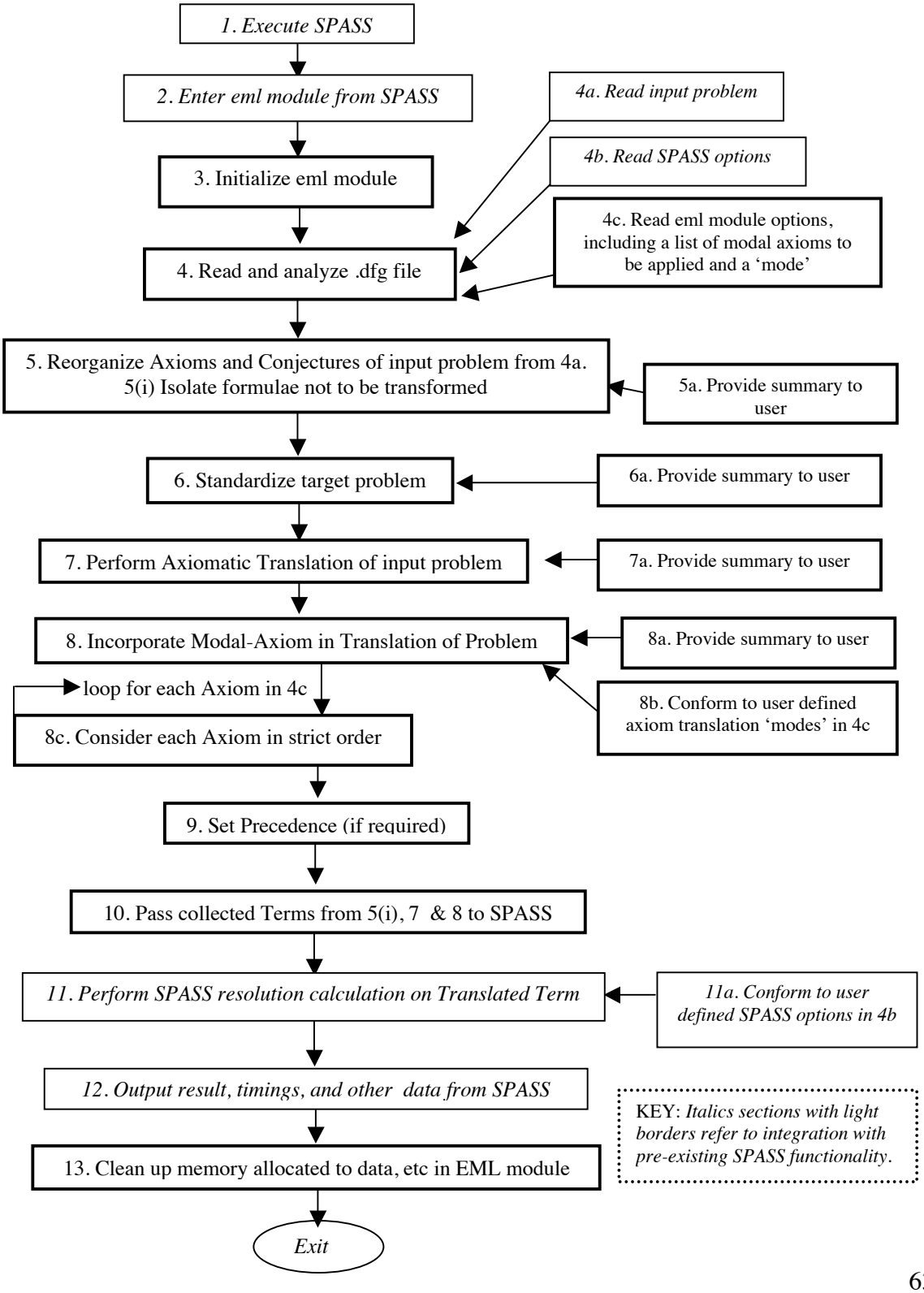
5. Specification.

The specification in this section is not of the formal kind that is favored by software engineers. Instead it has been developed beyond, into a format that is suitable for a user manual. Figure 5.1 shows an overview of the internal steps involved in the translation process. These steps are reflected in the output presented to the user that describes the axiomatic translation process.

Figure 5.1 Overview of Sequence of Tasks in Execution for Axiomatic Translation.



In the following description, an effort has been made to demonstrate the link between the translation produced by extended-SPASS and the output provided for the user. It is clear that it is possible to follow the translation process in detail. The tasks referred to are from figure 5.1.

5.1 Read and analyze .dfg file (Task 4).

In this section, subtasks related to the reading and analyses of the .dfg file are described.

5.1.1 Extensions to SPASS dfg syntax.

The basic SPASS input syntax can be seen in '*spass-input-syntax.pdf*' [17]. Extensions to this syntax implemented during this project were concentrated in the logical and settings sections of the .dfg file. Three different input modes were provided, in which the modal-axioms to be included in the translation can be defined. The use of these input modes is mutually exclusive. The priority is 'in-line definitions' > 'set_axiom definitions' > 'standard flag definitions'. In addition two flags that control the translation process were introduced. These bit-based `EMLAxiomOpt` and `EMLAxiomHidOpt` flags can be combined with all three modal-axiom definition modes in order to control the translation process. It is worth noting that all these three input methods are used to populate the *same* internal data-structure representing the modal-axioms that are to be used in the translation. (For details see section 6).

5.1.2 In-line axiom definition syntax.

For use in the special formulae section of the .dfg file, a collection of new multi-modal-axiom operators were added, with a syntax similar to *box* and *dia*. All modal-axioms are unary (take a single modal formula as an argument). A binary operator is implemented, the other argument being the modality index (for example, r). The dfg syntax is formally extended as shown in figure 5.2. (See also [17]). Typical usage of this syntax is to define the modal-axioms to be applied during the translation of a uni-modal and bi-modal target formula, as shown in figures 5.3 and 5.4 respectively. This in-line syntax is reminiscent of the shorthand syntax used in the theory section 2 to define applied modal-axioms. When printing of the details of the axiomatic translation is enabled, the software reports the applied axioms as shown in figure 5.5. During preparation for translation, the in-line syntax elements are stripped from the input modal formula.

Figure 5.2 Extensions to the dfg syntax for in-line axiom definitions.

The elements D, B, T, 4, 5, ALT1, 4Kk, 5Kk and ALTKKkk refer to modal-axioms D, B, T, 4, 5, Alt_1, 4^k, 5^k, $Alt_1^{k,k}$ respectively. The default mode of translation is an axiomatic translation without composition. The final letter 'o' refers to axiomatic translation of the

modal-axiom in compositional mode, and the final letter 'c' refers to the classical translation of the modal-axiom (correspondence property).

```
prop_quant_sym ::= axD    | axB     | axT      | ax4     | ax5  | axALT1       |
                   ax4K2  | ax4K3   | ax5K2    | ax5K3   |
               axALT1KK11 | axALT1KK12        | axALT1KK21     | axALT1KK22   |
                     axDc | axBc    | axTc     | ax4c    | ax5c | axALT1c      |
                     ax4K2c | ax4K3c | ax5K2c | ax5K3c |
               axALT1KK11c | axALT1KK12c       | axALT1KK21c    | axALT1KK22c |
                     axDo | axBo    | axTo     | ax4o    | ax5o | axALT1o      |
                     ax4K2o | ax4K3o | ax5K2o | ax5K3o |
               axALT1KK11o | axALT1KK12o       | axALT1KK21o    | axALT1KK22o
```

Figure 5.3 Typical use of the in-line syntax.

The figure shows an extract from a .dfg file using the in-line syntax. The example applies the modal-axioms 5 (with composition), and D (without composition), to each of the sub-formulae. The order in which the modal-axioms appear in the formula is significant, so in the case above the applied modal-axioms are $K5_oD$. The axiom syntax may be interspersed in any format within the terms defined in the formula. Any available letter can be used as the modality indices (depending on the target formulae), but r and s are the usual choices.

```
list_of_special_formulae(axioms,eml).
  prop_formula(
   ax5o(r, axD(r, and(box(r,p),dia(r,not(p)))))
).
```

Figure 5.4 Typical use of the in-line syntax for a multi-modal target formula.

The figure shows an extract from a .dfg file using the in-line syntax. In a multi-modal target formula, the axiom is only applied to sections of the formula with the correct modality index. Axiom 5^3 is applied to segments of the translation derived from `box(s,p),` and modal-axiom D (as correspondence property) is applied to segments of the translation derived from `dia(r,not(p))`.

```
list_of_special_formulae(axioms,eml).
  prop_formula(
   ax5K3(s,axDc(r, and(box(s,p),dia(r,not(p)))))
).
```

Figure 5.5 Typical output for the in-line syntax.

