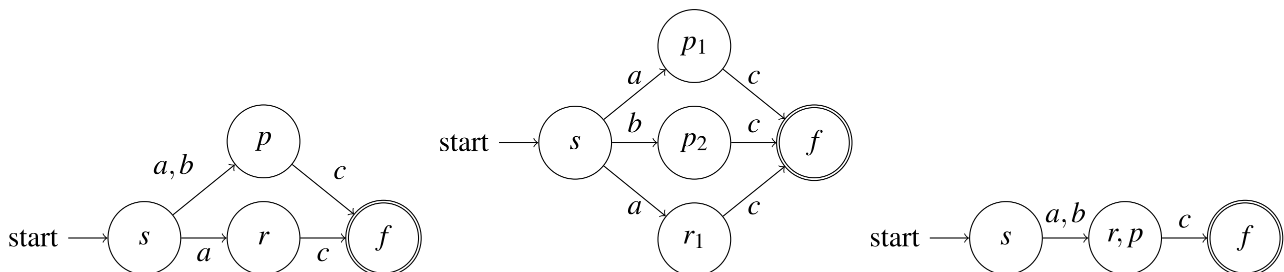
BRNO UNIVERSITY OF TECHNOLOGY FACULTY OF INFORMATION TECHNOLOGY
Excel@FIT 2021
http://excel.fit.vutbr.cz

SAT solver in nondeterministic automata minimization

Michal Šedý



Abstract
Nondeterministic finite automata (NFA) are widely used in computer science fields, such as regular languages in formal language theory, formal verification, high-speed network monitoring, image recognition, hardware modeling, or even in bioinformatic for the detection of the sequence of nucleotide acids in DNA. Automata minimization is a fundamental technique that helps to decrease resource claims (memory, time, or a number of hardware components) of implemented automata. Commonly used minimization techniques, such as state merging, transition pruning, and saturation, can leave potentially minimizable automaton subgraphs with duplicit language information. These fragments consist of a group of states, where the whole language of one state is piecewise covered by the other states in this group. The paper describes a new minimization approach, which uses SAT solver Z3, which provides information for efficient minimization of these so far nonminimizable automaton parts. Moreover, the newly investigated method, which only uses solver information and state merging, can minimize automata similarly and with a transition density up to 2.5 (from each state lead approximately 2.5 transition edges) faster than a tool RABIT, which uses state merging and transition pruning.

Keywords: Nondeterministic finite automata — Minimization — State merging — SAT solver

*xsedym02@stud.fit.vutbr.cz, *Faculty of Information Technology, Brno University of Technology*

1. Introduction

Nondeterministic finite automata (NFA) were investigated by Michael Rabin and Dana Scott [1]. In comparison with deterministic finite automata (DFA), NFA can make more than one transition after receiving the letter. This feature allows NFA to represent the language with fewer states and transitions than its deterministic variant. However, there are two sides to every story, the NFA is much harder to minimize. Nondeterministic finite automata are often used for a representation of regular languages, in data validation, web searching engines, pattern recognition, in network traffic monitoring, even genetic (matching of the sequence of nucleotide acids on DNA) [2], and so on.

An example of NFA usage is a representation of the regular expression for pattern matching in network traffic. Due to an increasing amount of data transmitted over the network and so increasing speed, it is necessary to improve the data scanning speed. Stan-

dard software solutions that can detect data fragments can not be used in high-speed networks. For the speed over 100 Gbps, it is required to implement a hardware analyzer [3]. To save space, resources, and cost of manufactured components, it is advisable to minimize the original automaton. Another big usage of automata minimization is formal verification, which uses language inclusion testing. The smaller the automaton is, the faster calculation is.

Nowadays, many efficient minimization techniques exist. The oldest method, state merging [4], investigated by Lucian Ille and Cheng Yu, merges two language equivalent states. Other successful procedures in the field of minimization are transition pruning and saturation published by Lorenzo Clemente and Richard Mayr in [5]. They said that the transition can be pruned if the better transition already exists (a state with stronger or equal language exists). On the contrary, saturation adds new already existing transitions. Despite the high efficiency of these methods, they are not omnipotent. They can still leave potentially minimizable automata subgraphs. These fragments consist of a set of states, where the whole language of some state is piecewise covered by other states. Each state can have a unique language, so language inclusion, which is necessary for minimization, does not exist.

