Implementação de Linguagens de Programação Lógica

Extended Andorra Model

Ricardo Lopes
rslopes@ncc.up.pt
DCC-FCUP

Tópicos Avançados de Informática
Mestrado em Informática 2003/04

The Andorra Principle

- Much research on Logic Programming has been performed on improving the performance of Prolog through And/Or parallelism and concurrent execution.
- The work on these areas is substantial and already mature, however the combination of this work has introduced new difficulties affecting both language design and implementation.
- In order to tackle this problem, several languages were proposed more recently, namely the **Andorra-based languages**.

The Andorra Principle

In 1987 D.H.D Warren proposed the Basic Andorra Model.

- A goal is *determinate* if it has at most one candidate clause.
- Determinate goals are executed first.
- When no determinate goal exists, a non-determinate goal is selected for execution.

Advantages of the Basic Andorra Model:

- May reduce the search space.
  - ⋆ Deterministic goals need to be tried only once, rather than re-executed at different branches of the search space.
  - ⋆ Constraints from the deterministic goals may reduce the number of alternatives for other goals, and even make them deterministic as well.
- All deterministic goals can execute in parallel. The model supports two forms of parallelism extracted implicitly from the program:
  - ⋆ And-Parallelism, by running deterministic goals in parallel.
  - ⋆ Or-Parallelism, by exploring different alternative clauses to a goal in parallel.
- Programs are less execution order sensitive than in Prolog.
  `?- member(X,L), L=[1,2,3].` is finite in the Andorra Model.

The Basic Andorra Model

An Andorra implementation has to deal with novel problems inherent to the model:

- Determine which goals are deterministic can be hard.
  `boss(mary, john).     boss(mary,mary).     ?- boss(X,X).`

- Concurrency can break Prolog semantics.
  `?- write(Solutions), fail.         ?- var(X), X=a.`

- Programs may do more work.
  `a(1).  a(2).  b(3).  b(4).   ?- a(X), b(X), lots_determ_work.`

Andorra-I

- The Andorra-I is the most well-known implementation of the Basic Andorra Model. It exploits or-parallelism and determinate dependent and-parallelism while fully supporting Prolog.

- The Andorra-I system consists of three main components:

  1. Preprocessor: responsible for generating the determinacy code and the sequencing information necessary to maintain the correct execution of programs.
  2. Engine: responsible for executing the Andorra-I programs. The engine consists of teams of workers, where each worker normally corresponds to a CPU.
  3. Scheduler: responsible for finding new tasks (work) for workers that have completed their tasks.

- Despite the excellent results that the Andorra-I attained, the system is limited by the fact that coroutining and and-parallelism can only be exploited between determinate goals.

EAM Motivation: Attain Maximum Performance

In 1990 D.H.D Warren proposed the Extended Andorra Model (EAM)

- **Mininum number of inferences:**
  $\longrightarrow$ try to never repeat the same execution step in different locations of the execution tree.

- **Maximum parallelism:**
  $\longrightarrow$ allow goals to be executed as independently as possible, and combine solutions as late as feasible.

- **Implicit control:**
  $\longrightarrow$ ideal behavior without requiring too much reliance on user annotation.

EAM Characteristics

- **Extends the Basic Andorra Model** by allowing non-deterministic goals to execute in and-parallel as long they do not bind external variables.
  - ⋆ **BAM** clause determinacy – don't guess a clause till you have to.
  - ⋆ **EAM** binding determinacy – don't guess a variable binding till you have to.

- **Explore the three main forms of Parallelism:**
  - ⋆ Or-Parallelism: between alternatives.
  - ⋆ Independent And-Parallelism: between goals that do not share variables.
  - ⋆ Dependent And-Parallelism: between goals that share variables.

EAM different approaches

- One approach was followed by researchers at SICS who concentrated on the **AKL**, the Andorra Kernel Language, based on the principle that the advantages of the Extended Andorra Model justified a new programming paradigm that could subsume both traditional Prolog and the committed choice languages.
  - ⋆ Explicit Control Scheme: AKL programs were constructed from guarded clauses, where the guard could be separated with a sequential conjunction, cut, or commit.
- In contrast, David H. D. Warren and researchers at Bristol and UP concentrated on the Extended Andorra Model with Implicit Control.
  - ⋆ The goal was to obtain the advantages of the Extended Andorra Model with the least effort from the programmer.
  - ⋆ BEAM was the first sequencial implementation of the EAM.