This information is printed by extended-SPASS for the problems in figures 5.3 and 5.4 .

```
[Axiomatic Translation Input] ax5o(r,axD(r,and(box(r,p),dia(r,not(p)))))
  Requested Composition ..... K,5o,D


[Axiomatic Translation Input] ax5K3(s,axDc(r,and(box(s,p),dia(r,not(p)))))
  Requested Composition ..... K,5K[3],Dc
```

5.1.3 Extensions to the standard flags mechanism to provide an axiom definition syntax and for control of axiomatic translation.

Many new flags were introduced into the set of those already available in SPASS. The additions are listed in figure 5.6. A list of all the flags available is created if SPASS is executed with no arguments. There are two input modes accessed via flags. In the default mode (referred to as the 'protected translation mode') the modal-axioms available are restricted to those presented in [1], that is:

K, K4, K5, KT, KB, KD, $KAlt_1$, $K4^2$, $K4^3$, $K5^2$, $K5^3$, $KAlt_1^{1,1}$, $KAlt_1^{1,2}$, $KAlt_1^{2,1}$, $KAlt_1^{2,2}$, KT4, KTB, KDB, KD4, K4B, $K5_oB$, KT5, KT4B, KD4B.

These axioms and axiom combinations have been shown to be sound and complete. If axioms are requested outside this range, an error is raised and the translation halts. The appropriate axiom is selected by setting the value of the flag in the range 1 to 9. The actual value used has no significance (since the order in which axioms are applied is determined by extended-SPASS). An axiomatic translation is requested using the flag `–EMLAxiomLogX=1`, with the correct mode (with or without composition) chosen by the software. A classical translation (correspondence property) is requested using the flag `–EMLClassLogX=1`. The exponent parameters for axioms 4^κ, 5^κ $Alt_1^{\kappa,\kappa}$, $4^\kappa c$, $5^\kappa c$, $Alt_1^{\kappa,\kappa}c$ are set with the values given to `–EMLAxiomLogFac4, -EMLAxiomLogFac5, -EMLAxiomLogFac1, -EMLAxiomLogFac2`, and the corresponding `–EMLClassLogFacX` flags respectively. Hence, the axiom $K5_oB$ is defined by the flags combination `'-EMLAxiom=1 –EMLAxiomLog5=1 –EMLAxiomLogB=1'`,

and the axiom combination $K5^3_c$ is defined by the flags combination

`'-EMLAxiom=1 –EMLClassLog5K=1 –EMLClassLogFac5=3'`.

When printing of the details of the axiomatic translation is enabled, the software reports the applied axioms as shown in figure 5.7.

In a second 'experimental translation mode' *any* combination of axioms is allowed, with the restriction that each axiom appears only once. This mode is accessed by setting the second bit in the flag `–EMLAxiomHidOpt= 901000000` ('Ignore logic flag warnings'; see figure 5.9). *The user must appreciate that there is no guarantee that the translation provided is complete*. The numerical value given to the flag `–EMLAxiomLogX=n` or `–EMLClassLogX=n`, is used to define the order in which the axioms are applied, lowest number applied first. If any values are the same then the arbitrary order ($T > D > 5 > 4 > B > ALT_1 > 5^K > 4^K > ALT_1^{KK} > T_c > D_c > 5_c > 4_c > B_c > ALT_{1c} > 5^K_c > 4^K_c > ALT_1^{KK}{}_c$) is imposed on the applied axioms. The default mode for axiomatic translation is composition. The input that is used to define the modal axiom

combination $K4_c5_oAlt_1^{2,1}$ is illustrated in figure 5.8. The default composition can be overridden using either a global command

`'Do not use composition for any axiom in axiomatic translation'` (`-EMLAxiomHidOpt=900001000`), or more targeted commands like

`'Do not use composition for axiomatic translation of Axiom B'` (`-EMLAxiomOpt=910000000`). A full list of the bit-based flags accessed by `EMLAxiomOpt` and `EMLAxiomHidOpt` is given in figure 5.9. It is often best to use the web-based interface (Parameters Tab and Modal sub-tab) as a calculator for these flag values; see figure 5.29). The user feedback on the modal-axioms requested and the order in which axioms are to be applied is identical to the previous section. When a particular control bit is set in the `EMLAxiomOptions` or `EMLAxiomHidOpt flag`, it is also reported, for example, using the inputs above then the output in figure 5.10 is seen.

Multimodal target formulae are *not* compatible with either of these modes of input, and a multimodal formula will raise a terminating error. A very large number of flags, or very complex bit-based control flags would have been necessary to implement multimodal formulae with this syntax.

These flags can be entered into the .dfg file in the `'list_of_settings(SPASS)'` section using the syntax `'set_flag(flagname, int).'`. This syntax was commonly used to create a standard *baseline* of flag options during testing (see figure 7.1). Such options can be overridden at the command-line with the syntax `'SPASS –flagname=int filename'` at higher priority. The flag-based syntax was hence often used at the command-line during testing: A set of more than 133 template .dfg files was formed, and then each was used as input to extended-SPASS with several dozen single axioms and axiom combinations each applied in turn. An extract from such a shell script, looping over all the pre-formed .dfg files is in figure 7.2. Finally, it is simple to produce JavaScript code that is able to set flags from within the web interface, by formulating a command-line to be submitted to extended-SPASS (see section 5.9). Hence, flags were used to set extended-SPASS options chosen in the Parameters section of the web interface.

Figure 5.6 Extensions to the standard flags mechanism.

The flags added to extended-SPASS, and the range of values accepted, are listed

`-EMLAxiom=1`	Enables Axiomatic Translation Module
`-EMLAxiomLog4=n`	`-EMLClassLog4=n`
`-EMLAxiomLog5=n`	`-EMLClassLog5=n`
`-EMLAxiomLogD=n`	`-EMLClassLogD=n`

```
-EMLAxiomLogT=n                    -EMLClassLogT=n
-EMLAxiomLogB=n                    -EMLClassLogB=n
-EMLAxiomLogAlt1=n                 -EMLClassLogAlt1=n
-EMLAxiomLog4K=n                   -EMLClassLog4K=n
-EMLAxiomLog5K=n                   -EMLClassLog5K=n
-EMLAxiomLogAlt1KK=n               -EMLClassLogAlt1KK=n
-EMLAxiomLogFac1=p                 -EMLClassLogFac1=p
-EMLAxiomLogFac2=p                 -EMLClassLogFac2=p
-EMLAxiomLogFac4=q                 -EMLClassLogFac4=q
-EMLAxiomLogFac5=q                 -EMLClassLogFac5=q
where n is an integer from 0 to 9, and p is an integer 1 to 2,
  and q is an integer 2 to 3
     ------------------------------------------------------------
-EMLAxiomOpt=m                     -EMLAxiomHidOpt=m
where m is an integer from 900000000 to 9999999999
```

Figure 5.7 The information printed by extended-SPASS for sequences of axioms requested.

The examples given are output from the examples $K5_oB$ and $K5K_c^3$.

```
   Requested Composition ..... K,5o,B
   Requested Composition ..... K,5Kc[3]
```

Figure 5.8 Use of the flag-based input syntax in experimental mode.

The flags combination defining the axiom combination $K.4_c.5_o.Alt_1^{2,1}$ is shown, together with the information printed by extended-SPASS, in response to these options.

```
INPUT:    SPASS -EMLAxiom=1 –EMLAxiomLog5=2 –EMLClassLog4=1
             –EMLAxiomLogAlt1KK=3 EMLAxiomLogFac1=2 -EMLAxiomLogFac2=1
             –EMLAxiomHidOpt=901000000 file.dfg

OUTPUT:   Building a General Composition Structure
          Requested Composition ..... K,4c,5o,ALT1KKo[2][1]
```

Figure 5.9 Bit based flags available in extended-SPASS.

The meaning of bits in the flags `EMLAxiomOpt` and EMLAxiomHidOpt is given.

