# The integration of simulation and knowledge-based systems

Groen, A.; van den Herik, H.J.; Hofland, H.I.; Kerckhoffs, E.J.H.; Stoop, J.C.; Varkevisser, P.R.

*Published in:*
AI applied to simulation

*Publication date:*
1986

# The integration of simulation and knowledge-based systems

A. Groen, H.J. van den Herik, A.G. Hofland,
E.J.H. Kerckhoffs, J.C. Stoop, P.R. Varkevisser
Delft University of Technology
Dept. of Mathematics and Informatics
Julianalaan 132, 2628 BL Delft
The Netherlands

ABSTRACT

The impact of knowledge-based systems in simulation environments is considered with respect to incomplete characterizations, yet requiring a great volume of knowledge. In order to compete with a real-world system, the knowledge-based system in the simulation is subject to severe speed requirements, which could be met with parallel-processing techniques. At the other hand, simulation is the appropriate tool for studying the performances of new proposed non-conventional designs for parallel knowledge-based systems.

As to this duality in the integration of simulation and knowledge-based systems, in this publication the emphasis is on simulating a parallel knowledge-based system. The simulation program HYDRA is described, distinct differences, even in the determinacy of results, are discussed and three different approaches of measuring the impact of parameters are investigated.

## 1. INTRODUCTION

The ability to use knowledge is one of the most fundamental attributes of intelligent behaviour. In any subdomain of Artificial Intelligence (AI), the questions 'which knowledge is important ?' and 'how to manipulate knowledge ?' are raised. Deciding which knowledge is relevant within a restricted, completely definable domain is a difficult problem, even for an expert. If the domain is highly complex, it is especially difficult to establish which knowledge is relevant. The complexity of any real-world domain compels compliance with the characterization of its structure. Therefore we must make do with incomplete characterizations,inadequate for modelling the domain. The inadequate models, in turn, bar the way to effective simulations.

In order to overcome these inadequacies as much as possible, one might attempt to use knowledge-based systems (KBSs) for the purpose of model synthesis. The central question in employing KBSs to create simulation models still is, whether the synthesis process (or part of it, be it the identification or validation process) can be adequately formalized and captured in rules. The well-known methodologies developed by Oren and Zeigler [ 9 ] provide some perspectives for applying Artificial Intelligence techniques in simulation.

A more direct way to very effective simulations is to shelter the decision making agents in a KBS. This is especially of interest to the service industries, but also to production control and quality control [12]. In order to decide within real time, high-speed execution of the knowledge available is necessary, because in many cases it will not be allowed that the real-world system is faster than its simulation with the built-in expert system.

A characteristic feature of expert systems is that the inference mechanisms and the knowledge are treated separately [ 4 ]. In most expert systems, knowledge is stored in the form of rules. This approach has the advantage that the knowledge is easily expressible and expandable. Rules, being modular, allow augmenting the knowledge incrementally and so to improve the model's verisimilitude.

Obviously, a knowledge-based simulation will be a better emulation of its counterpart in reality as the number of rules (and so the knowledge) increases. Hence, the problem arises how to treat the large numbers of rules adequately [ 5 ]. A parallel approach would seem suitable. However, parallel computers are still in their infancy ; therefore, many researchers resort to simulating a parallel computer. In this contribution, it is shown that this choice emphasizes the use of simulation in AI rather than of AI in simulation [ 7 ].

## 2. PARALLEL KNOWLEDGE-BASED SYSTEMS

The most prominent reason for parallelization is the increase of speed. As outlined before, three aspects can be called for which this may be of importance :
( i) performance improvement of systems with large rule bases ;
( ii) performance improvement in real-time applications ;
(iii) applications in a simulation environment.
In (iii) we are concerned above all with achieving the desired increase of speed when combining a conventional simulation model and a knowledge-based system (see section 1). When examining the possibilities of parallelization, the first idea is often to construct parallel inference engines. However, in such a construction a Master Control is to be introduced. The many tasks to be performed by the Master Control (communicating, determining priorities, registrating results, etc.) will probably cause a loss of the time gained by parallelization [ 3 ]. A second idea is to take one inference engine and parallelize it.

### 2.1. An Agenda-Driven Parallel Knowledge-Based System

In figure 1 the design of a parallel knowledge-based system with one inference engine consisting of a number of simultaneously operating rule processors (RPs) is presented. The rule processors find their tasks in the rule agenda (RA) arranged by the rule agenda manager (RAM). Whenever a rule processor can find neither a proof nor a disproof of a rule, nor other rules which might be of help, it sends a question to the question agenda manager (QAM). The QAM arranges the question agenda (QA) and communicates with the user by means of the user interface.

### 2.2. The Rule Agenda

The rule agenda is an idea adopted from Lenat [ 8 ]. Its structure has been kept simple. For every rule in the rule base, an entry with two fields is created : a status field and a priority field. Each status field can take three values :

0 : meaning that the rule is neither proved nor disproved, and therefore is due for processing ;
1 : meaning that the rule is being processed ;
2 : meaning that the rule has been proved or disproved, and therefore its conclusion acquires the status of a fact.

Rationally, we allow a single rule to be processed by one rule processor at a time only. The value of the priority field, a measure of the importance of the rule to be processed, is dependent on the strategy implemented. An optimal priority function will be strongly related to the structure of the domain.

In the current implementation, a plain strategy is employed, which is found to lead to surprising interruptions of the search process when acting in a promising part of the search space : this interruption is called the staggering effect. When a rule processor has found a set of rules, $\{R_i\}$ (i = 1...n), which might be applied to prove the current rule $R_j \notin \{R_i\}$, it transfers this result to the rule agenda manager, which marks the status field of $R_j$ and assigns a new priority value to the priority field of each of the rules $R_i$. The priorities of all rules of the set $\{R_i\}$, having a status field $\neq 2$, are updated simply as follows :

$$PR_i = \max(PR_i, PR_j) + 1 \qquad (i = 1...n),$$

$PR_k$ meaning the Priority of rule $R_k$.

