Simulation of Hybrid Circuits in Constraint Logic Programming

Thomas Graf and Pascal Van Hentenryck
European Computer-Industry Research Centre, ArabellastraBe 17, D-8000 Munchen 81
Claudine Pradelles and Laurent Zimmer
Avions Marcel Dassault-Breguet Aviation, 78, Quai Marcel Dassault, F-92214 Saint-Cloud

Abstract

This paper presents LOGISIM, a CAD tool to simulate the temporal behaviour of hybrid circuits containing electro-mechanical, electro-hydraulic, hydro-mechanic, and digital control devices. LOGISIM combines the advantages of both qualitative and quantitative reasoning by producing a high-level description (discrete states) of the circuit behaviour while reasoning at the quantitative level (physical values). In addition, device models in LOGISIM follow a particular description methodology proposed to avoid introducing an artificial computational complexity in the simulation. LOGISIM is fully implemented in the constraint logic programming language CHIP. The constraint-solving techniques of CHIP used in LOGISIM, i.e. an incremental decision procedure for linear constraints over rational numbers, consistency techniques on domain-variables and conditional propagation, are all necessary to solve the problem efficiently. LOGISIM has been applied successfully to real-life industrial circuits from aerospace industry in the ELSA project and clearly demonstrates the potential of this kind of tool to support the design process for these circuits.

1 Introduction

Many circuits in aerospace industry combine hybrid components such as electro-mechanical (e.g., relays), electro-hydraulic (e.g., electro-distributors), and hydro-mechanic (e.g., jacks) devices, together with control digital components (e.g., numeric commands). The circuit of landing gear and trap of a fighter aircraft is an example of such circuit combining these elements.

There is presently a lack of tools supporting the design and analysis of such circuits. To perform usual tasks such as simulation or troubleshooting, engineers often resort to empirical methods or try to adapt tools developed for other kinds of circuits (e.g., SPICE [8]). Unfortunately the information produced by these quantitative simulators is of very low level and engineers have to extract information relevant for their own purposes. It follows that designing and analysing hybrid circuits are costly and time-consuming activities, which deserve better support.

The ELSA project [9] has been initiated for this very reason and aims at providing engineers with an integrated environment for hybrid circuits. Inside this environment, engineers should be able to describe circuits and perform numerous activities such as simulation, diagnostic, test generation, and fault analysis.

This paper presents LOGISIM, the simulator of ELSA. It aims at presenting the simulation results at a sufficiently abstract level to deduce the functional behaviour of the circuit while reasoning with a sufficient accuracy to capture the physical behaviour of the circuit. Devices are modelled in terms of states representing significant abstractions of their behaviour (e.g., a light bulb "on", "off" or "blown out"). Accuracy is achieved by defining each state in terms of both quantitative and qualitative constraints. For instance, the light bulb model contains a constraint stating that its state is "on" if there is enough current. LOGISIM receives as input a description of the circuit and a set of actions (e.g., the pushing of a button or the closing of a switch). It produces the successive states reached by the circuit and reports anomalies such the blowing of components, oscillations and ambiguities in the circuit description.

LOGISIM is implemented in the constraint logic programming language CHIP[4]. CHIP provides both an adequate formalism for describing hybrid circuits and the constraint-solving techniques necessary to simulate them efficiently, i.e.,

- an incremental decision procedure for linear constraints over rational numbers [4];
- consistency techniques on domains-variables [14; 12; 13];
- conditional propagation techniques.

Implementing LOGISIM in a conventional language would have required a programming effort orders of magnitude greater.

LOGISIM has been applied to real-life circuits including hundreds of components like the landing gear and trap of a fighter aircraft.

The rest of the paper is organised as follows. Section 2 gives an overview of LOGISIM. Section 3 discusses the implementation of the device models. Section 4 illustrates LOGISIM on a particular example and gives execution results on real-life circuits. Section 5 discusses related research and the last section draws the conclusions of the paper.

2 Overview of LOGISIM

LOGISIM consists of (1) a library of device models and (2) a scheduler responsible for directing the simulation. The circuit description is expressed in terms of the device models and used by the scheduler to simulate the circuit behaviour.

2.1 Device Models

Any device model consists of

- *a static pari* defining the possible states of the device.
- *a dynamic part* defining the temporal aspects of the device behaviour.

