04241 Abstracts Collection

Graph Transformations and Process Algebras for Modeling Distributed and Mobile Systems

Barbara König [1], Ugo Montanari [2], Philippa Gardner [3]

[1] Institut für Formale Methoden der Informatik, Universität Stuttgart, Germany
koenigba@fmi.uni-stuttgart.de

[2] Computer Science Department, University of Pisa, Italy
ugo@di.unipi.it

[3] Department of Computing, Imperial College London, Great Britain
p.gardner@imperial.ac.uk

Dagstuhl Seminar Proceedings 04241
http://drops.dagstuhl.de/opus/volltexte/2005/27

Summary

Recently there has been a lot of research, combining concepts of process algebra with those of the theory of graph grammars and graph transformation systems. Both can be viewed as general frameworks in which one can specify and reason about concurrent and distributed systems. There are many areas where both theories overlap and this reaches much further than just using graphs to give a graphic representation to processes.

Processes in a communication network can be seen in two different ways: as terms in an algebraic theory, emphasizing their behaviour and their interaction with the environment, and as nodes (or edges) in a graph, emphasizing their topology and their connectedness. Especially topology, mobility and dynamic reconfigurations at runtime can be modelled in a very intuitive way using graph transformation. On the other hand the definition and proof of behavioural equivalences is often easier in the process algebra setting.

Also standard techniques of algebraic semantics for universal constructions, refinement and compositionality can take better advantage of the process algebra representation. An important example where the combined theory is more convenient than both alternatives is for defining the concurrent (noninterleaving), abstract semantics of distributed systems. Here graph transformations lack abstraction and process algebras lack expressiveness.

Another important example is the work on bigraphical reactive systems with the aim of deriving a labelled transitions system from an unlabelled reactive system such that the resulting bisimilarity is a congruence. Here, graphs seem to be a convenient framework, in which this theory can be stated and developed.

So, although it is the central aim of both frameworks to model and reason about concurrent systems, the semantics of processes can have a very different flavour in these theories. Research in this area aims at combining the advantages of both frameworks and translating concepts of one theory into the other. The Dagsuthl Seminar, which took place from 06.06. to 11.06.2004, was aimed at bringing together researchers of the two communities in order to share their ideas and develop new concepts. These proceedings[4] of the do not only contain abstracts of the talks given at the seminar, but also summaries of topics of central interest.

We would like to thank all participants of the seminar for coming and sharing their ideas and everybody who has contributed to the proceedings.

Stuttgart, *Barbara König*
November 2004

[4] The proceedings were compiled by Tobias Heindel.

Contents

1 Bigraphs, link graphs, transitions and Petri nets

Robin Milner

University of Cambridge, The Computer Laboratory,
J J Thomson Avenue, Cambridge CB3 0FD, UK

In my two lectures I introduced the bigraph model of mobile interactive systems, and explained some of the mathematics underlying the model. I illustrated this by formulating a model of a simple class of Petri nets, and then using bigraph theory to derive a behavioural congruence for them. The subject matter was treated in four parts.

In Part One I discussed in broad terms the way in which the two constituents of bigraphs –their *placing* and *linking*– combine to yield a very general dynamic structure. One way to think of it is: *where you are doesn't affect whom you can talk to*. In a bigraph, elements that are placed arbitrarily 'far apart' can be linked, and the bigraph can reconfigure both placing and linking. I illustrated this by presenting reaction rules –i.e. the discipline of reconfiguration– for the π-calculus and the ambient calculus, both of which employ placing and linking non-trivially. I isolated four distinct roles played by placing in bigraphs; when none of these is required then it is enough to employ just one of their constituents: the *link graphs*.

In Part Two I stepped back from bigraphs to the framework of *s-categories*, a form of category in which the composition of two arrows is defined iff their *support sets* are disjoint. In this framework, thinking of the arrows as *contexts* and the objects as *interfaces* between them, I defined an general notion of reactive system. I then recalled work done jointly with Jamey Leifer five years ago, in defining *relative pushouts* (RPOs) and *idem pushouts* (IPOs); when these exist in an s-category then it is possible to derive *labelled transition systems* uniformly for any reaction system, in such a way that familiar equivalences and preorders, such as bisimilarity and the failures preorder, are guaranteed to be *congruential* – i.e. preserved by all contexts.

In Part Three I returned to the more applied world of bigraphs, or more accurately of their link-graph constituents. I set up link graphs as an s-category whose interfaces are just finite sets of names, acting as links; the support of a link graph is just the sets of its nodes and edges. These support elements do the job of keeping track of different *occurrences* of the same kind of nodes, and this is essential in demonstrating that RPOs and IPOs do indeed exist in link graphs. (The same holds for place graphs, which are the other half of bigraphs, and thence it holds also for bigraphs themselves. However, it fails for any of these if we attempt to work in the associated, more abstract, *category*, gained by forgetting ther identity of nodes and edges.) From this, the background theory of reactive systems over s-categories yields a behavioural bisimilarity which is a congruence.

In Part Four I addressed the question: in a particular application, does this behavioural congruence correspond to anything that has been, or could be, defined by means independent of bigraphs? A particular application domain can be deter-

mined (in either link graphs or bigraphs) by two things: a *signature* that defines the kinds of node that will be used, and a *sorting* discipline that constrains the ways in which these nodes can be assembled into a graph. I therefore defined the signature and sorting discipline that allow link graphs to model a certain class of Petri nets – the so-called *condition-event* nets. Independently of link graphs I defined a natural transition system for these nets, yielding an associated bisimilarity. I then outlined the fairly simple proof that it coincides with the bisimilarity induced by link-graph theory; as a corollary, this yields that the former bisimilarity is also a congruence.

The lectures gave an introduction to bigraph theory by means of a particular application, rather than by a more rigorous theoretical development. Full references, and comparisons with other work, can be found in the two Technical Reports listed below. They can also be found via the authors website `http://www.cl.cam.ac.uk/users/rm135`.

References

[1] Jensen, O.-H. and Milner, R. (2004), Bigraphs and mobile processes (revised). Technical Report UCAM-CL-TR-580, University of Cambridge Computer Laboratory.

[2] Milner, R. (2004), Bigraphs and Petri nets. Technical Report UCAM-CL-TR-581, University of Cambridge Computer Laboratory.