The paper describes a method for minimizing this so far unsolvable automata subgraphs. The method works with sets of states with common successors or ancestors. All states from a set are substituted by states with the maximal one incoming and outcoming transition, then SAT solver Z3[1] is used in merging for maximizing the number of merged states. In comparison to an existing tool RABIT[2], which uses state merging and transition pruning, the solver gives a strong and on automata with a transition density up to 2.5 faster approximation of RABIT results.

2. Theoretical background

This chapter is dedicated to the theoretical background of nondeterministic finite automata. First, the NFA will be defined and then the related terms such as configuration, transition (taken from [6]), and language of the automaton and of the state.

2.1 Nondeterministic finite automaton

A *nondeterministic finite automaton* is a 5-tuple $M = (Q, \Sigma, \delta, I, F)$, where:

- Q is a finite set of states,
- Σ is an alphabet,
- $\delta \subseteq Q \times \Sigma \times Q$ is a transition relation,
- $I \subseteq Q$ is a finite set of initial states, and
- $F \subseteq Q$ is a finite set of final states.

The transition relation δ specifies a set of transition rules R. For the rule $r : q \in \delta(p, a)$, where $q, p \in Q$ and $a \in \Sigma$ we will use the notation $pa \to q$.

The automaton $M = (Q, \Sigma, \delta, I, F)$ will be used for the future definitions and examples.

2.2 Configuration

The *configuration* of NFA is a string $\chi \in Q\Sigma^*$. The automaton configuration displays an information about the current state and the remaining string at the input. For example, if the automaton M is in the state q and the string ab remains at the input, then the configuration is qab.

2.3 Transition

Let paw and qw be two configurations over an automaton M, where p and $q \in Q$, $a \in \Sigma$, and $w \in \Sigma^*$. Let $r : pa \to q \in R$. Then M can make a *transition* from qaw to qw, written as $paw \vdash qw$. If an automaton makes n transitions from χ to χ_n for some $n \geq 1$, then we write $\chi \vdash^+ \chi_n$. If $n \geq 0$, then we write $\chi \vdash^* \chi_n$.

2.4 Languages

The *accepting language* of an automaton M is $L(M) = \{w \mid w \in \Sigma^*, q_0 w \vdash^* f, q_0 \in I, f \in F\}$. It is a set of strings accepted by an automaton.

The *backward language of a state* consists of strings over the automaton alphabet, for which exists a sequence of transitions (route) from an initial state to an examined state. The backward language of a state q is $\overleftarrow{L}(q) = \{w_l \mid w_l \in \Sigma^*, q_0 w_l \vdash^* q,\ q_0 \in I\}$.

The *forward language of a state* is a set of strings, for which exists a sequence of transitions from the actual state to the final state. The forward language of a state $q \in Q$ is defined as $\overrightarrow{L}(q) = \{w_r \mid w_r \in \Sigma^*, qw_r \vdash^* f, f \in F\}$.

3. Existing minimization techniques

This chapter contains a description of all methods (state merging, transition pruning, and saturation) used by a tool RABIT, with which the investigated method using the solver is compared. The proofs of transition pruning are shown on Büchi word automata (NBA) in [5, p.16–20].

Calculation of a language inclusion used in this chapter is often approximated by a faster calculation of a simulation relation [7]. A *simulation* on automaton

[1] Z3 solver is available at https://github.com/Z3Prover/z3.

[2] RABIT is available at http://languageinclusion.org/doku.php?id=tools.

M is a relation $\preceq \subseteq Q \times Q$ such that $p \preceq r$ only if $p \in F \implies r \in F$ and for every transition $pa \to p'$, there exists a transiton $ra \to r'$, such that $p' \preceq r'$.

3.1 State merging

The most well-known minimization approach for nondeterministic finite automata is state merging. Two states p and $q \in Q$ can be merged only if at least one of the following conditions [8] is met:

- $\overleftarrow{L}(p) \subseteq \overleftarrow{L}(q) \wedge \overleftarrow{L}(q) \subseteq \overleftarrow{L}(p)$,
- $\overrightarrow{L}(p) \subseteq \overrightarrow{L}(q) \wedge \overrightarrow{L}(q) \subseteq \overrightarrow{L}(p)$, or
- $\overleftarrow{L}(p) \subseteq \overleftarrow{L}(q) \wedge \overrightarrow{L}(p) \subseteq \overrightarrow{L}(q)$.

