Discovering Inequality Conditions in the Analytical Solution of Optimization Problems

Paul A. Strooper
Bruce W. Char
Department of Computer Science

Alan R. Macnaughton
School of Accountancy

Research Report CS-87-18
July 1987

Discovering Inequality Conditions in the Analytical Solution of Optimization Problems

Paul A. Strooper, Bruce W. Char[1]

Department of Computer Science
University of Waterloo, Waterloo Ontario, Canada N2L 3G1

Alan R. Macnaughton

School of Accountancy
University of Waterloo, Waterloo Ontario, Canada N2L 3G1

ABSTRACT

Most problems in economics can be described as optimization problems of some kind. Numerical optimization techniques usually cannot be used to solve these problems because of the presence of symbolic parameters. This paper describes a theorem prover which will allow the user to deduce certain inequalities which must necessarily be true for the optimal solution. If the inequalities are simple enough, the optimal solution can be found in terms of the symbolic parameters.

1. Introduction

This paper describes the implementation of a theorem prover which can prove theorems for a list of inequalities and/or Kuhn-Tucker conditions. These conditions typically arise from analytical optimization problems where we are trying to optimize a function subject to several inequality or equality constraints ([3], [5] and [6]). The theorem prover can be used to deduce new inequalities which must hold given the constraints and Kuhn-Tucker conditions. In the case where these conditions are necessary and sufficient conditions these theorems can help us find the optimal solution.

For example, suppose we want to minimize:

$$ga - ea\,(na + 1 + ca) + gb - eb\,(nb + 1 + cb)$$

subject to the following constraint functions:

$$\begin{aligned} ca &\leq y - 1 \\ cb &\leq y - 1 \\ ca + cb &\leq y \\ ea &= ga \,/\, (\, y + na \,) \\ eb &= gb \,/\, (\, y + nb \,) \end{aligned}$$

where ea, eb, y, na, nb, ga and gb are the parameters and ca and cb are the choice variables. Then we can use the theorem prover to show that:

if $ea > eb$ then $ca = y - 1$, and $cb = 1$
if $eb > ea$ then $ca = 1$, and $cb = y - 1$

which gives us the solution to the optimization problem for two separate cases. More details of this example are given in the first appendix.

[1] This work is supported in part by grant A5471 of the Natural Science and Engineering Council of Canada.

In this paper, we first explain how a theorem prover can be used to help us solve certain types of analytical optimization problems. We then give an overview of how the theorem prover works and some of the implementation decisions made. The fourth section discusses the internal format that was used to represent the inequalities. A detailed description is given in sections 5 and 6 on how to use the theorem prover and of all the steps which can be performed by it. Section 7 gives a proof of the fact that the theorem prover always terminates. At this point we also discuss the restrictions this termination property imposes on the steps which can be performed, and on any new steps which could be implemented in the future. Finally, we discuss the performance of the theorem prover on problems it was tried on.

The theorem prover is implemented in *wup* (Waterloo Unix Prolog [4]). Throughout the paper Prolog predicates are shown in bold print, and all variables which have to be specified by the user are shown in italics.

2. Using a Theorem Prover on Analytical Optimization Problems

Standard references on optimization such as [3] discuss how we can calculate the Kuhn-Tucker conditions for the problem of optimizing a function of several choice variables subject to several inequality constraints. Given certain restrictions on the objective and constraint functions, these Kuhn-Tucker conditions are necessary and sufficient for optimality. If this is the case they can be used to find the solution. If the Kuhn-Tucker conditions are not necessary and sufficient conditions for the optimal solution, we can still use the theorem prover to obtain new expressions which are mathematical consequences of the original conditions. These expressions could illuminate the nature of the eventual solution.

Both the function to be optimized and the constraints can contain symbolic parameters. If we are given numeric values for these parameters we can use numerical optimization techniques to find the optimal values. However, if we can use the Kuhn-Tucker conditions and other constraints to express the solution in terms of these symbolic parameters we can simply express the optimal value in parameterized form. Our theorem prover derives new expressions which follow from the given Kuhn-Tucker conditions. If these expressions allow us to express the values of the variables in terms of the symbolic parameters of the system, we have in fact used the theorem prover to solve the optimization problem analytically.

An example of such an application is given in [6], where such theorems are used to develop simple decision rules for designating properties as principal residences for certain years in order to minimize the tax on capital gains for those properties. For one of the problems listed in that paper, 35 theorems (see [7]) were needed to find the optimal solution. Our theorem prover was successfully used to help check the proofs for all these theorems (The output produced for all these theorems by the theorem prover can be found in the second appendix of this paper). In the first appendix we show how we can use the theorem prover to solve one of the other problems discussed in [6]. For this problem we only need 4 theorems to find the optimal solution.

We expect our theorem prover would generally be useful as a tool for helping the user to solve the optimization problem. We do not expect that the system will always, or even, typically find the optimal solution automatically. It can be used to prove several results, which will have to be combined to find the actual solution. It is the user's responsibility to extract the helpful results and to combine them in such a way that the solution can be found[2].

In [3] and [5] it is explained how the Kuhn-Tucker conditions are usually only used to solve analytical optimization problems of two or three variables and constraints, since the complexity of the problem grows rapidly as the number of variables and constraints increases. The problem discussed in [7] has eight variables, and eight inequality constraints. Our theorem prover thus provides us with a significant increase in the size of problems we can reasonably deal with. It can be used to either check the (often tedious) proofs of existing theorems (as was done for the theorems in [7]), or to prove new theorems.

---

[2] Often the intermediate results are easy to find once the economic meaning of the variables and parameters are known. Experience in dealing with the Kuhn-Tucker conditions will certainly also help the user in determining what can be used and what not.

3. Overview of the Theorem Prover

3.1. Implementation Considerations

The proofs encountered in [7] emphasized the use of algebraic inferences, rather than logical ones. By this we mean that the hypotheses and conclusions of most proofs were a conjunction of several inequalities, while the rules used to deduce the conclusions were of an algebraic nature (such as the addition of inequalities, the division of an inequality by a variable or integer, etc.). This means it would not be advantageous to implement a resolution theorem prover (see [1]) in this case. Another difference between our situation and those for the other non-resolution theorem provers in [1] is that in our case all the occurrences of the symbolic variables are free, i.e. there are no quantifiers in any of the theorems. A final implementation consideration is that we would like our theorem prover to prove the theorems in the same way as in which humans would prove them, since the proofs could then be amenable to further economic insight and interpretation (often the proof of one theorem yields insights which will assist the user in developing proofs of similar theorems).

These considerations made us decide to implement the theorem prover as a forward theorem prover. For a particular proof, the user specifies the assumptions, the desired conclusions and possibly some other theorems that can be used inside the proof. The use of the theorem prover is also restricted to multivariate polynomials in the variables[3].

Since both the assumptions and conclusions are conjunctions of inequalities, the user is responsible for dealing with the logic of the proofs. For example, in the appendix we show how we can prove

$$\mathrm{ca} + \mathrm{cb} = \mathrm{y}$$

by proving the following two theorems

$$\text{if } \mathrm{c3} = 0 \text{ then } \mathrm{ca} + \mathrm{cb} = \mathrm{y}$$
$$\text{if } \mathrm{c3} > 0 \text{ then } \mathrm{ca} + \mathrm{cb} = \mathrm{y}$$

and using the fact that $\mathrm{c3} \geq 0$. This is an example of a proof by cases. For such proofs the user is responsible for finding the two cases and the theorem prover can be used to give the actual proof for each case.

3.2. The Form of the Kuhn-Tucker Conditions

Given a function $f(x_1,\ldots,x_n)$ of non-negative variables $x_1,\ldots,x_n$ which has to be minimized, and a set of constraint functions $g^1(x_1,\ldots,x_n)\leq 0$ to $g^m(x_1,\ldots,x_n)\leq 0$, we can calculate the Kuhn-Tucker conditions as follows. First of all we define the Lagrangian function Z:

$$Z(x_1,\ldots,x_n,\gamma_1,\ldots,\gamma_m)=f(x_1,\ldots,x_n)+\sum_{j=1}^{m}\gamma_j\left(g^j(x_1,\ldots,x_n)\right).$$

This function contains choice variables $(x_1,\ldots,x_n)$, newly introduced Lagrangian multipliers $(\gamma_1,\cdots\gamma_m)$, and possibly some symbolic parameters of the system[4]. The Kuhn-Tucker conditions for Z consist of the following (in)equalities:

$$x_i \geq 0, \text{ and } \gamma_j \geq 0 \qquad \text{for } 1 \leq i \leq n, \text{ and } 1 \leq j \leq m$$
$$\frac{\partial Z}{\partial x_i} \geq 0, \text{ and } x_i \frac{\partial Z}{\partial x_i} = 0 \qquad \text{for } 1 \leq i \leq n$$

---

[3] In the current implementation we only handle integer coefficients, but this can easily be changed so that we can deal with floating point numbers. Its use can also be extended to include functional expressions by creating new variables for each expression (for example, the inequality $\mathrm{x} * \cos(\mathrm{y}) \leq \sin(\mathrm{y})$ can be expressed as $\mathrm{x} * \mathrm{y1} \leq \mathrm{y2}$, where the user will know that $\mathrm{y1} = \cos(\mathrm{y})$ and $\mathrm{y2} = \sin(\mathrm{y})$).

[4] The theorem prover treats parameters, choice variables and Lagrange multipliers in exactly the same way, as they all are symbolic parameters (unknowns) to the system.

$$\frac{-\partial Z}{\partial \gamma_j} \geq 0, \text{ and } \gamma_j \frac{\partial Z}{\partial \gamma_j} = 0 \qquad \text{for } 1 \leq j \leq m$$

We can write the above conditions as a set of non-negativity constraints on the choice variables and Lagrange multipliers, and a set of complementary slackness conditions. Each slackness condition consists of a pair of (in)equalities of the form:

$$Expr \geq 0, \text{ and } Expr * Var = 0$$

where *Expr* is a polynomial expression involving the choice variables and the parameters of the system, and *Var* is either one of the choice variables, or a Lagrange multiplier. Each of the theorems we are trying to prove should be able to use all of these Kuhn-Tucker conditions as well as any other relevant constraints (such as constraints on the parameters).

3.3. Outline of the Theorem Prover

The prover starts with the Kuhn-Tucker conditions, any constraints existing on the parameters, and the inequalities in the hypothesis for the proof. Throughout the proof, the prover attempts to use this information to deduce new inequalities. The theorem prover halts when all the inequalities in the conclusion are proven, when a contradiction is found, or when it cannot deduce any new inequalities which it deems *useful* (the notion of usefulness will be explained when we discuss all the possible steps the theorem prover can perform). As we will show later, the prover is guaranteed to stop eventually, as it will end up in the third situation if it does not end up in either of the first two.

4. Internal Representation of Inequalities

Throughout the system we have to manipulate algebraic inequalities. We want to be able to compare inequalities so that we can check for contradictions, redundant inequalities, etc. In order to do this we devised our own internal representation for these inequalities (Prolog has no built-in representation for them). To accomplish this we define a canonical form for each of the inequalities the system can deal with, which can be obtained through the following steps[5]:

(1). All polynomials in the inequalities are expanded into sums of monomials. Each monomial is the product of variables and an integer, or just an integer.

(2). We transform inequalities of types $<$ and $\leq$ into inequalities of types $>$ and $\geq$ respectively by multiplying them by -1.

(3). We place the integer term to the right of the (in)equality sign (this integer part can be 0) and all other monomials to the left. Now each monomial to the left of the inequality sign contains at least one symbolic variable.

(4). Each monomial is represented as a list, consisting of the integer coefficient (even if the coefficient is 1 or -1) followed by the variables, ordered lexicographically.

(5). The whole inequality is represented as a list. The first element of this list is the type of the inequality (*g*, *ge*, or *e* standing for the types $>$, $\geq$, and $=$ respectively), the second element is the integer term, and all other elements in the list are the monomial-lists, ordered lexicographically.

(6) We require that for equalities the integer part must be greater than or equal to zero, and if it is zero that the integer part of the first monomial must be strictly positive. This way we have a canonical form for all the (in)equalities we handle.

Some examples of inequalities and their internal format are:

$$\begin{array}{rcl} 2 + y * (2 - x + y) < 0 & \rightarrow & [g \;\; -2 \;\; [1 \; x \; y] \;\; [-2 \; y] \;\; [-1 \; y \; y]] \\ x \leq a & \rightarrow & [ge \;\; 0 \;\; [1 \; a] \;\; [-1 \; x]] \\ x = y + 3 & \rightarrow & [e \;\; 3 \;\; [1 \; x] \;\; [-1 \; y]] \\ x = y - 3 & \rightarrow & [e \;\; 3 \;\; [-1 \; x] \;\; [1 \; y]] \end{array}$$

We can now compare inequalities directly in Prolog by checking if two inequalities have the same non-integer part using the built-in pattern matching mechanism. Once we have found two inequalities with the

[5] Since the internal format is sometimes hard to derive by hand, a predicate has been created to transform inequalities from an intermediate human-readable format to the required internal format. This predicate can be used as a front-end to the theorem prover.

same non-integer parts, we can compare the types of the inequalities and the integers to check for contradictions, redundant inequalities, etc. Using this representation we can also add inequalities together in an efficient manner since the monomials themselves are represented in a unique way and since they are also ordered in a unique way.

5. Using the Theorem Prover

Before using the theorem prover, all the Kuhn-Tucker conditions and other inequality constraints have to be defined to the system through two Prolog predicates. The **kt_cond** predicate is used to specify the complementary slackness conditions. The **hyp** predicate is used to specify all the other (in)equality constraints, such as the remaining Kuhn-Tucker conditions (non-negativity of choice variables and Lagrange multipliers) and any constraints on the parameters. For example, we can let the system know that $x > 2$ is a constraint by stating the fact:

hyp([g 2 [1 x]]);

As was mentioned earlier, the complementary slackness conditions each consist of a pair of (in)equalities of the form:

$$Expr \geq 0, \text{ and } Expr * Var = 0$$

where *Expr* is some expression and *Var* is a variable (or Lagrange multiplier). Such a condition can be stored by stating the fact:

kt_cond([*Ineq Eq*]);

where *Ineq* is the inequality $Expr \geq 0$ in its canonical form, and *Eq* is the equality $Var = 0$ in its canonical form. For example, the complementary slackness condition consisting of $x + y \geq 0$ and $u * (x + y) = 0$ can be expressed by the fact:

kt_cond([[ge 0 [1 x] [1 y]] [e 0 [1 u]]]);

Note that the first part of this list is an actual inequality of the system (and is treated as such throughout the system), whereas the first and second part have to be combined to form the actual second equality of the complementary slackness condition. This was done so that we can decompose these complementary slackness conditions easily, as is explained in the next section.

Any theorems that will be used later on in other proofs have to be defined using the **theorem** predicate, which allows the user to give the theorem a name and to define its hypothesis and conclusion. For example, to define a theorem by the name of thm1 to the system with hypotheses $x > 0$ and $x + y = 3$ and conclusion $2x - z \geq 0$ we state the fact:

theorem(thm1 [[g 0 [1 x]] [e 3 [1 x] [1 y]]] [[ge 0 [2 x] [1 z]]]);

The theorem prover can now be invoked using the following call:

?**proof**(*AL TL CL*);

where *AL* is the list (possibly empty) of additional assumptions, *TL* is a list (possibly empty) of theorems which can be used in this proof (these have to be defined using the **theorem** predicate), and *CL* is the list (possibly empty) of desired conclusions for this proof.

6. Steps Performed by the Prover

After the **proof** predicate has been called, the starting assumptions for the theorem are printed out. Then the assumptions are added to the list of inequalities which can be used for all proofs. Whenever new inequalities are added to the system, or existing ones are modified, we make sure that this does not introduce any contradictions or redundant inequalities. If the former happens we stop the proof after printing out the contradiction. In the latter case we simply eliminate the redundant inequality. At each stage of the theorem prover we try to modify the list of inequalities and complementary slackness conditions.

If there is a set of conclusions specified for the given theorem, we check at the beginning of each stage if all the inequalities in the conclusion hold. If this is the case we stop. In the other cases (i.e. either no conclusions were specified, or not all the conclusions were met), we try to apply one of the steps as discussed in the next sections. If any of these steps is successful, we add the newly obtained inequalities to the list and move on to the next stage, repeating the above steps. When none of these steps are successful, the theorem prover stops after printing out all of the remaining inequalities and complementary slackness conditions. The only other way the theorem prover can stop is if there is a contradiction found. We will now discuss each of the steps which are tried by the prover.

6.1. Using Another Theorem in the Proof

At this step the list of assumptions for each of the theorems in the list of theorems is checked. If all the assumptions for a certain theorem hold then we can use that theorem, and the conclusions of that theorem are added to the list of inequalities.

6.2. Substituting an Integer for a Variable

This step will be performed if we have an equality of the form $Var = Int$ for some variable Var and integer Int in the list of inequalities. Whenever we substitute an integer for a variable, we have to make this substitution in every inequality and complementary slackness condition, in all the inequalities in the conclusion of the proof, and in all the assumptions and conclusions of the theorems we are allowed to use during this proof. After this is done we have eliminated all occurrences of that variable and can simply ignore it from that point on.

6.3. Eliminating a Variable or Integer from an Inequality

If a variable which is greater than zero (this is checked by looking at the list of inequalities) occurs in every product of an inequality, and the inequality has an integer part of zero, we can eliminate this variable from the entire inequality by dividing it out. We also check if we can eliminate an integer which is bigger than 1 in absolute value, from an expression. However, we only perform this simplification if the integer or its negation occurs in every term of the inequality (this step could be applied in a lot more cases, however the work involved does not seem to be offset by the few additional cases in which this improvement can actually be used).

6.4. Decomposing a Complementary Slackness Condition

Recall that all complementary slackness conditions are of the form:

$$Expr \geq 0, \text{ and } Expr * Var = 0$$

At this stage we attempt to prove that either $Expr$ or Var cannot equal zero. If this is the case we eliminate this complementary slackness condition, but add the fact that now respectively Var or $Expr$ must be zero.

6.5. Replacing a Variable by a Bigger Integer

The first thing we do at this stage is check if there are any variables which can be proven to be less than a certain positive integer (e.g. if we have an inequality $-x > -1$ in our list). If this is the case, we see if we can use this fact to replace the variable by that integer in some other inequality. For example if we had the following inequalities in the list of inequalities:

$$x < 1, \ a\,x\,y + z \geq 0, \ y > 0, \text{ and } a > 0$$

we can obtain the new inequality:

$$a\,y + z > 0.$$

After the new inequality has been obtained, we check if it allows us to do one of the following things:

1. Prove one of the conclusions for this proof.
2. Prove an assumption for one of the theorems which can be used in this proof.

3. Obtain a contradiction.
4. Decompose one of the complementary slackness conditions.

We refer to an (in)equality meeting one of these criteria as a *useful* (in)equality. If the newly obtained inequality is *useful* we add it to the list of inequalities. If it is not, we check if we can divide the inequality by one or more variables/integers so that the resulting inequality is useful. Again, if this is the case we add the resulting inequality to the list of inequalities.

6.6. Adding Two Inequalities to Obtain another Useful Inequality

Here we attempt to add two inequalities to form a new one, which can be used later on in the theorem. Currently there are two versions of this step available. They differ in the number of inequalities they attempt to add. The first one, used by default, adds only those inequalities which have a cancellation of at least one product when they are added (this includes the case where the integer parts offset each other to get a merged integer part of 0). The second version adds every inequality to every other inequality. For example, the pairs of inequalities

$$2\,x\,y + z \geq 4, \text{ and } u - 2\,x\,y > 0$$
$$u + v = 5, \text{ and } 2\,x\,y > -5$$

would be added by both versions, whereas the inequalities

$$2\,x\,y + z \geq 4, \text{ and } u - x\,y > 0$$

would only be added by the second version. The second version involves a lot of extra additions, and the resulting inequalities are rarely used in the proofs. Therefore this option should not be used unless absolutely necessary.

After it has been decided which inequalities to merge, both versions behave in exactly the same way. The two inequalities are added by adding the left and right hand sides together. The type of the new inequality is the "stronger" of the types of the two inequalities (where $>$ is stronger than $\geq$, which in turn is stronger than $=$). After the new inequality is obtained, we check that it is not already in the list of inequalities, and that the new inequality is *useful* in our proof. This is done in exactly the same way as when we replace a variable by a bigger integer.

7. Termination Property

To ensure termination for each proof we had to restrict the steps which could be made by the theorem prover. In this chapter we will first of all show that the prover is indeed guaranteed to stop after a finite number of steps. We will then discuss how the user can guide the theorem prover to complete those proofs which it could not complete due to these restrictions. At the end of this chapter we will mention how more steps can be added to the prover, and the potential advantages and pitfalls in doing this.

We can view the theorem prover as a set of rewrite rules for the list of inequalities and complementary slackness conditions. As is pointed out in [2], proof of termination for such systems usually involves finding a measure on the expressions which decreases every time a rule is applied. This is clearly the case for some of the rewrite rules in our system, but it cannot be done for the step in which we add two inequalities together. The proof of termination becomes more complicated, but this step is needed so that we can deduce conclusions which are in some sense more complex than the assumptions.

We will look at each of the steps in turn, and prove they can only be applied a finite number of times.

- *The prover uses another theorem in the proof*: since the user can only specify a finite number of theorems which can be used, and each of the theorems can only be used once, this step can clearly be executed only a finite number of times.
- *Substituting an integer for a variable*: after this step is executed the total number of variables occurring in all of the inequalities and complementary slackness conditions is reduced by one. Therefore this step can only be executed a finite number of times.
- *Decomposing a complementary slackness condition*: the number of complementary slackness conditions is reduced by one when this step is executed, and there is only a finite number of them to begin with (note that none of the other steps can generate new complementary slackness conditions).

- *Eliminating a variable or integer from an inequality*: none of the previous steps creates new inequalities, and as we will prove for the next steps, we can never generate an infinite number of inequalities. Since during this step we eliminate one of the variables from an inequality (and no other step can add new variables to an inequality) and each inequality contains only a finite number of variables, this step can only be used a finite number of times (we can divide an entire inequality by an integer only once, since after this is done each product of the inequality has a coefficient of 1 or -1).
- *Replacing a variable by a bigger integer, and adding two inequalities to obtain another useful inequality*[6]: these two steps are discussed at the same time, since they are the only steps which add new inequalities to the list. Rather than looking at these steps in detail, we note that the newly obtained inequality has to be considered *useful* in the proof. We will show that only a finite number of inequalities can satisfy this property, which proves that these steps can only be applied a finite number of times. The new inequality has to satisfy one of the following criteria to be considered *useful*:

(1). It can prove one of the conclusions of this proof.
(2). It can prove an assumption for one of the theorems which can be used in this proof.
(3). It produces a contradiction.
(4). It allows us to decompose one of the complementary slackness conditions.

If it satisfies the third criterion, the proof terminates immediately. Since there are only finitely many complementary slackness conditions, and one of them is decomposed each time the fourth criterion is satisfied, this criterion can only be used a finite number of times.

The first criterion is satisfied by only finitely many inequalities, since we eliminate an inequality from the finite list of conclusions as soon as it has been proven (and no conclusions can be added to this list once a proof is started). Each time the second criterion is met we eliminate an assumption from the (finite) assumption list of one or more theorems which can be used, and again we can conclude this can only happen a finite number of times.

The fact that each of the steps can only be applied a finite number of times proves that the prover will always terminate, since it will eventually reach a state where it cannot apply any more steps.

By only allowing certain steps to be performed if the resulting inequality is considered *useful*, we have ensured termination of the theorem prover. However, this way the theorem prover cannot prove any theorems which need the use of these steps to obtain intermediate results (i.e. inequalities which do not appear in the conclusion, but which are needed during the proof). If this is the case the theorem prover will terminate before the conclusion is reached. To get around this problem, the user can add the intermediate inequalities to the list of conclusions. Now the theorem prover will consider them as *useful*, and it will make the required steps to complete the proof (an example of this is shown in the appendix). We could add more steps to the theorem prover to try to limit the number of intermediate inequalities that have to be specified. However, whenever we do this we also have to consider how helpful this new step is going to be whenever it is applied. One of the dangers is that the step can be used only in very few special cases. If this is the case a lot of time is spent to check if this step can be performed in all the other proofs which cannot use this step. A second danger is that the step can be applied very often, but that it very rarely does something helpful (adding 1 to both sides of all of the inequalities would be an extreme example of this). Again this would slow down the proofs of a lot of theorems without contributing to the actual proof.

If we add new steps to the theorem prover in the future, or if we expand one of the existing steps, we have to make sure one of the following conditions is met:

(1). The new (modified) step can only be applied a finite number of times.
(2). If this is not the case, we have to add a check at the end of the new (modified) step which makes sure the newly obtained inequality is *useful*.

If one of the above conditions is satisfied, our definition of *useful* guarantees us that the prover will indeed terminate. We might also want to add conditions to the definition of *useful* in the future, but we have to make sure this does not introduce an infinite number of inequalities which can be considered as *useful*.

---

[6] Although there are two versions of the step which adds inequalities, the proof is the same for each version.

8. Performance

The theorem prover was tried out on two separate problems. The proofs were produced with *wup* (Waterloo Unix Prolog [4]) using a VAX 11/785 running Berkeley Unix 4.2[7]. The following table summarizes some of the data on the performance of the theorem prover on the two problems. Ten of the proofs for the first problem required the version of the step in which we try to add all pairs of inequalities (i.e. the "total adder" version, as opposed to the "fast adder" version, which only attempts to add those inequalities in which at least one product gets merged). Separate statistics are given for those proofs to show how significant the difference between the two versions is.

	# of proofs	avg. time (sec.)	avg. stack use (words)	# of steps	# of bad steps	extra concl. needed
Appl. 1	85	177.4	5830	453	110	35
Appl. 1: fast adder	75	108.5	5680	371	86	28
Appl. 1: total adder	10	686.9	6960	82	24	7
Appl. 2	4	36.9	4360	20	3	4

The first application is discussed in [7], all the proofs from this paper are shown in the second appendix. It contains 8 constraint functions and 8 choice variables. The second application has 3 constraints and 2 choice variables, and is discussed in the first appendix. The maximum time for any proof was 1624 seconds for a proof which attempted to add all inequalities, and 561 seconds for any other proof (the maximum time for a proof of the second application was 60 seconds). On average, a step was performed every 33 seconds in the proofs of the first application (every 22 seconds for the proofs using the fast adder version, and once every 84 seconds for the other version). For the second application a step was performed every 7.4 seconds. The stack size never exceeded 8400 words for any proof. On average 1 out of 4 steps performed by the theorem prover did not contribute to the actual proof of the theorem. Although we had to add 39 intermediate conclusions for the proofs in both applications, these were spread out over 20 proofs. This means 69 out of the 89 proofs were able to complete without any user intervention.

9. Conclusion

The theorem prover successfully completed all the proofs given in [7], and derived the desired results for another system of Kuhn-Tucker conditions (see the appendix). Moreover, most of the proofs proceeded in the same way as in [7], and few irrelevant steps were performed. Future applications will give a clearer indication of the type and size of problems the theorem prover can reasonably deal with.

In order to ensure termination, we had to restrict certain steps of the theorem prover to only deduce inequalities that are considered *useful*. In doing this we reduced the number of theorems which could be proven automatically. However, with a little guidance from the user, these proofs can be completed by simply forcing certain intermediate results to be considered as *useful*. Again, future use of the theorem prover will have to determine how severe this drawback is to the productive use of the system.

There are two ways in which the theorem prover could be improved. We can add additional steps or enhance existing steps (for example we could try to add triples of inequalities rather than pairs) so that it can prove more theorems without any user help. We can also try to enhance it so that it can deal better with the logic of the problems, which now has to be done by the user. This can be done for both individual proofs (for example when a straight proof fails, try a proof by cases), as well as the overall strategy which is needed to solve the optimization problem.

Although the theorem prover can also be used to deduce new inequalities from any set of inequalities, the steps that are currently implemented are geared towards inequalities typically occurring in Kuhn-Tucker conditions for optimization problems. These steps are fairly simple and involve only trivial forms of symbolic manipulation. For more complicated steps (such as factoring of parts of the inequality), interaction with a symbolic computation package such as Maple ([8]) would probably be needed.

[7] With the current setup *wup* runs at approximately 800 logical inferences per second (LIPS), with a commercial Prolog interpreter we would expect a 10 fold increase in speed.

10. References

[1] Bledsoe W.W., "Non-resolution Theorem Proving", *Artificial Intelligence*, Vol. 9, 1977.

[2] Bundy A., Welham B., "Using Meta-level Inference for Selective Application of Multiple Rule Sets in Algebraic Manipulation", *Artificial Intelligence*, Vol. 16, 1981.

[3] Chiang A.C., "Fundamental Methods of Mathematical Economics", McGraw-Hill Inc., New York, 1984.

[4] van Emden M.H., Goebel R.G., "Waterloo UNIX Prolog User's Manual version 2.0", University of Waterloo, Waterloo, Ontario, 1985.

[5] Luenberger D.G., "Introduction to Linear and Nonlinear Programming", Addison-Wesley, 1973.

[6] Macnaughton A., "Minimizing Tax on Capital Gains on Principal Residences: A Mathematical Approach", paper presented to the annual meeting of the Canadian Academic Accounting Association in Winnipeg, May 1986.

[7] Macnaughton A., "Appendix to: Minimizing Tax on Capital Gains on Principal Residences", Accounting Group, University of Waterloo, Waterloo, Ontario, 1986.

[8] Char B.W., Geddes K.O., Gonnet G.H., Watt S.M., "Maple User's Guide", Watcom Publications, Waterloo, Ontario, 1985.

11. Appendix 1: Example of an Application

The problem discussed in this appendix is the total-total subproblem as explained in [6]. We will first give a very brief discussion of the problem solved in that paper. Under Canadian income tax law, an individual who owns two houses will be taxed on some of the capital gain realized on sale. The individual will wish to minimize the amount taxable:

$$ga - ea\,(na + 1 + ca) + gb - eb\,(nb + 1 + cb)$$

subject to the following constraint functions:

$$ca \leq y - 1$$
$$cb \leq y - 1$$
$$ca + cb \leq y$$

where ea, eb, y, na, nb, ga and gb are the parameters and ca and cb are the choice variables. The variables ca and cb are the number of years that houses a and b are designated as the principal residence. The parameter y is the number of years in which both houses were owned, we assume it is at least 2. The parameter na is the number of years house a, but not house b was owned, and this value is assumed to be non-negative (nb is defined similarly). The parameters ga and gb are the gain in value of each house, and are both positive. The parameters ea and eb represent the average gain in value of each house per year of ownership, and they are calculated as follows:

$$ea = ga\,/\,(\,y + na\,)$$
$$eb = gb\,/\,(\,y + nb\,)$$

These equations cannot be used in the way they are given here and thus have to be rewritten in the form:

$$ga = ea\,y + ea\,na$$
$$gb = eb\,y + eb\,na$$

which can be handled by the theorem prover. Further details about this problems are discussed in [6]. Since all the constraint functions for the problem are linear, the set of Kuhn-Tucker conditions are necessary and sufficient conditions for the optimal solution in this case.

