Dagstuhl Seminar: Synchronous Languages '96

edited by Klaus Winkelmann

January 20, 1997

(page 2 intentionally left blank)

Introduction

Nicolas Halbwachs, Willem-Paul de Roever, Klaus Winkelmann

The Synchronous Languages Seminar 1996 was the third of a seminar series started in Dagstuhl (Nov 94) and continued at the CIRM, Marseille-Luminy (Nov 95). It brought together 50 researchers, mainly from France and Germany, but also from the USA, Great Britain, Sweden, India, and Israel, originating from both academia and industry.

Presentations covered many topics related to the synchronous programming languages such as Signal, Lustre, Esterel, Argos and Statemate. They ranged from foundations to case studies, and include verification as well as synthesis efforts, amply illustrating both the already now highly successful application of the associated tools, and the residual problems in applying them which still await solution.

Combination of several languages and cross-compilation, as well as comparing different semantics was one major topic in the presentations. Especially several groups work on combining data-flow concepts with automata-based models. Also combining the synchronous approach with deductive methods appears a promising path.

For some of the attending young researchers this was their first acquaintance with the subject of synchronous languages and their application; their attendance was enabled by a TMR-ESPRIT grant, for which we express our gratitude.

This workshop furthered a loose style of explanation, collaboration and contact to such an extent that some of the experienced industrial researchers present declared this to be the most successful workshop they ever attended, where they learned most.

Contents

Synchrony in Nets

Philippe Darondeau

Synchronous nets are a class of self-modifying nets where the weights of input arcs are marking dependent (the vector pre(a) of the weights pre(a)(x) relative to places x used as input by transition a is a function of the marking M) while the weights of output arcs are not marking dependent. Let pre(A) and post(A) be the extensions of pre(a) and post(a) to multisets of actions (thus the vector pre(A) is a function of M while post(A) is a constant vector), then the firing rule for synchronous nets is the following step firing rule: $M|A > M' \quad if \quad M' = M + post(A) - pre(A)(M + post(A))$. This rule is motivated by giving a direct translation from Signal programs to synchronous nets.

A Specification Scheme for Heterogenenous Synchronous Systems

Stephen Edwards

In this talk, I present a specification scheme for synchronous systems that are heterogeneous – the compiler does not know the contents of the blocks of the system, only how to evaluate them. My SR systems are composed of fixed blocks that evaluate instantly and communicate through single-driver, multiple-receiver wires. I present a deterministic semantics for them based on the least fixed point of a continuous function on a complete partial order, and a chaotic-iteration-based execution scheme. My heuristic scheduling algorithm uses a recursive divide-and-conquer approach based on strongly-connected components. The experimental results I present suggest this scheme is practical for graphs of reasonable size (approx. 100 outputs).

Time Intervals and Preemptive Tasks in Signal and their Application in Experiments

Eric Rutten

The control of reactive systems involves regulation functions, which are equations specifiable by a data flow graph of operations, as well as sequencing of such tasks, possibly with preemption. The latter is best specified using imperative or state-machine-based formalisms. This presentation describes an extension of the Signal data flow language with a notion of preemptive task. It associates a data flow process with its time interval of activity. A time interval is entered and exited upon the occurrence of events. The activity is re-started every time the interval is (re-)entered, either in its current state, or back from its initial state.

An implementation of this has been developed, in the form of a pre-processor to Signal; intervals are encoded into boolean state variables, suspension of the activity of processes is achieved by filtering of their inputs, and re-initialisation involves propagating additional control signals towards all state variables of the process.

Applications feature a system for active robot vision, the controller of a power transformer, and behavioral animation in image synthesis and simulation.

SyncCharts: A Visual Representation of Complex Reactive Behaviors (YACC: Yet Another Control Characterization)

Charles André

Reactive systems involve communication, concurrency and preemption. Few models support these three concepts, even less can correctly deal with their coexistence. The synchronous paradigm allows a rigourous approach to this problem, crucial to reactive systems.

A new visual model (SyncCharts) is introduced. This graphical model is fully compatible with the imperative synchronous language "Esterel" and is specially convenient to express complex reactive behaviors.

SyncCharts support hierarchy (embedded macro-states), concurrency (parallel composition of constellations), and preemption (abortion and suspension of star activities). The way SyncCharts treat normal termination of parallel evolutions is an interesting feature of the model.

