
                (default 18)
                (if-needed))
         (relative-weight
                (value)
                (default .5)
                (if-needed))
         (LC50
                (value)
                (default 60)
                (if-needed))))

(setf (get 'chest 'frame)
      '(chest (initial-slope-of-burn
                (value )
                (default slow)
                (if-needed))
         (peak-rate-of-burn
                (value)
                (default 1000)
                (if-needed))
         (will-ignite
                (value)
                (default)
                (if-needed))
         (heat-of-combustion
                (value)
                (default 25)
                (if-needed))
         (relative-weight
                (value)
                (default 1)
                (if-needed))
         (LC50
                (value)
                (default 40)
                (if-needed))))

(setf (get 'wastebasket 'frame)
      '(wastebasket (initial-slope-of-burn
```

```
                (value )
                (default very-fast)
                (if-needed))
        (peak-rate-of-burn
                (value)
                (default 50)
                (if-needed))
        (will-ignite
                (value)
                (default bed)
                (if-needed))
        (heat-of-combustion
                (value)
                (default 20)
                (if-needed))
        (relative-weight
                (value)
                (default .05)
                (if-needed))
        (LC50
                (value)
                (default 60)
                (if-needed))))

(setf (get 'curtains-drapes 'frame)
      '(curtains-drapes (initial-slope-of-burn
                (value )
                (default very-fast)
                (if-needed))
        (peak-rate-of-burn
                (value)
                (default 100)
                (if-needed))
        (will-ignite
                (value)
                (default chair)
                (if-needed))
        (heat-of-combustion
                (value)
                (default 25)
                (if-needed))
        (relative-weight
                (value)
                (default .05)
                (if-needed))
        (LC50
                (value)
                (default 15)
                (if-needed))))

(setf (get 'occupant 'frame)
      '(occupant (awake
                (value )
                (default n)
```

```
                (if-needed))
        (mobile
                (value)
                (default y)
                (if-needed))
        (drunk
                (value )
                (default n)
                (if-needed))
        (heart-lung-condition
                (value )
                (default n)
                (if-needed))
        (time-to-escape
                (value )
                (default)
                (if-needed time-to-escape-d))
        (external-aid
                (value )
                (default n)
                (if-needed))
        (rescue-time
                (value )
                (default )
                (if-needed ext-rescue-time-d))))

(setf (get 'room 'frame)
      '(room (detector
                (value )
                (default n)
                (if-needed))
           (length
                (value)
                (default 4.572)
                (if-needed))
           (width
                (value)
                (default 3.658)
                (if-needed))
           (height
                (value)
                (default 2.438)
                (if-needed))
           (door-height
                (value)
                (default 2.134)
                (if-needed))
           (door-width
                (value)
                (default .762)
                (if-needed))
           (fraction-door-open
                (value)
                (default 1)
```

```
        (if-needed))
(window-width-opening
        (value)
        (default .762)
        (if-needed))
(window-height-opening
        (value)
        (default .61)
        (if-needed))
(in-big-building
        (value)
        (default y)
        (if-needed))
(dim-meter-feet
        (value)
        (default f)
        (if-needed))))
```

```
; file: fuelpack

(defun fuel-packages5 (x Z);list of all element in Z whose 1st
                           ; element is x
  (cond ((null x) z)       ; should have only one for fuel
                           ; packages
     ((null z) z)
     ((equal x (caar z))
      (cons (car z)(fuel-packages5 x (cdr z))))
     (t (fuel-packages5 x (cdr z)))))

(defun fuel-packages4 (z) ; fuel package due to 1st element- only
                          ; 1st pass
  (make-a-set
      (squash (cons (car z)
                (apply 'append (mapcar #'(lambda (w)
                                 (fuel-packages5 w (cdr z)))
                               (cdar z)))))))

(defun fuel-packages3 (z) ; fuel package from 1 pass and unused
                          ; elements
  (cons (fuel-packages4 z) ; cons the results of the 1st pass to
                           ; the cdr of the list
     (remove (car (fuel-packages5 (cadar z) z)) ;with the used
                                                ; part removed
          (cdr z))))

(defun fuel-packages2 (z) ; complete fuel package for 1st element
  (cond ((samesetp (car z) (fuel-packages4 z)) (car z))
     (t (setq z (fuel-packages3 z))
        (fuel-packages2 z))))

(defun fuel-packages1 (x z); x is an element of z. This determine
                           ; the fuel package due to this element
  (fuel-packages2 (cons x (remove x z))))

(defun fuel-packages (z) ; all the fuel packages
  (make-a-list-super (apply 'list (mapcar #'(lambda (w)
                                 (fuel-packages1 w z))
                               z))))

; We need the various items of furnishings that will be ignited by
; an item

(defun list-will-ignite-item (w) ; causal vector for w. Reports (w
x .. y)
                       ; where x .. y are caused by w. In this case
                       ; x .. y are items that w will-ignite.
  (remove nil (append (list w)(fget-v-d-p w 'will-ignite))))
          ; need to  remove nil because it ignites everything!

