Structural Operational Semantics for Kernel Andorra Prolog *

Seif Haridi[1] and Catuscia Palamidessi[1,2]

[1]Swedish Institute of Computer Science,
Box 1263, S - 164 28 KISTA, Sweden

[2]Centre for Mathematics and Computer Science,
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands
email: katuscia@cwi.nl
and
Department of Computer Science, Utrecht University
P.O. Box 80089 3508 TB Utrecht, The Netherlands

Abstract

Kernel Andorra Prolog is a framework for nondeterministic concurrent constraint logic programming languages. Many languages, such as Prolog, GHC, Parlog, and Atomic Herbrand, can be seen as instances of this framework, by adding specific constraint systems and constraint operations, and optionally by imposing further restrictions on the language and the control of the computation model.

We systematically revisit the description in Haridi and Janson [HJ90], adding the formal machinery which is necessary in order to completely formalize the control of the computation model. To this we add a formal description of the transformational semantics of Kernel Andorra Prolog. The semantics of Kernel Andorra Prolog is a set of *or-trees* which also captures infinite computations.

1 Introduction

Kernel Andorra Prolog is language framework that is specifically designed to combine the programming paradigms of Prolog and committed choice languages [HJ91], allowing fully general combinations. The proposed family of languages are guarded definite clause languages, with deep guards, and three guard operators (wait, cut, and commit). In general, the machinery of deep guards is necessary in nondeterministic languages, for selecting a single solution, or collecting all solutions for a given goal. In particular the generalization to deep guards is essential to achieve the goal of simultaneously subsuming Prolog and exploiting independent and dependent parallelism. Deep guards can

*The visit at SICS of Catuscia Palamidessi, during which this work was carried out, has been supported by the project Andorra

also be used to encapsulate nondeterministic transformational parts of a program while maintaining a reactive indeterministic computation at an outer level.

The computation model of Kernel Andorra Prolog (KAP) is a generalization of the Andorra Model for pure definite clauses [War87, HB88]. The Andorra Model exploits implicit and-parallelism in the execution of definite clauses. The generalized model features a carefully controlled nondeterminism, which is available uniformly in a computation.

The framework is parameterized with the constraint system used, and the chosen set of constraint operations with their respective activation conditions. Also, in some specific cases, sequential ordering between goals is necessary to achieve the desired synchronisation effects.

The exposition depends partly on an intuitive understanding of KAP as given in [HJ90]. However we have tried to make the paper as self containted as possible within the size limit.

2 The Basic Andorra Model

The Andorra model is defined for pure Horn clauses. It gives priority to deterministic computation over nondeterministic computation, as nondeterministic steps are likely to multiply work.

The Andorra model divides a computation into *deterministic* and *nondeterministic phases*. First, all atomic goals for which it is known that at most one clause would succeed are reduced using a single clause during the deterministic phase. (These goals can be reduced in and-parallel.) Then, when no such goal is left, some goal is chosen for which all clauses are tried; this is called the nondeterministic phase. The computation then proceeds with a deterministic phase on each or-branch.

The key concept here is the notion of determinacy. An atomic goal is said to be *deterministic* when there is at most one candidate clause that would succeed for the goal. As soon as it is known that an atomic goal has become deterministic, the goal can either be reduced by a single clause, or fail, if it was known that no clause would apply. It is not considered to be an error if the mechanism for detecting the determinacy of goals fails to detect that a goal is deterministic. In general, nothing less than complete execution will establish this property.

The Andorra model has a number of interesting consequences.

Firstly, the Andorra model allows deterministic goals to be run in and-parallel, extracting implicit and-parallelism from the program.

Secondly, the notion of determinacy in the Andorra model gives a reasonably strong form of *synchronization*. As long as a goal is able to produce data deterministically, no consumer of this data is allowed to run ahead (if it does not know what to consume). This allows specification of concurrent processes.

Thirdly, the Andorra model reduces the search space by executing the deterministic goals first. Goals can fail early, and the constraints produced by a reduction can reduce the number of alternatives for other goals. This has been proved to be very relevant for the coding of constraint satisfaction problems [Kor89, Sar89b, BG89, HB88, Yan89].

The Andorra model in items:

- An atomic goal fails if it is known that no clause would succeed for the goal.
- An atomic goal can be reduced using a single clause when it is known that all other clauses would necessarily fail for the goal.
- When no goal is known to be deterministic, all clauses in its definition are tried for some goal.

3 The Extended Computation Model

The extended computation model takes advantage of the principles underlying the Andorra model to control nondeterminism in a "deep" concurrent language. Atomic goals may start in and-parallel, performing local computations, by an operation called *local forking.* Local computations are recursive Andorra computations that are logically independent. A local computation may receive information from its environment, but cannot communicate its results to the uncle goals until a promotion operation is performed. In KAP each clause is divided into a guard and a body. Local computations are performed by the goals of the guards of the candidate clauses. Promotion takes place when one or more guard executions terminate (depending on the guard operator). Promotion has three main forms. *Determinate promotion* is performed when a local computation reduces to a single branch. This generalizes the Andorra determinacy test. *Nondeterminate promotion* is performed when a local computation reduces to several branches; this introduces nondeterminism by creating an or-tree and distributing the uncle goals into each branch of the local computation. *indeterminate* promotion is performed when the guard operator is a pruning operator, selecting only one successful branch of the local computation.

Now, the language and the configurations are defined. Then, the transition rules that start guard execution, perform commit, etc, are described.