We first have to calculate the Kuhn-Tucker conditions for the system. These consist of the non-negativity constraints on both the variables and the Lagrange multipliers (these Lagrange multipliers c1, c2 and c3 are introduced when we calculate the Kuhn-Tucker conditions), as well as the complementary slackness conditions. The complementary slackness conditions are:

$$c1 + c3 - ea \geq 0, \quad \text{and} \quad (c1 + c3 - ea)\,ca = 0$$

$$c2 + c3 - eb \geq 0, \quad \text{and} \quad (c2 + c3 - eb)\, cb = 0$$
$$-ca + y \geq 1, \quad \text{and} \quad (1 + ca - y)\, c1 = 0$$
$$-cb + y \geq 1, \quad \text{and} \quad (1 + cb - y)\, c2 = 0$$
$$-ca - cb + y \geq 0, \quad \text{and} \quad (ca + cb - y)\, c3 = 0$$

We then have to specify all the Kuhn-Tucker conditions and other constraints to the system using the **hyp** and **kt_cond** (to specify the complementary slackness conditions) predicates. These are shown here:

```
%
% Non-negativity of variables.
hyp( [ge 0 [1 ca]] );
hyp( [ge 0 [1 cb]] );
%
% Constraints on parameters.
hyp( [ge 0 [1 na]] );
hyp( [ge 0 [1 nb]] );
hyp( [g 0 [1 ga]] );
hyp( [g 0 [1 gb]] );
hyp( [ge 2 [1 y]] );
hyp( [g 0 [1 ea]] );
hyp( [g 0 [1 eb]] );
%
% Non-negativity of Lagrange multipliers.
hyp( [ge 0 [1 c1]] );
hyp( [ge 0 [1 c2]] );
hyp( [ge 0 [1 c3]] );
%
% Other constraints.
hyp( [e 0 [1 ea na] [1 ea y] [-1 ga]] );
hyp( [e 0 [1 eb nb] [1 eb y] [-1 gb]] );
%
% Complementary slackness conditions.
kt_cond( [[ge 0 [1 c1] [1 c3] [-1 ea]] [e 0 [1 ca]]] );
kt_cond( [[ge 0 [1 c2] [1 c3] [-1 eb]] [e 0 [1 cb]]] );
kt_cond( [[ge 1 [-1 ca] [1 y]] [e 0 [1 c1]]] );
kt_cond( [[ge 1 [-1 cb] [1 y]] [e 0 [1 c2]]] );
kt_cond( [[ge 0 [-1 ca] [-1 cb] [1 y]] [e 0 [1 c3]]] );
```

After these predicates are set up we can use the prover to help us find the solution. First of all we prove a theorem called "nowaste", which states:

$$ca + cb = y$$

To be able to use this theorem in other proofs we have to define it using the **theorem** predicate as follows:

```
theorem( nowaste [] [[e 0 [1 ca] [1 cb] [-1 y]]] );
```

Now other proofs can use the result of this theorem by specifying the name nowaste in the list of theorems that can be used. Using this theorem we can find the optimal solution for two separate cases by proving the theorems:

if $ea > eb$ then $ca = y - 1$, and $cb = 1$
if $eb > ea$ then $ca = 1$, and $cb = y - 1$

We will now show the proofs of all these theorems. The first one has to be proven by giving a proof by cases[8]. We know that $c3 \geq 0$, and we can prove the following two theorems:

[8] This theorem can be proven directly by adding the negation of the conclusion to the assumptions, from which we can derive a contradiction. We chose to prove it this way to demonstrate how the user can deal with some of the logic

if c3 > 0 then ca + cb = y
if c3 = 0 then ca + cb = y

By combining these two theorems we can conclude that it must always be the case that ca + cb = y. The two theorems which actually find the optimal solution, use this theorem in the proof. Both these theorems also require two additional conclusions to be specified in their list of conclusions for them to complete automatically. The following session shows how the theorems can be actually proven. The input which has to be typed by the user is shown in bold face.

?proof([[g 0 [1 c3]]] [] [[e 0 [1 ca] [1 cb] [-1 y]]]);

Starting assumptions:
c3 > 0
From c3 > 0, and (-ca-cb+y)(c3) = 0, conclude ca+cb-y = 0
Conclusion ca+cb-y = 0 has been proven.
Proof finished, all conclusions proven.
yes

?proof([[e 0 [1 c3]]] [] [[e 0 [1 ca] [1 cb] [-1 y]]]);

Starting assumptions:
c3 = 0
Substituting c3 = 0
- In compound assumption c1+c3-ea >= 0, ca = 0,
  to get c1-ea >= 0, ca = 0
- In compound assumption c2+c3-eb >= 0, cb = 0,
  to get c2-eb >= 0, cb = 0
- In compound assumption -ca-cb+y >= 0, c3 = 0,
  to get -ca-cb+y >= 0, 0 = 0

Eliminated 0 = 0, to get -ca-cb+y >= 0
From c1 > 0, and (-1-ca+y)(c1) = 0, conclude -ca+y = 1
by merging ea > 0 and c1-ea >= 0
From ca >= 1, and (c1-ea)(ca) = 0, conclude c1-ea = 0
by merging -ca+y = 1 and y >= 2
From c2 > 0, and (-1-cb+y)(c2) = 0, conclude -cb+y = 1
by merging eb > 0 and c2-eb >= 0
-ca >= -1 by merging -cb+y = 1 and -ca-cb+y >= 0
Obtained -ca = -1, from -ca >= -1 and ca >= 1
Substituting ca = 1
- In assumption -ca+y = 1, to get y = 2
- In assumption -ca-cb+y >= 0, to get -cb+y >= 1
- In conclusion ca+cb-y = 0, to get -cb+y = 1

Conclusion ca+cb-y = 0 has been proven.
Proof finished, all conclusions proven.
yes

?proof([[g 0 [1 ea] [-1 eb]]] [nowaste] [[e 1 [-1 ca] [1 y]] [e 1 [1 cb]]
[ge 0 [-1 c3] [1 eb]] [g 0 [-1 c3] [1 ea]]]);

Starting assumptions:
ea-eb > 0
Using theorem nowaste to add assumption(s):
ca+cb-y = 0

of the proofs.

From cb >= 1, and (c2+c3-eb)(cb) = 0, conclude c2+c3-eb = 0
by merging ca+cb-y = 0 and -ca+y >= 1
Conclude -c3+eb >= 0 by merging c2+c3-eb = 0 and c2 >= 0
Conclusion -c3+eb >= 0 has been proven.
Conclude -c3+ea > 0 by merging -c3+eb >= 0 and ea-eb > 0
Conclusion -c3+ea > 0 has been proven.
From c1 > 0, and (-1-ca+y)(c1) = 0, conclude -ca+y = 1
by merging -c3+ea > 0 and c1+c3-ea >= 0
Conclusion -ca+y = 1 has been proven.
Conclude cb = 1 by merging -ca+y = 1 and ca+cb-y = 0
Conclusion cb = 1 has been proven.
Proof finished, all conclusions proven.
yes

**?proof([[g 0 [-1 ea] [1 eb]]] [nowaste] [[e 1 [-1 cb] [1 y]] [e 1 [1 ca]]
[ge 0 [-1 c3] [1 ea]] [g 0 [-1 c3] [1 eb]]]);**

Starting assumptions:
-ea+eb > 0
Using theorem nowaste to add assumption(s):
ca+cb-y = 0
From cb >= 1, and (c2+c3-eb)(cb) = 0, conclude c2+c3-eb = 0
by merging ca+cb-y = 0 and -ca+y >= 1
From ca >= 1, and (c1+c3-ea)(ca) = 0, conclude c1+c3-ea = 0
by merging ca+cb-y = 0 and -cb+y >= 1
Conclude -c3+ea >= 0 by merging c1+c3-ea = 0 and c1 >= 0
Conclusion -c3+ea >= 0 has been proven.
Conclude -c3+eb > 0 by merging -c3+ea >= 0 and -ea+eb > 0
Conclusion -c3+eb > 0 has been proven.
From c2 > 0, and (-1-cb+y)(c2) = 0, conclude -cb+y = 1
by merging -c3+eb > 0 and c2+c3-eb = 0
Conclusion -cb+y = 1 has been proven.
Conclude ca = 1 by merging -cb+y = 1 and ca+cb-y = 0
Conclusion ca = 1 has been proven.
Proof finished, all conclusions proven.
yes

?quit;

12. Appendix 2: Using the Theorem Prover to Verify Theorems

In this appendix we show how the theorem prover was used to verify all the proofs from [7]. The problem which is solved is the total-transitional subproblem as discussed in [6]. The actual meaning of the symbolic variables, the function and all the constraints are of little importance to us and are therefore omitted. The problem discussed in [7] was to find the minimum of:

$$ta + sb1 + gb2 - eb2\ nb2$$

subject to the following constraints:

$$\begin{gathered} na1 + nb1 \leq y1 \\ na2 + nb2 \leq y2 \\ ga1 + ga2 - ea\ (\ na1 + na2 + pa\ ) \leq ta \\ pa \leq na1 + na2 \\ pa \leq 1 \\ gb1 - eb1\ (\ nb1 + pb1\ ) \leq sb1 \end{gathered}$$

$$pb1 \leq nb1$$
$$pb1 \leq 1$$

where na1, na2, nb1, nb2, ta, pa, sb1 and pb1 are the choice variables (assumed to be non-negative). The parameters of the system, which are strictly positive, are: y1, y2, ga1, ga2, gb1, gb2, ea, eb1, and eb2. When we calculate the Kuhn-Tucker conditions for the system, we obtain Lagrange multipliers c1 to c8, which are assumed to be non-negative. The following equalities also hold in the system, and thus if we want to be able to use them in the proofs they have to be added as assumptions.

$$eb1 = gb1 \ / \ y1$$
$$eb2 = gb2 \ / \ y2$$
$$ea = (\ ga1 + ga2 \ ) \ / \ (\ y1 + y2 \ )$$

They cannot be used in the way they are given here and thus have to be rewritten in the form:

$$gb1 = eb1 \ y1$$
$$gb2 = eb2 \ y2$$
$$ga1 + ga2 = ea \ y1 + ea \ y2$$

which can be handled by the theorem prover. We can calculate the Kuhn-Tucker conditions for this problem. These consist of the non-negativity constraints on both the variables and the Lagrange multipliers, as well as the following complementary slackness conditions:

$c1 - c3 \ ea - c4 \geq 0$, and $(c1 - c3 \ ea - c4) * na1 = 0$
$c1 - c6 \ eb1 - c7 \geq 0$, and $(c1 - c6 \ eb1 - c7) * nb1 = 0$
$c2 - c3 \ ea - c4 \geq 0$, and $(c2 - c3 \ ea - c4) * na2 = 0$
$c2 - eb2 \geq 0$, and $(c2 - eb2) * nb2 = 0$
$1 - c3 \geq 0$, and $(1 - c3) * ta = 0$
$1 - c6 \geq 0$, and $(1 - c6) * sb1 = 0$
$- c3 \ ea + c4 + c5 \geq 0$, and $(- c3 \ ea + c4 + c5 \ ) * pa = 0$
$- c6 \ eb1 + c7 + c8 \geq 0$, and $(- c6 \ eb1 + c7 + c8 \ ) * pb1 = 0$
$- na1 - nb1 + y1 \geq 0$, and $(- na1 - nb1 + y1) * c1 = 0$
$- na2 - nb2 + y2 \geq 0$, and $(- na2 - nb2 + y2) * c2 = 0$
$- ga1 - ga2 + ea \ (na1 + na2 + pa) + ta \geq 0$, and $(- ga1 - ga2 + ea \ (na1 + na2 + pa) + ta) * c3 = 0$
$na1 + na2 - pa \geq 0$, and $(na1 + na2 - pa) * c4 = 0$
$1 - pa \geq 0$, and $(1 - pa) * c5 = 0$
$- gb1 + eb1 \ (nb1 + pb1) + sb1 \geq 0$, and $(- gb1 + eb1 \ (nb1 + pb1) + sb1) * c3 = 0$
$nb1 - pb1 \geq 0$, and $(nb1 - pb1) * c7 = 0$
$1 - pb1 \geq 0$, and $(1 - pb1) * c8 = 0$

In total 35 theorems are given in [7] to find the optimal solution. To prove all these theorems using the theorem prover we needed 86 proofs. The theorems in [7] are grouped in 5 groups (A to E), we will now look at the theorems in each group, and also show what proofs were given by our theorem prover to show that the proofs are correct.

Part A:

a1: If $na1 + na2 < 1$ and $ta > 0$, then $pa = na1 + na2$
a2: If $nb1 < 1$ and $sb1 > 0$, then $pb1 = nb1$
a3: If $nb1 > 0$ and $ta > 0$, then $pb1 > 0$
a4: a) If $na1 > 0$ and $sb1 > 0$, then $pa > 0$
b) If $na2 > 0$, then $pa > 0$

The proofs given by the theorem prover are exactly the same as in [7]. In both cases a3 and a4 are proven by contradiction. The theorem prover actually uses two separate proofs (a4a, and a4b) to prove a4.

Part B:

b1: If $na1 + na2 < 1$ and $ta > 0$, then $2 \ ea \leq c1$ and $2 \ ea \leq c2$
b2: If $ta > 0$, then $ea \leq c1$ and $ea \leq c2$
b3: If $nb1 > 0$ and $ta > 0$, then $c1 \leq 2 \ eb1$

b4: If nb1 $>$ 1, then c1 $\leq$ eb1
b5: If nb2 $>$ 0, then c2 $=$ eb2
b6: If nb1 $>$ y1 - 1, y1 $\geq$ 2 and ta $>$ 0, then c1 $=$ 0
b7: If ta $>$ 0, nb1 $>$ 0 and y1 $\geq$ 2, then:
  a) 2 eb1 $\geq$ 2 ea if na1 + na2 $<$ 1
  b) 2 eb1 $\geq$ ea
  c) eb1 $\geq$ 2 ea if na1 + na2 $<$ 1 and 1 $<$ nb1 $\leq$ y1 - 1
  d) eb1 $\geq$ ea if 1 $<$ nb1 $\leq$ y1 - 1
  e) 0 $\geq$ 2 ea if na1 + na2 $<$ 1 and nb1 $>$ y1 - 1
  f) 0 $\geq$ ea if nb1 $>$ y1 - 1
b8: If ta $>$ 0 and nb2 $>$ 0, then:
  a) eb2 $\geq$ 2 ea if na1 + na2 $<$ 1
  b) eb2 $\geq$ ea

All proofs, except for b6 proceed in the same way as in [7]. To be able to proof b6, we prove the following three theorems:

b6_1: If nb1 $>$ y1 - 1, y1 $\geq$ 2 and ta $>$ 0 then c6 eb1 $=$ c8, and c7 $=$ 0
b6_2: If c6 $=$ c7 $=$ 0 and nb1 $>$ 0, then c1 $=$ 0
b6_3: c6 $>$ 0, c8 $>$ 0 and nb1 $>$ y1 - 1 lead to a contradiction

From b6_1 we get c6 eb1 $=$ c8, which means c6 $=$ c8 $=$ 0, or c6 $>$ 0 and c8 $>$ 0 (since we know that eb1 $>$ 0, c6 $\geq$ 0, and c8 $\geq$ 0). In b6_2 we show that the conclusion must hold in the first case, and b6_3 shows that the second case is actually impossible. Although we use 3 different proofs to prove the single theorem b6, we perform the same steps as in the proof of b6 in [7] (whenever we use more than one proof to prove one of the theorems from [7], we made sure that the logic used to combine these proofs is exactly the same as in [7]). For b7, 6 separate proofs are used, and for b8 we use 2 proofs. The only other thing to note is that in the proof of b3 we need 2 extra conclusions to allow the theorem prover to complete, and in b6_3 we need 1 extra conclusion.

Part C:

c1: a) If na1 $>$ 0 and sb1 $>$ 0, then c1 $\leq$ 2 ea
  b) If na2 $>$ 0, then c2 $\leq$ 2 ea
c2: a) If na1 + na2 $>$ 1 and na1 $>$ 0, then c1 $\leq$ ea
  b) If na1 + na2 $>$ 1 and na2 $>$ 0, then c2 $\leq$ ea
c3: If nb1 $<$ 1 and sb1 $>$ 0, then 2 eb1 $\leq$ c1
c4: If sb1 $>$ 0, then eb1 $\leq$ c1
c5: If na1 + na2 $>$ y1 + y2 - 1, y1 + y2 $\geq$ 2 and sb1 $>$ 0, then:
  a) c1 $=$ 0 if na1 $>$ 0
  b) c2 $=$ 0 if na2 $>$ 0
c6: If sb1 $>$ 0, na1 $>$ 0 and y1 + y2 $\geq$ 2, then:
  a) 2 ea $\geq$ 2 eb1 if nb1 $<$ 1
  b) 2 ea $\geq$ eb1
  c) ea $\geq$ 2 eb1 if nb1 $<$ 1 and na1 + na1 $>$ 1
  d) ea $\geq$ eb1 if na1 + na1 $>$ 1
  e) 0 $\geq$ 2 eb1 if nb1 $<$ 1 and na1 + na1 $>$ y1 + y2 - 1
  f) 0 $\geq$ eb1 if na1 + na2 $>$ y1 + y2 - 1
c7: If na2 $>$ 0 and y1 + y2 $\geq$ 2, then:
  a) 2 ea $\geq$ eb2
  b) ea $\geq$ eb2 if na1 + na2 $>$ 1
  c) 0 $\geq$ eb2 if na1 + na2 $>$ y1 + y2 - 1

As before, separate proofs are given for each part of a theorem if it consists of more than one part. We need 3 extra conclusions for both c1a and c1b. To prove c5, we use the following theorems:

c5_1: If na1 + na2 $>$ y1 + y2 - 1, y1 + y2 $\geq$ 2, na1 $>$ 0 and sb1 $>$ 0, then c3 ea $=$ c5 and c4 $=$ 0
c5_2: If na1 + na2 $>$ y1 + y2 - 1, y1 + y2 $\geq$ 2, na2 $>$ 0 and sb1 $>$ 0, then c3 ea $=$ c5 and c4 $=$ 0

c5_3: If $c3 = c4 = 0$ and $na1 > 0$, then $c1 = 0$
c5_4: If $c3 = c4 = 0$ and $na2 > 0$, then $c2 = 0$
c5_5: $c3 > 0$, $c5 > 0$ and $na1 + na2 > y1 + y2 - 1$ lead to a contradiction

These can be combined in the same way as for b6 to prove theorem c5. Both c5_1 and c5_2 need 1 extra conclusion to complete.

Part D:

d1: $na2 + nb2 = y2$
d2: If $na1 + nb1 < y1$, then $ta > 0$ or $sb1 > 0$ or $gb2 - eb2\ nb2 > 0$
d3: If $ta > 0$ or $sb1 > 0$ or $gb2 - eb2\ nb2 > 0$, then $na1 + nb1 = y1$
d4: $na1 + nb1 = y1$
d5: If $nb1 > y1 - 1$ and $y1 + y2 \geq 2$, then $ta > 0$ or $gb2 - eb2\ nb2 > 0$
d6: If $ta > 0$, and $y1 \geq 2$, then $nb1 \leq y1 - 1$
d7: If $ta = 0$, $gb2 - eb2\ nb2 > 0$ and $y1 \geq 2$, then $nb1 \leq y1 - 1$
d8: $nb1 \leq y1 - 1$

Theorem d2 is proven by 4 separate theorems. These theorems prove that we cannot have $ta = sb1 = 0$, $gb2 - eb2\ nb2 \leq 0$, and $na1 + nb1 < y1$ all at the same time, which is exactly what d2 says. Theorem d3 is proven by cases (one proof for each of the disjuncts in the hypothesis of the theorem). The same holds for theorem d4, where we prove that we get a contradiction from $na1 + nb1 < y1$ for each of the three cases of theorem d2. Theorem d5 is proven by proving:

If $nb1 > y1 - 1$, $y1 + y2 \geq 2$ and $gb2 - eb2\ nb2 \leq 0$, then $ta > 0$

which is equivalent to it. Theorems d6 and d7 are both proven by contradiction. The theorem prover showed that we actually do not need the extra assumption $gb2 - eb2\ nb2 > 0$ in the proof of d7, in which case the proof of d8 would become a lot simpler. To prove d8 in the same way as in [7], we need 2 proofs, one for each of the cases in d5. However, the theorem prover also showed that d8 needs the additional assumption $y1 \geq 2$. In total we need 11 extra conclusions for 4 of the proofs in this group.

Part E:

e1: If $ea < eb1$ and $y1 \geq 2$, then $nb1 = y1 - 1$
e2: If $ea < eb2$ and $y1 \geq 2$, then $na2 = 0$
e3: If $eb2 < ea < eb1$, then $nb2 = 0$
e4: If $2\ eb1 < ea < eb2$ and $y2 > 1$, then $na1 = y1$ and $nb1 = 0$
e5: If $eb1 < ea < eb2$ and $ea < 2\ eb1$, then $na1 = y1 - 1$ and $nb1 = 1$
e6: If $eb1 < ea$ and $eb2 < ea$, then $na1 + na2 = y1 + y2 - 1$ and $nb1 + nb2 = 1$
e7: If $nb1 + nb2 = 1$ and $eb2 < 2\ eb1$, then $nb1 = 1$ and $nb2 = 0$
e8: If $nb1 + nb2 = 1$, $2\ eb1 < eb2$ and $y2 \geq 1$, then $nb1 = 0$ and $nb2 = 1$

The theorem prover showed us we need the additional assumption $y1 \geq 2$ for theorems e3 to e8. Theorem e2 is proven by the following three theorems:

e2_1: If $y1 \geq 2$, then $na1 + na2 \geq 1$ and $na1 \geq 1$
e2_2: $y1 \geq 2$, $eb2 > ea$, $na2 > 0$ and $na1 + na2 > 1$ lead to a contradiction
e2_3: If $na1 + na2 = 1$ and $na1 \geq 1$, then $na2 = 0$

From the first theorem we can conclude that $na1 + na2 > 1$, or $na1 + na2 = 0$. Theorem e2_2 shows that the first case is impossible unless $na2 = 0$, and e2_3 shows that in the second case we must also have $na2 = 0$. Thus we have given a proof by cases of $na2 = 0$. Similarly, e3 ($ta > 0$, or $ta = 0$), e4 ($ta > 0$, or $ta = 0$) and e7 ($na1 > 0$, or $na1 = 0$) are also proven by cases. For both e4 and e7 we also prove a third theorem, which are:

e4_3: If $nb1 = 0$, then $na1 = y1$
e7_3: If $nb1 = 1$ and $nb1 + nb2 = 1$, then $nb2 = 0$

Theorem e5 is proven by showing that both $na1 < y1 - 1$, and $na1 > y1 - 1$ together with the assumptions lead to a contradiction. The third proof for e5 shows that if $na1 = y1 - 1$, then $nb1 = 1$. The

following 4 theorems are used to prove e6:

e6_1: If y1 + y2 > na1 + na2 + 1, then nb1 + nb2 > 1 and ta > 0
e6_2: ta > 0, ea > eb2 and nb2 > 0 lead to a contradiction
e6_3: ta > 0, y1 ≥ 2, ea > eb1, nb1 + nb2 > 1, and nb2 = 0 lead to a contradiction
e6_4: If y1 + y2 = na1 + na2 + 1, then nb1 + nb2 = 1

We can combine the first three theorems to give us a proof of y1 + y2 = na1 + na2 + 1, since e6_2 and e6_3 show by cases that from the assumptions of e6 and y1 + y2 > na1 + na2 + 1 we can derive a contradiction. The last proof shows that the second part of the conclusion of e6 also holds. Finally, theorem e8_1 is proven by the following theorems:

e8_1: If na2 = 0, then nb2 = y2
e8_2: If na2 > 0 and nb1 > 0, then c6 eb1 + c7 - eb2 ≥ 0
e8_3: c7 = 0, c6 eb1 + c7 - eb2 ≥ 0 and 2 eb1 < eb2 lead to a contradiction
e8_4: nb1 = pb1, nb1 > 0, c6 eb1 + c7 - eb2 ≥ 0 and 2 eb1 < eb2 lead to a contradiction

Theorem e8_1 shows that e8 is proven for the case when na2 = 0, since we have nb2 = y2 and y2 ≥ 1, which means we either have a contradiction (with nb1 + nb2 = 1) or the theorem is proven. The other three theorems show by contradiction that for the other case (na2 > 0) we must also have nb1 = 0. The contradiction is proven by the cases c7 = 0, and nb1 = pb1 (in e8_3 and e8_4 respectively) which is valid since we have c7 (nb1 - pb1) = 0 as part of one of our complementary slackness conditions. For 6 proofs in this group we needed to specify a total of 12 extra conclusions for them to complete.

The following 5 theorems were not explicitly stated in [7], but because they are used often in other proofs, we decided to state them as separate theorems:

z0: If na1 + na2 + pa < y1 + y2, then ga1 + ga2 > ea (na1 + na2 + pa)
z1: If nb1 + pb1 < y1, then gb1 > eb1 (nb1 + pb1)
z2: If na1 + na2 < y1 + y2 - 1, then ta > 0
z3: If nb1 < 1 and y1 ≥ 2, then sb1 > 0
z4: If ta = 0, then na1 + na2 + pa ≥ y1 + y2

Both z0 and z1 are proven by contradiction. To be able to complete, theorem z3 needs one extra conclusion.

All these theorems allow us to specify the solution for the optimization for 6 exhaustive cases (for the cases where ea = eb1, or ea = eb2 the solution is trivial). To be able to conclude these final results, we need the two additional assumptions y1 > 1, and y2 ≥ 2. For each of the cases, we also list the theorems that are used to give us these solutions.

If eb2 > ea and ea < eb1, then na1=1, na2=0, nb1=y1-1, and nb2=y2 (e1, e2)
If eb2 > ea > eb1 and ea < 2 eb1, then na1=y1-1, na2=0, nb1=1 and nb2=y2 (e2, e5)
If eb2 > ea > eb1 and ea > 2 eb1, then na1=y1, na2=0, nb1=0 and nb2=y2 (e2, e4)
If eb2 < ea < eb1, then na1=1, na2=y2, nb1=y1-1 and nb2=0 (e1, e3)
If eb2 < ea, ea > eb1 and eb2 < 2 eb1, then na1= y1-1, na2=y2, nb1=1 and nb2=0 (e6, e7)
If eb2 < ea, ea > eb1 and eb2 > 2 eb1, then na1=y1, na2=y2-1, nb1=0 and nb2=1 (e6, e8)