(defun list-will-ignite-items (y) ; Makes a list of will-ignite
                ; causal  vectors for list of furnishing y. It is a
                      ; input for the procedure "fuel-packages".
```

```
(apply 'list (mapcar #'(lambda (x) (list-will-ignite-item x))
                     y)))
```

```
;; flashovr

; Thomas Flashover Equation

(defun aw () ; computes the area of the surfaces of the room
  (+ (* 2 (car (fget-v-d-p 'room 'length))
     (car (fget-v-d-p 'room 'width)))
     (* 2 (car (fget-v-d-p 'room 'length))
     (car (fget-v-d-p 'room 'height)))
     (* 2 (car (fget-v-d-p 'room 'height))
     (car (fget-v-d-p 'room 'width)))))

(defun vent-eff () ; computes the vent term in the Thomas equation
                   ; for flashover
  (+ (* (car (fget-v-d-p 'room 'window-height-opening))
     (car (fget-v-d-p 'room 'window-width-opening))
     (our-sqrt (car (fget-v-d-p 'room 'window-height-opening))))
     (* (car (fget-v-d-p 'room 'door-height))
     (car (fget-v-d-p 'room 'door-width))
     (car (fget-v-d-p 'room 'fraction-door-open))
     (our-sqrt (car (fget-v-d-p 'room 'door-height))))))

(defun thomas-flashover-power () ; Thomas equation for the power
needed for
                      ; flashover
  (+ (* 278.0 (vent-eff)) (* 7.8 (aw))))

; Logically we now would go to the fuelpack file to get the
; procedure  "fuel-packages". We assume we have done that so we
; now have all the fuel  packages. We will now determine if
; burning any of the fuel packages will lead to flashover. We
; assune all the maximum burning rates will occur at the same
; time so that they add. This is a conservative estimation.

(defun total-peak-of-list (y) ; the sum of all the peak burning
                              ; rates for all items in list y
  (apply '+ (mapcar #' (lambda (w) (car (fget-v-d-p w 'peak-rate-
of-burn)))
               y)))

(defun sum-packages (y) ; the sum of the peak burning rate for
                  ; all fuel packages, (sum package) in list y
  (mapcar #'(lambda (w)
            (append (list (total-peak-of-list w))
                  w))
       (fuel-packages (list-will-ignite-items y))))

(defun max-pack (y) ; the max fuel package (power package)
  (assoc
  (apply 'max                ; max power
  (mapcar #'car (sum-packages y)))  ; a list of all powers
      (sum-packages y)))

;; What we want is all the packages that could cause flashover and
```

```
;; then select a package that would cause flashover in the
;; shortest time.

(defun flash-packs (y) ; a list of all fuel packages that could
cause
                  ; flashover (sum package)
  (remove nil (mapcar #'(lambda (w)
                  (cond ((> (car w)
                            (thomas-flashover-power))
                         w)
                        (t nil)))
                (sum-packages y))))

(defun name-pack-flash (y) ; name of the flashover fuel package
  (cddr (fastest-flash-pack (flash-packs-slope y))))

; If the room will flashover, we need to determine how long it
; will take to flashover.

; For this version we proceed as follows:
;    a. pick the item with DQmax > 200KW & with the fastest slope.
;    b. add all the peak burning rates in the fuel package,
;       DQpack.
;    c. use the slope of item found in step a, Kf. So we have
;         DQpack = Kf t^2
;    d. if DQfo < DQpack, combute time to flashover.

;----------------
; Slope for items with DQmax >200Kw

(defun list-slope (y); list of (item slope) if DQmax >200
  (remove nil (mapcar #'(lambda (w)
                  (cond ((> (car (fget-v-d-p  w 'peak-rate-of-
burn))
                            200)
                         (cons w
                               (fget-v-d-p  w 'initial-slope-of-burn)))
                        (t nil)))
                y)))

;---------------
;Now we want the fastest slope of any item in a list

(defun list-fastest-slope (y) ; fastest slope of a list
  (or
      (list-slope1 'very-fast y)
      (list-slope1 'fast y)
      (list-slope1 'moderate y)
      (list-slope1 'slow y)))

(defun list-slope1 (s y)
      (cond ((null y) nil)
         ((equal s (cadar (list-slope y)))
          s)
```

```
        (t (list-slope1 s (cdr y)))))

(defun flash-packs-slope (y) ; (fastest-slope sum-DQmax fuel-pack)
  (mapcar #'(lambda (w)      ; for flashover packages
              (cons (list-fastest-slope (cdr w))
                    w))
          (flash-packs y)))

(defun fastest-flash-pack (y) ; (fastest-slope sum-DQ package)
  (cond ((slope-pack 'very-fast y))
        ((slope-pack 'fast y))
        ((slope-pack 'moderate y))
        ((slope-pack 'slow y))))

(defun slope-pack (s y)
  (cond ((null y) nil)
        ((equal s (caar y)) (car y))
        (t (slope-pack s (cdr y)))))

(defun slope (y) ; converts a verbal slope into a numerical one
KW/s^2
  (cond ((equal 'slow y) (/ 1000 (* 600.0 600)))
        ((equal 'moderate y) (/ 1000 (* 300 300.0)))
        ((equal 'fast y) (/ 1000 (* 150 150.0)))
        ((equal 'very-fast y) (/ 1000 (* 75 75.0)))
        (t 0)))

(defun time-to-flashover (y) ; time to flashover
  (let ((x (fastest-flash-pack (flash-packs-slope y))))
  (cond (x
         (our-sqrt (/ (thomas-flashover-power)
                      (slope (car x)))))
        (t 100000.0))))
```

```
; toxic


(defun time-to-max (y) ; the time it takes an item to reach its
maximum
                  ; burning rate.
  (our-sqrt (/ (car (fget-v-d-p y 'peak-rate-of-burn))
               (slope (car (fget-v-d-p y 'initial-slope-of-burn))))))

(defun volume-m () ; volume of room in cubic meters
  (* (car (fget-v-d-p 'room 'width))
     (car (fget-v-d-p 'room 'height))
     (car (fget-v-d-p 'room 'length))))

(defun mass-burned (y t) ; mass of item,y, burned in t seconds
  (* 1000 (/ (* (slope (car (fget-v-d-p y 'initial-slope-of-
burn))) t t t)
     (* 3 (car (fget-v-d-p y 'heat-of-combustion))))))