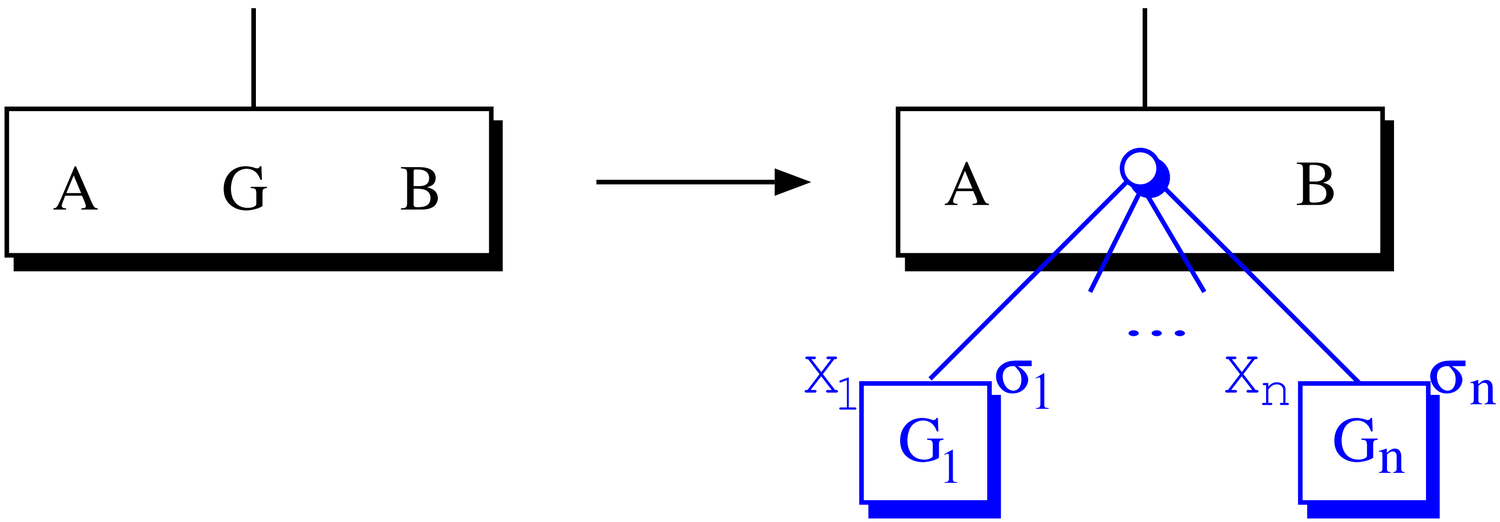
BEAM Concepts

The EAM is formally defined through rewrite rules that manipulate and-or trees.

- **And-boxes** – clause with subgoals $G_1, \ldots, G_n$; include:
  → local variables $X_1, \ldots, X_m$ created in the box,
  → and constraints, σ, on external variables imposed by the box.
- **Or-boxes** – alternative clauses for a goal, $C_1, \ldots, C_n$.
- **Rewrite rules:**
  → reduction, promotion, substitution (propagation) and forking (splitting).
- **Simplification and optimization rules:**
  → to simplify the and-or tree and to discard boxes.
- **Control strategies**
  → define how and when to apply the rules.

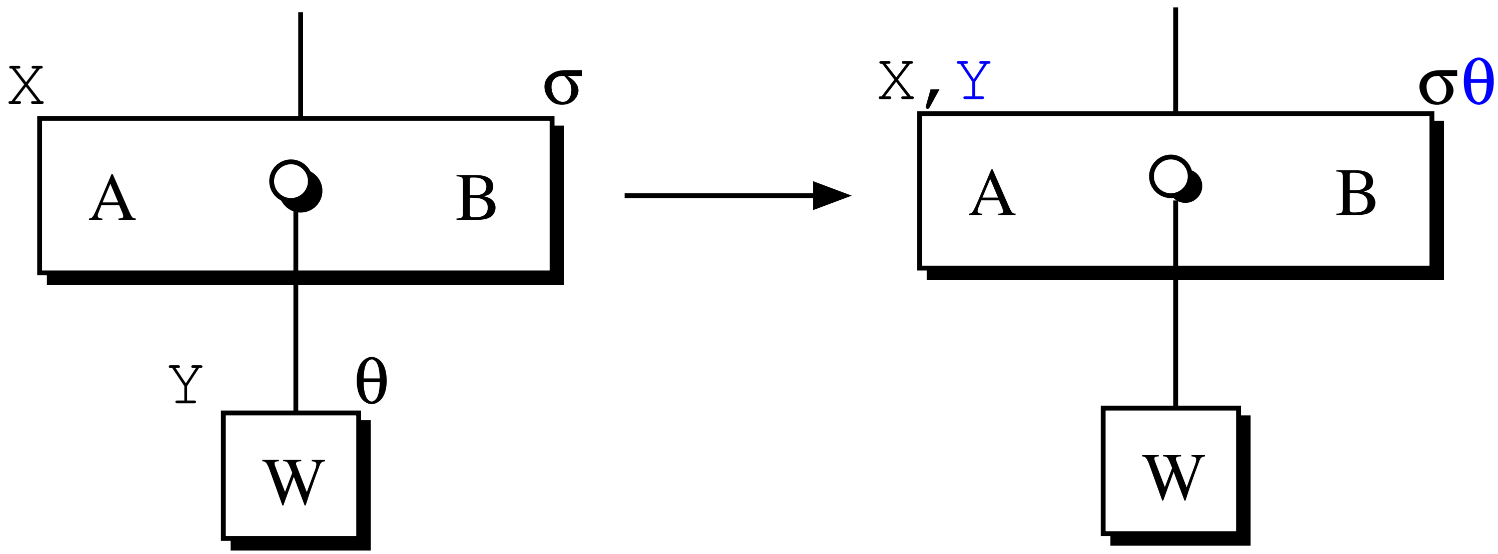
Main Operations of BEAM I

Reduction: expands a goal `G` into an or-box.



