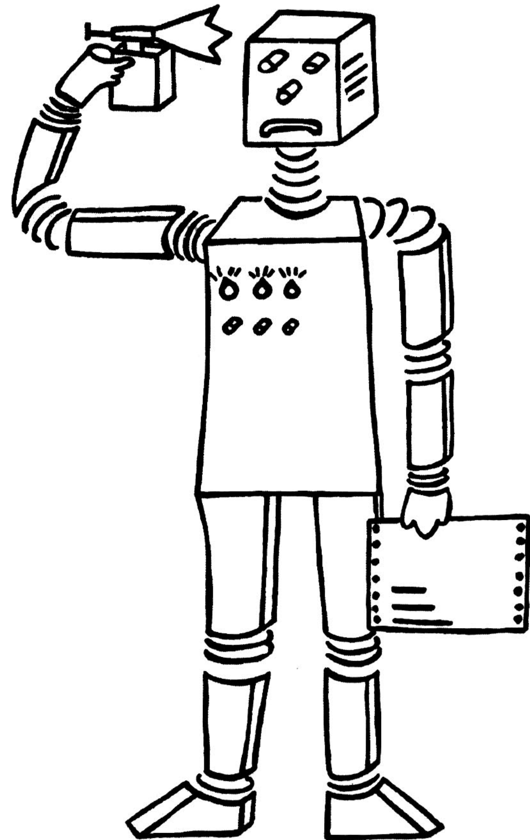
Fachbereich Informatik
Universität Kaiserslautern
Postfach 3049
D-6750 Kaiserslautern 1, W. Germany

SEKI-Working Paper



The MKRP User Manual

M. Beetz, H. Freitag, J. Klug
Ch. Lingenfelder (Ed.)

SEKI Working Paper SWP-88-01

The MKRP User Manual

M. Beetz, H. Freitag, J. Klug
Ch. Lingenfelder (Ed.)
SEKI Working Paper SWP-88-01

M. Beetz, H. Freitag, J. Klug

The MKRP User Manual

Contents

1. Introduction 1
2. The MKRP Operating System 9
3. The Input Language 15
4. The MKRP Editor 27
5. Setting the MKRP Parameters......... 33
6. Subsystem Commands 65
7. The Output Facility 66
8. A Test Run 72

1. Introduction

The current state of development of the Markgraf Karl Refutation Procedure (MKRP), a theorem proving system under development since 1977 at the Universities of Karlsruhe and Kaiserslautern, West Germany, is presented and evaluated in the sequel. The goal of this project can be summarized by the following three claims: it is possible to build a theorem prover (TP) and augment it by appropriate heuristics and domain-specific knowledge such that

i) it will display an active and directed behaviour in its striving for a proof, rather than the passive combinatorial search through very large search spaces, which was the characteristic behaviour of the TPs of the past. Consequently
ii) it will not generate a search space of many thousands of irrelevant clauses, but will find a proof with comparatively few redundant derivation steps.
iii) Such a TP will establish an unprecedented leap in performance over previous TPs expressed in terms of the difficulty of the theorems it can prove.

With about 25 man years invested up to now and a source code of almost 2000 K (bytes of Lispcode), the system represents the largest single software development undertaken in the history of the field and the results obtained thus far corroborate the first two claims.

The final (albeit essential) claim has not been achieved yet: although at present it performs substantially better than most other automatic theorem proving systems, on certain classes of examples (induction, equality) the comparison is unfavourable for the MKRP-system. But there is little doubt that these shortcomings reflect the present state of development; once the other modules (equality reasoning, a more refined monitoring and induction) are operational, traditional theorem provers will probably no longer competitive.

This statement is less comforting than it appears: the comparison is based on measures of the search space and it totally neglects the (enormous) resources needed in order to achieve the behaviour described. Within this frame of reference it would be possible to design the "perfect" proof procedure: the supervisor and the look-ahead heuristics would find the proof and then guide the system without any unnecessary steps through the search space.

In summary, although there are good fundamental arguments supporting the hypothesis that the future of TP research is with the finely knowledge

engineered systems as proposed here, there is at present no evidence that a traditional TP with its capacity to quickly generate many ten thousands of clauses is not just as capable. The situation is still (at the time of writing) reminiscent of todays chess playing programs, where the programs based on intellectually more interesting principles are outperformed by the brute force systems relying on advances in hardware technology.

The following paragraph summarizes the basic notions and techniques for theorem proving as far as they are relevant here (and may be skipped by a reader already familiar with the field).

Basic Techniques and Terminology

The language used in this report is that of first-order predicate logic with which we assume the reader to be familiar. From the primitve symbols of this logic we use: u,x,y,z as individual variables; a,b,c,d as individual constants; P, Q, R as predicate constants; f, g, h as function letters. The equality predicate will be denoted by E and mostly written in infix notation as ≡ to improve readability. Individual constants and variables are terms as well as n-place functions applied to n terms. As metasymbols for terms we use r, s and t. the arity of functions and predicates will be clear from the context. An n-place predicate letter applied to n terms is an atom. A literal is an atom or the negation thereof. For literals we use L, K. The absolute value |L| of a literal L is the atom K such that either L is K or L is ~K.

A clause is a finite set of literals for which the metasymbols C, D are used. A clause is interpreted as the disjunction of its literals, universally quantified (over the entire disjunction) on its individual variables. The empty clause is denoted as □. A ground clause, ground literal or ground term is one that has no variables occurring in it. A substitution δ is a mapping from variables to terms almost identical everywhere. Substitutions are extended to mappings from terms to terms by the usual morphism. Substitutions are also used to map literals (clauses) to literals (clauses) in the obvious way. A substitution is denoted as a set of pairs $\delta = \{(v_1 \leftarrow t_1) \ldots (v_n \leftarrow t_n)\}$ where the v_i are variables and the t_i are terms. The term δ(t) (the literal δ(L), the clause δ(C)) is called an instance of t (an instance of L, an instance of C). We use δ, σ for substitutions. A substitution σ is called a unifier for two atoms L and K, iff σ(L) = σ(K); σ is called a most general unifier (mgu) of L and K, if for any other unifying substitution δ there exists a substitution λ such that $\delta = \lambda \circ \sigma$, where ∘ denotes the functional composition of substitutions. A matcher (or one-way unifier) for two literals L and K relative to L is a substitution σ such that σL = K.

The Herbrand Universe H(S) of a set S of clauses is the set of all ground terms that can be constructed from the symbols occuring in S (if no individual constant occurs in S we add the single constant symbol c). A Herbrand instance H(t) of a term t is an instance $\delta(t)$, such that all terms in δ are from H(S); similarly we define a Herbrand instance of an atom, a literal, a clause. An interpretation T of S is a set of ground literals, whose absolute values are all the Herbrand instances of atoms of S such that for each Herbrand instance L of an atom exactly L or ~L is in T. An interpretation T satisfies a ground clause C iff $C \cap T \neq \emptyset$. T satisfies a clause C if it satisfies every ground instance of C in H(C); T satisfies a set of clauses S if it satisfies every clause in S. A model M of a set of clauses S is an interpretation that satisfies S. If S has no model it is unsatisfiable. For the equality predicate ≡ and a set of clauses S, a model M of S is an E-model if

i) $t \equiv t \in M$ for all terms t

ii) if the literals $L \in M$ and $s \equiv t \in M$ and if L' is obtained from L by replacing an occurence of s in L by t then $L' \in M$.

If S has no E-models then it is E-unsatisfiable.

Two literals are complementary if they have opposite sign and the same predicate letter.

If C and D are clauses with no variables in common and L and K are complementary literals in C and D respectively, and if $|L|$ and $|K|$ are unifiable with most general unifier σ, then $R = \sigma(C-\{L\}) \cup \sigma(D-\{K\})$ is a resolvent of C and of R and each literal L in R descends from a literal L' in C or D.

If C is a clause with two literals L and K and if a most general unifier σ exists such that $\sigma(L) = \sigma(K)$ then $F = \sigma(C-K)$ is called a factor of C. If C and D are clauses with no variables in common, and $s \equiv t$ is a literal in C, and r is a term occuring in D such that there exists σ with $\sigma(s) = \sigma(r)$, and D' is obtained from D by replacing r in D by t then $P = \sigma(D') \cup \sigma(C-\{ s \equiv t \})$ is a paramodulant of C and D. This inference rule is called paramodulation.

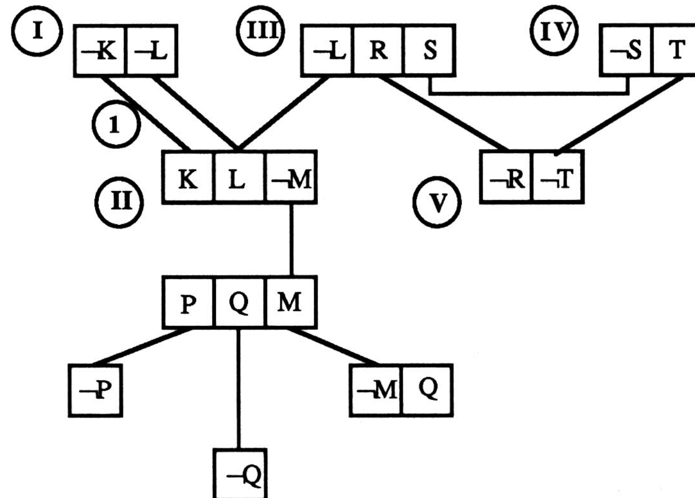
A connection graph CG is

i) a set of clauses S

ii) a binary relation **R** over literals in S, such that $(L, K) \in$ **R** if $|L|$ and $|K|$ are unifiable and L and K are of opposite sign. Sometimes we write $\langle S \rangle$ for the connection graph obtained from S.

A literal L in S is pure if it does not occur in any of the pairs of **R** i.e. it is not connected and the clause containing L may then be deleted in CG. A

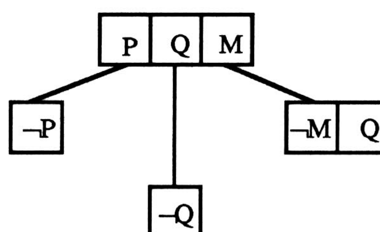
connection grph is graphically represented by drawing a link between L and K for every (L, K) ∈ R. L and K are said to be connected. Instead of repeating the definition of the connection graph proof procedure [Ko 75] we give an example for the derivation step. Consider the following connection graph:



Suppose we want to obtain the resolvent of clause (I) and clause (II), i.e. we want to resolve upon link (1). This is done by adding the resolvent to the graph and by connecting the resolvent in the following way: if a literal L in the resolvent descends from a literal L' in one of the parent clauses and if L' was connected to some literal K and if K and L are unifiable, then L and K are connected by a link. Finally the link resolved upon is deleted and all tautologies and all clauses containing pure literals are deleted.

For the connection graph above, resolving upon link (1) leads to a tautology, which is deleted, hence (I) and (II) are deleted since K, ~K is now pure.

Similarly clauses (III), (IV) and (V) are deleted; i.e. after one step the whole connection graph shrinks to:



This potentially rapid reduction of the original graph causes the practical attraction as well as the theoretical problems of this proof procedure. A more formal representation of the procedure is contained in [KARL MARK GRAPH paragraph 6.3.10].

Apart from the deletion of clauses containing pure literals there are additional deletion rules, which become particularily significant in the context of connection graphs: every deletion of a clause may cause further deletions of clauses (and links) and the resulting complex interplay is still not very well understood theoretically (see e.g. [BI 81] [EI 81] [EI 87] [SM 82]).

A clause C is a tautology if it contains two complementary literals L and K such that $|L|=|K|$ or a literal of the form $t \equiv t$.

A clause C subsumes a clause D if $|C| \leq |D|$ and there exists a substitution δ such that $\delta C \subset D$. (This is the definition of δ-subsumption in [LOV 78]).

Subsumed clauses and tautologeous clauses may be deleted from the graph, as discussed in [KARL MARK GRAPH section 6.3.3] and [KARL MARK GRAPH section 6.3.4] respectively. The unrestriced use of these deletion rules is known to make the respective proof procedure incomplete and even inconsistent.

A traditional refinement restricts the search space by blocking certain possible resolutions steps. For example a UNIT refutation, in which at least one parent clauses of a resolvent must be a unit clause, is such a refinement.
SET-OF-SUPPORT is also a refinement: the set of clauses is partitioned into two subsets (usually the set of the axiom clauses S and the set of the theorem clauses T) and resolution is only permitted if at least one parent clause is in T. The resolvents are put into T, i.e. the effect of set-of-support is most profitable at the beginning of the search, but it fades the more the deduction proceeds.

A LINEAR refinement selects a top clause from the set of theorem clauses and uses this clause as one of the parents for a resolution step. Then the resolvent becomes the top clause and so on either until the empty clause has been derived or backtracking is necessary.
The development of complete refinements was the main focus of research in theorem proving in the past and there may be close to a hundred now (see e.g. [LOV 78] [CHL 73]), some of those are used to advantage in the MKR-Procedure as well.

In contrast to a refinement, which only restricts the number of possible steps (and often "cuts off garbage and gold alike"), a strategy gives active advice as to what to do next. The development and integration of such strategies into one system was the main research problem of the MKRP project [cf. KARL

MARK GRAPH]. Strategic information overrides any other information: even if a particular refinement was chosen, the resulting deduction may be very different. Only if nothing better is known, does the system behave like a traditional theorem prover.

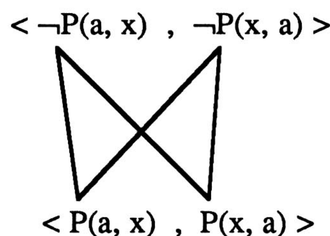
Completeness

The MKR-Procedure is incomplete, yet even worse it is inconsistent as it stands. This is partly so, because the implementation is not completed and partly because there are open theoretical problems in the connection graph procedure itself, see e.g. [BI 81] and [SM 82]. Most of the cases causing incompleteness (except paramodulation) however are irrelevant for practical examples; quite on the contrary, for some of them it is a hard job to find an example where it is in fact relevant.

In particular there are the following cases:

• As all reductions are performed before factorization, the graph may collapse although the clause set is unsatisfiable (i.e. the system is inconsistent):

Example:



All four R-links are tautology links and will be deleted causing purity deletion of both clauses, although the factors ‹P(a a)› and ‹NOT P(a a)› would allow for a refutation.

• Tautologies are deleted without any restriction, although this is known to be inconsistent, see [SM 82].

• Subsumed clauses and links are deleted without any restriction, which can also cause inconsistency, see [SM 82] [EI 81].

