State Space Reduction For Parity Automata

Christof Löding and Andreas Tollkötter

RWTH Aachen, Lehrstuhl 7 für Informatik

Abstract. Exact minimization of ω-automata is a difficult problem and heuristic algorithms are a subject of current research. We establish a framework to generalize the known notion of quotient automata and uniformly describe such algorithms. We investigate several approaches to reduce the state space of deterministic parity automata. These are based on extracting information from structures within the automaton, such as strongly connected components, coloring of the states, and equivalence classes of given relations, to determine states that can safely be merged. The description of these procedures consists of a theoretical analysis as well as data collected from experiments.

Keywords: foo · bar

1 Introduction

Finite automata are a long established computation model that dates back to sources such as [9] and [12]. A known problem for finite automata is state space reduction, referring to the search of a language-equivalent automaton which uses fewer states than the original object. For deterministic finite automata (DFA), not just reduction but minimization was solved in [6]. Regarding nondeterministic finite automata (NFA), [7] proved the PSPACE-completeness of the minimization problem, which is why reduction algorithms such as [3] and [1] are a popular alternative.

In his prominent work [2], Büchi introduced the model of Büchi automata (BA) as an extension of finite automata to read words of one-sided infinite length. As these ω-automata tend to have higher levels of complexity in comparison to standard finite automata, the potential gain of state space reduction is even greater. Similar to NFAs, exact minimization for deterministic Büchi automata was shown to be NP-complete in [13] and spawned heuristic approaches such as [13], [8], or [4].

As [14] displays, deterministic Büchi automata are a strictly weaker model than nondeterministic Büchi automata. It is therefore interesting to consider different models of ω-automata in which determinism is possible while maintaining enough power to describe all ω-regular languages. Parity automata (PA) are one such model, a combination of Büchi automata and Moore automata ([10]), that use a parity function, assigning a number (called priority or color) to each state. The convention used in this paper is that even priorities correspond to "good" states while odd priorities correspond to "bad" states. The smallest priority that is seen infinitely often during a run defines its acceptance; if it is even, the run will be accepting, and if it is odd, the run will be rejecting. [11], [15] showed that deterministic parity automata are indeed sufficient to recognize all ω-regular languages. As for DBAs, the exact minimization problem for DPAs is NP-complete ([13]).

In this thesis, we investigate multiple techniques of heuristic state space reduction for DPAs. We present a general framework to uniformly describe these algorithms in a transparent manner. As opposed to a black box which provides as output a reduced automaton for each input, the framework aims towards giving more precise information about how the reduced DPA comes to be. This in theory makes it easier to find structures in the DPAs which are responsible for the reduction being applicable. The base of the framework was the notion of quotient automata, which, for a given congruence relation on the states, merges all states of each equivalence class into one single representative. For example, building the quotient automaton of a DFA with the relation defined by the Myhill-Nerode theorem [] will yield the minimal DFA for that language.

Our proposal for this framework are merger functions which generalize quotient automata. Merger functions map sets of states in the original DPA, called the merge set, to other sets of states, called the candidate set. The easiest interpretation, which we refer to as representative merge, allows us to merge all states from the merge set into any single representative that is chosen from the candidate set.

The most basic merge simply adapts the algorithm from [6]. Every parity automaton can be interpreted as a Moore automaton which can then be minimized using said algorithm. In this context, we call the equivalence relation which considers two states to be equivalent if they are merged by this algorithm the *Moore equivalence*. That relation is used or modified at several later points in the thesis.

A simple merger function that is introduced is the *skip merger* which takes an equivalence relation that implies language equivalence on the states as a parameter and from that builds a merger function. The idea of using one given equivalence relation on the states and refining it is used several times in the thesis. This merge decides on one particular strongly connected component (SCC) of the automaton and removes all states of an equivalence class that do not lie in this SCC, essentially "skipping" all other SCCs.

We adapt the works of [5], who worked on alternating automata, to our case of deterministic parity automata and find that there are sufficient differences to warrant a separate analysis. The *delayed simulation merger* considers two states to be equal, if on every run from those states on a shared word, if one run visits a priority at some position, the other run must visit a priority at most as high at some point in the future. It then holds that both runs will see the same smallest priority infinitely often and therefore either both accept or both reject. It suffices to choose one representative of each such equivalence class that has minimal priority and build the quotient automaton with those representatives.