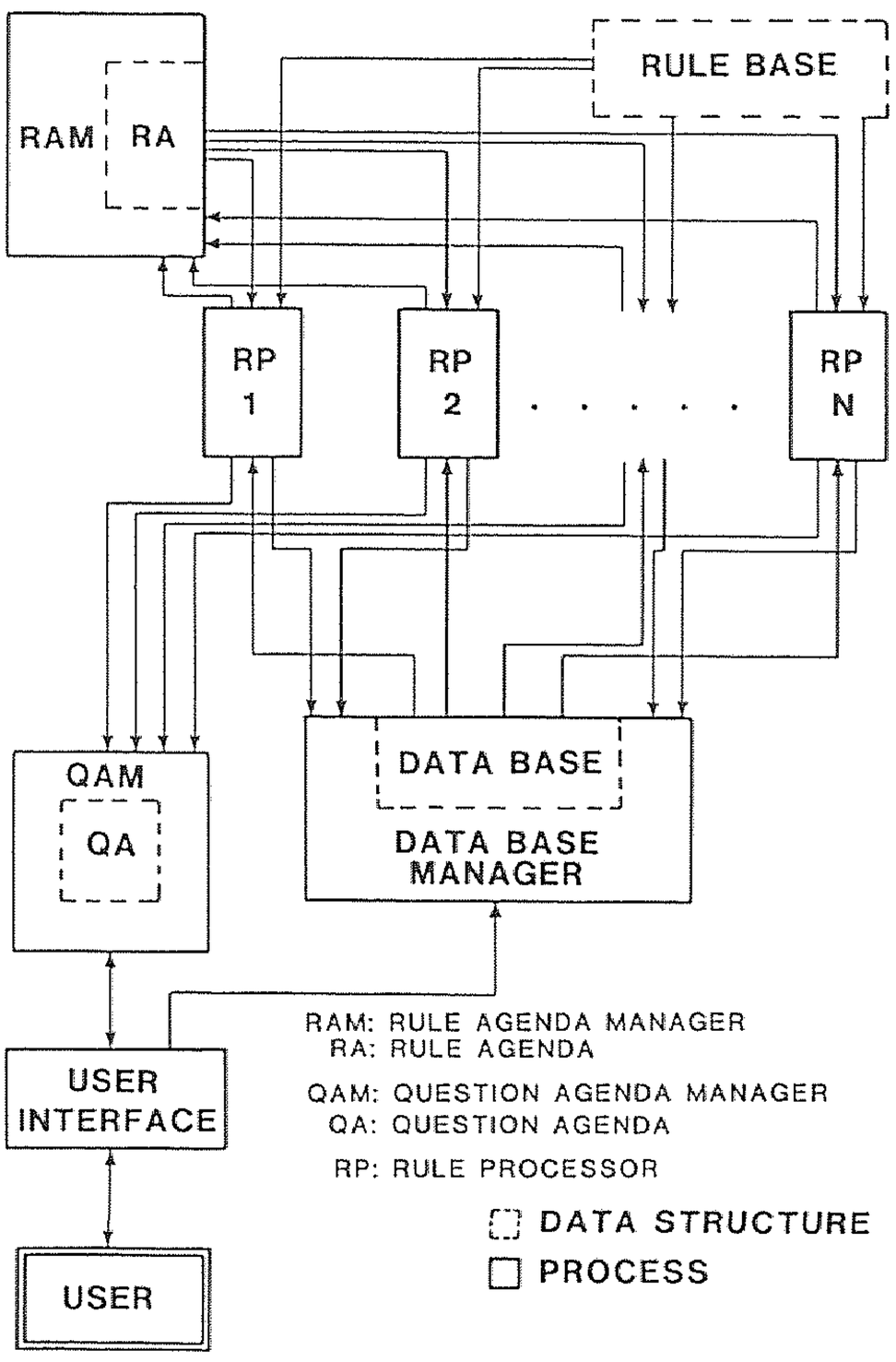


Figure 1. An Agenda-Driven Parallel Knowledge-Based System

An example may illustrate the idea. In figure 2, the situation is given when $R_3$ is being processed (the meaning of the status value is mentioned above).

| Rule Number | Priority | Status |
|---|---|---|
| 1 | 7 | 0 |
| 2 | - | 2 |
| 3 | 6 | 1 |
| 4 | 3 | 0 |
| ⋮ | ⋮ | ⋮ |
| 153 | 0 | 0 |

Figure 2. Example of the Rule Agenda Contents

Assume that a rule processor has found that $R_1$, $R_2$ and $R_4$ might be used to prove $R_3$. After the RAM has updated the situation as depicted in figure 2, a free rule processor encounters the situation of figure 3, which implies that it must take the rule with the highest priority, $R_1$, to process next.

| Rule Number | Priority | Status |
|---|---|---|
| 1 | 8 | 0 |
| 2 | - | 2 |
| 3 | 6 | 0 |
| 4 | 7 | 0 |
| ⋮ | ⋮ | ⋮ |
| 153 | 0 | 0 |

Figure 3. The Updated Contents of the Rule Agenda

## 2.3. The Tasks of a Rule Processor

In the parallel knowledge-based system described above, a rule processor is generally engaged in trying to (dis)prove one rule at a time. As soon as a rule processor is free, it consults the rule agenda for the next rule to be processed. A rule processor performs its tasks in the following order.

1. Take a rule with the highest priority from the RA.
2. For all clauses inspect whether they are (dis)proved facts.
3. IF definite answer THEN signal results to RAM and DBM [ rule is proved or disproved ], and terminate.
4. IF no definite answer THEN
   [ inspect RB for further applicable rules ;
   send result to RAM
   [ the rules selected will be included in the RA with their priority values updated ], and terminate.
5. IF no further rules are applicable THEN ask the user via the QAM, and terminate.

Depending on the results of such a rule processor one or more of the following actions have to be executed :
- updating the data base ;
- updating the rule agenda ;
- updating the question agenda.

### 2.4. Accessibility

Some remarks on accessibility are in order. Access to the rule agenda is exclusive to the RAM and the set of rule processors. We allow concurrent reading but exclude all reading while writing.

Concurrent reading in the data base is permitted to the rule processors ; writing in the data base is the privilege of the data base manager, receiving its information from the rule processors or the user interface. The rule processors have concurrent access to the rule base ; no writing is ever involved there.

## 3. HYDRA, A MODULA-2 SIMULATION PROGRAM

### 3.1. The Simulation and its Inputs

The absence of an appropriate multiprocessor induced us to simulate the parallel KBS on a uniprocessor system. The program, termed HYDRA (HYpothesis Deduction through Rule Application), is written in Modula-2 [15]. It runs on a PDP 11/45 of the Computing Centre of the Delft University of Technology.

Inputs to HYDRA in its present form are :
- the number of RPs ;
- the length of QA queue (QAQ) ;
- the name of the initial data base to be considered ;
- the name of the rule base to be considered.

HYDRA with one RP and with the RAM strategy outlined in section 2.2 is, very nearly, a conventional backward-chaining system with the due acceptance of staggering. Still, even in a backward-chaining system, the order of questions asked depends on its strategy, which may vary all along the path from depth-first to A* or AO* [10], hence staggering, though inconvenient, is not at all peculiar to HYDRA.