We will now show the proofs of all of the theorems, as produced by the theorem prover. They are shown in the same order as discussed here. Most of the proofs were produced using the "fast adder" version. The following 10 theorems need the use of the "total adder" version: d2_4, d5, d8a, d8b, e1, e2_1, e2_2, e5_1, e5_2 and z3.

```
?   pf(a1);
Starting assumptions:
   -na1-na2 > -1   ta > 0
From ta > 0, and (1-c3)(ta) = 0, conclude c3 = 1
Substituting c3 = 1
 - In compound assumption c1-c3 ea-c4 >= 0, na1 = 0,
     to get c1-c4-ea >= 0, na1 = 0
 - In compound assumption c2-c3 ea-c4 >= 0, na2 = 0,
     to get c2-c4-ea >= 0, na2 = 0
 - In compound assumption -c3 ea+c4+c5 >= 0, pa = 0,
     to get c4+c5-ea >= 0, pa = 0
 - In compound assumption ea na1+ea na2+ea pa-ga1-ga2+ta >= 0, c3 = 0,
     to get ea na1+ea na2+ea pa-ga1-ga2+ta >= 0, 0 = 1
Eliminated 0 = 1, to get ea na1+ea na2+ea pa-ga1-ga2+ta = 0
From -pa > -1, and (1-pa)(c5) = 0, conclude c5 = 0
 by merging -na1-na2 > -1 and na1+na2-pa >= 0
Substituting c5 = 0
 - In compound assumption c4+c5-ea >= 0, pa = 0,
     to get c4-ea >= 0, pa = 0
From c4 > 0, and (na1+na2-pa)(c4) = 0, conclude na1+na2-pa = 0
 by merging ea > 0 and c4-ea >= 0
Conclusion na1+na2-pa = 0 has been proven.
Proof finished, all conclusions proven.

Inferences    = 132319
Unifications  = 323739
Time (msec)   = 205400
Speed (LIPS)  =    644

Runtime stack = 12 % (4125 words)
Copy stack    =  6 % (1972 words)
Trail stack   =  1 % (448 words)

yes

?   pf(a2);
Starting assumptions:
   -nb1 > -1   sb1 > 0
From sb1 > 0, and (1-c6)(sb1) = 0, conclude c6 = 1
Substituting c6 = 1
 - In compound assumption c1-c6 eb1-c7 >= 0, nb1 = 0,
     to get c1-c7-eb1 >= 0, nb1 = 0
 - In compound assumption -c6 eb1+c7+c8 >= 0, pb1 = 0,
     to get c7+c8-eb1 >= 0, pb1 = 0
 - In compound assumption eb1 nb1+eb1 pb1-gb1+sb1 >= 0, c6 = 0,
     to get eb1 nb1+eb1 pb1-gb1+sb1 >= 0, 0 = 1
Eliminated 0 = 1, to get eb1 nb1+eb1 pb1-gb1+sb1 = 0
From -pb1 > -1, and (1-pb1)(c8) = 0, conclude c8 = 0
 by merging -nb1 > -1 and nb1-pb1 >= 0
Substituting c8 = 0
 - In compound assumption c7+c8-eb1 >= 0, pb1 = 0,
     to get c7-eb1 >= 0, pb1 = 0
From c7 > 0, and (nb1-pb1)(c7) = 0, conclude nb1-pb1 = 0
 by merging eb1 > 0 and c7-eb1 >= 0
Conclusion nb1-pb1 = 0 has been proven.
Proof finished, all conclusions proven.

Inferences    =  76409
Unifications  = 159055
Time (msec)   =  98483
Speed (LIPS)  =    775

Runtime stack = 11 % (3671 words)
Copy stack    =  5 % (1930 words)
Trail stack   =  1 % (429 words)

yes

?   pf(a3);
Starting assumptions:
   nb1 > 0   ta > 0   -pb1 > -1
   nb1-pb1 > 0
From nb1-pb1 > 0, and (nb1-pb1)(c7) = 0, conclude c7 = 0
Substituting c7 = 0
 - In compound assumption c1-c6 eb1-c7 >= 0, nb1 = 0,
     to get c1-c6 eb1 >= 0, nb1 = 0
 - In compound assumption -c6 eb1+c7+c8 >= 0, pb1 = 0,
     to get -c6 eb1+c8 >= 0, pb1 = 0
From -pb1 > -1, and (1-pb1)(c8) = 0, conclude c8 = 0
Substituting c8 = 0
 - In compound assumption -c6 eb1+c8 >= 0, pb1 = 0,
     to get -c6 eb1 >= 0, pb1 = 0
Eliminating eb1 from -c6 eb1 >= 0 giving -c6 >= 0
Obtained -c6 = 0, from -c6 >= 0 and c6 >= 0
Substituting c6 = 0
 - In compound assumption -c6 >= 0, pb1 = 0,
     to get 0 >= 0, pb1 = 0
 - In compound assumption c1-c6 eb1 >= 0, nb1 = 0,
     to get c1 >= 0, nb1 = 0
 - In compound assumption -c6 >= -1, sb1 = 0,
     to get 0 >= -1, sb1 = 0
 - In compound assumption eb1 nb1+eb1 pb1-gb1+sb1 >= 0, c6 = 0,
     to get eb1 nb1+eb1 pb1-gb1+sb1 >= 0, 0 = 0
Eliminated 0 >= -1, to get sb1 = 0
Eliminated 0 = 0, to get eb1 nb1+eb1 pb1-gb1+sb1 >= 0
Substituting sb1 = 0
 - In assumption eb1 nb1+eb1 pb1-gb1+sb1 >= 0, to get eb1 nb1+eb1 pb1-gb1 >= 0
From ta > 0, and (1-c3)(ta) = 0, conclude c3 = 1
Substituting c3 = 1
 - In compound assumption c1-c3 ea-c4 >= 0, na1 = 0,
     to get c1-c4-ea >= 0, na1 = 0
 - In compound assumption c2-c3 ea-c4 >= 0, na2 = 0,
     to get c2-c4-ea >= 0, na2 = 0
 - In compound assumption -c3 ea+c4+c5 >= 0, pa = 0,
     to get c4+c5-ea >= 0, pa = 0
 - In compound assumption ea na1+ea na2+ea pa-ga1-ga2+ta >= 0, c3 = 0,
     to get ea na1+ea na2+ea pa-ga1-ga2+ta >= 0, 0 = 1
Eliminated 0 = 1, to get ea na1+ea na2+ea pa-ga1-ga2+ta = 0
From nb1 > 0, and (c1)(nb1) = 0, conclude c1 = 0
Substituting c1 = 0
 - In compound assumption c1-c4-ea >= 0, na1 = 0,
     to get -c4-ea >= 0, na1 = 0
 - In compound assumption -na1-nb1+y1 >= 0, c1 = 0,
     to get -na1-nb1+y1 >= 0, 0 = 0
Eliminated 0 = 0, to get -na1-nb1+y1 >= 0
-c4 > 0 by merging ea > 0 and -c4-ea >= 0
Contradiction -c4 > 0 and c4 >= 0

Inferences    =  92853
Unifications  = 205203
Time (msec)   = 126300
Speed (LIPS)  =    735

Runtime stack = 23 % (7687 words)
Copy stack    =  8 % (2890 words)
Trail stack   =  2 % (759 words)

no

?   pf(a4a);
Starting assumptions:
   na1 > 0   sb1 > 0   -pa > -1
```

```
   na1+na2-pa > 0
From na1+na2-pa > 0, and (na1+na2-pa)(c4) = 0, conclude c4 = 0
Substituting c4 = 0
 - In compound assumption c1-c3 ea-c4 >= 0, na1 = 0,
     to get c1-c3 ea >= 0, na1 = 0
 - In compound assumption c2-c3 ea-c4 >= 0, na2 = 0,
     to get c2-c3 ea >= 0, na2 = 0
 - In compound assumption -c3 ea+c4+c5 >= 0, pa = 0,
     to get -c3 ea+c5 >= 0, pa = 0
From -pa > -1, and (1-pa)(c5) = 0, conclude c5 = 0
Substituting c5 = 0
 - In compound assumption -c3 ea+c5 >= 0, pa = 0,
     to get -c3 ea >= 0, pa = 0
Eliminating ea from -c3 ea >= 0 giving -c3 >= 0
Obtained -c3 = 0, from -c3 >= 0 and c3 >= 0
Substituting c3 = 0
 - In compound assumption -c3 >= 0, pa = 0,
     to get 0 >= 0, pa = 0
 - In compound assumption c2-c3 ea >= 0, na2 = 0,
     to get c2 >= 0, na2 = 0
 - In compound assumption c1-c3 ea >= 0, na1 = 0,
     to get c1 >= 0, na1 = 0
 - In compound assumption -c3 >= -1, ta = 0,
     to get 0 >= -1, ta = 0
 - In compound assumption ea na1+ea na2+ea pa-ga1-ga2+ta >= 0, c3 = 0,
     to get ea na1+ea na2+ea pa-ga1-ga2+ta >= 0, 0 = 0
Eliminated 0 >= -1, to get ta = 0
Eliminated 0 = 0, to get ea na1+ea na2+ea pa-ga1-ga2+ta >= 0
Substituting ta = 0
 - In assumption ea na1+ea na2+ea pa-ga1-ga2+ta >= 0,
                                 to get ea na1+ea na2+ea pa-ga1-ga2 >= 0
From sb1 > 0, and (1-c6)(sb1) = 0, conclude c6 = 1
Substituting c6 = 1
 - In compound assumption c1-c6 eb1-c7 >= 0, nb1 = 0,
     to get c1-c7-eb1 >= 0, nb1 = 0
 - In compound assumption -c6 eb1+c7+c8 >= 0, pb1 = 0,
     to get c7+c8-eb1 >= 0, pb1 = 0
 - In compound assumption eb1 nb1+eb1 pb1-gb1+sb1 >= 0, c6 = 0,
     to get eb1 nb1+eb1 pb1-gb1+sb1 >= 0, 0 = 1
Eliminated 0 = 1, to get eb1 nb1+eb1 pb1-gb1+sb1 = 0
From na1 > 0, and (c1)(na1) = 0, conclude c1 = 0
Substituting c1 = 0
 - In compound assumption c1-c7-eb1 >= 0, nb1 = 0,
     to get -c7-eb1 >= 0, nb1 = 0
 - In compound assumption -na1-nb1+y1 >= 0, c1 = 0,
     to get -na1-nb1+y1 >= 0, 0 = 0
Eliminated 0 = 0, to get -na1-nb1+y1 >= 0
-c7 > 0 by merging eb1 > 0 and -c7-eb1 >= 0
Contradiction -c7 > 0 and c7 >= 0

Inferences     = 314291
Unifications   = 608801
Time (msec)    = 449850
Speed (LIPS)   =    698

Runtime stack = 25 % (8408 words)
Copy stack    =  9 % (2954 words)
Trail stack   =  2 % (790 words)

no

?   pf(a4b);
Starting assumptions:
   na2 > 0   -pa > -1   na1+na2-pa > 0
From na1+na2-pa > 0, and (na1+na2-pa)(c4) = 0, conclude c4 = 0
Substituting c4 = 0
 - In compound assumption c1-c3 ea-c4 >= 0, na1 = 0,
     to get c1-c3 ea >= 0, na1 = 0
 - In compound assumption c2-c3 ea-c4 >= 0, na2 = 0,
     to get c2-c3 ea >= 0, na2 = 0
 - In compound assumption -c3 ea+c4+c5 >= 0, pa = 0,
     to get -c3 ea+c5 >= 0, pa = 0
From -pa > -1, and (1-pa)(c5) = 0, conclude c5 = 0
Substituting c5 = 0
 - In compound assumption -c3 ea+c5 >= 0, pa = 0,
     to get -c3 ea >= 0, pa = 0
Eliminating ea from -c3 ea >= 0 giving -c3 >= 0
Obtained -c3 = 0, from -c3 >= 0 and c3 >= 0
Substituting c3 = 0
 - In compound assumption -c3 >= 0, pa = 0,
     to get 0 >= 0, pa = 0
 - In compound assumption c2-c3 ea >= 0, na2 = 0,
     to get c2 >= 0, na2 = 0
 - In compound assumption c1-c3 ea >= 0, na1 = 0,
     to get c1 >= 0, na1 = 0
 - In compound assumption -c3 >= -1, ta = 0,
     to get 0 >= -1, ta = 0
 - In compound assumption ea na1+ea na2+ea pa-ga1-ga2+ta >= 0, c3 = 0,
     to get ea na1+ea na2+ea pa-ga1-ga2+ta >= 0, 0 = 0
Eliminated 0 >= -1, to get ta = 0
Eliminated 0 = 0, to get ea na1+ea na2+ea pa-ga1-ga2+ta >= 0
Substituting ta = 0
 - In assumption ea na1+ea na2+ea pa-ga1-ga2+ta >= 0,
                                 to get ea na1+ea na2+ea pa-ga1-ga2 >= 0
From na2 > 0, and (c2)(na2) = 0, conclude c2 = 0
Substituting c2 = 0
 - In compound assumption c2-eb2 >= 0, nb2 = 0,
     to get -eb2 >= 0, nb2 = 0
 - In compound assumption -na2-nb2+y2 >= 0, c2 = 0,
     to get -na2-nb2+y2 >= 0, 0 = 0
Contradiction -eb2 >= 0 and eb2 > 0

Inferences     = 27553
Unifications   = 48882
Time (msec)    = 32883
Speed (LIPS)   =   837

Runtime stack = 16 % (5359 words)
Copy stack    =  7 % (2424 words)
Trail stack   =  1 % (584 words)

no
```

```
?   pf(b1);
Starting assumptions:
   -na1-na2 > -1   ta > 0
From ta > 0, and (1-c3)(ta) = 0, conclude c3 = 1
Substituting c3 = 1
 - In compound assumption c1-c3 ea-c4 >= 0, na1 = 0,
     to get c1-c4-ea >= 0, na1 = 0
 - In compound assumption c2-c3 ea-c4 >= 0, na2 = 0,
     to get c2-c4-ea >= 0, na2 = 0
 - In compound assumption -c3 ea+c4+c5 >= 0, pa = 0,
     to get c4+c5-ea >= 0, pa = 0
 - In compound assumption ea na1+ea na2+ea pa-ga1-ga2+ta >= 0, c3 = 0,
     to get ea na1+ea na2+ea pa-ga1-ga2+ta >= 0, 0 = 1
Eliminated 0 = 1, to get ea na1+ea na2+ea pa-ga1-ga2+ta = 0
From -pa > -1, and (1-pa)(c5) = 0, conclude c5 = 0
 by merging -na1-na2 > -1 and na1+na2-pa >= 0
Substituting c5 = 0
 - In compound assumption c4+c5-ea >= 0, pa = 0,
     to get c4-ea >= 0, pa = 0
From c4 > 0, and (na1+na2-pa)(c4) = 0, conclude na1+na2-pa = 0
 by merging ea > 0 and c4-ea >= 0
From c2 > 0, and (-na2-nb2+y2)(c2) = 0, conclude na2+nb2-y2 = 0
 by merging eb2 > 0 and c2-eb2 >= 0
Conclude c2-2 ea >= 0 by merging c4-ea >= 0 and c2-c4-ea >= 0
Conclusion c2-2 ea >= 0 has been proven.
Conclude c1-2 ea >= 0 by merging c4-ea >= 0 and c1-c4-ea >= 0
Conclusion c1-2 ea >= 0 has been proven.
Proof finished, all conclusions proven.

Inferences     = 387913
Unifications   = 946704
Time (msec)    = 561516
Speed (LIPS)   =    690

Runtime stack = 21 % (6877 words)
Copy stack    =  7 % (2316 words)
Trail stack   =  1 % (580 words)

yes

?   pf(b2);
Starting assumptions:
   ta > 0
From ta > 0, and (1-c3)(ta) = 0, conclude c3 = 1
Substituting c3 = 1
 - In compound assumption c1-c3 ea-c4 >= 0, na1 = 0,
     to get c1-c4-ea >= 0, na1 = 0
 - In compound assumption c2-c3 ea-c4 >= 0, na2 = 0,
     to get c2-c4-ea >= 0, na2 = 0
 - In compound assumption -c3 ea+c4+c5 >= 0, pa = 0,
     to get c4+c5-ea >= 0, pa = 0
 - In compound assumption ea na1+ea na2+ea pa-ga1-ga2+ta >= 0, c3 = 0,
     to get ea na1+ea na2+ea pa-ga1-ga2+ta >= 0, 0 = 1
Eliminated 0 = 1, to get ea na1+ea na2+ea pa-ga1-ga2+ta = 0
From c2 > 0, and (-na2-nb2+y2)(c2) = 0, conclude na2+nb2-y2 = 0
 by merging eb2 > 0 and c2-eb2 >= 0
Conclude c2-ea >= 0 by merging c4 >= 0 and c2-c4-ea >= 0
Conclusion c2-ea >= 0 has been proven.
Conclude c1-ea >= 0 by merging c4 >= 0 and c1-c4-ea >= 0
Conclusion c1-ea >= 0 has been proven.
Proof finished, all conclusions proven.

Inferences     = 225877
Unifications   = 548607
Time (msec)    = 309483
Speed (LIPS)   =    729

Runtime stack = 14 % (4583 words)
Copy stack    =  5 % (1866 words)
Trail stack   =  1 % (415 words)

yes

?   pf(b3);
Starting assumptions:
   nb1 > 0   ta > 0
Using theorem a3 to add assumption(s):
   pb1 > 0
From pb1 > 0, and (-c6 eb1+c7+c8)(pb1) = 0, conclude c6 eb1-c7-c8 = 0
From ta > 0, and (1-c3)(ta) = 0, conclude c3 = 1
Substituting c3 = 1
 - In compound assumption c1-c3 ea-c4 >= 0, na1 = 0,
     to get c1-c4-ea >= 0, na1 = 0
 - In compound assumption c2-c3 ea-c4 >= 0, na2 = 0,
     to get c2-c4-ea >= 0, na2 = 0
 - In compound assumption -c3 ea+c4+c5 >= 0, pa = 0,
     to get c4+c5-ea >= 0, pa = 0
 - In compound assumption ea na1+ea na2+ea pa-ga1-ga2+ta >= 0, c3 = 0,
     to get ea na1+ea na2+ea pa-ga1-ga2+ta >= 0, 0 = 1
Eliminated 0 = 1, to get ea na1+ea na2+ea pa-ga1-ga2+ta = 0
From nb1 > 0, and (c1-c6 eb1-c7)(nb1) = 0, conclude c1-c6 eb1-c7 = 0
Conclude c6 eb1-c7 >= 0 by merging c6 eb1-c7-c8 = 0 and c8 >= 0
Conclusion c6 eb1-c7 >= 0 has been proven.
Conclude -c1+2 c6 eb1 >= 0 by merging c6 eb1-c7 >= 0 and c1-c6 eb1-c7 = 0
Conclusion -c1+2 c6 eb1 >= 0 has been proven.
Conclude -c1+2 eb1 >= 0 by merging -c6 >= -1 and -c1+2 c6 eb1 >= 0
Conclusion -c1+2 eb1 >= 0 has been proven.
Proof finished, all conclusions proven.

Inferences     = 127599
Unifications   = 276490
Time (msec)    = 192450
Speed (LIPS)   =    663

Runtime stack = 14 % (4666 words)
Copy stack    =  5 % (1898 words)
Trail stack   =  1 % (400 words)

yes

?   pf(b4);
Starting assumptions:
   nb1 > 1
From nb1 > 1, and (c1-c6 eb1-c7)(nb1) = 0, conclude c1-c6 eb1-c7 = 0
From nb1-pb1 > 0, and (nb1-pb1)(c7) = 0, conclude c7 = 0
 by merging nb1 > 1 and -pb1 >= -1
Substituting c7 = 0
 - In assumption c1-c6 eb1-c7 = 0, to get c1-c6 eb1 = 0
 - In compound assumption -c6 eb1+c7+c8 >= 0, pb1 = 0,
     to get -c6 eb1+c8 >= 0, pb1 = 0
Conclude -c1+eb1 >= 0 by merging -c6 >= -1 and -c1+c6 eb1 = 0
Conclusion -c1+eb1 >= 0 has been proven.
Proof finished, all conclusions proven.

Inferences     =  49067
Unifications   =  92571
Time (msec)    =  61050
Speed (LIPS)   =    803

Runtime stack =  7 % (2411 words)
Copy stack    =  5 % (1654 words)
Trail stack   =  0 % (315 words)
```

```
yes

?   pf(b5);
Starting assumptions:
   nb2 > 0
From nb2 > 0, and (c2-eb2)(nb2) = 0, conclude c2-eb2 = 0
Conclusion c2-eb2 = 0 has been proven.
Proof finished, all conclusions proven.

Inferences     =   6771
Unifications   =  11805
Time (msec)    =   8033
Speed (LIPS)   =    842

Runtime stack =  1 % (454 words)
Copy stack    =  3 % (1270 words)
Trail stack   =  0 % (193 words)

yes

?   pf(b6_1);
Starting assumptions:
   nb1-y1 > -1   y1 >= 2   ta > 0
From ta > 0, and (1-c3)(ta) = 0, conclude c3 = 1
Substituting c3 = 1
 - In compound assumption c1-c3 ea-c4 >= 0, na1 = 0,
     to get c1-c4-ea >= 0, na1 = 0
 - In compound assumption c2-c3 ea-c4 >= 0, na2 = 0,
     to get c2-c4-ea >= 0, na2 = 0
 - In compound assumption -c3 ea+c4+c5 >= 0, pa = 0,
     to get c4+c5-ea >= 0, pa = 0
 - In compound assumption ea na1+ea na2+ea pa-ga1-ga2+ta >= 0, c3 = 0,
     to get ea na1+ea na2+ea pa-ga1-ga2+ta >= 0, 0 = 1
Eliminated 0 = 1, to get ea na1+ea na2+ea pa-ga1-ga2+ta = 0
Obtained to use for theorem: nb1 > 1 by merging y1 >= 2 and nb1-y1 > -1
Using theorem a3 to add assumption(s):
   pb1 > 0
From pb1 > 0, and (-c6 eb1+c7+c8)(pb1) = 0, conclude c6 eb1-c7-c8 = 0
From nb1 > 1, and (c1-c6 eb1-c7)(nb1) = 0, conclude c1-c6 eb1-c7 = 0
From nb1-pb1 > 0, and (nb1-pb1)(c7) = 0, conclude c7 = 0
 by merging nb1 > 1 and -pb1 >= -1
Substituting c7 = 0
 - In assumption c1-c6 eb1-c7 = 0, to get c1-c6 eb1 = 0
 - In assumption c6 eb1-c7-c8 = 0, to get c6 eb1-c8 = 0
Conclusion c6 eb1-c8 = 0 has been proven.
Proof finished, all conclusions proven.

Inferences     = 117776
Unifications   = 263465
Time (msec)    = 155150
Speed (LIPS)   =    759

Runtime stack = 14 % (4787 words)
Copy stack    =  6 % (2128 words)
Trail stack   =  1 % (498 words)

yes

?   pf(b6_2);
Starting assumptions:
   c6 = 0   c7 = 0   nb1 > 0
Substituting c7 = 0
 - In compound assumption c1-c6 eb1-c7 >= 0, nb1 = 0,
     to get c1-c6 eb1 >= 0, nb1 = 0
 - In compound assumption -c6 eb1+c7+c8 >= 0, pb1 = 0,
     to get -c6 eb1+c8 >= 0, pb1 = 0
 - In compound assumption nb1-pb1 >= 0, c7 = 0,
     to get nb1-pb1 >= 0, 0 = 0
Eliminated 0 = 0, to get nb1-pb1 >= 0
Substituting c6 = 0
 - In compound assumption -c6 eb1+c8 >= 0, pb1 = 0,
     to get c8 >= 0, pb1 = 0
 - In compound assumption c1-c6 eb1 >= 0, nb1 = 0,
     to get c1 >= 0, nb1 = 0
 - In compound assumption -c6 >= -1, sb1 = 0,
     to get 0 >= -1, sb1 = 0
 - In compound assumption eb1 nb1+eb1 pb1-gb1+sb1 >= 0, c6 = 0,
     to get eb1 nb1+eb1 pb1-gb1+sb1 >= 0, 0 = 0
Eliminated 0 >= -1, to get sb1 = 0
Eliminated 0 = 0, to get eb1 nb1+eb1 pb1-gb1+sb1 >= 0
Substituting sb1 = 0
 - In assumption eb1 nb1+eb1 pb1-gb1+sb1 >= 0, to get eb1 nb1+eb1 pb1-gb1 >= 0
From nb1 > 0, and (c1)(nb1) = 0, conclude c1 = 0
Conclusion c1 = 0 has been proven.
Proof finished, all conclusions proven.

Inferences     =  10653
Unifications   =  19101
Time (msec)    =  14133
Speed (LIPS)   =    753

Runtime stack =  9 % (3165 words)
Copy stack    =  5 % (1926 words)
Trail stack   =  1 % (414 words)

yes

?   pf(b6_3);
Starting assumptions:
   c6 > 0   c8 > 0   nb1-y1 > -1
From c8 > 0, and (1-pb1)(c8) = 0, conclude pb1 = 1
Substituting pb1 = 1
 - In compound assumption -c6 eb1+c7+c8 >= 0, pb1 = 0,
     to get -c6 eb1+c7+c8 >= 0, 0 = 1
 - In compound assumption eb1 nb1+eb1 pb1-gb1+sb1 >= 0, c6 = 0,
     to get eb1+eb1 nb1-gb1+sb1 >= 0, c6 = 0
 - In compound assumption nb1-pb1 >= 0, c7 = 0,
     to get nb1 >= 1, c7 = 0
Eliminated 0 = 1, to get c6 eb1-c7-c8 = 0
From c6 > 0, and (eb1+eb1 nb1-gb1+sb1)(c6) = 0, conclude eb1+eb1 nb1-gb1+sb1 = 0
Conclude -eb1-eb1 nb1+gb1 >= 0 by merging eb1+eb1 nb1-gb1+sb1 = 0 and sb1 >= 0
Conclusion -eb1-eb1 nb1+gb1 >= 0 has been proven.
Conclude -nb1+y1 >= 1 by merging -eb1-eb1 nb1+gb1 >= 0 and eb1 y1-gb1 = 0
Contradiction -nb1+y1 >= 1 and nb1-y1 > -1

Inferences     =  64589
Unifications   = 125938
Time (msec)    =  81350
Speed (LIPS)   =    793

Runtime stack = 11 % (3914 words)
Copy stack    =  5 % (1788 words)
Trail stack   =  1 % (380 words)

no

?   pf(b7a);
Starting assumptions:
   nb1 > 0   y1 >= 2   ta > 0
   -na1-na2 > -1
Using theorem b1 to add assumption(s):
```

```
   c1-2 ea >= 0   c2-2 ea >= 0
Using theorem b3 to add assumption(s):
   -c1+2 eb1 >= 0
From ta > 0, and (1-c3)(ta) = 0, conclude c3 = 1
Substituting c3 = 1
 - In compound assumption c1-c3 ea-c4 >= 0, na1 = 0,
     to get c1-c4-ea >= 0, na1 = 0
 - In compound assumption c2-c3 ea-c4 >= 0, na2 = 0,
     to get c2-c4-ea >= 0, na2 = 0
 - In compound assumption -c3 ea+c4+c5 >= 0, pa = 0,
     to get c4+c5-ea >= 0, pa = 0
 - In compound assumption ea na1+ea na2+ea pa-ga1-ga2+ta >= 0, c3 = 0,
     to get ea na1+ea na2+ea pa-ga1-ga2+ta >= 0, 0 = 1
Eliminated 0 = 1, to get ea na1+ea na2+ea pa-ga1-ga2+ta = 0
From nb1 > 0, and (c1-c6 eb1-c7)(nb1) = 0, conclude c1-c6 eb1-c7 = 0
Conclude -2 ea+2 eb1 >= 0 by merging -c1+2 eb1 >= 0 and c1-2 ea >= 0
Conclusion -2 ea+2 eb1 >= 0 has been proven.
Proof finished, all conclusions proven.

Inferences    =  91863
Unifications  = 215179
Time (msec)   = 123183
Speed (LIPS)  =    745

Runtime stack = 11 % (3891 words)
Copy stack    =  5 % (1856 words)
Trail stack   =  1 % (402 words)

yes

?   pf(b7b);
Starting assumptions:
   nb1 > 0   y1 >= 2   ta > 0
Using theorem b2 to add assumption(s):
   c1-ea >= 0   c2-ea >= 0
Using theorem b3 to add assumption(s):
   -c1+2 eb1 >= 0
From ta > 0, and (1-c3)(ta) = 0, conclude c3 = 1
Substituting c3 = 1
 - In compound assumption c1-c3 ea-c4 >= 0, na1 = 0,
     to get c1-c4-ea >= 0, na1 = 0
 - In compound assumption c2-c3 ea-c4 >= 0, na2 = 0,
     to get c2-c4-ea >= 0, na2 = 0
 - In compound assumption -c3 ea+c4+c5 >= 0, pa = 0,
     to get c4+c5-ea >= 0, pa = 0
 - In compound assumption ea na1+ea na2+ea pa-ga1-ga2+ta >= 0, c3 = 0,
     to get ea na1+ea na2+ea pa-ga1-ga2+ta >= 0, 0 = 1
Eliminated 0 = 1, to get ea na1+ea na2+ea pa-ga1-ga2+ta = 0
From nb1 > 0, and (c1-c6 eb1-c7)(nb1) = 0, conclude c1-c6 eb1-c7 = 0
Conclude -ea+2 eb1 >= 0 by merging -c1+2 eb1 >= 0 and c1-ea >= 0
Conclusion -ea+2 eb1 >= 0 has been proven.
Proof finished, all conclusions proven.

Inferences    =  89804
Unifications  = 210177
Time (msec)   = 119866
Speed (LIPS)  =    749

Runtime stack = 11 % (3760 words)
Copy stack    =  5 % (1838 words)
Trail stack   =  1 % (398 words)

yes

?   pf(b7c);
Starting assumptions:
   nb1 > 0   y1 >= 2   ta > 0
   -na1-na2 > -1   nb1 > 1   -nb1+y1 >= 1
Using theorem b1 to add assumption(s):
   c1-2 ea >= 0   c2-2 ea >= 0
Using theorem b4 to add assumption(s):
   -c1+eb1 >= 0
From nb1 > 1, and (c1-c6 eb1-c7)(nb1) = 0, conclude c1-c6 eb1-c7 = 0
From ta > 0, and (1-c3)(ta) = 0, conclude c3 = 1
Substituting c3 = 1
 - In compound assumption c1-c3 ea-c4 >= 0, na1 = 0,
     to get c1-c4-ea >= 0, na1 = 0
 - In compound assumption c2-c3 ea-c4 >= 0, na2 = 0,
     to get c2-c4-ea >= 0, na2 = 0
 - In compound assumption -c3 ea+c4+c5 >= 0, pa = 0,
     to get c4+c5-ea >= 0, pa = 0
 - In compound assumption ea na1+ea na2+ea pa-ga1-ga2+ta >= 0, c3 = 0,
     to get ea na1+ea na2+ea pa-ga1-ga2+ta >= 0, 0 = 1
Eliminated 0 = 1, to get ea na1+ea na2+ea pa-ga1-ga2+ta = 0
Conclude -2 ea+eb1 >= 0 by merging -c1+eb1 >= 0 and c1-2 ea >= 0
Conclusion -2 ea+eb1 >= 0 has been proven.
Proof finished, all conclusions proven.

Inferences    =  96187
Unifications  = 225558
Time (msec)   = 130016
Speed (LIPS)  =    739

Runtime stack = 12 % (3971 words)
Copy stack    =  5 % (1858 words)
Trail stack   =  1 % (407 words)

yes

?   pf(b7d);
Starting assumptions:
   nb1 > 0   y1 >= 2   ta > 0
   nb1 > 1   -nb1+y1 >= 1
Using theorem b2 to add assumption(s):
   c1-ea >= 0   c2-ea >= 0
Using theorem b4 to add assumption(s):
   -c1+eb1 >= 0
From nb1 > 1, and (c1-c6 eb1-c7)(nb1) = 0, conclude c1-c6 eb1-c7 = 0