(defun weight-average-LC50 (y) ; weighted average of LC50 for list
of items
  (/ (apply '+ (mapcar #'(lambda (w)
                   (* (car (fget-v-d-p w 'relative-weight))
                      (car (fget-v-d-p w 'LC50))))
                 y))
     (apply '+ (mapcar #'(lambda (w)
                   (car (fget-v-d-p w 'relative-weight)))
                 y))))

(defun weight-average-heat-of-combustion (y) ; weighted average of
                          ; heat-of-combustion for list of items
  (/ (apply '+ (mapcar #'(lambda (w)
                   (* (car (fget-v-d-p w 'relative-weight))
                      (car (fget-v-d-p w 'heat-of-combustion))))
                 y))
     (apply '+ (mapcar #'(lambda (w)
                   (car (fget-v-d-p w 'relative-weight)))
                 y))))

; We need a list of all packages and their slopes. We can use
; "fastest-flash-pack y" to add the slope to "sum-packages y"

(defun packs-slope (y) ; (fastest-slope sum-DQmax fuel-pack)
  (mapcar #'(lambda (w)       ; for flashover packages
              (cons (list-fastest-slope (cdr w))
                w))
          (sum-packages y)))

(defun time-to-toxic4 (y) ; time to toxic for each package
  (mapcar #'(lambda (w)
              (cons       ; cons the time to toxic hazard with package
                 (our-cube-root (/ (* 3
                                      (weight-average-heat-of-combustion
                                                   (cddr w))
```

```
                        3 ; this is the "r" factor in our eq.
                        (weight-average-LC50 (cddr w))
                        .5 ; the volume of interest is
                        (volume-m)) ; half the total
                        (slope (car w))))
              (cddr w))) ; the package
       (packs-slope y)))

;; The room and occupant's conditions may change the time to toxic
;; condition but will not change the relative sizes for each fuel
;; package.

(defun time-to-toxic3 (y) ; minimum time to toxic conditions for
normal person
  (apply 'min (mapcar #'car (time-to-toxic4 y))))

(defun toxic-pack (y) ; list the package that gets to toxic hazard
first
  (cdr (assoc (time-to-toxic3 y) (time-to-toxic4 y))))

(defun window-area (y)
  (* (car (fget-v-d-p 'room 'window-height-opening))
     (car (fget-v-d-p 'room 'window-width-opening))))

(defun door-area (y)
  (* (car (fget-v-d-p 'room 'door-height))
     (car (fget-v-d-p 'room 'door-width))
     (car (fget-v-d-p 'room 'fraction-door-open))))

(defun time-to-toxic1 (y) ; adjustments due to venting and
                          ; environment
  (cond ((> (window-area y)
            (* 16 .3048 .3048)) ; converting 16 sq ft into m^2
         100000.0) ; window area > 16 sq ft, never become toxic
        ((and (> (+ (window-area y) (door-area y))
              (* 16 .3048 .3048))
              (equal 'y (car (fget-v-d-p 'room 'in-big-building))))
         100000.0) ; in big building & venting > 16 sq ft
        ((< (door-area y) .0001) ; door is closed if opening < 1 cm^2
         (/ (time-to-toxic3 y)
            (- 1 (/ (window-area y)(* 16 .3048 .3048)))))
        ((equal 'y (car (fget-v-d-p 'room 'in-big-building))) ; large
         (/ (time-to-toxic3 y)
              (- 1 (/ (+ (window-area y) (door-area y))
                   (* 16 .3048 .3048)))))
        ((> (door-area y) (* 16 .3048 .3048))
         (/ (* 1.44225 (time-to-toxic3 y)) ; 3^1/3
            (- 1 (/ (window-area y)(* 16 .3048 .3048)))))
        (t (/ (* (+ 1 (* (/ (door-area y)(* 16 .3048 .3048))
                         (- 1.44225 1))) ; 3^1/3 - 1
              (time-to-toxic3 y))
              (- 1 (/ (window-area y)(* 16 .3048 .3048)))))))

(defun time-to-toxic (y) ; takes into account lung-heart condition
```

```
                          ; & drunkness
  (cond ((or (equal 'y (car (fget-v-d-p 'occupant 'heart-lung-
condition)))
             (equal 'y (car (fget-v-d-p 'occupant 'drunk))))
         (/ (time-to-toxic1 y) (our-cube-root 3)))
        (t (time-to-toxic1 y))))

(defun driver (y) ; returns the fuel package that leads to the 1st
                  ; hazard
  (let ((w (time-to-flashover y))
        (v (time-to-toxic y)))
  (cond ((> w v)
         (toxic-pack y))
        ((< w v)
         (name-pack-flash y))
        (t nil))))

;; problem must use slope of package that will lead to hazardous
;; conditions first

(defun time-to-detection (y power) ; time for an awake occupant to
                                   ; detect a fire without a detector
  (cond ((> power 9000.0) 100000.0) ; default power to give
                                    ; default time
        (t (our-sqrt (/ power (slope (list-fastest-slope (driver
y)))))))

(defun time-to-awareness-detector (y)
  (let ((k (list-fastest-slope (driver y))))
  (cond ((equal 'very-fast k) 15)
        ((equal 'fast k) 30)
        ((equal 'moderate k) 50)
        ((equal 'slow k) 100)
        (t 100000.0))))

(defun time-to-hazard (Y)
  (let ((w (time-to-flashover y))
        (v (time-to-toxic y)))
    (cond ((> w v) v)
          (t w))))