### 3.2. Synchronism and Mutual Exclusion within HYDRA

Writing the HYDRA simulator posed a few unexpected synchronization problems such as the recurrent cropping up of the well-known producer-consumer problem, in particular in the shape of the readers-writers algorithm ([ 1 ] pp.56-88). One instance is the consequence of introducing a buffer between rule processors and RAM.

While Modula-2 has formal primitives for a standard solution, it lacks a unified concept. We therefore introduced the concept of a Room (see figure 4). A critical action such as writing, must be executed in a Room ; each Room has an arbitrary but fixed number of Doors. A process attempting to enter by a Door, it finds closed, cannot enter the Room by that Door. When it finds a Door open, the process may enter the Room only on the additional condition that its entering will not cause the Room to exceed the maximum number of processes allowed therein (thus, though it might be rational to set the number of Doors equal to the maximum number of processes, this is no formal requirement ; the former number may exceed the latter). When inside

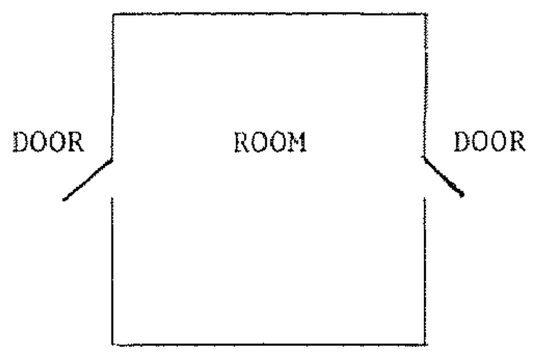


Figure 4. The Room-Door Concept

a Room, any process is able to open and close any of its Doors. The privilege of closing Doors is also available to a process waiting before an open Door until the desired Room is empty ; it is rational for such a process to close that Room's other Doors collectively. In the implementation of this concept, extensive use is made of semaphores [ 2 ]. In our opinion the concept 'Room' should be uniformly available in Modula-2.

An illustration of the concept Room is presented by showing how the RAM enters the rule agenda room (RAR) in order to pass down the results from a rule processor. Because the RAR also holds the RA, it is requisite for readers to visit the RAR too. In our implementation, the RAR has two Doors, a writers' Door and a readers' Door. We do not discuss how the RAM receives information. A situation as suggested in figure 2 is assumed when the RAM, the single admitted writer, wants to enter the RAR.

```
WHILE the RA is to be updated DO with RAR
BEGIN
   Close readers' Door ;                          (1)
   Wait until empty ;                             (2)
   Enter through writers' Door ;                  (3)
   Perform necessary updating actions ;           (4)
   Open Door for readers ;                        (5)
   Leave through writers' Door ;                  (6)
END
```

We assume that any incompleted action in course when a KB hypothesis is proved is aborted. Also it is assumed that RAM never enters RAR without attempting to pass down a result. After (1) no reader is able to enter the RAR because readers are not allowed to open Doors from the outside. In Modula-2, (2) and (3) are combined into 'Enter When Room Empty', which guarantees the exclusive Room entrance necessary for serialized writing. The consequence of (5) is that all waiting readers are activated and will rush into the Room. The addition 'through writers' Door'in (6) is given for reason of symmetry ; the writer can leave the RAR through any Door, because a closed Door may be opened from the inside. These problems, having been solved for the communication between RPs and the RAM, provide an immediately analogous situation for the communication between the RPs and the QAM.

## 4. SIMULATION RESULTS

Before entering upon a more detailed description of the simulation, we will provide some simulation results in order to show that the design of the agenda-driven parallel KBS as described in section 2 satisfies our requirements : the correct hypothesis is always proved in the end.

Consider, as a small but telling example, a FIFO case with a QAQ length 1 and a single processor, acting on a well-known animal-knowledge base [14] of 17 rules. There are 13 hypotheses and 43 distinct premises. Assume, it is to be proved that the animal in question is a penguin.
HYDRA now reaches its goals by successively asking the eight questions :

| | |
|---|---|
| - Is the animal a bird ? | (yes) |
| - Is the animal milk-yielding ? | (no) |
| - Does the animal have hair ? | (no) |
| - Is the animal oviparous ? | (yes) |
| - Can the animal fly well ? | (no) |
| - Is the animal long-necked ? | (no) |
| - Can the animal fly ? | (no) |
| - Can the animal swim ? | (yes) |

By varying the parameters, we obtain an indication of the efficiency of interrogation under various assumptions. Results are given in [ 3 ].

The second example reveals an unexpected but essential consequence of our parallel approach. HYDRA turns out to impose high demands on the consistency of the rule base as a whole, as appeared to its consistency as ordered rule base. While a sequential machine ran the flora-knowledge base, it always found one definite answer, due to implicit sequentialization of the rules within the Rule Base. However, this ranking being explicit by the accidental fact of a single RP, was not a true characteristic of the system. Hence, HYDRA running on 7 parallel RPs came to a different conclusion, indefinite to the extent of presenting a dichotomy.

- Are the leaves miniscule, squamiform and adjacent to the branch ? (no)
- Are the leaves aciform ? (no)
- Are the leaves distributed or variegated ? (no)
- Are the leaves squamiform ? (no)
- Are the leaves smooth ? (no)
- Are the leaves very sinuate ? (no)
- Is the plant a dwarfed tree or semi-bush, less than 30 cm in height, possibly recumbent ? (no)
- Are the leaves single-ribbed only ? (no)
- Are the leaves opposite ? (yes)
- Is the plant a tree or a bush ? (yes)
- Is the plant a climber or a winder ? (no)
- Do the midribs originate from one point ? (no)
- Are the leaves sinuate ? (no)
- Are its buds blue-green ? (no)
- Are its buds black ? (no)
- Is the plant a small parasite growing at tree branches ? (yes)
- Has its corolla five uniform leaves ? (no)
- Is its corolla papilionaceous ? (no)
- Is its petiole humpbacked twice ? (no)
- Have its flowers multiple symmetry ? (no)
- Are the leaves entire ? (yes)
- Are the leaves composed of a common rib with petals and sepals ? (no)

*The plant is a honeysuckle or a mistletoe.*

We stress again the extreme importance of this semi-indeterminacy. The same program, running on a single RP, provide the unambiguous but misleading answer :

*The plant is a honeysuckle.*

In a single-processor expert system, the determinacy of conclusions is a function of the sequence of processing rules. To some extent, this might be true for parallel KBSs too, but there the sequence is unpredictable; among others it is dependent on the number of rule processors and the strategy implemented. This leads to the high demands on the consistency of the rule base.