From ta > 0, and (1-c3)(ta) = 0, conclude c3 = 1
Substituting c3 = 1
 - In compound assumption c1-c3 ea-c4 >= 0, na1 = 0,
     to get c1-c4-ea >= 0, na1 = 0
 - In compound assumption c2-c3 ea-c4 >= 0, na2 = 0,
     to get c2-c4-ea >= 0, na2 = 0
 - In compound assumption -c3 ea+c4+c5 >= 0, pa = 0,
     to get c4+c5-ea >= 0, pa = 0
 - In compound assumption ea na1+ea na2+ea pa-ga1-ga2+ta >= 0, c3 = 0,
     to get ea na1+ea na2+ea pa-ga1-ga2+ta >= 0, 0 = 1
Eliminated 0 = 1, to get ea na1+ea na2+ea pa-ga1-ga2+ta = 0
Conclude -ea+eb1 >= 0 by merging -c1+eb1 >= 0 and c1-ea >= 0
Conclusion -ea+eb1 >= 0 has been proven.
Proof finished, all conclusions proven.

Inferences    =  94226
Unifications  = 220743
Time (msec)   = 127266
Speed (LIPS)  =    740

Runtime stack = 11 % (3842 words)
Copy stack    =  5 % (1838 words)
Trail stack   =  1 % (402 words)
```

```
yes

?   pf(b7e);
Starting assumptions:
   nb1 > 0   y1 >= 2   ta > 0
   -na1-na2 > -1   nb1-y1 > -1
Using theorem b1 to add assumption(s):
   c1-2 ea >= 0   c2-2 ea >= 0
Using theorem b6 to add assumption(s):
   c1 = 0
Substituting c1 = 0
 - In assumption c1-2 ea >= 0, to get -2 ea >= 0
 - In compound assumption c1-c3 ea-c4 >= 0, na1 = 0,
     to get -c3 ea-c4 >= 0, na1 = 0
 - In compound assumption c1-c6 eb1-c7 >= 0, nb1 = 0,
     to get -c6 eb1-c7 >= 0, nb1 = 0
 - In compound assumption -na1-nb1+y1 >= 0, c1 = 0,
     to get -na1-nb1+y1 >= 0, 0 = 0
Eliminated 0 = 1, to get -na1-nb1+y1 >= 0
Conclusion -2 ea >= 0 has been proven.
Proof finished, all conclusions proven.

Inferences    =   5282
Unifications  =   9765
Time (msec)   =   7600
Speed (LIPS)  =    695

Runtime stack =  6 % (2193 words)
Copy stack    =  5 % (1664 words)
Trail stack   =  0 % (323 words)

yes

?   pf(b7f);
Starting assumptions:
   nb1 > 0   y1 >= 2   ta > 0
   nb1-y1 > -1
Using theorem b2 to add assumption(s):
   c1-ea >= 0   c2-ea >= 0
Using theorem b6 to add assumption(s):
   c1 = 0
Substituting c1 = 0
 - In assumption c1-ea >= 0, to get -ea >= 0
 - In compound assumption c1-c3 ea-c4 >= 0, na1 = 0,
     to get -c3 ea-c4 >= 0, na1 = 0
 - In compound assumption c1-c6 eb1-c7 >= 0, nb1 = 0,
     to get -c6 eb1-c7 >= 0, nb1 = 0
 - In compound assumption -na1-nb1+y1 >= 0, c1 = 0,
     to get -na1-nb1+y1 >= 0, 0 = 0
Contradiction -ea >= 0 and ea > 0

Inferences    =   3777
Unifications  =   6860
Time (msec)   =   5666
Speed (LIPS)  =    666

Runtime stack =  6 % (2004 words)
Copy stack    =  4 % (1576 words)
Trail stack   =  0 % (311 words)

no

?   pf(b8a);
Starting assumptions:
   nb2 > 0   ta > 0   -na1-na2 > -1
Using theorem b1 to add assumption(s):
   c1-2 ea >= 0   c2-2 ea >= 0
Using theorem b5 to add assumption(s):
   c2-eb2 = 0
From ta > 0, and (1-c3)(ta) = 0, conclude c3 = 1
Substituting c3 = 1
 - In compound assumption c1-c3 ea-c4 >= 0, na1 = 0,
     to get c1-c4-ea >= 0, na1 = 0
 - In compound assumption c2-c3 ea-c4 >= 0, na2 = 0,
     to get c2-c4-ea >= 0, na2 = 0
 - In compound assumption -c3 ea+c4+c5 >= 0, pa = 0,
     to get c4+c5-ea >= 0, pa = 0
 - In compound assumption ea na1+ea na2+ea pa-ga1-ga2+ta >= 0, c3 = 0,
     to get ea na1+ea na2+ea pa-ga1-ga2+ta >= 0, 0 = 1
Eliminated 0 = 1, to get ea na1+ea na2+ea pa-ga1-ga2+ta = 0
From nb2 > 0, and (c2-eb2)(nb2) = 0, conclude c2-eb2 = 0
From c2 > 0, and (-na2-nb2+y2)(c2) = 0, conclude na2+nb2-y2 = 0
 by merging c2-eb2 = 0 and eb2 > 0
Conclude -2 ea+eb2 >= 0 by merging c2-eb2 = 0 and c2-2 ea >= 0
Conclusion -2 ea+eb2 >= 0 has been proven.
Proof finished, all conclusions proven.

Inferences    =  67102
Unifications  = 126609
Time (msec)   =  83600
Speed (LIPS)  =    802

Runtime stack = 13 % (4344 words)
Copy stack    =  5 % (1912 words)
Trail stack   =  1 % (413 words)

yes

?   pf(b8b);
Starting assumptions:
   nb2 > 0   ta > 0
Using theorem b2 to add assumption(s):
   c1-ea >= 0   c2-ea >= 0
Using theorem b5 to add assumption(s):
   c2-eb2 = 0
From ta > 0, and (1-c3)(ta) = 0, conclude c3 = 1
Substituting c3 = 1
 - In compound assumption c1-c3 ea-c4 >= 0, na1 = 0,
     to get c1-c4-ea >= 0, na1 = 0
 - In compound assumption c2-c3 ea-c4 >= 0, na2 = 0,
     to get c2-c4-ea >= 0, na2 = 0
 - In compound assumption -c3 ea+c4+c5 >= 0, pa = 0,
     to get c4+c5-ea >= 0, pa = 0
 - In compound assumption ea na1+ea na2+ea pa-ga1-ga2+ta >= 0, c3 = 0,
     to get ea na1+ea na2+ea pa-ga1-ga2+ta >= 0, 0 = 1
Eliminated 0 = 1, to get ea na1+ea na2+ea pa-ga1-ga2+ta = 0
From nb2 > 0, and (c2-eb2)(nb2) = 0, conclude c2-eb2 = 0
From c2 > 0, and (-na2-nb2+y2)(c2) = 0, conclude na2+nb2-y2 = 0
 by merging c2-eb2 = 0 and eb2 > 0
Conclude -ea+eb2 >= 0 by merging c2-eb2 = 0 and c2-ea >= 0
Conclusion -ea+eb2 >= 0 has been proven.
Proof finished, all conclusions proven.

Inferences    =  64989
Unifications  = 122397
Time (msec)   =  80250
Speed (LIPS)  =    809

Runtime stack = 12 % (4200 words)
Copy stack    =  5 % (1890 words)
Trail stack   =  1 % (407 words)
```

```
yes
```

```
?   pf(c1a);
Starting assumptions:
   na1 > 0   sb1 > 0
Using theorem a4a to add assumption(s):
   pa > 0
From pa > 0, and (-c3 ea+c4+c5)(pa) = 0, conclude c3 ea-c4-c5 = 0
From sb1 > 0, and (1-c6)(sb1) = 0, conclude c6 = 1
Substituting c6 = 1
 - In compound assumption c1-c6 eb1-c7 >= 0, nb1 = 0,
     to get c1-c7-eb1 >= 0, nb1 = 0
 - In compound assumption -c6 eb1+c7+c8 >= 0, pb1 = 0,
     to get c7+c8-eb1 >= 0, pb1 = 0
 - In compound assumption eb1 nb1+eb1 pb1-gb1+sb1 >= 0, c6 = 0,
     to get eb1 nb1+eb1 pb1-gb1+sb1 >= 0, 0 = 1
Eliminated 0 = 1, to get eb1 nb1+eb1 pb1-gb1+sb1 = 0
From na1 > 0, and (c1-c3 ea-c4)(na1) = 0, conclude c1-c3 ea-c4 = 0
Conclude -c1+c4+ea >= 0 by merging -c3 >= -1 and -c1+c3 ea+c4 = 0
Conclusion -c1+c4+ea >= 0 has been proven.
Conclude c3 ea-c4 >= 0 by merging c3 ea-c4-c5 = 0 and c5 >= 0
Conclusion c3 ea-c4 >= 0 has been proven.
Conclude -c4+ea >= 0 by merging -c3 >= -1 and c3 ea-c4 >= 0
Conclusion -c4+ea >= 0 has been proven.
Conclude -c1+2 ea >= 0 by merging -c4+ea >= 0 and -c1+c4+ea >= 0
Conclusion -c1+2 ea >= 0 has been proven.
Proof finished, all conclusions proven.

Inferences    = 115727
Unifications  = 227363
Time (msec)   = 177250
Speed (LIPS)  =    652

Runtime stack = 15 % (5220 words)
Copy stack    =  5 % (1946 words)
Trail stack   =  1 % (408 words)

yes

?   pf(c1b);
Starting assumptions:
   na2 > 0
Using theorem a4b to add assumption(s):
   pa > 0
From pa > 0, and (-c3 ea+c4+c5)(pa) = 0, conclude c3 ea-c4-c5 = 0
From na2 > 0, and (c2-c3 ea-c4)(na2) = 0, conclude c2-c3 ea-c4 = 0
Conclude -c2+c4+ea >= 0 by merging -c3 >= -1 and -c2+c3 ea+c4 = 0
Conclusion -c2+c4+ea >= 0 has been proven.
Conclude c3 ea-c4 >= 0 by merging c3 ea-c4-c5 = 0 and c5 >= 0
Conclusion c3 ea-c4 >= 0 has been proven.
Conclude -c4+ea >= 0 by merging -c3 >= -1 and c3 ea-c4 >= 0
Conclusion -c4+ea >= 0 has been proven.
Conclude -c2+2 ea >= 0 by merging -c4+ea >= 0 and -c2+c4+ea >= 0
Conclusion -c2+2 ea >= 0 has been proven.
Proof finished, all conclusions proven.

Inferences    = 103253
Unifications  = 194666
Time (msec)   = 133766
Speed (LIPS)  =    771

Runtime stack = 11 % (3684 words)
Copy stack    =  4 % (1618 words)
Trail stack   =  0 % (292 words)

yes

?   pf(c2a);
```

```
Starting assumptions:
   na1+na2 > 1   na1 > 0
From na1 > 0, and (c1-c3 ea-c4)(na1) = 0, conclude c1-c3 ea-c4 = 0
From na1+na2-pa > 0, and (na1+na2-pa)(c4) = 0, conclude c4 = 0
 by merging na1+na2 > 1 and -pa >= -1
Substituting c4 = 0
 - In assumption c1-c3 ea-c4 = 0, to get c1-c3 ea = 0
 - In compound assumption c2-c3 ea-c4 >= 0, na2 = 0,
     to get c2-c3 ea >= 0, na2 = 0
 - In compound assumption -c3 ea+c4+c5 >= 0, pa = 0,
     to get -c3 ea+c5 >= 0, pa = 0
Conclude -c1+ea >= 0 by merging -c3 >= -1 and -c1+c3 ea = 0
Conclusion -c1+ea >= 0 has been proven.
Proof finished, all conclusions proven.

Inferences    =  51547
Unifications  =  98886
Time (msec)   =  68700
Speed (LIPS)  =    750

Runtime stack =  8 % (2723 words)
Copy stack    =  5 % (1664 words)
Trail stack   =  0 % (316 words)

yes

?   pf(c2b);
Starting assumptions:
   na1+na2 > 1   na2 > 0
From na2 > 0, and (c2-c3 ea-c4)(na2) = 0, conclude c2-c3 ea-c4 = 0
From na1+na2-pa > 0, and (na1+na2-pa)(c4) = 0, conclude c4 = 0
 by merging na1+na2 > 1 and -pa >= -1
Substituting c4 = 0
 - In assumption c2-c3 ea-c4 = 0, to get c2-c3 ea = 0
 - In compound assumption c1-c3 ea-c4 >= 0, na1 = 0,
     to get c1-c3 ea >= 0, na1 = 0
 - In compound assumption -c3 ea+c4+c5 >= 0, pa = 0,
     to get -c3 ea+c5 >= 0, pa = 0
Conclude -c2+ea >= 0 by merging -c3 >= -1 and -c2+c3 ea = 0
Conclusion -c2+ea >= 0 has been proven.
Proof finished, all conclusions proven.

Inferences    =  52193
Unifications  = 100171
Time (msec)   =  69316
Speed (LIPS)  =    752

Runtime stack =  8 % (2720 words)
Copy stack    =  5 % (1670 words)
Trail stack   =  0 % (319 words)

yes

?   pf(c3);
Starting assumptions:
   -nb1 > -1   sb1 > 0
From sb1 > 0, and (1-c6)(sb1) = 0, conclude c6 = 1
Substituting c6 = 1
 - In compound assumption c1-c6 eb1-c7 >= 0, nb1 = 0,
     to get c1-c7-eb1 >= 0, nb1 = 0
 - In compound assumption -c6 eb1+c7+c8 >= 0, pb1 = 0,
     to get c7+c8-eb1 >= 0, pb1 = 0
 - In compound assumption eb1 nb1+eb1 pb1-gb1+sb1 >= 0, c6 = 0,
     to get eb1 nb1+eb1 pb1-gb1+sb1 >= 0, 0 = 1
Eliminated 0 = 1, to get eb1 nb1+eb1 pb1-gb1+sb1 = 0
From -pb1 > -1, and (1-pb1)(c8) = 0, conclude c8 = 0
 by merging -nb1 > -1 and nb1-pb1 >= 0
Substituting c8 = 0
 - In compound assumption c7+c8-eb1 >= 0, pb1 = 0,
     to get c7-eb1 >= 0, pb1 = 0
From c7 > 0, and (nb1-pb1)(c7) = 0, conclude nb1-pb1 = 0
 by merging eb1 > 0 and c7-eb1 >= 0
From c2 > 0, and (-na2-nb2+y2)(c2) = 0, conclude na2+nb2-y2 = 0
 by merging eb2 > 0 and c2-eb2 >= 0
Conclude c1-2 eb1 >= 0 by merging c7-eb1 >= 0 and c1-c7-eb1 >= 0
Conclusion c1-2 eb1 >= 0 has been proven.
Proof finished, all conclusions proven.

Inferences    = 176035
Unifications  = 378770
Time (msec)   = 244633
Speed (LIPS)  =    719

Runtime stack = 16 % (5437 words)
Copy stack    =  6 % (2220 words)
Trail stack   =  1 % (546 words)

yes

?   pf(c4);
Starting assumptions:
   sb1 > 0
From sb1 > 0, and (1-c6)(sb1) = 0, conclude c6 = 1
Substituting c6 = 1
 - In compound assumption c1-c6 eb1-c7 >= 0, nb1 = 0,
     to get c1-c7-eb1 >= 0, nb1 = 0
 - In compound assumption -c6 eb1+c7+c8 >= 0, pb1 = 0,
     to get c7+c8-eb1 >= 0, pb1 = 0
 - In compound assumption eb1 nb1+eb1 pb1-gb1+sb1 >= 0, c6 = 0,
     to get eb1 nb1+eb1 pb1-gb1+sb1 >= 0, 0 = 1
Eliminated 0 = 1, to get eb1 nb1+eb1 pb1-gb1+sb1 = 0
From c2 > 0, and (-na2-nb2+y2)(c2) = 0, conclude na2+nb2-y2 = 0
 by merging eb2 > 0 and c2-eb2 >= 0
Conclude c1-eb1 >= 0 by merging c7 >= 0 and c1-c7-eb1 >= 0
Conclusion c1-eb1 >= 0 has been proven.
Proof finished, all conclusions proven.

Inferences    =  90665
Unifications  = 195074
Time (msec)   = 125000
Speed (LIPS)  =    725

Runtime stack = 10 % (3264 words)
Copy stack    =  5 % (1754 words)
Trail stack   =  1 % (375 words)

yes

?   pf(c5_1);
Starting assumptions:
   na1+na2-y1-y2 > -1   y1+y2 >= 2   na1 > 0
   sb1 > 0
Using theorem a4a to add assumption(s):
   pa > 0
From pa > 0, and (-c3 ea+c4+c5)(pa) = 0, conclude c3 ea-c4-c5 = 0
From sb1 > 0, and (1-c6)(sb1) = 0, conclude c6 = 1
Substituting c6 = 1
 - In compound assumption c1-c6 eb1-c7 >= 0, nb1 = 0,
     to get c1-c7-eb1 >= 0, nb1 = 0
 - In compound assumption -c6 eb1+c7+c8 >= 0, pb1 = 0,
     to get c7+c8-eb1 >= 0, pb1 = 0
 - In compound assumption eb1 nb1+eb1 pb1-gb1+sb1 >= 0, c6 = 0,
```

```
     to get eb1 nb1+eb1 pb1-gb1+sb1 >= 0, 0 = 1
Eliminated 0 = 1, to get eb1 nb1+eb1 pb1-gb1+sb1 = 0
From na1 > 0, and (c1-c3 ea-c4)(na1) = 0, conclude c1-c3 ea-c4 = 0
Conclude na1+na2 > 1 by merging y1+y2 >= 2 and na1+na2-y1-y2 > -1
Conclusion na1+na2 > 1 has been proven.
From na1+na2-pa > 0, and (na1+na2-pa)(c4) = 0, conclude c4 = 0
 by merging na1+na2 > 1 and -pa >= -1
Substituting c4 = 0
 - In assumption c1-c3 ea-c4 = 0, to get c1-c3 ea = 0
 - In assumption c3 ea-c4-c5 = 0, to get c3 ea-c5 = 0
 - In compound assumption c2-c3 ea-c4 >= 0, na2 = 0,
     to get c2-c3 ea >= 0, na2 = 0
Conclusion c3 ea-c5 = 0 has been proven.
Proof finished, all conclusions proven.

Inferences    =  98627
Unifications  = 198576
Time (msec)   = 137150
Speed (LIPS)  =    719

Runtime stack = 15 % (4966 words)
Copy stack    =  6 % (2088 words)
Trail stack   =  1 % (481 words)

yes

?   pf(c5_2);
Starting assumptions:
   na1+na2-y1-y2 > -1   y1+y2 >= 2   na2 > 0
Using theorem a4b to add assumption(s):
   pa > 0
From pa > 0, and (-c3 ea+c4+c5)(pa) = 0, conclude c3 ea-c4-c5 = 0
From na2 > 0, and (c2-c3 ea-c4)(na2) = 0, conclude c2-c3 ea-c4 = 0
Conclude na1+na2 > 1 by merging y1+y2 >= 2 and na1+na2-y1-y2 > -1
Conclusion na1+na2 > 1 has been proven.
From na1+na2-pa > 0, and (na1+na2-pa)(c4) = 0, conclude c4 = 0
 by merging na1+na2 > 1 and -pa >= -1
Substituting c4 = 0
 - In assumption c2-c3 ea-c4 = 0, to get c2-c3 ea = 0
 - In assumption c3 ea-c4-c5 = 0, to get c3 ea-c5 = 0
 - In compound assumption c1-c3 ea-c4 >= 0, na1 = 0,
     to get c1-c3 ea >= 0, na1 = 0
Conclusion c3 ea-c5 = 0 has been proven.
Proof finished, all conclusions proven.

Inferences    =  86358
Unifications  = 165715
Time (msec)   = 114750
Speed (LIPS)  =    752

Runtime stack = 10 % (3388 words)
Copy stack    =  5 % (1764 words)
Trail stack   =  1 % (363 words)

yes

?   pf(c5_3);
Starting assumptions:
   c3 = 0   c4 = 0   na1 > 0
Substituting c4 = 0
 - In compound assumption c1-c3 ea-c4 >= 0, na1 = 0,
     to get c1-c3 ea >= 0, na1 = 0
 - In compound assumption c2-c3 ea-c4 >= 0, na2 = 0,
     to get c2-c3 ea >= 0, na2 = 0
 - In compound assumption -c3 ea+c4+c5 >= 0, pa = 0,
     to get -c3 ea+c5 >= 0, pa = 0
 - In compound assumption na1+na2-pa >= 0, c4 = 0,
     to get na1+na2-pa >= 0, 0 = 0
Eliminated 0 = 0, to get na1+na2-pa >= 0
Substituting c3 = 0
 - In compound assumption -c3 ea+c5 >= 0, pa = 0,
     to get c5 >= 0, pa = 0
 - In compound assumption c2-c3 ea >= 0, na2 = 0,
     to get c2 >= 0, na2 = 0
 - In compound assumption c1-c3 ea >= 0, na1 = 0,
     to get c1 >= 0, na1 = 0
 - In compound assumption -c3 >= -1, ta = 0,
     to get 0 >= -1, ta = 0
 - In compound assumption ea na1+ea na2+ea pa-ga1-ga2+ta >= 0, c3 = 0,
     to get ea na1+ea na2+ea pa-ga1-ga2+ta >= 0, 0 = 0
Eliminated 0 >= -1, to get ta = 0
Eliminated 0 = 0, to get ea na1+ea na2+ea pa-ga1-ga2+ta >= 0
Substituting ta = 0
 - In assumption ea na1+ea na2+ea pa-ga1-ga2+ta >= 0,
                                    to get ea na1+ea na2+ea pa-ga1-ga2 >= 0
From na1 > 0, and (c1)(na1) = 0, conclude c1 = 0
Conclusion c1 = 0 has been proven.
Proof finished, all conclusions proven.

Inferences    =  11123
Unifications  =  19898
Time (msec)   =  16816
Speed (LIPS)  =    661

Runtime stack = 11 % (3819 words)
Copy stack    =  5 % (1942 words)
Trail stack   =  1 % (418 words)

yes

?   pf(c5_4);
Starting assumptions:
   c3 = 0   c4 = 0   na2 > 0
Substituting c4 = 0
 - In compound assumption c1-c3 ea-c4 >= 0, na1 = 0,
     to get c1-c3 ea >= 0, na1 = 0
 - In compound assumption c2-c3 ea-c4 >= 0, na2 = 0,
     to get c2-c3 ea >= 0, na2 = 0
 - In compound assumption -c3 ea+c4+c5 >= 0, pa = 0,
     to get -c3 ea+c5 >= 0, pa = 0
 - In compound assumption na1+na2-pa >= 0, c4 = 0,
     to get na1+na2-pa >= 0, 0 = 0
Eliminated 0 = 0, to get na1+na2-pa >= 0
Substituting c3 = 0
 - In compound assumption -c3 ea+c5 >= 0, pa = 0,
     to get c5 >= 0, pa = 0
 - In compound assumption c2-c3 ea >= 0, na2 = 0,
     to get c2 >= 0, na2 = 0
 - In compound assumption c1-c3 ea >= 0, na1 = 0,
     to get c1 >= 0, na1 = 0
 - In compound assumption -c3 >= -1, ta = 0,
     to get 0 >= -1, ta = 0
 - In compound assumption ea na1+ea na2+ea pa-ga1-ga2+ta >= 0, c3 = 0,
     to get ea na1+ea na2+ea pa-ga1-ga2+ta >= 0, 0 = 0
Eliminated 0 >= -1, to get ta = 0
Eliminated 0 = 0, to get ea na1+ea na2+ea pa-ga1-ga2+ta >= 0
Substituting ta = 0
 - In assumption ea na1+ea na2+ea pa-ga1-ga2+ta >= 0,
                                    to get ea na1+ea na2+ea pa-ga1-ga2 >= 0
From na2 > 0, and (c2)(na2) = 0, conclude c2 = 0
Conclusion c2 = 0 has been proven.
Proof finished, all conclusions proven.
```

```
Inferences    =  11330
Unifications  =  20308
Time (msec)   =  16866
Speed (LIPS)  =    671

Runtime stack = 11 % (3826 words)
Copy stack    =  5 % (1944 words)
Trail stack   =  1 % (419 words)

yes

?   pf(c5_5);
Starting assumptions:
   c3 > 0   c5 > 0   na1+na2-y1-y2 > -1
From c5 > 0, and (1-pa)(c5) = 0, conclude pa = 1
Substituting pa = 1
 - In compound assumption -c3 ea+c4+c5 >= 0, pa = 0,
     to get -c3 ea+c4+c5 >= 0, 0 = 1
 - In compound assumption ea na1+ea na2+ea pa-ga1-ga2+ta >= 0, c3 = 0,
     to get ea+ea na1+ea na2-ga1-ga2+ta >= 0, c3 = 0
 - In compound assumption na1+na2-pa >= 0, c4 = 0,
     to get na1+na2 >= 1, c4 = 0
Eliminated 0 = 1, to get c3 ea-c4-c5 = 0
From c3 > 0, and (ea+ea na1+ea na2-ga1-ga2+ta)(c3) = 0,
                                  conclude ea+ea na1+ea na2-ga1-ga2+ta = 0
Conclude -ea-ea na1-ea na2+ga1+ga2 >= 0 by merging
                                  ea+ea na1+ea na2-ga1-ga2+ta = 0 and ta >= 0
Conclusion -ea-ea na1-ea na2+ga1+ga2 >= 0 has been proven.
Conclude -na1-na2+y1+y2 >= 1 by merging -ea-ea na1-ea na2+ga1+ga2 >= 0
                                  and ea y1+ea y2-ga1-ga2 = 0
Contradiction -na1-na2+y1+y2 >= 1 and na1+na2-y1-y2 > -1

Inferences    = 108293
Unifications  = 244350
Time (msec)   = 159950
Speed (LIPS)  =    677

Runtime stack = 14 % (4713 words)
Copy stack    =  5 % (1846 words)
Trail stack   =  1 % (414 words)

no

?   pf(c6a);
Starting assumptions:
   sb1 > 0   na1 > 0   y1+y2 >= 2
   -nb1 > -1
Using theorem c1a to add assumption(s):
   -c1+2 ea >= 0
Using theorem c3 to add assumption(s):
   c1-2 eb1 >= 0
From na1 > 0, and (c1-c3 ea-c4)(na1) = 0, conclude c1-c3 ea-c4 = 0
From sb1 > 0, and (1-c6)(sb1) = 0, conclude c6 = 1
Substituting c6 = 1
 - In compound assumption c1-c6 eb1-c7 >= 0, nb1 = 0,
     to get c1-c7-eb1 >= 0, nb1 = 0
 - In compound assumption -c6 eb1+c7+c8 >= 0, pb1 = 0,
     to get c7+c8-eb1 >= 0, pb1 = 0
 - In compound assumption eb1 nb1+eb1 pb1-gb1+sb1 >= 0, c6 = 0,
     to get eb1 nb1+eb1 pb1-gb1+sb1 >= 0, 0 = 1
Eliminated 0 = 1, to get eb1 nb1+eb1 pb1-gb1+sb1 = 0
Conclude 2 ea-2 eb1 >= 0 by merging c1-2 eb1 >= 0 and -c1+2 ea >= 0
Conclusion 2 ea-2 eb1 >= 0 has been proven.
Proof finished, all conclusions proven.

Inferences    =  64352
Unifications  = 131688
Time (msec)   =  89200
Speed (LIPS)  =    721

Runtime stack = 10 % (3323 words)
Copy stack    =  5 % (1746 words)
Trail stack   =  1 % (356 words)

yes

?   pf(c6b);
Starting assumptions:
   sb1 > 0   na1 > 0   y1+y2 >= 2
Using theorem c1a to add assumption(s):
   -c1+2 ea >= 0
Using theorem c4 to add assumption(s):
   c1-eb1 >= 0
From na1 > 0, and (c1-c3 ea-c4)(na1) = 0, conclude c1-c3 ea-c4 = 0
From sb1 > 0, and (1-c6)(sb1) = 0, conclude c6 = 1
Substituting c6 = 1
 - In compound assumption c1-c6 eb1-c7 >= 0, nb1 = 0,
     to get c1-c7-eb1 >= 0, nb1 = 0
 - In compound assumption -c6 eb1+c7+c8 >= 0, pb1 = 0,
     to get c7+c8-eb1 >= 0, pb1 = 0
 - In compound assumption eb1 nb1+eb1 pb1-gb1+sb1 >= 0, c6 = 0,
     to get eb1 nb1+eb1 pb1-gb1+sb1 >= 0, 0 = 1
Eliminated 0 = 1, to get eb1 nb1+eb1 pb1-gb1+sb1 = 0
Conclude 2 ea-eb1 >= 0 by merging c1-eb1 >= 0 and -c1+2 ea >= 0
Conclusion 2 ea-eb1 >= 0 has been proven.
Proof finished, all conclusions proven.

Inferences    =  60263
Unifications  = 124419
Time (msec)   =  82566
Speed (LIPS)  =    729

Runtime stack =  9 % (3200 words)
Copy stack    =  5 % (1728 words)
Trail stack   =  1 % (352 words)

yes

?   pf(c6c);
Starting assumptions:
   sb1 > 0   na1 > 0   y1+y2 >= 2
   -nb1 > -1   na1+na2 > 1
Using theorem c2a to add assumption(s):
   -c1+ea >= 0
Using theorem c3 to add assumption(s):
   c1-2 eb1 >= 0
From na1 > 0, and (c1-c3 ea-c4)(na1) = 0, conclude c1-c3 ea-c4 = 0
From sb1 > 0, and (1-c6)(sb1) = 0, conclude c6 = 1
Substituting c6 = 1
 - In compound assumption c1-c6 eb1-c7 >= 0, nb1 = 0,
     to get c1-c7-eb1 >= 0, nb1 = 0
 - In compound assumption -c6 eb1+c7+c8 >= 0, pb1 = 0,
     to get c7+c8-eb1 >= 0, pb1 = 0
 - In compound assumption eb1 nb1+eb1 pb1-gb1+sb1 >= 0, c6 = 0,
     to get eb1 nb1+eb1 pb1-gb1+sb1 >= 0, 0 = 1
Eliminated 0 = 1, to get eb1 nb1+eb1 pb1-gb1+sb1 = 0
Conclude ea-2 eb1 >= 0 by merging c1-2 eb1 >= 0 and -c1+ea >= 0
Conclusion ea-2 eb1 >= 0 has been proven.
Proof finished, all conclusions proven.

Inferences    =  65729
```

```
Unifications  = 134611
Time (msec)   =  89250
Speed (LIPS)  =    736

Runtime stack = 10 % (3383 words)
Copy stack    =  5 % (1752 words)
Trail stack   =  1 % (358 words)

yes

?   pf(c6d);
Starting assumptions:
   sb1 > 0   na1 > 0   y1+y2 >= 2
   na1+na2 > 1
Using theorem c2a to add assumption(s):
   -c1+ea >= 0
Using theorem c4 to add assumption(s):
   c1-eb1 >= 0
From na1 > 0, and (c1-c3 ea-c4)(na1) = 0, conclude c1-c3 ea-c4 = 0
From sb1 > 0, and (1-c6)(sb1) = 0, conclude c6 = 1
Substituting c6 = 1
 - In compound assumption c1-c6 eb1-c7 >= 0, nb1 = 0,
     to get c1-c7-eb1 >= 0, nb1 = 0
 - In compound assumption -c6 eb1+c7+c8 >= 0, pb1 = 0,
     to get c7+c8-eb1 >= 0, pb1 = 0
 - In compound assumption eb1 nb1+eb1 pb1-gb1+sb1 >= 0, c6 = 0,
     to get eb1 nb1+eb1 pb1-gb1+sb1 >= 0, 0 = 1
Eliminated 0 = 1, to get eb1 nb1+eb1 pb1-gb1+sb1 = 0