(defun cal-time-to-escape (y)
  (cond
     ((and (equal 'y (car (fget-v-d-p 'occupant 'awake)))
           (equal 'n (car (fget-v-d-p 'occupant 'drunk)))
           (equal 'y (car (fget-v-d-p 'occupant 'mobile))))
      (+ (time-to-detection y 25)            ; time to detect fire
         (car (fget-v-d-p 'occupant 'time-to-escape)))) ; travel
                                                        ; time
     ((and (equal 'n (car (fget-v-d-p 'occupant 'drunk)))
           (equal 'y (car (fget-v-d-p 'room 'detector)))
           (equal 'y (car (fget-v-d-p 'occupant 'mobile))))
      (+ (time-to-awareness-detector y)
         (car (fget-v-d-p 'occupant 'time-to-escape))))
```

```
     (t (+ (time-to-awareness-detector y)
           (car (fget-v-d-p 'occupant 'rescue-time))))))  ; if he
can't get
                                   ; out, use rescue time +

(defun time-to-escape-d (frame slot)
  (fput frame slot 'value
             (cal-time-to-escape f$f))
  (fget-v-d-p frame slot))

(defun time-to-escape-d (frame slot) ; travel time to escape
    (fput frame slot 'value
       (+ (car (fget-v-d-p 'room 'length))
          (car (fget-v-d-p 'room 'width))))
  (fget-v-d-p frame slot)) ; we had to add this or the 1st time we
called
                           ; this

(defun margin-of-safety (Y)
  (- (time-to-hazard y) (cal-time-to-escape y)))

(defun ext-rescue-time (y)
  (cond ((equal 'y (car (fget-v-d-p 'occupant 'external-aid)))
      (cond ((equal 'y (car (fget-v-d-p 'room 'detector)))
          (+ (time-to-awareness-detector y) 30))
            (t 300)))
     (t 100000.0)))

(defun ext-rescue-time-d (frame slot)
  (fput frame slot 'value
     (ext-rescue-time f$f))
  (fget-v-d-p frame slot))
```

```
;inputjr2

(defun framass ()  ; produces assertions from all our frames
  (framasso 'occupant)
  (framassr 'room))

(defun framasso (frame)  ; produces assertions from a occupant
                         ; frame
  (setq assertions (append assertions
  `((It will take ,(car (fget-v-d-p frame 'time-to-escape))
      seconds for the
     ,frame to escape the room)
    (The ,frame is
      ,(cond ((equal (car (fget-v-d-p frame 'awake)) 'y) 'awake)
          (t 'asleep)))
    (The ,frame is
      ,(cond ((equal (car (fget-v-d-p frame 'mobile)) 'y) 'mobile)
          (t 'nonmobile)))
    (The ,frame is
      ,(cond ((equal (car (fget-v-d-p frame 'drunk)) 'y) 'drunk)
          (t 'sober)))
    (The ,frame has
      ,(cond ((equal (car (fget-v-d-p frame 'heart-lung-
condition)) 'n)
           'n)
           (t 'a))
      heart-lung condition)
    (There is
        ,(cond ((equal (car (fget-v-d-p frame 'external-aid)) 'y)
                 'someone)
               (t 'no-one))
        to help)))))

(defun framassr (frame)          ; produces assertions from a room
                                 ; frame
  (setq assertions (append assertions
  `((The room
      ,(cond ((equal (car (fget-v-d-p frame 'detector)) 'y)
           'has-a-detector)
          (t 'does-not-have-a-detector)))
    (The length of the ,frame is ,(car (fget-v-d-p frame 'length))
meters)
    (The width of the ,frame is ,(car (fget-v-d-p frame 'width))
meters)
    (The height of the ,frame is ,(car (fget-v-d-p frame 'height))
meters)
    (The door-width of the ,frame is
      ,(car (fget-v-d-p frame 'door-width)) meters)
    (The door-height of the ,frame is meters
      ,(car (fget-v-d-p frame 'door-height)) meters)
    (The fraction the door is openned
      ,(car (fget-v-d-p frame 'fraction-door-open)))
    (The width of the window opening of the ,frame is
      ,(car (fget-v-d-p frame 'window-width-opening)) meters)
```

```
    (The height of the window opening of the ,frame is
      ,(car (fget-v-d-p frame 'window-height-opening)) meters)))))

(defun reset-values (frame slot) ; removes the value of the
"value" facet
  (fremove frame slot 'value (car (fget frame slot 'value))))

(defun reset-values-all (frame)
  (mapcar #'(lambda (y) (reset-values frame y))
                    (list-slots frame)))

(defun list-slots (frame) ; produces a list of all the slots of
this frame
  (mapcar 'car (cdr (get frame 'frame))))

(defun reset-all () ; fremoves all values
  (mapcar #'(lambda (w) (mapcar #'(lambda (y) (reset-values w y))
                    (list-slots w)))
       '(bed chest chair table wastebasket curtains-drapes
occupant room)))

(defun request-values ()  ; request values for all our frames
  (request-valuesf 'bed)
  (request-valuesf 'chair)
  (request-valuesf 'chest)
  (request-valuesf 'table)
  (request-valuesf 'wastebasket)
  (request-valuesf 'curtains-drapes)
  (request-valueso 'occupant)
  (request-valuesr 'room))

(defun message1 ()
        (terpri)
        (terpri)
        (princ '(It will take a few minutes to compute the
results))
        (terpri)
        (terpri)
        (princ `|While you are waiting, let me explain very
briefly what |)
           (terpri)
        (princ }|I am doing. I first use the "will-ignite"
information to|)
           (terpri)
        (princ `|compute all the fuel packages in the room and
their peak|)
           (terpri)
        (princ `|burning rates and initial slopes. I then
determine if|)
        (terpri)
           (princ `|the room will flashover. If it will, the time
to|)
           (terpri)
```

