ISRN SICS-R--91/12--SE

Logical Aspects of the Andorra Kernal Language

by

Torkel Franzén

SICS research report
R91:12
ISSN 0283-3638

SICS R91:12

Logical Aspects of the Andorra Kernel Language

Torkel Franzén

SICS

October 1991

SICS Research Report ISSN 0283-3638
Swedish Institute of Computer Science
Box 1263. S-164 28 Kista, SWEDEN

Abstract

The computation model for Kernel Andorra is presented, and some logical aspects treated. In particular a soundness theorem relative to a modified completion involving a logical interpretation of cut procedures is proved, together with a completeness theorem for wait-clauses.

Logical Aspects of the Andorra Kernel Language

Torkel Franzén
SICS, Box 1263, S-164 28 Kista, Sweden
telephone: +46 8 7521536
fax: +46 8 7517230
telex: 812 61 54 SICS
email: torkel@sics.se

0 Introduction

The work currently being done at SICS on the Andorra Kernel language, AKL, (for AKL and Andorra in general and related topics, see [HaJa], [JaHa], [Ma], [Sa], [Wa1], [Wa2], and further references given in these) has three main goals: i) implementing the language, ii) establishing its usefulness, iii) investigating its properties from a formal point of view. In this paper, some results pertaining to iii) are reported.

Since AKL incorporates committed choice programming as well as a variant of Prolog style programming (and combinations of these), its formal aspects are correspondingly various. Here we will regard AKL from a traditional logic programming point of view, inquiring into the soundness and completeness of its computation model under a suitable logical interpretation of programs. Basic results concerning these aspects will be presented.

The formal study of AKL is based on a system of rewrite rules on expressions that denote states in an AKL computation. The role of this system of rules corresponds to that of SLDNF resolution in the theoretical study of Prolog. However, the AKL rules have a much closer relation to the actual language than SLDNF resolution has to Prolog, since there is no control mechanism in AKL that is not represented in the system of rules. Further rules can and will be included to deal with special constructs in the language, such as bagof. The reason why this is feasible is of course that the rules operate on expressions containing a lot of information about the current state of a computation. In general, an informative representation of the execution of a program may lead to unwieldy constructs. We hope that the results and proofs to be presented here and elsewhere will convincingly show that the AKL model is actually useful for formulating and proving various results about the AKL language. The topics of the present paper - soundness and completeness under a logical interpretation of (a restricted class of) programs - are familiar from the analogous results for SLDNF and SLD resolution, but the results are necessarily somewhat different, and the proofs are very different. In particular, the separation of the

constraint theory from the logic programming mechanism (following [Co],[Ma],[Sa]) entails that there is nothing in the formulations or proofs of the results corresponding to the substitutions, liftings, etc, which figure so prominently in the investigation of SLD and SLDNF resolution. Also, the concepts of logical computations and logical programs defined below give a certain coherence to the the study of logical aspects of AKL. The topic is by no means exhausted in the present paper, which deals only with global logical properties of programs.

1 The syntax of clauses and goals

A *program atom* is an atomic formula of the form $p(x_1,..x_n)$ with different variables $x_1,..x_n$, called *head parameters*. A *constraint atom* is any atomic formula without function symbols in some constraint language. More about constraints in §2. It is assumed that these two classes of atomic formulas are disjoint. The remaining syntactic categories pertaining to programs are the following.

<guarded clause> ::= <head>:-<guard><guard operator><body>
<head> ::= <program atom>
<guard> ::= <non-empty sequence of atoms>
<atom> ::= <program atom>|<constraint atom>
<body> ::= <sequence of atoms>
<guard operator> ::= ':' | '!' | '|' (wait, cut, commit)

A *definition* is a finite sequence of guarded clauses with the same head atom and the same guard operator, defining the predicate of the head atom. We will speak of wait-definitions, cut-definitions, and commit-definitions. Cut and commit are the *pruning* guard operators. A *program* is a finite set of definitions where no two definitions define the same predicate, and the predicate of every program atom occurring in a clause in the program has a definition in the program. The *local parameters* of a clause are those variables that are not head parameters.

'%' below stands for any of the guard operators.

The execution of an Andorra program will be represented as a series of rewrites of goal expressions. The syntax of these expressions is defined as follows.

<and-box> ::= **and**(<non-empty sequence of local goals>)<set of variables>
<local goal> ::= <atom> | <choice-box>
<choice-box> ::= **choice**(<sequence of guarded goals with the same guard operator>)
<guarded goal> ::= <configuration><guard operator><sequence of atoms>

<or-box> ::= **or**(<sequence of configurations>)

<configuration> ::= <and-box> | <or-box>

<open goal> ::= <configuration> | <local goal>

<goal> ::= <open goal> | <guarded goal>

In the following, the letters R,S,T stand for sequences of goals, A for a sequence of atoms, and C for a sequence of constraint atoms. Any of these may stand for the empty sequence. The letter G (occasionally X) will be used for goals, and for the guard in a guarded clause. Concatenation of sequences will be written using comma, which is not likely to be confused with the use of comma for separating arguments.

The symbol **fail** will be used to denote the empty or-box and the empty choice-box, regarded as collapsing to the same object. G is a *total failure* if G is either **fail** or an or-box **or**(G_1,..G_n) where each G_i is a total failure.

The set of variables attached to an and-box contains the variables *local* to that box, introduced in local forking or in defining an initial goal, as described below. An and-box of the form **and**$(C)_V$ will normally be written as C_V.

A variable in G is *external* to a subgoal G' of G if it is local to some and-box properly containing G', or if it is not local to any and-box.

In a guarded goal, the configuration preceding the guard operator is the *guard* of the goal. The atoms following the guard operator are not regarded as goals, but become such when one of the promotion rules defined in §3 is applied to the goal.

We will speak of *subgoals* of goals in the obvious sense. That is,

G is a subgoal of G,

The subgoals of G_1,..G_n are subgoals of **and**$(G_1,..G_n)_V$ and of **or**(G_1,..G_n) and of **choice**(G_1,..G_n),

The subgoals of G are subgoals of G%A.

Subgoals of G other than G itself are called *proper* subgoals.

Usually, we will be talking about *occurrences* of subgoals, rather than of just the syntactic form of a subgoal. So the definitions below should in most cases be read as dealing with occurrences of goals. Since confusion between form and occurrence is unlikely, the distinction will mostly be implicit.

In the case where G is an and-box, an or-box, or a choice-box, we will speak of subgoals of G (including G itself) as occurring *in* G, and of G as a *surrounding* box. Again the qualification "properly" excludes G itself. The goals $G_1,..G_n$ will occasionally (mixing images) be called the *branches* of $\mathbf{or}(G_1,..G_n)$, $\mathbf{and}(G_1,..G_n)$, and $\mathbf{choice}(G_1,..G_n)$.

We will need the notion of *or-component*. A subgoal G of an or-box $G'=\mathbf{or}(G_1,..G_n)$ is an or-component of G' if G is G_i or is an or-component of G_i for some i=1,..n. For convenience we will also regard a goal of the form C_V as an (improper) or-component of itself.

Finally, the notion of *ordered context*. An and-box G in G' occurs in an ordered context if it is a subgoal of some cut-guarded choice-box G'' in G', but not of any commit-guarded choice-box in G''. More informally, the nearest surrounding pruning choice is a cut. Although used in the formulation of the non-deterministic promotion rule, this notion will otherwise play no role in the present paper.

2. Constraints

Constraints will be regarded as formulas in some constraint language. The only constraint formulas that occur in goals are atomic constraints without function symbols. The letters σ, τ, θ will be used for conjunctions of such constraints.

In the following, $\exists\sigma$ stands for the existential closure of σ, and $\exists V$, where V is the set $\{x_1,..x_n\}$, for (any permutation of) the quantifier sequence $\exists x_1..\exists x_n$. We allow V to be empty, in which case $\exists V$ is mere decoration. Bold letters $\mathbf{x}$, $\mathbf{y}$, $\mathbf{z}$ will be used for sequences $x_1,..x_n$ of variables when their length n (≥ 0) is immaterial. When the notation $G(\mathbf{x},\mathbf{y})$ or $\sigma(\mathbf{z})$ etc is used, it is not supposed that all the variables in $\mathbf{x}$,$\mathbf{y}$ or $\mathbf{z}$ actually occur in G or σ, but only that no other variables occur in G or σ.

We assume given some complete and consistent constraint theory **TC** defining the following logical properties of constraints:

σ is satisfiable iff $\mathbf{TC} \models \exists\sigma$,

θ is unconstrained by σ (or: σ does not restrict θ) outside V iff $\mathbf{TC} \models \theta \supset \exists V(\sigma \wedge \theta)$

This latter condition will also be expressed by saying that σ does not restrict θ with

respect to the complement of V.