The *iterated Moore* merge uses the idea that on ω-automata, states which are only visited finitely often are irrelevant to the acceptance of a state. In particular, trivial SCCs, i.e. states with no path back to itself, can have their priority freely changed without affecting the language of the automaton. The idea of this merger therefore is to loosen the constraints on those states and then build the usual Moore equivalence.

We bring up merging via *path refinement*, which uses an existing equivalence class of some congruence relation and refines it to a point where states can safely be merged. This refinement occurs so that two states from the class are equivalent if on each path from that state back to the class, both states visit the same minimal priority.

Another merger function is called *threshold Moore*, which again refines an existing relation. Two states are considered to be equivalent under the refinement if a relaxation of Moore equivalence, that considers all priorities greater than some k to be equal, matches them. In this case we require the two states to have equal priority and choose k to be that exact value.

A similar but slightly different approach is the *LSF* merger function. It removes the need for two states to have equal priority and instead takes the value k as a parameter. On the other side it adds a requirement similar to that of the skip merger, in that the candidates of the merge are those that all lie in one single SCC if we modify the automaton to only contain those states of priority at least k.

The thesis is structured as follows: In the first chapter, we define basics notations and conventions and present some data of our empirical testing environment used to collect practical data of each approach. In the second chapter, we establish some theoretical ground work that will be used by the rest of the thesis. After that, we present the algorithms for state space reduction, split into one chapter for each such procedure. Each such chapter is made up of at least three sections. The first section describes the idea and definition of the algorithm and proofs well behaving properties. The second section covers how to actually compute the reduction and provides a short run time analysis. The third section shows practical data to analyze "real world" usefulness. Potential other sections contain variants or extensions of the original procedure as well as potential open questions. At the very end, the appendix includes small examples to show how the individual merger functions are computed and work.

2 Merger Functions

Our definition of DPAs is a quadruple (Q, Σ, δ, c), where Q is the finite set of states, Σ is the finite set of symbols (the alphabet), $\delta : Q \times \Sigma \to Q$ is the deterministic transition function, and $c : Q \to \mathbb{N}$ is the priority function. The set of ω-words over Σ is then denoted by Σ^ω.

Definition 1. *Let $\mu : D \to (2^Q \setminus \{\emptyset\})$ be a function for some $D \subseteq 2^Q$. We call μ a* merger function *if all sets in D are pairwise disjoint and for all sets $M \in D$, $\mu(M) \cap (\bigcup D \setminus M) = \emptyset$.*

A representative merge *of $\mathcal{A}$ w.r.t. μ is constructed by choosing a representative $r_M \in \mu(M)$ for all $M \in D$ and then removing all states in $M \setminus \{r_M\}$. Transitions that originally lead to one of the removed states are redirected to the representative r_M instead.*

While merger functions are a generalization of the more restrictive combination of congruence relation and quotient automaton, we still often build up our various merger functions from the basis of an equivalence relation. We briefly go over a few cases of relations that are of interest in this context before moving on to the first reduction technique.

We consider several types of different relations, mostly over the state domain Q. A relation R is a preorder if it is reflexive and transitive. R is an equivalence relation if it is a symmetric preorder. R is a congruence relation if it is an equivalence relation that is compatible with δ, i.e. if $(p, q) \in R$, then also $(\delta(p, a), \delta(q, a)) \in R$ for all $a \in \Sigma$.

If $\sim$ is an equivalence relation and $\mathcal{A}$ is a DPA, we write $\mathfrak{C}(\tilde{)} \subseteq 2^Q$ for the set of equivalence classes in $\mathcal{A}$.

Definition 2. *The* language equivalence relation *is defined by $p \equiv_L q$ iff reading every ω-word α from either p or q gives the same acceptance.*

The priority almost equivalence relation *is defined by $p \equiv_{\dagger} q$ iff reading every ω-word α from either p or q yields two runs that differ in priorities at only finitely many positions.*

The Moore equivalence relation *is defined by $p \equiv_M q$ iff reading every finite word w from either p or q ends up in states with the same priority.*

All three of these equivalence relations imply language equivalence between states. However, only Moore equivalence is strong enough of a contract to allow for immediate merging of states.