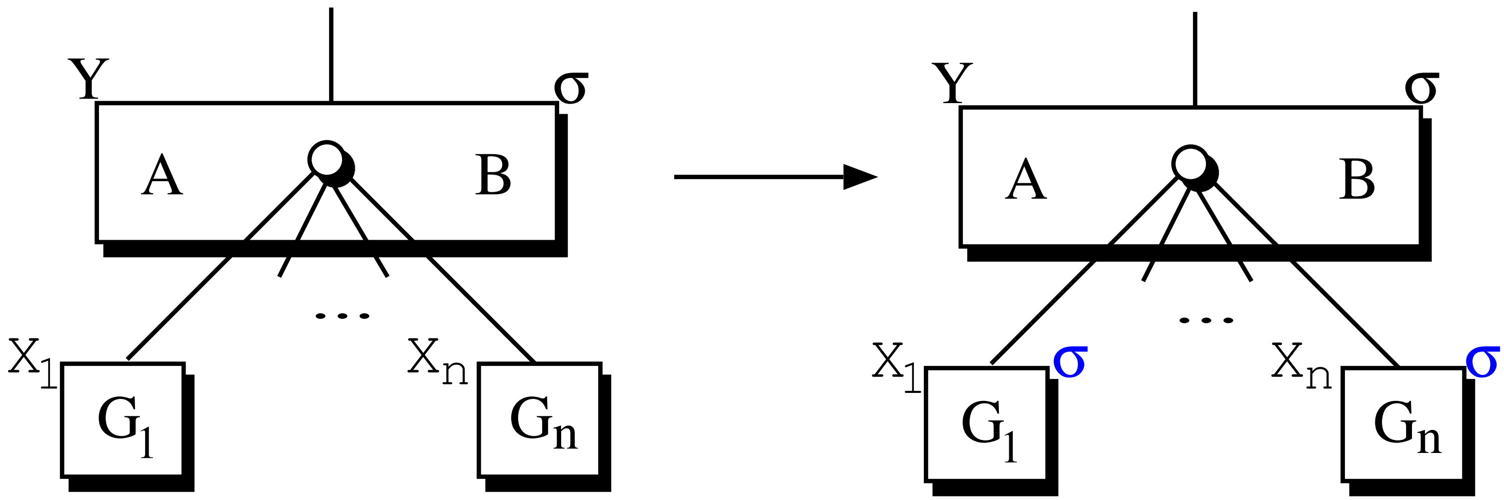
Main Operations of BEAM II

Promotion: promotes constrains from a inner-box to the outer-box.



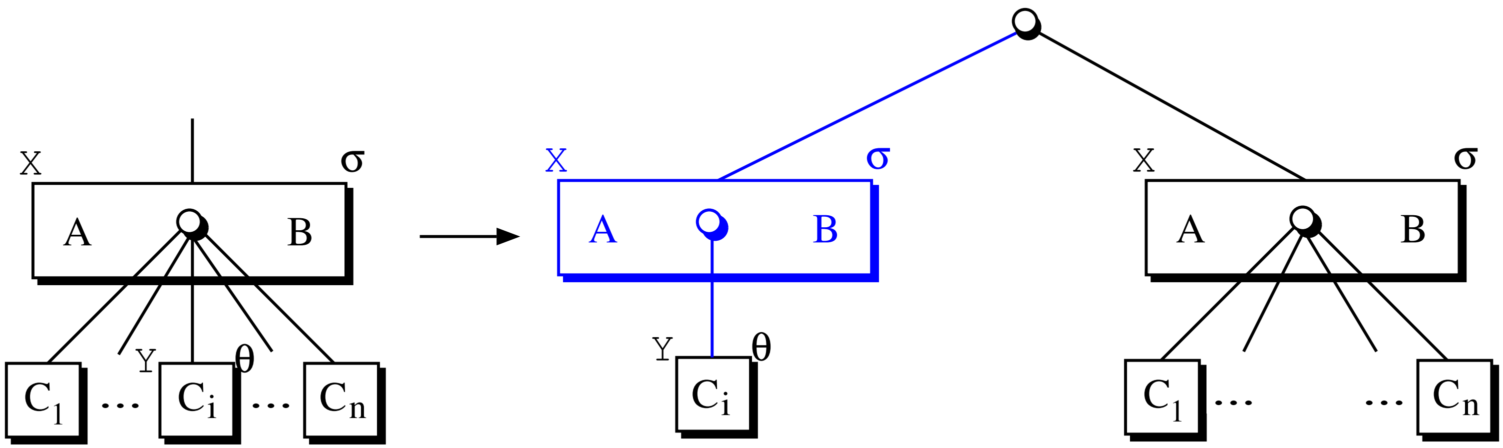
Main Operations of BEAM III

Propagation: propagates constrains from an outer-box to the inner-boxes.