A specification of a Cruise Speed Controller illustrates the use of SyncCharts. An editor of SyncCharts and a translator to Esterel programs are demonstrated.

A Synchronous, Modular Hierarchical Finite State Machine Model With Extensions

Joseph T. Buck

A graphical means of specifying hierarchical finite state machines is presented – a variant of Statecharts designed to be highly modular and to have strictly synchronous semantics. Of the models in the literature, it is closest to Argos, adding the concepts of weak and strong abortion, voluntary termination, and suspension from Esterel. The semantics are defined by specifying a translation to Esterel of each concept in the description; this translation is explained. The model is then extended to permit inclusion of dataflow specifications inside the hierarchical model at any level.

Compositional Approaches to Concurrent Performance Analysis

Chris Tofts

Recent extensions to process algebra can be used to describe performance or error rate properties of systems. We examine an abstract approach to the representation of time costs within these algebras that permits the efficient calculation of performance bounds on the systems. In particular we avoid the 'state explosion' caused by the parallel composition of the representations of probabilistic time distributions. A major advantage of one of our approaches is its uniformity which allows the eventual approximation level to be easily predicted from that of the approximation to the initial distributions.

A Real-Time Semantics for the Statemate Implementation of Statecharts

Carsta Petersohn

We formalize the central simulation algorithm Go-Step for the language of Statecharts which are part of the tool StateMate following the most recent description by Harel and Naamad [HN95]. Our semantics is defined in a modular way based on full compound transitions, different kinds of steps, and time abstractions. The semantics is given in terms of fair transition systems and clocked transition systems. This enables the usage of the proof systems for the verification of real–time properties given in linear temporal logic. We also discuss how typical concepts of synchronous languages are modeled by our semantics.

[HN96] David Harel, Amnon Naamad, The STATEMATE Semantics of Statecharts, ACM Trans. Soft. Eng. Method., 1996.

Automatic Verification of Parameterized Linear Networks of Processes

David Lesens, Nicolas Halbwachs, Pascal Raymond

We present a a method to verify safety properties of parameterized linear networks of processes. The method is based on the construction of a network invariant, defined as a fixpoint. Such invariants can often be automatically computed using heuristics based on Cousot's widening techniques. These techniques have been implemented and some non-trivial examples are presented. This work will be published in POPL'97.

Semantics and Compilation of Synchronous Specification

Klaus Winkelmann

A Specification is a set of constraints on the traces of an automaton, together with assumptions (also constraints) on the environment. The concept of controllability, known from Discrete Event Systems, can be generalized to polychronous systems by introducing controllable and uncontrollable clocks, and controllable and uncontrollable values of a signal.

Compiling a specification to executable code consists in two main phases, which we call conflict resolution and function extraction. Conflict resolution is the same as Ramadge/Wonham's supervisor synthesis - it generates the most general automaton that can always react to an environment action. Function extraction converts this most general automaton - an acceptor - to a transducer. It involves solving relational equations to a functional form, and scheduling function calls, i.e. topological sorting.

Some of these ideas underlie the CSLxt language and compiler by Siemens.

Program Synthesis With CSLxt for the Case Study Production Cell

Helmut Melcher

Controller synthesis is an approach for solving reactive problems by using a compiler (synthesizer) which automatically generates the control program from a description of requirements and environment.

We discuss practises for applying the synthesis technique using an example from factory automation, a modification of the Production Cell case study. For this purpose, we use CSLxt, a method developed by Siemens Corporate Research in Munich. CSLxt allows for specifications by implicit automata with fair process activation. Safety and liveness properties can be described by predicates over states.

We present several approaches which differ in the complexity of the resulting synthesis problem and the complexity of the description. A compromise between these conflicting requirements has to be found. We present and discuss some ideas towards a systematic solution for this trade-off.

Modelling and Verification of a Production Cell with Grafcet

D. L'Her, L. Marcé

Our aim is to specify and verify the production cell KORSO using the GRAFCET language.

The semantics of GRAFCET is given with timed automata to take time into account. A state is defined by a grafcet situation, values of inputs, temporizations and memorized actions. When an input changes its value or when a temporization is modified, a transition is defined.