The symbol **true** is used to denote a variable-free atomic formula for which it holds that **TC** $\models$ **true**. It will also be assumed that there is a variable-free atomic formula **false** for which **TC** $\models \neg$**false**. In definitions and proofs no further assumptions concerning the properties of **TC** will be made unless explicitly stated.

That **TC** is assumed to be complete is just a matter of theoretical convenience. Function symbols are excluded from constraint atoms because they have no role to play in the interpretation or execution of a program. This is no restriction from a theoretical point of view, since any definitions of the constraint atoms may be part of **TC**. In specifying a particular language following the AKL model we need to define not only the constraint theory **TC** but also the class of constraint atoms. In the familiar cases of Herbrand terms (or finite trees) and rational trees, we will speak of "AKL with Herbrand equalities" and "AKL with rational tree equalities", meaning that the predicate p of a constraint atom either is equality, or else has a definition in the constraint theory of the form

$$\forall \mathbf{x}(p(\mathbf{x}) \equiv s=t)$$

where s and t are terms containing no variable not in **x**. The constraint theories are in these cases **HC**, the complete elementary theory of Herbrand terms, and **RC**, the complete elementary theory of rational trees, both extended with the above definitions.

For a sequence R of goals, $\sigma(R)$ denotes the conjunction of the constraint atoms in the sequence, or **true** if there are no constraint atoms in R. The *environment* of (an occurrence of) a subgoal of a configuration G, or more explicitly its environment in G, is the conjunction of $\sigma(R)$ for every properly surrounding and-box $\mathbf{and}(R)_V$ in G, or **true** if there is no such and-box.

Constraints σ and τ are *incompatible* if $\sigma \wedge \tau$ is unsatisfiable. An occurrence of C_V in G is *quiet* if $\sigma(C)$ does not restrict the environment of C_V outside V.

3. Rewrite rules

The rewrite rules below are to be understood as applicable to any subgoal of a goal. That is, we define the rewrite relation on goals

$$G \rightarrow G'$$

as meaning that G' is obtainable from G by rewriting one occurrence of some subgoal X of G as X' in accordance with one of the rewrite rules

$$X \Rightarrow X'$$

defined below. (We will say that $G \rightarrow G'$ *via* $X \Rightarrow X'$.) So references to the environment of X refer to the environment of X as a subgoal of G. In speaking of the goal *to* which a rule is applied, we will always mean X rather than G; an application of a rule to a subgoal of G is an application *within* G. A *deterministic* operation is an application of any rule other than non-deterministic promotion.

Note that X and X' are always open goals, and X is never an or-box.

Local forking

A program atom subgoal A is rewritten by

$$A \Rightarrow \textbf{choice}(\textbf{and}(G_1)_{V_1}\%A_1,...\textbf{and}(G_n)_{V_n}\%A_n)$$

where H :- $G_i\%A_i$ (i=1,..n) is the sequence of clauses defining the predicate of A, with the arguments of A substituted for the head parameters, and the local parameters of the i:th clause replaced by the variables in the set V_i. When G is rewritten to G' by applying local forking to an atomic subgoal of G, the sets V_i are chosen to be disjoint from the set of variables in G.

Failure propagation

$$\textbf{and}(R,\textbf{fail},S)_V \Rightarrow \textbf{fail}$$

Choice elimination

$$\textbf{choice}(R,G\%A,S) \Rightarrow \textbf{choice}(R,S)$$

if G is a total failure.[1]

[1] G could be assumed to be **fail** here. The present formulation is needed when the soft pruning operations of §5 are used.

Environment synchronization

$$\mathbf{and}(R)_V \Rightarrow \mathbf{fail}$$

if $\sigma(R)$ is incompatible with the environment of the and-box.

Guard distribution

$$\mathbf{choice}(R,\mathbf{or}(G,S)\%A,T) \Rightarrow \mathbf{choice}(R,G\%A,\mathbf{or}(S)\%A,T)$$

Note that this rule does not apply to the soft pruning guards introduced in §5.

Deterministic promotion

$$\mathbf{and}(R,\mathbf{choice}(C_V\%A),S)_W \Rightarrow \mathbf{and}(R,C,A,S)_{V\cup W}$$

if C_V is satisfiable, and in case % is cut or commit is also quiet.

Cut promotion

$$\mathbf{choice}(R,C_V!A,S) \Rightarrow \mathbf{choice}(R,C_V!A)$$

if S is non-empty and C_V is satisfiable and quiet.

Commit promotion

$$\mathbf{choice}(R,C_V|A,S) \Rightarrow \mathbf{choice}(C_V|A)$$

if R or S is non-empty and C_V is satisfiable and quiet.

Non-deterministic promotion

$$\mathbf{and}(T_1,\mathbf{choice}(R,C_V{:}A,S),T_2)_W \Rightarrow \mathbf{or}(\mathbf{and}(T_1,C,A,T_2)_{V\cup W},\mathbf{and}(T_1,\mathbf{choice}(R,S),T_2)_W)$$

if R or S is non-empty. There are two further conditions associated with this rule. First, if the and-box occurs in an ordered context, it is required that R is empty. This stipulation is necessary for the concept of "the first solution" of a cut-guard to be well-defined. If it is

not included, we get the rule for *unordered hard cut*. In this paper, the distinction between the ordered and the unordered hard cut will play no role, and the concept of ordered context will not be used in the following. The second condition for non-deterministic promotion to be applicable is a bit complicated and merits separate treatment.

4. Stability and non-deterministic promotion.

A couple of definitions are useful in discussing non-deterministic promotion: a cut-guard or commit-guard in G of the form C_V is *suspended* if it is not quiet. A step *unfreezes* a suspended guard if the guard is quiet after that step. An application of environment synchronization, commit promotion, or cut promotion is *non-trivial* if the environment of the goal to which the operation is applied is consistent. Other operations are always non-trivial.

In an application of non-deterministic promotion as exhibited above we will say that the rule is applied *with respect to* the guard C_V. Also, an and-box of the form $\mathbf{and}(T_1,\mathbf{choice}(R,C_V{:}A,S),T_2)_W$ where R or S is non-empty will be called a *candidate* for non-deterministic promotion.

The basic Andorra principle of performing deterministic promotion in preference to non-deterministic promotion translates into a necessary condition for applying non-deterministic promotion, viz. that no deterministic operation is applicable to or within the and-box. This will be called the non-determinacy condition.

In AKL, the basic Andorra principle is given a stronger interpretation: non-deterministic promotion is to be delayed until i) the non-determinacy condition is satisfied, and ii) non-trivial deterministic operations on or within the and-box cannot become possible as a result of any future changes in the environment of the and-box through deterministic operations. This will be called the AKL principle.

The AKL principle will be seen to be a consequence of the restrictions on non-deterministic promotion now to be formulated. We distinguish between the AKL principle and the particular restrictions to be described here, since these restrictions, although guided by the AKL principle, also incorporate specific ideas concerning what is useful in programming. We need the concept of *stability:* an and-box G which is a subgoal of an and-box G' is stable (relative to G') if i) no deterministic operation is applicable to or within G, and ii) no series of rewrites of G', in which no restriction is imposed on non-deterministic promotion apart from the non-determinacy condition (and, for ordered hard cut, the restriction on ordered contexts), can lead to a goal in which a non-trivial deterministic operation is applicable to or within G. In speaking of stable and-boxes below, we will mean boxes stable relative to the whole configuration being rewritten. Note that the property of stability is well-defined (although in general undecidable) since

this definition does not invoke any further restrictions on non-deterministic promotion.

The conditions on non-deterministic promotion can now be stated: non-deterministic promotion is applicable to an and-box $G = \mathbf{and}(T_1,\mathbf{choice}(R,C_V{:}A,S),T_2)_W$ if and only if G has a nearest surrounding stable choice-box G' such that every candidate for non-deterministic promotion in G' has G' as its nearest surrounding stable and-box.

Given these restrictions, it easily follows that the AKL principle will be satisfied. For no deterministic operation is applicable to or within a subgoal of a stable and-box, and if G is a subgoal of a stable and-box, further rewrites may enable non-trivial deterministic operations within G only as a result of applications of non-deterministic promotion to surrounding and-boxes.

Additional conditions may be imposed on non-deterministic promotion, stipulating e.g. that non-deterministic promotion must be applied with respect to the leftmost possible solved guard, or that it must be applied to an innermost eligible candidate. However, such further restrictions need not be considered in this presentation, since the basic restriction alone implies that the Andorra principle holds, the soundness theorem is independent of any restrictions on non-deterministic promotion, and the completeness theorem holds whatever such further restrictions are imposed.