2 A Complete and Minimal Programming Language for Graph Transformation

Annegret Habel[1] and Detlef Plump[2]

[1] Universität Oldenburg, Germany
habel@informatik.uni-oldenburg.de
[2] The University of York, United Kingdom
det@cs.york.ac.uk

The use of graphs to represent and visualise complex structures is ubiquitous in computer science, and often these structures occur in contexts where they have to be dynamically changed. Functional and logic programming languages, on the other hand, are successful examples of high-level programming languages based on rules. Thus a natural idea is to design programming languages based on graph transformation rules, in order to combine the strengths of graphs and rule-based programming.

Existing programming languages of this type include PROGRES, AGG, GAMMA, GRRR, and DACTL. These languages have in common that they are based on graph transformation rules, but they vary strongly with respect to both the formalisms underlying the rules and the available constructs for controlling rule applications. In view of the variety of control mechanisms, the question arises what programming constructs are really needed on top of graph transformation rules to obtain a computationally complete language. By computational completeness we mean the ability to compute every computable partial function on labelled graphs. Identifying such a core language for graph transformation will be valuable for both the understanding of existing languages and the design of new programming languages of this kind.

We identify a set of programming constructs ensuring that a programming language based on graph transformation (in the so-called double-pushout approach) is computationally complete. These constructs are nondeterministic application of a finite set of graph transformation rules, either (a) one-step or (b) as long as possible, and (c) sequential composition of graph programs. Moreover, this language is minimal in that omitting any of the three constructs results in a computationally incomplete language. Our completeness proof is based on the sequential composition of three programs: the first encodes arbitrary graphs as string-like graphs, the second simulates Turing machines on these string graphs, and the third decodes the resulting string graphs back into general graphs.

A preliminary conference paper on this topic can be found in the proceedings of FOSSACS 2001, LNCS (volume 2030 of Lecture Notes in Computer Science, Springer-Verlag, 2001.

3 Completeness and Minimality of Rule-based Languages

Annegret Habel[1] and Detlef Plump[2]

[1] Department Informatik, Universität Oldenburg
[2] Department of Computer Science, The University of York

We study computational completeness and minimality of rule-based programming languages on arbitrary domains. Our abstract framework only assumes a universe of rules whose subsets induce binary relations on a given domain:

Let O and $\mathcal{R}$ be sets of objects *and* rules, *respectively, such that each subset R of $\mathcal{R}$ induces a binary relation $\Rightarrow_R$ on O.*

Three concrete examples are sets of string, term and graph rewriting rules with their one-step rewrite relations. *Programs* over $\mathcal{R}$ are built from three constructs: nondeterministic application of a finite set of rules, either (a) in one step or (b) as long as possible, and (c) sequential composition of programs. Formally:

(1) For every finite set R of rules, R and $R\downarrow$ are programs.
(2) If P_1 and P_2 are programs, then $P_1;P_2$ is a program.

The *semantics* of a program P is a binary relation $[\![P]\!]$ on O:

(1) $[\![R]\!] = \Rightarrow_R$ for a finite set of rules R.
(2) $[\![R\downarrow]\!] = \{\langle x,y\rangle \mid x \Rightarrow_R^* y \text{ and } y \notin \mathrm{Dom}(\Rightarrow_R)\}$.
(3) $[\![P_1;P_2]\!] = [\![P_1]\!] \circ [\![P_2]\!]$.

We have established a *completeness condition* guaranteeing that every computable binary relation (and hence every computable partial function) on O is computed by some program. Roughly, the condition requires that there is a subclass of "string-like" objects such that computable relations on strings can be computed by programs on string-like objects (e.g. by simulating Turing machines), and that there are programs for encoding objects as string-like objects and decoding the latter.

One can show that the completeness condition holds in the settings of string, term and graph rewriting programs, and hence these languages are computationally complete.

Moreover, we have developed abstract techniques for showing that programming constructs are necessary for computational completeness. Applying these techniques in the cases of string, term and graph rewriting, we can show that if either the construct "as long as possible" or the sequential composition is omitted, the resulting languages will be incomplete. For string and graph rewriting, the one-step application of rules cannot be omitted either and hence the string and graph rewriting languages are minimal in the sense that none of the three constructs can be dropped.

4 Deriving Bisimulation Congruences in the DPO Approach to Graph Rewriting*

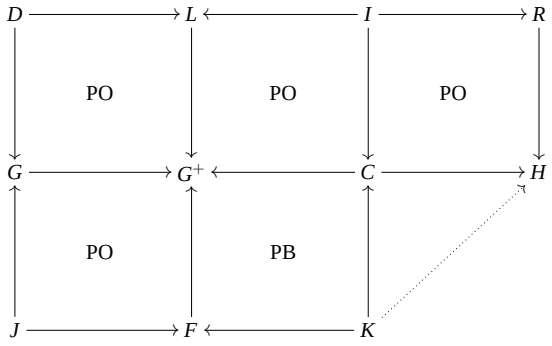
Barbara König

Institut für Formale Methoden der Informatik, Universität Stuttgart
koenigba@fmi.uni-stuttgart.de

Motivated by recent work on the derivation of labelled transitions and bisimulation congruences from unlabelled reaction rules, we show how to solve this problem in the DPO (double-pushout) approach to graph rewriting. Unlike in previous approaches, we consider graphs as objects, instead of arrows, of the category under consideration. This allows us to present a very simple way of deriving labelled transitions (called rewriting steps with borrowed context) which smoothly integrates with the DPO approach, has a very constructive nature and requires only a minimum of category theory. The core part of this paper is the proof sketch that the bisimilarity based on rewriting with borrowed contexts is a congruence relation.

Let us briefly summarize the main ideas of the paper. We consider double-pushout rewriting on graphs with interfaces. A graph with interface is given by an injective graph morphism $J \to G$ where J is called interface. Furthermore we are considering graph contexts of the form $J \to F \leftarrow K$, where J is the inner and K is the outer interface. A graph with interface can be composed with a context by taking the pushout of $J \to G$ and $J \to F$ which results in a new graph with interface K.

Whenever we have a graph with interface $J \to G$ and a rewriting rule $L \leftarrow I \to R$, then we say that $J \to G$ can be rewritten to $K \to H$ with label $J \to F \leftarrow K$ whenever there are graphs D, G^+, C and additional morphisms such that the following diagram commutes and the squares are either pushouts (PO) or pullbacks (PB) with injective morphisms.



* *Joint work with* Hartmut Ehrig

The cospan $J \rightarrow F \leftarrow K$ is called borrowed context and can be considered as the minimal context needed in order to complete a partial match of the rule's left-hand side.