We use this modelling to check properties with the model-checker KRONOS. In the case of the production cell, there are two problems to solve : size and environment. To reduce the size, we isolate each component of the cell to verify properties. We take environment into account also to reduce the size at the level of TCTL formula, at the level of GRAFCET or during the construction of timed automaton. For instance we have restraints on states, on transitions and between actions and sensors. With this strategy, we verify properties of safety. An interface helps the users to express the properties and the restraints.

An Operational Semantics for Modechart

Carlos Puchol

The Modechart specification language is a formalism for the specification of real-time systems. The semantics of the language was defined axiomatically in Real-Time Logic. We define the semantics for Modechart in an operational style. Modechart is a synchronous language that permits non-deterministic real-time specifications. The semantics for the class of deterministic specifications is introduced first, followed by the definition of entire class of non-deterministic specifications. The deterministic semantics naturally derives a Modechart compiler, which allows for automatic synthesis of formal specifications. An extension to the compiler presented provides support for a limited, but very useful in practice, subset of the class of non-deterministic specifications. We characterize this class and show how it can be used in automatic code generation.

Translating Statecharts to Promela/SPIN.

Erich Mikk, Yassine Lakhnech, Michael Siegel

Statecharts are compiled into Promela, the input language of the SPIN model checker. This allows the model-checking of Statecharts with respect to linear-time temporal logic while taking advantage of the advanced techniques implemented in SPIN for reducing the visited state space. These techniques include the partial-order approach for tackling the state explosion problem.

The experimental compiler works on a sub-language of statecharts and focuses on control issues of the language. The translation preserves the parallel structure present in source statecharts-program and exploits its hierarchy.

The compiler accepts statecharts specifications in an intermediate input format that enables to build links to graphical editors for drawing statecharts. The output of the tool is a readable well-structured Promela code.

The presentation focuses on the key translation step: transforming statecharts to hierarchical automata. The purpose of hierarchical automata is to provide a semantics model for statecharts where the semantics of a composed construct can be understood using the semantics of its parts and the way the parts coordinate with each other. The class on hierarchical automata is defined. The semantics of HA is given using structural operational semantics rules (SOS rules). These rules are directly implemented in the compiler. The presentation is built around Promela, but the approach is applicable for other model-checkers also.

Future work is further compiler development, adopting the SOS rules for compositional reasoning about statecharts and translation of statecharts to other model checkers (SMV).

Verifying the Esterel Compiling Algorithms Using COQ

Gerard Berry, D. [Kaplan / Terrasse]

The Esterel v5 compiler is based on two ingredients: the new constructive semantics based on a theory of information propagation in synchronous programs, and an improved translation of Esterel programs into digital circuits that fits well with the constructive semantics. The translation into circuits is quite subtle at places, especially in the handling of reincarnation of statements and signals by loops. Furthermore, there are several possible variations in gate placement that we would like to analyze. We think that a completely formal proof is the right tool to show the translation correctness and also to analyze variants: the most natural variants should be those that make the proof the most elegant.

We have started performing the roof in COQ. Describing the constructive semantics of Esterel in COQ was trivial. Describing circuits and the circuit translation turned out to be much more delicate. The solution we use is to represent circuits by synchronous functions on Scott-adic integers, which differ form 2-adic integers by the fact that bits may be 0, 1, or undefined. So far, we have carried out the proof for the loop-free programs that involve no reincarnation. We expect the general proof to be a smooth extension of the partial proof. Once the proof will be completed, we shall be able to use the program extraction facilities of COQ to extract a reference compiler out of the proof.

Certified compiler construction for a reactive language with the Coq prover

Olivier Roux, Pablo Argón

We first overview the Electre reactive language through its basic entities and operators. Then, we give some examples of the way the natural semantics of the language is expressed in SOS.

We intend to prove the following theorem:
$\forall p$ (program).$(p \in$ Electre$) \rightarrow \exists a$ (automaton).$(p \equiv_{Electre\ semantics} a)$.
Moreover, the proof (of $p \equiv a$) gives a construction of the translation function $p \rightarrow a$

We show that this can be achieved using Coq, and with the following steps:

- Coding the language structures,
- Modeling the semantics,
- Specification: translation function (compiler $p \rightarrow a$),

- Proofs: *consistency* and *completeness* of the specification,
- Extraction of the CAML program of the compiler.