Conclude ea-eb1 >= 0 by merging c1-eb1 >= 0 and -c1+ea >= 0
Conclusion ea-eb1 >= 0 has been proven.
Proof finished, all conclusions proven.

Inferences    =  61602
Unifications  = 127285
Time (msec)   =  84650
Speed (LIPS)  =    727

Runtime stack =  9 % (3261 words)
Copy stack    =  5 % (1734 words)
Trail stack   =  1 % (354 words)

yes

?   pf(c6e);
Starting assumptions:
   sb1 > 0   na1 > 0   y1+y2 >= 2
   -nb1 > -1   na1+na2-y1-y2 > -1
Using theorem c3 to add assumption(s):
   c1-2 eb1 >= 0
Using theorem c5a to add assumption(s):
   c1 = 0
Substituting c1 = 0
 - In assumption c1-2 eb1 >= 0, to get -2 eb1 >= 0
 - In compound assumption c1-c3 ea-c4 >= 0, na1 = 0,
     to get -c3 ea-c4 >= 0, na1 = 0
 - In compound assumption c1-c6 eb1-c7 >= 0, nb1 = 0,
     to get -c6 eb1-c7 >= 0, nb1 = 0
 - In compound assumption -na1-nb1+y1 >= 0, c1 = 0,
     to get -na1-nb1+y1 >= 0, 0 = 0
Eliminated 0 = 0, to get -na1-nb1+y1 >= 0
Conclusion -2 eb1 >= 0 has been proven.
Proof finished, all conclusions proven.

Inferences    =   5165
Unifications  =   9528
Time (msec)   =   7933
Speed (LIPS)  =    651

Runtime stack =  6 % (2104 words)
Copy stack    =  4 % (1618 words)
Trail stack   =  0 % (301 words)

yes

?   pf(c6f);
Starting assumptions:
   sb1 > 0   na1 > 0   y1+y2 >= 2
   na1+na2-y1-y2 > -1
Using theorem c4 to add assumption(s):
   c1-eb1 >= 0
Using theorem c5a to add assumption(s):
   c1 = 0
Substituting c1 = 0
 - In assumption c1-eb1 >= 0, to get -eb1 >= 0
 - In compound assumption c1-c3 ea-c4 >= 0, na1 = 0,
     to get -c3 ea-c4 >= 0, na1 = 0
 - In compound assumption c1-c6 eb1-c7 >= 0, nb1 = 0,
     to get -c6 eb1-c7 >= 0, nb1 = 0
 - In compound assumption -na1-nb1+y1 >= 0, c1 = 0,
     to get -na1-nb1+y1 >= 0, 0 = 0
Contradiction -eb1 >= 0 and eb1 > 0

Inferences    =   3672
Unifications  =   6653
Time (msec)   =   5933
Speed (LIPS)  =    618

Runtime stack =  5 % (1925 words)
Copy stack    =  4 % (1530 words)
Trail stack   =  0 % (289 words)

no

?   pf(c7a);
Starting assumptions:
   y1+y2 >= 2   na2 > 0
Using theorem c1b to add assumption(s):
   -c2+2 ea >= 0
From na2 > 0, and (c2-c3 ea-c4)(na2) = 0, conclude c2-c3 ea-c4 = 0
Conclude 2 ea-eb2 >= 0 by merging -c2+2 ea >= 0 and c2-eb2 >= 0
Conclusion 2 ea-eb2 >= 0 has been proven.
Proof finished, all conclusions proven.

Inferences    =  40863
Unifications  =  78272
Time (msec)   =  55650
Speed (LIPS)  =    734

Runtime stack =  4 % (1504 words)
Copy stack    =  4 % (1356 words)
Trail stack   =  0 % (219 words)

yes

?   pf(c7b);
Starting assumptions:
   y1+y2 >= 2   na2 > 0   na1+na2 > 1
Using theorem c2b to add assumption(s):
   -c2+ea >= 0
From na2 > 0, and (c2-c3 ea-c4)(na2) = 0, conclude c2-c3 ea-c4 = 0
Conclude ea-eb2 >= 0 by merging -c2+ea >= 0 and c2-eb2 >= 0
Conclusion ea-eb2 >= 0 has been proven.
```

```
Proof finished, all conclusions proven.

Inferences    =  41426
Unifications  =  79434
Time (msec)   =  56500
Speed (LIPS)  =    733

Runtime stack =  5 % (1632 words)
Copy stack    =  4 % (1372 words)
Trail stack   =  0 % (221 words)

yes

?   pf(c7c);
Starting assumptions:
   y1+y2 >= 2   na2 > 0   na1+na2-y1-y2 > -1
Using theorem c5b to add assumption(s):
   c2 = 0
Substituting c2 = 0
 - In compound assumption c2-c3 ea-c4 >= 0, na2 = 0,
     to get -c3 ea-c4 >= 0, na2 = 0
 - In compound assumption c2-eb2 >= 0, nb2 = 0,
     to get -eb2 >= 0, nb2 = 0
 - In compound assumption -na2-nb2+y2 >= 0, c2 = 0,
     to get -na2-nb2+y2 >= 0, 0 = 0
Contradiction -eb2 >= 0 and eb2 > 0

Inferences    =   3085
Unifications  =   5580
Time (msec)   =   5066
Speed (LIPS)  =    608

Runtime stack =  4 % (1499 words)
Copy stack    =  4 % (1528 words)
Trail stack   =  0 % (274 words)

no
```

```
?   pf(d1);
No starting assumptions.
From c2 > 0, and (-na2-nb2+y2)(c2) = 0, conclude na2+nb2-y2 = 0
 by merging eb2 > 0 and c2-eb2 >= 0
Conclusion na2+nb2-y2 = 0 has been proven.
Proof finished, all conclusions proven.

Inferences    =  25952
Unifications  =  48630
Time (msec)   =  50750
Speed (LIPS)  =    511

Runtime stack =  3 % (1063 words)
Copy stack    =  4 % (1320 words)
Trail stack   =  0 % (212 words)

yes

?   pf(d2_1);
Starting assumptions:
   ta = 0
Using theorem z4 to add assumption(s):
   na1+na2+pa-y1-y2 >= 0
Substituting ta = 0
 - In compound assumption -c3 >= -1, ta = 0,
     to get -c3 >= -1, 0 = 0
 - In compound assumption ea na1+ea na2+ea pa-ga1-ga2+ta >= 0, c3 = 0,
     to get ea na1+ea na2+ea pa-ga1-ga2 >= 0, c3 = 0
Eliminated 0 = 0, to get -c3 >= -1
Conclude 2 na1+2 na2-y1-y2 >= 0 by merging na1+na2+pa-y1-y2 >= 0
                                          and na1+na2-pa >= 0
Conclusion 2 na1+2 na2-y1-y2 >= 0 has been proven.
Proof finished, all conclusions proven.

Inferences    =  33754
Unifications  =  69586
Time (msec)   =  47500
Speed (LIPS)  =    710

Runtime stack =  6 % (1971 words)
Copy stack    =  4 % (1536 words)
Trail stack   =  0 % (278 words)

yes

?   pf(d2_2);
Starting assumptions:
   sb1 = 0
Substituting sb1 = 0
 - In compound assumption -c6 >= -1, sb1 = 0,
     to get -c6 >= -1, 0 = 0
 - In compound assumption eb1 nb1+eb1 pb1-gb1+sb1 >= 0, c6 = 0,
     to get eb1 nb1+eb1 pb1-gb1 >= 0, c6 = 0
Eliminated 0 = 0, to get -c6 >= -1
From c2 > 0, and (-na2-nb2+y2)(c2) = 0, conclude na2+nb2-y2 = 0
 by merging eb2 > 0 and c2-eb2 >= 0
Conclude nb1+pb1-y1 >= 0 by merging eb1 y1-gb1 = 0 and eb1 nb1+eb1 pb1-gb1 >= 0
Conclusion nb1+pb1-y1 >= 0 has been proven.
Conclude 2 nb1-y1 >= 0 by merging nb1+pb1-y1 >= 0 and nb1-pb1 >= 0
Conclusion 2 nb1-y1 >= 0 has been proven.
Proof finished, all conclusions proven.

Inferences    =  95061
Unifications  = 185868
Time (msec)   = 125666
Speed (LIPS)  =    756
```

```
Runtime stack = 12 % (3971 words)
Copy stack    =  5 % (1760 words)
Trail stack   =  1 % (350 words)

yes

?   pf(d2_3);
Starting assumptions:
   eb2 nb2-gb2 >= 0
From nb2 > 0, and (c2-eb2)(nb2) = 0, conclude c2-eb2 = 0
 by merging eb2 nb2-gb2 >= 0 and gb2 > 0
From c2 > 0, and (-na2-nb2+y2)(c2) = 0, conclude na2+nb2-y2 = 0
 by merging c2-eb2 = 0 and eb2 > 0
Conclude nb2-y2 >= 0 by merging eb2 nb2-gb2 >= 0 and eb2 y2-gb2 = 0
Conclusion nb2-y2 >= 0 has been proven.
Proof finished, all conclusions proven.

Inferences    =  75466
Unifications  = 141870
Time (msec)   = 111983
Speed (LIPS)  =    673

Runtime stack =  8 % (2904 words)
Copy stack    =  4 % (1622 words)
Trail stack   =  0 % (298 words)

yes

?   pf(d2_4);
Starting assumptions:
   -na1-nb1+y1 > 0   2 nb1-y1 >= 0   2 na1+2 na2-y1-y2 >= 0
   nb2-y2 >= 0
Using theorem d1 to add assumption(s):
   na2+nb2-y2 = 0
From -na1-nb1+y1 > 0, and (-na1-nb1+y1)(c1) = 0, conclude c1 = 0
Substituting c1 = 0
 - In compound assumption c1-c3 ea-c4 >= 0, na1 = 0,
     to get -c3 ea-c4 >= 0, na1 = 0
 - In compound assumption c1-c6 eb1-c7 >= 0, nb1 = 0,
     to get -c6 eb1-c7 >= 0, nb1 = 0
-na2 >= 0 by merging na2+nb2-y2 = 0 and nb2-y2 >= 0
Obtained -na2 = 0, from -na2 >= 0 and na2 >= 0
Substituting na2 = 0
 - In assumption na2+nb2-y2 = 0, to get nb2-y2 = 0
 - In assumption 2 na1+2 na2-y1-y2 >= 0, to get 2 na1-y1-y2 >= 0
 - In compound assumption c2-c3 ea-c4 >= 0, na2 = 0,
     to get c2-c3 ea-c4 >= 0, 0 = 0
 - In compound assumption -na2-nb2+y2 >= 0, c2 = 0,
     to get -nb2+y2 >= 0, c2 = 0
 - In compound assumption ea na1+ea na2+ea pa-ga1-ga2+ta >= 0, c3 = 0,
     to get ea na1+ea pa-ga1-ga2+ta >= 0, c3 = 0
 - In compound assumption na1+na2-pa >= 0, c4 = 0,
     to get na1-pa >= 0, c4 = 0
Eliminated 0 = 0, to get c2-c3 ea-c4 >= 0
Conclude 2 na1-y1 > 0 by merging 2 na1-y1-y2 >= 0 and y2 > 0
Conclusion 2 na1-y1 > 0 has been proven.
na1+nb1-y1 > 0 by merging 2 na1-y1 > 0 and 2 nb1-y1 >= 0
Contradiction na1+nb1-y1 > 0 and -na1-nb1+y1 > 0

Inferences    = 165946
Unifications  = 303929
Time (msec)   = 214350
Speed (LIPS)  =    774

Runtime stack = 14 % (4825 words)
Copy stack    =  5 % (1928 words)
Trail stack   =  1 % (395 words)

no

?   pf(d3a);
Starting assumptions:
   ta > 0
Using theorem b2 to add assumption(s):
   c1-ea >= 0   c2-ea >= 0
From ta > 0, and (1-c3)(ta) = 0, conclude c3 = 1
Substituting c3 = 1
 - In compound assumption c1-c3 ea-c4 >= 0, na1 = 0,
     to get c1-c4-ea >= 0, na1 = 0
 - In compound assumption c2-c3 ea-c4 >= 0, na2 = 0,
     to get c2-c4-ea >= 0, na2 = 0
 - In compound assumption -c3 ea+c4+c5 >= 0, pa = 0,
     to get c4+c5-ea >= 0, pa = 0
 - In compound assumption ea na1+ea na2+ea pa-ga1-ga2+ta >= 0, c3 = 0,
     to get ea na1+ea na2+ea pa-ga1-ga2+ta >= 0, 0 = 1
Eliminated 0 = 1, to get ea na1+ea na2+ea pa-ga1-ga2+ta = 0
From c2 > 0, and (-na2-nb2+y2)(c2) = 0, conclude na2+nb2-y2 = 0
 by merging c2-ea >= 0 and ea > 0
From c1 > 0, and (-na1-nb1+y1)(c1) = 0, conclude na1+nb1-y1 = 0
 by merging c1-ea >= 0 and ea > 0
Conclusion na1+nb1-y1 = 0 has been proven.
Proof finished, all conclusions proven.

Inferences    = 137293
Unifications  = 337357
Time (msec)   = 190883
Speed (LIPS)  =    719

Runtime stack = 11 % (3752 words)
Copy stack    =  5 % (1852 words)
Trail stack   =  1 % (410 words)

yes

?   pf(d3b);
   sb1 > 0
Using theorem c4 to add assumption(s):
   c1-eb1 >= 0
From sb1 > 0, and (1-c6)(sb1) = 0, conclude c6 = 1
Substituting c6 = 1
 - In compound assumption c1-c6 eb1-c7 >= 0, nb1 = 0,
     to get c1-c7-eb1 >= 0, nb1 = 0
 - In compound assumption -c6 eb1+c7+c8 >= 0, pb1 = 0,
     to get c7+c8-eb1 >= 0, pb1 = 0
 - In compound assumption eb1 nb1+eb1 pb1-gb1+sb1 >= 0, c6 = 0,
     to get eb1 nb1+eb1 pb1-gb1+sb1 >= 0, 0 = 1
Eliminated 0 = 1, to get eb1 nb1+eb1 pb1-gb1+sb1 = 0
From c1 > 0, and (-na1-nb1+y1)(c1) = 0, conclude na1+nb1-y1 = 0
 by merging c1-eb1 >= 0 and eb1 > 0
Conclusion na1+nb1-y1 = 0 has been proven.
Proof finished, all conclusions proven.

Inferences    =  38178
Unifications  =  78501
Time (msec)   =  49366
Speed (LIPS)  =    773

Runtime stack =  7 % (2533 words)
Copy stack    =  5 % (1662 words)
Trail stack   =  1 % (330 words)
```

```
yes

?   pf(d3c);
Starting assumptions:
   -eb2 nb2+gb2 > 0
Using theorem d1 to add assumption(s):
   na2+nb2-y2 = 0
Conclude -nb2+y2 > 0 by merging -eb2 nb2+gb2 > 0 and eb2 y2-gb2 = 0
Conclusion -nb2+y2 > 0 has been proven.
From na2 > 0, and (c2-c3 ea-c4)(na2) = 0, conclude c2-c3 ea-c4 = 0
 by merging -nb2+y2 > 0 and na2+nb2-y2 = 0
From c2 > 0, and (-na2-nb2+y2)(c2) = 0, conclude na2+nb2-y2 = 0
 by merging eb2 > 0 and c2-eb2 >= 0
Conclude c3 ea+c4 > 0 by merging c2 > 0 and c2-c3 ea-c4 = 0
Conclusion c3 ea+c4 > 0 has been proven.
From c1 > 0, and (-na1-nb1+y1)(c1) = 0, conclude na1+nb1-y1 = 0
 by merging c3 ea+c4 > 0 and c1-c3 ea-c4 >= 0
Conclusion na1+nb1-y1 = 0 has been proven.
Proof finished, all conclusions proven.

Inferences     = 147234
Unifications   = 279650
Time (msec)    = 190133
Speed (LIPS)   =    774

Runtime stack = 14 % (4591 words)
Copy stack    =  5 % (1762 words)
Trail stack   =  1 % (364 words)

yes

?   pf(d4a);
Starting assumptions:
   -na1-nb1+y1 > 0
Using theorem d2a to add assumption(s):
   ta > 0
Using theorem d3a to add assumption(s):
   na1+nb1-y1 = 0
Contradiction na1+nb1-y1 = 0 and -na1-nb1+y1 > 0

Inferences     =    995
Unifications   =   1574
Time (msec)    =   2016
Speed (LIPS)   =    493

Runtime stack =  2 % (662 words)
Copy stack    =  3 % (1258 words)
Trail stack   =  0 % (179 words)

no

?   pf(d4b);
Starting assumptions:
   -na1-nb1+y1 > 0
Using theorem d2b to add assumption(s):
   sb1 > 0
Using theorem d3b to add assumption(s):
   na1+nb1-y1 = 0
Contradiction na1+nb1-y1 = 0 and -na1-nb1+y1 > 0

Inferences     =    994
Unifications   =   1572
Time (msec)    =   2016
Speed (LIPS)   =    492

Runtime stack =  2 % (662 words)
Copy stack    =  3 % (1256 words)
Trail stack   =  0 % (178 words)

no

?   pf(d4c);
Starting assumptions:
   -na1-nb1+y1 > 0
Using theorem d2c to add assumption(s):
   -eb2 nb2+gb2 > 0
Using theorem d3c to add assumption(s):
   na1+nb1-y1 = 0
Contradiction na1+nb1-y1 = 0 and -na1-nb1+y1 > 0

Inferences     =   1134
Unifications   =   1845
Time (msec)    =   2133
Speed (LIPS)   =    531

Runtime stack =  1 % (642 words)
Copy stack    =  3 % (1262 words)
Trail stack   =  0 % (189 words)

no

?   pf(d5);
Starting assumptions:
   nb1-y1 > -1   eb2 nb2-gb2 >= 0   y1+y2 >= 2
Using theorem d1 to add assumption(s):
   na2+nb2-y2 = 0
Using theorem d4 to add assumption(s):
   na1+nb1-y1 = 0
Conclude -na1 > -1 by merging na1+nb1-y1 = 0 and nb1-y1 > -1
Conclusion -na1 > -1 has been proven.
Conclude -na1-pa > -2 by merging -na1 > -1 and -pa >= -1
Conclusion -na1-pa > -2 has been proven.
Conclude -na1-pa+y1+y2 > 0 by merging -na1-pa > -2 and y1+y2 >= 2
Conclusion -na1-pa+y1+y2 > 0 has been proven.
From nb2 > 0, and (c2-eb2)(nb2) = 0, conclude c2-eb2 = 0
 by merging eb2 nb2-gb2 >= 0 and gb2 > 0
From c2 > 0, and (-na2-nb2+y2)(c2) = 0, conclude na2+nb2-y2 = 0
 by merging c2-eb2 = 0 and eb2 > 0
Conclude nb2-y2 >= 0 by merging eb2 nb2-gb2 >= 0 and -eb2 y2+gb2 = 0
Conclusion nb2-y2 >= 0 has been proven.
-na2 >= 0 by merging nb2-y2 >= 0 and -na2-nb2+y2 = 0
Obtained -na2 = 0, from -na2 >= 0 and na2 >= 0
Substituting na2 = 0
 - In assumption na2+nb2-y2 = 0, to get nb2-y2 = 0
 - In compound assumption c2-c3 ea-c4 >= 0, na2 = 0,
     to get c2-c3 ea-c4 >= 0, 0 = 0
 - In compound assumption ea na1+ea na2+ea pa-ga1-ga2+ta >= 0, c3 = 0,
     to get ea na1+ea pa-ga1-ga2+ta >= 0, c3 = 0
 - In compound assumption na1+na2-pa >= 0, c4 = 0,
     to get na1-pa >= 0, c4 = 0
Eliminated 0 = 0, to get c2-c3 ea-c4 >= 0
Using theorem z0 to add assumption(s):
   -ea na1-ea pa+ga1+ga2 > 0
Conclude ta > 0 by merging -ea na1-ea pa+ga1+ga2 > 0
                                 and ea na1+ea pa-ga1-ga2+ta >= 0
Conclusion ta > 0 has been proven.
Proof finished, all conclusions proven.

Inferences     = 1222486
Unifications   = 2280861
Time (msec)    = 1624833
Speed (LIPS)   =    752
```

```
Runtime stack = 21 % (7168 words)
Copy stack    =  7 % (2313 words)
Trail stack   =  1 % (514 words)

yes

?   pf(d6);
Starting assumptions:
   ta > 0   y1 >= 2   nb1-y1 > -1
From ta > 0, and (1-c3)(ta) = 0, conclude c3 = 1
Substituting c3 = 1
 - In compound assumption c1-c3 ea-c4 >= 0, na1 = 0,
     to get c1-c4-ea >= 0, na1 = 0
 - In compound assumption c2-c3 ea-c4 >= 0, na2 = 0,
     to get c2-c4-ea >= 0, na2 = 0
 - In compound assumption -c3 ea+c4+c5 >= 0, pa = 0,
     to get c4+c5-ea >= 0, pa = 0
 - In compound assumption ea na1+ea na2+ea pa-ga1-ga2+ta >= 0, c3 = 0,
     to get ea na1+ea na2+ea pa-ga1-ga2+ta >= 0, 0 = 1
Eliminated 0 = 1, to get ea na1+ea na2+ea pa-ga1-ga2+ta = 0
Obtained to use for theorem: nb1 > 1 by merging nb1-y1 > -1 and y1 >= 2
Using theorem b7f to add assumption(s):
   -ea >= 0
Contradiction -ea >= 0 and ea > 0

Inferences    =  69118
Unifications  = 166494
Time (msec)   =  96350
Speed (LIPS)  =    717

Runtime stack =  9 % (3246 words)
Copy stack    =  5 % (1776 words)
Trail stack   =  1 % (380 words)

no

?   pf(d7);
Starting assumptions:
   ta = 0   nb1-y1 > -1   y1 >= 2
Using theorem d1 to add assumption(s):
   na2+nb2-y2 = 0
Using theorem d4 to add assumption(s):
   na1+nb1-y1 = 0
Using theorem z4 to add assumption(s):
   na1+na2+pa-y1-y2 >= 0
Substituting ta = 0
 - In compound assumption -c3 >= -1, ta = 0,
     to get -c3 >= -1, 0 = 0
 - In compound assumption ea na1+ea na2+ea pa-ga1-ga2+ta >= 0, c3 = 0,
     to get ea na1+ea na2+ea pa-ga1-ga2 >= 0, c3 = 0
Eliminated 0 = 0, to get -c3 >= -1
Conclude na1+na2-y1-y2 >= -1 by merging na1+na2+pa-y1-y2 >= 0 and -pa >= -1
Conclusion na1+na2-y1-y2 >= -1 has been proven.
Conclude -na1 > -1 by merging na1+nb1-y1 = 0 and nb1-y1 > -1
Conclusion -na1 > -1 has been proven.
Conclude na2-y1-y2 > -2 by merging -na1 > -1 and na1+na2-y1-y2 >= -1
Conclusion na2-y1-y2 > -2 has been proven.
Conclude na2-y2 > 0 by merging na2-y1-y2 > -2 and y1 >= 2
Conclusion na2-y2 > 0 has been proven.
Conclude -nb2 > 0 by merging na2-y2 > 0 and na2+nb2-y2 = 0
Contradiction -nb2 > 0 and nb2 >= 0

Inferences    = 202041
Unifications  = 392102
Time (msec)   = 276700
Speed (LIPS)  =    730

Runtime stack = 14 % (4780 words)
Copy stack    =  5 % (1836 words)
Trail stack   =  1 % (377 words)

no

?   pf(d8a);
Starting assumptions:
   y1 >= 2   nb1-y1 > -1
From nb1 > 1, and (c1-c6 eb1-c7)(nb1) = 0, conclude c1-c6 eb1-c7 = 0
 by merging nb1-y1 > -1 and y1 >= 2
From nb1-pb1 > 0, and (nb1-pb1)(c7) = 0, conclude c7 = 0
 by merging nb1 > 1 and -pb1 >= -1
Substituting c7 = 0
 - In assumption c1-c6 eb1-c7 = 0, to get c1-c6 eb1 = 0
 - In compound assumption -c6 eb1+c7+c8 >= 0, pb1 = 0,
     to get -c6 eb1+c8 >= 0, pb1 = 0
Obtained to use for theorem: y1+y2 > 2 by merging y1 >= 2 and y2 > 0
Using theorem d5a to add assumption(s):
   ta > 0
Using theorem d6 to add assumption(s):
   -nb1+y1 >= 1
Contradiction -nb1+y1 >= 1 and nb1-y1 > -1

Inferences    = 228933
Unifications  = 428175
Time (msec)   = 305050
Speed (LIPS)  =    750

Runtime stack =  8 % (2799 words)
Copy stack    =  5 % (1784 words)
Trail stack   =  1 % (371 words)

no

?   pf(d8b);
Starting assumptions:
   y1 >= 2   nb1-y1 > -1   ta = 0
Substituting ta = 0
 - In compound assumption -c3 >= -1, ta = 0,
     to get -c3 >= -1, 0 = 0
 - In compound assumption ea na1+ea na2+ea pa-ga1-ga2+ta >= 0, c3 = 0,
     to get ea na1+ea na2+ea pa-ga1-ga2 >= 0, c3 = 0
Eliminated 0 = 0, to get -c3 >= -1
From nb1 > 1, and (c1-c6 eb1-c7)(nb1) = 0, conclude c1-c6 eb1-c7 = 0
 by merging nb1-y1 > -1 and y1 >= 2
From nb1-pb1 > 0, and (nb1-pb1)(c7) = 0, conclude c7 = 0
 by merging nb1 > 1 and -pb1 >= -1
Substituting c7 = 0
 - In assumption c1-c6 eb1-c7 = 0, to get c1-c6 eb1 = 0
 - In compound assumption -c6 eb1+c7+c8 >= 0, pb1 = 0,
     to get -c6 eb1+c8 >= 0, pb1 = 0
Obtained to use for theorem: y1+y2 > 2 by merging y1 >= 2 and y2 > 0
Using theorem d5b to add assumption(s):
   -eb2 nb2+gb2 > 0
Using theorem d7 to add assumption(s):
   -nb1+y1 >= 1
Contradiction -nb1+y1 >= 1 and nb1-y1 > -1

Inferences    = 265906
Unifications  = 498953
Time (msec)   = 362216
Speed (LIPS)  =    734
```

```
Runtime stack = 11 % (3741 words)
Copy stack    =  6 % (2026 words)
Trail stack   =  1 % (449 words)

no
```

```
?   pf(e1);
Starting assumptions:
   -nb1+y1 > 1   y1 >= 2   -ea+eb1 > 0
Using theorem d4 to add assumption(s):
   na1+nb1-y1 = 0
Obtained to use for theorem: na1 > 1 by merging na1+nb1-y1 = 0 and -nb1+y1 > 1
From na1 > 1, and (c1-c3 ea-c4)(na1) = 0, conclude c1-c3 ea-c4 = 0
Obtained to use for theorem: na1+na2 > 1 by merging na1 > 1 and na2 >= 0
From na1+na2-pa > 0, and (na1+na2-pa)(c4) = 0, conclude c4 = 0
 by merging na1+na2 > 1 and -pa >= -1
Substituting c4 = 0
 - In assumption c1-c3 ea-c4 = 0, to get c1-c3 ea = 0
 - In compound assumption c2-c3 ea-c4 >= 0, na2 = 0,
     to get c2-c3 ea >= 0, na2 = 0
 - In compound assumption -c3 ea+c4+c5 >= 0, pa = 0,
     to get -c3 ea+c5 >= 0, pa = 0
Obtained to use for theorem: y1+y2 > 2 by merging y1 >= 2 and y2 > 0
Obtained to use for theorem: -nb1-pb1+y1 > 0 by merging -nb1+y1 > 1
                                        and -pb1 >= -1
Using theorem z1 to add assumption(s):
   -eb1 nb1-eb1 pb1+gb1 > 0
Obtained to use for theorem: sb1 > 0 by merging -eb1 nb1-eb1 pb1+gb1 > 0
                                        and eb1 nb1+eb1 pb1-gb1+sb1 >= 0
Using theorem c6d to add assumption(s):
   ea-eb1 >= 0
Contradiction ea-eb1 >= 0 and -ea+eb1 > 0

Inferences    = 875801
Unifications  = 1653413
Time (msec)   = 1182083
Speed (LIPS)  =    740

Runtime stack = 15 % (4903 words)
Copy stack    =  6 % (2028 words)
Trail stack   =  1 % (417 words)

no

?   pf(e2_1);
Starting assumptions:
   y1 >= 2
Using theorem d4 to add assumption(s):
   na1+nb1-y1 = 0
Using theorem d8 to add assumption(s):
   -nb1+y1 >= 1
From na1 >= 1, and (c1-c3 ea-c4)(na1) = 0, conclude c1-c3 ea-c4 = 0
 by merging -nb1+y1 >= 1 and na1+nb1-y1 = 0
Conclude na1+na2 >= 1 by merging na1 >= 1 and na2 >= 0
Conclusion na1+na2 >= 1 has been proven.
Proof finished, all conclusions proven.

Inferences    =  45490
Unifications  =  78811
Time (msec)   =  57200
Speed (LIPS)  =    795

Runtime stack =  5 % (1798 words)
Copy stack    =  4 % (1430 words)
Trail stack   =  0 % (246 words)

yes

?   pf(e2_2);
Starting assumptions:
   y1 >= 2   na2 > 0   na1+na2 > 1
   -ea+eb2 > 0
```

```
From na2 > 0, and (c2-c3 ea-c4)(na2) = 0, conclude c2-c3 ea-c4 = 0
From na1+na2-pa > 0, and (na1+na2-pa)(c4) = 0, conclude c4 = 0
 by merging na1+na2 > 1 and -pa >= -1
Substituting c4 = 0
 - In assumption c2-c3 ea-c4 = 0, to get c2-c3 ea = 0
 - In compound assumption c1-c3 ea-c4 >= 0, na1 = 0,
     to get c1-c3 ea >= 0, na1 = 0
 - In compound assumption -c3 ea+c4+c5 >= 0, pa = 0,
     to get -c3 ea+c5 >= 0, pa = 0
Obtained to use for theorem: y1+y2 > 2 by merging y1 >= 2 and y2 > 0
Using theorem c7b to add assumption(s):
   ea-eb2 >= 0
Contradiction ea-eb2 >= 0 and -ea+eb2 > 0

Inferences    = 341361
Unifications  = 645638
Time (msec)   = 458066
Speed (LIPS)  =    745

Runtime stack =  9 % (3031 words)
Copy stack    =  5 % (1752 words)
Trail stack   =  1 % (357 words)

no

?   pf(e2_3);
Starting assumptions:
   na1+na2 = 1   na1 >= 1
From na1 >= 1, and (c1-c3 ea-c4)(na1) = 0, conclude c1-c3 ea-c4 = 0
-na2 >= 0 by merging na1 >= 1 and na1+na2 = 1
Obtained -na2 = 0, from -na2 >= 0 and na2 >= 0
Conclusion na2 = 0 has been proven.
Proof finished, all conclusions proven.

Inferences    =  36949
Unifications  =  70102
Time (msec)   =  47016
Speed (LIPS)  =    785