We have shown that bisimilarity defined on top of these labelled transitions is a congruence relation, i.e., it is preserved under composition with graph contexts. Furthermore we have presented two proof techniques that help to simplify actual bisimulation proofs. These techniques are "bisimulation up to context" and a technique that avoids checking certain trivial transitions for which only the interface of the graph and the interface of the left-hand side overlap.

It has turned out our results can be generalized to adhesive categories [3] and that the labelled transitions obtained by our construction agree exactly with the transitions obtained in [4] where the notion of minimal context is formulated via a universal property, using so-called (groupoidal) relative pushouts.

The work presented in this talk has appeared in [1] and as a technical report [2].

References

[1] Hartmut Ehrig and Barbara König. Deriving bisimulation congruences in the DPO approach to graph rewriting. In *Proc. of FOSSACS '04*, pages 151–166. Springer, 2004. LNCS 2987.

[2] Hartmut Ehrig and Barbara König. Deriving bisimulation congruences in the DPO approach to graph rewriting. Technical Report 01/2004, Universität Stuttgart, 2004.

[3] Stephen Lack and Paweł Sobociński. Adhesive categories. In *Proc. of FOSSACS '04*, LNCS. Springer, 2004. to appear.

[4] Vladimiro Sassone and Paweł Sobociński. Congruences for contextual graph-rewriting. Technical Report RS-04-11, BRICS, June 2004.

5 Adhesive and Quasiadhesive Categories

Paweł Sobociński

IT University
Copenhagen, Denmark

Adhesive categories have structure ensuring that pushouts along monomorphisms are well-behaved. Many types of graphical structures used in computer science are examples of adhesive categories. Quasiadhesive categories can be viewed as a somewhat weaker notion and enjoy well-behaved pushouts along regular monomorphisms.

The talk will consist of a mini tutorial on adhesive and quasiadhesive categories. We shall also make precise the idea of how such categories provide a natural setting for DPO rewriting.

6 Adhesive High-Level Replacement Categories and Systems: New Abstract Framework for Graph Transformation

Hartmut Ehrig

Adhesive high-level replacement (HLR) categories and systems are introduced as a new categorical framework for graph transformation in a broad sense, which combines the well-known concept of HLR systems with the new concept of adhesive categories introduced by Lack and Sobocinski.

In this paper we show that most of the HLR properties, which had been introduced ad hoc to generalize some basic results form the category of graphs to high-level structures, are valid already in adhesive HLR categories. As a main new result in a categorical framework we show the Critical Pair Lemma for local confluence of transformations. Moreover we present a new version of embeddings and extensions for transformations in our framework of adhesive HLR systems.

7 Coinductive Reasoning for Contextual Graph-Rewriting

Vladimiro Sassone and Paweł Sobociński

We introduce a comprehensive operational semantic theory of graph-rewriting. Graph-rewriting here is meant in a broad sense as we aim to cover and extend previous work based both on Milner's bigraphs and Ehrig and König's rewriting via borrowed contexts. The central idea is recasting rewriting frameworks as Leifer and Milner's reactive systems. Consequently, graph-rewriting systems are associated with canonical labelled transition systems, on which bisimulation equivalence is a congruence with respect to arbitrary graph contexts (cospans of graphs). The central technical contribution of the paper is the construction of groupoidal relative pushouts, introduced and developed by the authors in recent work, in input-linear cospan (bi)categories over arbitrary adhesive categories.

8 Bigraphical Programming Languages

Arne John Glenstrup

Milner and Høgh Jensen's theory of bigraphs captures two of the most important aspects of mobile, distributed systems: location and connectivity.

The Bigraphical Programming Languages project at the IT University of Copenhagen aims to use the theory in practise, designing programming languages based on bigraphs.

Currently, this work in progress has defined a core syntax and static semantics, as well as prototype compilers in Java and ML.

9 Modelling Distributed Systems: some contributions from Pisa

Andrea Corradini

Several research threads have been developed at the Computer Science Department of Pisa within Ugo Montanari's group along the years.

The goal of this talk is to give a brief, personal and partial overview of those research threads which are more closely related to the topics of the Seminar, trying to make explicit their interrelationships. This overview isalso intented to provide a unifying context to the more specific contributions that will be presented during the Seminar by my collegues from Pisa: Marzia Buscemi, Roberto Bruni, Fabio Gadducci, Dan Hirsch, Ivan Lanese and Emilio Tuosto.

The topics I will survey include:

1. The concurrent semantics of Algebraic Graph Transformation Systems deveoped since 1990 also in cooperation with the team of Hartmut Ehrig in Berlin, in large part further developed in Paolo Baldan thesis;
2. The theory of Synchronized Hyperedge Replacement, started in the 1980's with the work on Grammars for Distributed Systems (GDS) by Castellani, Degano and Montanari, and revitalized in the recent years;
3. The theory of Structured Transition Systems, started around 1990 by generalizing the seminal work of "Petri Nets are Monoids" by Meseguer and Montanari;
4. The thery of the Tile Model, introduced around 1997, which generalizes Structured Transition Systems and Plotkin's SOS by equipping rules with pairs of labels (trigger, effect), allowing for vertical (besides horizontal) composition.

10 Stochastic Graph Transformation Systems

Reiko Heckel

To formalize, measure, and predict availability properties, stochastic concepts are required. Reconfiguration and communication in mobile and distributed environments, where due to the high volatility of network connections reasoning on such properties is most important, is naturally described by graph transformation systems.

In this talk we introduce stochastic graph transformation systems, following the outline of stochastic Petri nets. Besides the basic definition and a motivating example, we discuss the analysis of properties expressed in continuous stochastic logic including an experimental tool chain.

11 On Finite Interactive Systems

G. Stefanescu

In this talk we briefly introduce “Interactive systems with Registers and Voices” (shortly, RV-Systems), a new model for interactive systems obtained by applying a Space-Time Duality machinery to register machines.

A register machine consists of a finite automaton for control and registers for storing data. By Space-Time Duality this decomposition gives a decomposition of an RV-System into two parts: a “Finite Interactive System” (shortly, FIS) for its control and interaction part and registers and voices (a “voice” is the time dual of a register) to capture the data on the memory states and on interaction interfaces.

The core of the talk is focusing on FIS’s. We describe FIS’s semantics as sets of grids, compare them with MSC’s and with various devices used for describing 2-dimensional languages, and gives an estimation of the number of words associated to a grid by the flattening operator.