• Paramodulation and equality reasoning is not fully implemented. Especially the mechanism to generate only those P-links into variables which are necessary for completeness is not yet completed. Unrestriced generation of P-links from each side of an equation into each variable would blow up the graph without significantly increasing the total amount of information. Hence,

P-links into variables are not generated so far.

These deficiencies may be the reason that a proof exists, but cannot be found by the system. Even worse, the graph may collapse (usually indicating satisfiability), although the initial clause set is unsatisfiable.

As more theoretical results about clause graph procedures become known [EI83], we hope to eliminate at least the case causing inconsistency, whereas completeness results, although interesting as they may be from a theoretical point of view, are of course less important for practical purposes.

Overview of the System

The working hypothesis of the MKRP project first formulated in an early proposal in 1975, reflects the then dominating themes of artificial intelligence research, namely that TPs have attained a certain level of performance, which will not be significantly improved by:

i) developing more and more intricate refinements (like unit preference, linear solution, TOSS, MTOSS, ...), whose sole purpose is to reduce the search space, nor by

ii) using different "logics" (like natural deduction logics, sequence logics, matrix reduction methods etc.)

although this was the main focus of theorem proving deduction research in the past and of course it is not entirely without its merits even today.

The relative weakness of current TP-systems as compared to human performance is due to a large extent to their lack of the rich mathematical and extramathematical knowledge that human mathematicians have: in particular, knowledge about the subject and knowledge of how to find proofs in that subject.

To a lesser, but still important extent the relative weakness of current TP-systems can be attributed to the insufficient emphasis which in the past has been laid onto the software engineering problems and - sometimes even minor - design issues that in their combination account more for the strength of a system than any single refinement or "logical improvement".

Hence the object of the MKRP-project is firstly to carefully design and develop a TP system comparative in strength to traditional systems and secondly to

augment this system with the appropriate knowledge sources and heuristics methods. As a test case and for the final evaluation of the project's success or failure, the knowledge of an algebraic treatment of automata theory shall be made explicit and incorporated such that the theorems of a standard textbook [DE 71] can be proved mechanically. These proofs are to be transformed into ordinary natural language mathematical proofs, thus generating the first standard textbook entirely written by a machine.

In the following chapters we give a complete description of the user interface of the system. Most commands will work for any version of the MKRP, but the starting procedure is described for the SIEMENS BS2000 version.
The system also runs on a Symbolics Lisp Machine under Genera 7.1 (Common Lisp). The major differences to the SIEMENS version are mentioned at the end of each chapter.

2. The MKRP Operating System

After the logon command to a SIEMENS BS2000 system, the user types

/DO ATP, <load-subsystem>

to load the Markgraf Karl Refutation Procedure (MKRP for short). As the system as a whole doesn't fit into main memory, it is partitioned into four different subsystems. The <load-subsystem>-parameter selects one of them to be initially loaded. This parameter has one of the following four values:

E.COM for Edit
C.COM for Construct
R.COM for Refute
P.COM for Protocol

Independently from the subsystem being actually active, the system accepts any MKRP-command. If necessary, it automatically adds the subsystem needed to execute the command.

On a Symbolics the system is started by typing :Load System MKRP

(1) Architecture of the System from a User's View

The system consists of the following four subsystems:

- EDIT
- CONSTRUCT
- REFUTE
- PROTOCOL

EDIT provides a set of functions to create and manipulate sets of axioms and theorems.

Given a set of axioms and theorems (created by EDIT), **CONSTRUCT** generates the corresponding initial connection graph(s).

With the result of CONSTRUCT as input, **REFUTE** tries to detect a refutation by the application of resolution steps to the connection graph.

PROTOCOL produces a listing of initial clauses and the proof steps

which are necessary to deduce the empty clause.

(2) A More Detailed Description of the Four Subsystems

The subsystem **EDIT** enables the user to create axiom- and theorem-files, consisting of formulas of the input-language Predicate Logic Language PLL, which is discussed in chapter 3. The editor checks the syntactic and semantic correctness of the input formulas. These formulas are transformed into a formal problem description being stored in a so called problem file.

This problem file is the input for the **CONSTRUCT**-subsystem, which generates the initial connection graph(s) according to the reduction rules for initial graphs as specified by the user-defined adjustment of the options. The output of this subsystem is a graph file. Raw data for the protocol are written to a code file.

REFUTE then, tries to proof the theorem by resolving clauses in the connection graph. The user can control the deduction process by specifying various options (to be discussed in more detail in chapter 3.4). REFUTE also writes raw data to a code file. In order to get protocol raw data of CONSTRUCT and REFUTE on the same codefile use CR (see below) if possible.

PROTOCOL produces a listing of the initial clauses, together with the resolution proof as performed by REFUTE.

(3) A Precise Explanation of the Operating System Commands

At the toplevel of the MKRP-System, the following operating system commands are available:

```
V    H[ELP]    EX[IT]      O[PTIONS]          I[ND[UCTION]]
HC   L[ISP]    LO[GOFF]    S[UBSYSTEMS]
                                  EP
                                  FP
                           ER                 CP
                           FR
                 EC                    CR               RP
                 FC
  E[DIT]              C[ONSTRUCT]        R[EFUTE]     P[ROTOCOL]
  F[ORMULA]
```

D[EFINE.]D[IRECTORY] D[EFINE.]E[XAMPLENAME]
S[ORT.]G[ENERATION]

D[EFINE.]**D**[IRECTORY] <Directory1> <Directory2>
Defines the directories where the files will be written on and read from.

<Directory1> Directory for files to read and for files to write. As directory1 only a directory on the user's own user-id is allowed. The $-prefix is not allowed for directory1.

<Directory2> Directory for files to read.

D[EFINE.]**E**[XAMPLENAME] <Example-name>
Files generated during the session will be prefixed with the Example-name.

<Example-name> An example name; the Standard-name is 'TEMP.OS'.

V T|Y|YES turns the manual terminal control on.
V NIL|N|NO turns it off

H[ELP] <COM> explains the command <COM>
H[ELP] prints a list of all available OS-commands

EX[IT] terminates the MKRP-Session and returns to BS 2000

O[PTIONS] calls the option-module of the proof-control. It has its own self-explanatory command system. 'OK' returns to where you came from. The option module is described in detail in section 3.4

I[ND[UCTION]] Not yet fully implemented.
Calls the Karlsruhe Induction-Therorem-

Command	Description
	prover
HC <File>	Hardcopy of the Terminal-Session on <File>
HC T	Hardcopy on System File. Automatic Output on Printer
HC N |NIL	HC is switched off and <File> is closed
L[ISP]	Calls Interlisp. Return with 'OK'
LO[GOFF]	terminates this Terminal-Session
S[UBSYSTEMS]	Shows the actual system-configuration. It has its own self-explanatory command system. 'OK' returns to where you came from. The subsystem commands are explained in section 3.5.
S[ORT.]**G**[ENERATION]	<Input Graph File> <Output Graph File> Transforms unary predicates into sorts; in some cases a contradiction or a model is found.
<Input Graph File>	A file containing the initial graph(s). If nil, the last one created is used, if such a file exists.
<Output Graph File>	File containing new initial graphs. New split parts are possible.
E[DIT] [<Problem File>]	creates a problem description. To that end the formula editor is called twice, first for the axiom formulas, then for the theorem formulas. The problem description is written on
<Problem File>	, if given, otherwise a default name is used.
F[ORMULA]	<Axiom File> <Theorem File> [<Problem File>] creates a problem description from compiled formula files.
<Axiom File>	and
<Theorem File>	contain the compiled axiom and theorem

	formulas. The problem description is written to
<Problem File>	, if given, otherwise a default name is used. Using this command requires <Axiom File> and <Theorem File> to be compatible, i.e. they must be created during the same editor session.
C[ONSTRUCT]	[<Problem File>[<Graph File>[<Code File> [<Comment>[<Batch File>[<ATP Version>]]]]]] creates a set of initial graphs from a problem description.
<Problem File>	contains the problem description. If nil, the last one created is used, if such a file exists.
<Graph File>	is the file the initial graph(s) will be written on. If nil, a default file name is used.
<Code File>	is the file, where the raw data for the protocol will be written to. If nil, a default file name is used.
<Comment>	is inserted into the proof protocol. <Comment> must be a list, each element is printed in a separate line.
<Batch File>	,if given, causes the creation of a batch job using the given
<ATP-Version>	or the standard version, respectively.
R[EFUTE]	[<Graph File>[<Number>[<Code File>[<Batch File>[<ATP-Version<]]]]] refutes a set of initial graphs and creates raw data for the protocol.
<Graph File>	is the file containing the initial graph(s). If nil, the last one created is used, if such a file exists. If a
<Number>	say n, is given, only the nth graph is refuted, otherwise (i.e. <number> = nil) all graphs on

the file.

<Code File> is the file, where the raw data for the protocol will be written to.

<Batch File> if given, causes the creation of a batch job using the given

<ATP-Version> or the standard version, respectively.

P[ROTOCOL] [<Code File>[List File>[<Batch File> [<ATP-Version]]]]
creates a proof protocol from raw data.

<Code File> File containing the raw data.

<List File> File, where the processed data for protocol will be written to in a readable format.

<Batch File> ,if given, causes the creation of a batch job, using

<ATP-Version> or the standard version, respectively.

The commands Edit, Formula, Construct, Refute and Protocol can be combined in various different ways.
This causes several commands being executed after one another.
Possible combinations are:

EP
FP
ER CP
FR
EC CR RP
FC

For example the CP <Problem File> <Graph File> <Code File>
<comment> <List File>
causes first CONSTRUCT to create a <Graph File> and a <Code File> starting with <Problem File>. Then REFUTE tries to refute the graph and writes also protocol raw data to the <Code File> and at last PROTOCOL creates a <List File>.

3. The Input Language

The PREDICATE LOGIC LANGUAGE (PLL), a formal language in which sorted first-order predicate logic formulas can be formulated, is described. Axioms and theorems, which are given to MARKGRAF KARL REFUTATION PROCEDURE, are represented in PLL. The language constructs of PLL which reflect the special facilities of this system are exhibited, i.e.

- an inference mechanism based on a many-sorted calculus,
- the incorporation of special axioms into the inference mechansim, and
- the control of the inference mechanism using special derivation strategies.

Basic Concepts

In PLL all the usual junctors, denoted by OR, AND, IMPL, EQV, and NOT, the universal quantifier ALL and the existential quantifier EX are present. Junctors and quantifiers have the following priorities when used in a formula without parentheses:

(1) NOT
(2) AND
(3) OR
(4) IMPL
(5) EQV
(6) ALL, EX

In a formula without parentheses, the rightmost junctor has precedence over all junctors of the same priority to its left.

Example

NOT A OR B AND C is equivalent to (NOT A) OR (B AND C) and A IMPL B IMPL C is equivalent to A IMPL (B IMPL C).

In PLL the sign '=' denotes the equality symbol, i.e. we use a first-order predicate calculus with equality. As an example for using PLL, we axiomatize a group:

Example

```
* AXIOMATIZATION OF A GROUP WITH EQUALITY,
* F IS A GROUP OPERATOR AND 1 IS THE IDENTITY ELEMENT
```

```
ALL X,Y EX Z   F(X Y) = Z
ALL X,Y,Z      F(X F(Y Z)) = F(F(X Y) Z)
ALL X          F(1 X) = X AND F(X 1) = X
ALL X EX Y     F(X Y) = 1
```

A theorem given to the MKRP system could be for instance:

```
*IDEMPOTENCY IMPLIES COMMUTATIVITY

ALL X F(X X) = 1 IMPL (ALL X,Y,F(X Y ) = F(Y X))
```

The lines starting with a '*' are PLL-comments. We give another axiomatization of a group:

Example

```
* AXIOMATIZATION OF A GROUP WITHOUT EQUALITY
* P(X Y Z) DENOTES F(X Y) = Z WHERE F IS THE
* GROUP OPERATOR. E IS THE LEFTIDENTITY.

ALL X,Y EX Z        P(X Y Z)
ALL X,Y,Z,U,V,W     P(X Y U) AND P(Y Z V) IMPL
                    (P(X V W) EQV P(U Z W)
ALL X               P(E X X)
ALL X EX Y          P(X Y E)
```

Now a theorem could be for instance:

```
* LEFTIDENTITY IS RIGHTIDENTITY

ALL X P(X E X)
```

The Many-Sorted Calculus

Assume we have a set of symbols ordered by the subsort order, a partial order relation which is reflexive, antisymmetric and transitive. Variable, constant and function symbols are associated with a certain sort symbol. The sort of a variable or constant symbol is its rangesort and the sort of a term which is different from a variable or constant symbol is determined by the rangesort of its outermost function symbol.

All argument positions of a function or predicate symbol are associated with certain sort symbols, called the domainsorts. In the construction of the well formed formulas of the many-sorted calculus, only those terms may fill an argument position of a function or predicate symbol, whose sorts are subsorts of the domainsorts given for the argument position of the respective function or predicate symbol.

Besides the increase of readability of axiomatizations, the usage of the information given by the range- and domainsorts and by the subsort order prevents the inference mechanism of the theorem prover to do useless derivations. The theoretical foundation of the many-sorted calculus implemented in the MARKGRAF KARL REFUTATION PROCEDURE can be found in [Walther-83].

As an example for an application of a many-sorted calculus we axiomatize sets of letters and digits and some basic operations for these sets:
Example

```
* DEFINITION OF THE SORTS LETTER AND DIGIT; I.E.
* A, B, ... , Z ARE CONSTANTS OF SORT LETTER AND
* 0, 1, ... , 9 ARE CONSTANTS OF SORT DIGIT

TYPE A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R,S,T,U,V,W,X,Y,Z: LETTER
TYPE 0,9,8,7,6,5,4,3,2,1: DIGIT

* LETTER AND DIGIT ARE SUBSORTS OF SORT SIGN

SORT LETTER, DIGIT:SIGN

* DEFINITION OF THE EMPTY SET AND SET-MEMBERSHIP,
* I.E. EMPTY IS A CONSTANT OF SORT SET AND MEMBER
* IS A BINARY PREDICATE DEFINED ON (SIGN SET)

TYPE EMPTY:SET
TYPE MEMBER(SIGN SET)
ALL X:SIGN NOT MEMBER(X EMPTY)
ALL U,V:SET U = V EQV (ALL X:SIGN MEMBER (X U) EQV MEMBER (X V))
* DEFINITION OF SINGLETONS, I.E.
* SINGLETON IS A FUNCTION MAPPING SIGN TO SET

TYPE SINGLETON(SIGN):SET
ALL X:SIGN ALL U,V:SET (MEMBER(X U) OR MEMBER(X V))
```

EQV MEMBER (X UNION (U V))

Theorems to be proved by the ATP system could be for instance:

* UNION IS IDEMPOTENT AND EMPTY IS AN IDENTITY ELEMENT

ALL X:SET UNION (X Y) = X AND UNION(EMPTY X) = X

* SINGLETON IS INJECTIVE

ALL X,Y:SIGN SINGLETON(X) = SINGLETON(Y) IMPL X = Y

* EACH LETTER IS A SIGN

ALL Y:SET (EX U:LETTER MEMBER(U Y))
IMPL (EX X:SIGN MEMBER(X Y))

Attributes of Functions and Predicates

Attributes are abbreviations for their defining axioms, i.e. first-order axioms which axiomatize certain properties of functions or predicates.