This procedure build a theory of Electre in Coq. It makes it possible (1) to synthesize the compiler of the language, (2) to extend the semantics, (3) to make proofs of program properties, and (4) in a more general way, to customize the ideal kernel of reactive language (with the useful operators).

A Tutorial Introduction to TLT using the Production Cell

Jorge R. Cuellar

The TLT (Temporal Language of Transitions) specification method and tools are described using the example of the new (fault-tolerant) production-cell proposed by the German project KORSYS. The TLT semantical-objects, first-order automata (FOL-automata), are basically well-known. Indeed, many variants have been studied by Lamport, Kurshan, Wolper, Pnueli, Gurevich and others. The particular choices in the the definition of FOL-automata (the stuttering conditions, the type of fairness properties, etc.) are discussed.

The relatives of FOL-automata (Evolving Algebras, TLA, Boolean Automata, Fair Transition-Systems, etc.) are often used to give a "reference semantics" to more complex programming models or languages, such as Prolog, StateCharts, etc. Usually this involves a non-trivial translation or coding.

The purpose of TLT is to provide users with syntactical means to describe directly in FOL-automata (without coding) *a)* systems, *b)* their architecture and *c)* the user's knowledge or understanding of the sytem (in the form of annotations, or assumption-commitment predicates on the interfaces). In contrast to TLA, Cospan, Evolving Algebras etc, the TLT method introduces modules, interfaces, views, inputs/outputs, etc. during the refinement process. This TLT methodology may be seen as a structural guideline for creating *local* FOL verification-conditions to show that the specification is (globally) consistent and it satisfies its specification.

The fault-tolerant production-cell is particularly interesting because *a)* the sensor failures may not be diagnosed immediately, and *b)* a real-time aspect is inherent in the problem, namely, the use of timers to supervise the movement of the motors and the duration of the protocols.

Specifications of Reactive Systems with μ-Charts

Peter Scholz

During the last years, Statecharts have gained wide acceptance for the specification of reactive, embedded systems. However, most semantics suggested so far are either informal or overly complicated. In this contribution, we present a lean Statecharts dialect, called μ-Charts, that permits nondeterministic specifications, offers zero-delay broadcast communication, and handles negation in trigger expressions in a new way.

We give a compositional formal semantics for this dialect, which is abstract enough for formal reasoning and yet easy to operationalize for simulators, model checking tools, and code generation.

The well-known causality conflicts that arise under instantaneous feedback from negative trigger conditions are resolved semantically through oracle signals. We have implemented a prototypical tool that translates μ-Charts specifications into μ-calculus formulae. These formulae are checked against temporal specifications using a μ-calculus verifier.

As a further result of our work on symbolic verification of μ-Charts, we show how a μ-Chart can be implemented in hardware, using a register and a combinational logic block that represents the transition relation of the system.

Multi Clock Esterel

RK Shyamasundar[1]

We present a uniform framework referred to as, M-ESTEREL (Multi-Clock Esterel), for the modelling of heterogenous systems. The formalism has several clocks and generalizes the classical ESTEREL which is monochronous. M-ESTEREL supports multiple clocks within the same ESTEREL node. Events are no longer required to be tightly coupled to clocks. This implies the need of some form of latching to deal with signals which are no longer synchronous with the current clock; further, latching would have to be defined in the context of preemption. The formalism has been arrived at with very few additions to the classical ESTEREL. The additions are: latching of signals and the new statement "`newtick` t `in` STAT `end`". In the presentation, we describe the model and show its generality in describing multi-clocked systems. M-ESTEREL continues to remain synchronous with respect to the current clock in a module in as much as reactions occurred synchronous with them. With M-MESTEREL we hope to illustrate a model which can handle asynchronous as well as synchronous events with equal ease and also discuss the power of scalability of the model. The earlier model of CRP (Communicating Reactive Processes) can be obtained as special case. Further, several of the features of hardware specification languages can be described naturally in the model. Relative comparison with respect to other models is also done.

Compiling Argos into Boolean equations or : multi-language programming in the synchronous world

F. Maraninchi

In most imperative synchronous languages (Esterel, Argos, Statecharts,...), the semantics of the control structures may be conveniently described as compositions of Mealy machines. This constitutes the usual formal semantics of Argos, for instance, where basic components are Mealy machines. On the other hand, the compilation process should not be based upon an exhaustive generation of the Mealy machine that represents the behaviour of the whole program, because this machine may have a very large number of states. Hence we try to perform a symbolic compilation, representing Mealy machines by sets of equations. We give here the direct semantics of Argos in terms of such equations, and show that this semantics coincides with the usual one. The current implementation of the Argos compiler produces DC code, which is the common equational format for synchronous languages. This will allow to merge imperative and declarative synchronous languages (Argos and Lustre, for instance), by merging DC files.