In an implementation of AKL one will need to replace the abstract definition of stability by a more managable sufficient condition for stability. Of course the part of the definition stipulating that no deterministic operation is applicable to G is non-problematic. What we need to consider are sufficient conditions for ruling out any future non-trivial application of the rules of of environment synchronization, commit promotion, and cut promotion.

To deal with environment synchronization, we can give a condition that is independent of the constraint theory **TC**. It is easily shown that any future non-trivial environment synchronization in G can be ruled out if the following *environment condition* holds:

For every and-box $\mathbf{and}(R)_V$ in G with environment τ in G, $\sigma(R)\wedge\tau$ does not restrict the environment of G with respect to the variables external to G.

We also need to formulate a *suspension stability condition,* that is, a condition that will ensure that no future deterministic step will unfreeze any suspended guard in G, unless the environment of G becomes inconsistent. The environment condition is not in general itself a suspension stability condition. For a counterexample, suppose G has external variables y_1,y_2, on which the environment of G imposes no condition, and a suspended constraint $x<y_1$, where the environment of the constraint in G is $x<y_2$. Assuming $<$ to refer to real numbers, the conjunction of these does not restrict the environment of G, but if $y_2<y_1$ is added to that environment, $x<y_1$ will become quiet. Similar counterexamples apply if the constraints include equalities in the field of real

numbers, or inequalities between Herbrand terms. These examples suggest that a suspension stability condition that covers many different constraint theories must be rather stringent. One sufficient condition is clearly that there are in G no suspended constraints containing external variables.

For the purposes of the present paper, it is not necessary to make any particular stipulations regarding suspension stability conditions. The soundness proof in fact makes no use at all of the stability condition, and for the completeness proof the suspension stability condition is irrelevant.

In the special cases of Herbrand equalities and rational tree equalities it turns out that the environment condition is in fact itself a suspension stability condition. These two types of constraints seem to be the only naturally occurring ones for which this holds. The proof is simple but a bit lengthy and is omitted here.

5. Soft versions of cut and commit

In the following there will be occasion to consider the effect of making either or both pruning guards "soft", i.e. restrict the pruning to the choice between clauses in a definition, but admit multiple solutions of the guard. We therefore introduce two more guard operators [!] and [|] - soft cut and soft commit - with the stipulation that the guard distribution rule (which is responsible for pruning alternative solutions of pruning guards) does not apply to the soft guard operators (although in the remaining rules, % stands also for the soft pruning operators). Furthermore we add the following rules for soft cut:

Soft cut transformation

$$\textbf{choice}(G[!]A) \Rightarrow \textbf{choice}(G:A)$$

Soft cut promotion

$$\textbf{choice}(R,G[!]A,S) \Rightarrow \textbf{choice}(R,G[!]A)$$

provided S is non-empty and G has a satisfiable and quiet or-component.

These rules are just a device for achieving the effect that all solutions of the soft guards will be considered within the framework of the rules of §3. For soft commit we can combine these two rules into one.

Soft commit promotion

$$\textbf{choice}(R,G[\,|\,]A,S) \Rightarrow \textbf{choice}(G{:}A)$$

provided G has a satisfiable and quiet or-component.

6. Some basic properties of the rules

The first thing to note, by an inspection of the rules and of the syntax of goals, is that rewriting a goal in accordance with one of the rules does in fact result in another goal. Thus, for example, replacing an occurrence of an and-box in a goal by an occurrence of an or-box yields a new goal.

An *initial* goal has the form $\textbf{and}(A)_V$, with V a subset of the set of variables occurring in any of the atoms in the non-empty sequence A (the existentially quantified variables in the initial goal). We define a *computation* as a finite or infinite sequence $G_1,G_2,..$of goals, where G_1 is an initial goal and $G_i \rightarrow G_{i+1}$ for every non-final i. A derivation $G_1',G_2',..$is a *subcomputation* of $G_1,G_2,..$if each G_i' is a subgoal of G_{k_i}, for some subsequence $G_{k_1},G_{k_2},..$ of $G_1,G_2,..$ The concepts of *partial computation* and *partial subcomputation* are defined similarly, except that it is not required that G_1 is an initial goal. We will use the notation $G \rightarrow^* G'$ to mean that there is a computation in which $G=G_i$ and $G'=G_j$ for some $i<j$.

It is easily seen that all goals occurring in a computation are and-boxes or or-boxes, that is, configurations. In particular, all configurations after the first top-level application of non-deterministic promotion (if there is such) will be or-boxes, in general nested, with no flattening taking place, since there is no rule for flattening or-boxes. Or-boxes occurring in guards, on the other hand, may be flattened in the course of a computation. (By using the guard distribution rule in combination with the choice elimination rule we can also reduce **or**(G) to G and **or**(R,**fail**) to **or**(R).)

We'll adopt the usual Prolog terminology and speak of an initial goal $\textbf{and}(A)_V$ as a *query*. We will sometimes write the query as $\exists VA$, or just as A if V is empty. C_W is an *answer* to a query G (relative to a program) if C_W is an or-component of some G' for which $G \rightarrow^* G'$. Note that $\sigma(C)$ need not be satisfiable: if an answer $\sigma(C)$ is satisfiable, it is a *solution*. A computation *fails* if it ends with a total failure - inspection of the rules shows that no rewrite can be made of a total failure. A query G fails if at least one computation starting with G fails, it *succeeds* if it has at least one solution.

Some further observations concerning the rules: if C_W is an answer to the query **and**$(A)_V$, V is a subset of W. To verify this, note that for a top level or-box or new top level and-box to appear in a computation, deterministic or non-deterministic promotion must be applied to the top-level and-box, and the resulting and-boxes will inherit the variables in V. Further rewrites will operate on the resulting and-boxes, and the observation follows by induction.

We also note that only those variables in A that occur in V will belong to W, since the remaining variables are not inherited and local forking only introduces new variables.

Finally, if C_W is an answer to $\exists VA$, $C_{W\setminus V}$ is an answer to A. More generally, given a computation from $\exists VA$ we get a computation from A by everywhere replacing V by $\varnothing$. This is seen by inspection of the rules, where the set of local variables, although carried along, does not enter into the conditions for a rule to apply.

A *complete* computation is one that is either finite, with no rule applicable to the final goal, or is infinite and satisfies the condition that no subgoal appears in the course of the computation such that i) the subgoal remains unchanged throughout the computation, and ii) for infinitely many configurations in the computation, some rule is applicable to the subgoal. In other words: in a complete computation no stone is left forever unturned, but every subgoal that appears is either eventually deleted or transformed through an application of some rule to that or some other subgoal, or else after a certain point no rule ever becomes applicable to the subgoal.

Computations in which every guard computation terminates, which will be referred to as*normal* computations, play an important role in AKL. In formal terms, a normal computation is one in which there is no infinite subcomputation beginning with a guard, and no choice-box to which guard distribution is applied infinitely many times in the computation. The use of deep guards in AKL implies that non-terminating guard computations are possible, but theoretically as well as from the point of view of actual programming they are usually to be regarded as pathological.

We will need the following

Local variables lemma: The local variables V of a guarded subgoal G'=**and**$(R)_V$%A of a goal G occur in a subgoal G" of G only if G" is a subgoal of G', or G' and G" occur in different disjunctive branches.

That G' and G" occur in different disjunctive branches means that there is an or-box **or**$(G_1,..G_n)$ or choice-box **choice**$(G_1,..G_n)$ in G such that G' is in G_i and G" in G_j for some $i\neq j$. We omit the proof of the lemma.

As a corollary to the the local variables lemma, in any and-box $\mathbf{and}(G_1,..G_n)_V$, the local variables of an and-box in G_i do not occur in G_j if $i \neq j$. (The syntactic argument needed to prove rather than perceive that goals in G_i and G_j do not lie in different disjunctive branches is omitted.) In particular, the local variables of an and-box do not occur in the environment of the and-box.

As another corollary to the lemma, all variables occurring in the environment of G' are external to G'.

7. Soundness

The soundness result to be proved refers to the completion of a program, in a fairly obvious sense. That is, for each definition

$$p(\mathbf{x}) :- G_i \% A_i \qquad (i=1,..k)$$

in the program in which % is not (hard or soft) cut, we form the logical formula

$$\forall \mathbf{x}(p(\mathbf{x}) \equiv \exists V_1(G_1 \& A_1) \vee ... \exists V_k(G_k \& A_k))$$

where V_i is the set of local parameters of the i:th clause. Here and in the following we use the convention that sequences of atomic formulas, when they occur in formulas as above, stand for the conjunction of the formulas in the sequence, or for **true** if the sequence is empty. For cut definitions, we form instead the formula

$$\forall \mathbf{x}(p(\mathbf{x}) \equiv \exists V_1(G_1 \& A_1) \vee (\neg \exists V_1 G_1 \& \exists V_2(G_2 \& A_2)) \vee \\ ... \vee (\neg \exists V_1 G_1 \& .. \& \neg \exists V_{k-1} G_{k-1} \& \exists V_k(G_k \& A_k)))$$

We call the resulting set of formulas Σ^*.