The effect in stating a certain attribute of a function or predicate using an attribute declaration is formally the same as giving the defining axiom to the ATP. At the moment the following properties can be declared.

Attribute Declaration	Defining Axiom	
REFLEXIVE(P)	ALL X	P(X X)
IRREFLEXIVE(P)	ALL X	NOT P(X X)
SYMMETRIC(P)	ALL X,Y	P(X Y) IMPL P(Y X)
ASSOCIATIVE(F)	ALL X,Y,Z	F(X F(Y Z)) = F(F(X Y) Z)

The defining axioms of attributes are incorporated into the inference mechanism of the system as described above in [KARL MARK G RAPH-84].

Example

In the first example for instance the associativity of the group operator F could be stated by: ASSOCIATIVE (F). In the second example we could write as an axiom: ASSSOCIATIVE(UNION).

The Syntax of PLL

The formal syntax of PLL defined by a context free grammar is contained in [Walther-82], which presents additional details and examples.

Semantic Constraints for PLL

In the sequel we state the semantic constraints (i.e. the context dependent language features) for PLL. The strings in angle brackets, e.g. <term>, refer to the production rules of the PLL-grammar as defined in [Walther-82].

Sort Symbols

Sort symbols are introduced with their first usage in
- a <sort declaration>, e.g. SORT LETTER, DIGIT:SIGN, ALPHABET
- a <type declaration>, e.g. TYPE A,B:BOOL,
- a <variable declaration>, e.g. ALL Z:INT EX N:NAT ABS(Z) = N

The direct subsort relation imposed on the set of sort symbols is a partial, irreflexive and non-transitive relation such that the predefined sort symbol ANY is no direct subsort of each sort symbol and each sort symbol different from ANY is a direct subsort of at least one other sort symbol.

The subsort order imposed on the set of sort symbols is the reflexive and transitive closure of the direct subsort relation.

The subsort symbols left of the colon in a <sort declaration> are direct subsorts of each sort symbol to the right of the colon in the <sort declaration>.

The sort symbols right of the colon in a <sort declaration> are direct subsorts of ANY, provided these sort symbols are introduced by this <sort declaration>.

Example
For the <sort declaration> given above LETTER and DIGIT are subsorts of SIGN and of ALPHABET, and SIGN and ALPHABET are direct subsorts of ANY.
Hence LETTER, DIGIT and SIGN are subsorts of SIGN and ANY, SIGN, ALPHABET, LETTER and DIGIT are subsorts of ANY.

Variable Symbols

Variable symbols are introduced by a <variable declaration> in a <quantification>.

Example

ALL X,Y EX Z:S P(X Y Z)

The scope of a <variable declaration> is the <quantification> following the <variable declaration> in a <quantification>.

In its scope each <variable symbol> has as rangesort the sort symbol given by the <sort symbol> following the colon in its <variable sort> of the <variable declaration>. If no <variable sort> is present, the rangesort of the <variable symbol> is the predefined sort symbol ANY.

Example

The expression given in the above example has the following sorts:
rangesort(X) = rangesort(Y) = ANY and
rangesort(Z) = S.

In each <quantification> variable symbols are consistently renamed from left to right to resolve conflicts on multiple introductions of variable symbols.

Example

ALL Y,X P(Y) is the same as ALL X,Y P(Y) and
ALL X (EX X P(X)) IMPL Q(X) is the same as
ALL X (EX Y P(Y)) IMPL Q(X)

<u>Constant Symbols</u>

Constant symbols are introduced with their first usage
- in a <type declaration>, e.g. TYPE -1,+1:INT
- as <term>, e.g. ALL X P(X A) OR F(C) = D

Each constant symbol has a rangesort the <sort symbol> following the colon in the <type declaration> which introduces the <constant symbol>.

Example
For the expressions given above we find
rangesort(-1) = rangesort(+1) = INT.

The rangesort for a constant symbol which is introduced with its first usage as a <term> is ANY.

Note that in PLL variable symbols are always preceded by a quantifier and thereby can always be distinguished from constant symbols. As a consequence there is no concept of free variables in PLL.

Function Symbols

Function symbols are introduced with their first usage in
- a <type declaration>, e.g. TYPE ABS(INT):NAT
- an <attribute declaration>, e.g. ASSOCIATIVE(PLUS)
- a <term> , e.g. ALL X P(F(X)) OR G(X) = A

Each function symbol is associated with a sort symbol for each argument position i, called its ith domainsort, with a natural number, called its arity, and with a sort symbol, called its rangesort.

Function symbols which are introduced by <type declaration> have as their domainsorts the <sort symbol>s given on the appropriate positions in the list of <sort symbol>s following the <function symbol> in the <type declaration>.

Example
For the expression TYPE PRODUCT(SCALAR VECTOR):VECTOR we get domainsort (PRODUCT 1) =SCALAR and domainsort (PRODUCT 2)=VECOTR.

A <function symbol> which is introduced by a <attribute declaration> or by its first usage in a <term> has ANY as ith domainsort for each argument position i.

The arity of a function symbol is defined as
- the number of sort symbols in the list of <sort symbols> following the <function symbol> in the <type declaration> which introduces the <function symbol>
- two, for a <function symbol> introduced by an <attribute declaration>
- or else the number of arguments at its first usage in a <term>.

Example
For the expressions given above we get arity(ABS) = 1,
arity(PLUS) = 2 and arity(F) = arity(G) = 1.
The rangesort of a <function symbol> is defined by the <sort symbol> following

the colon in a <type declaration>. Its rangesort is ANY if the <function symbol> is introduced by a <attribute declaration> or by its first usage in a <term>.

Example
For the examples given in the above expressions we get
rangesort(ABS) = NAT
rangesort(PRODUCT) = VECTOR, and
rangesort(PLUS) = rangesort(F) = rangesort(G) = ANY.

Predicate Symbols

A predicate symbol is introduced with its first usage in
- a <type declaration>, e.g. TYPE MEMBER(SIGN SET)
- a <atom> in a formula, e.g. EX X,Y P(X Y) AND Q

Each predicate symbol is associated with a natural number, called its arity, and with a sort symbol for each argument position i, called its ith domainsort.

The arity and domainsort of predicate symbols are determined in the way arity and domainsorts are determined for function symbols.

The <equality symbol> is a predefined predicate symbol with arity 2 and 1st and 2nd domainsort ANY. It is the only predicate symbol which is written in infix notation.

'TRUE' and 'FALSE' are predefined predicate symbols with arity 0, which have the obvious meaning.
In the following the numbers in angle brackets, e.t. <23>, denote error code numbers returned by the PLL-compiler of the MARKGRAF KARL REFUTATION PROCEDURE (summarized below) when given a semantically incorrect <expression> as input. The phrase unknown symbol denotes a string of the terminal alphabet of the PLL-grammer, which was not used before.

Semantically correct Sort Declarations

A <sort declaration> SORT S1, ... , Sm:T1, ... , Tn is semantically correct, if
- all Si and all Tj (i=1...m, j=1...n) are sort symbols or else are unknown symbols (otherwise error message) <61,62,63,64> and Si is a direct subsort

of Tj <1> or else at least one of the symbols S and Tj is an unknown symbol <2>.

Semantically correct Type Declarations

A <type declaration> T is semantically correct if

- T is TYPE C1, ... , Cn:S and S is a sort symbol or else is an unknown symbol <61,62,63,64> and for all i=1...n Ci is a constant symbol with rangesort(Ci)= S <14> or Ci is an unknown symbol <11,12,16,17>
- or T is TYPE P(S1,...,Sn) and for all i=1...n Si is a sort symbol or else is an unknown symbol <61,62,63,64> and P is a predicate symbol with arity(P)=n <34> and domainsort(P=i)=Si<36> or else is an unknown symbol <31,32,36,37>
- or T is TYPE F(S1...Sn):S and for all i=1...n S and Si are sort symbols or else are unknown symbols <61,62,63,64> and F is a function symbol with arity(F)=n <23>, rangesort(F)=S <27> and domainsrot (F i)=Si <26> or unknown symbol <21,22,24,28>.

Semantically correct Attribute Declarations

A <attribute declaration> ASSOCIATIVE(F) is semantically correct if

- F is a function symbol with arity(F)=2 <23>, rangesort (F) = domainsort(F 1) = domainsort(F 2) or else is an unknown symbol <21,22,24,28>.

The <attribute declarations>s REFLEXIVE(P), IRREFLEXIVE(P) and SYMMETRIC(P) are semantically correct if

- P is a predicate symbol with arity(P)=2 <34> and domainsort (P 1) = domainsort (P 2) <36> or else is an unknown symbol <31,32,33,37>.

Semantically correct Terms, Atoms and Quantification

The sort of a term t, denoted sort(t), is the rangesort of t, if t is a variable or constant symbol, and else is the rangesort of the outermost function symbol of t.

A <term> T is semantically correct if

- T is a constant symbol, a variable symbol or an unknown symbol

<11,12,16,17>
- or T is F(T1...Tn) and for all i=1...n, Ti is a semantically correct term, F is a function symbol with arity(F)=n <23> and sort (Ti) is a subsort of domainsort(F i) <81> or else F is an unknown symbol <21,22,24,28>.

An <atom> A is semantically correct if
- A is a predicate symbol with arity(A)=0 <34> or A is an unknown symbol <31,32,33,37>
- or A is P(T1...Tn) and for all i=1...n, Ti is a semantically correct term, P is a predicate symbol with arity(P)=n <34> and sort(Ti) is a subsort of domainsort(P i) <81> or else P is an unknown symbol <31,32,33,37>
- or A is T1 = T2 and T2 are semantically correct terms and = is an <equality symbol>.

A <quantification>Q is semantically correct if
- Q is ALL X... or EX X... and X is a variable symbol or an unknown symbol <51,52,53,55> and each atom in Q is semantically correct.

<u>Errors detected by the Compiler</u>

The PLL compiler of the ATP system checks each input for syntactical and semantical correctness. An input containing signs which are not member of the terminal alphabet is responded by a message.

*** SYMBOL ERROR <<< xxx IS NO ADMISSIBLE SYMBOL

where xxx is a sign which is not member of the terminal alphabet.

For a syntactically incorrect input, the compiler responds

++++ SYNTAX ERROR >>> xxx NOT ACCEPTED
UNEXAMINED REMAINDER OF THE INPUT >>> zzz

where xxx is the sign which causes the syntactical incorrectness and zzz is the unanalysed remainder of the given input.

For a syntactically correct but semantically incorrect input, the compiler

responds

***** SEMANTIC ERROR nnn >>> message
UNEXAMINED REMAINDER OF THE INPUT >>> zzz

where 'nnn' is the semantic error code, 'message' is an error message explaining the kind of the semantic error and 'zzz' is the (not analysed) remainder of the given input.

<u>Particularities of the Input Routines</u>

Since the whole ATP system is an INTERLISP program, the special features of the INTERLISP input routines have to be taken into account, i.e.

- () is read as NIL
- 'X is read as (QUOTE X)
- < is read as (
- > is read as a non empty sequence of)' s
- > closes all left-brackets up to the first left-superbracket <
- each left-bracket has to be matched by a right-bracket or by a right-superbracket >
- each input has to contain an even number of " (i.e. the string indicator)
- a sequence of blanks is read as one blank (except in a string)
- + is read as a blank if it is followed by a sequence of digits, e.g. +4711 is read as 4711
- a sequence of zeroes is read as a zero, unless the sequence is preceded by non-zero sign, e.g. 007 is read as 7.

<u>Separator Characters</u>

In INTERLISP each of the following characters separates S-expressions:

- a blank
- a bracket, i.e.), (, > or <
- the quote sign, i.e. '
- the string indicator, i.e. "

Signs acting as separators in PLL are

- all INTERLISP separators
- the colon, i.e. X:Y is the same as X : Y and
- the comma, i.e. X,Y is the same as X , Y .

4. The MKRP-Editor

The MKRP-Editor is a screen-oriented, syntax-directed editor for the sorted logic formulas, written in PLL. The formulas are kept in two different areas: if a formula was accepted by the compiler, it is included into the active area. In this case symbol table entries and prefix form exist for the formula and are accessable.

Other formulas (e.g. such with syntax errors) are stored in the passive area. Thus even erroneous input is not lost and the passive area can be used as a scratch pad.

When terminated the editor returns the list of the formulas in the active area for further processing (e.g. by the theorem prover). The passive formulas are not considered.

Below is a list of the editor commands. Several commands can be concatenated by the separator | .
Every command must begin with an atom (insert without command name).

I[NSERT] <FORMULA>
<FORMULA>

If formula <FORMULA> is syntactically and semantically correct, it will be inserted as last one in active area, else as first one in passive area.

D[ELETE] [<FROM>] [-] [<TO>]
D[ELETE] [<FROM>] [/] [<TO>]

If <TO> is greater or equal than number of last formula in active area, the corresponding formulas will be deleted.

Examples:	D-	deletes all formulas,
	D 2	deletes formula 2,
	D 3-	deletes all formulas from the third one,
	D	deletes the last formula of active area if it exists, else the first of passive area.

+[SHIFT] [<NUMBER>]
++[SHIFT]

+SHIFT shifts the first <NUMBER> formulas of the passive area into the active area, if they are syntactically and semantically correct.
Default -value of <NUMBER> is 1.

++ SHIFT shifts all formulas.

```
Example:   * 1 *                 * 1 *
             2        + 2        * 2 *
             3    ---------->    * 3 *
             4                     4
```

-[SHIFT] [NUMBER]
--[SHIFT]

-SHIFT shifts the last <NUMBER> formulas of the active area into the passive area.

Default -value of <NUMBER> is 1.

--SHIFT shifts all formulas.

```
Example:   * 1 *       -         * 1 *
           * 2 *  ---------->      2
```

E[DIT] [<NR>]

The Lisp-Editor is called for the formula <NR>.

If the active area is not empty, the default-value of <NR> is the number of last formula in active area, else 1.

C[HANGE] [<NR>]

The formula <NR> is printed on the terminal. User shall enter a new formula to replace the printed one.