```
        (princ `|flashover is computed. For a normal person, I
assume a|)
           (terpri)
        (princ `|toxic condition exists whenever the average
concentration of|)
           (terpri)
        (princ `|burned material produces a concentration in all
the |)
           (terpri)
           (princ `|space accessible to the smoke of 1.5 times the
|)
           (terpri)
        (princ `|average LC50 of the burning materials. The time
to a |)
           (terpri)
        (princ `|hazardous condition is the smaller of the time to
|)
           (terpri)
        (princ `|flashover or the time to a toxic hazard. You
either|)
           (terpri)
        (princ `|provide me with the time of escape or rescue, or
I |)
           (terpri)
        (princ `|will compute them. Finally, the margin of safety
is |)
           (terpri)
        (princ `|computed. It is the difference between the time
to |)
           (terpri)
        (princ `|hazard and the escape or rescue time. Then, based
on|)
           (terpri)
        (princ `|rules provided by H. Nelson, I will draw a
conclusion|)
           (terpri)
        (princ `|about the risk level to which the occupant is
exposed.|)
        (terpri)
           (terpri)
        (princ `|After I have finished, I will print a"T". If you
want|)
           (terpri)
        (princ `|to know why I drew any conclusion I did, type |)
           (terpri)
        (princ `|           (whyjr 'identifyN) |)
           (terpri)
        (princ `|where N is the number given.|)
        (terpri))

(defun message2 ()
  (princ `|Do you want to change any of the parameters in the
status report? If you do,|)
  (terpri)
```

```
  (princ `!enter the item number and then either the associated
letter or X <ret> for!)
  (terpri)
  (princ `!all the parameters.If you enter X <ret> and p for any
value, you will get the!)
  (terpri)
  (princ `!default value for that parameter.To pass the rest, type
"p" <ret> "p" <ret>.!)
  (terpri))

(defun request-values2-all (n)  ; used to change all values for a
frame n
    (cond ((equal n 1) (reset-values-all 'bed)
        (request-valuesf 'bed)(request-values2))
       ((equal n 2) (reset-values-all 'chair)
        (request-valuesf 'chair)(request-values2))
       ((equal n 3) (reset-values-all 'chest)
        (request-valuesf 'chest)(request-values2))
       ((equal n 4) (reset-values-all 'table)
        (request-valuesf 'table)(request-values2))
       ((equal n 5) (reset-values-all 'wastebasket)
        (request-valuesf 'wastebasket)(request-values2))
       ((equal n 6) (reset-values-all 'curtains-drapes)
        (request-valuesf 'curtains-drapes)(request-values2))
       ((equal n 7) (reset-values-all 'occupant)
        (request-valueso 'occupant)(request-values2))
       ((equal n 8) (reset-values-all 'room)
        (request-valuesr 'room)(request-values2))))

(defun request-values2 ()  ; request values for all our frames
  (status)
  (message2)
  (let ((n (read))
     (m (read)))
   (cond ((equal n 'p) (message1))
       ((equal m 'x) (request-values2-all n))
       (t (change n m) (request-values2)))))

(defun add-value (frame slot)
    (let ((y (read)))
      (cond ((equal 'p y) 'pass)
         ((equal slot 'will-ignite)
          (cond ((or (equal y `bed)(equal y `chair)(equal y
`chest)
                 (equal y `table)(equal y `wastebasket)
                 (equal y `curtains-drapes))
                (fput frame slot 'value y))
               (t (reset-values frame slot)))
          (terpri)
          (princ `!If you do not want to add anymore items to the
will-ignite list type p!)
          (terpri)
          (add-value frame slot))
         ((not (equal frame 'room)) (fput frame slot 'value y))
```

```
        ((or (equal slot 'fraction-door-open)
             (equal slot 'detector)
             (equal slot 'in-big-building)
             (equal slot 'dim-meter-feet))(fput frame slot 'value
y))
        (t (fput frame slot 'value (/ y (metric))))))


(defun request-valuesf (frame)   ; request values for a furnishing
frame
  (print `(The initial slope of burning of the ,frame is - zero
slow moderate
            fast or very-fast))
  (add-value frame 'initial-slope-of-burn)
  (print `(The peak rate of burning of the ,frame is -in kW))
  (add-value frame 'peak-rate-of-burn)
  (print `(List the items the ,frame will ignite))
  (add-value frame 'will-ignite)
  (print `(The heat of combustion of the ,frame in KJ/g is?))
  (add-value frame 'heat-of-combustion)
  (print `(The ,frame LC50 value is in mg/l or g/m^3?))
  (add-value frame 'LC50))

(defun request-valueso (frame)     ; request values for a occupant
frame
  (print `(The ,frame is awake- y or n?))
  (add-value frame 'awake)
  (print `(The ,frame is mobile- y or n))
  (add-value frame 'mobile)
  (print `(Is the ,frame drunk?))
  (add-value frame 'drunk)
  (print `(Does the ,frame have a heart or lung condition?))
  (add-value frame 'heart-lung-condition)
  (print `(Is there someone outside the room that can come to the
aid of
           the ,frame - y or n))
  (add-value frame 'external-aid)
  (print `(How many seconds will it take for the ,frame to travel
out of
            the room?))
  (add-value frame 'time-to-escape)
  (print `(How many seconds will it take for the person outside
the room
            to travel into the room and take the occupant out?))
  (add-value frame 'rescue-time))


(defun m-or-f ()
  (cond ((equal 'm (car (fget-v-d-p 'room 'dim-meter-feet)))
      'meter)
     (t 'feet)))