12 CommUnity, Tiles and Connectors*

Roberto Bruni

Dipartimento di Informatica, Università di Pisa, Italia.
bruni@di.unipi.it

The *Categorical Approach* (CA) and the *Algebraic Approach* (AA) are two well known frameworks for the study of complex systems: The first is based on colimit constructions, the second on algebraic operators for composition. In CA, category objects model components and morphisms tell how systems are simulated, refined, etc. Complex systems are modeled as diagrams and composition is achieved via universal constructions (e.g., colimit), which encapsulate components and their interactions in a single object. In AA, constants and operations of a signature model basic processes and their admissible compositions. System behaviors are defined by LTSs in the SOS style. Abstract semantics (e.g. bisimilarity) collapse systems that exhibit the same observable behavior.

Reconciling CA and AA is useful for the mutual transfer of concepts and techniques. As a first step, we investigate the relation between two representatives: *CommUnity* [1] (for CA) and the *Tile Model* [2] (for AA). CommUnity is an architectural description language that provides a conceptual distinction between *computation* and *coordination* concerns in communicating distributed systems. The Tile Model [2] is an operational model for concurrent systems that deals uniformly with closed and open systems and where two dimensions coexist: *distribution* in space and *computation* in time. A translation from CommUnity diagrams into the Tile Model is given in [3], which exploits a novel decomposition of CommUnity diagrams in terms of elementary programs. Tile connectors for synchronization, hiding and mutual exclusion are used to coordinate elementary components. The main result of [3] is that the translation of a diagram is *tile bisimilar* to the translation of its colimit.

In the talk, we establish a stronger link between the colimit construction and the abstract semantics by showing that the encoding of a CommUnity diagram is *equal* to the encoding of its colimit up-to a suitable axiomatization of the connectors.

References

[1] Fiadeiro, J., Maibaum, T.: Categorical semantics of parallel program design. Science of Computer Programming **28** (1997) 111–138

* Joint work with José Luiz Fiadeiro, Ivan Lanese, Antónia Lopes, Ugo Montanari. Research supported by FET-GC Project IST-2001-32747 AGILE on *Software Architectures for Mobility*.

[2] Gadducci, F., Montanari, U.: The tile model. In Plotkin, G., Stirling, C., Tofte, M., eds.: Proof, Language and Interaction: Essays in Honour of Robin Milner. MIT Press (2000) 133–166
[3] Bruni, R., Fiadeiro, J., Lanese, I., Lopes, A., Montanari, U.: New insights on architectural connectors. In: Proc. of IFIP TCS 2004, Kluwer Academics (2004) To Appear.

13 Graph Rewriting for Nominal Calculi*

Fabio Gadducci

Dipartimento di Informatica, Università di Pisa
gadducci@di.unipi.it

The talk introduces a graphical implementation for (possibly) recursive processes of the π-calculus, encoding each process into a (hyper-)graph. The implementation is sound and complete with respect to the standard structural congruence for the calculus: Two processes are equivalent if and only if they are mapped into isomorphic graphs.

Most importantly, the encoding allows for using standard graph rewriting mechanisms in modeling the reduction semantics of the π-calculus. Furthermore, the implementation suggests a possible heuristics for describing graphically the semantics of other calculi with name mobility.

* Research partially supported by the EU within the FET - Global Computing Initiative, project AGILE IST-2001-32747 (*Architectures for Mobility*).

14 Bigraphs and Weak Bisimilarity

Ole Høgh Jensen

Bigraphs are a graphical formalism suited for modelling mobile processes. A main result in bigraphs is the uniform derivation of labelled transitions from (unlabelled) reaction in such a way that the associated strong bisimilarity is always a congruence. This talk addresses the issue of defining uniformly a suitable notion of weak bisimilarity for bigraphs.

15 A Context Logic for Tree Update

Philippa Gardner

Spatial logics have been used to describe properties of tree-like structures (Ambient Logic) and in a Hoare style to reason about dynamic updates of heap-like structures (Separation Logic). We integrate this work by analyzing dynamic updates to tree-like structures with pointers (such as XML with identifiers and idrefs). Naive adaptations of the Ambient Logic are not expressive enough to capture such local updates.

Instead we must explicitly reason about arbitrary tree contexts in order to capture updates throughout the tree. We introduce Context Logic, study its proof theory and models, and show how it generalizes Separation Logic and its general theory BI. We use it to reason locally about a small imperative programming language for updating trees, using a Hoare logic in the style of O'Hearn, Reynolds and Yang, and show that weakest preconditions are derivable. We demonstrate the robustness of our approach by using Context Logic to capture the locality of term rewrite systems.

16 Higher Order Mobile Embedded Resources

Thomas Hildebrandt

We present the calculus of Higher-order Mobile Embedded Resources (Homer), a higher-order calculus with local names and mobile computing resources in nested locations.

The Homer calculus is a simple extention of the core process-passing subset of Thomsen's Plain CHOCS. In particular, it is equipped with reduction and labelled transition semantics, which are simple conservative extensions of the semantics of Plain CHOCS.

We provide strong and weak contextual labelled transition bisimulation equivalences, which is proven to be congruences using Howe's method.

An interesting aspect of the bisimulation due to the presence of copyable computing resources in nested locations with local names is that bisimilar processes must have the same set of free names.

We provide examples of the expressiveness. In particular, we sketch an encoding of pi-calculus name-passing in Homer.

We describe a type system for distinguising between copyable and non-copyable mobile resources in the calculus.

17 Solos in D-Fusion

Björn Victor

Department of Information Technology, Uppsala University, Sweden

The *Solos Calculus* [3] is a "symmetrically asynchronous" version of the (ordinary) fusion calculus [4]. Neither input not output prefixes have continuations, and like in fusion, they are symmetric and non-binding. Interestingly, the solos calculus can still encode the full prefixes of fusion.

The *Solo Diagrams* [2] are a simple graphical formalism corresponding to the solos calculus. Nodes represent names, and edges represent solos. Putting a graph in a box corresponds to replication. Reduction in diagrams correspond 1-1 with reduction in the calculus, and graph isomorphism corresponds to structural congruence.

D-Fusion Calculus [1] is fusion calculus with constants, added by re-introducing the nu binder. In this calculus mixed guarded choice can be encoded, showing its increased expressive power.

By introducing nodes of another colour to solo diagrams, and modifying the reduction rules, we obtain D-Solo Diagrams, corresponding to D-Solos Calculus, a "symmetrically asynchronous" variant of the D-fusion calculus.