3.1 Kernel Andorra Prolog (KAP): the language

Let *Var* be an infinite set of variables, with typical elements $x, y, z, ...$, Let *Con* be a set of n-adic data constructors, with typical elements $a, b, c...$ (constant symbols) and $f, g, h, ...$ (function symbols). Terms (*Term*), $t, u, ...$, atoms (*Atom*), $A, B, ...$ and substitutions (*Subst*) are defined as usual. Elementary atoms $H, K, ...$, are atoms of the form $p(\bar{x})$, where p is a predicate and $\bar{x}$ is a tuple of distinct variables. A clause is an object of the form

$$H \text{ :- } \mathbf{choice}_{\%}(T_1, \ldots, T_n)$$

such that the T_i's are simple guarded goals (see below). The variables in H are called *parameters.* The symbol % stands for a guard operator. There are three possible guard operators, namely '!', '|' or ':'. A *KAP program* is a set of clauses. Each clause represents the definition of one predicate. We assume that each predicate occurring in the program is defined by exactly one clause of the program.

⟨*program*⟩	::=	⟨*set of clauses*⟩
⟨*clause*⟩	::=	⟨*head*⟩ :- ⟨*simple choice box*⟩
⟨*head*⟩	::=	⟨*elementary atom*⟩
⟨*simple choice box*⟩	::=	**choice**⟨*guard operator*⟩(⟨*sequence of simple guarded goals*⟩)
⟨*simple guarded goal*⟩	::=	[⟨*simple guard*⟩]⟨*body*⟩
⟨*simple guard*⟩	::=	∃⟨*set of variables*⟩.**and**(⟨*sequence of atomic goals*⟩;**true**)
⟨*body*⟩	::=	⟨*sequence of atomic goals*⟩
⟨*atomic goal*⟩	::=	⟨*elementary atom*⟩ | ⟨*constraint operation*⟩
⟨*guard operator*⟩	::=	':' | '!' | '|'

For technical reasons, we assume that in a simple choice box

$$\mathbf{choice}_{\%}([\exists Z_1.P_1]\bar{B}_1, \ldots, [\exists Z_n.P_n]\bar{B}_n)$$

the sets of variables $Z_1, \ldots, Z_n$ are pairwise disjoint. Moreover, we assume that $Z_1, \ldots, Z_n$ contain all local variables of a clause, namely those variables occurring in the choice box and not in the head. A program is considered to be closed under all possible variable renaming.

The language is parametrized with a *constraint theory*. The set of formulas of this theory, with typical elements $\vartheta, \sigma, \psi, \ldots$, will be denoted by *Constraints*. The existential closure of ϑ is denoted by $\exists(\vartheta)$. We say that ϑ is *consistent* iff $\models \exists(\vartheta)$, where the symbol $\models$ stands for logical validity with respect to the given theory. We assume that the theory is *decidable*, therefore ϑ is *inconsistent* iff $\models \neg\vartheta$. We say that ϑ *entails* σ iff $\models \vartheta \supset \sigma$.

Unification, and the like, are performed by primitive constraint operations. The notation $op(\psi)$ denotes a primitive operation *op* applied to the constraint ψ. A constraint operation may suspend until its *activation condition* is satisfied. Some primitive operations are described later (see section 6.1).

4 The transition system for Kernel Andorra Prolog

We define the operational semantics of KAP via a transition system. This system is essentially based on the rewriting system defined in [HJ90] for describing the computational model of Andorra Prolog.

The set of configurations is defined by the following grammar

$$
\begin{array}{lcl}
\langle goal \rangle & ::= & \langle or\ box \rangle \mid \langle and\ box \rangle \\
\langle or\ box \rangle & ::= & \mathbf{or}(\langle goal \rangle, \langle goal \rangle) \\
\langle and\ box \rangle & ::= & \exists \langle set\ of\ variables \rangle . \\
 & & \mathbf{and}(\langle sequence\ of\ local\ goals \rangle; \langle constraint \rangle) \\
\langle local\ goal \rangle & ::= & \langle atomic\ goal \rangle \mid \langle choice\ box \rangle \\
\langle choice\ box \rangle & ::= & \mathbf{choice}\langle guard\ operator \rangle(\langle sequence\ of\ guarded\ goals \rangle) \\
\langle guarded\ goal \rangle & ::= & [\langle goal \rangle]\langle body \rangle
\end{array}
$$

We use *Goal*, *LocalGoal*,... etc, to denote the sets generated by $\langle goal \rangle$, $\langle local\ goal \rangle$, ... etc. The symbols A, B, C stand for local goals, P, Q, R stand for goals, and S, T stand for guarded goals. Moreover, *GeneralGoal*, with typical element g, will denote the set $Goal \cup LocalGoal$.

The set of *sequences of existentially quantified constraints Seq*, with typical element s, is the smallest set such that

- $\lambda \in Seq$ (the empty sequence)
- $\forall \vartheta \in Constraints\ \forall s \in Seq\ \forall X \subseteq Var\ \exists X.\vartheta \diamond s \in Seq$

The symbol $\diamond$ stands for sequence concatenation.

The logical meaning Θ_s of a sequence s of existentially quantified constraints is defined as follows:

- $\Theta_\lambda = \mathbf{true}$
- $\Theta_{\exists X.\vartheta \diamond s} = \exists X.(\vartheta \wedge \Theta_s)$

Sometimes we will need the conjuction of constraints with all variables free; for this we use $\hat{\Theta}_s$ defined as follows:

- $\hat{\Theta}_\lambda = \mathbf{true}$
- $\hat{\Theta}_{\exists X.\vartheta \diamond s} = \vartheta \wedge \hat{\Theta}_s$

The notion of consistency and entailment is extended to sequences in the following way. A sequence s is *consistent* iff $\models \exists(\Theta_s)$, and it is *inconsistent* iff $\models \neg\Theta_s$. s *entails* $\exists X.\sigma$ (or, σ does not restrict the environment outside X) iff $\models \hat{\Theta}_s \supset \exists X.\sigma$ (or, equivalently, iff $\models \Theta_s \supset \Theta_{s \circ \exists X.\sigma}$).

