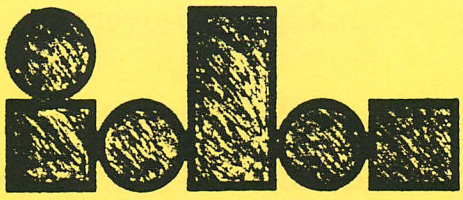
KB-RA-
E1

LiTH-IDA-R-88-05

ISSN-0281-4250



INSTITUTIONEN FÖR DATAVETENSKAP

UNIVERSITETET OCH TEKNISKA HÖGSKOLAN
LINKÖPING

Keeping and Forcing:
How to Represent Cooperating Actions

by

Christer Bäckström

UNIVERSITETSBIBLIOTEKET
88 -04- 11
LINKÖPING

RESEARCH REPORT
RKLLAB, March 1988

Postadress:
Institutionen för datavetenskap
Universitetet i Linköping och
Tekniska Högskolan
581 83 Linköping

Mailing address:
Department of Computer and
Information Science
Linköping University
S-581 83 Linköping, Sweden

Computer and Information Science Department
Linköping University

PhD theses:

* Available at: University of Microfilms Intl., 300 N. Zeeb Road, Ann Arbor, MI 48106, USA.

(*Linköping Studies in Science and Technology. Dissertations.*)

*No 77	**Östen Oskarsson:** Mechanisms of Modifiability in large Software Systems, 1982.
*No 94	**Hans Lunell:** Code Generator Writing Systems, 1983.
*No 97	**Andrzej Lingas:** Advances in Minimum Weight Triangulation, 1983.
*No 109	**Peter Fritzson:** Towards a Distributed Programming Environment based on Incremental Compilation, 1984.
*No 111	**Erik Tengvald:** The Design of Expert Planning Systems. An Experimental Operations Planning System for Turning, 1984.
No 155	**Christos Levcopoulos:** Heuristics for Minimum Decompositions of Polygons. 1987.
No 165	**James W. Goodwin:** A Theory and System for Non-Monotonic Reasoning. 1987.
No 170	**Zebo Peng:** A Formal Methodology for Automated Synthesis of VLSI Systems. 1987.

Licentiate of engineering Theses:

(*Linköping Studies in Science and Technology. Theses.*)

No 48	**Henrik Nordin:** On the Use of Typical Cases for Knowledge-Based Consultation and Teaching. 1985.
No 52	**Zebo Peng:** Steps Towards the Formalization of Designing VLSI Systems. 1985.
No 60	**Johan Fagerström:** Simulation and Evaluation of an Architecture based on Asynchronous Processes. 1985.
No 71	**Jalal Maleki:** ICONStraint, A Dependency Directed Constraint Maintenance System. 1987.
No 72	**Tony Larsson:** On the Specification and Verification of VLSI Systems. 1986.
No 73	**Ola Strömfors:** A Structure Editor for Documents and Programs. 1986.
No 74	**Christos Levcopoulos:** New Results about the Approximation Behaviour of the Greedy Triangulation. 1986.
No 104	**Shamsul I. Chowdhury:** Statistical Expert Systems - a Special Application Area for Knowledge-Based Computer Methodology. 1987.
No 108	**Rober Bilos:** Incremental Scanning and Token-based Editing. 1987
No 111	**Hans Block:** SPORT-SORT Sorting Algorithms and Sport Tournaments. 1987.
No 113	**Ralph Rönnquist:** Network and Lattice Based Approaches to the Representation of Knowledge. 1987.
No 118	**Mariam Kamkar, Nahid Shahmehri:** Affect-Chaining in Program Flow Analysis Applied to Queries of Programs. 1987
No 126	**Dan Strömberg:** Transfer and Distribution of Application Programs. 1987
No 127	**Kristian Sandahl:** Case Studies in Knowledge Acquisition, Migration and User Acceptance of Expert Systems. 1987

LiTH-IDA-R-88-05 ISSN-0281-4250

Keeping and Forcing: How to Represent Cooperating Actions

by

Christer Bäckström

Internet: CBA@IDA.LIU.SE
UUCP: enea!liuida!cba

Abstract: This report presents a formalism for representing action structures, i.e. complex systems of actions going on in sequence or in parallel and possibly interfering or interacting with each other. The same formalism supports all kinds of reasoning about action structures like e.g. planning, plan projection and plan observation. Actions are characterized with five conditions expressing how they relate to the world and to other actions. It is possible to express not only when two actions are allowed to occur in parallel, but also when they are required to co-occur. Such synchronization is implicitly forced by the definitions of the actions. The conditions can also be used to implicitly define hierarchies of actions which can be used both downwards, for hierarchical planning, and upwards, for plan observation.