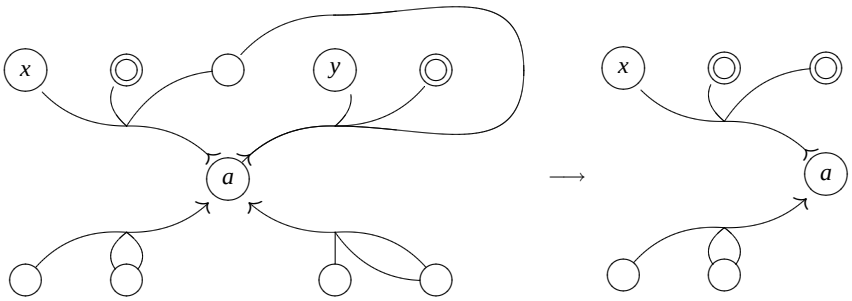


Fig. 17.1. A sample D-Solo diagram reduction: a choice encoding

References

[1] M. Boreale, M. G. Buscemi, and U. Montanari. D-fusion: a fusion calculus with distinction. To appear, 2004.

[2] C. Laneve, J. Parrow, and B. Victor. Solo diagrams. In N. Kobayashi and B. C. Pierce, editors, *Proceedings of TACS 2001*, volume 2215 of *Lecture Notes in Computer Science*, pages 127–144. Springer-Verlag, 2001.

[3] C. Laneve and B. Victor. Solos in concert. *Mathematical Structures in Computer Science*, 13(5):657–683, Oct. 2003. An earlier version appeared in the proceedings of ICALP'99.

[4] J. Parrow and B. Victor. The fusion calculus: Expressiveness and symmetry in mobile processes. In *Proceedings of LICS '98*. IEEE, Computer Society Press, June 1998.

18 D-Fusion: a Distinctive Fusion Calculus

Michele Boreale[1], Maria Grazia Buscemi[2], and Ugo Montanari[2]

[1] Dipartimento di Sistemi e Informatica, Università di Firenze, Italy.
[2] Dipartimento di Informatica, Università di Pisa, Italy.
boreale@dsi.unifi.it {buscemi,ugo}@di.unipi.it

Fusion calculus is commonly regarded as a generalisation of pi-calculus. And, indeed, the pi-calculus transition system can be embedded into Fusion's, provided that one identifies restriction (νx) with the (x) binder of Fusion. However, we claim that this embedding breaks down if comparing the two calculi on the basis of behavioural semantics. We prove that no uniform encoding exists of pi-calculus into Fusion that preserves any 'reasonable' behavioural equivalence (at least as fine as trace equivalence). By 'uniform' we mean homomorphic with respect to parallel composition and name substitution, mapping (νx) to (x) and preserving (a subset of) weak traces. The ultimate reason for this failure is that the binding mechanism of Fusion ignores the issue of uniqueness of newly generated names; in other words, in Fusion all names are like logical variables, namely, unification always succeeds, which is not true in the pi-calculus.

The above considerations motivate the introduction of a new calculus, *D-Fusion*, with two name binders, λ and ν: the former analogous to the only binder of Fusion, and the latter modelling restriction. D-Fusion calculus is at least as expressive as pi-calculus and Fusion separately, and, we strongly argue, *more* expressive than both.

This expressiveness gap is also explored shows up from a more concrete perspective. We a simple security protocol and a related *correlation* property that are readily translated into D-Fusion. The property breaks down if uniformly translating the protocol into Fusion. The failure is illuminating: in Fusion, one has no way of declaring unique fresh names to correlate different messages of the protocol.

We further clarify the gap between D-Fusion and Fusion/pi-calculus by proving that, under mild typing assumptions, there exists a fully abstract encoding of mixed guarded choice into the choice-free fragment of D-Fusion. The idea behind this encoding is that branches of a choice are represented as concurrent processes. Synchronisation is performed in the ordinary way, but it forces a fusion between a λ-name global to all branches and a ν-name local to the chosen branch. Excluded branches are atomically inhibited, since any progress would lead them to fusing two distinct ν-names.

19 Encoding the weak λ-calculus into the Calculus of Explicit Fusions

Tobias Heindel

Institut für Formale Methoden der Informatik, Universität Stuttgart
heindets@fmi.uni-stuttgart.de

We give a compositional encoding of Plotkin's weak call-by-value λ-calculus into a fragment of the Calculus of Explicit Fusions [Wis01] and show that there is a exacte correspondence between the λ-term and its encoding w.r.t. the number of computation steps (β-reductions/reactions). From this follows that the encoding is sound and hence adequate as has been shown by the author in [Hei03].

It is noteworthy that the fragment needed for the encoding consist only of asynchronous output and replicated (non-delayed) input, and of course explicit fusions. The latter play a crucial rôle, since they allow to "glue" together the encodings of the subterms of a λ-term without introducing spurious (reaction) behaviour. During the seminar a conjecture came up that the uniform receptive part of the used fragment of the Calculus of Explicit Fusions could make the encoding fully abstract.

References

[Hei03] Tobias Heindel. Cyclic λ-graph reduction, call-by-need and their process semantics. Master's thesis, Wilhelm-Schickard-Institut, Eberhard Karls Universität Tübingen, 2003.

[Wis01] Lucian Wischik. *Explicit Fusions: Theory and Implementation*. PhD thesis, Computer Laboratory, University of Cambridge, 2001.

20 Abstract Graph Transformation

Arend Rensink[1] and Dino Distefano[2]

[1] Department of Computer Science, University of Twente
P.O.Box 217, 7500 AE, The Netherlands
rensink@cs.utwente.nl
[2] Department of Computer Science, Queen Mary University of London
ddino@dcs.qmul.ac.uk

We study graph-based verification, in which graphs are used to representing program states and execution steps are modelled by derivations from graph production rules. We propose the resulting state space model as an alternative to traditional, state vector-based models.

Except for very small cases, however, the concrete graphs themselves as well as the number of them needed in the model will be too large to make verification practically feasible. One of the most promising techniques to deal with this problem is *abstraction*: by reducing the amount of information in the individual graphs, they will become smaller; moreover, states can be collapsed when their distinctions disappear. An obvious drawback is that the verfication will in all but a very few cases become approximative. Moreover, the effect of the graph transformations must also be lifted to the abstract level, which introduces another possible source of imprecision.

In this work we continue our investigation into a particular abstraction technique in this framework. The state graphs are contracted, using a technique first proposed in [1], by collecting nodes that have similar *local structure*. This results in smaller states and a smaller, in fact finite, state space. A new and (so far) unpublished extension, presented here, is lifting the application of the graph production rules to this abstract level. Since graph abstractions and rule applications can all be computed completely automatically, we believe that this can be the core of a practically feasible technique for software model checking.