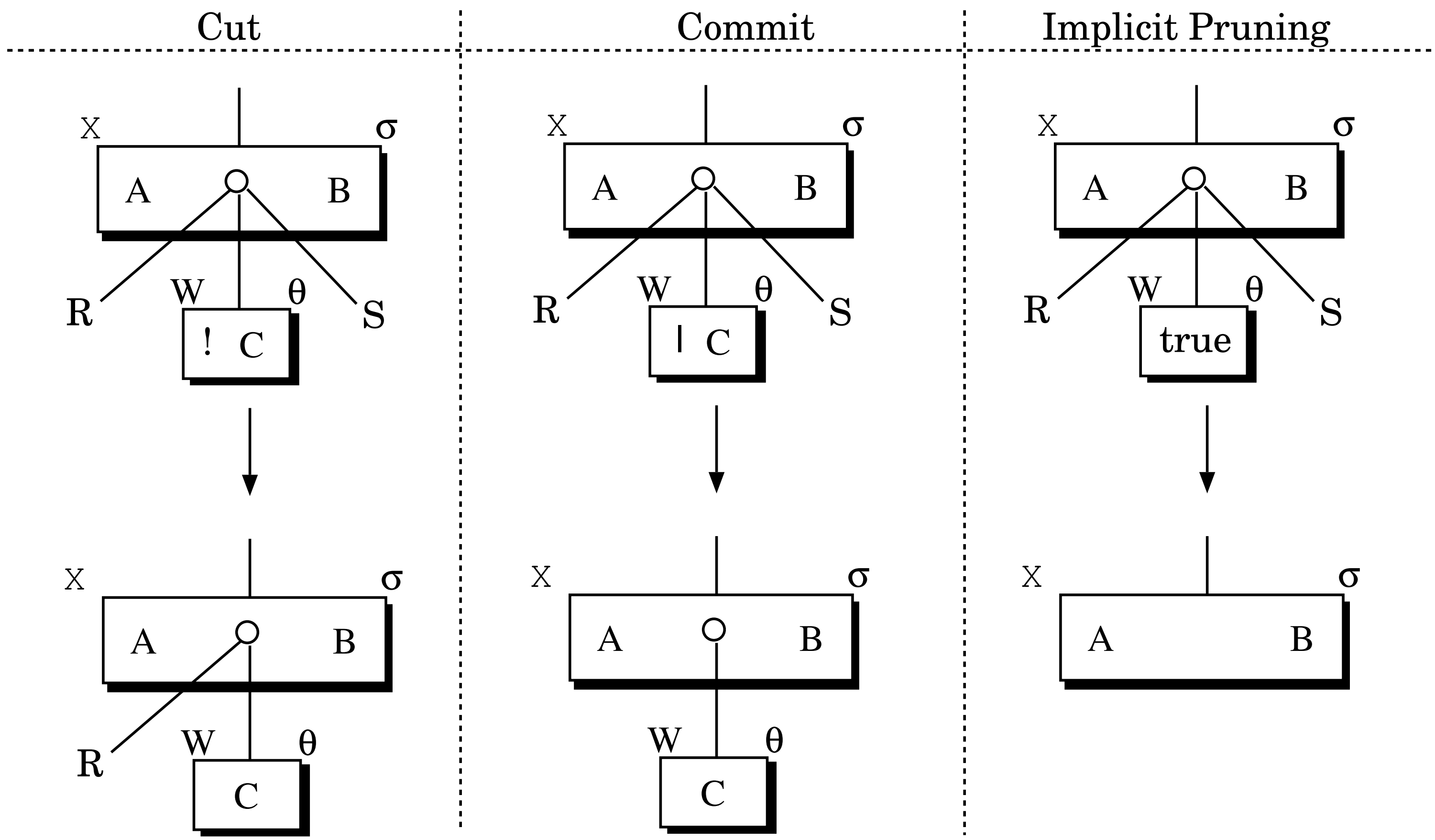


Main Operations of BEAM IV

Splitting: distributes a conjunction across a disjunction.