Flag.Bit = Value	Way in which the bit controls the translation process
EMLAxiomOpt.bit1=0/1	Do not use composition for axiomatic translation of Axiom B (on/off)
EMLAxiomOpt.bit2=0/1	Do not use composition for axiomatic translation of Axiom D (on/off)
EMLAxiomOpt.bit3=0/1	Do not use composition for axiomatic translation of Axiom alt_1 (off/on)
EMLAxiomOpt.bit4=0/1	Do not use composition for axiomatic translation of Axiom 4 (off/on)
EMLAxiomOpt.bit5=0/1/2	OverrideLocalGlobal is set: 0=off; 1=Global satisfiability; 2=Local satisfiability
EMLAxiomOpt.bit6=0/1	Do not use composition for axiomatic translation of Axiom 4^K (off/on)
EMLAxiomOpt.bit7=0/1	Do not use composition for axiomatic translation of Axiom 5^K (off/on)
EMLAxiomOpt.bit8=0/1	Do not use composition for axiomatic translation of Axiom $alt1^{KK}$ (off/on
EMLAxiomHidOpt.bit1=0/1	Do not use set precedence (off/on)
EMLAxiomHidOpt.bit2=0/1	Ignore logic flag warnings (activate experimental mode) (off/on)
EMLAxiomHidOpt.bit3=0/1	Exclude optional positive shortcuts in axiomatic translation (off/on)
EMLAxiomHidOpt.bit4=0/1	Do not sort term names (off/on)
EMLAxiomHidOpt.bit5=0/1	Do not use composition for any axiom in axiomatic translation (off/on)
EMLAxiomHidOpt.bit6=0/1	Always Use Substituted Symbol Names (off/on)
EMLAxiomHidOpt.bit7=0/1/2	Do Not Use Print Statements (off/on/full)
EMLAxiomHidOpt.bit8=0/1	Never Use Positive Shortcuts (off/on)

Figure 5.10 Typical output from extended-SPASS informing the user of the bit-based flag settings. This information is printed by extended-SPASS, when the corresponding bit-based flags are set. The examples are taken from the text of section 5.1.3:

```
–EMLAxiomHidOpt=901000000
-EMLAxiomHidOpt=900001000
-EMLAxiomOpt=910000000
```

```
'IgnoreLogicFlagWarnings = ON'
'DoNotUseAxiomBComposition = ON'
'DoNotUseComposition = ON'
```

5.1.4 The set_axiom flag based axiom definition syntax.

The third way in which the modal-axioms that are applied to a formula may be defined is by a new flag-like syntax. This input method may *only* be accessed in the `'list_of_settings(SPASS)'` section of a .dfg file (*not* at the command line). All modal-axioms are unary (take a single modal formula as an argument). The extended syntax takes the form shown in figure 5.11. The comma separated list of modality-index/axiom pairs is of arbitrary length (up to a maximum `MAXDEFINEDAXIOMS = 100`). The order in which modality-index/axiom pairs appear in the list is significant, with the first axiom in the list being applied first during translation. An example of the use of the set_axiom syntax is shown in figure 5.12. These syntaxes are reminiscent of the syntax used in `ml2dfg` (see section 3 and [1]). For definitions of these modal-axioms see figure 2.2.2.

There are subtle matters relating to composition to consider in order to get precisely the translation that is required. The software behaves in the following way, for an axiom combination X_oY_o. (X and Y are any of T, D, B, 4, Alt1, 4K2, 4K3, ALT1KK11, ALTKK12, ALT1KK21, ALT1KK22). The compositional subformulae for X (if any) are defined and added to the instantiation set *just before* Y is translated. In contrast although Y_o is requested, the axiomatic translation delivered is Y, because composition makes no sense (in terms of ensuring completeness) for the last axiom is a series. That is, while X_oY_o requested, only X_oY is delivered, and likewise when X_o is requested, then only is X delivered. Clearly, the software is making a sensible adjustment, and the user should have more sensibly requested the axioms X_oY and X in these cases. However, the user may wish to assess the timing implications of the additional compositional (non-essential terms) for axiom X. That is to request X_o and get all the X_o terms in the translation. The solution is to request X_oNone or X_oX, as appropriate. When requesting a mixed axiomatic and classical translation, then the classical axiom will not trigger use of compositional

axiomatic terms. Hence, when X_oY_c requested, then only XY_c is delivered In uni-modal logics there is one more consideration. For axioms 5, 5^K, SR, G, DEN, TR, M, W, B^K, DBBB the compositional terms are (or sometimes, may be) required to make the translation of the axiom *itself* complete. In these cases requesting Z_o delivers the translation Z_o (which is 'Z_oZ'). For multi-modal target formulae some additional care is needed with axioms T, D, B, 4, Alt1, 4K2, 4K3, ALT1KK11, ALTKK12, ALT1KK21, ALT1KK22. If the user wishes to apply the axiom X (without composition) to modalities r and s, the intuitive assumption is that she can request either ([r,X], [s,X]) or ([r,Xo], [s,Xo]). However, in the later case, the translation delivered will be ([r,Xo], [s,X]), since an axiom definition *does* follow [s,Xo] in the list, but *does not* follow [s,Xo] in the list. The user is of course recommended to request the correct (sensible) translation, and not to rely on the software to make these corrections.

Figure 5.11 Format of the set_axiom syntax.

Three different possible formats for the set_axiom syntax are shown.

- r is a modality index, which must appear in the formulae submitted in the logic section. Multi-modal formulae are supported by this syntax. Bimodal-axioms are also supported, in the syntax, by use of different r and s modality indices. (Note, any available letter may be used).
- X represents the unimodal-axiom to be applied to the modal formula during translation.
- Y represents a bimodal-axiom applied to the modal formula during translation.

```
set_axiom([r,X]).
set_axiom([r,X],[r,X]).
set_axiom([r,s,Y],[r,X]).
```

Allowed values for the axioms X and Y are given in (i) to (iii).

(i) modal-axioms in axiomatic translation without composition

```
X ={T, D, B, 4, 5, Alt1, 4K2, 4K3, 5K2, 5K3, ALT1KK11, ALTKK12,
ALT1KK21, ALT1KK22, SR, G, DEN, TR, M, W, B2, B3, DBBB, None }
Y = {CR, CR2, CR3}
```

(ii) modal-axioms in axiomatic translation with composition

```
X = {To, Do, Bo, 4o, 5o, Alt1o, 4K2o, 4K3o, 5K2o, 5K3o, ALT1KK11o,
ALT1KK12o, ALT1KK21o, ALT1KK22o, SRo, Go, DENo, TRo, Mo, Wo, B2o,
B3o, DBBBo }
Y = {CRo, CR2o, CR3o}
```

(iii) modal-axioms in classical translation (correspondence property)

```
X={Tc, Dc, Bc, 4c, 5c, Alt1c, 4K2c, 4K3c, 5K2c, 5K3c, ALT1KK11c,
ALT1KK12c, ALT1KK21c, ALT1KK22c}
```

Figure 5.12 An example of the use of the set_axiom syntax.

A fragment of a .dfg file is shown, defining the axiom combination $K.B_o.5_c.T$

```
list_of_settings(SPASS).
{*
  set_flag(EMLAxiom,1).
  set_axiomflag([r,Bo],[r,5c],[r,T]).
*}
```

5.1.5 Other ways to define correspondence properties.

The definition of correspondence properties for axioms need not involve coding within extended-SPASS. In the examples above correspondence properties have been provided simply as a convenience. In an alternative syntax, a mechanism is provided to except a correspondence formula (within a .dfg file) from axiomatic translation and to link the binary accessibility relation (for example, R) appearing in the correspondence property to the accessibility relation produced during certain axiomatic translations. An example of the syntax appears in figure 5.13. The syntax `noAxiom(`*`formula`*`)` protects an enclosed formula from axiomatic translation. The syntax `translpairs[(modality index, accessibility relation)]` defines a modality index (r) that can give rise to an accessibility relation (R_r), and is linked to the same accessibility relation (R_r) within the protected correspondence property formula. An arbitrary number of translpairs can be defined in a comma separated list. The definition of `translpairs` must appear after the predicates definitions within the `list_of_symbols` section. Typical output seen by the user is shown in figure 5.14. Note, `translpairs` was implemented previously in MSPASS for a related purpose (see section 1.6).

As described in section 1.3.2 (and figure 1.5), it is possible to use the Scott-Lemmon algorithm to define the correspondence properties for modal formulae of the type $\Diamond^h\Box^i p \rightarrow \Box^j\Diamond^k p$ (where h, i, j, k, are integers 0 or greater). This algorithm has been implemented in extended-SPASS. The syntax encloses a formula of the required format inside `slf(`*`modality index, formula`*`)` as shown in figure 5.15. The correspondence property is added to the list of formulae output to SPASS, and the user is informed of the interpretation that has been made. Typical output is shown in figure 5.16.

When `noAxiom()` and `slf()` syntaxes are used, it is recommended that the relevant terms appear as separate members of the `Axioms List` in extended-SPASS version 1.1.0, and in separate members of the `Conjectures List` in extended-SPASS version 1.1.2.