[1] With contributions from Basant Rajan, S. Ramesh and G. Berry.

A Smalltalk About Checking Properties In EmbeddedEifel

Reinhard Budde

Our goal is to develop a methodology for the construction and validation of medium-scale embedded systems with real-time constraints. We use object oriented methods during the whole development process to partition the system into classes. Classes are the units of information hiding and reuse. In a class both the reactive behavior as well as the datatype behavior of its objects are defined. Model checking is used to analyze the system, especially to support design decisions. The partition in classes and the synchronous semantics allow to describe and analyze crucial parts of the system in isolation. The methodology is supported by tools for the synchronous realtime language embeddedEifel.

Formal verification is very attractive for our system development methodology because reactive behaviors of objects are defined in a synchronous language. Synchronous languages are based on the synchrony hypothesis: outputs are synchronous with inputs, for the environment. Synchronous programs are compiled quite efficiently into boolean automata. This fact makes automatic verification like model checking feasible.

The systems designer expresses properties to be checked, gives them a name for referencing and attaches them to some class. The reactive behavior of an object-configuration to be checked is represented by a boolean automaton. The properties are translated to either boolean automata or directly to input for a model-checker (we support SMV and VIS). Boolean automata can translated to input for model-checker, too. For the systems designer the check of a property is reduced to a push-button activity of commanding tools to check whether a formula selected by its name is valid.

Verification is not used to show, that a final implementation meets an initial formal specification. We use an incremental approach, in which verification supports design decisions. Proofs refer to the actual implementation and not to a separate model. Proofs are performed for individual objects and for object-configurations of a system and their respective classes. In this way they are modularized. Different formalisms to express properties to be checked are integrated (based on Boolean Automata): CTL, PTL (past temporal logic), a Statechart-like graphic notation, and state transition diagrams.

Verification of Embedded Systems using Synchronous Observers

Simin Nadjm-Tehrani [2]

I present a study of observer-based proof techniques applied to the verification of a model of a real world embedded system, an aircraft landing gear. A formal description of these techniques (taken from Halbwachs et.al. [amast93]) is presented, followed by three ways of applying them. More specifically, one shot verification of the composed system is compared with two approaches to decompositional verification. The example illustrates that due to the tight interaction in a plant-controller setting there is often little to be gained by adopting a decompositional approach to verification. Nonetheless, two reasons are presented for separation between the controller and its environment at the modelling stage. Hence the result of the study is that in cases similar to this one, it is most expedient to prove system properties using the composed model derived from individual parts.

[2] Joint work with Martin Westhead, Dept. of AI, University of Edinburgh

Specifying Concurrent Processes with Statecharts and Z

Robert Büssow [3]

We specify embedded systems by decomposing them into concurrent, synchronous processes that communicate via shared variables. The system is described from three different views: The first view, the architectural view, describes the decomposition in components or processes. It also specifies the relationships between these processes, particularly their communication relations. The reactive view describes the reactive behavior of each class, that is when and how it reacts upon external and internal stimuli or events. The functional view describes the space of data states, and, for each process, the transformation of the data and data invariants.

The reactive behavior of the processes is described using statecharts (Statemate), the functional behavior is described using the Z specification language. For the implementation of statechart events, volatile Z variables are introduced. Z schemas are used to define the interfaces of a process as a set of shared variables. The reactive behavior of a process is described with statecharts and its data and data-transformations are described with Z. The interface between statecharts and Z are the statecharts' transition-labels, i.e. the guards are Z predicates and the actions are Z operations. The combination gives rise to the problem of executing Z operations over the same state in parallel. We propose and discuss two solutions for this problem – interleaved and concurrent execution:

Interleaved execution is defined as executing the operations sequentially in a non-determined order. This can be defined with the Z schema calculus for two Operations Op_1 and Op_2 as $Op_1; Op_2 \vee Op_2; Op_1$ The concurrent execution is defined as the conjunction of the operations. To avoid contradictions, variables that are not changed by the operations, are hidden from the operation's signature. Then the conjunction can be created, where further hiding, in cases of racing conditions, is allied.