References

[1] A. Rensink. Canonical graph shapes. In D. A. Schmidt, editor, *Programming Languages and Systems — European Symposium on Programming (ESOP)*, volume 2986 of *Lecture Notes in Computer Science*, pages 401–415. Springer-Verlag, 2004.

21 Unfolding Techniques for Verifying Graph Transformation Systems

Paolo Baldan[1]
Joint work with
Andrea Corradini[2], Barbara König[3], and Bernhard König[4]

[1] Dipartimento di Informatica, Università Ca' Foscari di Venezia, Italy
baldan@dsi.unive.it
[2] Dipartimento di Informatica, Università di Pisa, Italy
andrea@dsi.unive.it
[3] Institut für Formale Methoden der Informatik, Universität Stuttgart
koenigba@fmi.uni-stuttgart.de
[4] Department of Mathematics, University of California, Irvine, USA
bkoenig@math.uci.edu

Graph transformation systems (GTSs) are recognised as an expressive specification formalism, properly generalising Petri nets and especially suited for concurrent, distributed and mobile systems [7]: the (topo)logical distribution of a system can be naturally represented by using a graphical structure and the dynamics of the system, e.g., the reconfigurations of its topology, can be modelled by means of graph rewriting rules.

The concurrent behaviour of GTSs has been thoroughly studied and a consolidated theory of concurrency for GTSs is available, including the generalisation of several semantics of Petri nets, like process and unfolding semantics (see, e.g., [6, 14, 3]). However, only recently, building on these semantical foundations, some efforts have been devoted to the development of frameworks where behavioural properties of GTSs can be expressed and verified (see [9, 11, 10, 15, 13]).

In this talk we discuss how the unfolding semantics of GTSs can be used as a basis for their formal verification.

For general, possibly *infinite-state*, GTSs one can construct finite structures which provide under- and over- approximations of the (infinite) unfolding, with arbitrary accuracy. Such approximations can be used to check behavioural properties of a GTS, expressed in a suitable temporal graph logic. The logic is a variant of the propositional mu-calculus, where propositional symbols range over *state predicates*, i.e., closed formulae of a monadic monadic second-order logic, characterising static graph properties. Over- and under-approximations allow to check different fragments of the logic at hand, which are characterised through a type system. For details on this approach we refer the reader to [1, 4, 5].

For *finite-state* GTSs, a variant of McMillan's approach (originally developed for Petri nets) [12, 8] allows us to single out a finite under-approximation of the unfolding which is "complete", i.e., which provides an "exact" representation of the behaviour the original system as far as reachability properties are concerned.

Some problems related to the construction of the complete prefix and to its use are discussed. In particular we discuss how a finite complete prefix can be used to check properties of the kind "*eventually* ϕ" and "*always* ϕ", where ϕ is a formula of a monadic second-order logic (interpreted over graphs). This is done by exploiting both the graphical structure underlying the prefix and the concurrency information it provides. The relevant reference for this work is [2].

References

[1] P. Baldan, A. Corradini, and B. König. A static analysis technique for graph transformation systems. In K.G. Larsen and M. Nielsen, editors, *Proceedings of CONCUR 2001*, volume 2154 of *LNCS*, pages 381–395. Springer Verlag, 2001.

[2] P. Baldan, A. Corradini, and B. König. Verifying finite-state graph grammars: an unfolding-based approach. In P. Gardner and N. Yoshida, editors, *To appear in the Proceedings of CONCUR 2004*, LNCS. Springer Verlag, 2004.

[3] P. Baldan, A. Corradini, and U. Montanari. Unfolding and Event Structure Semantics for Graph Grammars. In W. Thomas, editor, *Proceedings of FoSSaCS '99*, volume 1578 of *LNCS*, pages 73–89. Springer Verlag, 1999.

[4] P. Baldan and B. König. Approximating the behaviour of graph transformation systems. In A. Corradini, H. Ehrig, H.-J. Kreowski, and G. Rozemberg, editors, *Proceedings of ICGT'02*, volume 2505 of *LNCS*, pages 14–30. Springer Verlag, 2002.

[5] P. Baldan, B. König, and B. König. A logic for analyzing abstractions of graph transformation systems. In R. Cousot, editor, *Proceedings of SAS'03*, volume 2694 of *LNCS*, pages 255–272. Springer Verlag, 2003.

[6] A. Corradini, U. Montanari, and F. Rossi. Graph processes. *Fundamenta Informaticae*, 26:241–265, 1996.

[7] H. Ehrig, J. Kreowski, U. Montanari, and G. Rozenberg, editors. *Handbook of Graph Grammars and Computing by Graph Transformation, Vol. 3: Concurrency, Parallelism and Distribution*. World Scientific, 1999.

[8] J. Esparza. Model checking using net unfoldings. *Science of Computer Programming*, 23(2–3):151–195, 1994.

[9] F. Gadducci, R. Heckel, and M. Koch. A fully abstract model for graph-interpreted temporal logic. In H. Ehrig, G. Engels, H.J. Kreowski, and G. Rozenberg, editors, *Proceedings of TAGT'98*, volume 1764 of *LNCS*, pages 310–322. Springer Verlag, 2000.

[10] R. Heckel. Compositional verification of reactive systems specified by graph transformation. In E. Astesiano, editor, *Proceedings of FASE'98*, volume 1382 of *LNCS*, pages 138–153. Springer Verlag, 1998.

[11] B. König. A general framework for types in graph rewriting. In *Proc. of FST TCS 2000*, volume 1974 of *LNCS*, pages 373–384. Springer Verlag, 2000.

[12] K.L. McMillan. *Symbolic Model Checking*. Kluwer, 1993.

[13] A. Rensink. Towards model checking graph grammars. In M. Leuschel, S. Gruner, and S. Lo Presti, editors, *Proceedings of the* 3rd *Workshop on Auto-*

mated Verification of Critical Systems, Technical Report DSSE–TR–2003–2, pages 150–160. University of Southampton, 2003.

[14] L. Ribeiro. *Parallel Composition and Unfolding Semantics of Graph Grammars*. PhD thesis, Technische Universität Berlin, 1996.

[15] Dániel Varró. Towards symbolic analysis of visual modelling languages. In P. Bottoni and M. Minas, editors, *Proc. GT-VMT 2002: International Workshop on Graph Transformation and Visual Modelling Techniques*, volume 72 of *Electronic Notes in Computer Science*, pages 57–70. Elsevier, 2002.