Merging states according to $\equiv_\dagger$ or $\equiv_L$ can change the language of the DPA. The idea of using already defined relations that imply language equivalence but are not strong enough to allow merging on their own and then refining those relations into finer equivalence classes can be seen in multiple of our techniques, such as the skip merger, path refinement, and LSF merger.

In fact, building the quotient automaton w.r.t. $\equiv_M$ is the canonical way to minimize a deterministic Moore automaton. We can express the same by a merger function to reduce the state space of a DPA.

Definition 3. *The* Moore merger function *is defined as* $\mu_M : \mathfrak{C}(\equiv_M) \to 2^Q$ *with* $\mu_M(\kappa) = \kappa$.

Theorem 1. *A representative merge of a DPA w.r.t.* μ_M *is language equivalent to the original.*

3 Skip Merger

The skip merger is the first reduction algorithm we want to introduce. It can be seen more of a first proof of concept rather than a practical novelty, as the underlying idea is neither complex nor had it great effect during empirical tests.

The skip merger considers the different strongly connected components (SCCs) of the automaton. An SCC is a set of states in which each element can reach every other via some path. We sometimes speak of the "deepest" SCCs with a certain property, which are those SCCs such that no other SCC with that property is reachable anymore.

If we have an equivalence relation that implies language equivalence but is not strong enough to warrant a merge of states on its own, each equivalence class of that relation only requires its states in the deepest SCC. We can therefore redirect any transition that would move the automaton to a state to a representative in a deepest SCC instead.

To formally capture this idea, the *reachability preorder* is defined as $p \preceq_{\text{reach}} q$ if and only if q is reachable by some path from p. Reachability can be computed in $\mathcal{O}(|Q|^3)$ which, in general, is too high of a complexity to efficiently deal with. It is sufficient to use a total extension of reachability though, which is a minimal superset of the reachability preorder that is a total preorder itself. Such a relation can be computed in linear time by a topological sorting on the SCCs of the automaton.

Definition 4. *Let* $\sim$ *be a congruence relation on* Q. *Let* $\preceq$ *be a total extension of reachability. For each* $\kappa \in \mathfrak{C}$, *we define* $C_\kappa \subseteq \kappa$ *to be the set of* $\preceq$*-maximal states and* $M_\kappa = \kappa \setminus C_\kappa$.

We define the skip merger function $\mu^{\sim}_{skip} : \{M_\kappa \mid \kappa \in \mathfrak{C}(\sim)\} \to 2^Q$ *with* $\mu^{\sim}_{skip}(M_\kappa) = C_\kappa$.

Theorem 2. *If* $\sim$ *implies language equivalence, then a representative merge of a DPA w.r.t.* $\mu^{\sim}_{skip}$ *is language equivalent to the original.*

Proof. If we have two runs, π and ρ, of the original DPA and the merged DPA on the same word α, we can observe that at every position, $\pi(i)$ and $\rho(i)$ will be $\sim$-equivalent, as $\sim$ is a congruence relation. Furthermore, as there are only finitely many SCCs in an automaton, ρ will eventually reach a point j from which on only transitions are taken that also exist in the original DPA. As $\pi(j) \sim \rho(j)$, that means that the two runs must have the same status of acceptance. $\square$

Theorem 3. *For a given* $\sim$ *in a suitable data structure,* $\mu^{\sim}_{skip}$ *can be computed in* $\mathcal{O}(|Q|)$.

4 Delayed Simulation

Based on [5], we consider reduction via delayed simulation. The original paper uses alternating parity automata, a much stronger model compared to our DPAs. Limiting ourselves to this special case not only reduces the run time of the algorithm but allows for a fundamentally better approach to improve the amount of reduction.

Similarly to other simulation techniques for reduction, the concept of delayed simulation is to consider two states to be mergeable if both "simulate" the behavior of the respective other. Delayed simulation in particular considers simulation in the sense that on every path, both states eventually see the same smallest priority.