3.2 Transition pruning

The basic idea of transition pruning is the existence of a better transition (stronger language), which can overtake the function of a deleting transition. A transition $ra \to p$ can be pruned if one of the following conditions is met:

- $\exists ra \to q \wedge \overrightarrow{L}(p) \subseteq \overrightarrow{L}(q)$
- $\exists qa \to p \wedge \overleftarrow{L}(r) \subseteq \overleftarrow{L}(q)$, or
- $\exists r'a \to p' \wedge \overleftarrow{L}(r) \subseteq \overleftarrow{L}(r') \wedge \overrightarrow{L}(p) \subseteq \overrightarrow{L}(p')$.

3.3 Saturation

The saturation adds a new transition to the automaton without changing language. It is an analogy of transition pruning. The transition $pa \to r$ can be added into an automaton only if one of the following conditions is met:

- $\exists qa \to r \wedge \overleftarrow{L}(p) \subseteq \overleftarrow{L}(q)$ or
- $\exists pa \to q \wedge \overrightarrow{L}(r) \subseteq \overrightarrow{L}(q)$.

4. Minimization using SAT solver Z3

This chapter describes the main approaches used in the minimization of nondeterministic finite automata by SAT solver Z3. The algorithm minimizes an automaton by parts, each set of states with a common successor or ancestor (family) is multiplied, and then the optimal merge based on the information from SAT solver is processed. The whole minimization algorithm is shown at the end of the chapter.

4.1 Family of states

A family is a set of states with a common ancestor or common successor and a nondeterministic transition. Two states p and q belong to the same family if there exists a common ancestor $s \in Q$ of the states and a transition rules $sa \to p$ and $sa \to q$, where $a \in \Sigma$. The states p and q are also a family if there exists a common successor $r \in Q$ of these states and such a letter $b \in \Sigma$, for which exist the transition rules $pb \to r$ and $qb \to r$.

For getting bigger families, all sets of families with a common state are joined. The bigger the family is, the more optimal the solution the solver returns.

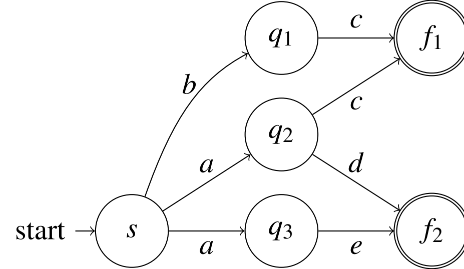


Figure 1. The figure shows two families of states with a nondeterministic transition. The first with a common ancestor s and a letter a consists of the states q_2 and q_3. The second with a common successor f_1 and a letter c consists of the states q_1 and q_2. After joining families with a common state we get a big family with states q_1, q_2, and q_3.

4.2 States multiplication

For getting out of a local automaton minimum, it is necessary to multiply each state of a family.

Let R be a set of transition rules, the state $s \in Q$ will be multiplied, $trans_{in}(s) = \{ra \,|\, ra \to s \in R,\ r \neq s,\ a \in \Sigma\}$ be a set of ancestors of s in a combination with a transition letter, $trans_{out}(s) = \{qb \,|\, sb \to q \in R,\ q \neq s,\ b \in \Sigma\}$ be a set of successors of s in a combination with a transition letter, and $\Sigma_{self}(s) = \{a \,|\, s \in \delta(s,a),\ a \in \Sigma\}$ be a self-loop alphabet of the states s. Then $\forall\ rasb \in trans_{in}(s) \times trans_{out}(s)$ transitions $ra \to s_i$ and $s_i b \to q$ and $\forall\ c \in \Sigma_{self}(s)$ transitions $s_i c \to s_i$, for $i = 0 \ldots n$, will be created. If the original state s is a final or initial state, then all new states s_i, for $i = 0 \ldots n$, will be initial or final too. The original state s is removed after multiplication. Multiplication example is show in the figure 2.