The transition system is a pair $\langle Conf, \rightarrow \rangle$, where $Conf = GeneralGoal \cup \{\mathbf{fail}, \mathbf{deadlock}\}$, and $\rightarrow$ is a class of transition relations on $Conf$

$$\{\ell \rightarrow_s^m : \ell \in \{o, u\}, m \in \{F, G\}, s \in Seq\}$$

o, u stand for *ordered* and *unordered* respectively, and refer to the context of the configuration. A transition $c\ \ell \rightarrow_s^m c'$ will be represented as $\ell \vdash c \rightarrow_s^m c'$. When the context information is irrelevant to a particular transition rule, we omit the symbol $\ell \vdash$.

F, G stand for *guess free* and *not guess free* respectively. The subscript s indicates the environment that the configuration is assumed to have when the transition takes place. Namely, $c \rightarrow_s^m c'$ means that it is possible to make a transition from c to c' in the mode m when the environment of c is s.

In the following, we assume the program W to be fixed. In an and-box $\exists X.\mathbf{and}(\bar{A};\vartheta)$, X represents the set of immediate local variables of the box. This information is necessary to deal with execution and suspension of the constraint operations.

A tuple of goals is represented by a bar, so, for instance $\bar{S}, \bar{T}$ stand for tuples of guarded goals, etc. The empty and-box $\exists X.\mathbf{and}(;\vartheta)$ is simply represented by $\exists X.\vartheta$.

We introduce the notion of *variables local to a general goal*, $var(g)$.

- $var(p(x_1, \ldots, x_n)) = \{x_1, \ldots, x_n\}$
- $var(op(\psi)) = var(\psi)$
- $var(\mathbf{or}(P, Q)) = var(P) \cup var(Q)$
- $var(\exists X.\ \mathbf{and}(A_1, \ldots, A_n;\vartheta)) = X \cup var(A_1) \cup \ldots \cup var(A_n) \cup var(\vartheta)$
- $var(\mathbf{choice}_{\%}(S_1, \ldots, S_n)) = var(S_1) \cup \ldots \cup var(S_n)$
- $var([P]\bar{B}) = var(P)$

Computation Rules

Local Forking

An elementary atom A can be transformed into the choice-box associated with the definition of the predicate of A. The parameters are replaced by the arguments of A. This is expressed by the following rule

If $A \equiv p(y_1, \ldots, y_n)$ and $p(x_1, \ldots, x_n)$:- B belongs to W, where $var(B) \cap \{y_1, \ldots, y_n\} = \emptyset$, then

$$A \rightarrow_s^F B\alpha$$

where $\alpha = \{x_1/y_1, \ldots, x_n/y_n\}$ and $B\alpha$ is the application of the substitution α to B.

The structural rules of the transition system will guarantee that variables introduced by the local forking are different from the variables of the global configuration.

Primitive Constraint Rules

Constraint operations are the only ones that can modify the environment. There are three transition rules corresponding to successful execution, failure and suspension.

Some constraint operations, corresponding to the actions of existing languages like Prolog, GHC, Parlog, and Atomic Herbrand, are described in section 6.1.

$$\exists X.\mathbf{and}(\bar{A}, op(\psi), \bar{B}; \sigma) \rightarrow_s^F \exists X.\mathbf{and}(\bar{A}, \bar{B}; \sigma \wedge \psi) \quad \textit{if} \quad \begin{array}{l} activ(op, X, \psi, \sigma, s) \\ \models \exists(\Theta_{s \circ \exists X.\sigma \wedge \psi}) \end{array}$$

i.e. if the activation condition of $op(\psi)$ holds (with respect to X, σ and s) and $\exists X.\sigma \wedge \psi$ is consistent with its environment. The activation condition will depend both on the specific constraint operation and s.

Determinate Promotion

If after the completion of the guard execution only one guarded goal is left within a choice box, then it can be extracted, according to the following rule

$$\exists X.\mathbf{and}(\bar{A}, \mathbf{choice}_{\%}([\exists Y.\psi]\bar{B}), \bar{A}'; \sigma) \rightarrow_s^F \exists X \cup Y.\mathbf{and}(\bar{A}, \bar{B}, \bar{A}'; (\sigma \wedge \psi))$$

$$\textit{if} \quad \models \exists(\sigma \wedge \psi)$$

Quiet Indeterministic Promotion

A guard execution is quiet if it results in an (empty) and-box, whose constraint does not restrict the environment outside the local variables of the box.

Cut

After a successful guard execution of one branch, the cut operator prunes all the branches to the right.

$$\mathbf{choice}_!(\bar{S}, [\exists Y.\sigma]\bar{B}, \bar{T}) \rightarrow_s^F \mathbf{choice}_!(\bar{S}, [\exists Y.\sigma]\bar{B}) \quad \textit{if} \quad \models \hat{\Theta}_s \supset \exists Y.\sigma.$$

Commit

After a successful guard execution of one branch, the commit operator prunes all the other branches.

$$\mathbf{choice}_|(\bar{S}, [\exists Y.\sigma]\bar{B}, \bar{T}) \rightarrow_s^F \mathbf{choice}_|([\exists Y.\sigma]\bar{B}) \quad \textit{if} \quad \models \hat{\Theta}_s \supset \exists Y.\sigma.$$

Or Reduction

The or boxes within a choice box are eliminated according to the following rule

$$\mathbf{choice}_{\%}(\bar{S}, [\mathbf{or}(P, Q)]\bar{B}, \bar{T}) \rightarrow_s^F \mathbf{choice}_{\%}(\bar{S}, [P]\bar{B}, [Q]\bar{B}, \bar{T})$$

Now we will describe the transitions of guessing rules, marked by the G flag. The use of these rules will later be restricted by the control principles of Kernel Andorra Prolog.