If the active area is not empty, default-value of <NR> is the number of last formula in active area, else 1.

R[EAD] <FILE>

<FILE> must be created by the write-command. If the editor is in the initial state, the formulas are inserted into the same areas containing them at write time. Otherwise all formulas are inserted into the passive area.

W[RITE] <FILE>

The contents of the editor will be saved on file <FILE>, so that it can be restored with the read-command.

G[ET] <FILE>

<FILE> contains a sequence of formulas in the following format:

(<FORMULA$_1$>)
(<FORMULA$_2$>)
.
.
.
(<FORMULA$_n$>)
STOP

the formulas are inserted into the passive area.

EXEC[UTE] <FILE>

The file <FILE> contains a sequence of editor commands in the following format:

<COMMAND$_1$> |
<COMMAND$_2$> |
.
.
<COMMAND$_n$> |
OK

The commands will be executed. For further dialogue the terminal is used. Take care of the right number of closing parentheses.

SW[ITCH] [<NR>1] [<NR2>]

If formula <NR1> and formula <NR2> are in the passive area they will be exchanged.
If the user only gives one number, this and first formula are taken, switch alone exchanges first and last formula of passive area.

U[NDO] [<NUMBER>]
U[NDO] ON
U[NDO] OFF

UNDO ON AND UNDO OFF are switching undo-mode ON OR OFF, respectively. UNDO undoes the last <NUMBER> destructive commands, AS INSERT, DELETE, EDIT, CHANGE, +SHIFT, -SHIFT, READ, AND GET.
NO <NUMBER> means one command.

REP[LACE] <OLD> <NEW>
X <OLD> <NEW>

REPLACE replaces the symbol <OLD> by the symbol <NEW> in active and passive area as well as in symbol-table.
REPLACE cannot be undone,
nor can any commands executed prior to a replace.

PP[RINT] <FILE>

Print writes the symobl-table and all formulas in a readable form on the given file.

S[HOW] <X1>...<XN>
S[HOW] -

Each <X1> must be a symbol-name or a list of kinds.
There are the kinds S[ORT], F[UNCTION], P[REDICATE] and C[ONSTANT] all symbols that are so specified will be shown on the terminal.

Example: S (P) F will show all predicates and symbol F,
S- all symbols.

SS[HOW] <X1>...<XN>
SS[HOW] -

SSHOW is a more beautyful version of show.
Each <XI> must be a symbol-name or a list of kinds.
There are the kinds S[ORT], F[UNCTION], P[REDICATE] and C[ONSTANT] all symbols that are so specified will be shown on the terminal.

Example: SS (P) F will show all predicates and symbol F,
SS- all symbols.

PRE[FIX] [<FROM>] [-] [<TO>]
PRE[FIX] [<FROM>] [/] [<TO>]
F [<FROM>] [-] [<TO>]
F [<FROM>] [/] [<TO>]

PREFIX or F writes the compiled (PREFIX-) form of the formulas <FROM> to <TO> on the screen.

Examples: PRE- or PRE/ all formulas of the active area,
PRE 3 formula 3, if active,
PRE -5 the formulas 1 to 5, if they are in the active area.

IN[FIX] [<FORM>] [-] [<TO>]
IN[FIX] [<FORM>] [/] [<TO>]
L[IST] [<FORM>] [-] [<TO>]
L[IST] [<FORM>] [/] [<TO>]

INFIX or LIST writes the INFIXFORM (INPUT-LANGUAGE) of the formulas <FROM> to <TO> on the sreen.

Examples: IN- OR IN/ all formulas,
IN7 the seventh formula,
IN -5 the formulas 1 to 5
IN the last formula of active area, or if it doesn't exist the first of passive area.

ST[ATE]

STATE shows the user, how the formulas are distributed on the areas.

Example: AREA 1 : 3 formulas (active area)
AREA 0 : 1 formula (passive area)

OK

OK terminates the editor and returns to the calling module.
OK used on a file for the execute command terminates the execute command.

!

! terminates the editor and returns to ATP-TOP-LEVEL.
! in the input of a command cancels it.

LISP

LISP calls the LISP system.
You can return to the editor by OK.

V ON
V [OFF]

V ON and V T will switch on the manual TELETYPE-CONTROL.
V OFF and V will switch it off.

H[ELP] [<COMMAND>]

HELP prints a list of all possible commands on the screen.
HELP <COMMAND> explaines the command <COMMAND>.

HH[ELP] <FILE>

HHELP prints explanations for all possible commands on file 'FILE'.

In the version running on Symbolics machines, the editor ZMACS is used to edit the formulae. It is called from the editor subsystem of MKRP by the command I[NSERT] without a parameter. Alternatively one may use G[ET], if the formulae have been written before.
In both cases every formula has to be enclosed in parentheses. It is then actually inserted into MKRP by marking it and typing the key <END>. If more than one formula is marked, all of them are entered in order.

5. Setting the MKRP Parameters

The OPTIONS command is the key into the options module. It offers the possibility to adjust various parameters to govern the overall search behavior and influence trace and protocol. The options are subdivided into several areas according to their tasks.

(1) Explanation of the Commands

In the OPTIONS-module, the following commands are available:

H[ELP] P[RINT] PP[RINT] R[EAD] W[RITE]
L[ISP] OK V

H[ELP]	prints a list of all available commands
H[ELP] <com>	explains the command <com>
P[RINT] <Area>	prints the options of <Area> and their current values
PP[RINT] <File>	prints all areas , their options and default values together with a detailed explanation on <File>
R[EAD] <File>	reads the option-values from <File>. This file must have been created by a W[RITE] <File> command
W[RITE] <File>	writes all options and their current values on <File>
L[ISP]	calls INTERLISP. Return with 'OK'.
V T|Y|YES|OK	turns the manual terminal control on.
V NIL|N|NO	turns it off again
<AREA>	prints all the options of the area <AREA>
<OPT><VAL>	sets the option <OPT> to the new value <VAL>

2) Explanation of the Options

The purpose of the options is to adjust the general search behavior of the theorem prover to the characteristics of the given problem. According to their tasks the options are grouped in the following categories:

TWO RED.I RED.D STR GEN TR PR

All options have default values.

To influence the construction of the initial connection graph (CONSTRUCT) the user can change the values of the options of TWO, RED.I and the option SPLITTING of the area GEN.

The search behaviour during the deduction process (REFUTE) can be governed by options of the areas RED.D and STR.

Tracing of the deduction process can be influenced by options of the area TR. The information on the protocol can be selected by options of the area PR (options of the areas TR and PR are discussed in detail in chapter 7).
All other options which do not fit in any of these areas are put into the area GEN.

To solve difficult problems it seems to be very useful to change the values of the options:

RED.I : LINK.INCOMPATIBILITY
RED.D : LINK.INCOMPATIBILITY
STR : TERM.DEPTH
STR : TERM.ITERATIONS
STR : TERM.BREADTH.FIRST

Advantages and disadvantages of the various option settings are discussed below. Reduction operations should be switched off for efficiency reasons if it is known that there is no possibility for applying a reduction operation (especially link reductions).

In the following description some effects of options of the areas TWO, RED.I and RED.D are demonstrated by examples of simple refutation graphs. The default values of options are marked with "▸".

OPTIONS OF THE CATEGORY TWO

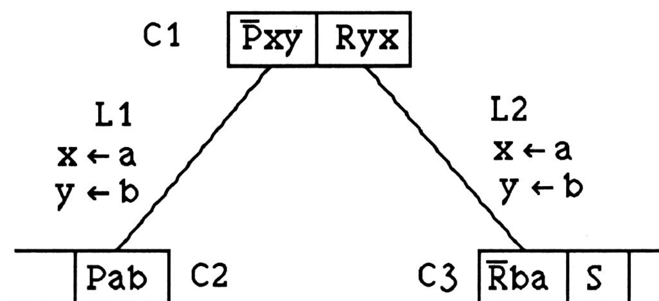
This category consists of options handling two-literal-clauses during the construction of the initial connection graph in a special way. The special handling results in the insertion of additional links representing a sequence of two or more deduction steps.

TWO:RULES

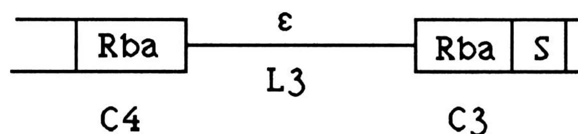
The value of this option controls the special handling of two-literal-clauses.

Examples

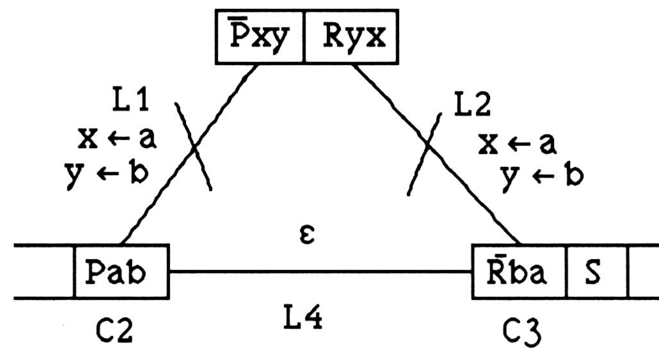
1)



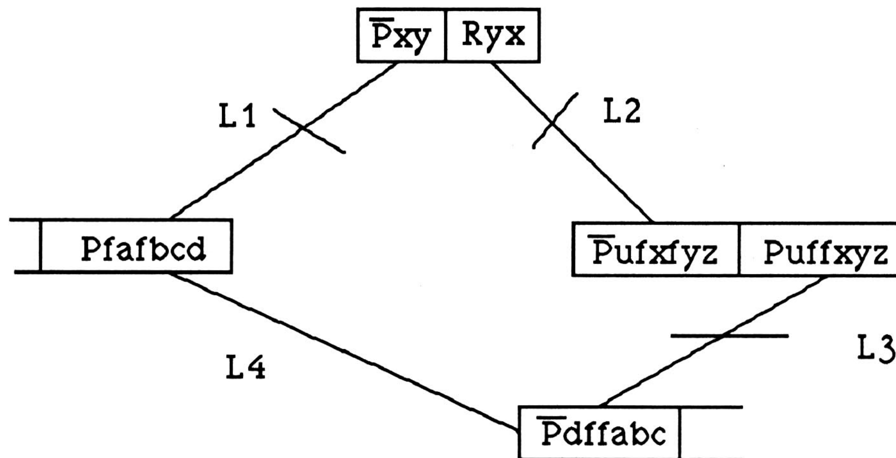
Resolving link L1 and inheriting link L2 creates:



This deduction process can be abbreviated by inserting a link between C2 and C3. Resolving on this new link has the same effect as resolving on L3, but takes only one resolution step.

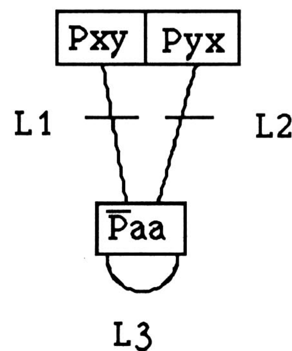


2) By this method we can save more than one resolution step:
(f is a binary function symbol)



By this way, chains of two-literal-clauses of any length can be simulated by one link.

3) The special handling of two-literal-clauses can also connect a clause with itself like:



In the original graph (without link L3) it is possible to deduce the empty clause. In the graph resulting from the special handling of two-literal-clauses this is no longer possible.

Possible Values

T/Y/YES :	Two-literal-clauses are treated as described above
PARTIAL :	Link insertion only if the two clauses connected by the new link are different.
▸ NIL/N/NO:	The two-literal-clause algorithm is switched off.

Effects of the special handling of two-literal-clauses

The advantages of the special handling are at one hand that the proof procedure needs less deduction steps to deduce the empty clause, on the other hand all clauses representing intermediate clauses are not inserted in the graph.
Switching on the two-literal-rule algorithm may cause the system to consume more time. The completion of the graph by this way may cause a increasing size of the graph.

The handling of the two-literal-clauses is executed during the construction of the initial connection graph (CONSTRUCT) but takes effect to the deduction steps during REFUTE.

TWO: RULES.MAXLEVEL

The value of this option is the maximal length of two-literal-clause-chains which are substituted by one link.

Possible Values

Natural Numbers
Default Value: 1

TWO: SUPPRESS.NO.RULES

This option controls the creation of intermediate result clauses if there are chains of two-literal-clauses which are longer than the maximal length.

Possible Values

T/Y/YES :	Intermediate result clauses are created and inserted in the graph
▸ NIL/N/NO:	The creation of intermediate clauses is suppressed.

Remark
With value N for this option the proof procedure becomes incomplete.

OPTIONS OF THE CATEGORY RED.I

This area contains options which influence the construction of the initial connection graph

RED.I: CLAUSE MULTIPLE.LITERALS

A clause containing some identical literals can be simplified by deletion of the multiple literals. Two literals are identical if they have

- the same sign
- the same predicate symbol
- equal term lists

(equal means here equal under a certain theory), or if they are connected by a R-link with ε-unifier caused by the two-rule-algorithm.

Example

Pafxfyz	Pffxyza

⇒

Pafxfyz

if P is symmetric and f is associative

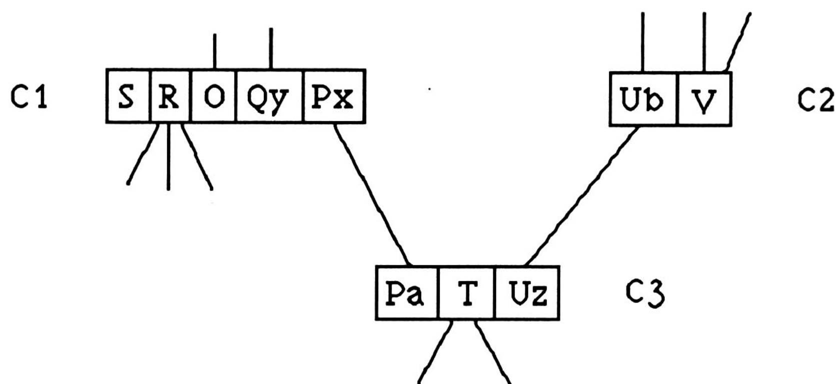
Possible Values

- T/Y/YES : Multiple Literals are removed
- NIL/N/NO : Clauses with multiple literals remain unchanged

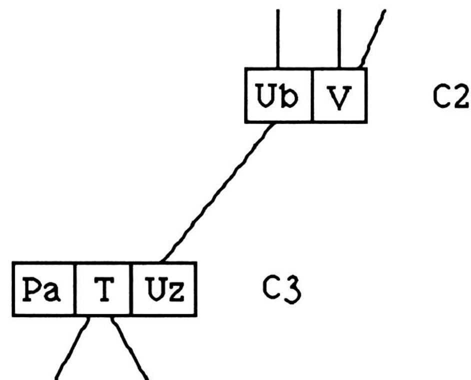
RED.I : CLAUSE.PURITY

This option offers the possibility to delete all clauses of the connection graph which cannot support the deduction of the empty clause, because they are "pure". A clause is pure, if and only if it contains a literal which is not connectecd by a R- or P-link. This means no deduction step is possible to reduce a pure clause to the empty clause, i.e. pure clauses cannot support the deduction of the empty clause

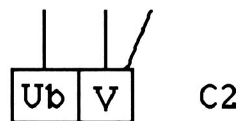
Example



The clause C1 is pure because the literal S is not connected by a R- or P-link and is therefore eliminated. All links connecting C1 to other clauses are also deleted:



This operation changes the clause C3 to a pure clause, causing the deletion of C3:



This example shows an effect which is caused by the reduction rules described in this and the following paragraph: Every deletion of a clause may cause further deletions of clauses and links (snowball effect)

Possible Values

‣ T/Y/YES : Pure clauses are deleted
NIL/N/NO: Pure clauses remain in the graph

Remark

This option should only be switched off for test purposes.