22 Shaped Hierarchical Architectural Design

Dan Hirsch and Ugo Montanari

Dipartimento di Informatica, Universitá di Pisa

The architectural design of systems deals with the high level structuring of configurations. Checking that a system belongs to an architectural style (or shape) implies that the architecture is an instance of a structurally defined class. On the other side, hierarchies allow modeling at different levels of detail: subsystems may be represented as single components to abstract structure and behavior.

This talk presents an approach for representing hierarchical software architecture shapes using types [1]. Typing proofs define a general framework based on inference rules where shape rules and graphs representing system configurations are represented as type judgements [2]. Therefore, if there is a typing proof for a judgment, then the system is correctly shaped (i.e. typed), where the axioms of the type system are the shaping rules of a style.

An aspect strongly related to shape is *SA reconfiguration*. Reconfiguration has to respect shape, i.e. type. But for design, just observing the actual configuration may not be enough. Instead, observing the steps taken to obtain the final system may provide important information about the process of construction. We claim that proof terms (i.e., terms of rule names encoding typing proofs) provide more information than just graphs about the process of constructing systems and allow to specify reconfigurations as proof term rewritings. Then, reconfiguration consistency is obtained as subject reduction: as long as cutting and pasting typing proofs still yields typing proofs, subject reduction is guaranteed.

Also, our approach allows the integration of shapes and hierarchies. Hierarchical composition allows to describe systems at different levels of detail. Hierarchical structures are present in many aspects related with system configuration, and in areas like process calculi (Ambient Calculus), concurrent system modeling (Bigraphs), UML (e.g. state charts with decomposition and refinement). In our case, hierarchical structure is captured via hierarchy constructors which are similar to basic constants, i.e. only type and name is specified. Then for each hierarchical constructor a standard ”symbolic” body is defined as a type judgment. Hierarchical graphs can be derived in the resulting type system.

References

[1] Hirsch, D. and Montanari, U. Shaped Hierarchical Architectural Design. In *GT-VMT 2004* (ETAPS 2004 workshop), ENTCS, to be published, 2004.

[2] Hirsch, D. *Graph Transformation Models for Software Architecture Styles*. PhD thesis, Dept. of Computer Science, Universidad de Buenos Aires, May 2003.

23 Old names for nu

Lucian Wischik

Microsoft

How should one understand *restriction* $(\nu x)P$ in the pi calculus? From CCS tradition it declares that the name x that occurs in P is hidden: no one outside the scope of $(\nu x)\_$ can communicate on it. But pi also allows for *scope extrusion*, for when P sends the name x to some third party who was outside its initial scope. To deal with this syntactically, pi adds rules for scope extrusion and alpha-renaming and it allow interaction to take place inside the scope of $(\nu x)\_$.

This syntactic baggage of scope extrusion is largely bypassed in graphical accounts of the pi calculus. This has been explained with in several of the current Dagstuhl presentations, for instance in Milner's *bigraphs*. It is bypassed because, graphically, scopes are unnamed and the extent of their scope is implicit.

For someone implementing the pi calculus, restriction has an entirely different meaning: it is not a declaration but a *command* which, when executed, generates a fresh name:

$$((\nu x)P) \mid R \quad \rightarrow \quad P\{x'/x\} \mid R \qquad x' \text{ fresh}$$

In this respect it is much like *malloc* from C, or the *socket* command in TCP. Using this *fresh name semantics* for pi also bypasses the syntactic baggage of scope extrusion: it is enough to add the communication rule

$$\overline{u}x.P \mid u(y).Q \mid R \quad \rightarrow \quad P \mid Q\{x/y\} \mid R$$

and assert that a pi term is a multiset with atoms separated by the parallel operator $|$. These two rules constitute a complete semantics of the pi calculus: they do not need scope extrusion or alpha-renaming, and they do not need to be closed under scopes. Multisets have previously been used for pi (*Engelfriet, 'A multiset semantics for pi', TCS 153(1-2):65-94, 1996*), but in the absence of fresh name semantics they had to use infinite multisets.

Why does this 'fresh name semantics' bypass the syntactic baggage of scope extrusion? It is because, similar to graphical presentations, scopes have globally unique names and the extent of their scope is implicit.

Fresh-name creation for pi has appeared previously, notably in the pi implementation Pict (*Turner, 'The Polymorphic Pi-Calculus', PhD thesis, Edinburgh 1996*). Actually it is ubiquitous in implementations of concurrent systems, for instance also CML (*Reppy, 'CML', ACM SIGPLAN 26(6):293-305, 1991*) and Klaim (*Nicola, Ferrari, Pugliese, 'Klaim', IEEE trans Soft.Eng. 24(5):315-330, 1998*). The join calculus also has a fresh name semantics as well as a traditional semantics (*Fournet,*

Gonthier, 'The reflexive chemical abstract machine and the join calculus', POPL 1996).

It might seem obvious that fresh name semantics and traditional pi semantics coincide, but to no one previously has made a complete proof for either join or pi. The original contribution of the current author is to demonstrate formally, with proofs for operational correspondence and full abstraction, that the two semantics do coincide. The essence of the proof is to turn each statement *'X can be alpha-renamed to Y'* into *'Given an execution history which produced X, there must have been possible an alternative execution history which produced Y.'* This is proved with an intermediate calculus which keeps a top-level record $(|\widetilde{x}|)$ of which names have been generated so far in the entire history of the computation: the *'old names'*. The intermediate calculus has appeared previously (*Gardner, Laneve, Wischik, 'The Fusion Machine', LNCS2421:418-433*). We believe that a vastly simpler proof can be found using the Gabbay-Pitts operator (*Gabbay, Pitts, 'A new approach to abstract syntax with variable binding', Formal Aspects of Computing 13(0):1-23, 2001*).

24 Synchronizations with Mobility for Graph Transformations

Ivan Lanese and Ugo Montanari

Computer Science Department, University of Pisa, Italy
{lanese,ugo}@di.unipi.it

Our work is aimed at developing high-level models for complex global computing systems. At the high level of abstraction, the primitives made available by the underlying middleware are used to coordinate the behaviours of different computational entities. Important features of such a kind of model are: (i) expressiveness, since complex forms of interaction have to be modelled; (ii) compositionality, which is required to allow a bottom-up development of systems.

We propose an approach based on graph transformations, which extends Synchronized Hyperedge Replacement (SHR) [2]. The main advantage of SHR is that it can be easily implemented in a distributed setting. In fact, transformations are specified by defining the behaviour of single hyperedges via productions with local effects. Productions interact via synchronizations on common nodes and exchange data via name mobility in the Fusion Calculus style.