Nondeterministic Promotion

A goal is in an *ordered context* if the closest surrounding pruning choice is a cut choice, otherwise it is in an *unordered context.*

Ordered Context

In an ordered context, after the successful execution of the guard, the leftmost branch of a nondeterministic choice within an and-box can be promoted if the computed constraint is consistent with the one of the and-box.

$$o \vdash \exists X.\mathbf{and}(\bar{A}, \mathbf{choice}_!([\exists Y.\psi]\bar{B}, \bar{T}), \bar{A}';\sigma) \rightarrow_s^G \mathbf{or}(\ \exists(X \cup Y).\mathbf{and}(\bar{A}, \bar{B}, \bar{A}';(\sigma \wedge \psi)),\ \exists X.\mathbf{and}(\bar{A}, \mathbf{choice}_!(\bar{T}), \bar{A}';\sigma))$$

if $\models \exists(\sigma \wedge \psi)$.

Unordered Context

In an unordered context, after the successful execution of the guard, any branch of a nondeterministic choice within an and-box can be promoted if the computed constraint is consistent with the one of the and box.

$$u \vdash \exists X.\mathbf{and}(\bar{A}, \mathbf{choice}_!(\bar{S}, [\exists Y.\psi]\bar{B}, \bar{T}), \bar{A}';\sigma) \rightarrow_s^G \mathbf{or}(\ \exists(X \cup Y).\mathbf{and}(\bar{A}, \bar{B}, \bar{A}';(\sigma \wedge \psi)),\ \exists X.\mathbf{and}(\bar{A}, \mathbf{choice}_!(\bar{S}, \bar{T}), \bar{A}';\sigma))$$

if $\models \exists(\sigma \wedge \psi)$.

Noisy Indeterministic Promotion

Cut

When the guard execution of the leftmost branch left in a choice-box is completed successfully and the computed constraint is consistent with the constraint of the closest sourrounding and-box, then the cut operator prunes all the other branches (to the right) and the computed constraint is made public.

$$\exists X.\mathbf{and}(\bar{A}, \mathbf{choice}_!([\exists Y.\psi]\bar{B}, \bar{T}), \bar{A}';\sigma) \rightarrow_s^G \exists(X \cup Y).\mathbf{and}(\bar{A}, \bar{B}, \bar{A}';(\sigma \wedge \psi))$$

if $\models \exists(\sigma \wedge \psi)$.

Commit

When the guard execution of one branch in a choice-box is completed successfully and the computed constraint is consistent with the constraint of the closest sourrounding and-box, then the commit operator prunes all the other branches and the computed constraint is made public.

$$\exists X.\mathbf{and}(\bar{A}, \mathbf{choice}_!(\bar{S}, [\exists Y.\psi]\bar{B}, \bar{T}), \bar{A}';\sigma) \rightarrow_s^G \exists X \cup Y.\mathbf{and}(\bar{A}, \bar{B}, \bar{A}';(\sigma \wedge \psi))$$

if $\models \exists(\sigma \wedge \psi)$.

Structural Rules

The following rules allow us to derive the transitions of the configurations depending on the transitions that can be made by the components of the configurations. The rule for and boxes will be restrained by the condition that the new local variables introduced in a transition of a subgoal must be disjoint

with the variables of the other subgoals. This will ensure that all the local variables of different and components are always disjoint, thus evoiding clashes of variables.

Or boxes

Any transition made by a goal inside an or-box is propagated to the parent box.

$$\frac{\ell \vdash Q \rightarrow_s^m Q'}{\begin{array}{l}\ell \vdash \mathbf{or}(P,Q) \rightarrow_s^m \mathbf{or}(P,Q') \\ \ell \vdash \mathbf{or}(Q,P) \rightarrow_s^m \mathbf{or}(Q',P)\end{array}}$$

Note that, since Q' is a metavariable on *Goals*, $Q' \neq$ **fail**, **deadlock**.

Choice boxes

Any transition made by a goal inside a choice box is propagated to the parent box. An ordered transition can take place if the closest surrounding pruning choice box is a cut choice box. An unordered transition can take place if the closest surrounding pruning choice box is a commit choice box.

To formalize this notion, we introduce the following function $\mathcal{OU} : \{o,u\} \times \{:,!,|\} \rightarrow \{o,u\}$ (ordered-unordered) that filters the context informations out of the guard operator.

- $\mathcal{OU}(\ell,:) = \ell$
- $\mathcal{OU}(\ell,!) = o$
- $\mathcal{OU}(\ell,|) = u$

$$\frac{\ell \vdash Q \rightarrow_s^m Q'}{\ell' \vdash \mathbf{choice}_{\%}(\bar{S},[Q]\bar{B},\bar{T}) \rightarrow_s^m \mathbf{choice}_{\%}(\bar{S},[Q']\bar{B},\bar{T})} \quad \ell = \mathcal{OU}(\ell',\%)$$

And boxes

Any transition made by a goal inside an and box, with the assumption that the external environment is s, generates a transition for the parent box, with a weaker assumption s', such that s is the result of appending the sequence s' and the constraint (existentially quantified with respect to the local variables) of the and box. Intuitively, this model the fact that the environment of the goal consists of the environment and the constraint of the parent and box.

$$\frac{\ell \vdash B \rightarrow_s^m B'}{\ell \vdash \exists X.\mathbf{and}(\bar{A},B,\bar{C};\psi) \rightarrow_s^m \exists X.\mathbf{and}(\bar{A},B',\bar{C};\psi)} \quad \begin{array}{l}(var(B') \setminus var(B)) \cap \\ var(\exists X.\mathbf{and}(\bar{A},B,\bar{C};\psi)) = \emptyset \\ s = s' \diamond \exists X.\psi\end{array}$$

Next we specify the transition rules for *failure* and *suspension*. Having explicitely these rules in the operational model of the language allows to gain in efficiency (the detection of *failure* allows to prune the failing choice branches).