RED.I : CLAUSE.TAUTOLOGY

This option influences the treatment of tautological clauses. A tautological clause is always true under the actual theory.

a) A clause containing two literals which have
- different signs
- the same predicate symobl
- equal term lists

is a tautology

Example

1) | Pa | $\overline{P}$a |

2) | $\overline{R}$ab | Rba |

if R is symmetric

b) A clause containing a literal of the form t1 = t2 with t1 equal to t2 is a tautology

Example

1) | g(a) = g(a) |

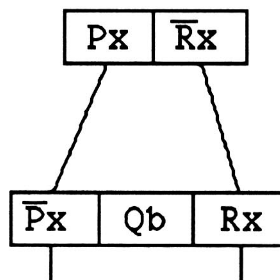
2) | f(a f(b c)) = f(f(a b) c) |

if f is associative

c) A clause containing two literals which
- are connected by a R-link with ε-unifier caused by the two-rule-algorithm

is a tautology

Example



Additional links created by the two-rule algorithm can also trigger other reduction rules. In the following we do not mention this explicitly.

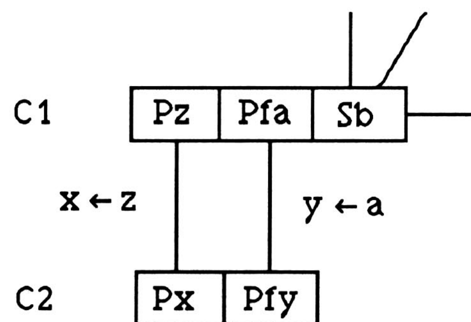
Possible Values

- ▸ T/Y/YES : Removal of tautological clauses
- NIL/N/NO: No Removal

RED.I : CLAUSE.SUBSUMPTION

This option controls the treatment of subsumed clauses in the initial graph. A clause C subsumes a clause D if $|C| \leq |D|$ and there exists a substitution σ that $\sigma(C) \subset D$. This inclusion must be bijective. If the deduction of the empty clause by D is possible it is also possible and in most cases even shorter by using the clause D. By this it seems useful to delete the subsumed clause (see also [Loveland-78], [Karl Mark Graph-84]).

Example



C1 can be deleted because it is subsumed by C2.

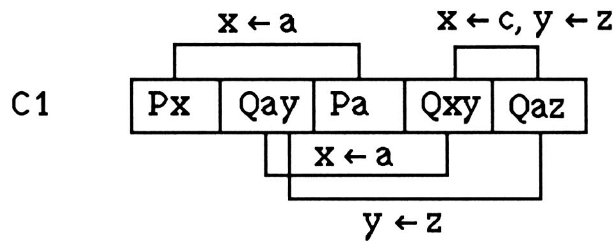
Possible Values

- ▸ T/Y/YES : Subsumed clauses are removed
- NIL/N/NO: No removal

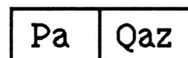
RED.I : REPL.FACTORING

A factor C may subsume its parent clause D and therefore D can be deleted. Instead of executing the factorization step and then deleting the subsumed clause C can be obtained by simply erasing the appropriate literal of the parent clause D, i.e. factorization and the application of the subsumption reduction rule can be grouped together to a macro graph operation.

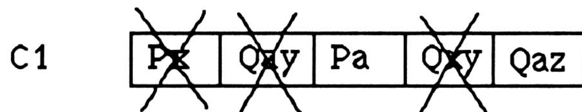
Example



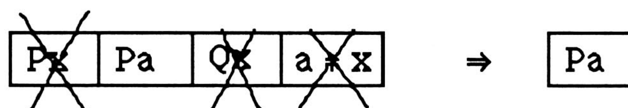
A factor of C1 is



C1 is subsumed by C2 and can be deleted. The instantiation and erasure of literals has the same effect



A possible generalization is the erasure of literals which become false under all theories by the instantiation step:



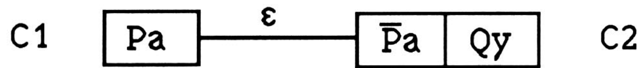
Possible Values

‣ T/Y/YES : switched on
NIL/N/NO: switched off

RED.I: CLAUSE . REPL. RESOLUTION

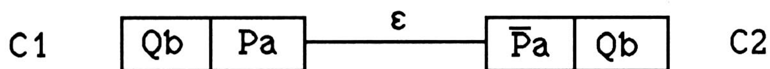
Just as a factor may subsume its parent, a resolvent may subsume one of its parents. As above (REPL.FACTORING) some deduction steps can be grouped together to a macro graph operation. The simplest case of this is the following:

a) Suppose we have a unit clause Pa and a clause ~Pa, ax

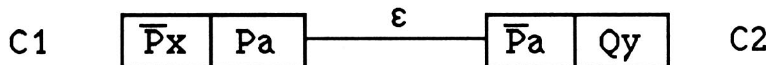


The result of the resolution step with subsequent subsumption can be obtained by simply erasing the literal ~Pa in clause C2.

b) One possible generalization is possible merging literals:



Here we still may just erase ~Pa.
c) Taking the instantiation process into account we can solve the following example by this method:



The literal ~Pa is substituted by the more general literal ~Px.

A further generalization possibility is the deletion of literals which become false in any interpretation by the instantiation process.

Possible Values

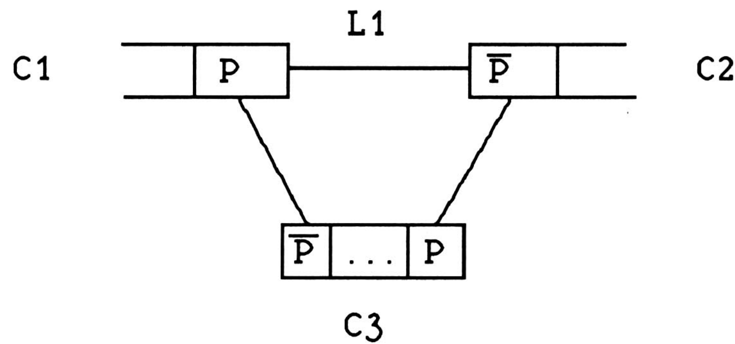
GENERALIZING: One of the resolution literals is substituted by a more general one of the other clause, as described in example c,

▸ SIMPLE: A Resolution literal is erased and possible merging, factoring and unit-resolution is done (as in example b)

UNIT: One of the parent clauses must be a unit clause (example a)

NIL/N/NO: Replacement Resolution is switched off.

The tautology reduction rule described so far is incomplete. For example suppose we have the following three clauses:



If the link L1 is missing, possible deduction steps can be lost by the deletion of the tautology C3. The existence of the link L1 is here termed as "link condition". For the other reduction rules subsumption, replacement factoring, replacement resolution similar link conditions exist. These link conditions make several additional checks necessary:

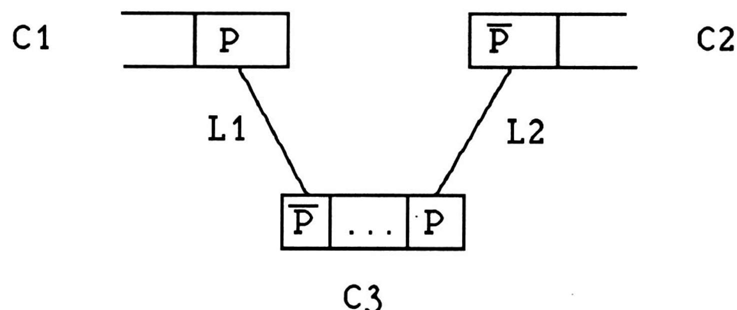
- after link insertion to check if a link condition is now fulfilled
- after link deletion to check if a link condition becomes now superfluous

The following options describe these additional checks.

RED.I : CLAUSE.TAUTOLOGY.RECHECK

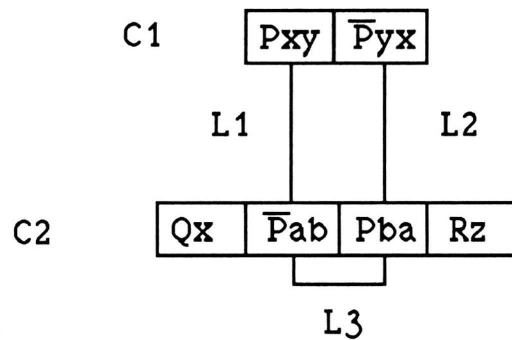
The adjustment of this option controls a renewed tautology check after link insertion and deletion. After link deletion the link condition can become superfluous.

Example



In this case the link condition is not fulfilled and the tautology C3 cannot be deleted. Deleting L1 or L2 causes the link condition to become superfluous and C3 can be deleted.
After link insertion a renewed check is also necessary because the inserted link can fulfill the link condition. If a link is inserted by the two-rule-algorithm a clause, which wasn't a tautology can now become a

tautology e.g.



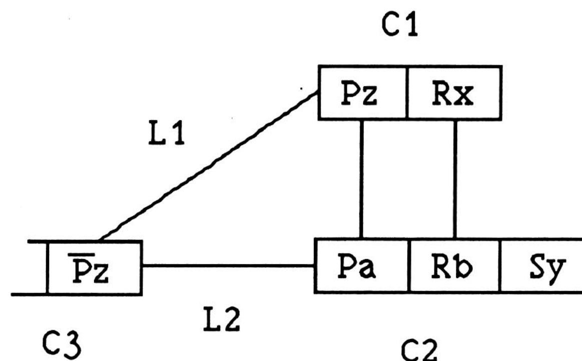
The two-rule-algorithm causes the insertion of L3 and C2 becomes a tautology.

Possible Values

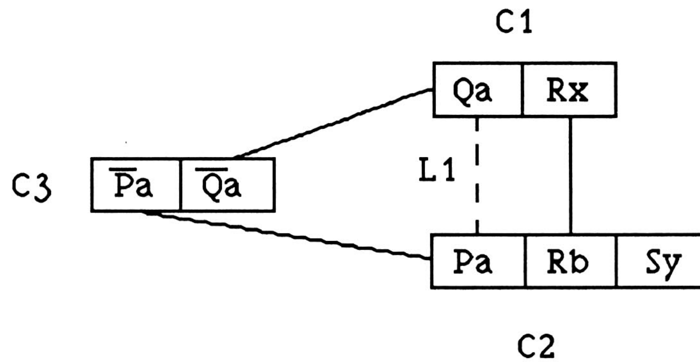
- ▸ PARTIAL/P : only clauses which are incident (i.e. directly connected) to the clauses the link is inserted (or deleted) between are checked
- NIL/N/NO : The option is switched off

RED.I: CLAUSE.SUBSUMPTION.RECHECK

The link condition for the subsumption rule is the following



The existence of L1 is the link condition for the subsumption rule. This option controls a renewed subsumption check after link deletion (the link condition is no longer necessary) and link insertion (the link condition becomes valid). The third case making a renewed check necessary is the insertion of a S-link by a two-literal-rule:



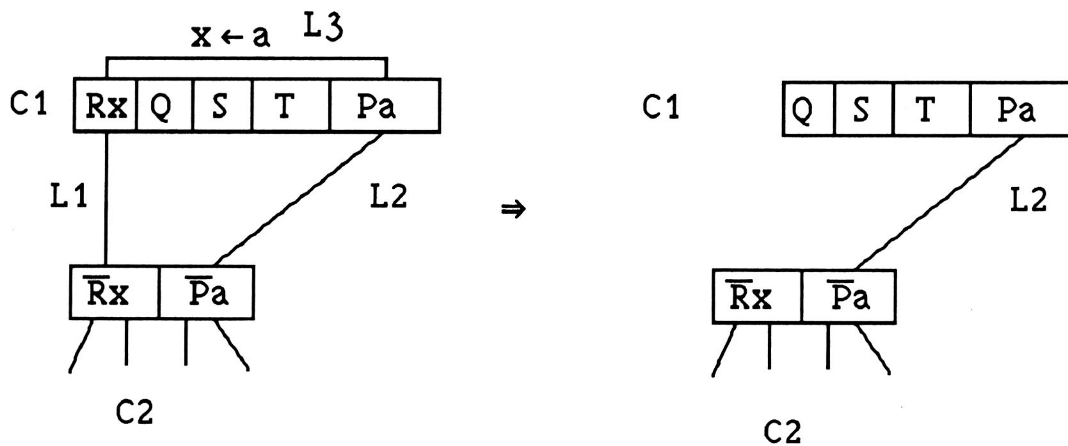
L1 is inserted by the two-literal-rule-algorithm and now C2 is subsumed by C1 and can be deleted.