Talk of soundness, correctness, and logical consequence will in the following refer to the theory $\mathbf{T}=\mathbf{TC} \cup \Sigma^*$. Recall that **T** may in general be inconsistent, in which case soundness with respect to **T** is not a property of great interest. It is occasionally of interest, for technical reasons, to investigate sufficient conditions for **T** to be consistent. However, that **T** is consistent does not guarantee that computed answers are *sound* in any interesting (not merely technical) sense. Rather, the soundness result to be presented and the terminology used are predicated on the assumption that the axioms of **T** are in fact true on the intended interpretation of the program. Preoccupation with soundness in this sense seems reasonable since this assumption is often justified in declarative-style logic

programming.

The property to be established (under suitable further assumptions) is that of

Soundness of answers: if C_W is an answer to $\exists VA$, $\mathbf{T} \models \sigma(C) \supset A$.

First we note that soundness of answers implies

Soundness of failure: if $\exists VA$ fails, $\mathbf{T} \models \neg A$.

This follows from the fact that with the definition

p :- A!**false**
p :- **true**!

the completion of which is equivalent to $p \equiv \neg \exists A$, p will succeed if $\exists VA$ fails. Conversely, general soundness of failure implies soundness of answers. For, given that C_W is an answer to $\exists VA$, if we define

p(**x**) :- A!**false**
p(**x**) :- **true**!

where **x** are the variables occurring in A, the goal **and**(C,p(**x**)) will fail, so soundness of failure implies that $\mathbf{T} \models \sigma(C) \supset A$.

For a restricted class of programs, soundness of failure and soundness of answers are equivalent provided the class of programs is closed under the constructions above, as is the case with the class to be defined below.

For these soundness properties to hold, some restrictions must be imposed on programs, and the following discussion is devoted to this point. First note that the quietness condition in pruning operations corresponds to the stipulation in SLDNF resolution that only variable-free negated goals may be selected for resolution. Consider the definitions[2]

p(X) :- q(X)!**false**
p(X) :- **true**!

q(X) :- **true**:X=a

[2] Here and in later examples some obvious assumptions are made concerning equality in **TC**.

Here the query p(X) fails if no quietness restriction is imposed, although $\neg p(X)$ is not a logical consequence. (The same example with commit instead of cut shows that quietness is required also for commit promotion.)

Comparing this with the condition in SLDNF-resolution, of only activating negated goals when they are ground, we find some differences. First, the definition of quietness makes no reference to bindings, since, following [Co], [Ma], [Sa], we use the more general constraint scheme. Thus the quietness condition can be regarded as the natural generalization to arbitrary constraints of the SLDNF groundness condition (or rather, of the semantic rather than syntactic version of that condition). Note that we need not demand that pruning proceed only on the basis of completely quiet solutions - i.e. solutions that impose no restrictions on any variables - but instead allow them to impose restrictions on local parameters. From the point of view of negation as failure, we can do this because we only use negation as failure with an implicit existential quantification over all local parameters.

However, the quietness condition is not sufficient. Consider the example

p :- r(X)!X=a

r(X) :- **true**:X=b
r(X) :- **true**:X=a

Here p fails, incorrectly. We can avoid this type of situation in two ways. The first is to cut only the later clauses in a procedure, not alternative solutions of the guard - that is, to use soft cut instead of hard cut. p will then succeed. But we must be able to do a bit better, since some uses of hard cuts are clearly desirable - for example in the definition of negation as failure, where it is clearly pointless to check all solutions of the first guard when one solution has been found - and the question is when this is possible. Clearly one sufficient condition for the above situation not to arise with a hard cut definition is that the guard is deterministic. From considerations that will be made clear in §8 we arrive at the following somewhat more general condition. A cut-guarded definition has *indifferent* guards if for every clause

$$p(\mathbf{x}) :- G(\mathbf{x},\mathbf{y})!B(\mathbf{x},\mathbf{y})$$

in the definition, where the variables in $\mathbf{y}$ are the local parameters of the clause, it holds that

$$T \models G(\mathbf{x},\mathbf{y}) \,\&\, G(\mathbf{x},\mathbf{z}) \supset (\exists \mathbf{w} B(\mathbf{x},\mathbf{y}) \equiv \exists \mathbf{w} B(\mathbf{x},\mathbf{z}))$$

where the sequence **w** contains those variables in **y** that do not occur in G(**x**,**y**). This condition allows the guard to have several solutions, as long as they are equivalent as far as satisfying the body of the clause is concerned.

A simple syntactic sufficient condition for the guards of a hard cut definition to be indifferent is that they are *isolated,* in the sense that that no local parameter in any clause in the definition occurs both in the guard and in the body. (Note that this comfortably covers negation as failure.)

Next we turn to commit. Since the logical interpretation of the program makes no distinction between wait and commit, it is clear that we must impose more far-reaching restrictions than in the case of cut. Consider

r :- **true**|**false**
r: - **true**|**true**

Here r fails although $T \models r$. The obvious way of preventing this situation is to restrict attention to programs in which all commits have *incompatible* guards, i.e.

$$T \models \neg(\exists V_i G_i \& \exists V_j G_j)$$

for every pair of guards G_i, G_j taken from different clauses in the definition. Incompatible guards are no guarantee for the logical correctness of hard commit definitions, however, in view of the second counterexample:

p :- r(X)|X=a

r(X) :- **true**:X=b
r(X) :- **true**:X=a

A sufficient condition for commit definitions to be correct is that the guards are *authoritative* in the sense that for every clause

$$p(\mathbf{x}) \text{ :- } G(\mathbf{x},\mathbf{y})|B(\mathbf{x},\mathbf{y})$$

in the definition it holds that

$$T \models G(\mathbf{x},\mathbf{y}) \supset (p(\mathbf{x}) \equiv \exists \mathbf{w} B(\mathbf{x},\mathbf{y}))$$

where the sequence **w** contains those variables in **y** that do not occur in G(**x**,**y**). For example, the following definition has authoritative but not incompatible guards:

min(X,Y,Z) :- X≤Y|Z=X
min(X,Y,Z) :- X≥Y|Z=Y

given that TC ⊨ X≤Y & X≥Y ⊃ X=Y.

In the case of soft commit, this can be weakened to the condition that the guards are *weakly authoritative:* for every clause

$$p(\mathbf{x}) :- G(\mathbf{x},\mathbf{y})|B(\mathbf{x},\mathbf{y})$$

in the definition, it holds that

$$T \models G(\mathbf{x},\mathbf{y}) \supset (p(\mathbf{x}) \equiv \exists y(G(\mathbf{x},\mathbf{y})\&B(\mathbf{x},\mathbf{y}))$$

This condition covers the case of incompatible guards, since incompatible guards are easily seen to be weakly authoritative. The indifference condition has no role to play in the case of commit guards since, as is easily verified, weakly authoritative and indifferent guards are authoritative, and authoritative guards are indifferent.

Such a paradigmatic commit procedure as the usual definition of merge does not satisfy the conditions given. This is unavoidable, since e.g. the goal merge([1],[2],[1,2]) may fail although its negation is not a consequence of the completed definition. Of course in general one expects the merge procedure as actually used to be logically sound relative to the completed definition. In other words, one only makes such use of merge as is independent of how the lists happen to be merged. However, this is a property of particular programs and queries, to be proved by a special argument, and not a consequence of the logical form of the program.

For the formulation of the soundness theorem, we will now reformulate the above in somewhat different terms.

8. Logical computations and logical programs

Every goal has a natural interpretation as a logical formula. The inductive structure of the definition given here is dictated by the soundness proof.

We define an operation * assigning a formula G* to any goal G:

For an atomic goal A, A* is A.

(**fail**%A)* is $\bot$
fail* is $\bot$

For guard operators other than hard cut, (**or**(G_1,..G_n)%A)* is

$(G_1\%A)^* \vee ..(G_n\%A)^*$.

For hard cut, (**or**(G_1,..G_n)!A)* is

$(G_1!A_1)^* \vee (\neg G_1^* \& (G_2!A_2)^*) \vee .. (\neg G_1^* \& .. \& \neg G_{n-1}^* \& (G_n!A_n)^*)$

or(G_1,..G_n)* is $G_1^* \vee ..G_n^*$.

(**and**(G_1,..G_n)$_V$%A)* is $\exists V(G_1^* \& ..G_n^* \& A)$

and(G_1,..G_n)$_V$ is $\exists V(G_1^* \& ..G_n^*)$

For wait and commit choices, **choice**(G_1,..G_n)* is

$G_1^* \vee ..G_n^*$.