Postadress:
Institutionen för datavetenskap
Universitetet i Linköping och
Tekniska Högskolan
581 83 Linköping

Mailing address:
Department of Computer and
Information Science
Linköping University
S-581 83 Linköping, Sweden

1. Introduction

This report presents a representation formalism for actions and action structures in particular. An action is something that happens in the world, and we won't distinguish between things that are brought about by an agent and things that happen without the direct participation of an agent, but simply use the word action for both cases. This means e.g. that both the occurence of an earthquake and the occurence of of having dinner will qualify perfectly well as actions. Among the most important characteristics of actions are that they have a duration in time, that they can go on in parallel and that they usually affect the world. Action structures mean more or less complex systems of actions going on in sequence and/or in parallel and possibly interacting and/or interfering with each other. We will for the moment informally define an action structure as a set of actions together with some further information regarding e.g. their relative time order.

Action structures arise in many important application areas for AI, like e.g. autonomous vehicles, automated manufacturing, robotics, natural language processing, office information systems, intelligent operating systems, reasoning about physical processes etc. An example from the first of these areas will follow later in this report.

It is desirable that a practical system incorporating reasoning about action structures is based on a solid theoretical ground. This has not always been true for systems developed so far, and it is sometimes unclear what is actually going on inside those systems. We think that this calls for an increased effort in research about action structures, and especially how to formally represent them in order to get a clearer understanding of what is going on. Such a formalism must also be flexible enough to support different kinds of reasoning about action structures, such as planning, plan projection, replanning, plan execution, plan monitoring and plan recognition. Most of these topics appear in most of the application areas mentioned above, so we think that it is important to be able to use the same formalism for all of them.

Planning is probably the one of the above topics that has been given the most attention in AI research, and it has been a hot topic for at least 20 years. Despite this, almost all of the work done has used the simplifying assumptions that actions have no duration in time and that all actions go on in sequence, and they do therefore not suffice for our purpose. A good summary of the state of the art in this kind of planning has been written by Chapman [Cha87].

During the past 5 years or so, there has been an increased interest in parallel actions, and some works in this area are discussed in section 8 of this report.

Some of the non-AI branches of computer science has been dealing with parallel actions for a long time in e.g. operating systems, shared data bases and concurrent programming languages. These applications do, however, almost alway use some kind of explicit synchronization or predefined order of execution for interfering actions, and they usually deal with a very limited set of possible actions.

The remainder of this report presents a formalism that can be used for all the kinds of reasoning mentioned above and where synchronization is implicit in the definition of actions.

2. Basic ideas

Most of the action structure formalisms presented so far in AI have been expressed in logic on the syntactical level, and the semantics has usually been left unattended, except for in the first order logic solutions. Although logic has been the standard tool for most research in knowledge representation, it is not obvious that it is a good or even sufficient tool for representing action structures and many other AI problems. This problem has been discussed by Drew McDermott, who in a recent paper [McDer87] questions whether logic is powerful enough for temporal reasoning. He suggests that a lot of temporal reasoning problems like e.g the frame problem might be impossible to solve with logic, but rather require model theoretic solutions.

The work presented in this report is an extension to an earlier formalism presented in [Bäc88], and which is an extension to a formalism presented by Erik Sandewall and Ralph Rönnquist in [San86]. Like Sandewall and Rönnquist, we will only define the semantics, and leave the question of language open, and we hope that this will lead to fewer problems than usually appear in logic formalisms for temporal reasoning. For further reading on this topic, see [San88]. Another feature with this approach is that there is a choice of what reasoning mechanism to use. An interesting idea would be to do reasoning by constraint propagation on the semantical level instead of logical inference on the syntactical level.