At the end of this section, we would like to remark that the outcome may surprise the layman, but naturalists (the real experts !) know better. The honeysuckle (Caprifolium) is a member of the Dicotyledoneae class (order Dipsacales), whereas the mistletoe (Viscum Album) is a member of the same class (order Santales). The ambiguity might be involved by the question on parasite growing. A mistletoe always does, but a honeysuckle might be considered as a parasite winder with its roots in the ground in stead of in a branch of another tree. An extra difficulty is that, in the U.S., the latter is regarded to be a member of the *same class but belonging* to the very large group of Phoradendron (in this instance, P. flavescens). To summarize it all in a question : may a parallel expert system have some difficulties when experts themselves have divergent opinions too ? At least, a parallel KBS is able to show the difficulties.

## 5. PARALLEL COMPUTER SIMULATION

On our way to achieve the aim of simulating a parallel computer, three different approaches have been investigated. All three have been implemented and examined on their merits. The first attempt, a concurrent process handler, was to simulate a parallel program by means of concurrent programming techniques [ 1 ]. Subsequently, to render it possible to record run and idle times, thus facilitating the understanding of the simulation, a time slicer was developed. This approach contained certain drawbacks, discussed below, resulting in the decision to construct a parallel computer simulator. All three approaches rely on the same set of mechanisms for mutual exclusion and synchronization ; the differences lie in the transfer mechanisms. The concepts on which the mechanisms are based have been presented in section 3.2. In all approaches, the processes to be parallelized are on a circular list in order to provide the scheduler with easy access to the processor.

### 5.1. Concurrent Process Handler

Initially, in the simulation use was made of a Concurrent Process Handler (CPH), determining the order in which the processes (RAM, DBM, one of the RPs, etc.) should be processed. The scheduler of the CPH uses a random number generator to choose the next process to be activated. When a running process calls a procedure for performing some action critical for synchronization, this procedure in its turn will call the scheduler and a transfer of control will always take place. This being a random transfer, the same process, when not held up, might be re-activated. This method clearly is explicit and this has one distinct drawback. Assume that the circular list consists of n processes with n-1 processes loosely coupled and one process independent of the others. If the latter is activated, it will never call the procedure for performing critical action or synchronization. This implies that in such a situation no transfer of control will take place, resulting in starvation. In HYDRA the user interface is independent of the other processes ; in order to prevent a starvation we have introduced a dummy call to the procedure which handles mutual exclusivity.
Disregarding this drawback, the main disadvantage of the CPH approach is the impossibility of estimating the impacts of parameters, such as the number of rule processors, the number of ports to the data base, the strategy in the RAM, the length of the QA queue, etc.. Such an estimate presupposes exact measurings of the run and idle times of the parallel processes, implying that a clock device is required. Since the Concurrent Process Handler provides no possibility for including a clock, this approach was dropped.

### 5.2. Time Slicer

In order to estimate the parameters' impacts, a time slicer was built [13], counting slices in multiples of n-hundreds of µs, n to be set by the user. It accumulates delays before individual actions take place and it records them as wait times. Moreover, it furnishes estimates of the various processes' idle times. As already mentioned, the processes are placed in a circular list and the scheduler inspects them in a round robin order imposed by the list. The scheduler is activated by a clock interrupt after a full slice has passed. The transfer of control is not due to an explicit procedure call as in the CPH, but by a clock interrupt, thus avoiding one of the CPH drawbacks ; this transfer mechanism is therefore implicit. We shall elucidate this below.

In the examples, we assume the slices to be of the same size for every process ; we have experimented with time-slice sizes which may vary with the nature of the processes, but this only complicates matters. In all cases, the time slicer turned out to be accompanied by disadvantages, arising mainly from the fact that the results are dependent on the way in which the processes are ordered in the circular list. We shall provide some simple instances to illustrate the problem of loosely coupled processes.

We assume the reader to be familiar with the semaphores as introduced by Dijkstra [ 2 ]. In HYDRA, the semaphores 'wait' and 'send', built up of Modula-2 statements, are used as P- and V-operation respectively. In a real parallel machine, two (loosely coupled) processes will be processed as shown in figure 5.

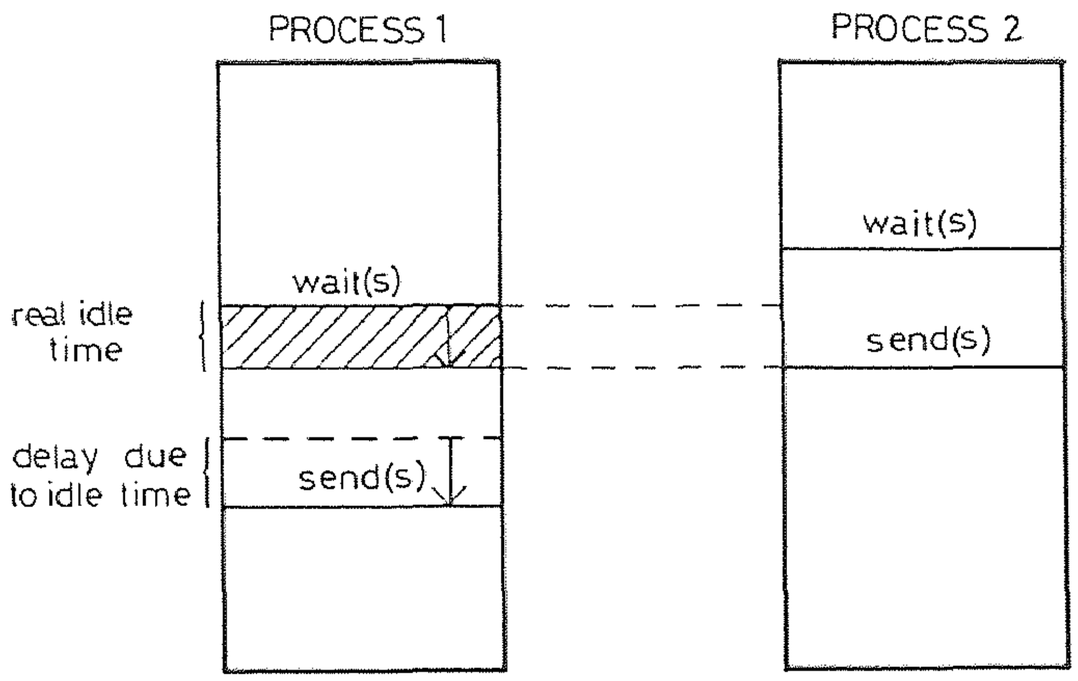


Figure 5. Two Real Parallel Processes

Looking at both processes in figure 5, we see that wait(s) is called first in process 2, allowing it to execute a critical action. At some time later, while process 2 is still occupied with its critical action, process 1 calls wait(s). Since the semaphore s is occupied, process 1 has to wait until s is set free. After process 2 has called send(s) the semaphore s admits process 1 to start its critical action. In figure 5, the real idle time constituting the part relevant to our discussion is indicated by a shaded area.