4.3 SAT Solver problem coding

Information obtained from a solver will help to merge the maximum of states. There could be a state p which is in backward language equivalency with a state q and in forward language equivalency with the state r, and so on. None of the states merged by backward language equivalency can be then merged based on forward language equivalency, and vice versa. The million dollar question is: Which group of states merge to get the most optimal solution? Let the solver decide.

The sets of backward and forward language equivalent states are the main information for coding a solver task. Let the set of backward equivalent states be

$B_{eq} = \{(q_1, q_2), (q_1, q_3)\}$ and the set of forward equivalent states be $F_{eq} = \{(q_1, q_4)\}$.

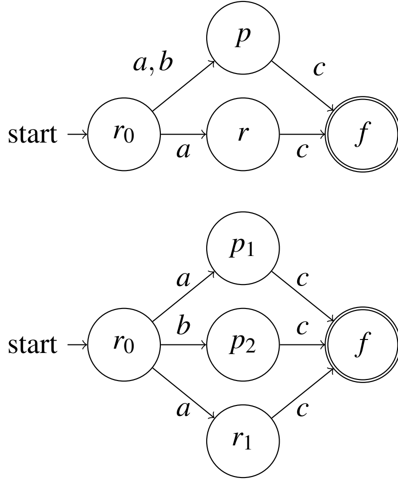


Figure 2. Multiplication of a state p to states p_1 and p_2 and state r to state r_1.

The solver variables will be the states used in backward equivalency (with prefix B) and the states used in forward equivalency (with a prefix F). Therefore, the variables are $\{Bq_1, Bq_2, Bq_3, Fq_1, Fq_4\}$.

For each state s used in the forward and backward language equivalency, the rule $Bs \implies \neg Fs$ will be created. That means that the state s can not be merged with some state in backward language equivalency and then with other state in forward language equivalency. In this example, the rule is $Bq_1 \implies \neg Fq_1$.

The information about mergeable pairs is coded by a logical *and* between states, because a pair of states can be merged only if both states allow this merge. Backward equivalent states (q_1, q_2) will be coded as $Bq_1 \wedge Bq_2$. All pairs coded in this way are interconnected by a logical *or*, because not all pairs need to be merged. Z3 solver is forced to maximize the number of these pairs evaluated as true [9].

After all, the coded problem will look like: $\big((Bq_1 \wedge Bq_2) \vee (Bq_1 \wedge Bq_3) \vee (Fq_1 \wedge Fq_4)\big) \wedge Bq_1 \implies \neg Fq_1$.

In this case, the solver will set as a true variables Bq_1, Bq_2, and Bq_3. Only pairs of backward or forward language equivalencies, which have both states truly evaluated by the solver result, will be merged. This means that only the pairs $\{(q_1, q_2), (q_1, q_3)\}$ of backward equivalence will be merged. After the joining of all pairs with a common state, it can be seen that all states q_1, q_2, and q_3 will be merged into one.

4.4 Approximation of language quivalence

The language equivalency calculation in a minimization using SAT solver is very hard because the minimization algorithm multiplies a family to many states (even hundreds of states). The calculation of the simulation relation is slow too. The multiplication is done many times in a minimization process and families are getting bigger and bigger. It is necessary to calculate only a strong approximation of language equivalencies (some equivalency might not be detected). The approximation approach has a specified distance, on which the language equivalency of two states must be confirmed, otherwise the states are not equivalent. The state equivalence checking algorithm with defined distance is an adaptation of an automata equivalence checking algorithm [10].

The algorithm for approximation of a forward language equivalence of states p and q at a distance 10 is shown below. The algorithm returns $True$ if the states are forward equivalent, otherwise $False$. The distance 10 is used in the experiments, but can be increased for better approximation, or decreased for faster approximation.

```
maxDistance = 10
visited = set()
closed = set()
todo = set((q,r))
distanceCnt = 0
while todo:
   (X,Y) = todo.pop()
   if (X,Y) not in closed:
     visited += X.union(Y)
     if (X,Y) is bad pair:
       return False
     for a in Σ:
       if distanceCnt >= maxDistance:
         if δ(X,a).union(δ(Y,A))
                   .difference(visited):
           return False
       todo += (δ(X,a),δ(Y,a))
     closed += (X,Y)
     distanceCnt += 1
return True
```

The pair (X, Y) is bad if one of the sets is empty, but the other is not, or if the state of X or Y is final whereas the other is not. The approximation of a backward language equivalence is defined similarly.