The basic ideas in the approach by Sandewall and Rönnquist are:

- An *action structure* is viewed as a set of actions, each of which has a start point and an end point, and where the set of all such points is partially ordered for time.
- *Partial state* descriptions are used, where each *'feature'* or proposition in a partial state description can have a definite value (e.g. a truth value, or some other value from a discrete range) or be undefined.
- Each action has a *pre-condition*, a *post-condition* and a *prevail-condition* associated with it, and all three of these are expressed as partial states. The pre-condition and post-condition characterize what must hold at the beginning and end of the action, respectively, and the prevail-condition characterizes what must hold for the whole duration of the action.

Although the pre- and post-conditions don't have to be different, they usually are and so expresses how the action changes the world. The prevail-condition, on the other hand, expresses what must be constant during the whole action, ussually because the action requires it to.

3. Keep-conditions

Although pre-, post- and prevail-conditions suffice to express when two actions are allowed to execute in parallel without interfering, they are not sufficient to express when they have to. This is important if two or more actions have to cooperate to succeed in achieveing a certain goal. This problem was observed already in [San86] and their formalism did not have the ambition of solving it. In their paper, the problem was, however, nicely exemplified by the 'parking scenario' which is defined as follows. The car in fig. 1 has the goal of parking parallel to the curb, and this problem can in simplified form be described by the following three actions:

Reverse:	Keep the car moving in the reverse direction
SteerLeft:	Keep the car's front wheels at an angle pointing left
SteerRight:	Keep the car's front wheels at an angle pointing right

The only action plan that will actually park the car as requested is to first do SteerRight, then do SteerLeft and to do Reverse in parallel to both the other actions (fig. 2). All these actions affect a common feature, namely the position of the car, but it is nevertheless necessary to do Reverse in parallel with the Steer actions. E.g. doing all three actions in sequence, would not produce the

desired result, but rather relocate the car to somewhere further back in the same lane as it started in.

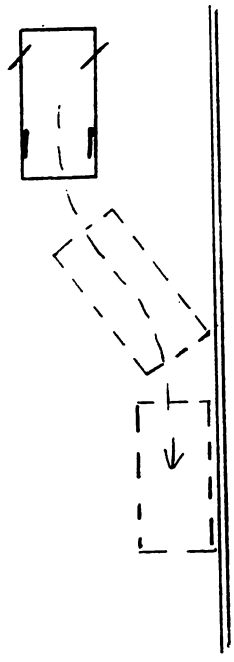


Fig. 1.

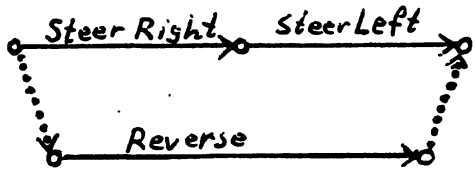


Fig. 2.

The solution we propose in this report is to associate with each action a fourth condition, the *keep-condition*, which states what will hold during the execution of the action as a cause of executing it. An action can now be implicitly forced to execute during the execution of another action by having a prevail-condition that is equal to the keep-condition of the latter action. A fifth condition called the force-condition will also be introduced to aid planning and plan observation.

The following three sections gives an informal presentation of the formalism. A formal presentation will not be given in this report since the formalism in [Bäc88] can easily be extended to cover the concept of keep-conditions.

4. Partial world states

It is assumed that the world under consideration can be modelled by a finite number of features. Every feature can take on any value from a corresponding predefined domain of values. The values of the feature domain can be any discrete range like e.g. the truth values domain {T, F}, the qualitative domain {+, 0, −} , the natural numbers, some subset thereof or a set of landmark values and the intermediate intervals. Each feature domain is also extended with the value u meaning unknown. A truth-valued domain will thus be very much like a three valued logic (see e.g. [Tur84]).

The cartesian product of the feature domains form the set of partial world states (they are partial because features are allowed to be undefined). By using partial states, we can express that some features are not affected by a certain action by leaving these features undefined. This means that two actions can execute in parallel without interfering if all features that are defined by the first action are undefined by the second action and vice versa.

5. Actions

With each action, we will associate five conditions expressing how the action relates to the world and to other actions, and all of these conditions are partial world states. The pre-condition expresses what must hold when the action starts and the post-condition expresses what will hold when the action terminates. The prevail-condition expresses what the action requires to hold during its occurence, and the keep-condition expresses what will hold during its occurence because of the action itself. The force-condition expresses that the action must be said to occur whenever this condition holds.