Possible Values

- ▸ PARTIAL/P : only clauses which are incident (i.e. directly connected) to the clauses the link is inserted (or deleted) between are checked
- NIL/N/NO : The option is switched off

RED.I : REPLACEMENT.FACTORING.RECHECK

The adjustment of this option controls a renewed check for replacement factoring after link deletion or link insertion. For example suppose the following situation:



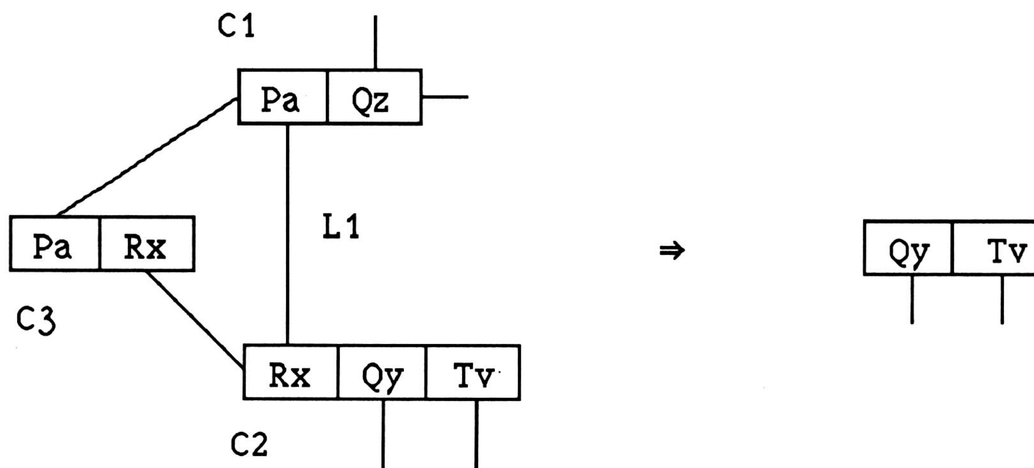
The link L3 is inserted by the two-rule-algorithm and makes a replacement factoring possible.

Possible Values

- ▸ T/J/JA : Replacement factoring recheck is switched on
- NIL/N/NEIN : switched off

RED.I : CLAUSE.REPL.RESOLUTION.RECHECK

After link deletion or link insertion it is necessary to check a second time for the link condition of replacement resolution. The adjustment of this option controls this renewed check for replacement resolution.
Suppose the following example:



L1 is inserted by the two-rule-algorithm. The clause <Qy, Tv> resulting by resolving upon link L1 would subsume C2. Therefore C2 can be replaced by <Qy, Tv>

Possible Values

- ▸ T/Y/YES : Replacement Resolution Recheck is switched on
- NIL/N/NO: switched off.

RED.I: CLAUSE. REWRITING

To make the axioms better readable (and to save time when typing the axioms) it is possible to abbreviate some complex terms. These abbreviations are replaced by the origin terms during construction of the initial graph. This replacement operation is controlled by this option.
There are two possibilities

1) $a = t$ with: a constant, t term and $a \notin t$
2) $f(x_1, \dots x_n) = t$ with: x_i variables with maximal domainsort, $f \notin t$, and t contains no other variables $\neq x_i$

Example

(i) sq (x) = times (x x)

(ii) x ≤ sq (x)

will be expanded to x ≤ times (x x)

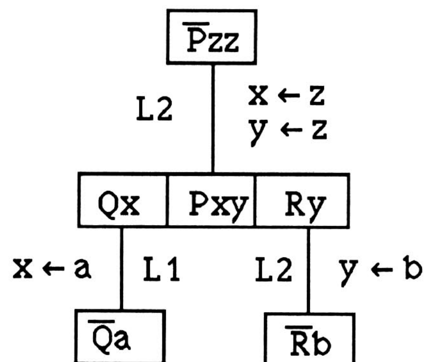
Possible Values

- ▸ T/Y/YES: Clause. Rewriting is switched on
- NIL/N/NO: Clause. Rewriting is switched off

RED.I: LINK. INCOMPATIBILITY

An R-link represents a possible deduction step and is marked with the most general unifier of the two literals connected by this R-link. Resolving upon an R-link makes instantiantions of some literals necessary. Such an instantiation can block further resolution possibilities. Two links are compatible if their unifiers do not contradict each other, i.e. resolution upon one link does not block resolving on the other link. Some links are simultaneously compatible if their substitutions don't contradict each other. A link can be deleted as incompatible, if not every other literal has at least one link so that they are simultaneously compatible.

Example



Link L1 ist compatible with L2 and L1 is compatible with L3, but L1, L2, L3 are not simultaneously compatible and L1 can therefore be deleted.

Possible Values

- ▸ T/Y/YES : Incompatible links are deleted.
- PARTIAL : Only one step purity check.
- NIL/N/NO: No removal of incompatible links.

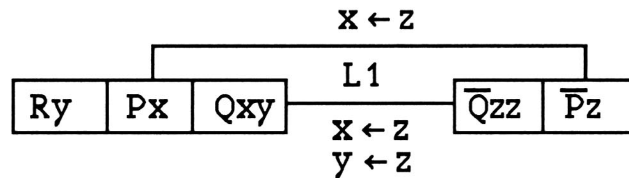
Remark: The check for link incompatibility is often very expensive and only a few links are removed. Therefore it seems very useful to choose only the "partial"-adjustment or even to switch this option off.

RED.I: LINK.TAUTOLOGY

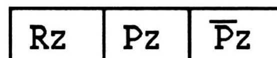
The resolvent of two clauses can be a tautology. Tautologies can be removed from the graph. Therefore it seems useful to remove links which would generate tautologies.

Examples

1)

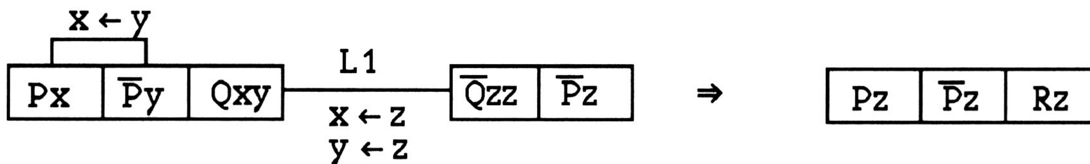


Resolving on L1 would generate the tautology



i.e. L1 can be removed from the graph.

2)



Remove L1 to prevent the generation of a tautology

3)

Removal of L1 is possible.

Possible Values

- T/Y/YES : Removal of tautology-generating links without Link-Condition check
- ▸ REMOVE-INHIBIT/RI: Removal of the links complying with the link condition. Inhibition of the others.
- INHIBIT/I : Inhibition of tautology-generating links
- REMOVE/R : Removal of the links complying with the link condition
- NIL/N/NO : No removal, no inhibition

RED.I: LINK. TAUTOLOGY. RECHECK

The adjustment of this option controls a renewed check for a link-tautologies after link insertion or link removal.

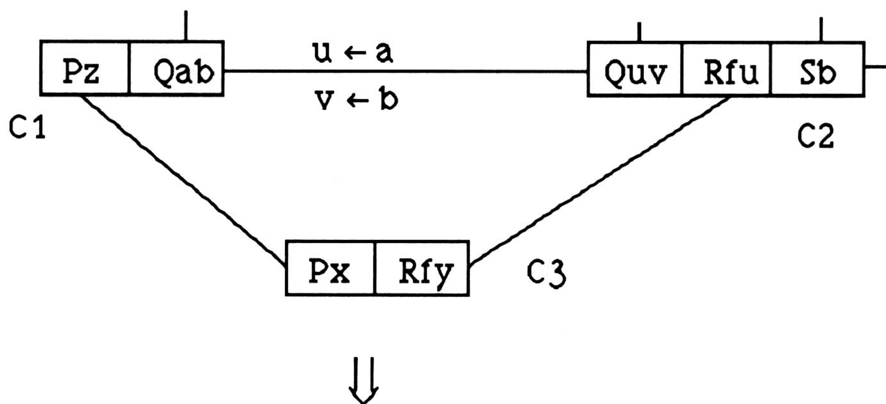
Possible Values

- T/Y/YES: The option is switched on
- ▸ NIL/N/NO: switched off

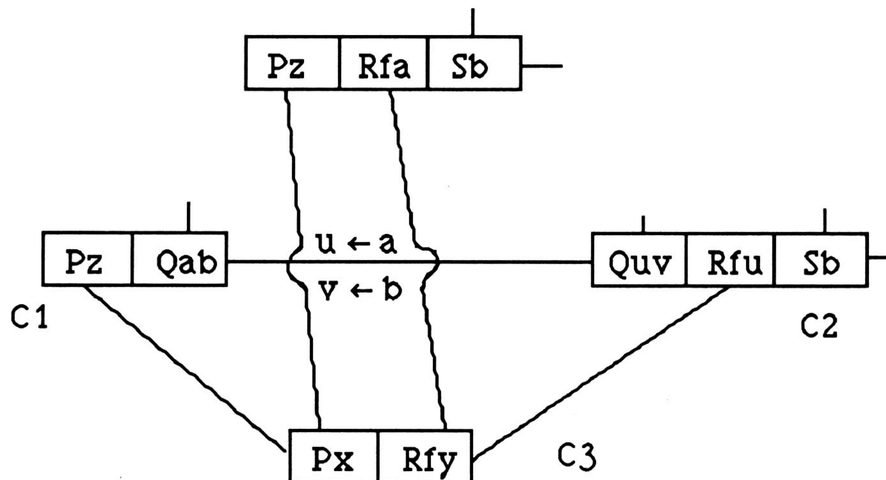
RED.I: LINK. SUBSUMPTION

Links which generate clauses, which are subsumed by other clauses can be removed. This removal operation is controlled by this option.

Example



⇓



C4 is subsumed by C3 and can be removed.

Possible Values

- T/Y/YES: Removal of the links without link condition check
- ▸ REMOVE-INHIBIT/RI: Removal of the links complying with the link condition. Inhibition of the others
- INHIBIT/I : Inhibition of the links
- REMOVE/R : Removal of the links complying with the link condition
- NIL/N/NO : No removal, no inhibition

RED.I: LINK. SUBSUMPTION. RECHECK

This option influences a renewed link subsumption check upon link removal or link insertion:

Possible Values

- T/Y/YES: switched on
- ▸ NIL/N/NO: switched off

Options of the category RED.D

This option-area consists of options for various reductionrules during deduction.

RED.D:CLAUSE.MULTIPLE.LITERALS

This option has the same effect as RED.I:CLAUSE.MULTIPLE.LITERALS.

Possible Values

▸ T/Y/YES:	option is switched on
NIL/N/NO:	option is switched off

RED.D:CLAUSE.PURITY

The option has the same effect as RED.I:CLAUSE.PURITY.

Possible Values

▸ T/Y/YES:	removal of the pure clauses
NIL/N/NO:	no removal

RED.D.:CLAUSE.TAUTOLOGY

Treatment of deduced tautology clauses.
Such clauses can be removed if no deduction possibility is lost by the removal (for precise explanation see RED.I:CLAUSE.TAUTOLOGY).

Possible Values

T/Y/YES	:	removal of the tautology clauses without link condition check.
▸ REMOVE-INHIBIT/RI	:	removal of the clauses where link condition is met, reinsertion and inhibition of creator links.
INHIBIT/I	:	removal of the clauses as well as reinsertion and inhibition of creator links
REMOVE/R	:	removal of the links complying with the link condition
NIL/N/NO	:	no removal, no inhibition

RED.D:CLAUSE.TAUTOLOGY.RECHECK

Renewed tautology check and treatment after insertion or removal of links (for precise explanation see RED.I:CLAUSE.TAUTOLOGY.RECHECK).

Possible Values

- T/Y/YES: switched on for insertion and removal
- Partial/P: switched on for insertion of incident links
- ▸ NIL: switched off

RED.D: CLAUSE.SUBSUMPTION.FORWARD

Treatment of deduced clauses which are subsumed by previous clauses.

Possible Values

- T/Y/YES : removal of the clauses without link condition check
- ▸ REMOVE-INHIBIT/R: in case of deduced subsumed clause, removal of the clause. If link condition is met reinsertion and inhibitation of creator link
- INHIBIT/I : removal of the clauses as well as reinsertion and inhibition of creator links
- REMOVE/R : removal of the clauses complying with the links condition
- NIL/N/NO : no removal, no inhibition

RED.D: CLAUSE.SUBSUMPTION.BACKWARD

Treatment of such clauses, which are subsumed by deduced new clauses.

Possible Values

- T/Y/YES: removal of all appropriate clauses without link condition check
- ▸ REMOVE: only removal of such clauses which comply with the link condition
- NIL/N/NO: the option is switched off

RED.D: CLAUSE.SUBSUMPTION.RECHECK

This option controls a renewed subsumption check, if subsumptions become possible respectively no longer possible by link insertion or removal

Possible Values

- T/Y/YES: switched on for insertion and removal of links
- ▸ REMOVE/R: removal of the clauses complying with the link condition
- NIL/N/NO: the option is switched off

RED.D: CLAUSE.REPL.FACTORING

The option has the same effect as RED.I:CLAUSE.REPL.FACTORING.

Possible Values
- ▸ T/YES/Y: switched on
- NIL/N/NO: switched off

RED.D: CLAUSE.REPL.FACTORING.RECHECK

The option has the same effect as RED.I:CLAUSE.REPL.FACTORING.RECHECK

Possible Values
- ▸ T/Y/YES: switched on
- NIL/N/NO: switched off

RED.D: CLAUSE.REPL.RESOLUTION

The option has the same effect as RED.I:CLAUSE.REPL.RESOLUTION

Possible Values
- GENERALIZING/G: one of the resolution literals is replaced by a more general one of the other clause
- ▸ SIMPLE/S: switched on, only for one step, without generalizing
- UNIT/U: switched on for unit partner only
- NIL/N/NO: switched off

RED.D: CLAUSE.REPL.RESOLUTION.RECHECK

The option has the same effect as RED.I:CLAUSE.REPL.RESOLUTION.RECHECK.

Possible Values
- ▸ T/YES/Y: switched on
- NIL/N/NO: switched off

RED.D: LINK.INCOMPATIBILITY

The option has the same effect as RED.I:LINK.INCOMPATIBILITY.

Possible Values
- ▸ T/YES/Y: removal of the links

NIL/N/NO: no removal

RED.D: LINK.TAUTOLOGY

The option has the same effect as RED.I:LINK.TAUTOLOGY

Possible Values
- T/Y/YES: removal of the links without link condition check
- ▸ REMOVE-INHIBIT/RI: removal of the links complying with the link condition. Inhibition of the others.
- INHIBIT/I: Inhibition of the links
- REMOVE/R: Removal of the links complying with the link condition
- NIL/N/NO: no removal, no inhibition

RED.D: LINK.TAUTOLOGY.RECHECK

The option has the same effect as RED.I:LINK.TAUTOLOGY.RECHECK.

Possible Values
- T/Y/YES: switched on for removal and insertion
- PARTIAL/P: switched on for insertion of adjacent links
- ▸ NIL/N/NO: switched off

RED.D: LINK.SUBSUMPTION

The option has the same effect as RED.I:LINK.SUBSUMPTION

Possible Values
- T/YES/Y: removal of links without link condition check
- ▸ REMOVE-INHIBIT/RI: removal of the links complying with the link condition. Inhibition of the others.
- INHIBIT/I: Inhibition of the links
- REMOVE/R: Removal of the links complying with the link condition.
- NIL/N/NO: no removal, no inhibition

RED.D: LINK.SUBSUMPTION.RECHECK

The option has the same effect as RED.I:LINK.SUBSUMPTION.RECHECK.