Figure 5.13 Use of the `noAxiom` syntax. A fragment from a .dfg file illustrating the use of the `noAxiom` syntax is given.

```
list_of_symbols.
    predicates[(Rr,2),(r,0),(p,0)].
    translpairs[(r,Rr)].
end_of_list.

list_of_special_formulae(axioms,eml).
 prop_formula(noAxiom(
    forall([x,y],equiv(Rr(x,y),equal(x,y))) % Axiom TR:   R(x,y)<->(x=y)
 )).
 prop_formula(
    and(box(r,p),dia(r,true))
 ).
end_of_list.
```

Figure 5.14 Typical output produced by the `noAxiom` syntax.

This information is printed by extended-SPASS when the .dfg file in figure 5.13 is run.

```
 [Axiomatic Translation Input]
 and(and(box(r,p),dia(r,true)),NoAxiom(forall([U,V],equiv(Rr(U,V),equal(U,V)))))
 Formula Exempt from Translation: forall([U,V],equiv(Rr(U,V),equal(U,V)))
```

Figure 5.15 Typical use of the `slf` syntax.

A fragment of a .dfg file defining the `slf` syntax for the formula ◇□p→□◇p is shown.

```
list_of_special_formulae(axioms,eml).
    prop_formula(
          not(implies(not(box(r,not(box(r,p)))),box(r,p)))
    ).
    prop_formula(slf(r,implies(dia(r,box(r,p)),box(r,dia(r,p))))).
end_of_list.
```

Figure 5.16 Typical output for the `slf` syntax.

This information is printed by extended-SPASS when the .dfg file sketched in figure 5.15 is run.

```
Scott-Lemmon
formula:forall([V,W,U],implies(and(Rr(U,V),Rr(U,W)),exists([X],and(Rr(V,X),
Rr(W,X)))))
<>^1[]^1(p)->[]^1<>^1(p).
```

5.2 Reorganize axioms and conjectures of input problem: (Task 5)

SPASS allows two kinds of modal formula to be entered, conjectures (C_n) and axioms (A_n). An arbitrary number of members of both the `AxiomList` and `ConjectureList` are supported. The software rearranges the input by concatenating all members of the both lists (by conjugation), but each member of the `ConjectureList` is negated before it is added. The user is informed that each member of each LIST has been

rolled-up into a single `AxiomList` term (which is then $(A_1 \wedge A_2 \wedge \ldots A_n) \wedge (\neg C_1 \wedge \neg C_2 \wedge \ldots \neg C_n)$). The result of the axiomatic translation using this composite term then becomes the first member of the `AxiomList` at output (for entry in the SPASS resolution prover), with the remaining elements in both LISTs being lost. Why is this necessary? It is clear from section 2 that for axiomatic translation of a modal logic problem, *all* the 'problem' information is required at the outset in order to define the instantiation set. The complications of running through the process with a series of partial inputs would be very great indeed. Typical output that informs the user of the transformations that have been made is shown in figure 5.17.

In version 1.1.2 of extended-SPASS, the makeup of the input problem is also used to set the flag `DoNotUseLocalSatisfiability`. If only axioms are present then `DoNotUseLocalSatisfiability` is active, defining use of the global satisfiability mode. If conjectures are present (with or without axioms) then the local satisfiability mode is defined (`DoNotUseLocalSatisfiability` is inactive). The choices are made based upon the length of the `Conjectures list` and `Axioms list`, and the user is informed of the choices made. (The user can override the default choice between the local or the global satisfiability translation that was just described, by setting bit 5 in the flag `EMLAxiomOpt`; see table 5.9). (In version 1.1.0 of extended-SPASS, only the local satisfiability translation was implemented).

Figure 5.17 Typical output indicating the reorganization of axioms and conjectures. This information is printed by extended-SPASS to inform the user of the rollup of members of the axioms and conjectures lists.

```
Input Axiom Term 0:  dia(r,box(r,p))
Input Conjecture Term 0:   not(not(dia(r,dia(r,not(p)))))
[Axiomatic Translation Input]
and(dia(r,box(r,p)),not(not(dia(r,dia(r,not(p))))))
```

5.3 Standardization of input (Task 6).

The input target formula is reduced (by modal and propositional algebra) to a target formula involving only 'false' ($\perp$), 'not' ($\neg$), 'and' ($\wedge$), and 'box' ($\Box$). Applying the formulae seen in figure 5.19 eliminates other symbols. The order in which these transformations are preformed is critical to success. Symbols are eliminated in favor of those lower down the list. The purpose of standardization is to produce a target formula with no trivial redundancy, in a simple and regular format that is easy to analyze in subsequent translation steps. The user is informed of the transformation that has taken

place (see for example figure 5.18).

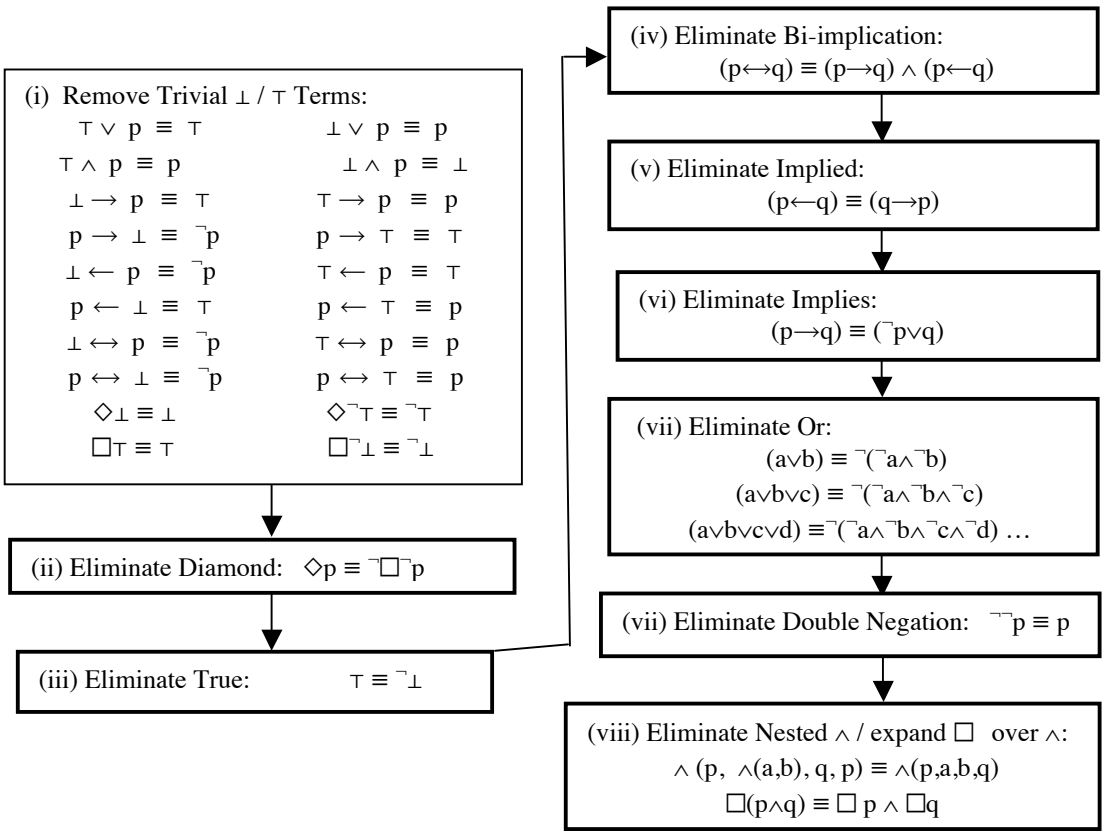
Figure 5.18 Typical output printed after standardization of a target formula.

This information is by extended-SPASS after standardization of a target formula. The standardization is seen by comparing the two lines in the output title `'Axiomatic Translation Input'` and `'Axiomatic Translation Standardized'`.

```
[Axiomatic Translation Input]
    not(implies(not(box(r,not(box(r,p)))),box(r,p)))
[Axiomatic Translation Standardized]
    and(not(box(r,not(box(r,p)))),not(box(r,p)))
```

Figure 5.19 Tasks: Standardize target formula.

The steps required to standardize the format of an input target formula are shown. The order in which the transformations take place is critical to success.



5.4 Perform axiomatic translation of input problem (Task 7).

Once a problem has been setup, using the methods described above, the user is informed of the details of the translation. The principle use of this output will be in education of the occasional user. In batch mode calculations, the output may be suspended using `-EMLAxiomHidOpt=900000010` (`'Do Not Use Print Statements'`).

Nevertheless the overheads involved in printing of information is generally small (a few milliseconds). The format of this output is illustrated in figure 5.20.