4.5 Minimization algorithm using SAT solver

With early defined methods for the minimization subproblems, the main algorithm can be described. Let minimize NFA automaton $M = (Q, \Sigma, \delta, I, F)$.

```
closedFamilies = set()
while True:
  families = getFamilies(M)
  families -= closedFamilies
  if families.empty():
    break
  for family in families:
    family.multiplyStates()
    while family.existEquivalentPair():
      family.minimizeBySolver()
    closedFamilies += family
```

5. Experiment results

The efficiency of the investigated approach is compared with a tool RABIT. The tests have been performed on 3730 automata, from an abstract regular model checking [11], with a total of 63538 states. Automata were modified to have one initial and maximal two final states (one can be final as well as initial). The size of each automaton is up to 400 states. The average transition density of automata is 1.3 (from each state leads approximately 1.3 transitions). The bigger the transition density is, the slower the solver minimization is. First, the solver is compared with RABIT, which uses state merging and transition pruning. Then the solver is used as a supplement of RABIT after running a merge, transition pruning, and saturation, which is the best-known combination. RABIT uses lookahead simulation for an approximation of language relations. Lookahead is set to 1 for all experiments. Bigger lookahead did not give better minimization results on the tested automata, only slower down the RABIT.

5.1 Solver vs RABIT

The new minimization approach using solver information for better state merging minimizes the input automata to a total of 59421 states. The tool RABIT using state merging and transition pruning minimizes the input automata to a total of 58734 states. This means that the minimization using the solver approximated the RABIT result with an accuracy of 98.84%. The comparison of the results of minimization methods is in the figure 3.

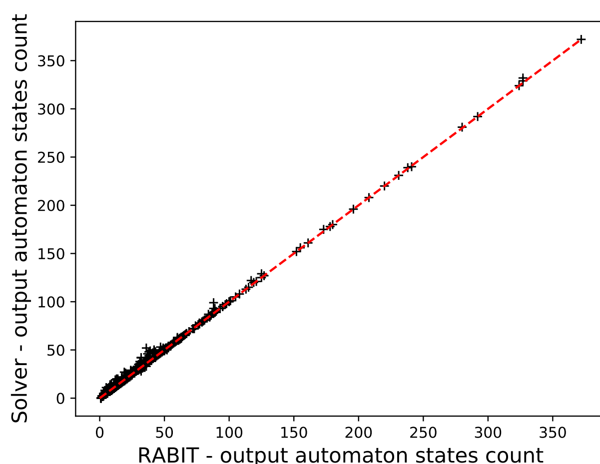


Figure 3. The graph compares output automata size (states count) of a solver based approach and a tool RABIT, which uses state merging and transition pruning. It can be seen, that solver strongly (on 98.84%) approximates RABIT solutions.

Another engaging part of the comparison is the minimization duration. The RABIT, using state merging and transition pruning minimizes 3730 automata in 317.830 s. On the contrary, the solver did approximately the same minimization in only 42.340 s. That is 13.32% of the time consumed by RABIT. The solver minimization is solver on automata with high transition density. The comparison is in the figure 4.

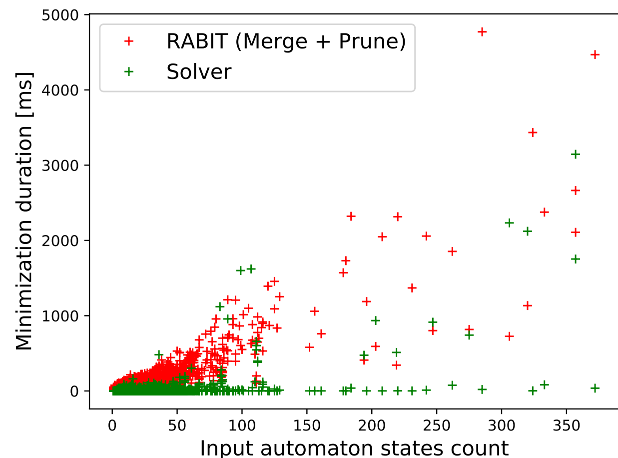


Figure 4. The comperison of time consumed by a tool RABIT and solver minimization approach from a figure 3. Solver is much faster (7.5 times) than RABIT.