Runtime stack =  4 % (1310 words)
Copy stack    =  4 % (1326 words)
Trail stack   =  0 % (210 words)

yes

?   pf(e3_1);
Starting assumptions:
   ea-eb2 > 0   nb2 > 0   ta > 0
Using theorem b8b to add assumption(s):
   -ea+eb2 >= 0
Contradiction -ea+eb2 >= 0 and ea-eb2 > 0

Inferences    =   1058
Unifications  =   1745
Time (msec)   =   2000
Speed (LIPS)  =    529

Runtime stack =  1 % (651 words)
Copy stack    =  3 % (1258 words)
Trail stack   =  0 % (180 words)

no

?   pf(e3_2);
Starting assumptions:
   -ea+eb1 > 0   y1 >= 2   ta = 0
   nb2 > 0
Using theorem d1 to add assumption(s):
   na2+nb2-y2 = 0
Using theorem e1 to add assumption(s):
   na1 = 1   -nb1+y1 = 1
Using theorem z4 to add assumption(s):
   na1+na2+pa-y1-y2 >= 0
Substituting na1 = 1
 - In assumption na1+na2+pa-y1-y2 >= 0, to get na2+pa-y1-y2 >= -1
 - In compound assumption c1-c3 ea-c4 >= 0, na1 = 0,
     to get c1-c3 ea-c4 >= 0, 0 = 1
 - In compound assumption -na1-nb1+y1 >= 0, c1 = 0,
     to get -nb1+y1 >= 1, c1 = 0
 - In compound assumption ea na1+ea na2+ea pa-ga1-ga2+ta >= 0, c3 = 0,
     to get ea+ea na2+ea pa-ga1-ga2+ta >= 0, c3 = 0
 - In compound assumption na1+na2-pa >= 0, c4 = 0,
     to get na2-pa >= -1, c4 = 0
Eliminated 0 = 1, to get c1-c3 ea-c4 = 0
Substituting ta = 0
 - In compound assumption ea+ea na2+ea pa-ga1-ga2+ta >= 0, c3 = 0,
     to get ea+ea na2+ea pa-ga1-ga2 >= 0, c3 = 0
 - In compound assumption -c3 >= -1, ta = 0,
     to get -c3 >= -1, 0 = 0
Eliminated 0 = 0, to get -c3 >= -1
From nb2 > 0, and (c2-eb2)(nb2) = 0, conclude c2-eb2 = 0
From c2 > 0, and (-na2-nb2+y2)(c2) = 0, conclude na2+nb2-y2 = 0
 by merging c2-eb2 = 0 and eb2 > 0
Conclude na2-y1-y2 >= -2 by merging na2+pa-y1-y2 >= -1 and -pa >= -1
Conclusion na2-y1-y2 >= -2 has been proven.
Conclude na2-y2 >= 0 by merging na2-y1-y2 >= -2 and y1 >= 2
Conclusion na2-y2 >= 0 has been proven.
Conclude -nb2 >= 0 by merging na2-y2 >= 0 and na2+nb2-y2 = 0
Contradiction -nb2 >= 0 and nb2 > 0

Inferences    = 132003
Unifications  = 251062
Time (msec)   = 187950
Speed (LIPS)  =    702

Runtime stack = 19 % (6510 words)
Copy stack    =  6 % (2194 words)
Trail stack   =  1 % (503 words)

no

?   pf(e4_1);
Starting assumptions:
   nb1 > 0   y1 >= 2   ta > 0
   ea-2 eb1 > 0
Using theorem b7b to add assumption(s):
   -ea+2 eb1 >= 0
Contradiction -ea+2 eb1 >= 0 and ea-2 eb1 > 0

Inferences    =   1224
Unifications  =   2064
Time (msec)   =   2783
Speed (LIPS)  =    439

Runtime stack =  2 % (806 words)
Copy stack    =  4 % (1306 words)
Trail stack   =  0 % (212 words)

no

?   pf(e4_2);
Starting assumptions:
```

```
   y2 > 1   y1 >= 2   ta = 0
   -ea+eb2 > 0
Using theorem d4 to add assumption(s):
   na1+nb1-y1 = 0
Using theorem e2 to add assumption(s):
   na2 = 0
Using theorem z4 to add assumption(s):
   na1+na2+pa-y1-y2 >= 0
Substituting na2 = 0
 - In assumption na1+na2+pa-y1-y2 >= 0, to get na1+pa-y1-y2 >= 0
 - In compound assumption c2-c3 ea-c4 >= 0, na2 = 0,
     to get c2-c3 ea-c4 >= 0, 0 = 0
 - In compound assumption -na2-nb2+y2 >= 0, c2 = 0,
     to get -nb2+y2 >= 0, c2 = 0
 - In compound assumption ea na1+ea na2+ea pa-ga1-ga2+ta >= 0, c3 = 0,
     to get ea na1+ea pa-ga1-ga2+ta >= 0, c3 = 0
 - In compound assumption na1+na2-pa >= 0, c4 = 0,
     to get na1-pa >= 0, c4 = 0
Eliminated 0 = 0, to get c2-c3 ea-c4 >= 0
Substituting ta = 0
 - In compound assumption ea na1+ea pa-ga1-ga2+ta >= 0, c3 = 0,
     to get ea na1+ea pa-ga1-ga2 >= 0, c3 = 0
 - In compound assumption -c3 >= -1, ta = 0,
     to get -c3 >= -1, 0 = 0
Eliminated 0 = 0, to get -c3 >= -1
Conclude na1-y1-y2 >= -1 by merging na1+pa-y1-y2 >= 0 and -pa >= -1
Conclusion na1-y1-y2 >= -1 has been proven.
Conclude na1-y1 > 0 by merging na1-y1-y2 >= -1 and y2 > 1
Conclusion na1-y1 > 0 has been proven.
Conclude -nb1 > 0 by merging na1-y1 > 0 and na1+nb1-y1 = 0
Contradiction -nb1 > 0 and nb1 >= 0

Inferences    =  90791
Unifications  = 170040
Time (msec)   = 128950
Speed (LIPS)  =    704

Runtime stack = 16 % (5369 words)
Copy stack    =  6 % (2010 words)
Trail stack   =  1 % (436 words)

no

?   pf(e4_3);
Starting assumptions:
   nb1 = 0
Using theorem d4 to add assumption(s):
   na1+nb1-y1 = 0
Substituting nb1 = 0
 - In assumption na1+nb1-y1 = 0, to get na1-y1 = 0
 - In compound assumption c1-c6 eb1-c7 >= 0, nb1 = 0,
     to get c1-c6 eb1-c7 >= 0, 0 = 0
 - In compound assumption -na1-nb1+y1 >= 0, c1 = 0,
     to get -na1+y1 >= 0, c1 = 0
 - In compound assumption eb1 nb1+eb1 pb1-gb1+sb1 >= 0, c6 = 0,
     to get eb1 pb1-gb1+sb1 >= 0, c6 = 0
 - In compound assumption nb1-pb1 >= 0, c7 = 0,
     to get -pb1 >= 0, c7 = 0
Eliminated 0 = 0, to get c1-c6 eb1-c7 >= 0
Obtained -pb1 = 0, from -pb1 >= 0 and pb1 >= 0
Conclusion na1-y1 = 0 has been proven.
Proof finished, all conclusions proven.

Inferences    =   3648
Unifications  =   6493
Time (msec)   =   6366
Speed (LIPS)  =    572

Runtime stack =  5 % (1928 words)
Copy stack    =  4 % (1516 words)
Trail stack   =  0 % (267 words)

yes

?   pf(e5_1);
Starting assumptions:
   na1-y1 > -1   y1 >= 2   -ea+2 eb1 > 0
Using theorem d4 to add assumption(s):
   na1+nb1-y1 = 0
Obtained to use for theorem: -nb1 > -1 by merging na1+nb1-y1 = 0 and na1-y1 > -1
Using theorem z3 to add assumption(s):
   sb1 > 0
From sb1 > 0, and (1-c6)(sb1) = 0, conclude c6 = 1
Substituting c6 = 1
 - In compound assumption c1-c6 eb1-c7 >= 0, nb1 = 0,
     to get c1-c7-eb1 >= 0, nb1 = 0
 - In compound assumption -c6 eb1+c7+c8 >= 0, pb1 = 0,
     to get c7+c8-eb1 >= 0, pb1 = 0
 - In compound assumption eb1 nb1+eb1 pb1-gb1+sb1 >= 0, c6 = 0,
     to get eb1 nb1+eb1 pb1-gb1+sb1 >= 0, 0 = 1
Eliminated 0 = 1, to get eb1 nb1+eb1 pb1-gb1+sb1 = 0
From -pb1 > -1, and (1-pb1)(c8) = 0, conclude c8 = 0
 by merging -nb1 > -1 and nb1-pb1 >= 0
Substituting c8 = 0
 - In compound assumption c7+c8-eb1 >= 0, pb1 = 0,
     to get c7-eb1 >= 0, pb1 = 0
Obtained to use for theorem: na1 > 1 by merging y1 >= 2 and na1-y1 > -1
From na1 > 1, and (c1-c3 ea-c4)(na1) = 0, conclude c1-c3 ea-c4 = 0
Obtained to use for theorem: na1+na2 > 1 by merging na1 > 1 and na2 >= 0
From na1+na2-pa > 0, and (na1+na2-pa)(c4) = 0, conclude c4 = 0
 by merging na1+na2 > 1 and -pa >= -1
Substituting c4 = 0
 - In assumption c1-c3 ea-c4 = 0, to get c1-c3 ea = 0
 - In compound assumption c2-c3 ea-c4 >= 0, na2 = 0,
     to get c2-c3 ea >= 0, na2 = 0
 - In compound assumption -c3 ea+c4+c5 >= 0, pa = 0,
     to get -c3 ea+c5 >= 0, pa = 0
Obtained to use for theorem: y1+y2 > 2 by merging y1 >= 2 and y2 > 0
Using theorem c6c to add assumption(s):
   ea-2 eb1 >= 0
Contradiction ea-2 eb1 >= 0 and -ea+2 eb1 > 0

Inferences    = 997983
Unifications  = 1877772
Time (msec)   = 1334383
Speed (LIPS)  =    747

Runtime stack = 20 % (6766 words)
Copy stack    =  8 % (2644 words)
Trail stack   =  2 % (665 words)

no

?   pf(e5_2);
Starting assumptions:
   -na1+y1 > 1   ea-eb1 > 0   y1 >= 2
   -ea+eb2 > 0
Using theorem d4 to add assumption(s):
   na1+nb1-y1 = 0
Using theorem e2 to add assumption(s):
   na2 = 0
Substituting na2 = 0
```

```
 - In compound assumption c2-c3 ea-c4 >= 0, na2 = 0,
     to get c2-c3 ea-c4 >= 0, 0 = 0
 - In compound assumption -na2-nb2+y2 >= 0, c2 = 0,
     to get -nb2+y2 >= 0, c2 = 0
 - In compound assumption ea na1+ea na2+ea pa-ga1-ga2+ta >= 0, c3 = 0,
     to get ea na1+ea pa-ga1-ga2+ta >= 0, c3 = 0
 - In compound assumption na1+na2-pa >= 0, c4 = 0,
     to get na1-pa >= 0, c4 = 0
Eliminated 0 = 0, to get c2-c3 ea-c4 >= 0
Obtained to use for theorem: nb1 > 1 by merging na1+nb1-y1 = 0 and -na1+y1 > 1
From nb1 > 1, and (c1-c6 eb1-c7)(nb1) = 0, conclude c1-c6 eb1-c7 = 0
From nb1-pb1 > 0, and (nb1-pb1)(c7) = 0, conclude c7 = 0
 by merging nb1 > 1 and -pb1 >= -1
Substituting c7 = 0
 - In assumption c1-c6 eb1-c7 = 0, to get c1-c6 eb1 = 0
 - In compound assumption -c6 eb1+c7+c8 >= 0, pb1 = 0,
     to get -c6 eb1+c8 >= 0, pb1 = 0
Obtained to use for theorem: -na1+y1+y2 > 1 by merging -na1+y1 > 1 and y2 > 0
Using theorem z2 to add assumption(s):
   ta > 0
Using theorem b7d to add assumption(s):
   -ea+eb1 >= 0
Contradiction -ea+eb1 >= 0 and ea-eb1 > 0

Inferences    = 519763
Unifications  = 984072
Time (msec)   = 698766
Speed (LIPS)  =    743

Runtime stack = 15 % (4957 words)
Copy stack    =  6 % (2148 words)
Trail stack   =  1 % (481 words)

no

?   pf(e5_3);
Starting assumptions:
   -na1+y1 = 1
Using theorem d4 to add assumption(s):
   na1+nb1-y1 = 0
Conclude nb1 = 1 by merging na1+nb1-y1 = 0 and -na1+y1 = 1
Conclusion nb1 = 1 has been proven.
Proof finished, all conclusions proven.

Inferences    =  22252
Unifications  =  38854
Time (msec)   =  29400
Speed (LIPS)  =    756

Runtime stack =  3 % (1086 words)
Copy stack    =  3 % (1298 words)
Trail stack   =  0 % (199 words)

yes

?   pf(e6_1);
Starting assumptions:
   -na1-na2+y1+y2 > 1
Using theorem d1 to add assumption(s):
   na2+nb2-y2 = 0
Using theorem d4 to add assumption(s):
   na1+nb1-y1 = 0
Using theorem z2 to add assumption(s):
   ta > 0
From ta > 0, and (1-c3)(ta) = 0, conclude c3 = 1
Substituting c3 = 1
 - In compound assumption c1-c3 ea-c4 >= 0, na1 = 0,
     to get c1-c4-ea >= 0, na1 = 0
 - In compound assumption c2-c3 ea-c4 >= 0, na2 = 0,
     to get c2-c4-ea >= 0, na2 = 0
 - In compound assumption -c3 ea+c4+c5 >= 0, pa = 0,
     to get c4+c5-ea >= 0, pa = 0
 - In compound assumption ea na1+ea na2+ea pa-ga1-ga2+ta >= 0, c3 = 0,
     to get ea na1+ea na2+ea pa-ga1-ga2+ta >= 0, 0 = 1
Eliminated 0 = 1, to get ea na1+ea na2+ea pa-ga1-ga2+ta = 0
Conclude -na2+nb1+y2 > 1 by merging na1+nb1-y1 = 0 and -na1-na2+y1+y2 > 1
Conclusion -na2+nb1+y2 > 1 has been proven.
Conclude nb1+nb2 > 1 by merging -na2+nb1+y2 > 1 and na2+nb2-y2 = 0
Conclusion nb1+nb2 > 1 has been proven.
Proof finished, all conclusions proven.

Inferences    = 100582
Unifications  = 226200
Time (msec)   = 139100
Speed (LIPS)  =    723

Runtime stack = 11 % (3725 words)
Copy stack    =  5 % (1770 words)
Trail stack   =  1 % (370 words)

yes

?   pf(e6_2);
Starting assumptions:
   nb2 > 0   ta > 0   ea-eb2 > 0
Using theorem b8b to add assumption(s):
   -ea+eb2 >= 0
Contradiction -ea+eb2 >= 0 and ea-eb2 > 0

Inferences    =   1052
Unifications  =   1733
Time (msec)   =   2083
Speed (LIPS)  =    504

Runtime stack =  1 % (651 words)
Copy stack    =  3 % (1264 words)
Trail stack   =  0 % (193 words)

no

?   pf(e6_3);
Starting assumptions:
   nb2 = 0   nb1+nb2 > 1   ea-eb1 > 0
   y1 >= 2   ta > 0
Substituting nb2 = 0
 - In assumption nb1+nb2 > 1, to get nb1 > 1
 - In compound assumption c2-eb2 >= 0, nb2 = 0,
     to get c2-eb2 >= 0, 0 = 0
 - In compound assumption -na2-nb2+y2 >= 0, c2 = 0,
     to get -na2+y2 >= 0, c2 = 0
Eliminated 0 = 0, to get c2-eb2 >= 0
Using theorem b7d to add assumption(s):
   -ea+eb1 >= 0
Contradiction -ea+eb1 >= 0 and ea-eb1 > 0

Inferences    =   3445
Unifications  =   6281
Time (msec)   =   5100
Speed (LIPS)  =    675

Runtime stack =  5 % (1774 words)
Copy stack    =  4 % (1580 words)
```

```
Trail stack   =  0 % (302 words)

no

?   pf(e6_4);
Starting assumptions:
   -na1-na2+y1+y2 = 1
Using theorem d1 to add assumption(s):
   na2+nb2-y2 = 0
Using theorem d4 to add assumption(s):
   na1+nb1-y1 = 0
Conclude -na2+nb1+y2 = 1 by merging na1+nb1-y1 = 0 and -na1-na2+y1+y2 = 1
Conclusion -na2+nb1+y2 = 1 has been proven.
Conclude nb1+nb2 = 1 by merging -na2+nb1+y2 = 1 and na2+nb2-y2 = 0
Conclusion nb1+nb2 = 1 has been proven.
Proof finished, all conclusions proven.

Inferences    =  48622
Unifications  =  84756
Time (msec)   =  59933
Speed (LIPS)  =    811

Runtime stack =  5 % (1770 words)
Copy stack    =  4 % (1398 words)
Trail stack   =  0 % (226 words)

yes

?   pf(e7_1);
Starting assumptions:
   na1 = 0   y1 >= 2   nb1+nb2 = 1
Using theorem d4 to add assumption(s):
   na1+nb1-y1 = 0
Substituting na1 = 0
 - In assumption na1+nb1-y1 = 0, to get nb1-y1 = 0
 - In compound assumption c1-c3 ea-c4 >= 0, na1 = 0,
     to get c1-c3 ea-c4 >= 0, 0 = 0
 - In compound assumption -na1-nb1+y1 >= 0, c1 = 0,
     to get -nb1+y1 >= 0, c1 = 0
 - In compound assumption ea na1+ea na2+ea pa-ga1-ga2+ta >= 0, c3 = 0,
     to get ea na2+ea pa-ga1-ga2+ta >= 0, c3 = 0
 - In compound assumption na1+na2-pa >= 0, c4 = 0,
     to get na2-pa >= 0, c4 = 0
Eliminated 0 = 0, to get c1-c3 ea-c4 >= 0
From nb1 >= 2, and (c1-c6 eb1-c7)(nb1) = 0, conclude c1-c6 eb1-c7 = 0
 by merging nb1-y1 = 0 and y1 >= 2
-nb2 >= 1 by merging nb1 >= 2 and nb1+nb2 = 1
Contradiction -nb2 >= 1 and nb2 >= 0

Inferences    =  48409
Unifications  =  87561
Time (msec)   =  62466
Speed (LIPS)  =    774

Runtime stack =  9 % (3200 words)
Copy stack    =  5 % (1634 words)
Trail stack   =  0 % (312 words)

no

?   pf(e7_2);
Starting assumptions:
   na1 > 0   -nb1 > -1   2 eb1-eb2 > 0
   nb1+nb2 = 1   y1 >= 2
Using theorem z3 to add assumption(s):
   sb1 > 0
Using theorem c3 to add assumption(s):
   c1-2 eb1 >= 0
From sb1 > 0, and (1-c6)(sb1) = 0, conclude c6 = 1
Substituting c6 = 1
 - In compound assumption c1-c6 eb1-c7 >= 0, nb1 = 0,
     to get c1-c7-eb1 >= 0, nb1 = 0
 - In compound assumption -c6 eb1+c7+c8 >= 0, pb1 = 0,
     to get c7+c8-eb1 >= 0, pb1 = 0
 - In compound assumption eb1 nb1+eb1 pb1-gb1+sb1 >= 0, c6 = 0,
     to get eb1 nb1+eb1 pb1-gb1+sb1 >= 0, 0 = 1
Eliminated 0 = 1, to get eb1 nb1+eb1 pb1-gb1+sb1 = 0
From na1 > 0, and (c1-c3 ea-c4)(na1) = 0, conclude c1-c3 ea-c4 = 0
Conclude -c1+c2 >= 0 by merging c1-c3 ea-c4 = 0 and c2-c3 ea-c4 >= 0
Conclusion -c1+c2 >= 0 has been proven.
Obtained to use for theorem: nb2 > 0 by merging nb1+nb2 = 1 and -nb1 > -1
Using theorem b5 to add assumption(s):
   c2-eb2 = 0
From nb2 > 0, and (c2-eb2)(nb2) = 0, conclude c2-eb2 = 0
From c2 > 0, and (-na2-nb2+y2)(c2) = 0, conclude na2+nb2-y2 = 0
 by merging c2-eb2 = 0 and eb2 > 0
Conclude -c1+eb2 >= 0 by merging c2-eb2 = 0 and -c1+c2 >= 0
Conclusion -c1+eb2 >= 0 has been proven.
Conclude -2 eb1+eb2 >= 0 by merging -c1+eb2 >= 0 and c1-2 eb1 >= 0
Contradiction -2 eb1+eb2 >= 0 and 2 eb1-eb2 > 0

Inferences    = 191863
Unifications  = 371391
Time (msec)   = 247616
Speed (LIPS)  =    774

Runtime stack = 20 % (6733 words)
Copy stack    =  6 % (2230 words)
Trail stack   =  1 % (522 words)

no

?   pf(e7_3);
Starting assumptions:
   nb1 = 1   nb1+nb2 = 1
Substituting nb1 = 1
 - In assumption nb1+nb2 = 1, to get nb2 = 0
 - In compound assumption c1-c6 eb1-c7 >= 0, nb1 = 0,
     to get c1-c6 eb1-c7 >= 0, 0 = 1
 - In compound assumption -na1-nb1+y1 >= 0, c1 = 0,
     to get -na1+y1 >= 1, c1 = 0
 - In compound assumption eb1 nb1+eb1 pb1-gb1+sb1 >= 0, c6 = 0,
     to get eb1+eb1 pb1-gb1+sb1 >= 0, c6 = 0
 - In compound assumption nb1-pb1 >= 0, c7 = 0,
     to get -pb1 >= -1, c7 = 0
Eliminated 0 = 1, to get c1-c6 eb1-c7 = 0
Conclusion nb2 = 0 has been proven.
Proof finished, all conclusions proven.

Inferences    =   3100
Unifications  =   5511
Time (msec)   =   4800
Speed (LIPS)  =    645

Runtime stack =  5 % (1709 words)
Copy stack    =  4 % (1488 words)
Trail stack   =  0 % (258 words)

yes

?   pf(e8_1);
Starting assumptions:
```

```
   na2 = 0
Using theorem d1 to add assumption(s):
   na2+nb2-y2 = 0
Substituting na2 = 0
 - In assumption na2+nb2-y2 = 0, to get nb2-y2 = 0
 - In compound assumption c2-c3 ea-c4 >= 0, na2 = 0,
     to get c2-c3 ea-c4 >= 0, 0 = 0
 - In compound assumption -na2-nb2+y2 >= 0, c2 = 0,
     to get -nb2+y2 >= 0, c2 = 0
 - In compound assumption ea na1+ea na2+ea pa-ga1-ga2+ta >= 0, c3 = 0,
     to get ea na1+ea pa-ga1-ga2+ta >= 0, c3 = 0
 - In compound assumption na1+na2-pa >= 0, c4 = 0,
     to get na1-pa >= 0, c4 = 0
Eliminated 0 = 0, to get c2-c3 ea-c4 >= 0
Conclusion nb2-y2 = 0 has been proven.
Proof finished, all conclusions proven.

Inferences    =   3440
Unifications  =   6128
Time (msec)   =   5300
Speed (LIPS)  =    649

Runtime stack =  5 % (1802 words)
Copy stack    =  4 % (1478 words)
Trail stack   =  0 % (257 words)

yes

?   pf(e8_2);
Starting assumptions:
   na2 > 0   nb1 > 0
From nb1 > 0, and (c1-c6 eb1-c7)(nb1) = 0, conclude c1-c6 eb1-c7 = 0
From na2 > 0, and (c2-c3 ea-c4)(na2) = 0, conclude c2-c3 ea-c4 = 0
Conclude c1-c2 >= 0 by merging c2-c3 ea-c4 = 0 and c1-c3 ea-c4 >= 0
Conclusion c1-c2 >= 0 has been proven.
Conclude c1-eb2 >= 0 by merging c1-c2 >= 0 and c2-eb2 >= 0
Conclusion c1-eb2 >= 0 has been proven.
Conclude c6 eb1+c7-eb2 >= 0 by merging c1-eb2 >= 0 and c1-c6 eb1-c7 = 0
Conclusion c6 eb1+c7-eb2 >= 0 has been proven.
Proof finished, all conclusions proven.

Inferences    =  87650
Unifications  = 158520
Time (msec)   = 108150
Speed (LIPS)  =    810

Runtime stack = 10 % (3417 words)
Copy stack    =  4 % (1520 words)
Trail stack   =  0 % (270 words)

yes

?   pf(e8_3);
Starting assumptions:
   c7 = 0   c6 eb1+c7-eb2 >= 0   -2 eb1+eb2 > 0
Substituting c7 = 0
 - In assumption c6 eb1+c7-eb2 >= 0, to get c6 eb1-eb2 >= 0
 - In compound assumption c1-c6 eb1-c7 >= 0, nb1 = 0,
     to get c1-c6 eb1 >= 0, nb1 = 0
 - In compound assumption -c6 eb1+c7+c8 >= 0, pb1 = 0,
     to get -c6 eb1+c8 >= 0, pb1 = 0
 - In compound assumption nb1-pb1 >= 0, c7 = 0,
     to get nb1-pb1 >= 0, 0 = 0
Eliminated 0 = 0, to get nb1-pb1 >= 0
c6 > 2 by merging c6 eb1-eb2 >= 0 and -2 eb1+eb2 > 0
Contradiction c6 > 2 and -c6 >= -1

Inferences    =  27305
Unifications  =  50027
Time (msec)   =  35033
Speed (LIPS)  =    779

Runtime stack =  6 % (2255 words)
Copy stack    =  4 % (1552 words)
Trail stack   =  0 % (281 words)

no

?   pf(e8_4);
Starting assumptions:
   nb1 > 0   nb1-pb1 = 0   -2 eb1+eb2 > 0
   c6 eb1+c7-eb2 >= 0
From nb1 > 0, and (c1-c6 eb1-c7)(nb1) = 0, conclude c1-c6 eb1-c7 = 0
From pb1 > 0, and (-c6 eb1+c7+c8)(pb1) = 0, conclude c6 eb1-c7-c8 = 0
 by merging nb1-pb1 = 0 and nb1 > 0
Conclude c6 eb1-c7 >= 0 by merging c6 eb1-c7-c8 = 0 and c8 >= 0
Conclusion c6 eb1-c7 >= 0 has been proven.
Conclude 2 c6 eb1-eb2 >= 0 by merging c6 eb1-c7 >= 0 and c6 eb1+c7-eb2 >= 0
Conclusion 2 c6 eb1-eb2 >= 0 has been proven.
Conclude 2 eb1-eb2 >= 0 by merging -c6 >= -1 and 2 c6 eb1-eb2 >= 0
Contradiction 2 eb1-eb2 >= 0 and -2 eb1+eb2 > 0

Inferences    = 123425
Unifications  = 233236
Time (msec)   = 157200
Speed (LIPS)  =    785

Runtime stack = 10 % (3289 words)
Copy stack    =  4 % (1526 words)
Trail stack   =  0 % (278 words)

no
```

```
?   pf(z0);
Starting assumptions:
   -na1-na2-pa+y1+y2 > 0   ea na1+ea na2+ea pa-ga1-ga2 >= 0
na1+na2+pa-y1-y2 >= 0 by merging ea na1+ea na2+ea pa-ga1-ga2 >= 0
                                 and ea y1+ea y2-ga1-ga2 = 0
Contradiction na1+na2+pa-y1-y2 >= 0 and -na1-na2-pa+y1+y2 > 0

Inferences    =  31802
Unifications  =  65859
Time (msec)   =  58750
Speed (LIPS)  =    541

Runtime stack =  6 % (2248 words)
Copy stack    =  4 % (1416 words)
Trail stack   =  0 % (256 words)

no

?   pf(z1);
Starting assumptions:
   -nb1-pb1+y1 > 0   eb1 nb1+eb1 pb1-gb1 >= 0
nb1+pb1-y1 >= 0 by merging eb1 nb1+eb1 pb1-gb1 >= 0 and eb1 y1-gb1 = 0
Contradiction nb1+pb1-y1 >= 0 and -nb1-pb1+y1 > 0

Inferences    =  26731
Unifications  =  49192
Time (msec)   =  33450
Speed (LIPS)  =    799

Runtime stack =  5 % (1658 words)
Copy stack    =  4 % (1330 words)
Trail stack   =  0 % (212 words)

no

?   pf(z2);
Starting assumptions:
   -na1-na2+y1+y2 > 1
Obtained to use for theorem: -na1-na2-pa+y1+y2 > 0
                                 by merging -na1-na2+y1+y2 > 1 and -pa >= -1
Using theorem z0 to add assumption(s):
   -ea na1-ea na2-ea pa+ga1+ga2 > 0
Conclude ta > 0 by merging -ea na1-ea na2-ea pa+ga1+ga2 > 0
                                 and ea na1+ea na2+ea pa-ga1-ga2+ta >= 0
Conclusion ta > 0 has been proven.
Proof finished, all conclusions proven.

Inferences    =  51902
Unifications  =  96257
Time (msec)   =  67983
Speed (LIPS)  =    763

Runtime stack =  6 % (2198 words)
Copy stack    =  4 % (1364 words)
Trail stack   =  0 % (213 words)

yes

?   pf(z3);
Starting assumptions:
   -nb1 > -1   y1 >= 2
Conclude -nb1+y1 > 1 by merging y1 >= 2 and -nb1 > -1
Conclusion -nb1+y1 > 1 has been proven.
Obtained to use for theorem: -nb1-pb1+y1 > 0 by merging -nb1+y1 > 1
                                                and -pb1 >= -1
Using theorem z1 to add assumption(s):
   -eb1 nb1-eb1 pb1+gb1 > 0
Conclude sb1 > 0 by merging -eb1 nb1-eb1 pb1+gb1 > 0
                                                and eb1 nb1+eb1 pb1-gb1+sb1 >= 0
Conclusion sb1 > 0 has been proven.
Proof finished, all conclusions proven.

Inferences    = 128447
Unifications  = 231502
Time (msec)   = 177450
Speed (LIPS)  =    723

Runtime stack =  5 % (1931 words)
Copy stack    =  4 % (1466 words)
Trail stack   =  0 % (248 words)

yes

?   pf(z4);
Starting assumptions:
   ta = 0
Substituting ta = 0
 - In compound assumption -c3 >= -1, ta = 0,
     to get -c3 >= -1, 0 = 0
 - In compound assumption ea na1+ea na2+ea pa-ga1-ga2+ta >= 0, c3 = 0,
     to get ea na1+ea na2+ea pa-ga1-ga2 >= 0, c3 = 0
Eliminated 0 = 0, to get -c3 >= -1
From c2 > 0, and (-na2-nb2+y2)(c2) = 0, conclude na2+nb2-y2 = 0
 by merging eb2 > 0 and c2-eb2 >= 0
Conclude na1+na2+pa-y1-y2 >= 0 by merging ea y1+ea y2-ga1-ga2 = 0
                                 and ea na1+ea na2+ea pa-ga1-ga2 >= 0
Conclusion na1+na2+pa-y1-y2 >= 0 has been proven.
Proof finished, all conclusions proven.

Inferences    =  74374
Unifications  = 152220
Time (msec)   = 121400
Speed (LIPS)  =    612

Runtime stack = 11 % (3599 words)
Copy stack    =  5 % (1784 words)
Trail stack   =  1 % (370 words)

yes
```