Failure Rules

The following rules describe the transitions that bring to failure.

Constraint operations

Any primitive operation *op* will fail whenever the constraint is not consistent with the environment

$$op(\psi) \rightarrow^F_{s \diamond \exists X.\sigma} \mathbf{fail} \quad \textit{if} \quad \models \neg\Theta_{s \diamond \exists X.\sigma \wedge \psi}$$

And boxes

An and box fails when the constraint is inconsistent with its global environment. Notice that this is the only rule (apart from the rules on primitives) that depends upon the constraints of the environment in the transition relation.

$$\exists X.\mathbf{and}(\bar{A};\sigma) \rightarrow^F_s \mathbf{fail} \quad \textit{if} \quad \models \neg\Theta_{s \diamond \exists X.\sigma}$$

Choice boxes

A choice box fails when there are no alternatives left.

$$\mathbf{choice}_{\%}() \rightarrow^F_s \mathbf{fail}$$

Structural Rules for Failure

The following rules describe the propagation of failing transitions from inner goals to the external configurations.

Choice boxes

Choice boxes simply eliminate failing branches.

$$\frac{Q \rightarrow^F_s \mathbf{fail}}{\mathbf{choice}_{\%}(\bar{S}, [Q]\bar{B}, \bar{T}) \rightarrow^F_s \mathbf{choice}_{\%}(\bar{S}, \bar{T})}$$

And boxes

An and box fails whenever one of the local goals fails. Again, the environment condition of the internal transition is weakened in the external transition

$$\frac{B \rightarrow^F_s \mathbf{fail}}{\exists X.\mathbf{and}(\bar{A}, B, \bar{C};\psi) \rightarrow^F_{s'} \mathbf{fail}} \quad s = s' \diamond \exists X.\psi$$

Suspension Rules

Here we describe the rules that bring a configuration to be suspended. We only describe *global suspension*, namely the *deadlock* of the whole goal.

Constraint operations

A constraint operation may suspend when the activation condition of the operation is not satisfied and the constraint is consistent with the environment (otherwise it would fail).

$$op(\psi) \rightarrow^F_{s \diamond \exists X.\sigma} \mathbf{deadlock} \quad \textit{if} \quad \begin{array}{l} \neg activ(op, X, \psi, \sigma, s), \\ \models \exists\Theta_{s \diamond \exists X.\sigma \wedge \psi} \end{array}$$

Structural Rules for Deadlock

The following rules describe the propagation of deadlock from the inner configurations to the outer ones.

Choice boxes

Commit and Wait

If the the guard operator is | or :, then a choice box deadlocks whenever all the branches get deadlocked.

$$\frac{P_1 \rightarrow_s^F \mathbf{deadlock}, \ldots, P_n \rightarrow_s^F \mathbf{deadlock}}{\mathbf{choice}_{\%}([P_1]\bar{B}_1, \ldots, [P_n]\bar{B}_n) \rightarrow_s^F \mathbf{deadlock}} \quad \% \in \{|, :\}$$

Cut

If the the guard operator is !, then a choice box deadlocks whenever the first branch gets deadlocked.

$$\frac{P \rightarrow_s^F \mathbf{deadlock}}{\mathbf{choice}_!([P]\bar{B}, \ldots, \bar{S}) \rightarrow_s^F \mathbf{deadlock}}$$

And boxes

An and box deadlocks whenever all the internal local goals deadlock. Observe that the deadlock of one local goal does not cause the deadlock of the whole and box since it can be resumed after the execution of some other local goals. This can be modeled by the following rule.

$$\frac{B_1 \rightarrow_s^F \mathbf{deadlock}, \ldots, B_n \rightarrow_s^F \mathbf{deadlock}}{\exists X.\mathbf{and}(B_1, \ldots, B_n; \psi) \rightarrow_{s'}^F \mathbf{deadlock}} \quad s = s' \diamond \exists X.\psi$$

We don't have failure and suspension rules for the or boxes. Internally or boxes are collapsed into choice boxes. For external or boxes, we want to preserve the shape of the tree. This issue will be exposed in details in the section on operational semantics.

5 Control of the computation model

A sequence of applications of the transition rules described in the previous section constitute an unrestricted derivation or computation of the extended Andorra computation. KAP computations are restricted. The control of KAP has been described informally and motivated in [HJ90]. We will just give a brief review to aid the understanding of the following semantic description. KAP always prefer deterministic steps over nondeterministic and noisy indeterministic steps. The F-marked rules: local forking, deterministic promotion, quiet pruning, constraint rules, etc. are called *guess-free* rules. Nondeterministic promotion and noisy pruning are called *guessing* rules.

Central to the control of the model is the notion of stability of and-boxes. Stable boxes are boxes that cannot be affected by computations in its surrounding environment regardless of the context in which it is running. An and-box is called *stable* if no guess-free rules are applicable to or within the box, and no constraint or constraint operation occurring in the box imposes new constraints on variables that are external to the box.

A context is a general goal "with a hole".

Definition 5.1 *The set Context, with typical element $C[\,]$, is the smallest set such that*

1. $[\,] \in \mathit{Context}$
2. *if* $C[\,] \in \mathit{Context}$ *then*
   - $\mathbf{or}(P, C[\,])$, $\mathbf{or}(C[\,], P) \in \mathit{Context}$
   - $\exists X.\ \mathbf{and}(\bar{A}, C[\,], \bar{B}; \vartheta) \in \mathit{Context}$
   - $\mathbf{choice}_{\%}(\bar{S}, [C[\,]]\bar{B}, \bar{T}) \in \mathit{Context}$

Whenever the expression $C[g]$ is legal it will denote the general goal obtained by "filling the hole" of $C[\,]$ with g. Note that g is a general goal.