For cut choices, **choice**($G_1\%A_1$, ..$G_n\%A_n$)* is

$(G_1\%A_1)^* \vee (\neg G_1^* \& (G_2\%A_2)^*) \vee .. (\neg G_1^* \& .. \& \neg G_{n-1}^* \& (G_n\%A_n)^*)$

A computation $G_1 \rightarrow G_2 \rightarrow G_3$ ... is *logical* if it holds that $\mathbf{T} \models G_i^* \equiv G_{i+1}^*$ for every non-final i. A program is *logical* if every computation from that program is logical. Clearly all answers and failures obtained in a logical computation are sound.

The soundness theorem can now be formulated.

Soundness theorem:

All programs satisfying the following three conditions are logical:

i) All hard cut definitions have indifferent guards.
ii) All hard commit definitions have authoritative guards.
iii) All soft commit definitions have weakly authoritative guards.

While the quietness restrictions on pruning promotion operations are essential for the soundness theorem to hold, the restrictions on non-deterministic promotion are irrelevant to the proof of the theorem and will accordingly be ignored in the proof. Also, the quietness condition in the case of deterministic promotion of a pruning guard will not be used.

To check how optimal this theorem is, suppose a program Σ has the property that all computations are logical. Also pretend that a certain form of logical completeness holds. It is easily shown that this implies that every cut definition has the following property A. For every clause

$$p(\mathbf{x}) \text{ :- } G(\mathbf{x},\mathbf{y})!B(\mathbf{x},\mathbf{y})$$

in the definition, and for every constraint $\sigma(\mathbf{x},\mathbf{y})$, it holds that if

$$\mathbf{T} \models \sigma(\mathbf{x},\mathbf{y}) \supset G(\mathbf{x},\mathbf{y}) \text{ and } \mathbf{T} \models \exists \mathbf{x} \exists \mathbf{y} \sigma(\mathbf{x},\mathbf{y})$$

then

$$\mathbf{T} \models \exists \mathbf{y}(G(\mathbf{x},\mathbf{y}) \& B(\mathbf{x},\mathbf{y})) \supset \exists \mathbf{y}(\sigma(\mathbf{x},\mathbf{y}) \& B(\mathbf{x},\mathbf{y}))$$

If the definition has indifferent guards, property A clearly holds. In fact the proof of the soundness theorem could be carried through for property A instead of that of having indifferent guards, but the latter property is probably as significant in practice.

Similarly, it is easily shown under the same assumptions that every hard commit definition must have the following property B: For every clause

$$p(\mathbf{x}) \text{ :- } G(\mathbf{x},\mathbf{y})/B(\mathbf{x},\mathbf{y})$$

in the definition, and for every constraint $\sigma(\mathbf{x},\mathbf{y})$, it holds that

$$\text{if } \mathbf{T} \models \sigma(\mathbf{x},\mathbf{y}) \supset G(\mathbf{x},\mathbf{y}) \text{ then } \mathbf{T} \models \sigma(\mathbf{x},\mathbf{y}) \& p(\mathbf{x}) \supset \exists \mathbf{y}(\sigma(\mathbf{x},\mathbf{y}) \& B(\mathbf{x},\mathbf{y}))$$

Soft commit definitions must satisfy a weaker version of B:

$$\text{if } \mathbf{T} \models \sigma(\mathbf{x},\mathbf{y}) \supset G(\mathbf{x},\mathbf{y}) \text{ then } \mathbf{T} \models \sigma(\mathbf{x},\mathbf{y}) \& p(\mathbf{x}) \supset \exists \mathbf{y}(G(\mathbf{x},\mathbf{y}) \& B(\mathbf{x},\mathbf{y}))$$

Again these properties are implied by the conditions of having authoritative and weakly authoritative guards respectively.

9. Proof of the soundness theorem

For the proof of the soundness theorem, we need the following variant of $\Rightarrow$. Supposing θ to be a constraint not containing any variable local to an and-box in G, the relation

$$G \rightarrow\theta\ G' \text{ via } X \Rightarrow\theta\ X'$$

is defined as before, with the difference that references to the environment τ of X in the rewrite rules are replaced by references to $\theta\wedge\tau$[3]. This affects the incompatibility condition in environment synchronization, and the quietness conditions in pruning promotion. (Also the conditions on non-deterministic promotion, but as pointed out above, these are irrelevant to the proof.) Also, in local forking, the variables in V_i must not occur in θ.

Clearly, if $\mathbf{TC} \models \sigma\supset\theta$, $X \Rightarrow\theta\ X'$ implies $X \Rightarrow\sigma\ X'$.

A *θ-computation* is a computation using $\rightarrow\theta$, with the quietness restrictions on pruning promotion.

The observations in §6 apply also to $\Rightarrow\theta$ and to θ-computations. The difference between $\Rightarrow$ and $\Rightarrow\theta$ is relevant only in the observation that a θ-computation starting with $\mathbf{and}(A)_V$ remains a θ-computation when V is replaced everywhere by $\varnothing$. In view of the quietness restrictions, it is no longer true that the set of local variables does not enter into the conditions for a rule to apply. However, the variables in V are never relevant to the quietness test. Formally we can see this by noting that the test is only applied to an and-box within a choice-box, and the variables in V are never joined to the local variables of such an and-box.

In the following we will need a slightly modified notion of subgoal: in addition to the clauses defining the subgoal relation in §1, we stipulate that

if G' is an or-component of G, G'%A is a subgoal of G%A.

The soundness theorem follows (taking θ to be **true** and G to be G_i) from the following

[3] For some reason, conjunctions of constraints are written using $\wedge$ rather than &.

Main lemma: Under the assumptions of the soundness theorem, for every θ-computation

$$G_1 \rightarrow\theta\ G_2 \rightarrow\theta\ \ldots \text{ via } X_1 \Rightarrow\theta\ X_2 \Rightarrow\theta\ \ldots$$

and for every subgoal G of G_i such that X_i is a subgoal of G,

$$\text{if } G \rightarrow\theta\ G' \text{ via } X_i \Rightarrow\theta\ X_{i+1}, \text{ then } \mathbf{T} \cup \{\theta \wedge \tau\} \models G^* \equiv G'^*$$

where τ is the environment of G in G_i.

Proof of the main lemma. We prove the lemma by main induction on the length of the computation $G_1, G_2, \ldots G_i$ and subinduction on the complexity of G. That is, by the induction hypothesis the assertion holds for every subgoal G of a goal in a shorter computation, and in the given computation it holds for every proper subgoal G' of G.

The symbols $G_1, \ldots G_n$ will be continually recycled in the following, so it should not be assumed that they refer to the given computation, except for the particular index i - G_i is the last configuration in the computation considered. The plain letter G will usually be reserved for the subgoal at issue, and we will assume that $G \rightarrow\theta\ G'$ via $X \Rightarrow\theta\ X'$. Also, the letter τ will throughout stand for the environment of G. References to the environment of a goal, with no further qualification, will always mean the environment of the goal in G_i.

For a sequence of goals R, R* below will stand for either the conjunction or the disjunction of G_1* for G_1 in R, as determined by the context. The letters ϕ and ψ will be used for logical formulas in general.

The use of the main induction hypothesis can be factored out of the argument. Suppose we have a subgoal of G_i of the form G%A. By the new definition of subgoal, there is a goal G'%A occurring in a choice-box in G_i, with G an or-component of G'. The choice-box has been created by local forking; the definition giving rise to the choice-box will be referred to as the *parent definition* of G; the program atom to which the local forking is applied is the *parent atom*.Thus there is a sequence

$$\mathbf{and}(A')_V\%A = G_1\%A, \ldots G'\%A$$

where $\mathbf{and}(A')_V\%A$ is a guarded goal created with the choice-box and associated with a

clause of the parent definition, and each guarded goal in the sequence is obtained from its immediate predecessor either by an application of a rewrite rule, or as one of the two goals G"%A and or(S)%A obtained by application of guard distribution to the choice-box. The box **and**$(A')_V$ will be referred to as the *parent guard* of G.