Figure 5.20 Typical output describing the details of axiomatic translation in logic K.
First, the user gets feedback on the instantiation set under the title `'[Axiomatic Translation Def Units]'` with members of the instantiation set listed as `DEFUNITS 1 to n`. Each member of the instantiation set is then translated (under the title `'Translation of DEF = …'`; Def(ψ) is defined in formulae 2.2 and 2.3), with `1stTERM` representing $\forall x(Q_{\psi}(x) \rightarrow \pi(\psi, x))$ (from formula 2.2), `2ndTERM` representing $\forall x(Q_{\psi}(x) \rightarrow \neg Q_{\sim\psi}(x))$, `2ndTERMSC` representing $\forall x(Q_{\psi}(x) \leftarrow \neg Q_{\sim\psi}(x))$ (the positive shortcut term in formula 2.2), and `3rdTERM` representing $\forall x(Q_{\sim\psi}(x) \rightarrow \pi(\sim\psi, x))$. Finally the conjugation of all the translated terms is presented under the title `'[Axiomatic Translation: Logic K]'.`

```
[Axiomatic Translation Standardized] and(not(box(r,not(box(r,p)))),not(box(r,p)))
[Axiomatic Translation Def Units]
DEFUNITS 1 :box(r,p)
DEFUNITS 2 :box(r,not(box(r,p)))
DEFUNITS 3 :p
DEFUNITS 4 :and(not(box(r,not(box(r,p)))),not(box(r,p)))
Translation of DEF = box(r,p)
2ndTERM = forall([U],implies(Qbox_r_p(U),not(Qnot_box_r_p(U))))
2ndTERMSC = forall([U],implied(Qbox_r_p(U),not(Qnot_box_r_p(U))))
1stTERM = forall([U],implies(Qbox_r_p(U),forall([V],implies(Rr(U,V),Qp(V)))))
3rdTERM = forall([U],implies(Qnot_box_r_p(U),exists([V],and(Rr(U,V),Qnot_p(V)))))
Translation of DEF = box(r,not(box(r,p)))
2ndTERM = forall([U],implies(Qbox_r_not_box_r_p(U),not(Qnot_box_r_not_box_r_p(U))))
2ndTERMSC = forall([U],implied(Qbox_r_not_box_r_p(U),not(Qnot_box_r_not_box_r_p(U))))
1stTERM = forall([U],implies(Qbox_r_not_box_r_p(U),forall([V],implies(Rr(U,V),Qnot_box_r_p(V)))))
3rdTERM = forall([U],implies(Qnot_box_r_not_box_r_p(U),exists([V],and(Rr(U,V),Qbox_r_p(V)))))
Translation of DEF = and(not(box(r,not(box(r,p)))),not(box(r,p)))
2ndTERM =
forall([U],implies(Qand_not_box_r_not_box_r_p_not_box_r_p(U),not(Qnot_and_not_box_r_not_box_r_p_not_box_r_p(U))))
2ndTERMSC =
forall([U],implied(Qand_not_box_r_not_box_r_p_not_box_r_p(U),not(Qnot_and_not_box_r_not_box_r_p_not_box_r_p(U))))
1stTERM =
forall([U],implies(Qand_not_box_r_not_box_r_p_not_box_r_p(U),and(Qnot_box_r_p(U),Qnot_box_r_not_box_r_p(U))))
3rdTERM =
forall([U],implies(Qnot_and_not_box_r_not_box_r_p_not_box_r_p(U),or(Qbox_r_p(U),Qbox_r_not_box_r_p(U))))
Translation of DEF = p
2ndTERM = forall([U],implies(Qp(U),not(Qnot_p(U))))
2ndTERMSC = forall([U],implied(Qp(U),not(Qnot_p(U))))
[Axiomatic Translation: Logic K]
and(exists([U],Qand_not_box_r_not_box_r_p_not_box_r_p(U)),forall([U],implies(Qnot_box_r_p(U),exists([V],and(Rr(U,V),Qnot_p(V))))),forall([U],implies(Qbox_r_p(U),forall([V],implies(Rr(U,V),Qp(V))))),forall([U],implied(Qbox_r_p(U),not(Qnot_box_r_p(U)))),forall([U],implies(Qbox_r_p(U),not(Qnot_box_r_p(U)))),forall([U],implies(Qnot_box_r_not_box_r_p(U),exists([V],and(Rr(U,V),Qbox_r_p(V))))),forall([U],implies(Qbox_r_not_box_r_p(U),forall([V],implies(Rr(U,V),Qnot_box_r_p(V))))),forall([U],implied(Qbox_r_not_box_r_p(U),not(Qnot_box_r_not_box_r_p(U)))),forall([U],implies(Qbox_r_not_box_r_p(U),not(Qnot_box_r_not_box_r_p(U)))),forall([U],implies(Qnot_and_not_box_r_not_box_r_p_not_box_r_p(U),or(Qbox_r_p(U),Qbox_r_not_box_r_p(U)))),forall([U],implies(Qand_not_box_r_not_box_r_p_not_box_r_p(U),and(Qnot_box_r_p(U),Qnot_box_r_not_box_r_p(U))))),forall([U],implied(Qand_not_box_r_not_box_r_p_not_box_r_p(U),not(Qnot_and_not_box_r_not_box_r_p_not_box_r_p(U)))),forall([U],implies(Qand_not_box_r_not_box_r_p_not_box_r_p(U),not(Qnot_and_not_box_r_not_box_r_p_not_box_r_p(U)))),forall([U],implied(Qp(U),not(Qnot_p(U)))),forall([U],implies(Qp(U),not(Qnot_p(U)))))
```

5.5 Incorporate modal axiom(s) into the translation of the input problem (Task 8).

Once the basic translation of the modal formula (in Axiom K) has been made, the formula is translated with respect to the defined modal axioms.

In some cases the translation is very simple, for example with axiom T. Under a title recording the axiom name, the member of the instantiation set contributing to the

translation, and the modality index being considered, the newly translated `TERM` is displayed. A typical example is shown in figure 5.21. In other cases, new subformulae arise in the incorporation of the axiom in the translation, and so must be Defined, for example axiom D (where a new term $Q_{\Box\neg p}$ must be defined for translation of $\Box p$). Figure 5.22 shows an example. In figure 5.22a, `DEFUNIT 1` gives rise to the new `TERM box(r,not(p))`, and in figure 5.22b `DEFUNIT 2` gives rise to the new `TERM box(r,box(r,p))`. (`DEFUNITS 3` and `4` make no contribution). In other respects the pattern of the output is as already described in figure 5.21.

When composition is not used, the process is this simple for all axioms, and conforms to the pattern already seen in figure 5.21 and 5.22. If composition is needed to perform the requested translation, then new subformulae accumulate in the instantiation set. These subformulae need to be Defined as seen above, and the new `TERMs` are listed under the title `'[Summary Axiomatic DEFUNITS for X Composition were]'`, where X is the modal axiom in question. An example of the output for composition of axiom 5 is seen in figure 5.23.

In the case of a classical axiom (correspondence property) the information given is very simple, just the axiom name, the modality index (r) corresponding to the accessibility relation (Rr), and the new `TERM` that is introduced. An example is seen figure 5.24.

Figure 5.21 Typical output describing the incorporation of modal axioms in the axiomatic translation.

A typical translation in modal axiom T is illustrated by an extract from the output of extracted-SPASS. The current instantiation set is $\{\Box p, \Box\neg\Box p\}$.

```
[Axiom T]:[r in term:box(r,p)].
forall([U],or(not(Qbox_r_p(U)),Qp(U)))
[Axiom T]:[r in term:box(r,not(box(r,p)))].
forall([U],or(not(Qbox_r_not_box_r_p(U)),Qnot_box_r_p(U)))
```

Figure 5.22 Typical output describing the incorporation of modal axioms in the axiomatic translation