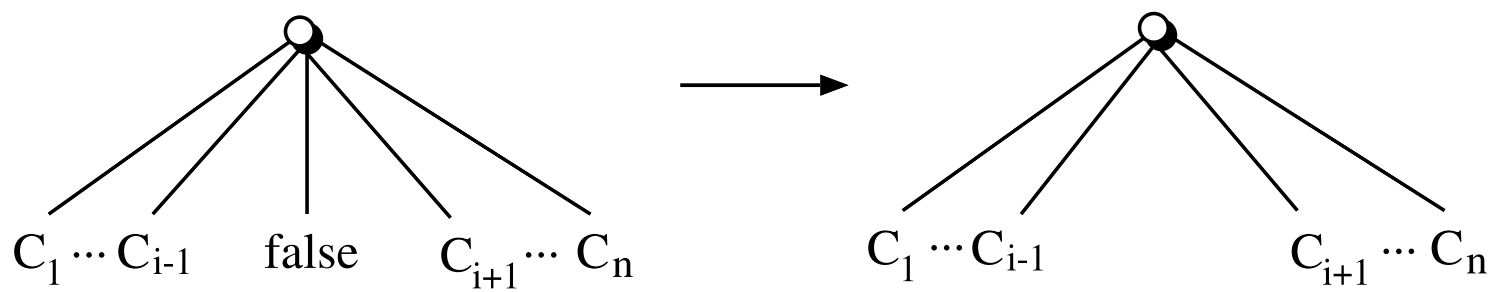


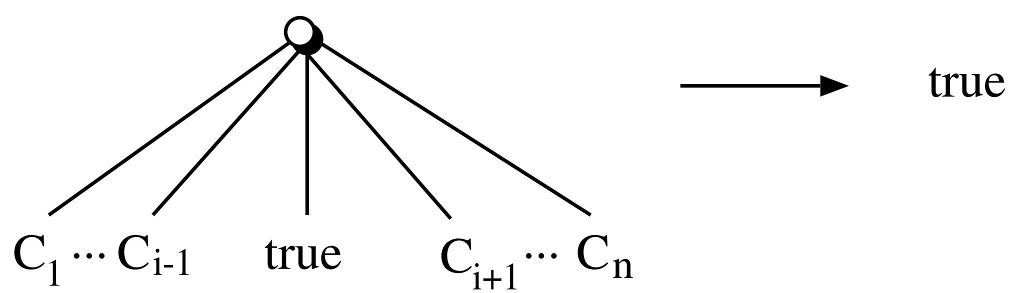
EAM pruning rules



Simplification Rules and Implicit Prunning I

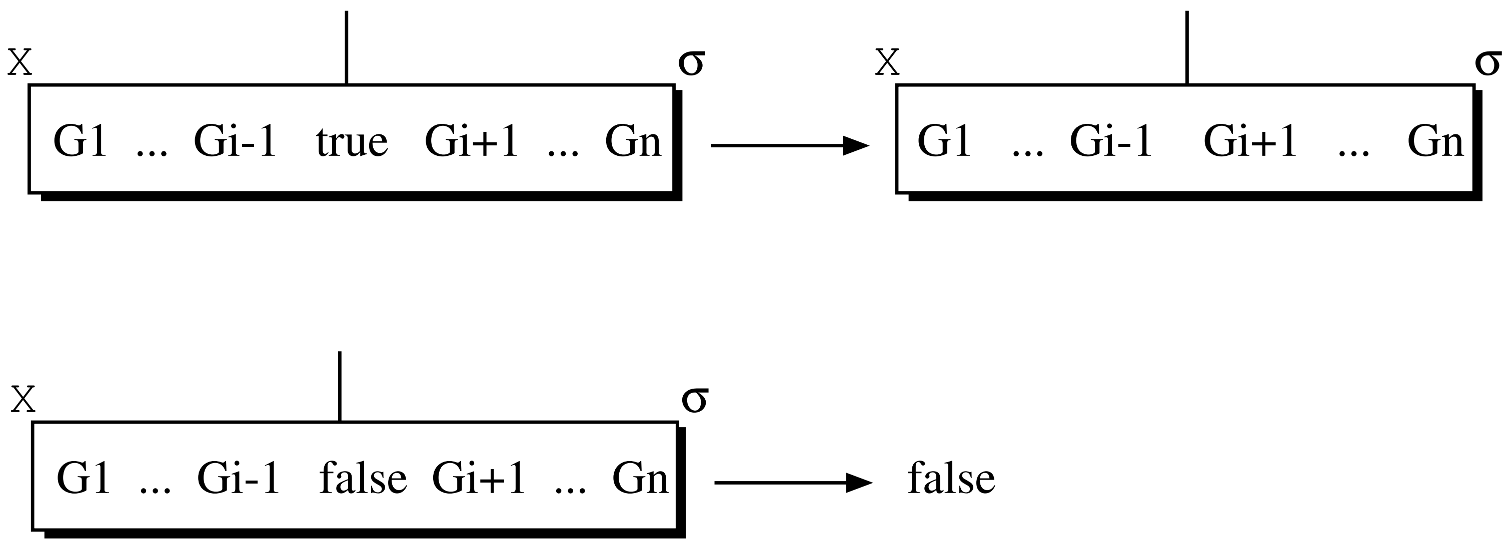
false-in-or & true-in-or Simplifications:





Simplification Rules and Implicit Prunning II

true-in-and & false-in-and Simplifications:



Execution Control

- The Extended Andorra Model gives us a set of basic operations which are logically valid, and which can be controlled in a variety of ways.
- The unrestricted application of the described rules will surely lead to a completely unpredictable and undesirable computation.
- The original EAM design tries to keep the control implicit, as much as possible, that is, it does not rely on information supplied by the programmer.
- The control decisions are based exclusively on information implicitly extracted from the program.
- One of the aims of the Extended Andorra Model is to perform the least number of reductions to obtain the solutions for a goal.