In a simulation environment with time slices, we would like to see that the amount of "real" idle time recorded is identical with the real idle time as presented in figure 5 ; or, if not identical, that the differences are at least acceptable, meaning that they can be accounted for. However, experiments showed that the order of the processes plays an exacting role. Different recorded idle times may be arrived at depending on the order of the processes in the circular list.

To state it more strongly, we cannot even predict whether the recorded idle times will be smaller or larger than the real idle time, both may occur with different ordering of the processes ; it is even perceivable that the recorded idle times interchange from one process to another, as can be seen in the figures 6 and 7. The time slices are indicated by $[ t_0,t_1 ]$, $[ t_1,t_2 ]$, etc. A time slice $[ t_{i-1},t_i ]$ $(i = 1,2,...)$ in figure 6 is first given to process 1 and then to process 2. In the time slice $[ t_1,t_2 ]$ process 1 calls wait(s) ; semaphore s is not engaged and so process 1 starts its critical action. At the end of the time slice (at time $t_2$) when it is still in its critical action, process 1 is interrupted. The scheduler offers process 2 a time slice. In this time slice process 2 calls wait(s), but the semaphore s is occupied, so process 2 cannot start its critical action, implying that the rest of the time slice is recorded as idle time (see the shaded area in figure 6). In the next time slice $[ t_2,t_3 ]$ , process 1 resumes its critical action. In our example, within its time slice process 1 calls send(s); it then continues its normal actions until the end of the time slice. In its turn, process 2 now finds the semaphore s free, and can perform a critical action.

We can see that the recorded idle time of figure 6 is clearly longer than the idle time in figure 5. But what is worse, in figure 5 the idle time is recorded for process 1, whereas in figure 6, the simulated case, the idle time is due to the waiting of process 2. If process 1 in figure 6 would call send(s) a fraction before $t_3$ instead of just after $t_2$, the idle time of process 2 would be the same. However, in case process 1 would call send(s) a fraction after $t_3$, the idle time of process 2 would be prolonged with a full time slice, making the simulation even worse. This implies that the simulation is also dependent on the time slice size.

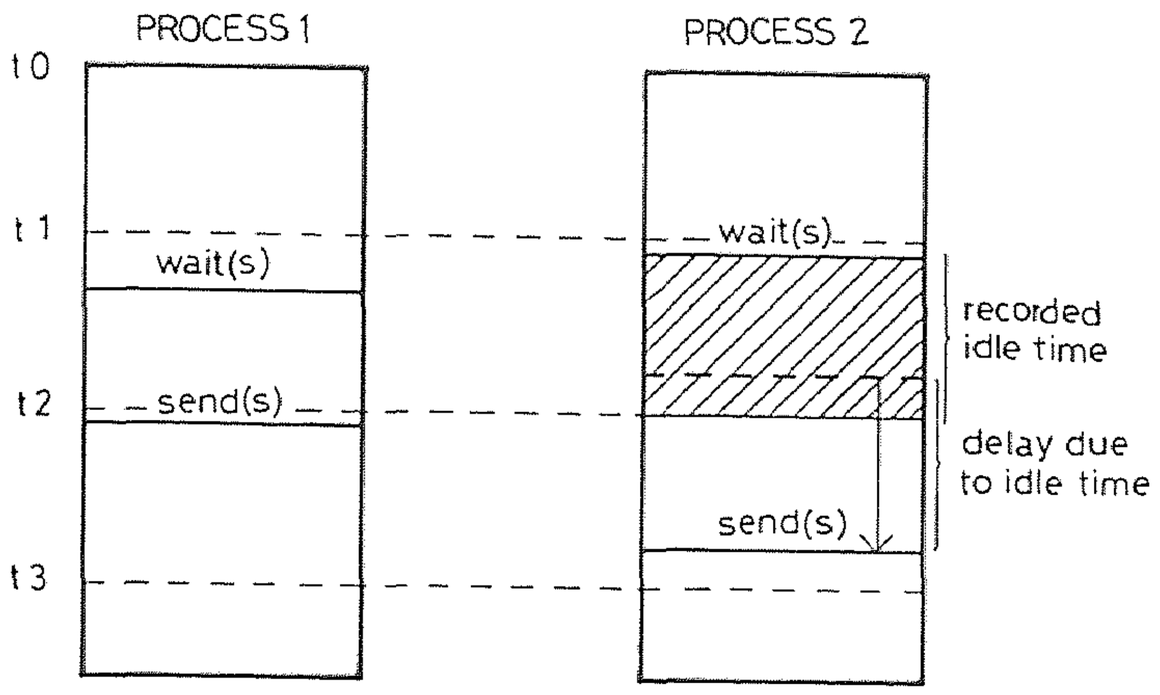


Figure 6. Simulation of two Parallel Processes with Process 1 first

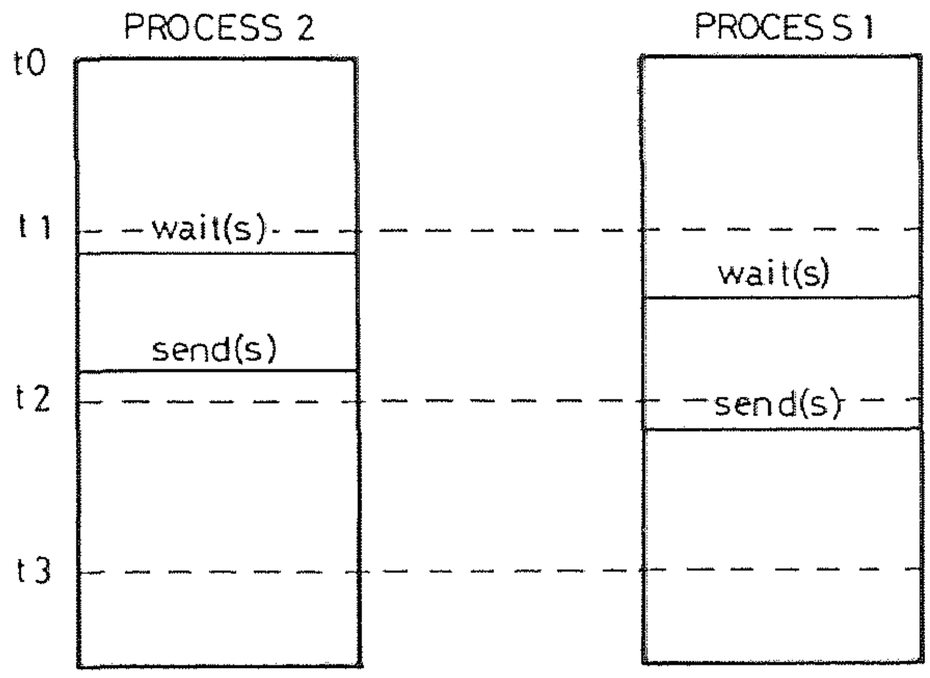


Figure 7. Simulation of two Parallel Processes with Process 2 first

From figure 6 the conclusion may not be drawn that the recorded idle times will always be longer than the real idle time. In figure 7 we have depicted a simulation of the parallel process of figure 5 again, but we have switched the order of processes with respect to figure 6. Looking at figure 7, we see that the simulation shows no recorded idle time at all. In process 2 the wait(s) and send(s) are in the same time slice, implying that the critical action performed between wait(s) and send(s) has no influence on process 1's critical action to be performed.

