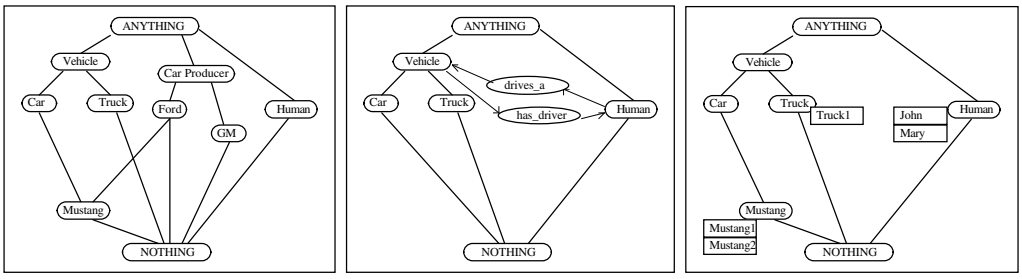
# Data Parallelism in Description Logics

Technische Universität Berlin, Franklinstr. 28/29, D-Berlin
As part of the thesis work of Frank Bergmann, fraber@cs.tu-berlin.de
Supervised by
Prof. Dr. -Ing. Walter Ameling, RWTH Aachen
o. Prof Mahr, TU-Berlin
J. Joachim Quantz, jjq@cs.tu-berlin.de
August 1994

## Abstract
Today both, knowledge representation systems (that are in some way similar to Expert Systems) and massive parallel computers have each reached a level of maturity which makes their real world use possible. This paper presents one possible approach to unify both areas to achieve a performance boost for the traditionally rather slow knowledge representation systems. In this project we consider computers based on the well established Transputer technology to parallelize the FLEX description logic systems, which belongs to the group of KL-ONE based languages. In contrast to the mainstream of parallel logic research we use coarse grain data parallelism for our implementation with the perspective to achieve high efficiencies on the given hardware architecture.

## Knowledge Representation
Based on a logical representation formalism, this system represent mechanisms which allow to deduce implicit knowledge and to detect inconsistencies in a knowledge base. The anterior system of FLEX, BACK has been widely used for several research projects such as machine translation or in the processing of medical data.
FLEX can be seen as a formal elaboration of the ideas underlying semantic networks and frames. The representation formalism is based on the idea of hierarchically organised knowledge in which information is inherited from general concepts to more specific ones. It also provides means for an internal structuring of concepts or objects which leads to horizontal connections: Objects contain slots whose fillers are known to be instances of other concepts.




## Flex Terminology
The three basic elements to represent world knowledge in FLEX are 'concepts', 'roles' and 'objects'. Concepts are something like classes in object oriented languages. Objects always are instances of some concept. A concept may have several roles attached. An object which is an instance of such a concept inherits these roles from the concept. Each role can have several other objects as 'role fillers'. These three elements are sufficient to express a broad variety of relations between objects. Additionally 'situations' present a 'chronological' order of concepts, objects and roles.

## FLEX Operation

The **class hierarchy** is implemented as a 'lattice' data structure: The top is represented by the built-in concept 'anything' (the most general) and the bottom by the concept 'nothing' (most special one). All user defined concepts are located inside the lattice between anything and nothing. 'Higher' concepts are more general than lower concepts. Lower concepts inherit properties from their (higher) parent concepts.

A **normalform** is the internal representation of all information about objects and concepts. It consists of a conjunctive and a disjunctive part and the operations 'normalise' and 'classify' are defined on it.

**Normalisation** is an operation on the normalform of some entity, when information is added. The new information becomes integrated into the existing normalform, and inconsistencies are detected. The normalise operation is atomic, and uses information only from the class hierarchy that does not change during this action.

**Classification** is an algorithm that determines the position of a normalform in the class hierarchy. It only uses the normalform and the class hierarchy to determine the position of the normalform.

Normalisation of one object can produce new information that affects other objects. This information has to be **propagated** to the related objects and potentially causes a normalisation there. This in turn may produce further information that has to be propagated back and so forth. This kind of 'Ping-Pong' interaction terminates only when a 'fix point' is reached. This fix point is always reached because FLEX has the very important quality of 'confluence', that is the independence of the result from the order of execution.

## Parallel Paradigm

Normalise, Classify and Propagate serve as a base for all consideration towards parallelism. The idea that results is to represent each object as a parallel process and to interpret propagation as message passing. This scheme is called 'communicating objects'. Features:

- the explored data independence promises an efficient parallel implementation.
- good availability of massive parallel computers that fit very well to the data parallelism.
- availability of a suitable parallel Prolog for Transputer systems.
- communication complexity increases slower than algorithm complexity.
- a locality principle, suggested by the grouping of objects in actual FLEX applications suggest a further reduction of overall communication complexity.
- we even expect synergetic effects due to the increase of efficiency if several propagations are processed at the same time.

## Conclusion

The proposed communicating objects implementation of FLEX is different to many other possible paradigms in a way that any central management institution is replaced by local knowledge of objects about their neighbours. Objects communicate directly with each other in a 'chaotic' way, that helps to avoid communication bottlenecks in a large network. Theoretical calculations arise expectations that this paradigm will achieve high efficiencies even on large massive parallel computers.
Brain Aid Prolog, a CSP Prolog for distributed memory machines exactly fits the needs of this paradigm and is used to implement Parallel FLEX on a Parsytec Multicluster II. A first version is expected for October 1994 and will be presented in the next report.