Abstract Object Systems

Matthias Weber

Abstract object systems are a mathematical model of dynamically changing collections of interacting objects whose basic operations have non-zero data-dependent durations and where communication is based on asynchronous message-passing and controlled by a state machine with operation and timeout transitions and run-to-completion semantics. Abstract object systems can be specified by a combination of object diagrams, state charts, and (nested) Z schemas.

[3] joint work with partners from the Espress project, see http://www.first.gmd.de/espress/

Modelling Parallelization with Partial Orders

Benoit Caillaud, Paul Caspi, Alain Girault, Claude Jard

Our objective is to prove that a given parallelization algorithm is correct, that is, that the behavior of the initial centralized program is equivalent to the behavior of the final parallel program.

Our parallelization algorithm is intended for distributed memory machines, where parallel programs communicate through a networks of FIFO channels. This algorithm has been applied to sequential imperative programs as well as synchronous programs. In both cases, the source program is given as a finite deterministic automaton labeled with actions. The behavior of the centralized program is the language of this automaton, i.e., the set of finite and infinite traces of actions it generates. Finally the parallelization specifications are given as a partition of the set of actions into as many subsets as there are processors in the distributed memory machine.

We define a commutation relation between actions according to the data dependencies. This commutation relation induces a rewriting relation over traces of actions. The set of all possible rewritings is the set of all admissible behaviors of the centralized program according to the commutation relation. The problem is that this set cannot, in general, be recognized by a finite automaton. The intuition of our proof is that this set is identical to the set of linear extensions of some partial order. For this reason we introduce a new model based on partial orders.

First, we build a centralized order automaton by turning each action labeling the initial automaton into a partial order capturing the data dependencies between this action and the remaining ones. The language of our order automaton is the set of finite and infinite traces of partial orders it generates. By defining a concatenation relation between partial orders, each trace is then itself a partial order. Thus the language of our order automaton is a set of finite and infinite partial orders. We show that the set of linear extensions of all these partial orders is identical to the set of all admissible behaviors of the centralized program according to the commutation relation.

Second, we show that our order automaton can be transformed into a set of parallel automata by turning the data dependencies between actions belonging to distinct processors into communication actions, and by projecting the resulting automaton onto each processor. These transformations are shown to preserve the behavior of our order automaton.

Sequential Optimization in Esterel

Horia Toma

In a gate-level description of a finite state machine, there is a tradeoff between the number of latches and the size of the logic implementing the next-state and output functions. Typically, an initial implementation is generated via explicit state assignment or translation from a high-level language, and this tradeoff is only lightly explored via logic synthesis from that point on.

We address the problem of efficiently exploring good latch/logic tradeoffs for large designs generated from high-level specifications. We describe algorithms for reducing the number of latches while controlling the size of the intermediate logic. We use these algorithms to generate good final implementations (e.g., hardware or software) and good intermediate representations (e.g., for symbolic state traversal). We demonstrate the efficacy of our techniques on some large industrial examples.

On Modeling Small Size Automotive Controller Using Lustre

Rainer Gmehlich

In present cars there are many safety-critical systems which are realized with micro-controllers or other digital devices. To improve the safety, formal methods are needed.

In the talk first results of a case study on an automotive application was presented. In this study the synchronous language Lustre is used to model the boolean part. On this model safety properties are proven. Problems to ensure the synchrony hypotheses in real applications are discussed.

Esterel++, an Object Oriented Extension of Esterel

Eric Nassor

With the last version of the Esterel compiler (Esterel V5), large applications may be analyzed, but the syntax of the language is not well adapted : many parts of the code must be duplicated (interfaces, external declarations ...), and in many cases code reutilization is difficult.

Esterel++ introduces some new notions in the Esterel language in order to solve these problems.

- lines: lines are a structuration of the signals. With this mechanism, the size of the interfaces decreases.
- packages: all the external declarations may be grouped in one place, and then be used in differents modules.
- synchronous statecharts: based on the Argos semantic, synchronous statecharts may be used to describe the behavior of modules. These modules can be freely mixed with the Esterel code.
- object orientation: Esterel++ introduces the notion of inheritance between modules. Modules may inherit interfaces, code, and attributes (all the module instances are attributes, this means all the *run* instructions). As the Eiffel language, Esterel++ support multiple inheritance with renaming and redefinition. In particular, with the inheritance of attributes, this means that the types of the instances may be changed. This is a powerful mechanism which allows a better code reutilization.