Since the method of a time slicer turned out not to yield reliable results, a new method (see section 5.3) was introduced [13].

## 5.3. Parallel Computer Simulator

To avoid the disadvantages of the time-slicer approach, another way of scheduling was developed in order to obtain more reliable estimates of the various processors' idle times.

The approach, named the Parallel Computer Simulator (PCS), is based on the allocation of an own time space for every process. So far, no shortcomings have been discovered in this approach (cf. Note at the end of this section). Since the PCS strategy results in a rather complex transfer of control, partly due to the presence of loosely coupled processes as well as independent processes, we shall elucidate its behaviour applied on the parallel processes given in figure 5.

The execution time (run time and idle time together) can be measured in absolute time units (abstime), indicating to what point the process has advanced. The abstime of each process, accordingly called abstime 1, abstime 2, etc., is initialized at zero. The general strategy of the scheduler is to select the first non-waiting process with the smallest abstime in the circular list. For our convenience we have placed some marks in the time space, e.g. $t_0 = 0$. Initially, abstime 1 and abstime 2 are both zero. The scheduler selects process 1 to be activated. The execution continues until wait(s) is called. The call wait(s) is a two-stage procedure. The first stage serves only to announce the "real" wait(s) call. In the first stage of the wait(s) procedure the scheduler is activated. According to its general strategy, process 2, now with the smallest abstime, is selected. Process 2 is executed until wait(s) is called, the first stage of which activates the scheduler again. The scheduler sees that abstime 2 is still smaller than abstime 1 and therefore (re-)activates process 2. The second part of its wait(s) call is executed and process 2 can perform its critical actions until send(s) is called. The procedure send(s), also built up of two stages, activates the scheduler before executing the "real" send(s) call. Since abstime 1 is smaller than abstime 2 the scheduler activates process 1. Process 1 proceeds to execute the "real" wait(s) procedure (the second stage of wait(s)) and finds the semaphore s occupied (in this instance, by process 2). Process 1 puts itself in a wait position and activates the scheduler ; meanwhile abstime 1 should increase as before (of course, run time and idle time are registered separately). Since only process 2 is non waiting, it is activated. It can execute the second stage of the send(s) call (the "real" send(s) call) and set the semaphore s free. With the information available, it is easy to establish the abstime 1 and so the idle time of process 1, and thereafter to change the state of process 1 from waiting to non-waiting.

The rest of process 2 as shown in figure 8 is not coupled with process 1 and, therefore, treated as an independent process ; process 2 may continue to its end, then the scheduler is activated. Since process 1 is non-waiting and has evidently the smallest abstime, it is activated and performs its delayed critical action. At the end of the critical action it passes the two-stage send(s) procedure via the intermediate actions of the scheduler as described above and finishes all its actions.

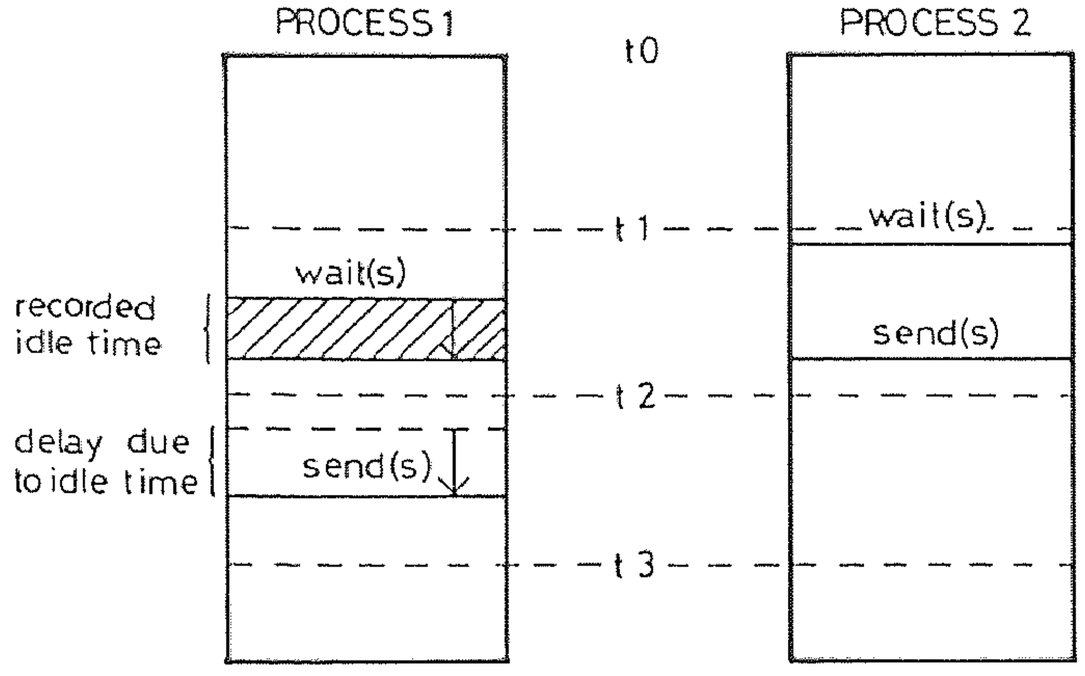


Figure 8. Two Parallel Processes and its Real Simulation

In figure 8 the shaded area, indicating the idle time of process 1, is identical in size to that of process 1 in figure 5. Furthermore, it is situated at the same moment of time. The example presented gives insights in the way in which complex cases with n loosely coupled processes are treated by the PCS. Although no formal proof has been provided that the PCS always performs the simulation correctly, it seems to us that the PCS appropriately simulates a parallel computer with n (loosely coupled) processes. Finally, we would like to remind the reader of the potential presence of independently acting processes. To deal with these processes, we have also adopted the techniques of implicit transfer of control by time slicing in the scheduling mechanisms. Considering the scope of this article we will not discuss this matter here, but refer to [13].