Possible Values
- T/Y/YES: switched on for removal and insertion
- Partial/P: switched on for insertion of adjascent links

▸ NIL/N/NO: switched off

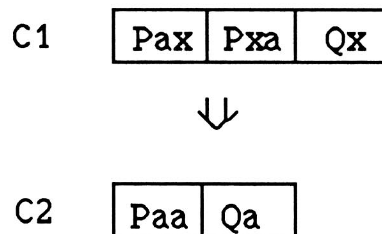
OPTIONS OF THE CATEGORY STR

This area consists of options, which direct the search-behaviour while looking for a proof.

FAC:INITIAL

This option controls factorizations in the initial graph

Example



Instantiation of the variable x causes the original clause C1 to be transformed into the shorter clause C2.

Possible Values

T/Y/YES	the option is switched on
▸ NIL/N/NO	the option is switched off

FAC:EACH.STEP

Factorizing after each deduction step

Possible Values

T/Y/YES	the option is switched on
▸ NIL/N/NO	the option is switched off

STR:RESOLUTION

Adjustment of the basic deduction strategy during proof search, which is activated only if the selection-module (chap.2) is not in control. Various classical refinement strategies like set-of-support, linear etc. are available and are simulated in the connection graph calculus by marking R and P-links as "active" or "passive".

Possible Values

Value	Description
BASIC-RESOLUTION/BASIC/B	All links are marked as active such that the selection module does a breadth first search.
▸ SET-OF-SUPPORT/SOS/S	Only if there is at least one theorem clause the Set-of-Support strategy will be applied, else the strategy is switched to basic.
UNIT-REFUTATION/UNIT/U	All links connected to unit clauses are activated, the other ones are marked as passive.
INDUCTION-SPECIAL/I-S	not yet fully implemented
linear/L/linear.axm#/linear.thm#/L.axm#/L.thm#	The first clause the linear strategy starts with (i.e. the "top clause") is either defined by using a concatenation of "linear." and the printname of the top clause as the strategy name, or is asked for by the system if one uses Str: Resolution = linear (in this case the system prints all clauses with their printname).
combined strategies:	
U - B	Unit - Basic,
U - SOS	- Set-of-Support
U- L/U-l.axm#/U-l.thm#	- linear.
	Unit Resolution prunes the search space considerably, but unfortunately it is not complete for all clause sets. The user can define strategies: U-SOS, U-B and U-L saying "if the clause set is horn renamable the unit-Resolution, else Set-of-Support", respec. Basic or linear strategy.

STR: E-RESOLUTION

This resolution strategy is provided for special equality proofs but is not yet fully implemented and therefore switched off.

Possible Values

Value	Description
T/Y/YES	the option is switched on
▸ NIL/N/NO	the option is switched off

STR:PARAMODULATION

By this option the required Paramodulation-Strategy can be chosen.

Possible Values

NONE	A basic setting of the option; as long as any active R-link is available, no paramodulation is performed.
▸ UNIT-ANCESTRY	Paramodulation between unit clauses
REWRITE	Paramodulation between unit clauses if the term complexity is decreasing.

STR: LINK.DEPTH

This option restricts the length of the deduction-steps performed at each clause (with respect to the initial clauses).
This link depth is defined as follows:

Links between clauses in the initial graph have the link depth d(L) = 0
For all other L: d(L) = 1 + max (d(L1), d(L2)) where L1 and L2 are the links which generated the two clauses connected by L.

Possible Values

positive integer	upper bound for link depth
▸ N/NO/NIL	no upper bound

STR: TERM.DEPTH

By this option an upper bound for the term nesting depth can be specified. This prevents deducing terms with depth greater than the specified one and thereby restricting the search-space.

Possible Values

positive integer	upper bound for term nesting depth
▸ N/NO/NIL	no upper bound

TERM: UNITS

The option controls processing of pattern leading to unit clauses found by the terminator

Possible Values

▸ T/Y/YES	the resolution steps necessary to produce the unit clauses proposed by the terminator are performed
NIL/N/NO	resolution possibilities leading to unit clauses found by the terminator are ignored.

TERM: ITERATIONS

This option influences the length of a path which will be processed by the terminator proceeding from each clause to detect a "terminator situation" (cf. chap.2).

Possible Values

▸ 0	no new unit clauses are generated and only the one level terminator situations can be found.
integer > 0	deeper level terminator situations can be found, but more time consuming.

remark: The number of term:iterations should not be chosen too high (normally max. 3-5) because the time for one deduction step increases explosively. However this option should be coordinated with STR:TERM.DEPTH to restrict the set of clauses to examine.

TERM:SET-OF-SUPPORT

Allows restricting the set of unit clauses examined by the terminator to detect a terminator situation.

Possible Values

▸ NIL/N/NO	no restriction of the unit clauses
T/Y/YES	the terminator only uses unit clauses of the set-of-support

TERM: BREADTH.FIRST

Allows switching the search strategy of the terminator to breadth first.

Possible Values

T/Y/YES	pure breadth first search
▸ NIL/N/NO	the basic search strategy of the terminator is used (similar to the usual linear strategies)

OPTIONS OF THE CATEGORY GEN

This area consists of several general options.

GEN: SPLITTING

This option allows partitioning of the theorem into several independent subproblems if the theorem contains conjunctions. (Reduction of a problem in several smaller subproblems) The option causes a transformation of the theorem in DNF and then the DNF is splitted.

Example

(Andrews example)

$[(\exists x\ \forall y\ Px = Py) \equiv ((\exists x\ Qx) \equiv (\forall y\ Py))]$
$\equiv$
$[(\exists x\ \forall y\ Qx = Qy) \equiv ((\exists x\ Px) \equiv (\forall y\ Qy))]$

Direct transformation into clausal normal form would generate thousands of clauses to be inserted in the graph. Splitting of the theorem produces 8 independent graphs with 58 clauses each which can be refuted by the MKRP.

Possible Values

NIL/N/NO	option is switched off
integer ≥ 0	switched on. maximal nesting depth up to which multiplication into DNF takes place in order to enable splitting.
T/Y/YES	switched on. Multiplication in all nesting depth.

▸ Default Value : 0

GEN: MANUAL.CONTROL

Allows the user to influence the proof. Links to be used for deduction can be chosen interactively by the user. The use of this option requires detailed knowledge of the internal structure of the MKRP and should therefore not be used.

Possible Values

T/Y/YES	the option is switched on
▸ NIL/N/NO	the option is switched off

GEN: MAXIMUM.STEPS

Allows to limit the number of deduction steps of a proof. When reaching the specified number of steps MKRP stops with the message "RESULT:FAILURE.ABORTED.MAXSTEPS".

Possible Values

▸ NIL	infinite
positive integer	maximum number of deduction-steps of a proof

Remark
This option is important especially if the prover runs in batch. If the specified time limit of a batch job is reached and no refutation has been found so far, the prover stops "abnormal", i.e. is not possible to create a listing of the deduction steps so far made. If one can approximate the maximal number of deduction steps, GEN:MAXIMUM.STEPS can be set to this number. After reaching the specified step the deduction process stops and it is now possible to create a listing.

GEN:GRAPH.SAVING

Enables the user to store the actual state of the graph after a certain number of deduction-steps in order to have the possibility to restart the deduction process at this state. It is useful to save the graph a few steps before GEN:MAXIMUM.STEPS is reached, so that the graph can be continued, if this seems promising.

Possible Values

▸ NIL	no effect

positive integer	number of deduction-steps between two savings of the graph

Remark: The option Gen:save.file allows specifying a file-name for the graph stored by the user.

GEN:SAVE.FILE

Allows to specify a file name on which the graph(s) (see GEN:GRAPH.SAVING) will be stored.

Possible Values

<File name>	the system saves the graph on this file
▸ NIL	a default name for the saved graph will be used

On a Symbolics the values of the options can be chosen using a mouse and a window oriented menu system.

6. Subsystem Commands

These commands control the actual subsystem configuration of the system to be used for the present run.
They are:

H[ELP] P[RINT] R[EMOVE] A[DD] OK V L[ISP]

H[ELP] prints a list of all available commands
H[ELP] <COM> explains the command <COM>

P[RINT] prints the actual system-configuration

R[EMOVE] <$SUBSYSTEM_1$ $SUBSYSTEM_2$ ... $SUBSYSTEM_n$>
removes the modules of
$SUBSYSTEM_1$ $SUBSYSTEM_2$ ... $SUBSYSTEM_n$
needed no more. (Except those of the basic <Atp-version>)

A[DD] <$SUBSYSTEM_1$ $SUBSYSTEM_2$ ... $SUBSYSTEM_n$>
loads the not yet existing modules of
$SUBSYSTEM_1$ $SUBSYSTEM_2$ ... $SUBSYSTEM_n$

OK leaves the system-configuration part

V T/Y/YES/OK turns the manual terminal control on
V NIL/N/NO turns if off again

L[ISP] calls INTERLISP. Return with 'OK'

Subsystems are not needed on the Symbolics, since the entire system is always loaded.

7. The Output Facilities

When the user sets the options for a proof, he can determine the output by specifying various parameters in the trace and protocal option areas.

7.1 Protocol

A protocol listing normally includes the following informations:

- the user's input: axioms, theorems, options
- preliminary transformation: axioms and theorems in clausal normal form and the system table of all symbols used
- proof steps: parent clause(s), type of operation, resulting clause unifier

A listing can be generated by calling the PROTOCOL module with a code file as parameter.

The information in the code file depends on the adjustment of the protocol options.

PR: PROTOCOL

This option controls the generation of a CODE-FILE.

Possible Values:

▸ T/Y/YES	Raw data for the protocol are written to a file
NIL/N/NO	No raw data for the protocol are written to a file

PR: INFIX.FORM

Infix form of input formulae in protocol

Possible Values

▸ T/Y/YES	switched on
NIL/N/NO	switched off

PR: PREFIX.FORM

Prefix form of input formulae in protocol

Possible Values

T/Y/YES	switched on
▸ NIL/N/NO	switched off

PR: OPTIONS

Values of proof options in protocol

Possible Values

T/Y/YES	switched on
▸ NIL/N/NO	switched off

PR: SYMBOLS

Symbol Table in Protocol

Possible Values

▸ T/Y/YES	switched on
NIL/N/NO	switched off

PR:NEW.PNAMES

The value of this option controls the generation of new pnames for clauses in protocol.

Possible Values

▸ T/Y/YES	New Pnames for Clauses in Protocol are generated
NIL/N/NO	Same clause names as in proof will be used (In the proof identical names may be used for logically different clauses)

PR: DIFFERENT.VARIABLES

The value of this option controls the generation of names for the variables in protocol.

Possible Values

T/Y/YES	Different names for all variables in protocol
▸ NIL/N/NO	Different clauses may contain the same variables

PR: DIRECT.PROOF

The value of this option influences which deduction steps are written to the protocol file.

Possible Values

▸ T/Y/YES	:	Only deduction steps necessary for the proof are written to the protocol
NIL/N/NO	:	All steps including unnecessary ones appear in the protocol

PR: LEFT.MARGIN

First position to be printed in each line
▸ Default Value : 0

PR: LINELENGTH

Number of characters per line
▸ Default Value : 120

7.2 Trace

For debugging purposes, the MKRP-system has special TRACE-functions, tracing the preprocessing or the deduction steps for instance. These functions can be invoked by setting trace options.

TR: PREPROCESSING

Trace of the intermediate results of the preprocessors.

Possible Values

T/Y/YES	switched on
▸ NIL/N/NO	switched off

TR: STEP.MODE

Trace of each deduction step

Possible Values

NIL/N/NO	No trace is done
I/IMP	Detailed implementational protocol of all changes (clauses and

	links)
L/LOG	Protocol of all changes in a more logical form (clauses)
▸ LR	same as under L, additionally variables are renamed.

TR: DUMP

Dump of the current graph after certain intervals

Possible Values

▸ NIL/N/NO	No dump is done
pos. integer	number of deduction-steps between two subsequent dumps

TR: CLAUSE.MODE

Format of the output of clauses if TR:DUMP is set

Possible Values

▸ NIL/N/NO	No output at all
I/IMPL	Output is implementational
L/LOG	Logical output
LR	same as under L, additionally variables are renamed

TR: LINK.MODE

Format of the output of links if TR:DUMP IS SET

Possible Values

NIL/N/NO	No output at all
▸ I/IMPL	Output is implementational

TR:TRACE.FILE

Output file for traces and dumps

Possible Values

▸ NIL/N/NO	No output at all
T/TERMINAL	Output on terminal
<FILE>	Output on <File>

TR. TERMINAL

Brief information about the proof displayed on terminal (in addition to the displayed statistics)

Possible Values

▸ NIL/N/NO	switched off
T/Y/YES	switched on

Remark
It is advisable to switch on on the terminal trace unless the proof is done in a batch job.

8. A Test Run

The following protocol lists a typical session: the first set of instructions is used to set up the database etc. The second set is the final output protocol.