(defun request-valuesr (frame)          ; request values for a
room frame
```

```
  (print `(The dimensions are in feet (f) or meters (m)? - f or
m?))
  (add-value frame 'dim-meter-feet)
  (print `(The length of the ,frame is ?  ,(m-or-f)?))
  (add-value frame 'length)
  (print `(The width of the ,frame is ? ,(m-or-f)?))
  (add-value frame 'width)
  (print `(The height of the ,frame is ? ,(m-or-f)?))
  (add-value frame 'height)
  (print `(The width of the door of the ,frame is ? ,(m-or-f)?))
  (add-value frame 'door-width)
  (print `(The height of the door of the ,frame is ? ,(m-or-f)?))
  (add-value frame 'door-height)
  (print `(The fraction the door of the ,frame is openned?))
  (add-value frame 'fraction-door-open)
  (print `(The width of the window opening of the ,frame is ? ,(m-
or-f)?))
  (add-value frame 'window-width-opening)
  (print `(The height of the window opening of the ,frame is ?
,(m-or-f)?))
  (add-value frame 'window-height-opening)
  (print `(The ,frame has a fire detector - y or n))
  (add-value frame 'detector)
  (print `(The ,frame is in a large building or no building - y or
n))
  (add-value frame 'in-big-building))

(defun input ()
  (princ `|Type any letter and "return" to continue. |)
  (read)
   (terpri)
   (terpri)
  (status)
  (print `(Do you want to input data instead of using default
data? Y or N))
  (cond ((equal 'y (read)) (request-values2))
     (t (message1))))

(defun time-to-toxic-assertion (y)
  (setq t$t (time-to-toxic y))
  (setq tp$tp (toxic-pack y))
  (princ `(The room will be toxic with the fuel package
          ,tp$tp
          burning in
          ,t$t seconds))
  (terpri)
      (setq assertions (append assertions
          `((The room will be toxic with the fuel package
                 ,tp$tp
                 burning in
                 ,t$t seconds)))))

(defun thomas-flashover-package (y) ; produces assertion about the
                                    ; room flashing over
```

```
  (setq o$o (time-to-flashover y))
  (cond ((flash-packs y)
      (princ `((The room will flashover with the fuel package
                  ,(name-pack-flash y)
                  burning in
                  ,o$o seconds)))
      (setq assertions (append assertions
          `((The room will flashover with the fuel package
                  ,(name-pack-flash y)
                  burning in
                  ,o$o seconds)))))
     (t (setq assertions (append assertions
          `((The room will not flashover with the biggest fuel
package
                  burning))))
        (princ `((The room will not flashover with the biggest
fuel package
                  burning ,(cdr (max-pack y)) at a maximum power of
                 ,(car (max-pack y)) KW)
               (The power needed for flashover is
                 ,(thomas-flashover-power) KW))))))

(defun flash-or-toxic (y)
  (princ `(The time to a hazardous condition is
            ,(cond (() o$o t$t)
                  t$t)
                 (t o$o)) seconds))
  (terpri)
  (terpri)
  (princ `(The hazardous conditions are due to
            ,(cond (() o$o t$t)
                  'due-to-toxic-gases)
                 (t 'due-to-flashover))))
  (terpri)
  (terpri)
  (princ `(The occupant will need ,(cal-time-to-escape y) seconds
to escape from the room))
   (terpri)
   (terpri)
  (princ `(Therefore the margin of safety for the occupant to
escape harm is,(margin-of-safety y) seconds))
   (terpri)
   (terpri))

(defun flash-or-toxic-ass (y)
  (cond (() o$o t$t)
      (setq assertions (append assertions
          `((The time to flashover is greater than the time to
                  toxic hazard)))))
     (t (setq assertions (append assertions
          `((The time to toxic hazard is greater than the time to
                   flashover))))))

(defun margin-of-safety-ass (y) ; fast version
```

```
  (setq m$m (margin-of-safety f$f))
  (cond (() (time-to-hazard y) 99000.0)
      (setq assertions (append assertions
          `((The room never becomes hazardous)))))
     ((< m$m 0)
      (setq assertions (append assertions
          `((The margin of safety is negative)))))
     (() m$m 300)
      (setq assertions (append assertions
          `((The margin of safety is greater than five
minutes)))))
     ((< m$m 60)
      (setq assertions (append assertions
          `((The margin of safety is less than one minute)))))
     (() m$m 60)
      (setq assertions (append assertions
          `((The margin of safety is greater than one minute
                and less than five minutes)))))
     (t nil)))

(defun margin-saf-evacuate-ratio-assx (y) ;slow version
  (cond (() (/ (margin-of-safety y) (cal-time-to-escape y)) 2)
      (setq assertions (append assertions
          `((The ratio of the margin-of-safety time to the
                escape-time is greater than two)))))
     (() (/ (margin-of-safety y) (cal-time-to-escape y)) 1)
      (setq assertions (append assertions
          `((The ratio of the margin-of-safety time to the
                escape-time is greater than one and less than
              two)))))
     (() (/ (margin-of-safety y) (cal-time-to-escape y)) 0)
      (setq assertions (append assertions
          `((The ratio of the margin-of-safety time to the
                escape-time is less than one)))))
     (t `((The ratio of the margin-of-safety time to the escape-
time is
          negative)) nil)))

(defun margin-saf-evacuate-ratio-ass (y) ; fast version
  (setq e$e (cal-time-to-escape f$f))
  (cond (() (/ m$m e$e) 2)
      (setq assertions (append assertions
          `((The ratio of the margin-of-safety time to the
                escape-time is greater than two)))))
     (() (/ m$m e$e) 1)
      (setq assertions (append assertions
          `((The ratio of the margin-of-safety time to the
                escape-time is greater than one and less than
              two)))))
     (() (/ m$m e$e) 0)
      (setq assertions (append assertions
          `((The ratio of the margin-of-safety time to the
                escape-time is less than one)))))
```