Note

Although the simulation has provided us with very hopeful results, we are happy to mention a last-minute result. In september 1985, the Philips Laboratory for Physics Research enabled us to run HYDRA on a real parallel machine (consisting of 2 to 15 68000 processors). The program had to be translated into the language C, a task performed in one week by J.C. Stoop and P.R. Varkevisser [11]. The results obtained by the parallel machine excellently fit the simulation results when using the PCS.

## 6. PERFORMANCE ANALYSIS OF THE PARALLEL COMPUTER SIMULATOR

In order to examine the impact of the different processes, we ran HYDRA with a varying number of RPs. At every run we let HYDRA deduce the same item in the flora-knowledge base ; in this examples below, it is the black pine tree. A special statistics program was built for analyzing the measurement data as they are produced by the PCS. The time is measured in multiples of n-hundreds of μs. To eliminate the influence of the human factor in the results, the answers to all the questions

which can be asked by HYDRA concerning the black pine tree are stored in a file accessible to the user interface.

In tables 1 and 2 we provide the run and idle times of the processes and the total idle time for each semaphore when HYDRA runs with one RP only. We note that the total of the run and idle times amounts to 58975 units. The differences in the RAM and DBM total times are due to the Room-Door mechanisms as introduced in section 3 : both managers will always have to wait until the "last" reader has left the Room. As expected, when all processes are running in parallel, the length of the time, the system needs to find a solution, is determined by the run time of the rule processor, by far the longest run time.

PROCESSES

| PROCESS | TOTAL RUN TIME | TOTAL IDLE TIME |
|---|---|---|
| Main Program | 11 | 58964 |
| Rule Agenda Manager | 4931 | 54133 |
| Question Agenda Manager | 1881 | 57094 |
| Data Base Manager | 4400 | 54620 |
| Rule Processor | 58049 | 926 |
| User Interface | 8810 | 50166 |

Table 1. Run and Idle Times with One Rule Processor

SEMAPHORES

| SEMAPHORES | ROOM | TOTAL IDLE TIME |
|---|---|---|
| Readers Door | Rule Agenda Room | 525 |
| Writers Door | Rule Agenda Room | 1432 |
| Empty Update Door | Update Room Rule Agenda | 52738 |
| Readers Door | Database Room | 259 |
| Empty Update Door | Update Room Database | 53876 |
| File Update Door | Update Room Database | 7 |
| Question Agenda Queue Nonempty | | 49716 |
| Get Door | Question Agenda Room | 445 |
| Empty Update Door | Update Room Question Agenda | 57091 |

Table 2. Idle Times of Semaphores with One Rule Processor

We can now proceed to consider how much time may be gained by using more than one rule processor. We provide the run and idle times of HYDRA when running with two, three, four, five, seven and fifteen rule processors in the tables 3, 4, 5, 6, 7 and 8 respectively. The idle times of the semaphores will not be given in these cases, a discussion thereof does not fall within the scope of this article ; for a detailed analysis, see [13]. In all tables 3 to 8, RAM and DBM times are longer than the total time of any rule processor due to the Room-Door mechanism. Table 3 shows that the use of two processors decreases the time it takes for HYDRA to solve the problem substantially, namely from 58975 to 43424. However, the idle times of the rule processors increase, resulting from the RPs' obliged waiting before the RAR's Door. The total run time of the RAM shows that it is more often updating the Rule Agenda than with one processor. This means that the RPs find the Door closed more often. This tendency is continued in table 4, where the solution time is further reduced to 39100 and the idle time of the processors further increased. Although the solution time is approximately the same in table 5, the load balance over all rule processors is not as good as in table 4, neither is the idle time as well-balanced as in table 4. In table 6, when HYDRA is running with five rule processors, we see the load balancing for the RPs improving and their idle times decreasing. Still, the solution time is slightly increasing even though there are more rule processors to do the job.

A further addition of rule processors worsens the situation, as can be observed in tables 7 and 8 ; the solution time increases slowly. Numbers reported in [13] even suggest that this increase is linear with the addition of rule processors. Starting from HYDRA with one rule processor we see that, the more rule processors are involved, the more work will have to be performed by the RAM. Once the optimal number of RPs has been passed, the run time of the RAM decreases, maybe since load balancing is a major factor to deal with. However, when as many as fifteen rule processors are in use, the run time of the RAM increases considerably. The reason is that the rule processors do not only search the fruitful branches of the search graph, but also enter blind alleys.

| PROCESS | TOTAL RUN TIME | TOTAL IDLE TIME |
|---|---|---|
| Main Program | 12 | 43412 |
| Rule Agenda Manager | 6275 | 37281 |
| Question Agenda Manager | 2319 | 41105 |
| Data Base Manager | 4514 | 38997 |
| Rule Processor | 36894 | 6530 |
| Rule Processor | 36598 | 6836 |
| User Interface | 9738 | 33687 |

Table 3. Run and Idle Times with Two Rule Processors

| PROCESS | TOTAL RUN TIME | TOTAL IDLE TIME |
|---|---|---|
| Main Program | 12 | 39088 |
| Rule Agenda Manager | 7358 | 31958 |
| Question Agenda Manager | 2248 | 37065 |
| Data Base Manager | 4605 | 34705 |
| Rule Processor | 28314 | 10786 |
| Rule Processor | 28665 | 10435 |
| Rule Processor | 26916 | 12184 |
| User Interface | 10187 | 28914 |

Table 4. Run and Idle Times with Three Rule Processors

| PROCESS | TOTAL RUN TIME | TOTAL IDLE TIME |
|---|---|---|
| Main Program | 10 | 39066 |
| Rule Agenda Manager | 7148 | 32064 |
| Question Agenda Manager | 2181 | 37028 |
| Data Base Manager | 4585 | 34621 |
| Rule Processor | 30490 | 8586 |
| Rule Processor | 28567 | 10509 |
| Rule Processor | 27701 | 11375 |
| Rule Processor | 29567 | 9513 |
| User Interface | 9825 | 29252 |

Table 5. Run and Idle Times with Four Rule Processors

| PROCESS | TOTAL RUN TIME | TOTAL IDLE TIME |
|---|---|---|
| Main Program | 13 | 39238 |
| Rule Agenda Manager | 6714 | 32677 |
| Question Agenda Manager | 2178 | 37210 |
| Data Base Manager | 4536 | 34849 |
| Rule Processor | 30264 | 8987 |
| Rule Processor | 29514 | 9737 |
| Rule Processor | 29899 | 9352 |
| Rule Processor | 30673 | 8585 |
| Rule Processor | 29873 | 9382 |
| User Interface | 9630 | 29622 |