We extend standard SHR by parameterizing the inference rules that are used to compose productions using a generalization of synchronization algebras that allows to specify the underlying synchronization model [4]. Thus by changing the synchronization algebra one can model systems based on different middlewares. The proposed extension adds structures suitable to deal with mobility and local resources, which are two main features of global computing systems.

We formalize the model using a Labelled Transition System (LTS) where labels contain synchronizations and names of nodes. We prove that the resulting bisimilarity is a congruence w.r.t. the operators of a suitable algebra of graphs by showing that the LTS can be specified in De Simone format [1]. As an example, we study the equivalence of different implementations of dynamic routers.

As a larger example, we show how to map Fusion Calculus into our model [3] and that this mapping provides a concurrent compositional semantics for it.

References

[1] M. Buscemi and U. Montanari. A first order coalgebraic model of pi-calculus early observational equivalence. In *Proc. CONCUR'02*, *LNCS* 2421, pages 449–465.

[2] G. Ferrari, U. Montanari, and E. Tuosto. A LTS semantics of ambients via graph synchronization with mobility. In *Proc. of ICTCS'01*, *LNCS* 2202, pages 1–16.

[3] I. Lanese and U. Montanari. A graphical fusion calculus. In *Proceedings of CoMeta Final Workshop*, ENTCS, 2003. To appear.

[4] I. Lanese and U. Montanari. Synchronization algebras with mobility for graph transformations. In *Proc. FGUC'04*, ENTCS, 2004. To appear.

25 Generalizing Interaction Nets: which generalization for which properties

Lionel Khalil[1] and Maribel Fernandez[2]

[1] LIPN, Institut Galilée,
Université Paris 13
99 av. J-B. Clément, 93430 Villetaneuse, France
lionel.khalil@lipn.univ-paris13.fr
[2] Computer Science
King's College London
Strand, London WC2R 2LS, U.K.
maribel@dcs.kcl.ac.uk

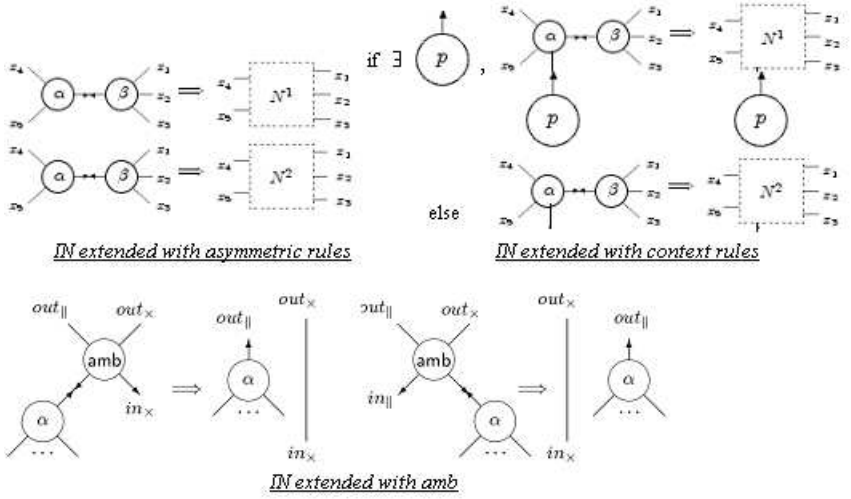
Interaction nets, introduced by Lafont [2], consists of a graph with agents at the nodes, and a set of graph rewriting rules which specify the interaction between two agents connected through their principal ports (each agent has a unique principal port, and there is a unique rule for each pair of agents). However, they are intrinsically deterministic and this prevents from applying these techniques to concurrent languages where non-determinism plays a key rôle.

As an example of a non-deterministic process, we consider a parallel merge: it can be specified in three ways, called *fair merge*, *angelic merge*, and *infinity merge* [5]. All the merge primitives have a pair of input sequences and one output sequence. The elements of the input sequences appear unaltered in the output sequence, and their relative order in the input sequence is preserved (but elements from different input sequences can appear in any order in the output). The difference between these primitives is that, in a fair merge, every element of an input sequence will eventually appear in the output, whereas for an angelic merge all that is guaranteed is that the output sequence is infinite if at least one of the input sequences is infinite. The infinity merge has the dual property: it guarantees that if one of the input sequences is infinite then all the elements of the other one will appear in the output.

Panangaden [4] has proved that infinity merge can be implemented with angelic merge and that angelic merge can be implemented using fair merge. Moreover, these three levels of expressivity are fundamentally different: fair merge cannot be implemented by angelic merge, which in turn cannot be implemented by infinity merge.

Our aim is to increase the expressive power of the interaction net framework, but remaining as close as possible to the original definition. A first extension, called IN with asymmetric rules, would be to allow two rules randomly chosen for the same left member. A second extension, called IN extended with context rules, is to allow a conditional selection between two rules depending on the context of the left-hand side rule. In a third extension, called IN extended with amb, we add one agent with two principal ports (used as inputs) and two auxiliary ports. The agent,

which we call amb inspired by McCarthy's work [3], is defined by rules as shown below, where α is any agent and described in [1].



We arrived to the following result. IN with asymmetric rules can be implemented with IN extended with amb, which can be implemented IN extended with context rules. IN with asymmetric rules can implement infinity merge, but not angelic merge and fair merge. IN extended with amb can implement infinity and angelic merge but not fair merge. IN extended with context rules can implement infinity, angelic merge and fair merge.

References

[1] M. Fernández and L. Khalil. Interaction Nets with McCarthy's amb. *Electronic Notes in Theoretical Computer Science, vol.68(2), 2002. Proceedings of the 9th Int. Workshop on Expressiveness in Concurrency, EXPRESS'02, Brno, Czech Republic.*

[2] Y. Lafont. Interaction nets. In *Proceedings, 17th ACM Symposium on Principles of Programming Languages*, pages 95–108, 1990.

[3] J. McCarthy. A basis for a mathematical theory of computation. In P. Braffort and D. Hirschberg, editors, *Computer Programming and Formal Systems*, pages 33–69. North Holland, 1963.

[4] P. Panangaden and V. Shanbhogue. The expressive power of indeterminate dataflow primitives. *Information and Computation*, 1992.

[5] D. Park. The fairness problem and non-deterministic computing networks. In *Proceedings of the 4th Advanced Course on Theoretical Computer Science*. Mathematisch Centrum, 1982.