A simple action like flipping a switch can often be adequately described by having as pre-condition that the switch is open and as post-condition that the switch is closed. The prevail- and keep-conditions can be left undefined if one does not aim at a to detailed description, but a typical prevail-condition here could be that there is a suitable agent positioned by the switch.

Keep-conditions are mainly intended to describe contoller-like actions that keeps some feature at an otherwise unstable value. An example of such an action could be 'Keep the speed at 50 km/h' where the keep condition would be that the speed is 50 km/h. Another very important use of keep conditions

occurs in actions like 'Heat the water until it starts boiling'. Such an action could have as pre- and keep-condition that the water is not boiling and as post-condition that it is boiling. This means that the conditions defines the termination conditions for the action.

Prevail- and keep-conditions are also often used to express implicit synchronization between actions. Consider e.g. the scenario of changing gears in a car with a manual gearbox. Two actions are necessary to perform in this scenario and these are 'Keep the clutch pressed down' and 'Change the gear lever'. These two actions can however not be performed in arbitrary order, since the gear lever must be changed while keeping the clutch pressed down. Any other temporal ordering between these two actions would fail the goal and, in many cases, even destroy the gearbox. The way to solve this problem is to let the gear-changing action have as prevail-condition that the clutch is pressed down, and to let the clutch-pressing action have that same partial state as keep-condition. Since the prevail-condition of the first action is equal to the keep-condition of the other action there is only one valid temporal ordering between these actions, namely the one mentioned above.

Force-conditions, finally, are mainly used for planning or to infer abstract actions from an observed structure of physical actions. The force condition will, if defined, often be equal to the prevail-condition. Consider e.g. the scenario of road traffic. Assume that there are two physical actions Drive and SteerLeft, and that these two actions have as keep-conditions that the car is moving in the forward direction and that its front wheels are pointing left, respectively. An abstract action TurnLeft can now be defined with both prevail- and force-conditions defined as the state where the car is both moving in the forward direction and where its front wheels are pointing to the left. This means that the TurnLeft action occurs iff the car is both moving forward and steering to the left. This information can be used in two ways. A planner can e.g. produce a structure containing the TurnLeft action, and the physical actions really performing the turn are implicitly forced to occur by the definitions of the actions. A plan observer, on the other hand, can work the other way around and observe that the car is moving forward and steering left. It can then, from the definitions of the actions, infer that the car is turning to the left.

There is also a special class of actions called persistence actions and which have the same partial state as both pre-, keep- and post-condition. Such actions are used to carry persistence information in the structure, and

persistence actions are defined for every partial state that can persist. The persistence actions correspond to the Noop actions used in [San86].

6. Action structures

Action strucures can briefly be explained as a set of action occurences and a set of partially ordered time points, and where an action occurence is an actions together with begin and end time point from that set. The formal definition of action structures is principally the same as in [Bäc88], and we will only define the graphical representation here. An action structure can be visualized as an acyclic graph, where the time order is reflected by the direction of the arrows. Full arrows represent actions, dash-dotted arrows represent Persistence actions and dotted arrows represent extra time orderings not reflected by any action arrows. Two actions that must start at the same time originate from the same node in the graph, and analogously for actions that terminate at the same time. If one action starts immedaitely after the termination of another action, then it must originate from the node the first action terminates at. On the other hand, if some time is allowed to flow between these two actions, there must be a dotted arrow between them.

A very interesting property for action structures is admissability. The following definition of admissability is informal and almost correct, and a formal definition can easily be defined in the same way as is done [Bäc88].

- The post-conditions of two actions pointing to the same node must not have any defined features in common, and similarily for pre-conditions.
- If a certain feature is defined for some action pointing in to a node, then some action pointing out from that node must specify that same value for that feature, and vice versa.
- For every defined feature in the prevail-condition of an action, there must be an action or chain of actions temporally subsuming this action, and that has the same feature value in its keep-condition.
- For every chain of actions keeping a certain partial state, if there is an action having this state as force-condition, then there must be such an action occurence temporally subsuming this chain.

An action structure is also admissable if it can fulfill the above requirements by adding some number of persistence actions to it.

The interesting thing with admissable action structures is that they are guaranteed to execute correctly for any temporal ordering which is a strengthening of the original partial ordering.