Terminal session:

```
(IN)    DO ATP,C.COM
(OUT)   % BLS0500 PROGRAM 'LISPV4', VERSION '~~~' OF '81-09-29' LOADED.
(OUT)   ATP SYSTEM: MARKGRAF KARL REFUTATION PROCEDURE, UNI KAISERSLAUTERN
(OUT)   VERSION:  4-APR-85
(OUT)   PLEASE ENTER THE INFO.AND.REPAIR FILE NAME. THE STANDARD NAME IS:
(OUT)   $KINF4511.ATP.INFO.AND.REPAIR
(OUT)
(IN)    $KINF4511.ATP.INFO.AND.REPAIR
(OUT)
(OUT)
(OUT)   PRETTYDEF FORMAT ATP.INFO.AND.REPAIR CREATED 31-AUG-85 13:52:06
(OUT)   FILENAME: ATP.INFO.AND.REPAIR.01
(OUT)   ATP.INFO.AND.REPAIRCOMS
(OUT)
(OUT)   PRETTYDEF FORMAT ADD.PRESIMPLIFICATION CREATED 21-DEC-84 14:37:08
(OUT)   FILENAME: ADD.PRESIMPLIFICATION.01
(OUT)   ADD.PRESIMPLIFICATIONCOMS
(OUT)
(OUT)   PRETTYDEF FORMAT ADD.NORM CREATED 11-FEB-85 20:44:32
(OUT)   FILENAME: ADD.NORM.00
(OUT)   ADD.NORMCOMS
(OUT)
(OUT)   PRETTYDEF FORMAT ADD.TEMP.REDUCTION CREATED 27-SEP-85 15:06:41
(OUT)   FILENAME: ADD.TEMP.REDUCTION.00
(OUT)   ADD.TEMP.REDUCTIONCOMS
(OUT)
(OUT)   PRETTYDEF FORMAT ADD.TEMP.RED.DATASTRUCTURE CREATED 27-SEP-85 15:11:41
(OUT)   FILENAME: ADD.TEMP.RED.DATASTRUCTURE.00
(OUT)   ADD.TEMP.RED.DATASTRUCTURECOMS
(OUT)
(OUT)   PRETTYDEF FORMAT ADD.TEMP.TWO CREATED  2-AUG-85 14:45:20
(OUT)   FILENAME: ADD.TEMP.TWO.00
(OUT)   ADD.TEMP.TWOCOMS
(OUT)
(OUT)   PRETTYDEF FORMAT ADD.OPERATION CREATED 19-JUN-85 14:53:40
(OUT)   FILENAME: ADD.OPERATION.00
(OUT)   ADD.OPERATIONCOMS
(OUT)
(OUT)   PRETTYDEF FORMAT ADD.TEMP.PROTOCOL CREATED 26-JUN-87 12:17:34
(OUT)   FILENAME: ADD.TEMP.PROTOCOL.00
(OUT)   ADD.TEMP.PROTOCOLCOMS
(OUT)
(OUT)   PRETTYDEF FORMAT ADD.CONNECTIONGRAPH CREATED  1-AUG-85 20:04:44
(OUT)   FILENAME: ADD.CONNECTIONGRAPH.02
(OUT)   ADD.CONNECTIONGRAPHCOMS
(OUT)
(OUT)   PRETTYDEF FORMAT ADD.TEMP.MEMORY CREATED  7-AUG-85 15:18:38
```

```
(OUT)   FILENAME: ADD.TEMP.MEMORY.00
(OUT)   ADD.TEMP.MEMORYCOMS
(OUT)
(OUT)   @
(OUT)
(IN)    help
(OUT)   THE FOLLOWING OS-COMMANDS ARE AVAILABLE IN THIS SYSTEM:
(OUT)
(OUT)   V      H[ELP]        EX[IT]        O[PTIONS]       I[ND[UCTION]]
(OUT)   HC     L[ISP]        LO[GOFF]      S[UBSYSTEMS]
(OUT)                                   EP
(OUT)                                   FP
(OUT)                        ER                    CP
(OUT)                        FR
(OUT)                EC                 CR                    RP
(OUT)                FC
(OUT)   E[DIT]         C[ONSTRUCT]        R[EFUTE]         P[ROTOCOL]
(OUT)   F[ORMULA]
(OUT)
(OUT)   D[EFINE.]D[IRECTORY]              D[EFINE.]E[XAMPLENAME]

(OUT)
(OUT)   S[ORT.]G[ENERATION]
(OUT)
(OUT)   H[ELP]  <COM>      PRINTS AN EXPLANATION OF THE COMMAND <COM>.

(OUT)
(OUT)   @
(OUT)
(IN)    h c
(OUT)   C[ONSTRUCT]    [<PROBLEM FILE> [<GRAPH FILE> [<CODE FILE> [<COMMENT>
(OUT)   [<BATCH FILE> [<ATP VERSION>]]]]]]
(OUT)   CREATES A SET OF INITIAL GRAPHS FROM A PROBLEM DESCRIPTION.
(OUT)   <PROBLEM FILE> FILE CONTAINING THE PROBLEM DESCRIPTION. IF OMITTED,
(OUT)                  THE LAST CREATED ONE IS USED, IF SUCH A FILE EXISTS.
(OUT)   <GRAPH FILE>   FILE THE INITIAL GRAPHS ARE WRITTEN ON. IF OMITTED,
(OUT)                  A STANDARD FILE NAME IS USED.
(OUT)   <CODE FILE>    FILE, WHERE THE RAW-DATAS FOR THE PROTOCOL ARE
(OUT)                  WRITTEN ON.
(OUT)   <COMMENT>      IS INSERTED INTO THE PROOF PROTOCOL.
(OUT)   <BATCH FILE>   IF GIVEN, CAUSES THE CREATION OF A BATCH JOB USING
(OUT)                  THE GIVEN <ATP VERSION>  OR THE STANDARD VERSION,
(OUT)                  RESPECTIVELY.
(OUT)
(OUT)   @
(OUT)
(IN)    c thief.problem thief.graph thief.ccode "THIEF"
(OUT)   CONSTRUCT:     *******    CONSTRUCTION OF THE INITIAL GRAPHS.    *******
(OUT)   AXIOM   50  MSEC  LINKS:   0 +  0 -  0 CLAUSES:  1 + 1 - 0 STORE:156  10
(OUT)   AXIOM   63  MSEC  LINKS:   1 +  1 -  0 CLAUSES:  2 + 1 - 0 STORE:155  10
(OUT)   AXIOM   75  MSEC  LINKS:   3 +  2 -  0 CLAUSES:  3 + 1 - 0 STORE:155   9
(OUT)   AXIOM  101  MSEC  LINKS:   9 +  6 -  0 CLAUSES:  4 + 1 - 0 STORE:154   9
(OUT)   AXIOM  280  MSEC  LINKS:  18 +  9 -  0 CLAUSES:  5 + 1 - 0 STORE:153   9
(OUT)   AXIOM  228  MSEC  LINKS:  23 +  5 -  0 CLAUSES:  6 + 1 - 0 STORE:152   9
(OUT)   AXIOM  225  MSEC  LINKS:  33 + 10 -  0 CLAUSES:  7 + 1 - 0 STORE:151   9
(OUT)   AXIOM  194  MSEC  LINKS:  33 +  0 -  0 CLAUSES:  7 + 0 - 0 STORE:150   9
(OUT)   AXIOM  588  MSEC  LINKS:   7 +  1 - 27 CLAUSES:  4 + 1 - 4 STORE:146   9
```

```
(OUT)   AXIOM  500  MSEC  LINKS:   3 +  1 -  5 CLAUSES:  4 + 1 - 1 STORE:143   9
(OUT)   RED.SUC 43  MSEC  LINKS:   3 +  2 -  2 CLAUSES:  5 + 1 - 0 STORE:143   9
(OUT)   REFUTATION DETECTED
(OUT)   CONSTRUCT:     **********          THEOREM PROVED          **********
(OUT)
(OUT)   @
(OUT)
(IN)    ex
(IN)    DO ATP,P.COM
(OUT)   %  BLS0500 PROGRAM 'LISPV4', VERSION '~~~' OF '81-09-29' LOADED.
(OUT)   ATP SYSTEM: MARKGRAF KARL REFUTATION PROCEDURE, UNI KAISERSLAUTERN
(OUT)   VERSION:  4-APR-85
(OUT)   PLEASE ENTER THE INFO.AND.REPAIR FILE NAME. THE STANDARD NAME IS:
(OUT)   $KINF4511.ATP.INFO.AND.REPAIR
(OUT)
(IN)    $KINF4511.ATP.INFO.AND.REPAIR
(OUT)
(OUT)
(OUT)   PRETTYDEF FORMAT ATP.INFO.AND.REPAIR CREATED 31-AUG-85 13:52:06
(OUT)   FILENAME: ATP.INFO.AND.REPAIR.01
(OUT)   ATP.INFO.AND.REPAIRCOMS
(OUT)
(OUT)   PRETTYDEF FORMAT ADD.TEMP.MEMORY CREATED  7-AUG-85 15:18:38
(OUT)   FILENAME: ADD.TEMP.MEMORY.00
(OUT)   ADD.TEMP.MEMORYCOMS
(OUT)
(OUT)   PRETTYDEF FORMAT ADD.TEMP.PROT.PREPARE CREATED 30-JUN-87 14:11:30
(OUT)   FILENAME: ADD.TEMP.PROT.PREPARE.08
(OUT)   ADD.TEMP.PROT.PREPARECOMS
(OUT)
(OUT)   PRETTYDEF FORMAT ADD.SYST.PROT.DATASTRUCTURE CREATED 31-AUG-85 12:41:00
(OUT)   FILENAME: ADD.SYST.PROT.DATASTRUCTURE.00
(OUT)   ADD.SYST.PROT.DATASTRUCTURECOMS
(OUT)
(OUT)   @
(OUT)
(IN)    p thief.ccode thief.list
(OUT)   PROTOCOL:    THE PROTOCOL WAS CREATED ON FILE THIEF.LIST.00

(OUT)
(OUT)   @
(OUT)
(IN)    ex
```

Final output protocol:

```
*********************************************************************************
*                                                                               *
*   ATP-SYSTEM :  MARKGRAF KARL REFUTATION PROCEDURE, UNI KAISERSLAUTERN        *
*                                                                               *
*      VERSION : 4-APR-85                                                       *
*         DATE : 8-JUL-87  13:39:58                                             *
*                                                                               *
*********************************************************************************


**************************************************************************
 EDIT:     AXIOMS AND THEOREMS EDITED:    6-JUL-87 15:23:12
**************************************************************************

*************************************************************************

 CONSTRUCT:   THIEF

*************************************************************************




FORMULAE GIVEN TO THE EDITOR
============================


AXIOMS    : * THERE ARE THREE SUSPECTS TO HAVE STOLEN A SILVER RING IN A HOTEL,
            * LUCKY, WILLIE, AND JACKY. ALL THREE OF THEM ARE KNOWN TO BE UNABLE
            * TO MAKE THREE CONSECUTIVE STATEMENTS WITHOUT LYING.
            * WILLIE: I AM NO THIEF. ALL OF LUCKY'S STATEMENTS ARE FALSE.
            * LUCKY:  I'M INNOCENT. I HAVE NEVER BEEN IN A HOTEL.
            *         WILLIE IS THE THIEF.
            * WILLIE: I AM NO THIEF. ALL OF LUCKY'S STATEMENTS ARE FALSE.
            *         JACKY IS INNOCENT
            * JACKY:  I DIDN'T STEAL ANYTHING. LUCKY WAS IN THE HOTEL. AT
            *         LEAST ONE OF LUCKY'S STATEMENTS WAS TRUE.
            * GIVEN THAT THERE WAS ONLY ONE THIEF, WHO IS IT?
            THIEF(WILLIE) OR THIEF(LUCKY) OR THIEF(JACKY)
            NOT(THIEF(WILLIE) AND THIEF(LUCKY) OR THIEF(WILLIE) AND THIEF(JACKY)
              OR THIEF(LUCKY) AND THIEF(JACKY))
            ALL X THIEF(X) IMPL IN.HOTEL(X)
            NOT(NOT THIEF(LUCKY) AND NOT IN.HOTEL(LUCKY) AND THIEF(WILLIE))
            NOT(NOT THIEF(WILLIE) AND THIEF(LUCKY) AND IN.HOTEL(LUCKY) AND NOT
              THIEF(WILLIE) AND NOT THIEF(JACKY))
            NOT(NOT THIEF(JACKY) AND IN.HOTEL(LUCKY) AND(NOT THIEF(LUCKY) OR NOT
              IN.HOTEL(LUCKY) OR THIEF(WILLIE)))

THEOREMS  : THIEF(JACKY)
```

CONSTANTS :
===========

NAME	SORT		ATTRIBUTES
WILLIE	ANY		
LUCKY	ANY		
JACKY	ANY		

PREDICATES :
============

NAME	DOMAIN		ATTRIBUTES
TRUE			DEFINED
FALSE			DEFINED
THIEF	ANY		
IN.HOTEL	ANY		

SET OF AXIOM CLAUSES RESULTING FROM NORMALIZATION
===

```
* A1   :   - THIEF(WILLIE)   - THIEF(LUCKY)
* A2   :   - THIEF(WILLIE)   - THIEF(JACKY)
* A3   :   - THIEF(LUCKY)   - THIEF(JACKY)
* A4   :   ALL X:ANY  - THIEF(X)   + IN.HOTEL(X)
* A5   :   + THIEF(WILLIE)   + THIEF(LUCKY)   + THIEF(JACKY)
* A6   :   + THIEF(LUCKY)   + IN.HOTEL(LUCKY)   - THIEF(WILLIE)
  A7   :   + THIEF(JACKY)   - IN.HOTEL(LUCKY)   + THIEF(LUCKY)
  A8   :   + THIEF(JACKY)   - IN.HOTEL(LUCKY)   + IN.HOTEL(LUCKY)
* A9   :   + THIEF(JACKY)   - IN.HOTEL(LUCKY)   - THIEF(WILLIE)
* A10  :   + THIEF(WILLIE)   - THIEF(LUCKY)   - IN.HOTEL(LUCKY)   + THIEF(WILLIE)
           + THIEF(JACKY)
```

OPERATIONS ON AXIOMS
====================

```
A6,1 & A1,2     -->  * R1  : + IN.HOTEL(LUCKY)   - THIEF(WILLIE)   - THIEF(WILLIE)
R1 2=3          -->  * D2  : + IN.HOTEL(LUCKY)   - THIEF(WILLIE)
A9,2 & D2,1     -->  * R3  : + THIEF(JACKY)   - THIEF(WILLIE)   - THIEF(WILLIE)
R3 2=3          -->  * D4  : + THIEF(JACKY)   - THIEF(WILLIE)
D4,1 & A2,2     -->  * R5  : - THIEF(WILLIE)   - THIEF(WILLIE)
R5 1=2          -->  * D6  : - THIEF(WILLIE)
A5,1 & D6,1     -->  * R7  : + THIEF(LUCKY)   + THIEF(JACKY)
A10 1=4         -->  * D8  : + THIEF(WILLIE)   - THIEF(LUCKY)   - IN.HOTEL(LUCKY)   +
                             THIEF(JACKY)
```

```
D8,4 & A3,2      -->  * R9  : + THIEF(WILLIE)  - THIEF(LUCKY)  - IN.HOTEL(LUCKY)  -
                                THIEF(LUCKY)
R9 2=4           -->  * D10 : + THIEF(WILLIE)  - THIEF(LUCKY)  - IN.HOTEL(LUCKY)
D10,3 & A4,2     -->  * R11 : + THIEF(WILLIE)  - THIEF(LUCKY)  - THIEF(LUCKY)
R11 2=3          -->  * D12 : + THIEF(WILLIE)  - THIEF(LUCKY)
D12,1 & D6,1     -->  * R13 : - THIEF(LUCKY)
R7,1 & R13,1     -->  * R14 : + THIEF(JACKY)


SET OF THEOREM CLAUSES RESULTING FROM NORMALIZATION
===================================================


* T11  : - THIEF(JACKY)


INITIAL OPERATIONS ON THEOREMS
==============================


T11,1 & R14,1   --> * R15 :  []


--------------------------------------------------------------------------------


  Q. E. D.
STOP
```