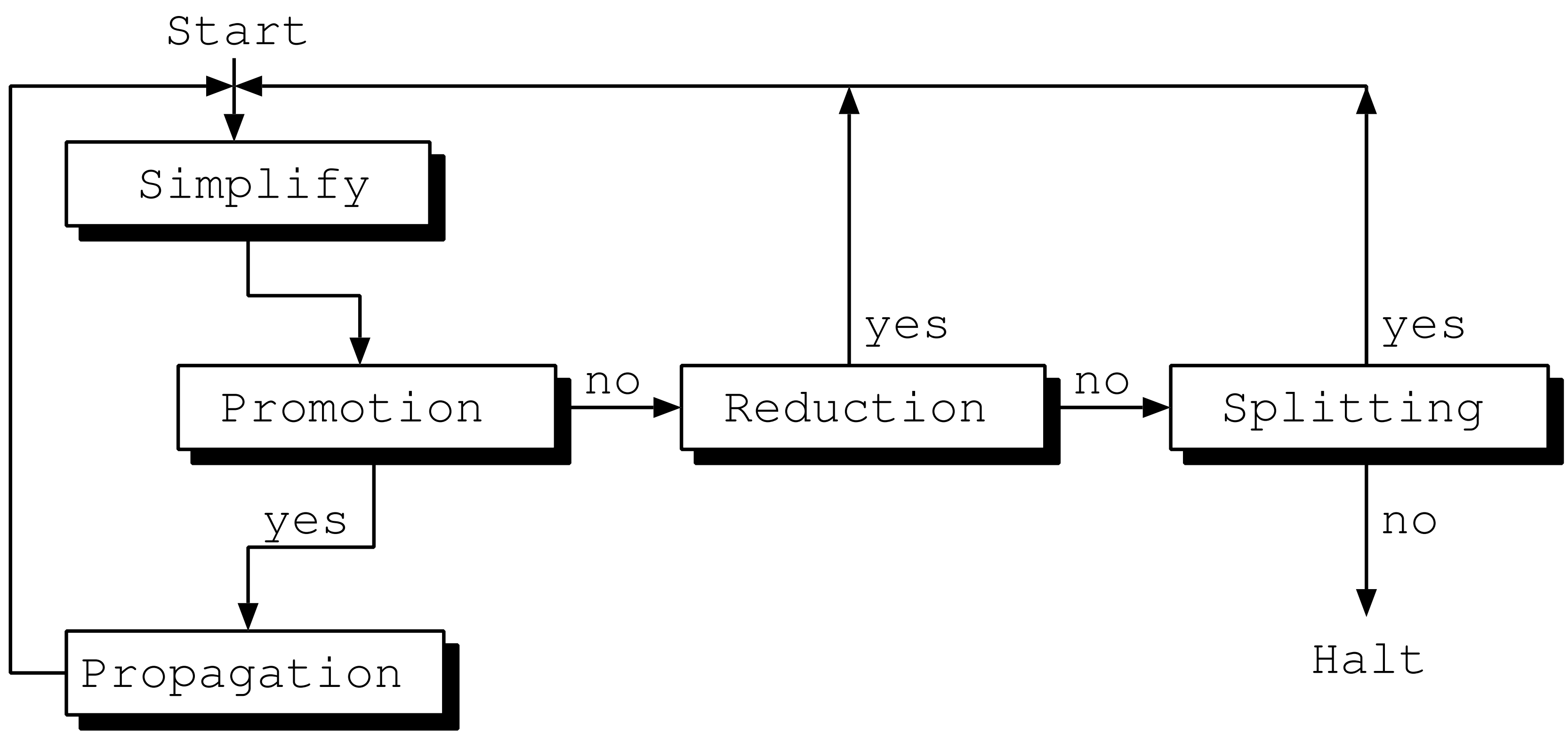
Execution Control

David Warren presented a set of intuitions on how to optimally evaluate $[\exists X : (P_1 \ \& \ldots \& \ P_n)]$:

- if we have no solution for X, we only want to work on goals which will lead to failure;
- if we have one or more solutions to X, it is safe to work on all goals;
- if there are multiple solutions for X, we want to perform all work that does not depend on X before installing different instantiations for X;
- if one goal can generate a unique value for X, allow that goal to be the *producer* and others to be *consumers*. Otherwise we need to select one goal to be a *non-determinate producer* and let other goals be consumers;
- use *splitting* solely to transmit non-determinate bindings from producer to consumers, making copies of consumers in the process.