Definition 5. *We define the* delayed simulation equivalence relation *as $p \equiv_{de} q$ if and only if the following property holds for all $w \in \Sigma^*$: Let $p' = \delta^*(p, w)$ and $q' = \delta^*(q, w)$. Every run in the automaton that starts in p' or q' eventually sees a priority less than or equal to $\min\{c(p'), c(q')\}$.*

$\equiv_{\mathrm{de}}$ is a congruence relation that implies language equivalence but states that are $\equiv_{\mathrm{de}}$-equivalent do in general not have the same priority. It is therefore not trivial to see that equivalent states can be merged. In fact, the merger function is not as simple as a quotient automaton. Rather, we have to add the additional requirement for the chosen representative state to have minimal priority.

Definition 6. *We define the* delayed simulation merger function $\mu_{de} : \mathfrak{C}(\equiv_{de}) \to 2^Q$ *with* $\mu_{de}(\kappa) = \{q \in \kappa \mid c(q) = \min c(\kappa)\}$.

Theorem 4. *A representative merge of a DPA w.r.t. μ_{de} is language equivalent to the original.*

Proof. The main idea of the proof is to show that for two states $p \equiv_{\mathrm{de}} q$ with $c(p) < c(q)$, we can set the priority of q to $c(p)$ without changing the language of the automaton. Then we can build an equivalent DPA to the original in which $\equiv_{\mathrm{de}}$ implies priority equivalence, which shows that the quotient automaton of that DPA is language equivalent to the original. That quotient automaton is equivalent to our merger function μ_{de}.

To see that the priority of q can be changed, assume there is a run ρ of $\mathcal{A}$ on some ω-word α such that changing the priority of q to $c(p)$ also changes the acceptance status of ρ. As the argument works symmetrically, we can assume that $c(p)$ is even. We call the DPA with modified priorities $\mathcal{A}'$ with function c'. As the priority of every state in $\mathcal{A}$ is at least as good as in $\mathcal{A}'$, we can assume that $c(\rho)$ is accepting and $c'(\rho)$ is rejecting.

ρ must visit q infinitely often and in $c'(\rho)$, $c'(q)$ must be the smallest priority. Otherwise, the two runs would have the same smallest priority that occurs infinitely often. Hence, there is a word w such that reading w from q moves back to q and only priorities greater than $c'(q)$ are seen in between.

Now consider the two runs π_p and π_q on the word w^ω starting in p and q respectively. By choice of w, $c(\pi_q)$ only visits priorities strictly greater than $c'(q)$. On the other hand, π_p starts at state p with $c(p) = c'(q)$. As $p \equiv_{\mathrm{de}} q$, $c(\pi_q)$ must see a priority of at most $c(p)$ on all paths eventually. This is a contradiction and the run ρ cannot exist. □

Computation of delayed simulation is not trivial, especially if it is to be done efficiently. Taken from the original paper [5], our approach is to use a Büchi automaton and reduce $\equiv_{\mathrm{de}}$ in the original DPA to language acceptance in that automaton.

The Büchi automaton, called $\mathcal{G}_{\mathrm{de}}$, is a product automaton with an additional component that it uses to track the visited priorities. This third component, called "obligations", makes sure that whenever a priority occurs in the first component, the second component eventually has to simulate one at most as large. For each state pair p, q we can then use $\mathcal{G}_{\mathrm{de}}$ to determine whether q simulates p. If this holds in both directions, the two states are $\equiv_{\mathrm{de}}$-equivalent.

Definition 7. *We define the deterministic Büchi automaton* $\mathcal{G}_{de} = (Q_{de}, \Sigma, \delta_{de}, F_{de})$ *as*

- $Q_{de} = Q \times Q \times (c(Q) \cup \{\checkmark\})$
- $\delta_{de}((p, q, k), a) = (p', q', \gamma(c(p'), c(q'), k))$, *where* $p' = \delta(p, a)$ *and* $q' = \delta(q, a)$
- $F_{de} = Q \times Q \times \{\checkmark\}$.

The obligation function γ *is defined as*

$$\gamma(i, j, k) = \begin{cases} \checkmark & \text{if } j \leq i \text{ and } j \leq_{\checkmark} k \\ \min_{\leq_{\checkmark}}\{i, k\} & \text{else} \end{cases}$$

where $0 \leq_{\checkmark} 1 \leq_{\checkmark} 2 \leq_{\checkmark} \cdots \leq_{\checkmark} \checkmark$.