13. Appendix 3: The Code

```
%
% proof( A T C ) - starting from list of assumptions A and the hypotheses
%   given by hyp and kt_cond, construct a proof of anything assumed
%   to be interesting, or of anything in conclusion C.
%   Can make use of theorems in list T.
%
proof( Assump Thms [] ) <-
    startup( Assump Thms Ass Comp_ass Thm_list )
    proof_step( 0 Ass Comp_ass Thm_list [] [] )
    ;
proof( Assump Thms [C | Cs] ) <-
    startup( Assump Thms Ass Comp_ass Thm_list )
    proof_step( 1 Ass Comp_ass Thm_list [C | Cs] [C | Cs] )
    ;
%
% startup( A T ) - do some initialization.
%
startup( Assump Thms A CA T ) <-
    write_start( Assump )
    all_of( Hyp X hyp( X ))
    all_of( Comp_assump X kt_cond( X ))
    cleanup( Assump [] Hyp Comp_assump A CA )
    make_thm_list( Thms T )
    ;
%
% write_start( A ) - Print out the starting assumptions.
%
write_start( [] ) <-
    write( "No starting assumptions." )
    nl
    ;
write_start( [A | As] ) <-
    write( "Starting assumptions: " )
    nl
    print_ass( 0 [A | As] )
    ;
%
% proof_step( N A CA T C OC ) - perform a "useful" step in a proof with ass.
%   A and compound assumptions CA. Try either of the following six steps:
%     1 - use an existing theorem in list T
%     2 - simple substitution with equality
%     3 - simplify by dividing an inequality by a variable or integer
%     4 - eliminate one of the branches from a compound assumption
%     5 - try to substitute a bigger integer for a variable in a product
%     6 - add two inequalities to create a new useful one
%
proof_step( 1 _ _ _ [] [] ) <-
    write( "Proof finished, all conclusions proven." )
    nl
    ;
proof_step( 1 Ass Comp_ass Thm Conc Original ) <-
    conc_elim( Conc Original Ass Comp_ass New_conc New_original )
    cut
    proof_step( 1 Ass Comp_ass Thm New_conc New_original )
    ;
proof_step( N Ass Comp_ass Thm Conc Original ) <-
    use_thm( Thm Conc_ass New_thm )
    cut
    cleanup( Conc_ass [] Ass Comp_ass New_ass New_comp_ass )
    cut
    proof_step( N New_ass New_comp_ass New_thm Conc Original )
    ;
proof_step( N Ass Comp_ass Thm Conc Original ) <-
    thm_ass_elim( Thm Ass Comp_ass New_thm )
    cut
    proof_step( N Ass Comp_ass New_thm Conc Original )
    ;
proof_step( N Ass Comp_ass Thm Conc Original ) <-
    try_subst( Ass Comp_ass Thm Conc Temp_a1 Temp_a2 Temp_ca1 Temp_ca2
               New_thm New_conc )
    cut
    cleanup( Temp_a1 Temp_ca1 Temp_a2 Temp_ca2 New_ass New_comp_ass )
    cut
    proof_step( N New_ass New_comp_ass New_thm New_conc Original )
    ;
proof_step( N Ass Comp_ass Thm Conc Original ) <-
    simp( Ass Comp_ass Temp_a1 Temp_a2 Temp_ca1 Temp_ca2 )
    cut
    cleanup( Temp_a1 Temp_ca1 Temp_a2 Temp_ca2 New_ass New_comp_ass )
    cut
    proof_step( N New_ass New_comp_ass Thm Conc Original )
    ;
proof_step( N Ass Comp_ass Thm Conc Original ) <-
    elim( Ass Comp_ass Temp_ass Temp_comp_ass )
    cut
    cleanup( [Temp_ass] [] Ass Temp_comp_ass New_ass New_comp_ass )
    cut
    proof_step( N New_ass New_comp_ass Thm Conc Original )
    ;
proof_step( N Ass Comp_ass Thm Conc Original ) <-
    prod_merge( Ass Comp_ass Temp_ass Temp_comp_ass Conc Thm )
    cut
    cleanup( Temp_ass [] Ass Temp_comp_ass New_ass New_comp_ass )
    cut
    proof_step( N New_ass New_comp_ass Thm Conc Original )
    ;
proof_step( N Ass Comp_ass Thm Conc Original ) <-
    mod( Mod )
    prove( Mod
       merge( Ass Comp_ass Ass Comp_ass Temp_ass Temp_comp_ass Conc Thm ))
    cut
    cleanup( Temp_ass [] Ass Temp_comp_ass New_ass New_comp_ass )
    cut
    proof_step( N New_ass New_comp_ass Thm Conc Original )
    ;
proof_step( N Ass Comp_ass Thm Conc _ ) <-
    write( "Proof finished, no further steps can be performed." )
    nl
    print_list( Ass "assumptions" )
    print_list( Comp_ass "compound assumptions" )
    print_list( Thm "theorems" )
    print_list( Conc "conclusions" )
    ;
%
% print_list( L S ) - print out remaining list of type S.
%
print_list( [] S ) <-
    write( "No " ) write( S ) write( " left over." )
    nl
    ;
print_list( [Hd | Tl] "compound assumptions" ) <-
    cut
    write( "The compound assumptions remaining are:" )
    nl
    print_comp_ass( [Hd | Tl] )
    ;
print_list( [Hd | Tl] "theorems" ) <-
    cut
    write( "The list of theorems not used is:" ) write( [Hd | Tl] )
    nl
    ;
print_list( [Hd | Tl] S ) <-
```

```
    write( "The " ) write( S ) write( " remaining are:" )
    nl
    print_ass( 0 [Hd | Tl] )
    ;
%
% make_thm_list( T L ) - let L be the list with assumptions and conclusions
%    from all the theorems in list T.
%
%
make_thm_list( [] [] );
make_thm_list( [Thm | Tl] [[Thm Ass Conc] | New_tl] ) <-
    theorem( Thm Ass Conc )
    make_thm_list( Tl New_tl )
    ;
%
% total_merge and fast_merge are two predicates which allow you to load
% the total merge version and the faster version of the merge algorithm
% respectively.
%
total_merge <-
    retract( mod( _ ) )
    assert( mod( total ) [] )
    ;
fast_merge <-
    retract( mod( _ ) )
    assert( mod( fast ) [] )
    ;
%
% pre_proof( A T C ) - proof the theorem with assumptions A, conclusion
%     C and where theorems T can be used. Both the assumptions and
%     conclusions are in the simple format.
%
%
pre_proof( Ass Thm Conc ) <-
    convert_list( Ass New_ass )
    convert_list( Conc New_conc )
    proof( New_ass Thm New_conc )
    ;
%
% convert_list( L1 L2 ) - convert each inequality in list L1 into
%     canonical form and put it in list L2.
%
%
convert_list( [] [] );
convert_list( [Hd | Tl] [New_hd | New_tl] ) <-
    convert( Hd New_hd )
    convert_list( Tl New_tl )
    ;
%
% convert( S C ) - convert string S in simple syntax to expression in
%     our canonical form C.
%
%
convert( String New_e ) <-
    name( String Char_list )
    reverse( Char_list Rev )
    skip_bl( Rev Rev_wo_bl )
    reverse( Rev_wo_bl New_cl )
    parse( New_cl Type Int Terms )
    cut
    check_canon( [Type Int | Terms] New_e )
    ;
%
% parse( L T I TL ) - parse list of characters L, get the inequality
%   type T, the integer I and the term-list TL out of it.
%
parse( ['>' '=' | Tl] ge Int [] ) <-
    skip_bl( Tl New_tl )
    own_int_to_list( Int New_tl )
    ;
parse( ['>' | Tl] g Int [] ) <-
    skip_bl( Tl New_tl )
    own_int_to_list( Int New_tl )
    ;
parse( ['=' | Tl] e Int [] ) <-
    skip_bl( Tl New_tl )
    own_int_to_list( Int New_tl )
    ;
parse( [' ' | List] Type Int Term_l ) <-
    skip_bl( List New_list )
    parse( New_list Type Int Term_l )
    ;
parse( List Type Int New_term_l ) <-
    get_term( 0 List Tl Term )
    parse( Tl Type Int Term_l )
    add_term1( Term Term_l New_term_l )
    ;
%
% get_term( N L1 L2 T ) - get term T from beginning of list L1 and let
%   the remainder be L2, N indicates if the sign has already been recognized.
%
%
get_term( 1 ['+' | Tl] Tl [1] );
get_term( 0 ['+' | Tl] New_tl Term ) <-
    get_term( 1 Tl New_tl Term )
    ;
get_term( 1 ['-' | Tl] ['-' | Tl] [1] );
get_term( 0 ['-' | Tl] New_tl [Neg_int | Term] ) <-
    get_term( 1 Tl New_tl [Int | Term] )
    mul( -1 Int Neg_int )
    ;
get_term( 1 ['>' | Tl] ['>' | Tl] [1] );
get_term( 1 ['=' | Tl] ['=' | Tl] [1] );
get_term( L [' ' | Tl] New_tl Term ) <-
    skip_bl( Tl Temp_tl )
    get_term( L Temp_tl New_tl Term )
    ;
get_term( 0 [Hd | Tl] New_tl Term ) <-
    get_term( 1 [Hd | Tl] New_tl Term )
    ;
get_term( 1 [Hd | Tl] New_tl Term ) <-
    le( Hd '9' ) ge( Hd '0' )
    get_atom( [Hd | Tl] Temp_tl L )
    int_to_list( Atom L )
    get_term( 1 Temp_tl New_tl Temp_term )
    add_atom( Atom Temp_term Term )
    ;
get_term( 1 [Hd | Tl] New_tl Term ) <-
    get_atom( [Hd | Tl] Temp_tl L )
    name( Atom L )
    get_term( 1 Temp_tl New_tl Temp_term )
    add_atom( Atom Temp_term Term )
    ;
%
% get_atom( L1 L2 A ) - get atom from list L1 leaving list L2.
%
%
get_atom( [Hd | Tl] [Hd | Tl] [] ) <-
    is_delim( Hd )
    ;
get_atom( [Hd | Tl] New_tl [Hd | Rest] ) <-
    get_atom( Tl New_tl Rest )
    ;
is_delim( ' ' );
is_delim( '+' );
is_delim( '-' );
is_delim( '=' );
is_delim( '>' );
```

```
%
% add_atom( A T NT ) - add atom A to term T giving new term NT.
%
add_atom( I1 [I2 | Rest] [I3 | Rest] ) <-
    is_int( I1 )
    mul( I1 I2 I3 )
    ;
add_atom( At [I2 | Rest] [I2 | New_rest] ) <-
    is_int( I2 )
    add_atom( At Rest New_rest )
    ;
add_atom( At1 [At2 | Rest] [At1 At2 | Rest] ) <-
    lt( At1 At2 )
    ;
add_atom( At1 [At2 | Rest] [At2 | New_rest] ) <-
    add_atom( At1 Rest New_rest )
    ;
add_atom( At [] [At] );
%
% skip_bl( L1 L2 ) - L2 is list L1 without any blanks at the beginning.
%
skip_bl( [' ' | Tl] New_tl ) <-
    skip_bl( Tl New_tl )
    ;
skip_bl( [Hd | Tl] [Hd | Tl] ) <-
    ne( Hd ' ' )
    ;
%
% own_int_to_list( I L ) - own version of int_to_list which handles
%    negative integers.
%
own_int_to_list( Neg_int ['-' | Tl] ) <-
    int_to_list( Int Tl )
    mul( -1 Int Neg_int )
    ;
own_int_to_list( Int L ) <-
    int_to_list( Int L )
    ;
%
% conc_elim( C OC A CA NC NOC ) - eliminate one conclusion from list C, given
%      assumptions A and compound assumptions CA, to produce new list NC.
%
conc_elim( [Hd | Tl] [OHd | OTl] Ass Comp_ass Tl OTl ) <-
    contain_ass( Hd Ass Comp_ass )
    write( "Conclusion " )
    print_exp( OHd )
    write( " has been proven." )
    nl
    ;
conc_elim( [Hd | Tl] [OHd | OTl] Ass Comp_ass [Hd | New_tl] [OHd | NOTl] ) <-
    conc_elim( Tl OTl Ass Comp_ass New_tl NOTl )
    ;
%
% can_elim_conc( C E ) - given expression E we can eliminate a conclusion
%    from the list C.
%
can_elim_conc( [Hd | Tl] E ) <-
    contain_ass( Hd [E] [] )
    ;
can_elim_conc( [Hd | Tl] E ) <-
    can_elim_conc( Tl E )
    ;
%
% thm_ass_elim( T A CA NT ) - eliminate an assumption from a theorem in list
%      T, given list of assumptions A and compound assumptions CA, to
%      produce new list NT.
%
thm_ass_elim( [[T [Hd_a | Tl_a] C] | Tl] Ass Comp_ass [[T Tl_a C] | Tl] ) <-
    contain_ass( Hd_a Ass Comp_ass )
    ;
thm_ass_elim( [[T [A | Tl_a] C] | Tl] Ass CA [[T [A | New_tl] C] | Tl] ) <-
    thm_ass_elim( [[T Tl_a C] | Tl] Ass CA [[T New_tl C] | Tl] )
    ;
thm_ass_elim( [Hd | Tl] Ass Comp_ass [Hd | New_tl] ) <-
    thm_ass_elim( Tl Ass Comp_ass New_tl )
    ;
%
% contain_ass( A AL CAL ) - succeeds if assumption A is part of assumptions
%    in list of assumptions AL and compound assumptions CAL.
%
contain_ass( Ineq Ass Comp_ass ) <-
    useless( Ineq )
    ;
contain_ass( [Type1 Int1 | Rest] Ass Comp_ass ) <-
    member( [Type2 Int2 | Rest] Ass )
    contain_ass( Type1 Int1 Type2 Int2 )
    ;
contain_ass( [Type Int1 | Rest] Ass Comp_ass ) <-
    ne( Type e )
    negate( Rest Neg_rest )
    member( [e Int2 | Neg_rest] Ass )
    plus( Int2 Neg_int2 0 )
    neg_contain_ass( Type Int1 Neg_int2 )
    ;
contain_ass( [Type1 Int1 | Rest] Ass Comp_ass ) <-
    member( [[Type2 Int2 | Rest] _] Comp_ass )
    contain_ass( Type1 Int1 Type2 Int2 )
    ;
contain_ass( [Type Int1 | Rest] Ass Comp_ass ) <-
    ne( Type e )
    negate( Rest Neg_rest )
    member( [[e Int2 | Neg_rest] _] Comp_ass )
    plus( Int2 Neg_int2 0 )
    neg_contain_ass( Type Int1 Neg_int2 )
    ;
contain_ass( T Int T Int );
contain_ass( g Int1 _ Int2 ) <-
    lt( Int1 Int2 )
    ;
contain_ass( ge Int1 _ Int2 ) <-
    le( Int1 Int2 )
    ;
neg_contain_ass( _ Int1 Int2 ) <-
    lt( Int1 Int2 )
    ;
neg_contain_ass( ge Int Int );
%
% cleanup( A CA OA OCA NA NCA ) - add assumptions A and CA to old assumptions
%    OA and OCA to give new ones NA and NCA. Check for contradictions and
%    useless assumptions.
%
cleanup( [] [] Old_ass Old_ca Old_ass Old_ca );
cleanup( [Hd | Tl] C Ass CA NA NCA ) <-
    contrad( 0 Hd Ass CA )
    cut
    fail
    ;
cleanup( [Hd | Tl] C Ass CA NA NCA ) <-
    useless( Hd Ass CA Temp_ass )
    cut
    cleanup( Tl C Temp_ass CA NA NCA )
    ;
```

```
cleanup( [Hd | Tl] C Ass CA NA NCA ) <-
    cleanup( Tl C [Hd | Ass] CA NA NCA )
    ;
cleanup( [] [[Hd1 Hd2] | Tl] A CA NA NCA ) <-
    contrad( 0 Hd1 A CA )
    cut
    fail
    ;
cleanup( [] [[[T1 I1] [T2 I2 | Rest]] | Tl] A CA NA NCA ) <-
    ne( I1 0 )
    check_canon( [e I2 | Rest] New_hd )
    write( "Eliminated " )
    print_exp( [T1 I1] )
    write( ", to get " )
    print_exp( New_hd )
    nl
    cut
    cleanup( [New_hd] Tl A CA NA NCA )
    ;
cleanup( [] [[[T 0] Rest] | Tl] A CA NA NCA ) <-
    cut
    cleanup( [] Tl A CA NA NCA )
    ;
cleanup( [] [[[T1 I1 | Rest] [T2 I2]] | Tl] A CA NA NCA ) <-
    ne( I2 0 )
    check_canon( [e I1 | Rest] New_hd )
    write( "Eliminated " )
    print_exp( [T2 I2] )
    write( ", to get " )
    print_exp( New_hd )
    nl
    cut
    cleanup( [New_hd] Tl A CA NA NCA )
    ;
cleanup( [] [[Hd [e 0]] | Tl] A CA NA NCA ) <-
    write( "Eliminated 0 = 0, to get " )
    print_exp( Hd )
    nl
    cut
    cleanup( [Hd] Tl A CA NA NCA )
    ;
cleanup( [] [[[T1 I1 | Rest] Sec] | Tl] A CA NA NCA ) <-
    ne( Rest [] )
    find_new( T1 I1 Rest A CA New_ass )
    cut
    cleanup( [] [[[T1 I1 | Rest] Sec] | Tl] New_ass CA NA NCA )
    ;
cleanup( [] [Comp_ass | Tl] A CA NA NCA ) <-
    cleanup( [] Tl A [Comp_ass | CA] NA NCA )
    ;
%
% contrad( L I A CA ) - test if (in)equality I is a contradiction, given list
%   of assumptions A and list of compound assumptions CA, if L is 0 print
%   out what the contradiction is.
%
contrad( Lev [g Int] _ _ ) <-
    cut
    ge( Int 0 )
    print_contrad( Lev [g Int] )
    ;
contrad( Lev [e Int] _ _ ) <-
    cut
    ne( Int 0 )
    print_contrad( Lev [e Int] )
    ;
contrad( Lev [ge Int] _ _ ) <-
    cut
    gt( Int 0 )
    print_contrad( Lev [ge Int] )
    ;
contrad( Lev [T1 I1 | Rest] Ass _ ) <-
    member( [T2 I2 | Rest] Ass )
    contr( T1 I1 T2 I2 )
    print_contrad( Lev [T1 I1 | Rest] [T2 I2 | Rest] )
    ;
contrad( Lev [T1 I1 | Rest] _ Comp_ass ) <-
    member( [[T2 I2 | Rest] _] Comp_ass )
    contr( T1 I1 T2 I2 )
    print_contrad( Lev [T1 I1 | Rest] [T2 I2 | Rest] )
    ;
contrad( Lev [T1 I1 | Rest] Ass _ ) <-
    negate( Rest Neg_rest )
    member( [T2 I2 | Neg_rest] Ass )
    plus( I2 Neg_i2 0 )
    neg_contr( T1 I1 T2 Neg_i2 )
    print_contrad( Lev [T1 I1 | Rest] [T2 I2 | Neg_rest] )
    ;
contrad( Lev [T1 I1 | Rest] _ Comp_ass ) <-
    negate( Rest Neg_rest )
    member( [[T2 I2 | Neg_rest] _] Comp_ass )
    plus( I2 Neg_i2 0 )
    neg_contr( T1 I1 T2 Neg_i2 )
    print_contrad( Lev [T1 I1 | Rest] [T2 I2 | Neg_rest] )
    ;
contr( e Int1 e Int2 ) <- ne( Int1 Int2 );
contr( e Int1 g Int2 ) <- le( Int1 Int2 );
contr( e Int1 ge Int2 ) <- lt( Int1 Int2 );
contr( g Int1 e Int2 ) <- ge( Int1 Int2 );
contr( ge Int1 e Int2 ) <- gt( Int1 Int2 );
neg_contr( e Int1 e Int2 );
neg_contr( g Int1 T Int2 ) <- ge( Int1 Int2 );
neg_contr( T Int1 g Int2 ) <- ge( Int1 Int2 );
neg_contr( T1 Int1 T2 Int2 ) <- gt( Int1 Int2 );
%
% useless( I A CA NA ) - test if (in)equality I is useless given list of
%   assumptions A and producing new list NA.
% useless( I ) - (in)equality I is completely useless.
% NOTE: I assume a check for a contradiction has been done already!!
%
useless( Ineq Ass Comp_ass Ass ) <- useless( Ineq );
useless( [T1 I1 | Rest] Ass Comp_ass [New_hd | Temp_ass] ) <-
    ne( Rest [] )
    delete( [T2 I2 | Rest] Ass Temp_ass )
    useless( T1 I1 T2 I2 T3 I3 )
    check_canon( [T3 I3 | Rest] New_hd )
    ;
useless( [T1 I1 | Rest] Ass Comp_ass New_ass ) <-
    ne( Rest [] )
    member( [[T2 I2 | Rest] _] Comp_ass )
    comp_useless( T1 I1 T2 I2 Rest Ass New_ass )
    ;
useless( [T1 I1 | Rest] Ass Comp_ass [New_hd | Temp_ass] ) <-
    ne( Rest [] )
    negate( Rest Neg_rest )
    delete( [T2 I2 | Neg_rest] Ass Temp_ass )
    plus( I2 Neg_i2 0 )
    neg_useless( T1 I1 T2 Neg_i2 T3 I3 Rest )
    check_canon( [T3 I3 | Rest] New_hd )
    ;
useless( [T1 I1 | Rest] Ass Comp_ass New_ass ) <-
    ne( Rest [] )
    negate( Rest Neg_rest )
```

```
    member( [[T2 I2 | Neg_rest] | _] Comp_ass )
    plus( I2 Neg_i2 0 )
    neg_comp_useless( T1 I1 T2 Neg_i2 Rest Ass New_ass )
    ;
useless( T Int T Int T Int );
useless( g Int ge Int g Int );
useless( ge Int g Int g Int );
useless( e Int ge Int e Int );
useless( ge Int e Int e Int );
useless( T1 Int1 T2 Int2 T1 Int1 ) <- gt( Int1 Int2 );
useless( T1 Int1 T2 Int2 T2 Int2 ) <- lt( Int1 Int2 );
neg_useless( e Int1 T Int2 e Int1 _ );
neg_useless( T Int1 e Int2 e Int2 _ );
neg_useless( ge Int ge Int e Int Rest ) <-
    write( "Obtained " ) print_exp( [e Int | Rest] )
    write( ", from " ) print_exp( [ge Int | Rest] )
    negate( Rest NR ) sum( Int NI 0 )
    write( " and " ) print_exp( [ge NI | NR] ) nl
    ;
useless( [e 0] );
useless( [g Int] ) <- lt( Int 0 );
useless( [ge Int] ) <- le( Int 0 );
%
% comp_useless( T1 I1 T2 I2 R A NA ) - check if the new inequality is worth
%     adding after it has been found also in one of the compound assumptions.
%
comp_useless( T1 I1 T2 I2 Rest Ass Ass ) <-
    useless( T1 I1 T2 I2 T2 I2 )        % stronger one in compound
    ;
comp_useless( T1 I1 T2 I2 Rest Ass [New_hd | Ass] ) <-
    useless( T1 I1 T2 I2 T3 I3 )        % new expression found.
    ne( pair( T1 I1 ) pair( T3 I3 ))
    ne( pair( T2 I2 ) pair( T3 I3 ))
    check_canon( [T3 I3 | Rest] New_hd )
    ;
neg_comp_useless( T1 I1 T2 I2 Rest Ass Ass ) <-
    neg_useless( T1 I1 T2 I2 T2 I2 _ )      % stronger one in compound
    ;
neg_comp_useless( T1 I1 T2 I2 Rest Ass [New_hd | Ass] ) <-
    neg_useless( T1 I1 T2 I2 T3 I3 Rest )   % new expression found.
    ne( pair( T1 I1 ) pair( T3 I3 ))
    ne( pair( T2 I2 ) pair( T3 I3 ))
    check_canon( [T3 I3 | Rest] New_hd )
    ;
%
% find_new( T I R A CA NA ) - try if we can form new list of assumptions NA
%   from inequality of type T and integer part I with rest R as first part
%   of one of the compound expressions.
%
find_new( T1 I1 Rest Ass Comp_ass [New_hd | Temp_ass] ) <-
    delete( [T2 I2 | Rest] Ass Temp_ass )
    useless( T1 I1 T2 I2 T3 I3 )
    ne( pair( T1 I1 ) pair( T3 I3 ))
    ne( pair( T2 I2 ) pair( T3 I3 ))
    check_canon( [T3 I3 | Rest] New_hd )
    ;
find_new( T1 I1 Rest Ass Comp_ass [New_in | Ass] ) <-
    member( [[T2 I2 | Rest] _] Comp_ass )
    useless( T1 I1 T2 I2 T3 I3 )
    ne( pair( T1 I1 ) pair( T3 I3 ))
    ne( pair( T2 I2 ) pair( T3 I3 ))
    check_canon( [T3 I3 | Rest] New_in )
    ;
find_new( T1 I1 Rest Ass Comp_ass [New_hd | Temp_ass] ) <-
    negate( Rest Neg_rest )
    delete( [T2 I2 | Neg_rest] Ass Temp_ass )
    plus( I2 Neg_i2 0 )
    neg_useless( T1 I1 T2 Neg_i2 T3 I3 Rest )
    ne( pair( T1 I1 ) pair( T3 I3 ))
    ne( pair( T2 Neg_i2 ) pair( T3 I3 ))
    check_canon( [T3 I3 | Rest] New_hd )
    ;
find_new( T1 I1 Rest Ass Comp_ass [New_in | Ass] ) <-
    negate( Rest Neg_rest )
    member( [[T2 I2 | Neg_rest] | _] Comp_ass )
    plus( I2 Neg_i2 0 )
    neg_useless( T1 I1 T2 Neg_i2 T3 I3 Rest )
    ne( pair( T1 I1 ) pair( T3 I3 ))
    ne( pair( T2 Neg_i2 ) pair( T3 I3 ))
    check_canon( [T3 I3 | Rest] New_in )
    ;
%
% use_thm( TL NA NT ) - try to use theorem from list TL, if the list of
%   assumptions is empty, adding list of assumptions NA and giving new list
%   NT.
%
use_thm( Thm_l Concl New_thm_l ) <-
    delete( [Name [] Concl] Thm_l New_thm_l )
    write( "Using theorem " )
    write( Name )
    write( " to add assumption(s): " )
    nl
    print_ass( 0 Concl )
    ;
%
% try_subst( A CA T C OA OCA NA NCA NT NC ) - try to make substitutions
%   for assumptions A and CA, split these lists in A1 and A2 and CA1 and
%   CA2 where the 1-lists contain those assumptions with substitutions and
%   the 2-lists contain the unchanged assumptions. Change list of theorems
%   T into new list NT and list of conclusions C into new list NC.
%
try_subst( Old_ass Old_comp_ass Thm Conc NA1 NA2 NCA1 NCA2 Nthm Nconc ) <-
    delete( Hd Old_ass Temp_ass )
    find_subst( Hd Var Int )
    write( "Substituting " )
    write( Var )
    write( " = " )
    write( Int )
    nl
    subst_ass( Int Var Temp_ass NA1 NA2 )
    subst_comp_ass( Int Var Old_comp_ass NCA1 NCA2 )
    subst_thm( Int Var Thm Nthm )
    subst_conc( Int Var Conc Nconc )
    ;
try_subst( Old_ass Old_comp_ass Thm Conc NA1 NA2 NCA1 NCA2 Nthm Nconc ) <-
    delete( [H1 H2] Old_comp_ass Temp_comp_ass )
    find_subst( H1 Var Int )
    write( "Substituting " )
    write( Var )
    write( " = " )
    write( Int )
    nl
    subst_ass( Int Var Old_ass NA1 NA2 )
    subst_comp_ass( Int Var Temp_comp_ass NCA1 NCA2 )
    subst_thm( Int Var Thm Nthm )
    subst_list( Int Var Conc Nconc )
    ;
%
% find_subst( A AL V I ) - find a substitution for assumption A with list of
%   assumptions AL and the integer to be substituteed I for variable V.
%
find_subst( [e Int1 [Int2 Var]] Var Val ) <-
```

```
    div( Int1 Int2 Val )
    ;
%
% find_new_subst( E A CA ) - try if we can apply a substitution given the
%   expression E and list of assumptions A and compound assumptions CA.
%
find_new_subst( [e Int1 [Int2 Var]] _ _ );
find_new_subst( [ge Int1 [Int2 Var]] Ass CA ) <-
    sum( Int1 NI1 0 )
    sum( Int2 NI2 0 )
    or( member( [ge NI1 [NI2 Var]] Ass )
        member( [[ge NI1 [NI2 Var]] _] CA ))
    ;
%
% subst_ass( X Y L L1 L2 ) - substitute integer X for Y in list L, to
%   give lists L1 (those inequalities containing Y) and list L2.
%
subst_ass( X Y [] [] [] );
subst_ass( X Y [Hd | Tl] [Sub_hd | L1] L2 ) <-
    try_subst_exp( X Y Hd Temp_hd )
    check_canon( Temp_hd Sub_hd )
    print_subst( "assumption" Hd Sub_hd )
    subst_ass( X Y Tl L1 L2 )
    ;
subst_ass( X Y [Hd | Tl] L1 [Hd | L2] ) <-
    subst_ass( X Y Tl L1 L2 )
    ;
%
% subst_comp_ass( X Y L L1 L2 ) - substitute integer X for Y in list L, to
%   give lists L1 (those inequalities containing Y) and list L2.
%
subst_comp_ass( X Y [] [] [] );
subst_comp_ass( X Y [[Hd1 Hd2] | Tl] [[Sh1 Sh2] | L1] L2 ) <-
    try_subst_exp( X Y Hd1 Temp_hd1 )
    check_canon( Temp_hd1 Sh1 )
    subst_comp_ass1( X Y Hd2 Sh2 )
    print_subst( "compound assumption" [Hd1 Hd2] [Sh1 Sh2] )
    subst_comp_ass( X Y Tl L1 L2 )
    ;
subst_comp_ass( X Y [[Hd1 Hd2] | Tl] [[Hd1 Sh2] | L1] L2 ) <-
    try_subst_exp( X Y Hd2 Temp_hd2 )
    check_canon( Temp_hd2 Sh2 )
    print_subst( "compound assumption" [Hd1 Hd2] [Hd1 Sh2] )
    subst_comp_ass( X Y Tl L1 L2 )
    ;
subst_comp_ass( X Y [[Hd1 Hd2] | Tl] L1 [[Hd1 Hd2] | L2] ) <-
    subst_comp_ass( X Y Tl L1 L2 )
    ;
subst_comp_ass1( X Y Exp Sub_exp ) <-
    try_subst_exp( X Y Exp Temp_exp )
    check_canon( Temp_exp Sub_exp )
    ;
subst_comp_ass1( X Y Exp Exp );
%
% subst_thm( X Y L NL ) - substitute integer X for Y in list of theorems L,
%   to give lists L1.
%
subst_thm( X Y [] [] );
subst_thm( X Y [[Name Ass Conc] | Tl] [[Name New_ass New_c] | Sub_tl] ) <-
    subst_list( X Y Ass New_ass )
    subst_list( X Y Conc New_c )
    subst_thm( X Y Tl Sub_tl )
    ;
%
% subst_conc( X Y L NL ) - substitute integer X for Y in list of conc. L,
%   to give new list NL.
%
subst_conc( X Y [] [] );
subst_conc( X Y [Hd | Tl] [Sub_hd | Sub_tl] ) <-
    try_subst_exp( X Y Hd Temp_hd )
    check_canon( Temp_hd Sub_hd )
    print_subst( "conclusion" Hd Sub_hd )
    subst_conc( X Y Tl Sub_tl )
    ;
subst_conc( X Y [Hd | Tl] [Hd | Sub_tl] ) <-
    subst_conc( X Y Tl Sub_tl )
    ;
%
% print_subst( S O N ) - print out substitution took place from old
%   inequality O of type S to get new one N.
%
print_subst( S O N ) <-
    ne( S "compound assumption" )
    write( " - In " ) write( S ) put( ' ' )
    print_exp( O )
    write( ", to get " )
    print_exp( N )
    nl
    ;
print_subst( "compound assumption" [O1 O2] [N1 N2] ) <-
    write( " - In compound assumption " )
    print_exp( O1 ) write( ", " ) print_exp( O2 ) put( ',' )
    nl
    write( "    to get " )
    print_exp( N1 ) write( ", " ) print_exp( N2 )
    nl
    ;
%
% subst_list( X Y L NL ) - substitute integer X for Y in list of ineq. L,
%   to give new list NL.
%
subst_list( X Y [] [] );
subst_list( X Y [Hd | Tl] [Sub_hd | Sub_tl] ) <-
    try_subst_exp( X Y Hd Temp_hd )
    check_canon( Temp_hd Sub_hd )
    subst_list( X Y Tl Sub_tl )
    ;
subst_list( X Y [Hd | Tl] [Hd | Sub_tl] ) <-
    subst_list( X Y Tl Sub_tl )
    ;
%
% check_canon( E C ) - put expression E into canonical form C.
%
check_canon( [e 0 [Int | Tl1] | Tl2] [e 0 | Neg_tl] ) <-
    lt( Int 0 )
    cut
    negate( [[Int | Tl1] | Tl2] Neg_tl )
    ;
check_canon( [e Int | Tl] [e | Neg_tl] ) <-
    lt( Int 0 )
    cut
    negate( [Int | Tl] Neg_tl )
    ;
check_canon( Exp Exp );
%
% negate( L1 L2 ) - list L2 contains negated elements of list L1.
%
negate( [] [] );
negate( [[Int | Tl1] | Tl2] [[Neg_int | Tl1] | Neg_tl2] ) <-
    plus( Int Neg_int 0 )
    negate( Tl2 Neg_tl2 )
    ;
```

```
negate( [Int | Tl] [Neg_int | Neg_tl] ) <-
    is_int( Int )
    plus( Int Neg_int 0 )
    negate( Tl Neg_tl )
    ;
%
% try_subst_exp( X Y E1 E2 ) - substitute integer X for variable Y in the
%   expression E1 if Y occurs in E1, to produce expression E2.
%
try_subst_exp( X Y [T I | Tl] New_exp ) <-
    in_prod( Y Tl Term )
    delete( Term Tl New_tl )
    subst_prod( X Y Term New_term )
    add_term( New_term [T I | New_tl] Temp_exp )
    subst_exp( X Y Temp_exp New_exp )
    ;
%
% subst_exp( X Y E1 E2 ) - substitute integer X for variable Y in the
%   expression E1, to produce expression E2.
%
subst_exp( X Y [T I | Tl] New_exp ) <-
    in_prod( Y Tl Term )
    delete( Term Tl New_tl )
    subst_prod( X Y Term New_term )
    add_term( New_term [T I | New_tl] Temp_exp )
    subst_exp( X Y Temp_exp New_exp )
    ;
subst_exp( X Y [T I | Tl] [T I | Tl] ) <-
    not( in_prod( Y Tl _ ))
    ;
%
% in_prod( Y E T ) - variable Y occurs in term T of expression E.
%
in_prod( Y [Hd | Tl] Hd ) <-
    member( Y Hd )
    ;
in_prod( Y [Hd | Tl] Term ) <-
    in_prod( Y Tl Term )
    ;
%
% subst_prod( X Y T NT ) - substitute integer X for variable Y in term
%   T to get new term NT.
%
subst_prod( Int Var [I | Tl] New_term ) <-
    mul( Int I New_I )
    delete( Var Tl New_tl )
    check_one( [New_I | New_tl] New_term )
    ;
%
% add_term( NT E1 E2 ) - add new term NT to expression E1 to give E2.
%
add_term( Term [T I | Expr] [T New_I | Expr] ) <-
    is_int( Term )
    cut
    plus( Term I New_I )
    ;
add_term( Term [T I | Expr] [T I | New_expr] ) <-
    add_term1( Term Expr New_expr )
    ;
add_term1( Term [] [Term] ) <-
    ;
add_term1( [Int | Rest] [[Neg_int | Rest] | Tl] Tl ) <-
    sum( Int Neg_int 0 )
    cut
    ;
add_term1( [I1 | Rest] [[I2 | Rest] | Tl] [[New_int | Rest] | Tl] ) <-
    sum( I1 I2 New_int )
    cut
    ;
add_term1( [I1 | Tl1] [[I2 | Tl2] | Tl3] [[I1 | Tl1] [I2 | Tl2] | Tl3] ) <-
    less_strong( Tl1 Tl2 )
    cut
    ;
add_term1( Term [Hd | Tl] [Hd | New_tl] ) <-
    add_term1( Term Tl New_tl )
    ;
%
% check_one( T1 T2 ) - check if term T1 consists of a single element,
%   if so return the single element, otherwise return T1.
%
check_one( [0 | Tl] 0 )
    cut
    ;
check_one( [Int] Neg_int ) <-