Extracts from the output of extended-SPASS is given. The different subsections are described in the text of section 5.5. The output illustrates the new subformulae that need to the Defined as a result of the instantiation of axiom D for the instantiation set members $\{\Box p, \Box\neg\Box p\}$. $\Box(r,p)$ (`DEFUNIT 1`) is dealt with as shown in figure 5.22a, and $\Box(r,\Box(r,p))$ (`DEFUNIT 2`) is dealt with as shown in figure 5.22b. Only part of the output is listed. The remainder follows the format already seen in figure 5.21.

```
      DEFUNITS 1 :box(r,p)
      DEFUNITS 2 :box(r,not(box(r,p)))
      DEFUNITS 3 :p
      DEFUNITS 4 :and(not(box(r,not(box(r,p)))),not(box(r,p)))
5.22(a)      For Axiom D:
      Translation of DEF = box(r,not(p))
      2ndTERM = forall([U],implies(Qbox_r_not_p(U),not(Qnot_box_r_not_p(U))))
      2ndTERMSC = forall([U],implied(Qbox_r_not_p(U),not(Qnot_box_r_not_p(U))))
      1stTERM = forall([U],implies(Qbox_r_not_p(U),forall([V],implies(Rr(U,V),Qnot_p(V)))))
      3rdTERM = forall([U],implies(Qnot_box_r_not_p(U),exists([V],and(Rr(U,V),Qp(V)))))
      [Axiom D]:[r in term:box(r,p)].
      and(and(forall([U],implies(Qnot_box_r_not_p(U),exists([V],and(Rr(U,V),Qp(V))))),forall([U],
       implies(Qbox_r_not_p(U),forall([V],implies(Rr(U,V),Qnot_p(V))))),forall([U],
       implied(Qbox_r_not_p(U),not(Qnot_box_r_not_p(U)))),forall([U],implies(Qbox_r_not_p(U),
       not(Qnot_box_r_not_p(U))))),forall([U],or(not(Qbox_r_p(U)),Qnot_box_r_not_p(U))))
5.22(b)      For Axiom D:
      Translation of DEF = box(r,box(r,p))
      2ndTERM = forall([U],implies(Qbox_r_box_r_p(U),not(Qnot_box_r_box_r_p(U))))
      2ndTERMSC = forall([U],implied(Qbox_r_box_r_p(U),not(Qnot_box_r_box_r_p(U))))
      1stTERM = forall([U],implies(Qbox_r_box_r_p(U),forall([V],implies(Rr(U,V),Qbox_r_p(V)))))
      3rdTERM = forall([U],implies(Qnot_box_r_box_r_p(U),exists([V],and(Rr(U,V),Qnot_box_r_p(V)))))
      [Axiom D]:[r in term:box(r,not(box(r,p)))].
      and(and(forall([U],implies(Qnot_box_r_box_r_p(U),exists([V],and(Rr(U,V),Qnot_box_r_p(V))))),
       forall([U],implies(Qbox_r_box_r_p(U),forall([V],implies(Rr(U,V),Qbox_r_p(V))))),forall([U],
       implied(Qbox_r_box_r_p(U),not(Qnot_box_r_box_r_p(U)))),forall([U],implies(Qbox_r_box_r_p(U),
      not(Qnot_box_r_box_r_p(U))))),forall([U],or(not(Qbox_r_not_box_r_p(U)),Qnot_box_r_box_r_p(U)))
)
```

Figure 5.23 **Typical output describing the incorporation of modal axioms in the axiomatic translation with composition**.

The example seen is an extract from the output of extended-SPASS relating to axiom 5_o for the instantiation set members $\{\Box p, \Box\neg\Box p\}$. New subformulae are introduced in the form $\{\Box\neg\Box p, \Box\neg\Box\neg\Box p\}$. Clearly $\Box\neg\Box p$ has been encountered before in the translation, and (as an optimization) need not be re-Defined.

```
Translation of DEF = box(r,not(box(r,not(box(r,p)))))
2ndTERM =
forall([U],implies(Qbox_r_not_box_r_not_box_r_p(U),not(Qnot_box_r_not_box_r_not_box_r_p(U))))
2ndTERMSC =
forall([U],implied(Qbox_r_not_box_r_not_box_r_p(U),not(Qnot_box_r_not_box_r_not_box_r_p(U))))
1stTERM =
forall([U],implies(Qbox_r_not_box_r_not_box_r_p(U),forall([V],implies(Rr(U,V),Qnot_box_r_not_box_r_p
(V)))))
3rdTERM =
forall([U],implies(Qnot_box_r_not_box_r_not_box_r_p(U),exists([V],and(Rr(U,V),Qbox_r_not_box_r_p(V))
)))
[Summary Axiomatic DEFUNITS for FIVE Composition were] :
1 box(r,not(box(r,not(box(r,p)))))
[Axiom 5]:[r in term:box(r,p)].
forall([U],or(Qbox_r_p(U),forall([V],or(not(Rr(U,V)),not(Qbox_r_p(V))))))
[Axiom 5]:[r in term:box(r,not(box(r,p)))].
forall([U],or(Qbox_r_not_box_r_p(U),forall([V],or(not(Rr(U,V)),not(Qbox_r_not_box_r_p(V))))))
[Axiom 5]:[r in term:box(r,not(box(r,not(box(r,p))))))].
forall([U],or(Qbox_r_not_box_r_not_box_r_p(U),forall([V],or(not(Rr(U,V)),not(Qbox_r_not_box_r_not_bo
x_r_p(V))))))
```

Figure 5.24 **Typical output describing the incorporation of modal axioms as correspondence properties**.

A fragment of the output of extended-SPASS is shown, relating to the incorporation of axiom 5_c.

```
[Class.Rel.Translation 5].[r].
   forall([U,V,W],implies(and(Rr(U,V),Rr(U,W)),Rr(V,W)))
```

5.6 Consider each axiom in strict order (Task 8c).

When more than one modal-axiom is requested, the information printed shows the new `TERMs` created by the encoding of each modal-axiom, the new `TERMs` that need to be Defined (if any), and the new `TERMs` that are added to the instantiation set (if any), just as described above. The only difference is that information is printed for each modal-axiom in turn, in the order in which the modal-axioms have been requested. An example is not necessary.

5.7 Predicate names and setting the Precedence (Task 9).

The new predicate symbols that are created during the axiomatic translation process are stored temporarily in a cache that is local to the eml-module. These symbols are printed in a list for the information of the user. An example is shown in figure 5.25. The index simply refers to the (arbitrary) position in the local cache, and reflects the order in which terms are created during internal functioning of extended-SPASS. In some cases the name of a symbol would exceed the maximum space allocated within the SPASS resolution prover (`symbol__SYMBOLMAXLEN 64`), and so the predicate symbol is allocated a unique name Q_{int}, where 'int' is an integer corresponding to the position within the local cache. Since the user may wish to follow the resolution process, the full long name that would otherwise have been allocated is printed with the list. In some cases, it is desirable to make the names of all new predicates fit a standard format. In this case all predicates are renamed according to the position in the local cache, and adopt the format Q_{int}. This option is requested by setting `-EMLAxiomHidOpt=900000100` (`'Always Use Substituted Symbol Names'`; see figure 5.9).

The SPASS resolution prover is capable of deciding upon the precedence for the newly defined predicate terms. (This default precedence can be requested if the mode `'Do not use Set Precedence'` is on, and then the arbitrary order in which predicates are defined internally is simply used to set precedence). However, in many cases, this default precedence is far from optimal. Instead the eml-module automatically sets the precedence according to these rules shown below : (ordered by priority, the highest priority at the top)

- Accessibility relation predicates (R) > other predicate symbols (Q)
- The primary ordering of Q–predicates is based upon the number of times that the predicate appears in the `'FINAL TERM:'` (see section 5.8). The least frequently occurring symbol is assigned the highest precedence. However, duplication in frequency means that this is only a partial ordering.
- The terms are further ordered (at lower priority) so that:

- Q-predicates containing 'and' symbols > Q-predicates without 'and'. (The later is a sub-formula of the former).
- Deep Q-predicates > shallow Q-predicates (such that, $Q_{\Box\Box\Box_p} > Q_{\Box\Box_p}$).
- Qp > Qq, by simple string comparison on p and q.
- $Q_{\neg TERM} > Q_{TERM}$.

The precedence which is passed to the SPASS resolution prover is printed under the title `'Re-Organized Precedence'`, and the ordering as described above is reported as `'Using Inverse Popularity + Subformulae Ordering'.` An example is seen in figure 5.26. The SPASS resolution prover is directed to honor this precedence whether or not it is in `AUTO` mode. It is possible to allow the SPASS resolution prover itself to set the precedence using the option `-EMLAxiomHidOpt=910000000` (`'DoNotUseSetPrecedence = ON'`). `'No Re-organized precedence'` is reported and the user is informed that `'Precedence reverting to SPASS control'`. In the case that only classical modal axioms are used (correspondence properties) then SPASS is allowed to set the precedence.

Precedence can also be set by the user in the `list_of_settings(SPASS)` section using the syntax `set_precedence(a,b,c,f,F)` (see [17]). The axiomatic translation module honors these settings. There is an undocumented way to influence the precendence used by SPASS. The (unnecessary) appearance of a symbol needed by the axiomatic translation module (for example, Rr) in the `list_of_symbols/predicates` section of the .dfg file can influence the default precedence set by SPASS alone (in the `'Do not use Set Precedence = on'` mode). The symbol is promoted to a higher level in the precedence list.