7. The parking scenario, revisited

The car parking scenario will now be used as an example of action structures. We will, in order to keep the problem small and clear, ignore a lot of important aspects of the scenario like e.g. other parked cars and position along the street. The simplified problem can be described by four feature domains as follows (see fig. 3 for clarification):

- d: Distance from right rear wheel to curb. Possible values are {+,0,−,u}.
- v: Speed of car. Possible values {+,0,−,u}.
- Θ: Orientation of car relative to the curb. Possible values {+,0,−,u}.
- φ: Orientation of steering wheels relative to the length axis of the car. Possible values {+,0,−,u}.

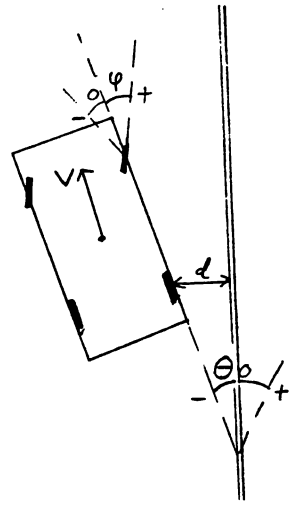


Fig. 3.

The following action occurences are needed:

SteerLeft:

Pre: φ=0

Keep: φ=−

Post: φ=0

SteerRight:

Pre: φ=0

Keep: φ=+

Post: φ=0

Reverse:

Pre: v=0

Keep: v=−

Post: v=0

ParkRight:

Force: v=−, φ=+

Prevail: v=−, φ=+

Pre: d=+, Θ=0

Keep: d=+, Θ=−

Post: d=0, Θ=−

ParkLeft:

Force: v=−, φ=−

Prevail: v=−, φ=−

Pre: d=0, Θ=−

Keep: d=0, Θ=−

Post: d=0, Θ=0

Only the features that are defined for a certain condition are given above, i.e. all other features have the value u. These actions are of course not the only actions possible, but they are the only ones needed for this scenario. The Reverse actions starts with the car not moving, keeps the car moving backwards during its execution and finally terminates with the car not moving. The SteerLeft and SteerRight actions work in the same way as Reverse but they affect the front wheel angle instead of the speed. The ParkRight action occurs iff the car is steering right and reversing at the same time and the orientation of the car is initially 0. As soon as the action starts, the orientation of the car will immediately change to a negative value and will remain so when the action terminates. The ParkRight action is furthermore defined so that it goes on as long as the car is distant from the curb, and terminates as soon as the right rear wheel has reached the curb. The ParkLeft action similarily occurs iff the car is steering left and reversing, and it keeps the distance to the curb at the 0 value and terminates as soon as the car is parallel to the curb.

We won't explicitly list all persistence actions here, but trust the reader to see which feature values can persist.

The initial world state is

$d=+, v=0, \Theta=0, \varphi=0$

i.e. the car is distant from the curb, not moving, parallel to the curb and has its steering wheels pointing straight forward. The goal state is similarily

$d=0, v=0, \Theta=0, \varphi=0$

i.e. exactly as the initial state except that the distance between the car and the curb is 0.

The only admissable action structure leading from the initial state to the goal state is shown in fig. 4. This is the maximally flexible plan that a planner could produce, since no other time constraints than the absolutely necessary ones are specified. Note e.g. that SteerRight may start before Reverse but never later than the start of Reverse. The chain ParkRight-ParkLeft and the action Reverse, on the other hand, must necessarily go on exactly in parallel for any correct execution of the plan.

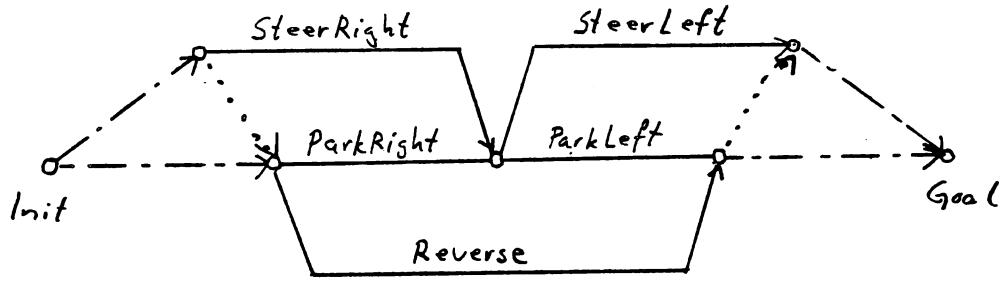


Fig. 4.