The sequence $G_1,..G'$ of guards is not in general a θ-computation. However, it becomes a $\theta\wedge\sigma$-computation, where σ is the environment of the choice-box (and of G), if we simply undo all guard distribution applications, re-uniting G" and S into an or-box or(G",S). (Here we must note that environments never shrink, so σ is logically stronger than the environments of the earlier occurrences of the choice-box.) Finally we replace V everywhere by $\varnothing$, obtaining a computation beginning with A' and ending with some configuration G_m. Since m<i, we can apply the induction hypothesis and conclude that

(1) $T \cup \{\theta\wedge\sigma\} \models A' \equiv G_m{}^*$

We will draw some conclusions from (1). First, suppose G has an or-component **and**$(R)_W$. Since **and**$(R)_{W\setminus V}$ is an or-component of G_m, it follows that

(2) $T \cup \{\theta\wedge\sigma\} \models R^* \supset A'$

and (since the variables in W do not occur in $\theta\wedge\sigma$)

(3) $T \cup \{\theta\wedge\sigma\} \models \exists W(R^*\&A) \supset \exists V(A'\&A)$

From (3) and the definition of (G%A)* we see that it holds generally (i.e. also in the case where G is an or-box) that

(4) $T \cup \{\theta\wedge\sigma\} \models (G\%A)^* \supset \exists V(A'\&A)$

Finally, suppose G%A is a branch in a soft pruning choice-box in G_i. In this case, there are no applications of guard distribution within the choice-box, and we can conclude that

(5) $T \cup \{\theta\wedge\sigma\} \models (G\%A)^* \equiv \exists V(A'\&A)$.

(2), (4), and (5) will be referred to as the *parent guard relations*.

Now for the proof. There are a number of cases to consider. G may be a program atom, an and-guarded goal **and**$(G_1,..G_n)_V$%A, an and-box **and**$(G_1,..G_n)_V$, an or-guarded goal **or**$(G_1,..G_n)$%A, an or-box **or**$(G_1,..G_n)$, or a choice-box **choice**$(G_1,..G_n)$.

If G is a program atom A, X must be A, and X' is obtained by local forking:

$$A \Rightarrow\theta \textbf{ choice}(\textbf{and}(G_1)_{V_1}\%A_1,...\textbf{and}(G_n)_{V_n}\%A_n)$$

In this case it is easily seen that $\Sigma^* \models G^* \equiv G'^*$ by the definition of the completion.

Next assume G is $\textbf{and}(G_1,..G_n)_V\%A$. First consider the subcase where X is a proper subgoal of $\textbf{and}(G_1,..G_n)_V$. We may assume that X is a subgoal of G_1. So the step from G to G' is

$$\textbf{and}(G_1,..G_n)_V\%A \rightarrow\theta \textbf{ and}(G_1',..G_n)_V\%A$$

where $G_1 \rightarrow\theta\ G_1'$. The environment of G_1 is $\sigma\wedge\tau$, where $\sigma=\sigma(G_2,..G_n)$. By induction hypothesis, $\mathbf{T} \cup \{\theta\wedge\sigma\wedge\tau\} \models G_1^* \equiv G_1'^*$, from which it follows that

$$\mathbf{T} \cup \{\theta\wedge\tau\} \models \exists V(G_1^* \,\&\, ..G_n^*\&A) \equiv \exists V(G_1'^* \,\&\, ..G_n^*\&A)$$

since $G_2^* \,\&\, ..G_n^* \models \sigma$, and the variables in V do not occur in $\theta\wedge\tau$.

If X is not a proper subgoal of $\textbf{and}(G_1,..G_n)_V$, we have several possible rules. First, failure propagation:

$$\textbf{and}(R,\textbf{fail},S)_V\%A \rightarrow\theta \textbf{ fail}\%A$$

Here $(\textbf{fail}\%A)^*$ is $\perp$ by definition, and $(\textbf{and}(R,\textbf{fail},S)_V\%A)^*$ is $\exists V(R^*\&\perp\&S^*\&A)$, so $G^* \equiv G'^*$ is a logical truth in this case.

The next possibility is environment synchronization:

$$\textbf{and}(R)_V\%A \rightarrow\theta \textbf{ fail }\%A$$

where $\mathbf{TC} \models \neg(\theta\wedge\tau\wedge\sigma(R))$. $(\textbf{and}(R)_V\%A)^*$ is $\exists V(R^*\&A)$, and since $R^* \models \sigma(R)$ and the variables in V do not occur in $\theta\wedge\tau$, it follows that $\mathbf{TC} \cup \{\theta\wedge\tau\} \models \neg\exists V(R^*\&A)$.

Next deterministic promotion:

$$\textbf{and}(R,\textbf{choice}(C_V\%A'),S)_W\%A \rightarrow\theta \textbf{ and}(R,C,A',S)_{V\cup W}\%A$$

Here, as is easily checked, G* and G'* are logically equivalent, given that the variables in V do not occur in R or S or A.

Finally (within the case where G is $\textbf{and}(G_1,..G_n)_V\%A$), we have non-deterministic

promotion:

$$\textbf{and}(T_1,\textbf{choice}(R,C_V{:}A',S),T_2)_W\%A$$
$$\rightarrow\theta$$
$$\textbf{or}(\textbf{and}(T_1,C,A',T_2)_{V\cup W},\textbf{and}(T_1,\textbf{choice}(R,S),T_2)_W)\%A$$

In the case where % is not hard cut, it is again easily verified that G* and G'* are logically equivalent. So suppose % is !. Let ϕ_1 be $T_1{*}\&T_2{*}$, ϕ_2 be $R{*}\vee S{*}$, and ϕ_3 be $\exists V(C\&A')$. With this notation, G* is

$$\exists W(\phi_1 \,\&\, (\phi_2 \vee \phi_3) \,\&\, A)$$

and G'* is equivalent, given that the variables in V do not occur in ϕ_1, to

$$\exists W(\phi_1 \,\&\, \phi_3 \,\&\, A) \vee (\neg\exists W(\phi_1 \,\&\, \phi_3) \,\&\, \exists W(\phi_1 \,\&\, \phi_2 \,\&\, A))$$

Some logical manipulation yields that $\mathbf{T} \cup \{\theta\wedge\tau\} \models G{*}\equiv G'{*}$ holds if and only if

(6) $\quad \mathbf{T} \cup \{\theta\wedge\tau\} \models \exists W(\phi_1 \,\&\, \phi_2 \,\&\, A) \,\&\, \exists W(\phi_1 \,\&\, \phi_3) \supset \exists W(\phi_1 \,\&\, \phi_3 \,\&\, A)$

Let $\textbf{and}(A'')_{V'}$ be the parent guard of $\textbf{and}(T_1,\textbf{choice}(R,C_V{:}A',S),T_2)_W$. Since hard cut definitions have indifferent guards, it holds that

$$\mathbf{T} \models \exists V'(A''\&A)\supset\forall V'(A''\supset\exists U A)$$

where U is the set of variables in V' that do not occur in A''. From (2) we get

(7) $\quad \mathbf{T} \cup \{\theta\wedge\tau\} \models \phi_1 \,\&\, \phi_2\supset A''$
(8) $\quad \mathbf{T} \cup \{\theta\wedge\tau\} \models \phi_1 \,\&\, \phi_3\supset A''$

Hence

$$\mathbf{T} \cup \{\theta\wedge\tau\} \models \exists V'(\phi_1 \,\&\, \phi_2 \,\&\, A)\supset\forall V'(\phi_1 \,\&\, \phi_3\supset A)$$

from which (6) follows by some further manipulations, using the fact that the variables in W\V' do not occur in A or A'' and the variables in U do not occur in ϕ_1 & ϕ_3.

This concludes the proof for the case in which G is of the form $\textbf{and}(G_1,..G_n)_V\%A$. The case where G is an and-box $\textbf{and}(G_1,..G_n)_V$ requires no further work, since the reasoning is just the same as in the case $\textbf{and}(G_1,..G_n)_V{:}\textbf{true}$. So we turn to the case of a guarded goal with an or-box as guard.

Let G be **or**(G_1,..G_n)%A. There is only one subcase to consider, since X must be a proper subgoal of **or**(G_1,..G_n), say a subgoal of G_1. The step from G to G' is

$$\textbf{or}(G_1,..G_n)\%A \rightarrow\theta\ \ \textbf{or}(G_1',..G_n)\%A$$

The environment of G_1 is τ, so by the induction hypothesis $T \cup \{\theta\wedge\tau\} \models G_1^* \equiv G_1'^*$. If % is not hard cut, G* is $(G_1\%A)^* \vee ..(G_n\%A)^*$ and G'* is $(G_1'\%A)^* \vee ..(G_n\%A)^*$, so $T \cup \{\theta\wedge\tau\} \models G^* \equiv G'^*$ follows by the induction hypothesis applied to G_1%A. If % is !, G* is

$$(G_1!A_1)^* \vee (\neg G_1^* \& (G_2!A_2)^*) \vee .. (\neg G_1^* \& .. \& \neg G_{n-1}^* \& (G_n!A_n)^*)$$

and G'* is

$$(G_1'!A_1)^* \vee (\neg G_1'^* \& (G_2!A_2)^*) \vee .. (\neg G_1'^* \& .. \& \neg G_{n-1}^* \& (G_n!A_n)^*)$$

so again the conclusion follows by the induction hypothesis applied to G_1 and to $G_1!A_1$.

The case where G is **or**(G_1,..G_n) is treated just like **or**(G_1,..G_n):**true**.