    cut
    plus( Int Neg_int 0 )
    ;
check_one( X X );
%
% less_strong( T1 T2 ) - test if term T1 should come before term T2 or not.
%
less_strong( [] X );
less_strong( [Hd1 | Tl1] [Hd2 | Tl2] ) <-
    lt( Hd1 Hd2 )
    cut
    ;
less_strong( [Hd | Tl1] [Hd | Tl2] ) <-
    less_strong( Tl1 Tl2 )
    ;
less_strong( [Hd1 | Tl1] [Hd2 | Tl2] ) <-
    gt( Hd1 Hd2 )
    cut
    fail
    ;
%
% simp( A CA NA NCA ) - try to eliminate one of the variables from an
%   assumption in lists of simple and compound assumptions A and CA to
%   get new lists NA and NCA.
%
simp( Ass Comp_ass [New_hd] New_ass [] Comp_ass ) <-
    delete( Hd Ass New_ass )
    find_simp( 0 Hd Ass Comp_ass New_hd )
    ;
simp( Ass Comp_ass [] Ass [[New_h1 H2]] New_comp_ass ) <-
    delete( [H1 H2] Comp_ass New_comp_ass )
    find_simp( 0 H1 Ass Comp_ass New_h1 )
    ;
%
% find_simp( N X A CA NX ) - try to eliminate one of the vars from expression
%   X to give new expression NX, given list of assumptions A and list of
%   compound assumptions CA. N is a number indicating if it should be checked
%   that the resluting expression is already in the list (0-no, 1-yes).
%
find_simp( N [Type 0 Hd | Rem] Ass Comp_ass New_ineq ) <-
    ne( Rem [] )
    check_list( Hd Ass Comp_ass Temp_l )
    check_simp( Temp_l Rem Var_list )
    subst_simp( N Var_list [Type 0 Hd | Rem] New_ineq Ass Comp_ass )
    ;
find_simp( N [Type 0 [Int | Rest]] Ass Comp_ass New_ineq ) <-
    ne( Int 1 ) ne( Int -1 ) ne( Rest [] )
    check_list( [Int | Rest] Ass Comp_ass Var_list )
```

```
negate( [Int | Tl] [Neg_int | Neg_tl] ) <-
    is_int( Int )
    plus( Int Neg_int 0 )
    negate( Tl Neg_tl )
    ;
%
% try_subst_exp( X Y E1 E2 ) - substitute integer X for variable Y in the
%   expression E1 if Y occurs in E1, to produce expression E2.
%
try_subst_exp( X Y [T I | Tl] New_exp ) <-
    in_prod( Y Tl Term )
    delete( Term Tl New_tl )
    subst_prod( X Y Term New_term )
    add_term( New_term [T I | New_tl] Temp_exp )
    subst_exp( X Y Temp_exp New_exp )
    ;
%
% subst_exp( X Y E1 E2 ) - substitute integer X for variable Y in the
%   expression E1, to produce expression E2.
%
subst_exp( X Y [T I | Tl] New_exp ) <-
    in_prod( Y Tl Term )
    delete( Term Tl New_tl )
    subst_prod( X Y Term New_term )
    add_term( New_term [T I | New_tl] Temp_exp )
    subst_exp( X Y Temp_exp New_exp )
    ;
subst_exp( X Y [T I | Tl] [T I | Tl] ) <-
    not( in_prod( Y Tl _ ))
    ;
%
% in_prod( Y E T ) - variable Y occurs in term T of expression E.
%
in_prod( Y [Hd | Tl] Hd ) <-
    member( Y Hd )
    ;
in_prod( Y [Hd | Tl] Term ) <-
    in_prod( Y Tl Term )
    ;
%
% subst_prod( X Y T NT ) - substitute integer X for variable Y in term
%   T to get new term NT.
%
subst_prod( Int Var [I | Tl] New_term ) <-
    mul( Int I New_I )
    delete( Var Tl New_tl )
    check_one( [New_I | New_tl] New_term )
    ;
%
% add_term( NT E1 E2 ) - add new term NT to expression E1 to give E2.
%
add_term( Term [T I | Expr] [T New_I | Expr] ) <-
    is_int( Term )
    cut
    plus( Term I New_I )
    ;
add_term( Term [T I | Expr] [T I | New_expr] ) <-
    add_term1( Term Expr New_expr )
    ;
add_term1( Term [] [Term] ) <-
    ;
add_term1( [Int | Rest] [[Neg_int | Rest] | Tl] Tl ) <-
    sum( Int Neg_int 0 )
    cut
    ;
add_term1( [I1 | Rest] [[I2 | Rest] | Tl] [[New_int | Rest] | Tl] ) <-
    sum( I1 I2 New_int )
    cut
    ;
add_term1( [I1 | Tl1] [[I2 | Tl2] | Tl3] [[I1 | Tl1] [I2 | Tl2] | Tl3] ) <-
    less_strong( Tl1 Tl2 )
    cut
    ;
add_term1( Term [Hd | Tl] [Hd | New_tl] ) <-
    add_term1( Term Tl New_tl )
    ;
%
% check_one( T1 T2 ) - check if term T1 consists of a single element,
%   if so return the single element, otherwise return T1.
%
check_one( [0 | Tl] 0 )
    cut
    ;
check_one( [Int] Neg_int ) <-
    cut
    plus( Int Neg_int 0 )
    ;
check_one( X X );
%
% less_strong( T1 T2 ) - test if term T1 should come before term T2 or not.
%
less_strong( [] X );
less_strong( [Hd1 | Tl1] [Hd2 | Tl2] ) <-
    lt( Hd1 Hd2 )
    cut
    ;
less_strong( [Hd | Tl1] [Hd | Tl2] ) <-
    less_strong( Tl1 Tl2 )
    ;
less_strong( [Hd1 | Tl1] [Hd2 | Tl2] ) <-
    gt( Hd1 Hd2 )
    cut
    fail
    ;
%
% simp( A CA NA NCA ) - try to eliminate one of the variables from an
%   assumption in lists of simple and compound assumptions A and CA to
%   get new lists NA and NCA.
%
simp( Ass Comp_ass [New_hd] New_ass [] Comp_ass ) <-
    delete( Hd Ass New_ass )
    find_simp( 0 Hd Ass Comp_ass New_hd )
    ;
simp( Ass Comp_ass [] Ass [[New_h1 H2]] New_comp_ass ) <-
    delete( [H1 H2] Comp_ass New_comp_ass )
    find_simp( 0 H1 Ass Comp_ass New_h1 )
    ;
%
% find_simp( N X A CA NX ) - try to eliminate one of the vars from expression
%   X to give new expression NX, given list of assumptions A and list of
%   compound assumptions CA. N is a number indicating if it should be checked
%   that the resluting expression is already in the list (0-no, 1-yes).
%
find_simp( N [Type 0 Hd | Rem] Ass Comp_ass New_ineq ) <-
    ne( Rem [] )
    check_list( Hd Ass Comp_ass Temp_l )
    check_simp( Temp_l Rem Var_list )
    subst_simp( N Var_list [Type 0 Hd | Rem] New_ineq Ass Comp_ass )
    ;
find_simp( N [Type 0 [Int | Rest]] Ass Comp_ass New_ineq ) <-
    ne( Int 1 ) ne( Int -1 ) ne( Rest [] )
    check_list( [Int | Rest] Ass Comp_ass Var_list )
```

```
    subst_simp( N Var_list [Type 0 [Int | Rest]] New_ineq Ass Comp_ass )
    ;
find_simp( N [Type 0 [Int | Rest]] Ass Comp_ass New_ineq ) <-
    or( eq( Int 1 ) eq( Int -1 ))
    length( Rest L ) ge( L 2 )
    check_list( [Int | Rest] Ass Comp_ass Var_list )
    subst_simp( N Var_list [Type 0 [Int | Rest]] New_ineq Ass Comp_ass )
    ;
find_simp( N [Type Int | Rest] Ass Comp_ass New_ineq ) <-
    ne( Int 0 ) ne( Int 1 ) ne( Int -1 )
    ne( Rest [] )
    check_simp( [Int] Rest [Int] )
    subst_simp( N [Int] [Type Int | Rest] New_ineq Ass Comp_ass )
    ;
%
% subst_simp( L E NI A CA  ) - find variable from list L to eliminate from
%   expression E to give new inequality NI, which is not inside
%   list of assumptions Ass or compound assumptions CA.
%
subst_simp( 0 [Var | Tl] Expr New_ineq Ass Comp_ass ) <-
    subst_simp( Var Expr New_ineq )
    write( "Eliminating " )
    write( Var )
    write( " from " )
    print_exp( Expr )
    write( " giving " )
    print_exp( New_ineq )
    nl
    ;
subst_simp( 1 [Var | Tl] Expr New_ineq Ass Comp_ass ) <-
    subst_simp( Var Expr New_ineq )
    not( contain_ass( New_ineq Ass Comp_ass ))
    write( "Eliminating " )
    write( Var )
    write( " from " )
    print_exp( Expr )
    write( " giving " )
    print_exp( New_ineq )
    nl
    ;
subst_simp( [Var | Tl] Expr New_ineq Ass Comp_ass ) <-
    subst_simp( Tl Expr New_ineq Ass Comp_ass )
    ;
%
% subst_simp( V E NE ) - eliminate variable V (possibly an integer) from
%    expression E to give new expression NE.
%
subst_simp( I [Type Int | Rest] [Type New_int | New_rest] ) <-
    is_int( I )
    div( Int I New_int )
    list_div( Rest I New_rest )
    ;
subst_simp( Var Expr New_ineq ) <-
    not( is_int( Var ))
    try_subst_exp( 1 Var Expr Sub_expr )
    check_canon( Sub_expr New_ineq )
    ;
%
% check_list( L1 A CA L2 ) - get integers and variables from L1 which are > 0
%   given assumptions A and comp. hyp. CA, put them into list L2.
%
check_list( [] Ass Comp_ass [] );
check_list( [Hd | Tl] Ass Comp_ass [Hd | New_tl] ) <-
    is_int( Hd )
    ne( Hd 1 ) ne( Hd -1 )
    cut
    check_list( Tl Ass Comp_ass New_tl )
    ;
check_list( [Hd | Tl] Ass Comp_ass [Hd | New_tl] ) <-
    contain_ass( [g 0 [1 Hd]] Ass Comp_ass )
    cut
    check_list( Tl Ass Comp_ass New_tl )
    ;
check_list( [Hd | Tl] Ass Comp_ass New_tl ) <-
    check_list( Tl Ass Comp_ass New_tl )
    ;
%
% check_simp( IL L OL ) - create list OL by taking all terms of list
%   IL which appear in every term of list L.
%
%
check_simp( In_list [] In_list );
check_simp( L1 [Term1 | Rest_term] L3 ) <-
    check_simp( L1 Rest_term L2 )
    check_simp_term( L2 Term1 L3 )
    ;
%
% check_simp_term( IL T OL ) - create list OL by taking all terms of list
%   IL which appear in term T.
%
%
check_simp_term( [] Term [] );
check_simp_term( [Hd | Tl] Term [Hd | New_tl] ) <-
    member( Hd Term )
    cut
    check_simp_term( Tl Term New_tl )
    ;
check_simp_term( [Hd | Tl] [Neg_hd | Term_tl] [Hd | New_tl] ) <-
    is_int( Hd )
    sum( Hd Neg_hd 0 )
    cut
    check_simp_term( Tl [Neg_hd | Term_tl] New_tl )
    ;
check_simp_term( [Hd | Tl] Term New_tl ) <-
    check_simp_term( Tl Term New_tl )
    ;
%
% list_div( T I NT ) - divide list of terms T by integer I giving new list
%      NT.
%
%
list_div( [] _ [] );
list_div( [[Int | Term] | Tl] I [[Div_int | Term] | Div_tl] ) <-
    div( Int I Div_int )
    list_div( Tl I Div_tl )
    ;
%
% elim( A CA NA NCA ) - eliminate part of the compound assumption CA using
%   an assumption from A to give you new assumption NA and new compound
%   assumptions NCA respectively.
%
%
elim( Ass Comp_ass Add_ass New_comp_ass ) <-
    member( Exp Ass )
    in_compound( Exp Comp_ass Add_ass New_comp_ass )
    ;
%
% in_compound( E CA NA NCA ) - expression E occurs in one of the compound
%   assumptions of CA. When eliminated it gives the new single assumption
%   NA and the new list of compound assumptions NCA.
%
%
in_compound( [T1 I1 | Rest] Comp_ass New_ass NC ) <-
    delete( [[T2 I2 | Rest] [T3 I3 | Rest3]] Comp_ass NC )
    not_zero( T1 I1 T2 I2 )
    check_canon( [e I3 | Rest3] New_ass )
    print_step( [T1 I1 | Rest] [T2 I2 | Rest] [T3 I3 | Rest3] New_ass )
```

```
    ;
in_compound( [T1 I1 | Rest] Comp_ass New_ass NC ) <-
    negate( Rest Neg_rest )
    delete( [[T2 I2 | Neg_rest] [T3 I3 | Rest3]] Comp_ass NC )
    plus( I2 Neg_i2 0 )
    not_zero( T1 I1 T2 Neg_i2 )
    check_canon( [e I3 | Rest3] New_ass )
    print_step( [T1 I1 | Rest] [T2 I2 | Neg_rest] [T3 I3 | Rest3] New_ass )
    ;
in_compound( [T1 I1 | Rest] Comp_ass New_ass NC ) <-
    delete( [[T2 I2 | Rest2] [T3 I3 | Rest]] Comp_ass NC )
    not_zero( T1 I1 T3 I3 )
    check_canon( [e I2 | Rest2] New_ass )
    print_step( [T1 I1 | Rest] [T2 I2 | Rest2] [T3 I3 | Rest] New_ass )
    ;
in_compound( [T1 I1 | Rest] Comp_ass New_ass NC ) <-
    negate( Rest Neg_rest )
    delete( [[T2 I2 | Rest2] [T3 I3 | Neg_rest]] Comp_ass NC )
    plus( I3 Neg_i3 0 )
    not_zero( T1 I1 T3 Neg_i3 )
    check_canon( [e I2 | Rest2] New_ass )
    print_step( [T1 I1 | Rest] [T2 I2 | Rest2] [T3 I3 | Neg_rest] New_ass )
    ;
%
% not_zero( T1 I1 T2 I2 ) - true if expression with type T1 and integer I1
%   is non_zero as implied by expression with type T2 and integer I2.
%
%
not_zero( g X T Y ) <- ge( X Y );
not_zero( ge X T Y ) <- gt( X Y );
not_zero( e X T Y ) <- ne( X Y );
%
% prod_merge( A CA NA NCA C T ) - try to substitute bigger integer for
%   one of the variables in asumptions A or compound assumptions CA to
%   create new lists of normal and compound assumptions A and CA. C is the
%   list of conclusions, t a list of theorems which can be used.
%
%
prod_merge( Ass CA New_ass NCA Conc Thm ) <-
    member( [Type1 Int [-1 Var]] Ass )
    lt( Int 0 )
    check_prod( Var Ass CA Ass Term Prod_type Old_expr )
    merge_prod_exp( Int Var Term Old_expr Type1 Prod_type New_expr )
    not( contain_ass( New_expr Ass Comp_ass ))
    usef_merge( New_expr Ass CA New_ass NCA Conc Thm )
    print_merge( [Type1 Int [-1 Var]] Old_expr )
    ;
prod_merge( Ass CA New_ass NCA Conc Thm ) <-
    member( [[Type1 Int [-1 Var]] H2] CA )
    lt( Int 0 )
    check_prod( Var Ass CA Ass Term Prod_type Old_expr )
    merge_prod_exp( Int Var Term Old_expr Type1 Prod_type New_expr )
    not( contain_ass( New_expr Ass CA ))
    usef_merge( New_expr Ass CA New_ass NCA Conc Thm )
    print_merge( [Type1 Int [-1 Var]] Old_expr )
    ;
%
% check_prod( V A CA A T Pt Ty E ) - check if variable V occurs in term T
%   of expression E of lists of assumptions and compound assumptions
%   C and CA. Ty is the type of term T with V removed (must be ge or g).
%
check_prod( Var [[T I | Rest] | T1] Comp_ass Ass Term PT [T I | Rest] ) <-
    in_comp_prod( Var Rest Term )
    prod_type( Var Term Ass T PT )
    ;
check_prod( Var [[e I | Rest] | T1] Comp_ass Ass Term PT [e NI | Neg_r] ) <-
    negate( [I | Rest] [NI | Neg_r] )
    in_comp_prod( Var Neg_r Term )
    prod_type( Var Term Ass e PT )
    ;
check_prod( Var [Hd | T1] Comp_ass Ass Term Prod_type Old_expr ) <-
    check_prod( Var T1 Comp_ass Ass Term Prod_type Old_expr )
    ;
check_prod( Var [] [[[T I | Rest] Hd2] | T1] Ass Term PT [T I | Rest] ) <-
    in_comp_prod( Var Rest Term )
    prod_type( Var Term Ass T PT )
    ;
check_prod( Var [] [[[e I | Rest] Hd2] | T1] Ass Term PT [e NI | Neg_r] ) <-
    negate( [I | Rest] [NI | Neg_r] )
    in_comp_prod( Var Neg_r Term )
    prod_type( Var Term Ass e PT )
    ;
check_prod( Var [] [[Hd1 Hd2] | T1] Ass Term Prod_type Old_expr ) <-
    check_prod( Var [] T1 Ass Term Prod_type Old_expr )
    ;
%
% in_comp_prod( V E T ) - variable V occurs in compound product term T
%   of expression E.
%
%
in_comp_prod( Var [[1 | Rest] | T1] [1 | Rest] ) <-
     length( Rest L )
     gt( L 1 )
     member( Var Rest )
     ;
in_comp_prod( Var [[-1 | Rest] | T1] [-1 | Rest] ) <-
     length( Rest L )
     gt( L 1 )
     member( Var Rest )
     ;
in_comp_prod( Var [[Int | Rest] | T1] [Int | Rest] ) <-
     ne( Int 1 ) ne( Int -1 )
     member( Var Rest )
     ;
in_comp_prod( Var [Hd | T1] Term ) <-
     in_comp_prod( Var T1 Term )
     ;
%
% merge_prod_exp( I V T OE Ty Pt NE ) - form new expression NE from old
%   one OE by substituting integer I for variable V in term T. Ty is
%   the type of the first inequality and Pt is the product type.
%
merge_prod_exp( Int Var Term [Old_t I1 | Old_l] g g [g I2 | New_l] ) <-
    cut
    plus( Int Neg_int 0 )
    delete( Term Old_l Temp_l )
    subst_prod( Neg_int Var Term New_term )
    add_term( New_term [Old_t I1 | Temp_l] [Old_t I2 | New_l] )
    ;
merge_prod_exp( Int Var Term [Old_t I1 | Old_l] T1 T2 [ge I2 | New_l] ) <-
    plus( Int Neg_int 0 )
    delete( Term Old_l Temp_l )
    subst_prod( Neg_int Var Term New_term )
    add_term( New_term [Old_t I1 | Temp_l] [Old_t I2 | New_l] )
    ;
%
% prod_type( V T A Et Nt ) - determine new type Nt from product T with var
%   V removed and expression type Et. A is the list of assumptions.
%
prod_type( Var [] Ass Type g );
prod_type( Var [Var | T1] Ass Type New_type ) <-
    cut
    prod_type( Var T1 Ass Type New_type )
    ;
prod_type( Var [Hd | T1] Ass Type New_type ) <-
```

```
    is_int( Hd )
    gt( Hd 0 )
    prod_type( Var Tl Ass Type New_type )
    ;
prod_type( Var [Hd | Tl] Ass Type New_type ) <-
    member( [g 0 [1 Hd]] Ass )
    prod_type( Var Tl Ass Type New_type )
    ;
prod_type( Var [Hd | Tl] Ass g New_type ) <-
    member( [ge 0 [1 Hd]]  Ass )
    prod_type( Var Tl Ass g New_type )
    ;
prod_type( Var [Hd | Tl] Ass ge ge ) <-
    member( [ge 0 [1 Hd]] Ass )
    prod_type( Var Tl Ass ge New_type )
    ;
%
% merge( A CA A CA NA NCA C T ) - try to merge two assumptions from list of
%   ass. A and compound assumptions CA to create new list of assumptions NA,
%   and new list of compound assumptions NCA. C is a list of desired
%   conclusions, T a list of theorems which can be used.
%
merge( [Hd | Tl] Comp_ass Ass Comp_ass New_ass New_comp_ass Conc Thm ) <-
    decomp( Hd Type Int_part Expr )
    decomp_list( Expr L1 L2 )
    ne( L1 [] )
    negate( L1 Neg_expr )
    check_inside( Neg_expr Tl Comp_ass Merge_expr Old_expr )
    merge_expr( Type Int_part L2 Merge_expr New_expr )
    not( contain_ass( New_expr Ass Comp_ass ))
    usef_merge( New_expr Ass Comp_ass New_ass New_comp_ass Conc Thm )
    print_merge( Hd Old_expr )
    ;
merge( [Hd | Tl] Comp_ass Ass Comp_ass New_ass New_comp_ass Conc Thm ) <-
    decomp( Hd Type Int_part Expr )
    ne( Int_part 0 )
    sum( Int_part Neg_int 0 )
    check_inside( [Neg_int] Tl Comp_ass [Merge_t | Merge_tl] Old_expr )
    merge_expr( Type 0 Expr [Merge_t 0 | Merge_tl] New_expr )
    not( contain_ass( New_expr Ass Comp_ass ))
    usef_merge( New_expr Ass Comp_ass New_ass New_comp_ass Conc Thm )
    print_merge( Hd Old_expr )
    ;
merge( [Hd | Tl] Comp_ass Ass Comp_ass New_ass New_comp_ass Conc Thm ) <-
    merge( Tl Comp_ass Ass Comp_ass New_ass New_comp_ass Conc Thm )
    ;
merge( [] [[H1 H2] | Tl] Ass Comp_ass New_ass New_comp_ass Conc Thm ) <-
    decomp( H1 Type Int_part Expr )
    decomp_list( Expr L1 L2 )
    ne( L1 [] )
    negate( L1 Neg_expr )
    check_inside( Neg_expr [] Tl Merge_expr Old_expr )
    merge_expr( Type Int_part L2 Merge_expr New_expr )
    not( contain_ass( New_expr Ass Comp_ass ))
    usef_merge( New_expr Ass Comp_ass New_ass New_comp_ass Conc Thm )
    print_merge( H1 Old_expr )
    ;
merge( [] [[H1 H2] | Tl] Ass Comp_ass New_ass New_comp_ass Conc Thm ) <-
    decomp( H1 Type Int_part Expr )
    ne( Int_part 0 )
    sum( Int_part Neg_int 0 )
    check_inside( [Neg_int] [] Tl [Merge_t | Merge_tl] Old_expr )
    merge_expr( Type 0 Expr [Merge_t 0 | Merge_tl] New_expr )
    not( contain_ass( New_expr Ass Comp_ass ))
    usef_merge( New_expr Ass Comp_ass New_ass New_comp_ass Conc Thm )
    print_merge( H1 Old_expr )
    ;
merge( [] [[H1 H2] | Tl] Ass Comp_ass New_ass New_comp_ass Conc Thm ) <-
    merge( [] Tl Ass Comp_ass New_ass New_comp_ass Conc Thm )
    ;
%
% decomp_list( L L1 L2 ) - decompose list L into lists L1 and L2.
%
decomp_list( [] [] [] );
decomp_list( [Hd | Tl] [Hd | Sub_tl1] Sub_tl2 ) <-
    decomp_list( Tl Sub_tl1 Sub_tl2 )
    ;
decomp_list( [Hd | Tl] Sub_tl1 [Hd | Sub_tl2] ) <-
    decomp_list( Tl Sub_tl1 Sub_tl2 )
    ;
%
% check_inside( E A CA ME OE ) - check if expression E
%   occurs inside list of assumptions A or CA with the remaining part
%   being ME, obtained from old expression OE.
%
check_inside( E1 [[T | E2] | Tl] Comp_ass [T | Rem_expr] [T | E2] ) <-
    decomp_list( E2 E1 Rem_expr )
    ;
check_inside( E1 [[e | E2] | Tl] Comp_ass [e | Rem_expr] [e | E2] ) <-
    negate( E2 Neg_expr )
    decomp_list( Neg_expr E1 Rem_expr )
    ;
check_inside( Expr1 [Hd | Tl] Comp_ass Remain Old_expr ) <-
    check_inside( Expr1 Tl Comp_ass Remain Old_expr )
    ;
check_inside( E1 [] [[[T | E2] Hd2] | Tl] [T | Rem_expr] [T | E2] ) <-
    decomp_list( E2 E1 Rem_expr )
    ;
check_inside( E1 [] [[[e | E2] Hd2] | Tl] [e | Rem_expr] [e | E2] ) <-
    negate( E2 Neg_expr )
    decomp_list( Neg_expr E1 Rem_expr )
    ;
check_inside( Expr1 [] [[Hd1 Hd2] | Tl] Remain Old_expr ) <-
    check_inside( Expr1 [] Tl Remain Old_expr )
    ;
%
% merge( A CA A CA NA NCA C T ) - try to merge two assumptions from list of
%   ass. A and compound assumptions CA to create new list of assumptions NA,
%   and new list of compound assumptions NCA. C is a list of desired
%   conclusions, T a list of theorems which can be used.
%
merge( [Hd | Tl] Comp_ass Ass Comp_ass New_ass New_comp_ass Conc Thm ) <-
    decomp( Hd Type Int_part Expr )
    check_inside( Merge_expr Tl Comp_ass Type )
    merge_expr( Type Int_part Expr Merge_expr New_expr )
    not( contain_ass( New_expr Ass Comp_ass ))
    usef_merge( New_expr Ass Comp_ass New_ass New_comp_ass Conc Thm )
    print_merge( Hd Merge_expr )
    ;
merge( [Hd | Tl] Comp_ass Ass Comp_ass New_ass New_comp_ass Conc Thm ) <-
    merge( Tl Comp_ass Ass Comp_ass New_ass New_comp_ass Conc Thm )
    ;
merge( [] [[H1 H2] | Tl] Ass Comp_ass New_ass New_comp_ass Conc Thm ) <-
    decomp( H1 Type Int_part Expr )
    check_inside( Merge_expr [] Tl Type )
    merge_expr( Type Int_part Expr Merge_expr New_expr )
    not( contain_ass( New_expr Ass Comp_ass ))
    usef_merge( New_expr Ass Comp_ass New_ass New_comp_ass Conc Thm )
    print_merge( H1 Merge_expr )
    ;
merge( [] [[H1 H2] | Tl] Ass Comp_ass New_ass New_comp_ass Conc Thm ) <-
    merge( [] Tl Ass Comp_ass New_ass New_comp_ass Conc Thm )
```

```
    ;
%
% check_inside( E A CA T ) - find expression E inside A or CA.
%
%
check_inside( Exp Ass Comp_ass _ ) <-
    member( Exp Ass )
    ;
check_inside( Exp Ass Comp_ass _ ) <-
    member( [Exp _] Comp_ass )
    ;
check_inside( [e | Neg_tl] Ass Comp_ass Type ) <-
    ne( Type e )
    member( [e | Tl] Ass )
    negate( Tl Neg_tl )
    ;
check_inside( [e | Neg_tl] Ass Comp_ass Type ) <-
    ne( Type e )
    member( [[e | Tl] _] Comp_ass )
    negate( Tl Neg_tl )
    ;
%
% print_merge( I1 I2 ) - print message with inequalities I1 and I2.
%
print_merge( Ineq1 Ineq2 ) <-
    write( " by merging " )
    print_exp( Ineq1 )
    write( " and " )
    print_exp( Ineq2 )
    nl
    ;
%
% merge_expr( T I ME E NE ) - get new expression NE from expression E and
%   inequality T with integer part I and merge-part ME.
%
merge_expr( e I1 L1 [T I2 | L2] New_exp ) <-
    add_exp( L1 L2 L3 )
    plus( I1 I2 I3 )
    check_canon( [T I3 | L3] New_exp )
    ;
merge_expr( g I1 L1 [T I2 | L2] [g I3 | L3] ) <-
    add_exp( L1 L2 L3 )
    plus( I1 I2 I3 )
    ;
merge_expr( ge I1 L1 [g I2 | L2] [g I3 | L3] ) <-
    add_exp( L1 L2 L3 )
    plus( I1 I2 I3 )
    ;
merge_expr( ge I1 L1 [T I2 | L2] [ge I3 | L3] ) <-
    ne( T g )
    add_exp( L1 L2 L3 )
    plus( I1 I2 I3 )
    ;
%
% decomp( A T I L ) - decompose assumption A in inequality type T, integer
%   part I and rest-part L.
%
%
decomp( [T Int | List] T Int List );
decomp( [e I | List] e Neg_int Neg_list ) <-
    negate( [I | List] [Neg_int | Neg_list] )
    ;
%
% add_exp( E E1 E2 ) - add new exp E to expression E1 to give E2.
%
%
add_exp( [] Expr Expr );
add_exp( Expr [] Expr ) <-
    ne( Expr [] )
    ;
add_exp( [[I | Rest] | Tl1] [[Neg_I | Rest] | Tl2] New_tl ) <-
    plus( I Neg_I 0 )
    add_exp( Tl1 Tl2 New_tl )
    ;
add_exp( [[I1 | Rest] | Tl1] [[I2 | Rest] | Tl2] [[I3 | Rest] | NT] ) <-
    plus( I1 I2 I3 )
    ne( I3 0 )
    add_exp( Tl1 Tl2 NT )
    ;
add_exp( [[I1 | R1] | Tl1] [[I2 | R2] | Tl2] [[I1 | R1] | New_tl] ) <-
    ne( R1 R2 )
    less_strong( R1 R2 )
    cut
    add_exp( Tl1 [[I2 | R2] | Tl2] New_tl )
    ;
add_exp( [[I1 | R1] | Tl1] [[I2 | R2] | Tl2] [[I2 | R2] | New_tl] ) <-
    ne( R1 R2 )
    add_exp( [[I1 | R1] | Tl1] Tl2 New_tl )
    ;
%
% usef_merge( E A CA NA NCA C T ) - try if merged expression E can be used
%   with assumptions A and compound assumptions CA to give new list of
%   assumptions NA and list of compound assumptions NCA. C is the list of
%   conclusions, and T the list of theorems.
%
usef_merge( Exp Ass Comp_ass [Exp] Comp_ass Conc Thm ) <-
    can_elim_conc( Conc Exp )
    write( "Conclude " )
    print_exp( Exp )
    ;
usef_merge( Exp Ass Comp_ass [Exp] Comp_ass Conc Thm ) <-
    thm_ass_elim( Thm [Exp] [] _ )
    write( "Obtained to use for theorem: " )
    print_exp( Exp )
    ;
usef_merge( Exp Ass Comp_ass [Exp] Comp_ass Conc Thm ) <-
    contrad( 1 Exp Ass Comp_ass )
    print_exp( Exp )
    ;
usef_merge( Exp Ass Comp_ass [Exp] Comp_ass Conc Thm ) <-
    find_new_subst( Exp Ass Comp_ass )
    print_exp( Exp )
    ;
usef_merge( Exp Ass Comp_ass [Add_ass Exp] New_comp_ass Conc Thm ) <-
    in_compound( Exp Comp_ass Add_ass New_comp_ass )
    ;
usef_merge( Exp Ass Comp_ass [New_exp] Comp_ass Conc Thm ) <-
    find_simp( 1 Exp Ass Comp_ass New_exp )
    ;
%
% print_exp( E ) - print out expression E in a readable manner.
%
print_exp( [Type Int] ) <-
    put( '0' )
    print_ineq( Type )
    write( Int )
    ;
print_exp( [Type Int | Rest] ) <-
    print_exp( 0 Rest )
    print_ineq( Type )
    write( Int )
    ;
print_exp( L [] );
print_exp( 0 [[1 Var] | Tl] ) <-
    cut
```

```
    write( Var )
    print_exp( 1 Tl )
    ;
print_exp( 1 [[1 Var] | Tl] ) <-
    cut
    put( '+' ) write( Var )
    print_exp( 1 Tl )
    ;
print_exp( L [[-1 Var] | Tl] ) <-
    cut
    put( '-' ) write( Var )
    print_exp( 1 Tl )
    ;
print_exp( 0 [[Int Var] | Tl] ) <-
    cut
    write( Int ) put( ' ' ) write( Var )
    print_exp( 1 Tl )
    ;
print_exp( 1 [[Int Var] | Tl] ) <-
    lt( Int 0 ) cut
    write( Int ) put( ' ' ) write( Var )
    print_exp( 1 Tl )
    ;
print_exp( 1 [[Int Var] | Tl] ) <-
    gt( Int 0 ) cut
    put( '+' ) write( Int ) put( ' ' ) write( Var )
    print_exp( 1 Tl )
    ;
print_exp( 0 [[1 | Rest] | Tl] ) <-
    cut
    print_exp( 2 Rest )
    print_exp( 1 Tl )
    ;
print_exp( 1 [[1 | Rest] | Tl] ) <-
    cut
    put( '+' ) print_exp( 2 Rest )
    print_exp( 1 Tl )
    ;
print_exp( L [[-1 | Rest] | Tl] ) <-
    cut
    put( '-' ) print_exp( 2 Rest )
    print_exp( 1 Tl )
    ;
print_exp( 0 [[Int | Rest] | Tl] ) <-
    cut
    write( Int )
    print_exp( 3 Rest )
    print_exp( 1 Tl )
    ;
print_exp( 1 [[Int | Rest] | Tl] ) <-
    lt( Int 0 ) cut
    write( Int )
    print_exp( 3 Rest )
    print_exp( 1 Tl )
    ;
print_exp( 1 [[Int | Rest] | Tl] ) <-
    gt( Int 0 ) cut
    put( '+' ) write( Int )
    print_exp( 3 Rest )
    print_exp( 1 Tl )
    ;
print_exp( 2 [Hd | Tl] ) <-
    write( Hd )
    print_exp( 3 Tl )
    ;
print_exp( 3 [Hd | Tl] ) <-
    put( ' ' ) write( Hd )
    print_exp( 3 Tl )
    ;
%
% print_ineq( T ) - print out what type of inequality it is.
%
print_ineq( ge ) <-
    write( " >= " )
    ;
print_ineq( e ) <-
    write( " = " )
    ;
print_ineq( g ) <-
    write( " > " )
    ;
%
% print_ass( N A ) - print out assumptions in A, N is a counter.
%
print_ass( 0 [] );
print_ass( N [] ) <-
    ne( N 0 )
    nl
    ;
print_ass( 0 [Hd | Tl] ) <-
    write( "   " )
    print_exp( Hd )
    print_ass( 1 Tl )
    ;
print_ass( 1 [Hd | Tl] ) <-
    write( "   " )
    print_exp( Hd )
    print_ass( 2 Tl )
    ;
print_ass( 2 [Hd | Tl] ) <-
    write( "   " )
    print_exp( Hd )
    nl
    print_ass( 0 Tl )
    ;
%
% print_comp_ass( CA ) - print out compound assumptions in CA.
%
print_comp_ass( [] );
print_comp_ass( [[Hd1 Hd2] | Tl] ) <-
    write( "   " )
    print_exp( Hd1 )
    write( ",  " )
    print_exp( Hd2 )
    nl
    print_comp_ass( Tl )
    ;
%
% print_step( E A1 A2 N ) - print out how new expression N was obtained
%    from expression E and compound hypothesis A1 A2.
%
print_step( Exp [T1 I1 | R1] [T2 I2 | R2] New_ass ) <-
    write( "From " )
    print_exp( Exp )
    write( ", and (" )
    print_int( I1 Pr_val1 )
    print_exp( Pr_val1 R1 )
    write( ") (" )
    print_int( I2 Pr_val2 )
    print_exp( Pr_val2 R2 )
    write( ") = 0, conclude " )
    print_exp( New_ass )
```

```
    nl
    ;
print_int( 0 0 );
print_int( I 1 ) <- ne( I 0 ) plus( I Neg_i 0 ) write( Neg_i );
%
% print_contrad( L E ) - print out contradiction of expression E, if L is 0.
%
print_contrad( 0 Exp ) <-
    write( "Contradiction " )
    print_exp( Exp )
    nl
    ;
print_contrad( 1 Exp );
%
% print_contrad( L E1 E2 ) - print out contradiction of inequalities
%   with expressions E1 and E2 if L is 0.
%
print_contrad( 0 E1 E2 ) <-
    write( "Contradiction " )
    print_exp( E1 )
    write( " and " )
    print_exp( E2 )
    nl
    ;
print_contrad( 1 E1 E2 );
```