The behaviour of the device is described in terms of a finite set of states, each of which is defined through linear constraints on the physical values of the device. Device states include not only working states (e.g. a light bulb is "on") but also faulty states (e.g. a light bulb is "blown out"). The faulty states[1] are of primary importance to report anomalies which might occur in the circuit. The constraints defining a device state can be classified into constraints defining

- physical laws applying to the state (e.g. Ohm's laws);
- conditions on the device physical values proper to the state (e.g. the current in one port is higher than a threshold value).

In general, constraints of the first type are equations while constraints of the second type are inequalities.

The dynamic part defines *temporal aspects* of the device behaviour by defining

- the possible *transitions* between the device states;
- the *events* implied by the transitions; an event is a demand to fix up, or to restrict the possible values of, the state of a device.

LOGISIM distinguishes between two types of transitions.

- *internal transitions:* transitions which occur because of component values, e.g. the current present at the ports of a device. A typical example is the transition from "on" to "off" for a light bulb.
- *external transitions:* transitions which occur because of events.

Note that the device models in LOGISIM has to be context-free since they will be used for all other activities of the ELSA project. This requirement is satisfied by meeting criterias put forward by Qualitative Physics [2] such as *no function in structure* and *locality.*

We now illustrate these principles on two examples: a light bulb and a relay. We make use of state diagrams where states are represented by circles, internal transitions by thin arrows and external transitions by thick arrows. It is assumed that a state can be its own successor.

[1] The notion of faulty states is conceptual. These states are perfectly valid for LOGISIM and are not handled differently from working states.

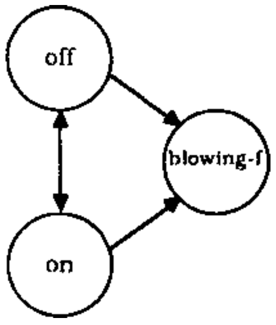

Figure 1: Transition graph and state definitions for a light bulb

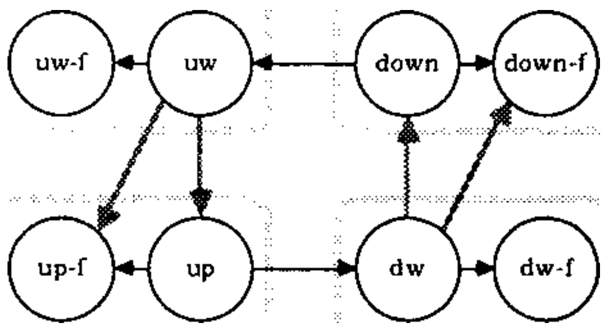

Figure 2: Transition graph of a relay

The behaviour of the light bulb is approximated through three different states: on, off and blowing_f. The transition graph and the state definitions are shown in figure 1. The states shares two constraints expressing Kirchhoff's current law and Ohm's law. They differ by their respective constraints on the current flowing through ports 1 and 2.

A relay with six ports and one anchor is modelled by 8 states. The working states are

- up: there is no current at ports 1 and 2 and the anchor connects ports 5 and 6;
- down: there is a current flowing in ports 1 and 2 and the anchor connects ports 3 and 4;
- dw (downward): there is a current flowing in ports 1 and 2 and the anchor does not connect anything. This is a transition state between up and down;
- uw (upward) : there is no current flowing in ports 1 and 2 and the anchor does not connect anything. This is a transition state between down and up;

The faulty states up_f, down-f, uw_f and dw_f handle erroneous behaviours of the relay (e.g. blowings of components, oscillations).

The transition graph (see figure 2) has both internal and external transitions. The two internal transitions generate events that will result in external transitions.

The constraints defining these states are very similar to those of the light bulb and the reader should have no difficulty to define them.

```
schedule(Circuit,Agenda,State) :-
    empty(Agenda).

schedule(Circuit,Agenda0,State0) :-
    notempty(Agenda) ,
    impose_trans_constraints(State0,StateN) ,
    apply_first_event(Agenda0,StateN,Agenda,Now) ,
    compute_newstate(Circuit,StateN) ,
    update_agenda(State0,StateN,Agenda,AgendaN,Now),
    print_and_report(StateN,Now),
    schedule(Circuit,AgendaN,StateN).

compute_newstate(Circuit,State) :-
    Call =..  [Circuit|State],
    Call,
    labeling(State).
```

Figure 3: A meta program for the scheduler

2.2 The Scheduler

The scheduler makes use of an agenda which contains the current set of events to carry out. Events are the smallest relevant units of the simulation. Each event in the agenda is characterised by a device identification, a set of possible states and the time at which the action takes place. Initially the agenda contains the set of external events. During the simulation, state changes of some devices can introduce new internal events to be executed at a later step.