Table 6. Run and Idle Times with Five Rule Processors

| PROCESS | TOTAL RUN TIME | TOTAL IDLE TIME |
|---|---|---|
| Main Program | 13 | 42485 |
| Rule Agenda Manager | 6539 | 36242 |
| Question Agenda Manager | 2037 | 40741 |
| Data Base Manager | 4311 | 38464 |
| Rule Processor | 31760 | 11012 |
| Rule Processor | 32059 | 10710 |
| Rule Processor | 32345 | 10153 |
| Rule Processor | 32427 | 10071 |
| Rule Processor | 31853 | 10655 |
| Rule Processor | 32442 | 10063 |
| Rule Processor | 32477 | 10025 |
| User Interface | 9199 | 33380 |

Table 7. Run and Idle Times with Seven Rule Processors

| PROCESS | TOTAL RUN TIME | TOTAL IDLE TIME |
|---|---|---|
| Main Program | 20 | 57205 |
| Rule Agenda Manager | 9403 | 48449 |
| Question Agenda Manager | 2050 | 55799 |
| Data Base Manager | 4281 | 53566 |
| Rule Processor | 34395 | 22830 |
| Rule Processor | 34426 | 22799 |
| Rule Processor | 34110 | 23115 |
| Rule Processor | 34615 | 22610 |
| Rule Processor | 34495 | 22730 |
| Rule Processor | 34725 | 22500 |
| Rule Processor | 34831 | 22394 |
| Rule Processor | 35202 | 22023 |
| Rule Processor | 34862 | 22363 |
| Rule Processor | 34613 | 22612 |
| Rule Processor | 34951 | 22274 |
| Rule Processor | 34884 | 22341 |
| Rule Processor | 34382 | 22843 |
| Rule Processor | 34674 | 22551 |
| Rule Processor | 35106 | 22119 |
| User Interface | 9288 | 47938 |

Table 8. Run and Idle Times with Fifteen Rule Processors

## 7. CONCLUSIONS

In this article we have established that simulation plays a prominent role when testing various AI techniques. In turn, the application of AI techniques allow a rather effective simulation of inadequate models by sheltering the decision making agents in a KBS. A remarkable result of our experiments is that n simulated parallel rule processors do not only improve the processing speed, but also influence the accuracy of the deductions, as mentioned in section 4. The HYDRA concept for parallel machines therefore imposes high demands on the consistency of the rule base as a whole.

Moreover it is shown that additional rule processors lead to an increase of deduction speed up to a certain point. We can conclude that the optimum number of rule processors is to be found between three and four, though it should be immediately added that the optimal number is dependent on the domain in question and the inference mechanism used. However, we remark that, e.g., in chess-tree searching the concept of each processor working at the same depth begins to fail beyond four processors [ 6 ]. Our experiences with HYDRA using more than four processors have proven that, with the present algorithm and the flora-knowledge base, the performance cannot be improved by adding more rule processors. The complexity of the search graph, determined by the coherence of the rules, may vary between domains, implying that other domains can have an other optimal number of processors.

*In fine*, we may conclude, as a consequence of the efforts mentioned in the note at the end of section 5, that simulation of a parallel machine on a sequential machine using the Parallel Computer Simulator provides an excellent testing ground for models of which the structure is hard to characterize.

## ACKNOWLEDGEMENTS

The authors wish to thank Ms. K.E. Börjars for the many valuable suggestions on the formulation of our results, described in successive versions, and for the scrutiny of the English text. Mr. J.C. den Hartog's contribution is acknowledged for placing the flora-knowledge base at our disposal. The competent technical assistance of Ms. J.W. Pesch and Ms. A. Gevaert, who did all the typing is also gladly recognized. Finally, we are grateful for Mr. J. Schoonewille's careful drawing of the figures, essential to the production of this publication.

## REFERENCES

[ 1 ] Ben-Ari, M., *Principles of Concurrent Programming*, Prentice Hall International, Englewood Cliffs, N.J.; London, 1982.

[ 2 ] Dijkstra, E.W., "Co-operating Sequential Processes", *Programming Languages* (ed. F. Genius), 1968, 43-112, Academic Press, London, New York.

[ 3 ] Groen, A., H.J. van den Herik, A.G. Hofland, N. Kasabov, E.J.H. Kerckhoffs and J.C. Stoop, "Parallelizing Knowledge-Based Systems: Expectations and Explorations", 1985. Report Delft University of Technology.

[ 4 ] Hayes-Roth, R., D.A. Waterman and D.B. Lenat (eds.), *Building Expert Systems*, Addison-Wesley Publishing Company, Inc., Reading, Mass., 1983.

[ 5 ] Herik, H.J. van den, "Het Gebruik van Kennis in Expert Systemen", *Expert Systemen* (eds. A. Nijholt, L. Steels), Academic Service, 's-Gravenhage (to appear in 1985). (Dutch language).

[ 6 ] Hyatt, R.M., "Parallel Chess on the Cray X-MP/48", *ICCA Journal*, 8, 2, 1985, 90-99.

[ 7 ] Klahr, P., "Artificial Intelligence Approaches to Simulation", *Proc. 1984 UKSC Conf.on Computer Simulation* (ed. D.J. Murray-Smith), 1984, 87-92, Publ. Butterworth, Bath, England.

[8] Lenat, D.B., "AM : An Artificial Intelligence Approach to Discovery in Mathematics as Heuristic Search", *Knowledge-Based Systems in Artificial Intelligence* (eds. R. Davis and D.B. Lenat), McGraw-Hill, New York, 1982.

[9] Ören, T.I. and B.P. Zeigler, "Concepts for advanced simulation methodologies", *Simulation*, 32, 3, 1979, 69-82.

[10] Rich, E., *Artificial Intelligence*, McGraw-Hill Book Company, Inc., New York, 1983.

[11] Stoop, J.C. and P.R. Varkevisser, *De implementatie van HYDRA op een parallelle machine*, Internal Report, TH Delft, 1985. (Dutch language).

[12] Vansteenkiste, G.C. and E.J.H. Kerckhoffs, "Informationbase Support in Simulation of Biological Systems", *Proc. of the 1984 UKSC Conference on Computer Simulation* (ed. E.J. Murray-Smith), 1984, 198-218, Publ. Butterworths, Bath, England.

[13] Varkevisser, P.R., *Measurements of Idle Times of the Parallel Processes in HYDRA*, Internal Report, Delft University of Technology, 1985 (in preparation)

[14] Winston, P.H., *Artificial Intelligence*, Second Edition, Addison-Wesley Publishing Company, Inc., Reading, Mass., 1984.

[15] Wirth, N., *Programming in Modula-2*, Third Edition 1985, Springer-Verlag, Berlin, Heidelberg, New York, 1982.