F1: spec(mach1, ibm-pc, 320).
F2: spec(mach2, mac, 1000).
F3: spec(mach3, ibm-pc, 640).
F4: runs(ibm-pc, spreadsheet, 500).
F5: runs(ibm-pc, basic, 128).
F6: runs(ibm-pc, pascal, 286).
F7: runs(mach3, basic, 200).
F8: runs(mach3, smalltalk, 1000).
F9: access(sue, mach1).
F10: access(jerry, mach3).
F11: access(sam, mach1).
F12: access(sam, mach2).
F13: written_in(spreadsheet, pascal).
R1: can_use(P, SW) :- access(P, W),
    can_run(W, SW).
R2: can_run(M, SW) :- spec(M, HW, Mem1),
    runs(HW, SW, Mem2),
    Mem1 >= Mem2.
R3: can_run(M, SW) :- written_in(SW, L), can_run(L, W).

If we run the corresponding program with the goal

?- can_use(X, spreadsheet).

we will obtain the following output:

X = sue
X = jerry
X = jerry
X = sam

Notice that jerry is derived twice, indicating that the program is doing some unnecessary work. Let's see why this happened. The object jerry will satisfy can_use once because jerry has access to mach3, which can run a spreadsheet directly. But the goal can also be derived from the second rule for can_run since Pascal can run on mach3 as well and the spreadsheet is written in Pascal.

The cut is considered to be a subgoal that is universally satisfied but that cannot be backed up over. It then stands as a one-way gate for the subgoal generation process, which, once passed, cannot be backed through.

In the previous example, the first rule for can_run could be rewritten with the cut as follows:

can_run(MACH, SW) :- spec(MACH, HW, MEM1),
    runs(HW, SW, MEM2),
    MEM1 >= MEM2.

The cut is signified by an exclamation point (!).

After the first three subgoals are satisfied, the cut is then satisfied as well, completing the derivation of can_run. After the entire derivation is completed, the cut then prevents the backing up process from trying other alternatives for can_run. Thus unnecessary checking is avoided, since it is irrelevant how many ways a computer can_run a software product.

Now the goal

?- can_use(X, spreadsheet).

will find only one derivation for each of the three people, and the derivation process is displayed in Figure 11.9. The derivation process, when backing up to a cut, will automatically produce a failure for all subgoals
1. #can_use(X,spreadsheet).
2. #access(X,M),can_run(M,spreadsheet).
3. #can_run(mach1,spreadsheet).
4. #spec(mach1,HW,M1),runs(HW,spreadsheet,M2),M1>M2,!. R1/SW=spreadsheet.
5. #run(lbapo,spreadsheet,M2),320>M2,!. F1/M1=lbapo,M1=320
6. #320>500,!. F4/M2=500
7. #run(lbapo,spreadsheet,M2),320>M2,!. Fails
8. #spec(mach1,HW,M1),runs(HW,spreadsheet,M2),M1=M2,!. Fails
9. #can_run(mach1,spreadsheet).
10. #written_in(spreadsheet,L),can_run(mach1,L).
11. #can_run(mach1,pascal).
12. #spec(mach1,HW,M1),runs(HW,pascal,M2),M1>M2,!. R15/L1=pascal
13. #run(lbapo,pascal,M2),320>M2,!. R2
14. #320>500,!. P1/F1=lbapo,M1=320
15. #run(lbapo,pascal,M2),320>M2,!. F6/M2=256 satisfied
16. #! Skip backup because of cut - 5. completed satisfied X=sue
17. #written_in(spreadsheet,L),can_run(mach1,L).
18. #can_run(mach1,spreadsheet).
19. #access(X,M),can_run(M,spreadsheet).
20. #can_run(mach3,spreadsheet).
21. #spec(mach3,HW,M1),runs(HW,spreadsheet,M2),M1>M2,!. R3/SW=spreadsheet.
22. #run(lbapo,spreadsheet,M2),640>M2,!. P3/M1=640,HW=lbapo
23. #run(lbapo,spreadsheet,M2),640>M2,!. F4/M2=500 satisfied
24. #run(lbapo,spreadsheet,M2),640>M2,!. satisfied X=lbapo
25. #spec(mach1,HW,M1),runs(HW,spreadsheet,M2),M1>M2,!. F11/X1=san,M1=mach1
26. #can_run(mach1,spreadsheet).
27. #spec(mach1,HW,M1),runs(HW,spreadsheet,M2),M1>M2,!. R2/SW=spreadsheet.
28. #run(lbapo,spreadsheet,M2),320>M2,!. P1/F1=lbapo,M1=320
29. #run(lbapo,spreadsheet,M2),320>M2,!. F4/M2=500 satisfied
30. #run(lbapo,spreadsheet,M2),320>M2,!. satisfied X=lbapo
31. #! Skip backup because of cut - 3. completed satisfied X=san
32. #written_in(spreadsheet,L),can_run(mach1,L).
33. #can_run(mach1,spreadsheet).
34. #spec(mach1,HW,M1),runs(HW,spreadsheet,M2),M1>M2,!. R5/SW=spreadsheet.
35. #run(lbapo,spreadsheet,M2),320>M2,!. P5/M1=640,HW=lbapo
36. #run(lbapo,spreadsheet,M2),320>M2,!. F4/M2=500 satisfied
37. #run(lbapo,spreadsheet,M2),320>M2,!. satisfied X=lbapo
38. #! Skip backup because of cut - 5. completed satisfied X=san
39. #written_in(spreadsheet,L),can_run(mach1,L).
40. #can_run(mach1,spreadsheet).
41. #access(X,M),can_run(M,spreadsheet).
42. #can_run(mach2,spreadsheet).
43. #spec(mach2,HW,M1),runs(HW,spreadsheet,M2),M1>M2,!. R2/SW=spreadsheet.
44. #run(mac,pascal,M2),1000>M2,!. P2/FS=mac,M1=1000
45. #run(mac,pascal,M2),1000>M2,!. R2/SW=spreadsheet.
46. #spec(mach2,HW,M1),runs(HW,spreadsheet,M2),M1>M2,!. R3/SW=spreadsheet.
47. #run(mac,pascal,M2),1000>M2,!. R2/SW=spreadsheet.
48. #spec(mach2,HW,M1),runs(HW,spreadsheet,M2),M1>M2,!. R3/SW=spreadsheet.
49. #can_run(mach2,pascal).
50. #written_in(pascal,L),can_run(mach2,L).
51. #can_run(mach2,pascal).
52. #spec(mach2,HW,M1),runs(HW,pascal,M2),M1>M2,!. R3/SW=spreadsheet.
53. #run(mac,pascal,M2),1000>M2,!. R2/SW=spreadsheet.
54. #spec(mach2,HW,M1),runs(HW,pascal,M2),M1>M2,!. R3/SW=spreadsheet.
55. #can_run(mach2,pascal).
56. #written_in(pascal,L),can_run(mach2,L).
57. #can_run(mach2,pascal).
58. #access(X,M),can_run(M,spreadsheet).
59. #can_use(X,spreadsheet).