The scheduler behaviour is defined by the successive application of the following steps:

- impose the constraints implied by the transition graph;
- remove the first event from the agenda and apply it;
- compute the new circuit state;
- update the agenda;
- report possible anomalies;

until the agenda is empty.

The scheduler is a meta-program whose definition is shown in figure 3. The first clause defines the halting condition. The second clause is the core of the scheduler.

The predicate impose_trans.constraints constructs the new state skeleton and imposes the transition constraints. The state skeleton is a list of domain variables, one for each device. Each variable has in its domain the set of possible states of the device. The transition constraints prevent the device from receiving a state that cannot be reached from the current state using the transition graph.

The predicate apply_first_event removes the first event in chronological order from the agenda and applies it to the state skeleton.

The predicate compute_newstate is the core of the simulator. It computes a new consistent circuit state. It constructs the call to the circuit and executes it. This sets up the constraints implied by the devices and the connections. The labeling procedure is then used to assign a state to each device. The way this computation is achieved is the topic of the next section.

The predicate update_agenda adds to the agenda new events that might have been generated by some transitions.

The last goal in the body is a recursive call to the scheduler with the new agenda and the new state.

3 Device Models

3.1 Principles

Given a circuit state, a simulation step consists in computing the next state in time. This can be seen as a constrained search problem. The constraints are defined by the device models and the transitions and a solution is the assignment of states to the devices which satisfies the constraints. Three principles for the device models have been defined to solve this problem efficiently.

First, constraints have to be taken into account as soon as possible. Consider the light bulb example. Since the first two constraints are shared by all states, they can be stated once for all without knowing the state of the device. This argument can be generalized. As soon as a device can only be assigned a subset of the possible states, the constraints common to all these states can be taken into account immediately.

Second, the search space is not made up by the device states but by the topologies of the devices. All physical values throughout the circuit can be computed as soon as the circuit topology is fixed. This idea can be used to classify the devices and to identify those which really need choices.

Definition 3.1 *A transition is* weak *if it does not change the device topology. Otherwise it is* strong.

Definition 3.2 *A model is an* S-model *if it does not contain strong internal transitions. Otherwise it is a* G-model.

For instance, the light bulb was modelled by an S-model while the relay was modelled by a G-model. When computing the new state, only the G-models have to be considered for the choices.

Finally device models should include value-to-state constraints to ensure that all information is deduced as soon as possible. Value-to-state constraints restrict the possible states of the device as soon as some conditions on the component values are satisfied. For instance, when $|I1| < It$, the state of the light bulb must be off.

All three techniques allow to avoid redundant work and to reduce the search space significantly.

3.2 Example

Figures 4 depicts the implementation of the light bulb. The first two constraints are constraints expressing Kirchhoff's current law and Ohm's law. The remaining goal defines the value-to-state constraint. It makes use of conditional propagation, an important constraint-solving technique of CHIP. Declaratively, it is a simple *if_then-else* construct.

```
light_bulb(Name,State,[R,It,Im],[I1,U1,I2,U2]) :-
   I1 + I2 = 0,
   U1 - U2 = R * I1,
   if I1 ≥ It then
      State ≠ off,
      if I1 ≥ Im then
         State = blowing_f
      else
         State = on
      endif
   else if I1 > -It then
      State = off,
   else
      State ≠ off,
      if I1 ≥ -Im then
         State = on
      else
         State = blowing_f
      endif
   endif
```

Figure 4: A data-driven implementation of the light bulb

if *CONDITION* then *GOAL-1* else *GOAL-2* endif

Procedurally, it provides an efficient demon-driven mechanism. It is handled in the following way. For any allowed condition, CHIP has at his disposal a procedure able to decide

- if the condition is always true or always false for all instances of the condition.
- if the condition is true for some instances and false for some others.