Note that this is only a very simplified scenario, and that a real problem would require a more detailed description. It would e.g. be necessary to have a more detailed coordinate representation, and to take other cars into account. Some of the actions should probably be represented as several actions. Special monitoring actions could e.g be used to monitor the distance to the curb, and the reversing action could be replaced by acceleration, retardation and speed-keeping actions. One could also have separate actions specifying the acceleration in their keep-conditions and other actions having the acceleration as prevail- and force-conditions and specifying the resulting speed in their keep-conditions.

8. Related works

Temporal logics are usually very powerful but also too complex for temporal reasoning in many AI applications. The most referenced early works on temporal logics for AI have been done by Drew McDermott [McDer82] and James Allen [All84], but neither of them gives formal descriptions of the underlying semantics. A recent paper by Yoav Shoham [Sho87] presents a termporal logic with a formal semantics, and which subsumes both Allens and McDermotts systems, while also being a cleaner and more powerful approach.

Both Mike Georgeff [Geo86] and Amy Lansky [Lan86] have presented formalisms for events and causality. In both formalisms, events can be simultaneous but have no duration in time so it is not possible to express that an event occurs during another event.

Edwin Pednault [Ped86] expresses structures of parallel actions in an ordinary sequential planning formalism. Functions are used to describe how parameters change over time, and the history can be described by a sequence of time points at which any such function is replaced. Our approach provides more advanced synchronization and supports actions hierarchies. Pednaults formalism, on the other hand, can represent continous change, which is probably indispensable to many AI applications.

An early work was carried out by Gary Hendrix [Hen73], but his system was mainly a plan execution simulator, and there was no formal description except for some pieces of program code. His actions were STRIPS-like operators, but augmented with two extra conditions. One of these expresses how some parameter will change over time during the action and the other expresses the conditions for termination.

Acknowledgements

Many thanks to Erik Sandewall and Ralph Rönnquist for invaluable comments, ideas and discussions.

References:

[All84] **Allen James:** "Towards a General Theory of Action and Time", AI Journal 23, pp. 123 - 154, North-Holland 1984.

[Bäc88] **Bäckström Christer:** "A Representation of Coordinated

Actions", Proc. SCAI'88 (Scandinavian Conference on AI), Tromso 1988.

[Cha87] **Chapman David:** "Planning for Conjunctive Goals", AI Journal 32, pp. 333 - 377, North-Holland 1987.

[Geo86] **Georgeff Michael:** "The Representation of Events in Multiagent Domains", Proc. AAAI-86, Philadelphia 1986.

[Hen73] **Hendrix Gary:** "Modeling Simultaneous Actions and Continous Processes", AI Journal 4, pp. 145 - 180, North-Holland 1973

[Lan86] **Lansky Amy:** "A Representation of Parallel Activity Based on Events, Structures, and Causality", SRI Technical Note 401, SRI 1986.

[McDer82] **McDermott Drew:** "A Temporal Logic for Reasoning About Processes and Plans", Cognitive Science 6, pp. 101 - 155, 1982.

[McDer87] **McDermott Drew:** "AI, Logic and the Frame Problem",, Proc. Workshop on the Frame Problem, Lawrence 1987, pp. 105 - 118.

[Ped86] **Pednault Edwin:** "Formulating Multiagent Dynamic-World Problems in the Categorical Planning Framework", Proc. Workshop on Reasoning About Actions Plans, pp. 47 - 82, Timberline 1986.

[San86] **Sandewall Erik, Rönnquist Ralph:** "A Representation of Action Structures",, Research Report LITH-IDA-R-86-13, Linköping University 1986. Also in Proc. AAAI-86, Philadelphia 1986.

[San88] **Sandewall Erik:** "Formal Semantics for Reasoning about Change with Ramified Causal Minimization", Internal Department Report, Dept. of Computer Science, Linköping University 1988.

[Sho87] **Shoham Yoav:** "Temporal Logics in AI: Semantical and Ontological Considerations", AI Journal 33, pp. 89 - 104, North-Holland 1987.

[Tur84] **Turner Raymond:** "Logics for Artificial Intelligence", Chapter 3, Ellis Horwood, West Sussex 1984.