Esterel++ is compatible with Esterel: a correct Esterel code is a correct Esterel++ code. A preprocessor has been implemented, which translates Esterel++ in Esterel. This compiler is currently being tested.

Symbolic Simulation of Interpreted Automata

Christophe Mauras

We present a simulation tool for interpreted automata. Its input language is a small subset of DC (declarative code for synchronous languages), containing only assertions about boolean and integer variables defined on the same global clock.

The tool inherits from a previous one, dealing with boolean automata and from Bac. It has been enhanced with some numerical computations based on work by Halbwachs about linear relation analysis.

We illustrate with some examples, how to use it to discover linear relations over the variables of a synchronous system, and for symbolic debugging of a temporal specification.

The simulator performs forward analysis, and computes an upper approximation of the set of reachable states. We use Binary Decision Diagrams to encode set of states in $\mathbb{B}^m * \mathbb{Z}^n$, and relations.

An algorithm is presented to apply standard logical operations on such data structures.

We then talk about relationship with verification tools devoted to synchronous programs, and with some works related to constraint logic programming: Toupie by Rauzy, abstract interpretation of CLP by Handjieva.

We conclude that using relational computations seems to be useful for debugging but inefficient for verification purposes. So, would a symbolic debugger for DC be a friendly companion to verification tools as Lesar and Polka ?

Reasoning about Time in SIGNAL

Apostolos A. Kountouris

Reasoning about Time in SIGNAL

In R/T systems development, temporal correctness signifies that an implemented system respects its real-time constraints. Currently in the SIGNAL environment the temporal dimension of a system is entirely abstracted by the use of the synchrony hypothesis, and the focus is mainly put on the functional aspects. The temporal dimension remains a quite unexplored area. We present an approach that aims to serve as a facility for the evaluation of the temporal behavior of a system when implemented on a particular target architecture, in respect to its R/T constraints. We argue that this should happen at the specification level so that "temporal debugging" can be more effective and the development cycle shorter. At a first stage we investigate the factors influencing execution time and we attempt a classification. These factors come into play at various points during the process of transforming a functional specification to an operational system. Consequently it is quite important to identify all of these factors, in order to effectively take them into account.

What is equally important is the aspect of building tools that can exploit this information and assist the user when the system evaluation passes from the functional to the temporal domain. To this end we currently studying how to model low-level issues at the specification level and how to automatically extract temporal properties of SIGNAL programs given an intended implementation. Such tools can find applications in various fields relating yo performance evaluation like for instance R/T system development, design space exploration in HW/SW co-design etc.

Handling data-flow programs in PVS

Saddek Bensalem, Paul Caspi and Catherine Parent-Vigouroux

This presentation investigates the use of the PVS tool for handling data-flow programs. In particular, we show how to express the constructs of the Lustre synchronous data-flow language. We then provide examples of program derivation and proofs within this framework, which hopefully illustrate the interest of the approach.

Verification and Control of Polynomial Dynamical Systems over Galois fields: Application to a Power Transformer Station Controller

Hervé Marchand

We have presented a methodology for the verification and the automatic controller synthesis of reactive systems, and its application to a case study. Systems are specified using the synchronous data-flow language **Signal**. In order to check various kind of properties, and synthesize controller, we need to translate its boolean part into a system of polynomial equations over three values denoting *true*, *false* and *absent* (this translation is natural because **Signal** is based on an equationnal approach(i.e. **Signal** programs are constraint equations between signals). Using operations of algebraic geometry, on the polynomials, it is possible to check properties concerning the system, such as *liveness*, *invariance*, *reachability* or *attractivity*. We have also presented computational methods for the synthesis of controllers for discrete event systems, modeled by polynomial dynamical systems. We also presented our approach to the optimal control synthesis problem as well as other interesting aspects in control synthesis like for example ensuring the invariance of a property.

Finally, we applied these methods to the verification of the automatic circuit breaking control system of an electric power transformer station controller. This controller handles the reaction to electrical defects on high voltage. On the other hand, we applied the controller synthesis methods to a general safety property concerning the global system.