Therefore facing such a constraint, CHIP uses a procedure to evaluate the condition. If the condition is true, GOAL-1 is executed. If it is false, GOAL-2 is executed. Otherwise the if_jthen_else construct delays waiting for more information. Any constraint on domain-variables and rational terms can make up an allowed condition. Conditional propagation enables to deduce the state of the device from the current flowing through ports 1 and 2. Note also the place where the constraint

State off

is set up. It illustrates the first principle, i.e., state the constraints as soon as possible. In this particular example, the constraint is set up even if the particular state is not yet fully defined. Its effect is to remove the state off as soon as either II > It or II < -It- Finally it is interesting to note that the state definitions contain both quantitative and qualitative constraints.

3.3 Refining Device Models

In LOGISIM, the search space is defined by the the circuit topologies accessible from the current circuit state. Since real circuits are purely deterministic, the combinatorial aspect of the problem must be artificial and depends only on the way device models are designed. The question is then

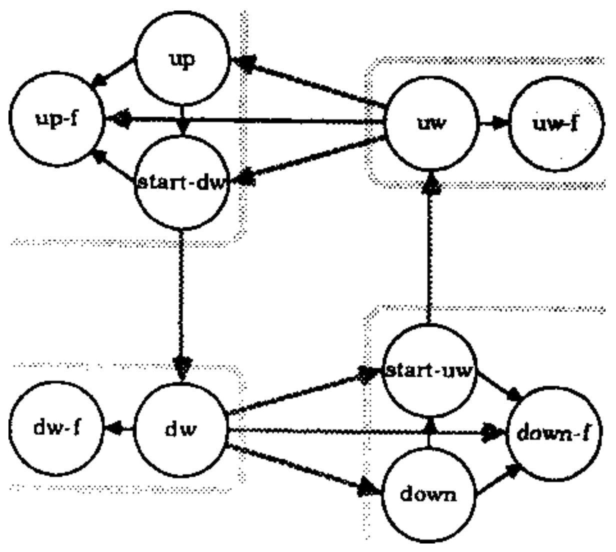

Figure 5: New transition graph for a relay

How to keep an appropriate level of abstraction without introducing an artificial complexity ?

The answer to this question falls back once again to the importance of the topology. At some point of the simulation, G-models still have several possible states; many of them correspond to different device topologies and can be reached from the current state by internal transitions. It follows that cutting down the number of possible strong internal transitions for a particular device directly reduces the size of the search space. This amounts to transform G-models into S-models by adding new intermediary states. This transformation is by no way artificial; it really captures some states of the devices which were not covered by the previous models and which represent important transition steps.

In implementation terms, transforming a G-model into an S-model means that strong transitions are only achieved through external transitions. Hence the simulator is sequentialized by transforming backtracking during the search of a new state into iteration through the scheduler.

Applying this idea to the relay leads to the transition graph depicted in figure 5. Two new states are introduced:

- s-uw: this state is reached from the state down when not sufficient current is available in port 1.
- s-dw: this state is reached from the state up when sufficient current is available in port 1.

The state s-uw (resp. s-dw) has the same topology as down (resp. up). Moving from down (resp. up) to s-uw (resp. s-dw) generates a new event in the agenda specifying that the device will reach the state uw (resp. dw) after a certain delay.

As it can be seen from the above diagram, topological changes can only occur due to an event in the agenda.

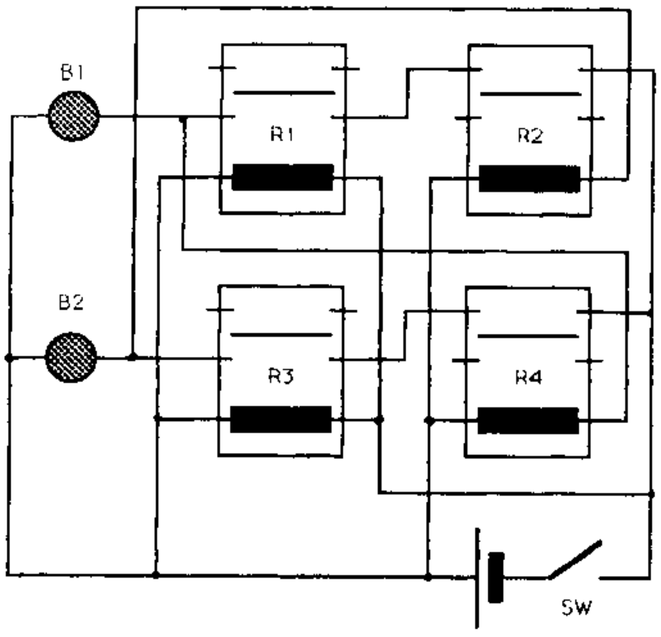

Figure 6: A simple electromechanical circuit

As a consequence, the topology of the relay is known at any simulation step.