Now using this automaton, we can relate delayed simulation to the question of universal language. A state is *language universal* if starting from it, every path is accepted.

Theorem 5. *For two states* p *and* q*, let* $q^0_{de}(p, q) = (p, q, \gamma(c(p), c(q), \checkmark))$*. Then* $p \equiv_{de} q$ *if and only if* $q^0_{de}(p, q)$ *and* $q^0_{de}(q, p)$ *are language universal states.*

The proof of this theorem is just a technical comparison between the theoretical definition of $\equiv_{\mathrm{de}}$ and the algorithmic definition of $\mathcal{G}_{\mathrm{de}}$. We omit it, as no particular insight is gained.

Theorem 6. μ_{de} *can be computed in* $\mathcal{O}(|Q|^2 \cdot |c(Q)|)$.

Proof. Assuming that we can compute $\equiv_{\mathrm{de}}$ in a suitable data structure in the described time, building μ_{de} from that is rather trivial. To see how we compute $\equiv_{\mathrm{de}}$, observe that the size of $\mathcal{G}_{\mathrm{de}}$ is $\mathcal{O}(|Q|^2 \cdot |c(Q)|)$. The set of language universal states in a DBA can be computed in linear time: we are looking for loops in the subgraph that only consists of the non-accepting states. Then every state from which such a loop is reachable is not language universal. These operations can all be done with classic graph operations such as depth first search.

5 Path Refinement

The upcoming technique, called congruence path refinement or just path refinement, is again one which takes in an already defined relation on the states and refines it to a point where merging classes is a valid operation. To be precise, we define the PR-equivalence for each equivalence class independently. Given such a class, which we call λ, we consider "loops" which move from λ back to some state in λ via some finite path. If two states in λ see the same smallest priority on every such loop and we can guarantee that all states they will reach via the loop have the same property, then every run that visits λ infinitely often can be divided into path segments on which runs starting in these two states see the same smallest priority. As this then holds for every segment in the run, the set of infinitely often occurring priorities must then also have the same minimum.

Formally, we describe these loops in a set called $L_{\lambda\hookleftarrow}$. This is the set of all finite words, such that reading them from λ will move back to λ at the end and never before.

Definition 8. *Let $\sim$ be a congruence relation and let $\lambda \in \mathfrak{C}(\sim)$ be an equivalence class. We define a relation R_λ on λ as $(p, q) \in R_\lambda$ if and only if for all $w \in L_{\lambda\hookleftarrow}$, the smallest priority seen on the path induced by w is the same starting from p and from q.*

We define path refinement equivalence $\equiv^\lambda_{PR}$ *on λ as the largest subset of R_λ such that $p \equiv^\lambda_{PR} q$ if and only if for all $w \in L_{\lambda\hookleftarrow}$, $\delta^*(p, w) \equiv^\lambda_{PR} \delta^*(q, w)$.*

Again, note that this relation is only defined on λ. Using the merger function below, one can perform this reduction independently on every class in $\mathfrak{C}(\sim)$.

Definition 9. *We define the* path refinement merger function $\mu^\lambda_{PR} : \mathfrak{C}(\equiv^\lambda_{PR}) \to 2^Q$ *with* $\mu^\lambda_{PR}(\kappa) = \{q \in \kappa \mid c(q) = \min c(\kappa)\}$.

Theorem 7. *If all states in λ are pairwise language equivalent, then a representative merge of a DPA w.r.t. μ^λ_{PR} is language equivalent to the original.*

Proof.

As for delayed simulation, the definition of path refinement is more theoretical and less constructive. We have to dedicate some additional thought to the question of how to actually compute $\mu^\lambda_{\mathrm{PR}}$. We use a similar approach as before and reduce the computation of PR-equivalence to a known automata problem, which in this case is the Moore equivalence on DPAs.

A direct translation of the definition to an algorithm would be a similar automaton as for delayed simulation. One can build a deterministic finite product automaton with an additional third component that tracks the smallest priority so far and which component it was seen in, and at every visit to λ makes sure that the tracked values coincide.

The *visit graph* that we now define instead is a less intuitive solution but has a size only linear in $|Q|$ instead of quadratic. It also uses a "tracker" of the smallest priorities between one visit to λ and the next but only does so for each state individually instead of tracking each state pair.