Avdelning, institution, fakultet
Division, department, faculty

Department of Computer and
Information Science
Linköping University

ISBN:

ISSN: ISSN-0281-4250

Rapportnr:
Report no: LiTH-IDA-R-88-05

Upplagans storlek:
Number of copies:

Datum:
Date: March 1988

Projekt:
Project:

Titel:
Title: Keeping and Forcing: How to Represent Cooperating Actions

Författare:
Author: Christer Bäckström

Uppdragsgivare:
Commissioned by:

Dnr.:
Call no:

Rapporttyp:
Kind of report:

- ☐ Examensarbete/Final project
- ☐ Delrapport/Progress report
- ☐ Reserapport/Travel report
- ☐ Slutrapport/Final report
- ☒ Övrig rapport/Other kind of report

Rapportspråk:
Language:

- ☐ Svenska/Swedish
- ☒ Engelska/English
- ☐ _____

Sammanfattning (högst 150 ord):
Abstract (150 words):

This report presents a formalism for representing action structures, i.e. complex systems of actions going on in sequence or in parallel and possibly interfering or interacting with each other. The same formalism supports all kinds of reasoning about action structures like e.g. planning, plan projection and plan observation. Actions are characterized with five conditions expressing how they relate to the world and to other actions. It is possible to express not only when two actions are allowed to occur in parallel, but also when they are required to co-occur. Such synchronization is implicitly forced by the definitions of the actions. The conditions can also be used to implicitly define hierarchies of actions which can be used both downwards, for hierarchical planning, and upwards, for plan observation.

Nyckelord (högst 8):
Keywords (8):

Bibliotekets anteckningar

A Selection of Previous Research Reports.

LiTH-IDA-R-87-22 **Lars Ahrenberg, Arne Jönsson:** An Interactive System for Tagging Dialogues. To appear in *Proc. of 14th Conference of the Association for Literary and Linguistic Computing*, Göteborg 1-5 June, 1987.

LiTH-IDA-R-87-21 **Harold W. Lawson, Jr.:** Challenges and Directions in Computers and Education.

LiTH-IDA-R-87-20 **Krzysztof Kuchcinski, Zebo Peng:** Parallelism Extraction from Sequential Programs for VLSI Applications. This paper is to appear in *Microprocessing and Microprogramming, the Euromicro Journal*, 1988.

LiTH-IDA-R-87-19 **Zebo Peng:** Construction of Asynchronous Concurrent Systems from their Behavioral Specifications. Also in *Proc. of the 10th World Computer Congress*, Dublin, Ireland, Sept. 1-5, 1986.

LiTH-IDA-R-87-18 **Henrik Nordin:** Reuse and Maintenance Techniques in Knowledge-Based Systems.

LiTH-IDA-R-87-17 **Tony Larsson:** Specification and Verification of VLSI Systems Actional Behaviour This is a close version of a paper presented at *the 8th international conference on Computer Hardware Description Languages, CHDL*, 87.

LiTH-IDA-R-87-16 **Shamsul I. Chowdhury:** State of the Art in Statistical Expert Systems. Also in *Proc. of the Conference on Expert Systems and their Applications*, Avignon, May 1987.

LiTH-IDA-R-87-15 **Nils Dahlbäck:** Kommunikation med datorer i naturligt språk - vad är det och vem behöver det?

LiTH-IDA-R-87-14 **Peter Fritzson:** Window System Architectures - an Overview. Also presented at *A Tutorial on Software Development Environments*, Linköping University, Nov 24-25 1986.

LiTH-IDA-R-87-13 **Mats Wirén:** A Comparison of Rule-Invocation Strategies in Context-Free Chart Parsing. Also in *Proc. of the Third Conference of the European Chapter of the Association for Computational Linguistics*, Copenhagen, April 1—3, 1987.

LiTH-IDA-R-87-11 **Sture Hägglund, Christer Hansson, Tomas Sokolnicki:** Knowledge-Based Training of Case Management Routines and Emergency Procedures. Also in *Proc. of the 3rd Int. Conf. on Expert Systems*, London, 1987.

LiTH-IDA-R-87-10 **Andrzej Lingas:** On Parallel Complexity of the Subgraph Isomorphism Problem.

LiTH-IDA-R-87-09 **Andrzej Lingas, Marek Karpinski:** Subtree Isomorphism and Bipartite Perfect Matching are Mutually NC Reducible.