Some device types might be difficult to model by an S-model. In that case, a G-model can be defined and used inside LOGISIM.

4 Results

LOGISIM has been tested on a variety of examples ranging from small to real-life hybrid circuits. Real-life circuits LOGISIM has been applied to include a controller for the landing gear and trap of a fighter aircraft and the command of a machine tool. The first circuit is fully hybrid since it contains electro-mechanical, electro-hydraulic and hydro-mechanic devices. It contains 57 components from 15 different device types and 112 connections. The second circuit is a machine tool containing more than hundred components. Experimental results show that the time for a simulation step is constant for a given circuit and linear in the number of components. For the above real-life circuits, it takes around 1.3 seconds for a simulation step on a SUN-3 using the CHIP interpreter.

To illustrate the overall approach of LOGISIM, a simple example is presented. Figure 6 describes the circuit under study. Assume now that initially all the relays are in the up position and that the switch is off. If the set of events to carry out contains the closing and the opening of the switch, LOGISIM will successively produce the states for the circuit shown in figure 7. LOGISIM also produces any quantitative information required by the

T	SW	R1	R2	R3	R4	B1	B2
1	off	up	up	up	up	off	off
2	on	s-dw	s-dw	up	up	off	off
3	on	s-dw	dw	up	up	off	off
4	on	dw	dw	up	up	off	off
5	on	dw	down	s-dw	up	off	on
6	on	dw	down	dw	up	off	on
7	on	down	down	dw	up	off	on
8	on	down	down	down	up	off	on
9	off	s-uw	s-uw	s-uw	up	off	off
10	off	s-uw	uw	s-uw	up	off	off
11	off	uw	uw	s-uw	up	off	off
12	off	uw	uw	uw	up	off	off
13	off	up	uw	uw	up	off	off
14	off	up	uw	up	up	off	off
15	off	up	up	up	up	off	off

Figure 7: LOGISIM successive states for the simple circuit

user but they are not shown here.

As mentioned previously, LOGISIM is able to detect ambiguities in a design. Ambiguities arise when LOGISIM yields several solutions producing different circuit states. This happens when several events in the agenda occur at the same time. For instance in the above circuit, if the delay parameters in the circuit relays are given equal, LOGISIM yields two different solutions; the first solution is the one shown above and the second one have light bulb 1 on and light bulb 2 off. Therefore it is not possible to predict the behaviour of the real circuit.

5 Related work

Much work has been done for the software simulation of circuits especially for digital circuits and quantitative simulators but, to our knowledge, the problem as defined in this paper has not been addressed in the literature. The most relevant work has to be found in study of the *static* behaviour of analog circuits.

de Kleer and Brown apply qualitative reasoning to the analysis of analog circuits [2]. They use the notion of qualitative states which divide the behaviour of a component into different regions, each of which is described by a different set of confluences. Unfortunately, the loss of information induced by the use of confluences is very important. At each step of the simulation, qualitative reasoning returns a large set of possible states for the circuit, many of them being inconsistent. Moreover qualitative reasoning is not able to detect anomalies such as blowings of components. Therefore it turns out that an approach based on qualitative reasoning is not practical for simulating hybrid circuits.

In ARS and EL [10; 11], Stallman and Sussman use piecewise linear models [3; 5] to analyse analog circuits. Piecewise linear models capture much more information than qualitative states. However their constraint-solver based on *expression inference* [1] is too weak. Its inability to solve systems of inequalities induces several drawbacks. On the one hand, they may generate inconsistent circuit states even for simple circuits [10]. On the other hand, inequalities cannot be used for pruning the search space in an a priory way but simply as tests (when all variables are instantiated). This leads to a pathological behaviour known as trashing [7] where failures are detected very late in the computation implying deep backtracking. Stallman and Sussman propose the use of dependency-directed backtracking to overcome this difficulty but this remedies only partly to the problem.