```
     (t `((The ratio of the margin-of-safety time to the escape-
time is
            negative)) nil)))

(defun change (x y)
  (cond ((equal x 1) (change2 'bed y))
     ((equal x 2) (change2 'chair y))
     ((equal x 3) (change2 'table y))
     ((equal x 4) (change2 'chest y))
     ((equal x 5) (change2 'wastebasket y))
     ((equal x 6) (change2 'curtains-drapes y))
     ((equal x 7) (change2 'occupant y))
     ((equal x 8) (change2 'room y))))

(defun change2 (z y)
  (cond ((or (equal z 'bed)(equal z 'chair)(equal z 'table)(equal
z 'chest)
          (equal z 'wastebasket)(equal z 'curtains-drapes))
       (cond ((equal y 'A)(change1 z 'peak-rate-of-burn))
             ((equal y 'B)(change1 z 'heat-of-combustion))
             ((equal y 'C)(change1 z 'LC50))
             ((equal y 'D)(change1 z 'initial-slope-of-burn))
             ((equal y 'E)(change1 z 'will-ignite))))
     ((equal z 'occupant)
       (cond ((equal y 'A)(change1 z 'awake))
             ((equal y 'B)(change1 z 'mobile))
             ((equal y 'C)(change1 z 'drunk))
             ((equal y 'D)(change1 z 'heart-lung-condition))
             ((equal y 'E)(change1 z 'external-aid))
             ((equal y 'F)(change1 z 'time-to-escape))
             ((equal y 'G)(change1 z 'rescue-time))))
     ((equal z 'room)
       (cond ((equal y 'A)(change1 z 'detector))
             ((equal y 'B)(change1 z 'length))
             ((equal y 'C)(change1 z 'width))
             ((equal y 'D)(change1 z 'height))
             ((equal y 'E)(change1 z 'door-height))
             ((equal y 'F)(change1 z 'door-width))
             ((equal y 'G)(change1 z 'fraction-door-open))
             ((equal y 'H)(change1 z 'window-width-opening))
             ((equal y 'I)(change1 z 'window-height-opening))
             ((equal y 'J)(change1 z 'in-big-building))
             ((equal y 'K)(change1 z 'dim-meter-feet))))))

(defun change1 (x y)
  (terpri)
  (cond ((not (equal x 'room))
      (princ `(Old value was ,(car (fget-v-d-p x y ))))
      (terpri)
      (fremove x y 'value (car (fget x y 'value)))
      (princ `|What is new value?|)
      (terpri)
      (fput x y 'value (read)))
     ((or (equal y 'detector)
```

```
      (equal y 'fraction-door-open)
      (equal y 'in-big-building)
      (equal y 'dim-meter-feet))
  (princ `(Old value was ,(car (fget-v-d-p x y ))))
  (terpri)
  (fremove x y 'value (car (fget x y 'value)))
  (princ `|What is new value?|)
  (terpri)
  (fput x y 'value (read)))
 (t (cond ((equal 'm (car (fget-v-d-p x 'dim-meter-feet)))
      (princ `(Old value was ,(car (fget-v-d-p x y ))))
      (terpri)
      (fremove x y 'value (car (fget x y 'value)))
      (princ `|What is new value?|)
      (terpri)
      (fput x y 'value (read)))
        (t (princ `(Old value was ,(* (metric)
                            (car (fget-v-d-p x y )))))
        (terpri)
        (fremove x y 'value (car (fget x y 'value)))
        (princ `|What is new value?|)
        (terpri)
        (fput x y 'value (/ (read) (metric)))))))))
```

```
;file status

(defun header ()
  (princ `|              A         B           C          D
E|)
  (terpri)
  (princ `|furnishing   peak  h.-of-comb.  LC50 initial-slope
will-ignite|))

(defun statusf1 (y)
  (terpri)
  (cond ((equal y 'bed)(princ `|1. Bed        |))
     ((equal y 'chair) (princ `|2. Chair      |))
     ((equal y 'table) (princ `|3. Table      |))
     ((equal y 'chest) (princ `|4. Chest      |))
     ((equal y 'wastebasket)(princ '|5. Wastebasket |))
     ((equal y 'curtains-drapes)(princ '|6. Curtains  |))))

(defun statusf (y)
  (statusf1 y)
  (princ (car (fget-v-d-p y 'peak-rate-of-burn)))
  (cond ((or (equal y 'bed)(equal y 'wastebasket)(equal y 'chest))
      (princ `|       |))
     (t (princ `|        |)))
  (princ (car (fget-v-d-p y 'heat-of-combustion)))
  (princ `|         |)
  (princ (car (fget-v-d-p y 'LC50)))
  (princ `|       |)
  (princ (car (fget-v-d-p y 'initial-slope-of-burn)))
  (cond ((or (equal (car (fget-v-d-p y 'initial-slope-of-burn))
'slow)
          (equal (car (fget-v-d-p y 'initial-slope-of-burn))
'fast))
      (princ `|         |))
     (t (princ `|     |)))
  (princ (fget-v-d-p y 'will-ignite)))

(defun statusf-all ()
  (header)
  (statusf 'bed)
  (statusf 'chair)
  (statusf 'table)
  (statusf 'chest)
  (statusf 'wastebasket)
  (statusf 'curtains-drapes)
  (terpri)
  (terpri))

(defun headero ()
  (princ `|7. Occupant|)
  (terpri)
  (princ `|  A       B      C        D             E              F
G|)
  (terpri)
```