5.2 Solver as RABIT's supplement

The solver with merge can be used alone or as a supplement of a RABIT. The solver with merge is processed over a RABIT best minimization. The new reduced automaton is a little more minimal than the RABIT result. The difference between the minimized states (the difference between a number of states of the input and output automaton) of RABIT itself and its version enriched by the solver is the main measurement. The total input number of minimized states is the same as in the previous example (63538). RABIT itself minimizes the automata by 5322 states. The extended version of the algorithm using RABIT and solver minimizes the automata by 5356 states. That is, about 0.63% minimized states more than only by using RABIT, which uses the strongest minimization algorithms known to mankind.

6. Conclusions

The minimization of nondeterministic finite automata is a fundamental problem in computer science. The main usages of NFAs are regular expression representation and formal verification. A formal verification works widely with language inclusion testing. The

smaller an automaton is, the faster inclusion is calculated.

The paper described new methods for finding a potentially minimizable subgraph of an automaton, multiplication of the states from these fragments for getting from a local automaton minimum, and for using information from SAT solver Z3 for a more optimal state (re)merging. The solver approach strongly (with 98.84% accuracy) and 7.5 times faster approximates the minimization done by RABIT, using merge and transition pruning. Moreover, the solver information can improve the already effective result of RABIT minimization, using merge, transition pruning, and saturation, by 0.63%.

7. Future work

The algorithm of an automaton minimization based on solver information, could be done faster with the usage of better solver problem coding.

As has been already mentioned, the minimization using a solver works slower for automata with a high transition density, such as 3 and more. The method could be improved to work better for automata with dense transitions.

Due to a high approximation of RABIT merge and pruning results, the solver minimization could replace the merging and transition pruning phase in standard minimization algorithms.

Acknowledgements

I would like to thank my supervisor Mgr. Lukáš Holík, Ph.D. for his help.

References

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

[2] Qurat Ain, Yousaf Saeed, Shahid Naseem, Fahad Ahamd, Tahir Alyas, and Nadia Tabassum. Dna pattern analysis using finite automata. *International Research Journal of Computer Science (IRJCS)*, 1:1–4, 10 2014.

[3] HyunJin Kim and Kang-Il Choi. A pipelined non-deterministic finite automaton-based string matching scheme using merged state transitions in an fpga. *PLOS ONE*, 11, 10 2016.

[4] Lucian Ilie and Sheng Yu. Algorithms for computing small nfas. In *Proceedings of the 27th International Symposium on Mathematical Foundations of Computer Science*, MFCS '02, page 328–340, Berlin, Heidelberg, 2002. Springer-Verlag.

[5] Lorenzo Clemente and Richard Mayr. Efficient reduction of nondeterministic automata with application to language inclusion testing. *CoRR*, abs/1711.09946, 2017.

[6] Alexander Meduna and Roman Lukáš. Models for regular languages. lecture notes for Formal Languages and Compilers, 2017.

[7] A. Parosh Abdulla, Lukáš Holík, Yu-Fang Chen, Richard Mayr, and Tomáš Vojnar. When simulation meets antichains (on checking language inclusion of nondeterministic finite (tree) automata). Technical report, 2010.

[8] Lucian Ilie, Gonzalo Navarro, and Sheng Yu. *On NFA Reductions*, pages 112–124. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.

[9] Nikolaj Bjørner and Phan Anh Dung. vz - maximal satisfaction with z3. In *Proceedings of the 6th International Symposium on Symbolic Computation in Software Science (SCSS 2014)*, 2015.

[10] Chen Fu, Yuxin Deng, David Jansen, and Lijun Zhang. On equivalence checking of nondeterministic finite automata. In Kim Guldstrand Larsen, Oleg Sokolsky, and Ji Wang, editors, *Dependable Software Engineering. Theories, Tools, and Applications*, pages 216–231, Cham, 2017. Springer International Publishing.

[11] Ahmed Bouajjani, Peter Habermehl, Lukáš Holík, Tayssir Touili, and Tomáš Vojnar. Antichain-based universality and inclusion testing over nondeterministic finite tree automata. In Oscar H. Ibarra and Bala Ravikumar, editors, *Implementation and Applications of Automata*, pages 57–67, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.