1 Hml Proof tool

There are three versions of the modal proof tool.

1. One finds proofs in Hennessy Milner Logic, (no recursion)
2. one finds proofs in Modal Mu (with recursion)
3. one finds proofs in Modal Mu with "types" (independence of actions)

To run the proof tool use the following command:
"proveIt hmlExample"
The files "testSh, hmlSml.sparc-solaris" must be in the current directory.
Create your own script from the hmlExample file.

2 Commands

The commonly used commands in the HML tool are:

P parses a back quoted string and returns an HML proposition

⊢ uses the characters "|" and "-" to turn a pair of lists of propositions into an assumption

xdviProofOrCe takes an assumption (or judgement) and returns either a proof (displayed on xdvi) or an agent (a counter example in the form of a tree) which satisfies the antecedent and does not satisfy the succedent.

Examples:

```
xdviProofOrCe ([P `<a><b><c>T`] |- [P `<a><b>T`]);
```

Displays a proof that, being able to perform an "a" action followed by a "b" action, followed by a "c" action implies that it is possible to perform an "a" action followed by a "b" action.

```
xdviProofOrCe ([P `<a><b>T`] |- [P `<a><b><c>T`]);
```

Displays an agent which can perform an "a" action followed by a "b" action but can't subsequently perform a "c" action.

```
xdviProofOrCe ([P `<a>T \/ <b><c>T`] |- [P `<a>T`, P `<b>T`]);
```

Displays another proof. What this really shows is how to use lists; the succedent is a list of propositions.

These commands are the only ones which I expect that a casual user might use. A more complete set follows.

3 More commands, for the cognescenti

A Proposition is a HML proposition (And's, Or's, Boxes, Posses, Variables) (e.g.

```
\Poss{a}\Bx{b}F\AND\Bx{c}F
```

) A Sequent is a pair of lists of Propositions.

```
(\Poss{a}\Bx{b}F,\Bx{c}F$\vdash$ \Poss{a}\Bx{b}F,\Bx{c}F)
```

A Rule is one of the lAnd,rOr, ... proof rules.

A Tableau is a proof tree (possibly closed) built up from a sequent and rules.

3.1 dp

The function "dp" will perform the decision procedure on the (cut free) tableau. It will return a tableau.

3.2 Rule functions

Some commands:

- antecedent : JUDGEMENT $\mapsto$ hml list
- succedent : JUDGEMENT $\mapsto$ hml list
- proofFold : (JUDGEMENT $\mapsto$ JUDGEMENT) $\mapsto$ JUDGEMENT $\mapsto$ JUDGEMENT
- isAProof : JUDGEMENT $\mapsto$ bool

The following functions apply the rules to a Judgement. They only make sense at Assumptions, and at Judgements they are the identity function. To be useful the proofFold function should be applied to them.

- lAnd
- rOr
- lFalse
- rTrue
- lOr
- rAnd
- boxRule
- possRule
- idRule

Example:

```
lAnd ([P /\ Q, <a>T, [b]F] |- DELTA)
```

will return

```
lAnd ([P, Q, <a>T, [b]F] |- DELTA)
```

A more sensible example would be:

$$proofFoldlAndtableau;$$

or $proofFold(lAndorOr)tableau$; where "o" is the composition operator. This returns a TABLEAU, and does not print out anything.

The preceeding list of commands either do too little or too much. Without the proofFold the commands do too little, and with the proof fold the commands work over every assumption. To regain a little more control, use the "at" function.

$$at : (Tableau \mapsto Tableau) \mapsto intlist \mapsto Tableau$$

The function

$$at : (proofFoldlOr)(0 :: 0 :: 2 :: nil)tableau$$

will take the left most child (the zero), its leftmost child (the second zero) and its "third" child (the 2) and apply the function (proofFold lOr) to that. Thus, only a part of the tree have the subsequent explosion due to distribution.

3.3 dualising functions

- dualJudgement (TABLEAU $\mapsto$ TABLEAU)
- dualProp : (PROP $\mapsto$ PROP)
- dualSeq
- dualRule (RULE $\mapsto$ RULE)

The function "dualJudgement" will take a tableau and return the dual of it (i.e. flip antecedent and seccedent, flip props to dual props)

3.4 Cuts

exception UN_CUTABLE

upL TABLEAU $\mapsto$ TABLEAU tries to push the cut up the left branch. If the rule isn't a cut, then it does nothing.

upR TABLEAU $\mapsto$ TABLEAU tries to push the cut up the right branch. If the rule isn't a cut, then it does nothing.

upBoth TABLEAU $\mapsto$ TABLEAU tries to push the cut up both branches (works only for a few rules) If the rule isn't a cut (or of the right form) then does nothing.

cutElim TABLEAU $\mapsto$ TABLEAU Tries upL; if it is unsuccessful tries upR; if it is unsuccessful tries upBoth.

cutElimR TABLEAU $\mapsto$ TABLEAU Tries cutElim with "fold"; thus, cutElimR at the root of a proof tree could affect internal nodes of the tree.

mkCut PROP * TABLEAU * TABLEAU ↦ TABLEAU Allows the user to take two proofs and "cut" them together. e.g. (mkCut (P Q, dp proof1, dp proof2) where proof1 and proof2 have Q in the succedent and antecedent respectively. Otherwise raises the exception UN_CUTTABLE

3.5 Pretty Print Proofs

ppPropText prop ↦ string takes a prop and pretty prints in in TEXT (not latex)

ppPropLatex prop ↦ string takes a prop and returns a string, which when run through latex will pretty print a proposition

ppJudgement TABLEAU ↦ string takes a tableau, and return a string, which when run through latex will pretty print a proof.

latexProlog string used as a prolog for latex and the proof

latexEpilog string used as a epilog for latex and the proof

val latexCaption] string ↦ string

Example:
The following string is a fine latex program:
print (ppProp.latexProlog ^(ppProp.ppJudgement proof) ^ppProp.latexEpilog);
So is the following, but it has a caption on the picture.
print (ppProp.latexProlog ^(ppProp.ppJudgement proof) ^(ppProp.caption "A Fine caption") ^ppProp.latexEpilog);

3.6 States

oops exception HAS_A_PROOF

counter TABLEAU ↦ state Finds a counter example for the failed proof; returns an agent (state) which satisfies the antecedent and doesn't satisfy the succedent. If there is a proof, it raises an exception.

ppStateText state ↦ string

ppStateLatex state ↦ string Given a counter example (an agent, a state) returns some latex which draws the darned thing.

latexProlog string

latexEpilog string

Example: print (State.latexProlog ^(State.ppStateLatex (State.counter failedProof)) ^(State.latexEpilog));
will print out a latex program which will draw the agent.