```
  (princ `|awake  mobile  drunk  heart-lung  external-aid  time-
to-es  rescue-time|)
  (terpri))

(defun statuso ()
  (headero)
  (princ `|  |)
  (princ (car (fget-v-d-p 'occupant 'awake)))
  (princ `|      |)
  (princ (car (fget-v-d-p 'occupant 'mobile)))
  (princ `|       |)
  (princ (car (fget-v-d-p 'occupant 'drunk)))
  (princ `|      |)
  (princ (car (fget-v-d-p 'occupant 'heart-lung-condition)))
  (princ `|              |)
  (princ (car (fget-v-d-p 'occupant 'external-aid)))
    (princ `|            |)
  (princ (car (fget-v-d-p 'occupant 'time-to-escape)))
    (princ `|            |)
  (princ (car (fget-v-d-p 'occupant 'rescue-time)))
  (terpri))

(defun headerr1 ()
  (cond ((equal 'm (car (fget-v-d-p 'room 'dim-meter-feet)))
      (princ '|8. ROOM - dimensions in meters|))
     (t (princ '|8. ROOM - dimensions in feet|)))
  (terpri)
  (princ `|   A         B      C      D         E             F
G|)
  (terpri)
  (princ `|detector  length  width  height  door-height  door-
width  fraction-open|)
  (terpri))

(defun headerr2 ()
  (princ `|      H             I             J          K|)
  (terpri)
  (princ `|window-width  window-height  in-big-bldg  dim-m-f|)
  (terpri))

(defun metric ()
  (cond ((equal 'm (car (fget-v-d-p 'room 'dim-meter-feet))) 1.0)
     (t (/ 100 (* 2.54 12)))))

(defun statusr ()
  (headerr1)
  (princ `|  |)
  (princ (car (fget-v-d-p 'room 'detector)))
  (princ `|        |)
  (princ (* (metric)(car (fget-v-d-p 'room 'length))))
  (princ `|  |)
  (princ (* (metric)(car (fget-v-d-p 'room 'width))))
  (princ `|   |)
  (princ (* (metric)(car (fget-v-d-p 'room 'height))))
```

```
  (princ `|  |)
  (princ (* (metric)(car (fget-v-d-p 'room 'door-height))))
    (princ `|         |)
  (princ (* (metric)(car (fget-v-d-p 'room 'door-width))))
    (princ `|           |)
  (princ (car (fget-v-d-p 'room 'fraction-door-open)))
  (terpri)
  (headerr2)
    (princ `|     |)
  (princ (* (metric)(car (fget-v-d-p 'room 'window-width-
opening))))
    (princ `|          |)
  (princ (* (metric)(car (fget-v-d-p 'room 'window-height-
opening))))
    (princ `|             |)
  (princ (car (fget-v-d-p 'room 'in-big-building)))
    (princ `|            |)
  (princ (car (fget-v-d-p 'room 'dim-meter-feet)))
  (terpri))

(defun status ()
  (princ `|The present status of the room is:|)
  (terpri)
  (terpri)
  (statusf-all)
  (statusr)
  (statuso))
```

```
; whyjr

(defun whyjr (rule)
  (car (remove nil (mapcar #'(lambda (w)
           (cond ((equal rule (cadr w))
                  (cdr (caddr w)))
                 (t nil)))
       rules))))
```

NBS-114A (REV. 2-8C)

U.S. DEPT. OF COMM. BIBLIOGRAPHIC DATA SHEET (See instructions)	1. PUBLICATION OR REPORT NO. NBSIR-86/3319	2. Performing Organ. Report No.	3. Publication Date March 1986

4. TITLE AND SUBTITLE

ASKBUDJr: A Primitive Expert System for the Evaluation of the Fire Hazard of a Room

5. AUTHOR(S)

Richard L. Smith

6. PERFORMING ORGANIZATION (If joint or other than NBS, see instructions)	7. Contract/Grant No.
NATIONAL BUREAU OF STANDARDS DEPARTMENT OF COMMERCE Gaithersburg, MD 20899	8. Type of Report & Period Covered

9. SPONSORING ORGANIZATION NAME AND COMPLETE ADDRESS (Street, City, State, ZIP)

10. SUPPLEMENTARY NOTES

☐ Document describes a computer program; SF-185, FIPS Software Summary, is attached.

11. ABSTRACT (A 200-word or less factual summary of most significant information. If document includes a significant bibliography or literature survey, mention it here)

The Center for Fire Research (CFR) has a long term project to develop expert systems as a technology transfer mechanism. CFR has as the long term goal of this project: to develop a computer program which will make an expert estimate of the fire safety of a building based on CFR's deterministic physical models, technical data, and the expert judgment of its staff. The first major program to be developed by this project is based on the expertise of Harold E. (Bud) Nelson. Thus, this program will be called ASKBUD. In this report, the first exploratory steps taken to develop an expert system for fire hazard evaluation are described. Also, the progress made to date, as well as some of the major problems that must be solved, will be discussed. Since the ASKBUD expert system discussed in this report is in its infancy, we call it ASKBUDJr.

12. KEY WORDS (Six to twelve entries; alphabetical order; capitalize only proper names; and separate key words by semicolons)

Artificial intelligence; computer programs; expert systems; fire safety; fire hazards

13. AVAILABILITY

☒ Unlimited

☐ For Official Distribution. Do Not Release to NTIS

☐ Order From Superintendent of Documents, U.S. Government Printing Office, Washington, D.C. 20402.

☒ Order From National Technical Information Service (NTIS), Springfield, VA. 22161

14. NO. OF PRINTED PAGES	15. Price
64	$11.95