LiTH-IDA-R-87-08 **Christos Levcopoulos, Andrzej Lingas, Jörg-R. Sack:** Nearly Optimal Heuristics for Binary Search Trees with Geometric Generalizations. Also in *Proc of ICALP'87*, Karlsruhe, July 1987.

LiTH-IDA-R-87-07 **Peter Haneclou:** A Formal Approach to Reason-maintenance Based on Abstract Domains.

LiTH-IDA-R-87-06 **Johan Hultman:** COPPS - A Software System for Defining and Controlling Actions in a Mechanical System.

LiTH-IDA-R-87-05 **Christer Bäckström:** Logical Modelling of Simplified Geometrical Objects and Mechanical Assembly Processes.

LiTH-IDA-R-87-03 **Mikael R.K. Patel:** A Threaded Interpretive Language Supporting Programming in the Large. Also in *Proc. of The sixth Rochester Forth Conference*, University of Rochester, Rochester, New York, June 11-14, 1986.

LiTH-IDA-R-87-02 **Rober Bilos:** A Token-Based Syntax Sensitive Editor. Also presented at the *Workshop on Programming Environments - Programming Paradigms*, Roskilde, Denmark, October 22-24, 1986.

LiTH-IDA-R-87-01 **Wlodzimierz Drabent:** Do Logic Programs Resemble Programs in Conventional Languages? Also in *Proc of the Fourth IEEE Symposium on Logic Programming*, San Francisco, Aug. 31- Sept. 4, 1987.

LiTH-IDA-R-86-44 **Erik Sandewall et al.:** Department of Computer and Information Science Annual Research Report 1986.

The Department of Computer and Information Science
Linköping University

organizes undergraduate and graduate studies in *Computer Science*, *Telecommunication and Computer Systems*, and *Administrative Data Processing*. Research activities have an emphasis on advanced software technology and computer systems design and are organized in a number of research laboratories:

- *ACTLAB - Laboratory for Complexity of Algorithms*, which is concerned with the design and analysis of efficient sequential and parallel algorithms, and complexity theory, especially in the areas of computational geometry, data structures on bounded domains and graph algorithms.
- *AIELAB - Artificial Intelligence Environments Laboratory*, which conducts research on representation and manipulation of knowledge, problem solving techniques and expert systems with mechanical engineering applications.
- *ASLAB - Application Systems Laboratory*, which studies design of advanced support systems for interactive use of computers, including tools for automated construction of applications software.
- *CADLAB - Laboratory for Computer-Aided Design of Electronics*, which concentrates its research activities around tools for integrated development of hardware and software, graphics-based modelling and simulation techniques.
- *LIBLAB - Laboratory for Library and Information Science*, which studies methods for access to documents and the information contained in the documents, concentrating on catalogs and bibliographic representations, and on the human factors of library use.
- *LOGPRO - Laboratory for Logic Programming*, which concentrates its research activities around foundations of logic programming, relations to other programming paradigms and methodology.
- *NLPLAB - Natural Language Processing Laboratory*, which conducts research related to the development and use of natural language interfaces to computer software.
- *PELAB - Programming Environments Laboratory*, which works with design of tools for software development, specific functions in such tools and theoretical aspects of programs under construction.
- *RKLLAB - Laboratory for Representation of Knowledge in Logic*, which covers issues and techniques such as non-monotonic logic, probabilistic logic, modal logic and truth maintenance algorithms and systems.

Research Reports 1987 and 1988

LiTH-IDA-R-88-05	**Christer Bäckström:** Keeping and Forcing: How to Represent Cooperating Actions.
LiTH-IDA-R-88-04	**Wlodek Drabent, Simin Nadjm-Tehrani, Jan Maluszynski:** Algorithmic Debugging with Assertions.
LiTH-IDA-R-88-03	**Staffan Bonnier, Jan Maluszynski:** Towards a Clean Amalgamation of Logic Programs with External Procedures.
LiTH-IDA-R-88-02	**Ulf Nilsson:** Inferring Restricted AND-Parallelism in Logic Programs using Abstract Interpretation.
LiTH-IDA-R-88-01	**Jacek Leszczylowski, Staffan Bonnier, Jan Maluszynski:** Logic Programming with External Procedures: Introducing S-Unification. A shorter version is to appear in *Information Processing Letters*, North Holland.
LiTH-IDA-R-87-26	...ing System to Control
LiTH-IDA-R-87-24	

LINKÖPINGS UNIVERSITET
30580 007212462