There remains only the case where G is **choice**($G_1\%A_1$,..$G_n\%A_n$). First suppose X is a subgoal of G_1:

$$\textbf{choice}(G_1\%A_1,..G_n\%A_n) \rightarrow\theta\ \ \textbf{choice}(G_1'\%A_1,..G_n\%A_n)$$

If % is wait or commit, G* is $(G_1\%A_1)^* \vee ....(G_n\%A_n)^*$, if % is hard or soft cut, G* is $(G_1\%A_1)^* \vee (\neg G_1^* \& (G_2\%A_2)^*) \vee .. (\neg G_1^* \& .. \& \neg G_{n-1}^* \& (G_n\%A_n)^*)$. In both cases, the equivalence of G* and G'* follows from the induction hypothesis applied to G_1 and $G_1\%A_1$.

If X is G, there are seven rules to consider. First choice elimination:

$$\textbf{choice}(R,G''\%A,S) \Rightarrow\theta\ \textbf{choice}(R,S)$$

where G'' is a total failure. Here it easily checked, distinguishing between the wait or commit case and the cut case, that G* and G'* are logically equivalent.

In the case of guard distribution:

$$\textbf{choice}(R,\textbf{or}(G_1,..G_n)\%A,T) \Rightarrow\theta\ \textbf{choice}(R,G_1\%A,\textbf{or}(G_2,..G_n)\%A,T)$$

we also need to check the case where % is wait or commit, and the case where it is hard cut. (The case n=1 is trivial, so we assume n>1.) In the former case, G* and G'* are both clearly equivalent to

$$R^* \vee (G_1 \% A)^* \vee ..(G_n \% A)^* \vee T^*$$

In the latter case, for suitable ϕ_1, ϕ_2, ψ, G* is equivalent to

$$\phi_1 \vee (\neg\psi \;\&\; (\mathbf{or}(G_1,..G_n)!A)^*) \vee (\neg\psi \;\&\; \neg\mathbf{or}(G_1,..G_n)^* \;\&\; \phi_2)$$

and G'* to

$$\phi_1 \vee (\neg\psi \;\&\; (G_1!A)^*) \vee (\neg\psi \;\&\; \neg G_1^* \;\&\; (\mathbf{or}(G_2,..G_n)\%A)^*) \vee (\neg\psi \;\&\; \neg G_1^* \;\&\; \neg\mathbf{or}(G_2,..G_n)^* \;\&\; \phi_2)$$

Since $(\mathbf{or}(G_1,..G_n)!A)^*$ is

$$(G_1!A_1)^* \vee (\neg G_1^* \& (G_2!A_2)^*) \vee .. (\neg G_1^* \& .. \& \neg G_{n-1}^* \& (G_n!A_n)^*))$$

and $(\mathbf{or}(G_1,..G_n)!A)^*$ is $(G_1!A)^* \vee ..(G_n!A)^*$, G* and G'* are seen to be logically equivalent.

Next the case of cut promotion:

$$\mathbf{choice}(R,C_V!A,S) \Rightarrow\theta\ \mathbf{choice}(R,C_V!A)$$

G* is of the form

$$\phi_1 \vee (\neg\psi \;\&\; \exists V(C\&A)) \vee (\neg\psi \;\&\; \neg\exists VC \;\&\; \phi_2)$$

and G'* is

$$\phi_1 \vee (\neg\psi \;\&\; \exists V(C\&A))$$

Given that C_V is quiet, i.e. $\mathbf{TC} \cup \{\theta\wedge\tau\} \models \exists VC$, it follows that these two are equivalent in $\mathbf{TC} \cup \{\theta\wedge\tau\}$.

Commit promotion is a bit less straightforward:

$$\textbf{choice}(R,C_V \mid A,S) \Rightarrow\theta\ \textbf{choice}(C_V \mid A)$$

To show that $\mathbf{T} \cup \{\theta\wedge\tau\} \models G^*\equiv G'^*$ we need to establish that

(9) $\quad \mathbf{T} \cup \{\theta\wedge\tau\} \models (G_1 \mid A_1)^*\supset\exists V(C\&A)$

for any guarded goal $G_1 \mid A_1$ in R,S. So let the parent guard of C_V be $\textbf{and}(A')_{V'}$, and of G_1 be $\textbf{and}(A'')_{V''}$, and let their parent atom be $p(x_1,..x_n)$. By the parent goal relations, it holds that

$\mathbf{T} \cup \{\theta\wedge\tau\} \models (G_1 \mid A_1)^*\supset p(x_1,..x_n)$
$\mathbf{T} \cup \{\theta\wedge\tau\} \models C\supset A'$

and since hard commit definitions have authoritative guards, it also holds that

$\mathbf{T} \models p(x_1,..x_n)\supset(A'\supset\exists UA)$

where U is the set of variables in V' that do not occur in A'.

Combining these three, we get

$\mathbf{T} \cup \{\theta\wedge\tau\} \models (G_1 \mid A_1)^*\supset(C\supset\exists UA)$

and since $\mathbf{T} \cup \{\theta\wedge\tau\} \models \exists VC$, the variables in V do not occur in $\theta\wedge\tau$ or free in $(G_1 \mid A_1)^*$, and the variables in U do not occur in C, (9) follows.

Next we consider soft cut transformation:

$$\textbf{choice}(G_1[!]A) \Rightarrow\theta\ \textbf{choice}(G_1{:}A)$$

If G_1 is an and-box, G^* and G'^* are identical. If G_1 is an or-box, $\textbf{or}(G_2,..G_n)$, G^* is $(G_2[!]A)^* \vee ..(G_n[!]A)^*$ and G'^* is $(G_2{:}A)^* \vee ..(G_n{:}A)^*$. By induction, then, (independent of the induction in the proof as a whole), G^* and G'^* are identical.

The case of soft cut promotion:

$$\textbf{choice}(R,G[!]A,S) \Rightarrow\theta\ \textbf{choice}(R,G[!]A)$$

is similar to that of hard cut promotion. We need only note that if G has a quiet or-

component C_V, $\mathbf{TC} \cup \{\theta \wedge \tau\} \models G^*$.

Finally, soft commit promotion:

$$\textbf{choice}(R, G_1[|]A_1, S) \Rightarrow_\theta \textbf{choice}(G_1 : A_1)$$

First, by the same argument as in the case of soft cut transformation, $\textbf{choice}(G_1:A_1)^*$ is the same as $\textbf{choice}(G_1[|]A_1)^*$. Again let the parent guard of G_1 be $\textbf{and}(A')_{V'}$, and of G_2, for a guarded goal $G_2 | A_2$ in R,S, be $\textbf{and}(A'')_{V''}$, and let their parent atom be $p(x_1,..x_n)$. G_1 has a quiet or-component C_V. By the parent goal relations (with a minor twist, combining the arguments for (1) and (2)), it again holds that

$\mathbf{T} \cup \{\theta \wedge \tau\} \models (G_2 | A_2)^* \supset p(x_1,..x_n)$
$\mathbf{T} \cup \{\theta \wedge \tau\} \models C \supset A'$

Since soft commit definitions have weakly authoritative guards, it holds that

$\mathbf{T} \models p(x_1,..x_n) \supset (A' \supset \exists V'(A' \& A_1))$

and so it follows that

$\mathbf{T} \cup \{\theta \wedge \tau\} \models (G_2 | A_2)^* \supset \exists V'(A' \& A_1)$

By the parent goal relations for soft commit,

$\mathbf{T} \cup \{\theta \wedge \tau\} \models (G_1 | A_1)^* \equiv \exists V'(A' \& A_1)$

and hence

$\mathbf{T} \cup \{\theta \wedge \tau\} \models (G_2 | A_2)^* \supset (G_1 | A_1)^*$

10. A completeness theorem

The completeness theorem given here deals only with wait-clauses, i.e. the part of AKL corresponding to the execution of Horn clauses in Prolog. So given a program composed entirely of wait-definitions, let Σ be the set of corresponding definite clauses:

$$\forall \mathbf{x} \forall V_i (G_i \& A_i \supset p(\mathbf{x}))$$

It is easily verified that $\Sigma \models G_{i+1}{}^* \supset G_i{}^*$ for all configurations in a computation $...G_i \rightarrow G_{i+1}..$ Hence AKL execution is sound with respect to Σ, in the following sense: if $C(\mathbf{x},\mathbf{y})_W$ is a solution to the query $\mathbf{and}(A(\mathbf{x}))_V$, it holds that i) $\Sigma \models C(\mathbf{x},\mathbf{y}) \supset A(\mathbf{x})$, and ii) $\mathbf{TC} \models \exists \mathbf{x} \exists \mathbf{y} C(\mathbf{x},\mathbf{y})$. (Here we use the observation that if C_W is an answer to $\exists V A$, $C_{W \setminus V}$ is an answer to A.) The following restricted converse holds:

Completeness theorem

Suppose Π is a complete normal computation starting with the query $\mathbf{and}(A(\mathbf{x}))_V$. Then if i) and ii) hold, Π contains (i.e. some configuration in Π has as an or-component) a solution $D(\mathbf{x},\mathbf{z})_W$ such that $\models C(\mathbf{x},\mathbf{y}) \supset \exists \mathbf{z} D(\mathbf{x},\mathbf{z})$.