BEAM Execution Control

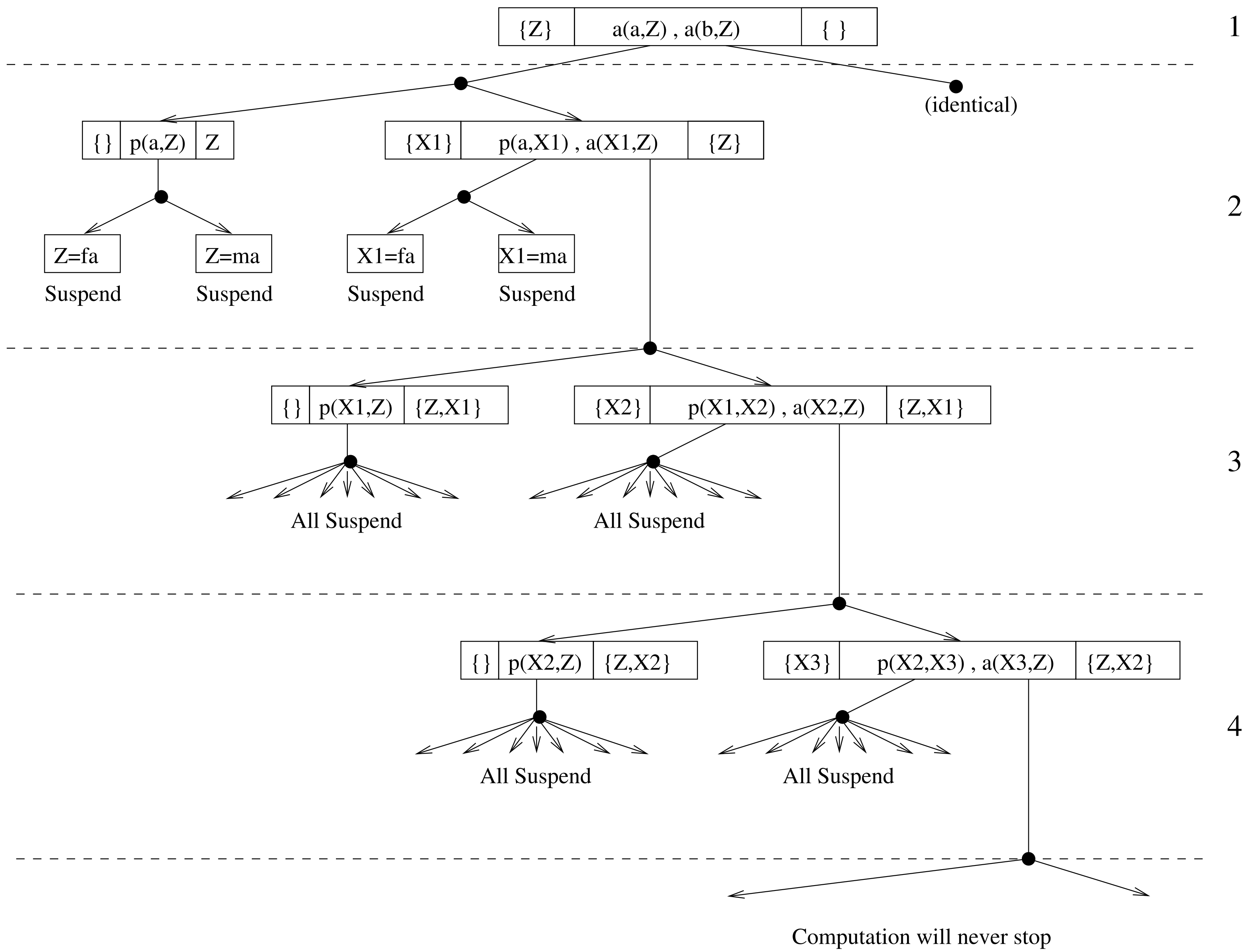
Control strategies: define how and when to apply the rules.



Non-Termination

- As long as it does not bind variables, the EAM allows the parallel early execution of non-determinate goals. In some cases, this may create speculative and-work, as some other goals might fail. This may result in a larger search space or even lead to **non-termination**.

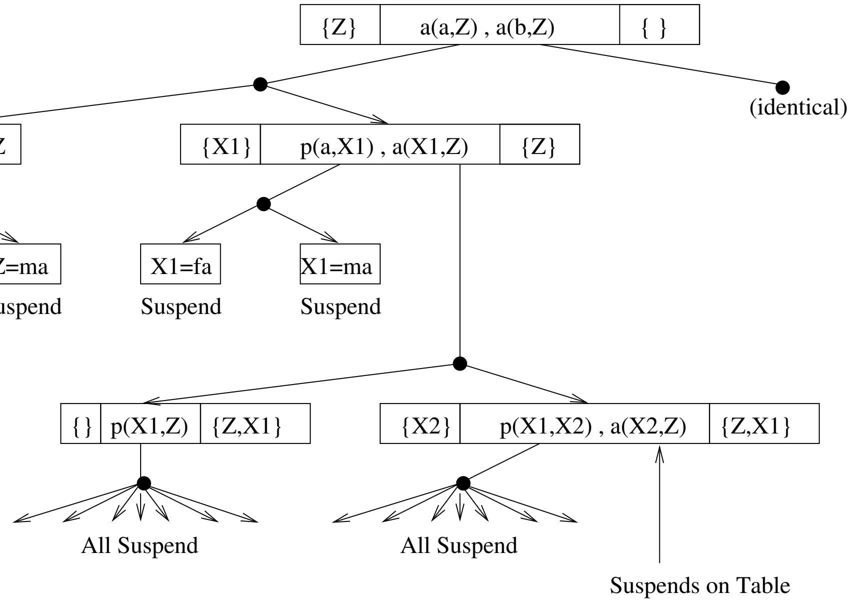
```
ancestor(X,Y):- parent(X,Y).
ancestor(X,Z):- parent(X,Y), ancestor(Y,Z).

parent(a,fa).    parent(a,ma).    parent(b,fb).  parent(b,mb).
parent(ma,mma).  parent(ma,fma).  parent(mb,c).  parent(mb,d).
parent(fb,mfb).  parent(fb,ffb).  parent(fa,c).  parent(fa,d).

?- ancestor(a,Z), ancestor(b,Z).
```


{Z}
a(a,Z) , a(b,Z)
{ }
1
(identical)
{ }
p(a,Z)
Z
{X1}
p(a,X1) , a(X1,Z)
{Z}
2
Z=fa
Z=ma
X1=fa
X1=ma
Suspend
Suspend
Suspend
Suspend
{ }
p(X1,Z)
{Z,X1}
{X2}
p(X1,X2) , a(X2,Z)
{Z,X1}
3
All Suspend
All Suspend
{ }
p(X2,Z)
{Z,X2}
{X3}
p(X2,X3) , a(X3,Z)
{Z,X2}
4
All Suspend
All Suspend
Computation will never stop