The next definition formalizes the notion of total constraint of a global goal, which gives the set of all the constraint sequences in all the subgoals.

Definition 5.2 *The function* $tc : \mathit{GlobalGoal} \to \mathcal{P}(\mathit{Seq})$ *(the set of all sets of sequences) is defined as follows*

- $A \in \mathit{Atom} \Rightarrow tc(A) = \{\lambda\}$
- $op(\sigma) \in \mathit{Primitives} \Rightarrow tc(op(\sigma)) = \{\sigma\}$
- $S_i = [P_i]\bar{B}_i \Rightarrow tc(\mathbf{choice}_{\%}(S_1, \ldots, S_n)) = tc(P_1) \cup \ldots \cup tc(P_n)$
- $tc(\mathbf{or}(P, Q)) = tc(P) \cup tc(Q)$
- $tc(\exists X.\ \mathbf{and}(A_1, \ldots, A_n; \sigma)) = \exists X.\sigma \diamond (tc(A_1) \cup \ldots \cup tc(A_n))$

where the operation $\diamond$ *is extended on sets.*

We now introduce the notion of *stability* of and-boxes (with respect to an environment).

$$\mathit{stable}(A, s) \text{ if } \nexists c \in \mathit{Conf}\ [A \to_s^F c] \text{ and } \forall s' \in tc(A)\ [\models \hat{\Theta}_s \supset \Theta_{s'}]$$

The next definition formalizes the notion of *environment*: the environment of a configuration is the environment of the closest surrounding and-box plus its constraint. We define the environment as a function *env* from contexts to sequences of constraints in such a way that $env(C[\,])$ is the environment of the "hole" of $C[\,]$, i.e. the constraint seen by a legal goal g in $C[g]$.

Definition 5.3 (environment of a context) *The function* $env : \mathit{Context} \to \mathit{Seq}$ *is defined as follows:*

- $env([\,]) = \lambda$
- $env(\mathbf{or}(P, C[\,])) = env(\mathbf{or}(C[\,], P)) = env(C[\,])$
- $env(\exists X.\mathbf{and}(\bar{A}, C[\,], \bar{B}; \sigma)) = \exists X.\sigma \diamond env(C[\,])$
- $env(\mathbf{choice}_{\%}(\bar{S}, [C[\,]]\bar{B}, \bar{T})) = env(C[\,])$.

The following definition extends the function $\mathcal{OU}$ so to filter the informations of ordered and unordered out of generic contexts.

Definition 5.4

- $\mathcal{OU}_{\text{ext}}(\ell, [\,]) = \ell$
- $\mathcal{OU}_{\text{ext}}(\ell, \mathbf{or}(P, C[\,])) = \mathcal{OU}_{\text{ext}}(\ell, \mathbf{or}(C[\,], P)) = \mathcal{OU}_{\text{ext}}(\ell, C[\,])$
- $\mathcal{OU}_{\text{ext}}(\ell, \exists X.\mathbf{and}(\bar{A}, C[\,], \bar{B}; \sigma)) = \mathcal{OU}_{\text{ext}}(\ell, C[\,])$
- $\mathcal{OU}_{\text{ext}}(\ell, \mathbf{choice}_{\%}(\bar{S}, [C[\,]]\bar{B}, \bar{T})) = \mathcal{OU}_{\text{ext}}(\mathcal{OU}(\ell, \%), C[\,])$

The restricted Andorra Computation model is defined by the following transition system (based on the previous one). The transition relations are labeled by A and describe *admissible* applications:

- An application of a guess-free rule is always admissible

$$\frac{\ell \vdash g \rightarrow^{F}_{s} g'}{\ell \vdash g \rightarrow^{A}_{s} g'} \quad \textit{where } g' \in \textit{GlobalGoal} \cup \{\mathbf{fail}, \mathbf{deadlock}\}$$

- An application of a guessing rule is admissible iff

  1. it is applied to (or to a subgoal of) a stable and-box, and
  2. there are no admissible applications of guessing rules to proper subgoals of the rewritten box, i.e. it is innermost. Innermost application of G-transition of a goal g is defined as follows:

$$\textit{innermost}(g, \ell\, s) \text{ iff } \forall C[\,] \neq [\,]\, \forall g'\, [g = C[g'] \Rightarrow \nexists c\, (\mathcal{OU}_{\text{ext}}(\ell, C[\,]) \vdash g' \rightarrow^{G}_{s \circ env(C[\,])} c)]$$

$$\frac{\ell \vdash g \rightarrow^{G}_{s} g'}{\ell' \vdash C[g] \rightarrow^{A}_{s'} C[g']} \quad \begin{array}{l} \textit{stable}(C[g], s'), \\ \textit{innermost}(g, \ell, s) \\ \ell = \mathcal{OU}_{\text{ext}}(\ell', C[\,]) \\ s = s' \circ env(C[\,]) \end{array}$$

- (Compositional Rule) An admissible transition of a stable box can be propagated to outer configuarations.

$$\frac{\ell \vdash g \rightarrow^{A}_{s} g'}{\ell' \vdash C[g] \rightarrow^{A}_{s'} C[g']} \quad \begin{array}{l} \ell = \mathcal{OU}_{\text{ext}}(\ell', C[\,]) \\ s = s' \circ env(C[\,]) \end{array}$$

Further restrictions of KAP computations are possible, but we regard such restriction as part of the semantics of the particular user languages based on KAP. For discussions of possible user languages see [HJ90].