Heintze et al also used piecewise linear models to analyse analog circuits [6]. Their constraint-solver, CLP(R), aims at propagating inequalities. This overcomes the first drawback of ARS and EL. However inequalities are not used to prune the search space in an a priori way in their approach either. Their system is then based on brute-force search; it assigns sequentially a state to each device and tests if it is compatible with already assigned devices. If not, the program backtracks to the last choice point.

6 Conclusion

This paper has presented LOGISIM, a CAD tool to simulate the temporal behaviour of hybrid circuits containing electro-mechanical, electro-hydraulic, hydro-mechanic, and digital control devices. LOGISIM combines the advantages of both qualitative and quantitative reasoning by producing a high-level description (discrete states) of the circuit behaviour while reasoning at the quantitative level (physical values). Apart from supporting engineers at an appropriate level of abstraction, efficiency was the key issue addressed by LOGISIM. To achieve that goal, LOGISIM makes use of sophisticated constraint-handling techniques available in CHIP. The decision procedure for linear constraints over rational numbers, consistency techniques on domain-variables and conditional propagation make possible to deduce all the implications of the choices as soon as possible. Hence they prune the search space by reducing the possible values of the device states. However, different ways of modelling a device can result in very different efficiency. Therefore, in LOGISIM, a particular methodology, the use of S-models, was adopted which avoids introducing an artificial complexity inside the simulation. LOGISIM has been applied successfully to real-life industrial circuits from aerospace industry in the ELSA project and clearly demonstrates the potential of this kind of tool to support the design process for these circuits.

Acknowledgments

We gracefully thank Mehmet Dincbas, Herve Gallaire and Alexander Herold and the members of the CHIP team at ECRC for their support and useful comments on this material. We also acknowledge the contributions of Jean Ruhla and the members of the ELSA project at AMD-BA.

References

[1] E. Davis. Constraint Propagation with Interval Labels. Artificial Intelligence, 32:281-331, 1987.

[2] J. de Kleer and J.S. Brown. A Qualitative Physics Based on Confluences. Artificial Intelligence, 24:7-84, 1984.

[3] C.A. Desoer and E.S. Kuh. Basic Circuit Theory. Electronic Engineering Series. McGraw-Hill, 1969.

[4] M. Dincbas, P. Van Hentenryck, H. Simonis, A. Aggoun, T. Graf, and F. Berthier. The Constraint Logic Programming Language CHIP. In FGCS'88, Tokyo, Japan, December 1988.

[5] P.E. Gray and C.L. Searle. Electronic Principles: Physics, Models and Circuits. John Wiley &; Sons, Inc, New York, 1969.

[6] N.C. Heintze, S. Michaylov, and P.J. Stuckey. CLP(R) and Some Electrical Engineering Problems. In Fourth International Conference on Logic Programming, Melbourne, Australia, May 1987.

[7] A.K. Mackworth. Consistency in Networks of Relations. AI Journal, 8(1):99-118, 1977.

[8] L.W. Nagel and D.O. Pederson. Simulation Program with Integrated Circuit Emphasis. In Proceedings of the 16th Midwest Symposium Circuit Theory, Waterloo, Canada, 1973.

[9] C. Pradelles-Lasserre, J. Ruhla, and L. Zimmer. Specification d'un systeme de simulation modulaire de circuits analogiques. Project ELSA, Rapport interne AMD-BA, 1987.

[10] R.M. Stallman and G.J. Sussman. Forward Reasoning and Dependency-directed Backtracking in a System for Computer-Aided Circuit Analysis. Artificial Intelligence, 9:135-196, 1977.

[11] R.M. Stallman and G.J. Sussman. Problem-Solving About Electrical Circuits, volume Artificial Intelligence: An MIT Perspective, pages 31-91. The MIT Press, Cambridge, Massachusetts, 1979.

[12] P. Van Hentenryck. A Framework for Consistency Techniques in Logic Programming. In IJCAI-87, Milan, Italy, August 1987.

[13] P. Van Hentenryck. Constraint Satisfaction in Logic Programming. Logic Programming Series. The MIT Press, Cambridge, MA, 1989.

[14] P. Van Hentenryck and M. Dincbas. Forward Checking in Logic Programming. In Fourth International Conference on Logic Programming, pages 229-256, Melbourne, Australia, May 1987.