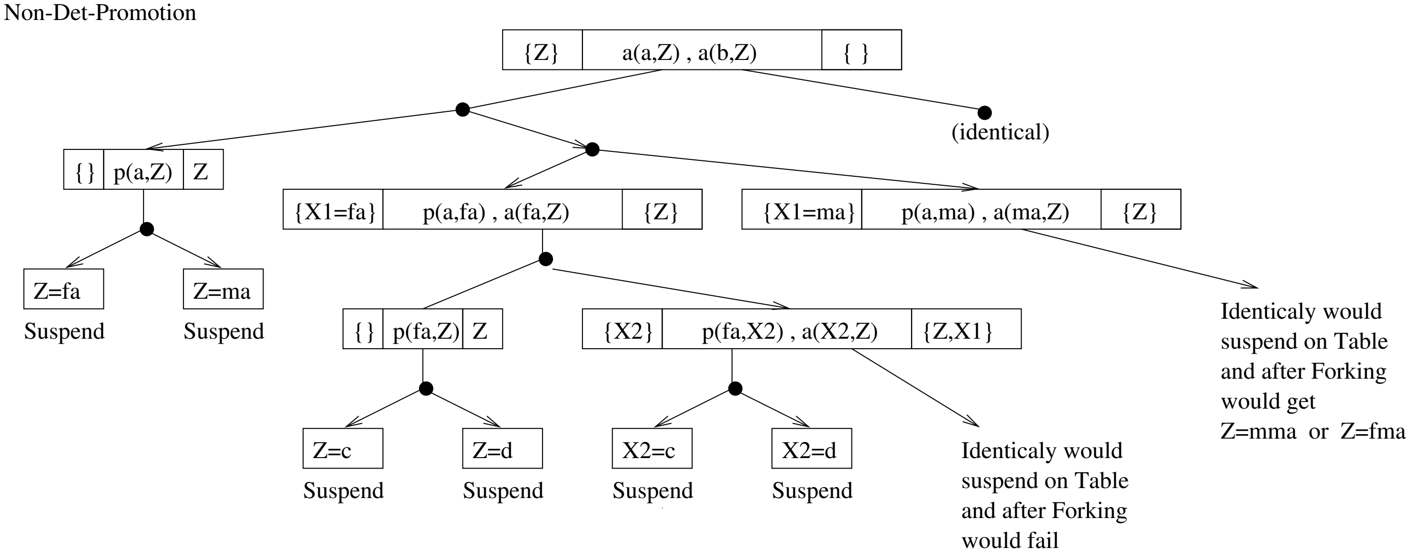
Dealing with Non-Termination

- The solution that we proposed is the combination of:
  - ⋆ Eager Non-Determinate Promotion
  - ⋆ Tabling
- This solutions is ideal in two contexts:
  - ⋆ First it guarantees that the computation end in programs that have finite solutions.
  - ⋆ Second, it allows the reuse of goals.

TABLED: a



Non-Det-Promotion



On the End we would have for a(a,Z) the solutions :
Z=fa or Z=ma or Z=c or Z=d or Z=mma or Z=fma

On the End we would have for a(b,Z) the solutions :
Z=fb or Z=mb or Z=c or Z=d or Z=mfb or Z=ffb

Deterministic Benchmarks

Benchs.	BEAM	AKL	Andorra-I	YAP 98	YAP 4.4	SICStus 3.8.6
cal	0.010	0.020	0.020	0.009	0.005	0.012
deriv	0.060	0.380	0.050	0.019	0.013	0.024
qsort	0.49	1.07	0.46	0.18	0.11	0.38
serialise	0.23	0.69	0.38	0.08	0.06	0.18
reverse_1000	0.30	1.60	0.37	0.13	0.12	0.17
nreverse_1000	130	780	140	50	40	60
kkqueens	240	460	240	127	40	70
tak	110	154	140	60	30	50
average	62%	24%	49%	136%	213%	100%

Non-deterministic Benchmarks

Benchs.	BEAM Default	BEAM ES	AKL	AND.-I	YAP 98	YAP 4.4	SICStus 3.8.6
ancestor	NA	0.29	0.7	0.19	0.02	0.018	0.059
houses	5.0	4.1	13	1.4	0.6	0.5	1.1
query	34	5.5	45	2.1	0.73	0.35	0.97
zebra	100	24	54	46	13.5	10.5	19.4
puzzle4x4	2,730	-	2,850	960	320	270	360
average	14%	37%	13%	47%	173%	229%	100%

Reduced Search Benchmarks

Benchs.	BEAM	AKL	And-I	YAP 4.4
send_money 1st	14	20	1.7	17,820
send_money all	82	120	9.0	129,840
queens-9	12	20	1.6	30
queens-16	19,990	26,820	720	>24 hours
queens-20	787,030	962,550	19,460	>24 hours
check_list-10	0.26	1.13	343,680	56,180
check_list-11	0.29	1.47	11,221,800	1,826,400
check_list-15	0.49	3.44	>24 hours	>24 hours
ppuzzle-A	22	28	246,460	>24 hours
ppuzzle-B	60	62	>24 hours	>24 hours
ppuzzle-C 1st	7	10	3,747,980	>24 hours

To End...

- The EAM with Implicit Control can be implemented efficiently.
- The model performs well, even when just using implicit control.
- Simple programmer annotations and prunning can often lead to a better performance.
- The EAM can extend logic programming for applications where Prolog would not cope well.