Figure 5.25 Typical content of the symbol cache.

This is an extract from typical information printed by extended-SPASS, providing a list of contents of the axiom cache in the eml-module. Some lines have been omitted for brevity. Note that some symbol names exceed the maximum length allowed in SPASS and need to be abbreviated (as described in section 5.7). The format is "Index: Symbol Name".

```
 0: Q0 ... renamed from ...
Qand_not_box_r_not_box_r_not_box_r_not_box_r_not_box_r_not_box_r_not_box_r_not
_box_r_not_box_r_not_box_r_p_not_q
 1: Qbox_r_p
 2: Qnot_box_r_p
 3: Qp
 4: Rr
 5: Qnot_p
 6: Qbox_r_not_box_r_p
 7: Qnot_box_r_not_box_r_p
 8: Qbox_r_not_box_r_not_box_r_p
…
 14: Qbox_r_not_box_r_not_box_r_not_box_r_not_box_r_not_box_r_p
 15: Qnot_box_r_not_box_r_not_box_r_not_box_r_not_box_r_not_box_r_p
```

```
 16: Q16 ... renamed from ...
   Qbox_r_not_box_r_not_box_r_not_box_r_not_box_r_not_box_r_not_box_r_p
 17: Q17 ... renamed from ...
   Qnot_box_r_not_box_r_not_box_r_not_box_r_not_box_r_not_box_r_not_box_r_p
 18: Q18 ... renamed from ...

Qbox_r_not_box_r_not_box_r_not_box_r_not_box_r_not_box_r_not_box_r_p
... etc
```

Figure 5.26 Typical output indicating the precedence of symbols from the symbol cache.

An extract from the output of extended-SPASS is shown, relating to the precedence of new predicate symbols introduced into the axiomatic translation (and those already present in the extended-SPASS). The names of the newly formed predicates are printed as already seen in figure 5.25.

```
Re-Organized Precedence = equal > true > false > div > id > R > r > p > q >
Rr > Qnot_and_not_box_r_not_box_r_p_not_box_r_p > Qnot_p > Qp >
Qand_not_box_r_not_box_r_p_not_box_r_p > Qnot_box_r_not_box_r_p >
Qbox_r_not_box_r_p > Qnot_box_r_p > Qbox_r_p
```

5.8 Pass collected terms from tasks 7 & 8 to SPASS (Task 9).

The `TERM` which is passed to the SPASS resolution prover is printed in full under the title `'FINAL TERM:'`. It represents a conjugation over all the sub-`TERM`s accumulated during the translation of all the members of the instantiation set over all the requested axioms. The format in which the `TERM` is printed is suitable for cut-and-paste into a manually constructed .dfg file for further manipulation. An example is not necessary.

5.9 The user interface.

The prototype user interface can be seen at `http://www2.cs.man.ac.uk/~smithk/KJSpass/main.html`.

The opening screen of the user interface is shown in figure 5.27. Key features are numbered, and correspond to the listed explanation. Other screens are illustrated in a similar way in figure 5.28 and 5.29. All of the features described below are illustrated.

A typical workflow, is to open the interface, to choose a template .dfg file, for example from the `Sc tab/Misc sub-tab`, to *copy* this into the editor in the `Main/Script` page, and then to edit this file according to the current purpose. Parameters can then be chosen from the `Parameters tab`, and *set* into the `Command Line` box of the `Main/Script` page. The input problem with the chosen options is then executed (*Go*) and the results are displayed in the `Main/Results` page. If there are

syntax errors, then these will be reported with respect to a line number, and can be identified in the lower pane of the `Main/Results` page, where a copy of the input problem is seen with associated line numbers. If the .dfg script executes correctly, then the output can be examined in the `Main/Results` page. If the user wishes to store a particular calculation, all of the script, command line options, and execution output can be saved in the `History` mechanism by pressing the `+-tab`. (An unlimited number of calculations can be stored in the history mechanism, and are always available for inspection or to be copied back into the `Main/Script` editor). All the information is stored within the user's browser, except for problems submitted to extended-SPASS, which are automatically submitted to and retrieved from a central server.

The color scheme shows currently active segments of the interface in light-blue. The interface changes, in terms of the color scheme, according to the currently active mode. Each mode is associated with a distinct activity, named in the upper (principal) tabs and left hand-side (sub) tabs. Buttons that change the currently active mode are incorporated into these left and top edges of the main pane, and those that are currently available to be pressed (to effect a change) are colored dark-blue. Buttons associated with functions are of the traditional type. In some cases, functions are not sensible from the current context (or have not yet been implemented because the project did not require this functionality), and in these cases an error is raised in a dialogue box.

Figure 5.27 Key features of the Main Editor of the Web interface. A screen capture of opening screen of the web-based interface is seen. The annotations are described below.

1. The **`Main` window** with editable (cut/paste/copy) text. The content is used as the source `.dfg` file when extended-SPASS is executed.
2. **`Command Line`** flags may be entered into this text box.
3. Click the borrowed **SPASS logo** to open `http://spass.mpi-sb.mpg.de/`.
4. **More info** via a link to `http://project.kjsmith.net/`
5. Click the **`Main` tab** to reach the `Main` screen (It is colored light blue, indicating that it is currently active).
6. Click the **`Parameters` tab** to enter the `Parameters` Screen.
7. Click the **+ tab** to add the contents of the `Main` screens (both `Script` and `Results`) to the next slot in `History` list.
8. Click the **Sc tab** to enter `Sample Scripts` screen.
9. Click the **History tab** to enter `History` Screen.
10. Click the **Help tab** to enter `Help` Screen.
11. Click the **`Script` tab** to reach this Script view of the `Main` screen.
12. Click the **`Results` tab** to enter the `Results` view of the `Main` Screen.
13. Click the **`Clear button`** to clear the contents of 1.
14. Click the **`Go Button`** to execute SPASS with .dfg file from 1 and the flags from 2.

15 and 16. Not currently used (available for additional features).

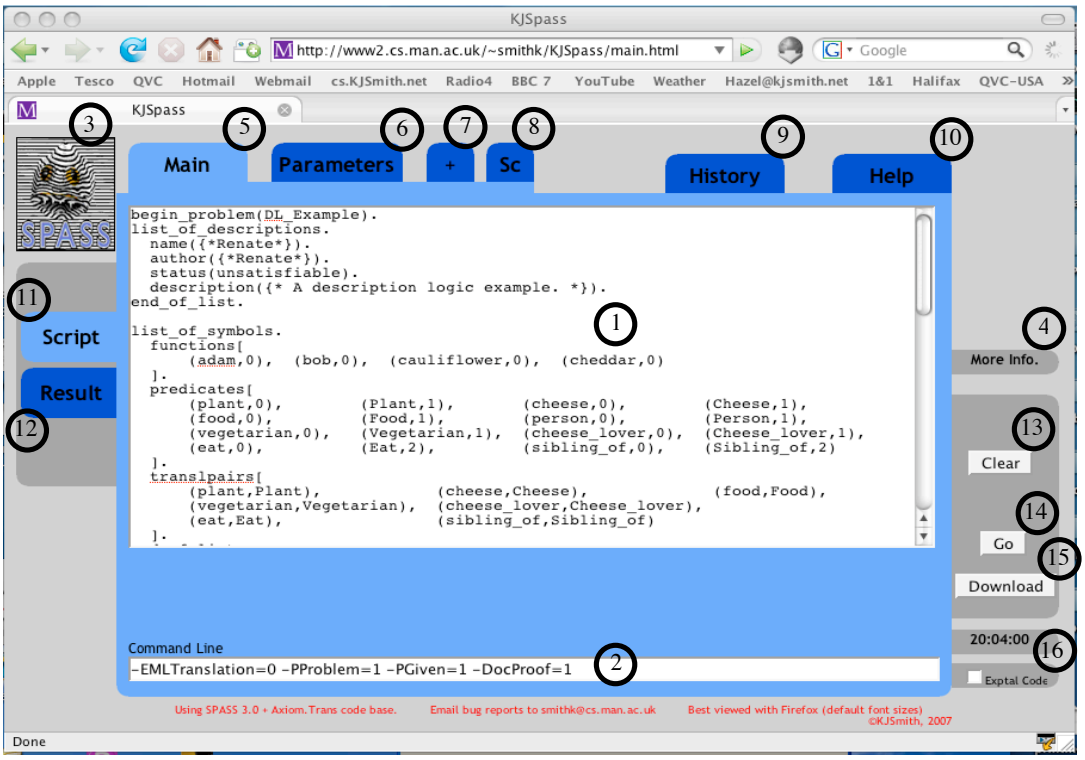


Figure 5.28 Details of the user interface.