Definition 10. *The visit graph is a DPA $(Q^\lambda_{visit}, \Sigma, \delta^\lambda_{visit}, c^\lambda_{visit})$ defined with*

- $Q^\lambda_{visit} = Q \times c(Q) \times (c(Q) \cup \{\perp\})$
- $delta^\lambda_{visit}((q, k, k'), a) = \begin{cases} (q', \min\{c(q'), k\}, \perp) & \text{if } q' \notin \lambda \\ (q', c(q'), \min\{c(q'), k\}) & \text{if } q' \in \lambda \end{cases}$, *where* $q' = \delta(q, a)$
- $c_{visit}((q, k, k')) = k'$

Theorem 8. *For a state $q \in Q$, let $\iota_q = (q, c(q), \max c(Q)) \in Q^\lambda_{visit}$. Then $p \equiv^\lambda_{PR} q$ if and only if $\iota_p \equiv_M \iota_q$.*

Proof.

Theorem 9. $\equiv^\lambda_{PR}$ *can be computed in* $\mathcal{O}(|Q| \cdot |c(Q)|^2 \cdot \log |Q|)$.

Proof. Moore equivalence can be computed in time $\mathcal{O}(n \log n)$ of its automaton ([6]). The visit graph has size $n \in \mathcal{O}(|Q| \cdot |c(Q)|^2)$. As $|c(Q)|$ is always at most $|Q|$, this gives us the desired complexity.

References

[1] Filippo Bonchi and Damien Pous. Checking nfa equivalence with bisimulations up to congruence. In *Proceedings of the 40th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '13, pages 457–468, New York, NY, USA, 2013. ACM.

[2] Julius Richard Büchi. On a decision method in restricted second order arithmetic. 1966.

[3] J.-M. Champarnaud and F. Coulon. Nfa reduction algorithms by means of regular inequalities. *Theoretical Computer Science*, 327(3):241 – 253, 2004. Developments in Language Theory.

[4] Kousha Etessami, Thomas Wilke, and Rebecca A. Schuller. Fair simulation relations, parity games, and state space reduction for büchi automata. In *Automata, Languages and Programming*, pages 694–707, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[5] Carsten Fritz and Thomas Wilke. Simulation relations for alternating büchi automata. *Theor. Comput. Sci.*, 338(1-3):275–314, June 2005.

[6] John Hopcroft. An n log n algorithm for minimizing states in a finite automaton. *An N Log N Algorithm for Minimizing States in A Finite Automaton*, page 15, 01 1971.

[7] Tao Jiang and B. Ravikumar. Minimal nfa problems are hard. In *Automata, Languages and Programming*, pages 629–640, Berlin, Heidelberg, 1991. Springer Berlin Heidelberg.

[8] Richard Mayr and Lorenzo Clemente. Advanced automata minimization. In *POPL 2013*, Oct 2012.

[9] Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. 1943. *Bulletin of mathematical biology*, 52 1-2:99–115; discussion 73–97, 1990.

[10] Edward F. Moore. Gedanken-experiments on sequential machines. In Claude Shannon and John McCarthy, editors, *Automata Studies*, pages 129–153. Princeton University Press, Princeton, NJ, 1956.

[11] A.W. Mostowski. Hierarchies of weak automata and weak monadic formulas. *Theoretical Computer Science*, 83(2):323 – 335, 1991.

[12] Michael O Rabin and Dana Scott. Finite automata and their decision problems. *IBM Journal of Research and Development*, 3:114–125, 04 1959.

[13] Sven Schewe. Beyond Hyper-Minimisation—Minimising DBAs and DPAs is NP-Complete. In Kamal Lodaya and Meena Mahajan, editors, *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2010)*, volume 8 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 400–411, Dagstuhl, Germany, 2010. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[14] Wolfgang Thomas. Handbook of theoretical computer science (vol. b). chapter Automata on Infinite Objects, pages 133–191. MIT Press, Cambridge, MA, USA, 1990.

[15] Wolfgang Thomas. Handbook of formal languages, vol. 3. chapter Languages, Automata, and Logic, pages 389–455. Springer-Verlag New York, Inc., New York, NY, USA, 1997.