6 Operational Semantics

We give now the definition of the operational semantics of AP in set-theoretic terms. Intuitively, the operational semantics of a program is defined by the semantics of the and-boxes that can be run under that program, and the semantics of an and-box consists of the set of all the possible collections of answers that can be obtained by running it. In general, all the answers that are delivered under certain particular indeterministic and nondeterministic choices are collected in an or box. The or box has a tree like structure, with the intermediate nodes labeled by **or** and the leaves labeled by and-boxes. The nonempty and-boxes correspond to computations not terminated yet, and we can define the semantics as the limit of the or trees obtained along a certain computation (transition) chain. This limit can be, in general, an infinite tree, as the computation can deliver an infinite set of answers. This allows us to preserve all the information about the computation in the semantic structure, and it leaves space for further abstractions.

Indeed, different implementations may lead to different notions of *observables*. For instance, we may imagine a situation similar to Prolog, in which the answers are presented in the order they are collected by the usual depth-first strategy. A loop along one branch will cause the unobservability of the answers possibly generated at the right of that branch. To model this, we can abstract from our semantics the sequence of answers obtained by collecting left-to-right the leaves of the tree corresponding to a successfully terminated computation. The sequence ends when we get in correspondence of a nonterminating and-box.

Another possibility is to collect in parallel all the answers generated, without any restriction on the order in which they appear. To model this we can abstract from our semantics the set of the leaves corresponding to a successfully terminated computation.

Definition 6.1 *The set $\mathcal{T}$ (the set of or trees) is the minimal set that satisfies the following conditions:*

- $\bot \in \mathcal{T}$
- $\mathbf{fail}, \mathbf{deadlock} \in \mathcal{T}$
- *if* $\vartheta \in$ *Constraints and* $X \subseteq$ *Var then* $\exists X.\vartheta \in \mathcal{T}$
- *if* $t, t' \in \mathcal{T}$ *then* $\mathbf{or}(t, t') \in \mathcal{T}$

The set $\mathcal{T}$ is ordered as follows

Definition 6.2 *The relation $\leq$ is the minimal ordering relation on $\mathcal{T}$ that satisfies the following conditions*

- $\forall t \in \mathcal{T}.\ \bot \leq t$
- $\forall t, t', u, u' \in \mathcal{T}.(t \leq u \wedge t' \leq u') \supset \mathbf{or}(t, t') \leq \mathbf{or}(u, u')$

Let $(P, \leq)$ be an poset (partially ordered set). A *directed set* in P is a subset D of P such that

$$\forall a, b \in D\ \exists c \in D\ [a \leq c\ \wedge\ b \leq c].$$

An *ideal* S is a directed set which is *downward closed*, i.e. such that

$$\forall a \in S\ [b \leq a \Rightarrow b \in S].$$

The set of ideals of P, ordered by set inclusion, we will denote by $(Id(P), \subseteq)$. It is well known that it is a Complete Partial Order (i.e. it has a minimum, and each non-empty set of elements admits a least upper bound) and it contains a sub-CPO isomorphic to $(P, \leq)$. $(Id(P), \subseteq)$ is called *completion by ideals* of $(P, \leq)$. The elements of the subset isomorphic to $(P, \leq)$ will be denoted by the corresponding elements of P.

Definition 6.3 (The domain of interpretation) *The complete partial order $(\mathcal{T}^\omega, \leq)$ (the domain of finite and infinite or trees) is the completion by ideals of the poset $(\mathcal{T}, \leq)$. The least upper bound of a directed subset $\mathcal{D} \in \mathcal{T}^\omega$ will be denoted by $\sqcup\mathcal{D}$.* □

Definition 6.4 (Operational Semantics) *The operational semantics of a program is a function* $\mathcal{O}$:$\{o,u\} \times Goal \to \mathcal{P}(\mathcal{T}^\omega)$, *where Goal is the set of all the goals and* $\mathcal{P}(\mathcal{T}^\omega)$ *is the set of all the subsets of* $\mathcal{T}^\omega$. $\mathcal{O}$ *is defined as follows.*

$$\mathcal{O}_\ell(P) = \{\sqcup_i \mathcal{O}'(P_i) \colon i \in \{0,1,2,\ldots\}, P \equiv P_0,\ \ell \vdash P_i \to^A_\lambda P_{i+1}\}$$

Note that the environment of the whole configuration is always empty. The argument ℓ *indicates the dependence of the operational semantics upon the global computation beeing considered ordered or unordered.* $\mathcal{O}'$:$Goal \to \mathcal{T}^\omega$ *is defined as follows:*

- $\mathcal{O}'(\mathbf{or}(Q,R)) = \mathbf{or}(\mathcal{O}'(Q), \mathcal{O}'(R))$.
- $\mathcal{O}'(\exists X.\mathbf{and}(\bar{A};\vartheta)) = \begin{cases} \mathbf{fail} & \text{if } \exists X.\mathbf{and}(\bar{A};\vartheta) \to^m_\lambda \mathbf{fail} \\ \mathbf{deadlock} & \text{if } \exists X.\mathbf{and}(\bar{A};\vartheta) \to^m_\lambda \mathbf{deadlock} \\ \exists X.\vartheta & \text{if } \bar{A} \equiv \lambda \\ \bot & \text{otherwise} \end{cases}$

Note that, if $P_0 \to^A_\lambda \ldots P_i \to^A_\lambda \ldots$, then (since P_i ranges over goals, hence it cannot be **fail** or **deadlock**), $\{\mathcal{O}'(P_i)\}_{i\geq 0}$ is a chain. Therefore (since a chain is a particular case of directed set), the definition of $\mathcal{O}$ is correct.