The figure illustrates the principal interface components in a series of screen captures taken during a user session. The roman numerals refer to the snapshots taken at different points during the user session. The annotations are described below.

(i) **Results view of the Main Screen.**

17. Click the Results side tab from Main/Script to get here.
18. Contains the results of execution of SPASS with a list of the flags used.
19. Contains the dfg file (as executed) from 1 with line numbers.
    (This is useful during debugging the dfg syntax).
20. Clear 18 and 19.
4. Click here to open `http://spass.mpi-sb.mpg.de/`

(ii) Tooltip for Sample Scripts (Sc) tab.

24. A mouseover event raises this tooltip from the Sc tab.

(iii) Sample Scripts (Sc) Screen.

21. The sample script that is chosen in fig (iv).
22. Click here to clear 21.
23. Click here to copy from 21 to 1, and switch to the Main/Script Screen (by default, or the Main/Results screen if this was previously the active view).
4. Click here to open `/mspass_opInfo/mspass/contents.html` on the local server, that contains many 100's of example files.

(iv) A sample script is chosen from one of the three pull-down menus.

25. A mouseover event on the Misc menu item.
26. Click here to choose a particular script from the menu, and place in 21 (see (iii)).

(v) Tooltip for + (add history) tab.

27. A mouseover event raises a tooltip from the +-tab indicating add history.

(vi) Information popup after pressing the + (add history) tab.

28. Click the + tab (from the Main Screen) to record information from 1, 2, 18, and 19 in the next slot in the History mechanism. A popup appears indicating the slot used.

(vii) Parameters screen showing the Modal tab as the currently active sub-screen.

29. Click on the Standard tab to change any regular SPASS flag parameter.
    The default values are chosen automatically in the first instance.
30. Not currently used (available for additional features).
31. Clicking the Modal Parameters tab opens this screen (also shown in figure 5.29).
32. Click to Reset the parameters in 29 and 31 to the default values.
33. Click to Set the parameters that have been modified (*only those that are not set to the default parameters*) in 29 and 31 into 2 in the format '–FlagName=value chosen'.
34. This screen is seen in more detail in figure 5.29.
4. Click to open http://spass.mpi-sb.mpg.de/webspass/help/options.html

(viii) `History` Screen

(Information is stored here by clicking the + tab from the `Main` screen).

35. The different entries in the `History` store are listed. Click the hyperlinks to view a particular entry in the `History` store.
36. The problem .dfg file copied from 1 for a particular entry in the `History` store.
37. The problem options copied from 2 for a particular entry in the `History` store.
38. The result of execution copied from 18 for a particular entry in the `History` store.
39. Click to Copy the .dfg file from 36 into 1.

(ix) The `Help` screen.

40. Click the `Help` tab to access the project Wiki.

41. The body of the Wiki: Available for users to add files, comments, bug reports, advice, building into an online manual.
4. Click for information on Wiki technology at `http://www.mediawiki.org/wiki/MediaWiki`.

(x) An example of an error dialogue box.

42. This dialogue box is raised when the +tab is pressed outside the Main screen.

Fig. i

Fig. ii

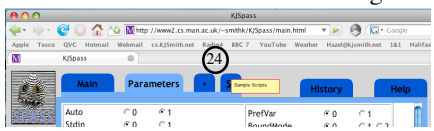


Fig. iii

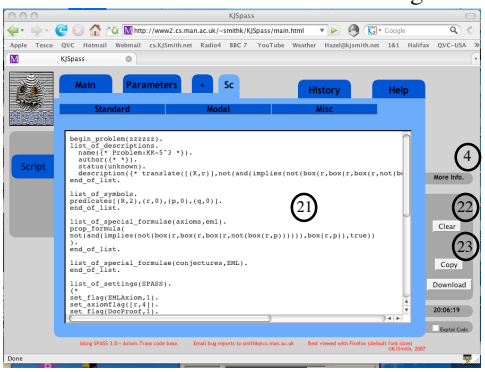


Fig. iv

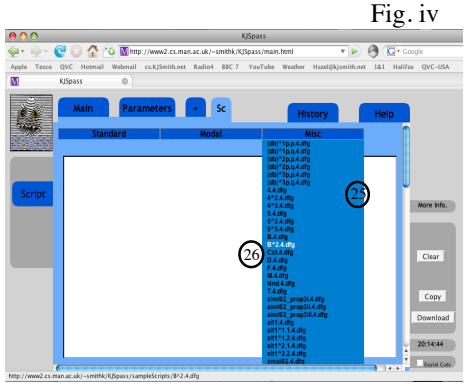


Fig. v

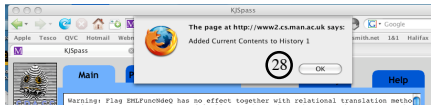


Fig. vi

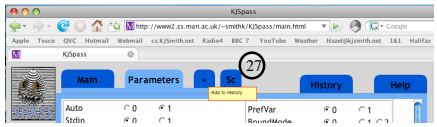


Fig. vii

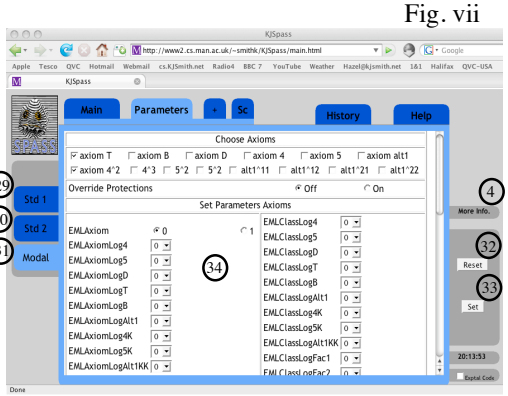


Fig. viii

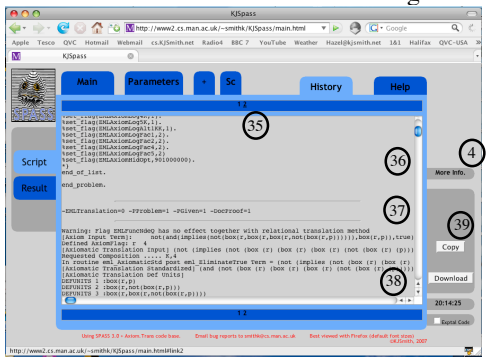


Fig. ix

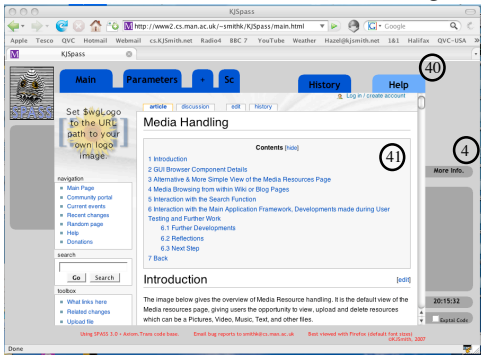


Fig. x

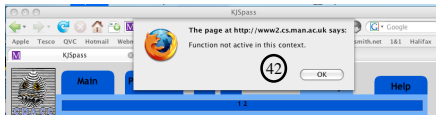


Figure 5.29 Details of the Modal Parameters tab of the user interface.
A screen capture from the user interface is shown. The figure illustrates the multiple ways in which the user may set the flag based syntax of the axiomatic translation (see section 5.1.2). It is expected that sections A, B, and E will be the most useful to the user, and in particular the bit-based flag calculator in section E.

Section A: Checkboxes are used to choose particular modal axioms to be set active.
Section B: The `Override Protection` radio-button switches between the 'protected translation mode' (`off`) and the 'experimental translation mode' (`on`) (see section 5.1.3).
Section C: Access is provided to command-line flags relevant to the axiomatic translation. (Non-sense combinations of flags are caught by extended-SPASS at execution time).
Section D: Access is provided to each element of the two bit-based flags relevant to axiomatic translation. (Non-sense combinations of flags are caught by extended-SPASS at execution time).
Section E: This is the bit-based flag calculator for axiomatic translation. The user is able to choose particular functions by name, and the interface calculates the appropriate values to enter into the bit-based flags `EMLAxiomOptions` and `EMLAxiomHidOpt`. For example, if the user wishes to exclude composition altogether from axioms B, D, alt_1, 4, 5, 4^K, 5^K, and alt_1^{kk} then `Do not use composition for any axiom in axiomatic translation` is set `on`. See section 5.1.3 for more details.