The restriction to normal computations is needed because of the conditions for non-deterministic promotion to be applicable in AKL. A goal of the form **and**(p,q) will loop if p is non-deterministic and q is deterministic and loops, even if p has **false** as the body of each clause. When such a goal occurs in a guard it may prevent a solution given by another clause from being promoted. So the theorem above states that essentially this is the only problematic situation. The theorem holds, as the proof will show, for any restriction on the application of non-deterministic promotion, as long as the following principle is satisfied:

If G is a stable and-box containing (properly or improperly) at least one candidate for non-deterministic promotion, then there is at least one such candidate to which non-deterministic promotion is applicable.

If there is no recursion involved in the guard of any clause in the program, i.e. the guards are essentially flat, *any* complete computation will clearly be a normal one. With recursive guards, there may or may not be any complete normal computation from $\mathbf{and}(A(\mathbf{x}))_V$, and for a given logical consequence there may or may not be any computation in which that consequence is found.

Proof of the completeness theorem.

Let $\sigma(\mathbf{x},\mathbf{y})$ be the conjunction of the atomic constraints $\sigma_1,...\sigma_n$, where $\mathbf{TC} \models \exists \mathbf{x} \exists \mathbf{y} \sigma$. Assume Π is a complete normal computation (which we may assume does not contain any of the variables in $\mathbf{y}$) starting with the query $G_1 = \mathbf{and}(A(\mathbf{x}))$ (we let $V=\varnothing$ to save some notation). Also assume $\Sigma \models \sigma(\mathbf{x},\mathbf{y}) \supset A(\mathbf{x})$, i.e. $\Sigma' \models A(\mathbf{x})$, where Σ' is $\Sigma \cup \{\sigma_1,...\sigma_n\}$.

Σ' is a set of definite clauses, and we will need the familiar characterization of its set of atomic logical consequences: an atomic formula p is a logical consequence of Σ' if

and only if it either has rank 0, i.e. belongs to $\{\sigma_1,...\sigma_n\}$, or has rank n+1 for some n, i.e. there is a substitution instance G&B⊃p of some clause in Σ such that every formula in G,B has rank n or smaller. This characterization is more familiar from the literature with constants instead of variables, but this is immaterial. (Another immaterial circumstance is that the set of logical consequences of Σ' is decidable, since there are no function symbols involved.) For a conjunction A of atoms, we will say that ∃**z**A has lower rank than p if there is a substitution instance A[**u**/**z**] such that every atom in the conjunction has lower rank than p.

We will also use the ∃-constructive property of definite clauses, which tells us that if $\Sigma' \models \exists \mathbf{z} A(\mathbf{x},\mathbf{z})$, there is some substitution instance A(**x**,**w**) such that $\Sigma' \models A(\mathbf{x},\mathbf{w})$. In particular, if A is a conjunction of constraint atoms, the atoms in A(**x**,**w**) are a subset of $\{\sigma_1,...\sigma_n\}$. Finally, defining the completion Σ* of the program as before, it is easily shown by essentially the usual argument that Σ*∪**TC** is consistent if **TC** is, and every computation from a program containing only wait-clauses is trivially logical, by the soundness theorem.

We need some variants of concepts defined in §9. Suppose G'%A is a branch in a choice-box G occurring in some configuration in Π. The choice-box was created by local forking applied to a program atom, the *parent atom* of G. There is a sequence

$$\mathbf{and}(A')_W\%A,..G'\%A$$

where $\mathbf{and}(A')_W\%A$ is a guarded goal created with G, and each guarded goal in the sequence is obtained from its immediate predecessor either by an application of a rewrite rule or as one of the two goals obtained by application of guard distribution to the choice-box. $\mathbf{and}(A')_W\%A$ will be referred to as the *parent branch* of G'%A.

A branch G'%A in a choice-box G is *valid* if $\Sigma' \models G'^*$ and the parent atom p of G and the parent branch $\mathbf{and}(A')_W\%A$ of G'%A satisfy the following two conditions: $\Sigma' \models p$, and ∃W(A'&A) has lower rank than p.

The or-components of the configurations in Π will be referred to as *top-level and-boxes*. A top-level and-box $\mathbf{and}(G_1,..G_n)_W$ is *valid* if there is a substitution for the variables in W such that every resulting atomic G_i is a logical consequence of Σ' and every choice-box G_i has a valid branch.

We prove that for every valid and-box G, the partial subcomputation beginning with G contains a solution $D(\mathbf{x},\mathbf{z})_w$ such that $\models \sigma(\mathbf{x},\mathbf{y}) \supset \exists \mathbf{z} D(\mathbf{x},\mathbf{z})$. The proof is by induction on the complexity of G, where a less complex and-box is one obtained by replacing a

choice-box G with a sequence C,B of atoms, where $\exists W(C\&B)$ has lower rank than the parent atom of G.

Now if every goal in G is a constraint, G itself is the sought solution $D(\mathbf{x},\mathbf{z})_W$. So suppose this is not the case. We consider the first step $G' \Rightarrow G''$ which is applied at top level, i.e. to the configuration as a whole in the partial subcomputation beginning with G. First note that this cannot be environment synchronization or failure propagation in view of the assumption that G is a valid and-box. Also, it follows from our assumptions that there must be such a top-level step. For suppose otherwise. Then, when every guard computation has ceased, we have a stable top-level and-box which must contain some candidate for non-deterministic promotion. So, by the assumption concerning the conditions for non-deterministic promotion to apply, there must be some possible step, applicable either to a proper subgoal, in which case the guard computations have not been completed after all, or to the whole configuration, contradicting the completeness of the computation.

First we need to verify that G' is a valid and-box. This is so because every choice-box created in the course of the computation leading from G to G' will have a valid branch. If every solution of the corresponding guard $G^\dagger$ has been eliminated in the course of the computation, it follows from the main lemma of §9 that $\Sigma^* \cup \mathbf{TC} \models \neg G^\dagger$, which together with the validity in Σ' of some substitution instance of $G^\dagger$ and the consistency of $\Sigma^* \cup \mathbf{TC}$ implies $\mathbf{TC} \models \neg\exists \mathbf{x} \exists \mathbf{y} \sigma$.

So suppose the step $G' \Rightarrow G''$ is one of non-deterministic promotion:

$$\mathbf{and}(T_1,\mathbf{choice}(R,C_U{:}B,S),T_2)_{U'} \Rightarrow$$
$$\mathbf{or}(\mathbf{and}(T_1,\mathbf{choice}(R,S),T_2)_{U'},\mathbf{and}(T_1,C,B,T_2)_{U\cup U'})$$

If $C_U{:}B$ is a valid branch, we apply the induction hypothesis to the valid box $\mathbf{and}(T_1,C,B,T_2)_{U\cup U'}$. Otherwise $\mathbf{and}(T_1,\mathbf{choice}(R,S),T_2)_{U'}$ must be valid, and we apply the argument to the partial subcomputation beginning with that box. After a finite number of top-level applications of non-deterministic promotion we must come to a step in which we either apply non-deterministic promotion using a valid branch $C_U{:}B$, or else an application of deterministic promotion, in which case the activated branch must be valid. In either case we can apply the induction hypothesis. This argument also covers the case of deterministic promotion.

References

[Co] A.Colmerauer, Equations and Inequations on Finite and Infinite Trees, *FGCS '84 Proceedings.*

[HaJa] Seif Haridi & Sverker Janson, Kernel Andorra Prolog and its Computation Model, in *Proceedings of the International Conference on Logic Programming,* Jerusalem, MIT Press 1990.

[JaHa] Sverker Janson & Seif Haridi, Programming Paradigms of the Andorra Kernel Language. ICLP '91 Proceedings.

[Ma] Michael J.Maher, Logic Semantics for a Class of Committed-Choice Programs, in: *Proceedings of the Fourth International Conference on Logic Programming,* Melbourne, MIT Press, 1987.

[Sa] Vijay Saraswat, *Concurrent Constraint Programming Languages,* PhD dissertation, Carnegie-Mellon University, 1989.

[Wa1] David H.D. Warren, *The Andorra Principle,* presented at Gigalips workshop, 1987.

[Wa2] David H.D. Warren, *The Extended Andorra Model with Implicit Control,* presented at a Parallel Logic Programming workshop in Eliat, 1990.