If the constraint system is decidable, then the rules for *failure* and *suspension* cover all the cases in which the computation rules are not applicable, excepting the or-boxes. Namely, a configuration is final only if it is of one of the following forms: **fail**, **deadlock**, an empty and-box, or an or-box containing only final configurations. This means that the semantics of a configuration is always a set maximal objects (possible infinite) in $\mathcal{T}^\omega$ (i.e., the leaves are not labeled by $\bot$).

6.1 Primitive Operations

The primitive operations are the only ones that can modify the environment. The four primitives we describe differ in the level of the environment they are allowed to impose constraint on. In particular, **ask** cannot impose any constraint (its constraint must be completely *entailed* by the environment), whilst $\mathbf{tell}_\omega$ can always do it. Between these two extreme cases, there are $\mathbf{tell}_0$ and $\mathbf{tell}_1$. The first can impose contraints on the local variables of the parent and box, whilst the second can impose contraints on the local variables of the "granparent" and box.

- $activ(\mathbf{ask}, X, \psi, \sigma, s) \equiv \models \hat{\Theta}_{s \diamond \exists X.\sigma} \supset \psi$
- $activ(\mathbf{tell}_0, X, \psi, \sigma, s) \equiv \models \hat{\Theta}_{s \diamond \exists X.\sigma} \supset \exists X.\sigma \wedge \psi$
- $activ(\mathbf{tell}_1, X, \psi, \sigma, \lambda) \equiv true$
- $activ(\mathbf{tell}_1, X, \psi, \sigma, s \diamond \exists Y.\vartheta) \equiv \models \hat{\Theta}_{s \diamond \exists Y.\vartheta \diamond \exists X.\sigma} \supset \exists Y \cup X.(\vartheta \wedge \psi \wedge \sigma)$
- $activ(\mathbf{tell}_\omega, X, \psi, \sigma, s) \equiv true$

7 Related Work

Vijay Saraswat defined a language CP [Sar87] that provides among other things deep guards, an operator called "don't know commit", related to our "wait" operator, and the concept of blocks, which are similar to and-boxes. One of the main differences between CP and our work is our control of *when* to promote nondeterminism. This is also true of Saraswat's thesis [Sar89a]. Also, we emphasize fully interleaved execution in a language with deep guards. However, Kernel Andorra Prolog is definitely a concurrent constraint language.

8 Discussion

We presented a formal transformational semantics for Kernel Andorra Prolog. The semantics is transformational since it desribes the final results of a KAP computation (as a set of abstract trees), and there is no notion of interaction with the environment. A number of issues will be addressed in the near future. Firstly, since KAP is intended as a framework for implementing some user oriented languages, by using the proper constraint operations, we like to prove the correctness of the implementation w.r.t a priori given semantics of the user language.

Another issue is proving the properties of the sublanguages given in [HJ90]. This is partly done in [Fra90].

The semantics given treats and-boxes seen at the outermost level as black boxes until the box either succeeds, fails or suspends. This notion is not sufficient, for some of the sublanguages of KAP. In particular the Reactive Andorra Prolog which encapsulates nondeterminism in pruning guards, needs a more refined notion where actions in an and-box can affect its environment, i.e. some sort of a reactive semantics, and observational equivalence based on it.

Finally we remind the reader, that other more important issues, like efficient implementation, of both sequential and parallel machines, and programming methodology and techniques have the highest priority and much effort is devoted to them.

Acknowledgements

The authors wish to thank Sverker Janson, Vijay Saraswat, Torkel Franzén, D.H.D. Warren and Bill Kornfeld for many valuable comments and suggestions. This work is part ofthe PEPMA ESPRIT Project (P2471), and is supported by the Swedish Board of Technical Development, Televerket, and Ericsson.

References

[BG89] R. Bahgat and S. Gregory. Pandora: Non-deterministic Parallel Logic Programming. In Giorgio Levi and Maurizio Martelli, editors, *Proc. of the Sixth International Conference on Logic Programming*, Series in Logic Programming, pages 471–486, Lisboa, 1989. The MIT Press.

[Fra90] T. Franzén. Formal aspects of Kernel Andorra Prolog. Technical Report R90008, SICS, Sweden, 1990.

[HB88] S. Haridi and P. Brand. Andorra Prolog:, an integration of Prolog and commited choice languages. In *Proc. of the International Conference on Fifth Generation Computer Systems*, pages 745–754, Tokyo, 1988. Institute for New Generation Computer Technology (ICOT).

[HJ90] S. Haridi and S. Janson. Kernel Andorra Prolog and its computation model. In *Proc. of the Seventh International Conference on Logic Programming*, 1990.

[HJ91] S. Haridi and S. Janson. Programming paradigms of the Andorra Kernel Language. Technical report, SICS, Sweden, 1991.

[Kor89] W. Kornfeld. Constraint programming in Andorra Prolog. Presented at the Swedish-Japanese-Italian workshop, 1989.

[Sar87] V.A. Saraswat. The concurrent logic programming language CP: definition and operational semantics. In *Conference Record of the Fourteenth Annual ACM Symposium on Principles of Programming Languages*, pages 49–63. ACM, New York, 1987.

[Sar89a] V.A. Saraswat. *Concurrent Constraint Programming Languages.* PhD thesis, Carnegie-Mellon University, january 1989. Published by The MIT Press, U.S.A., 1990.

[Sar89b] V.A. Saraswat. Programming in Andorra Prolog. Technical report, Xerox PARC, 1989.

[War87] D. H. D. Warren. The Andorra principle. Presented at the Gigalips workshop, Stockholm, 1987.

[Yan89] R. Yang. Solving simple substitution ciphers in Andorra-I. In Giorgio Levi and Maurizio Martelli, editors, *Proc. of the Sixth International Conference on Logic Programming*, Lisboa, 1989. The MIT Press.