Visualização e Extensão de um Verificador de Modelos para a ferramenta Verigraph-GUI *

Arthur L. Fuchs, Rodrigo Machado, Leila Ribeiro

[1]Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
Caixa Postal 15.064 – 91.501-970 – Porto Alegre – RS – Brazil

alfuchs@inf.ufrgs.br, rma@inf.ufrgs.br, leila@inf.ufrgs.br

Abstract. *Verigraph-GUI is a graphical editor for the Verigraph tool, a graph transformation tool. This work reports on the construction of a graphical user interface for the model checking module of the Verigraph tool, as well as further planned extensions and further improvements.*

Resumo. *Verigraph-GUI é um editor gráfico desenvolvido para a ferramenta Verigraph, uma ferramenta de transformação de grafos. Esse trabalho relata sobre a construção de uma GUI para o módulo de verificação de modelos da ferramenta Verigraph, assim como extensões e outras melhorias planejadas.*

1. Introdução

O Verigraph-GUI [1] [Fuchs et al. 2019] é uma interface gráfica para a ferramenta de transformação de grafos Verigraph [2] [Azzi et al. 2018]. Essa interface inclui um editor de gramáticas de grafos e oferece acesso à execução e análise de especificações. Uma dessas funcionalidades é a verificação de modelos – técnica que permite verificar se um sistema satisfaz propriedades expressas por fórmulas de uma Lógica Temporal [Clarke 2008]. Tal módulo, contudo, estava até então restrito a uma interface de linha de comando. Este trabalho trata da construção de uma interface gráfica para a verificação de modelos no Verigraph-GUI e de melhorias planejadas para essa funcionalidade.

2. Gramáticas de Grafos e a ferramenta Verigraph-GUI

Gramáticas de grafos permitem especificar sistemas utilizando grafos para representar o estado do sistema, e regras de reescrita para descrever o comportamento do mesmo. Considerando a abordagem algébrica [Ehrig et al. 2006], no estilo *Double Pushout* (DPO), uma regra de grafos é representada por um span de (homo)morfismos totais de grafos $r : L \leftarrow K \rightarrow R$. Um morfismo de grafos é o mapeamento de vértices e arcos de um grafo G para vértices e arcos de um grafo H de forma a preservar os nodos de origem e o destino dos arcos. Nessa forma de representar regras, K serve de interface entre os grafos L e R e contém os elementos *preservados*. Os elementos de L e de R não inclusos nos morfismos K são, respectivamente, *deletados* e *criados* pela regra. Uma regra pode ser aplicada se for encontrado um *match* de L no grafo de estado G que queremos transformar. Essa ocorrência é modelada por um morfismo total de grafos, que faz o mapeamento de todos os elementos de L para G. A aplicação de uma regra é feita formalmente através

*Trabalho em andamento

[1]https://github.com/artFuchs/verigraph-GUI

[2]https://github.com/Verites/verigraph

de um diagrama de pushout duplo DPO. É comum o uso de mecanismos de tipagem de grafos para especificar diferentes categorias de nodos e arestas, e restringir as possibilidades de conexão de arestas nos grafos instância.

O Verigraph-GUI nasceu da necessidade de um editor de gramáticas de grafos para a ferramenta Verigraph, bem como da necessidade de uma interface para visualização de suas análises. Sua versão inicial foi apresentada em [Fuchs et al. 2019]. Desde então o editor de gramáticas de grafos foi estendido ter suporte à NACs e, recentemente, o Verigraph-GUI ofereceu suporte a módulos de análise já presentes no Verigraph. Até o momento, foram escritas mais de 7000 linhas no desenvolvimento do Verigraph-GUI.

A verificação de modelos foi implementada originalmente em [Becker 2014], em uma versão inicial do sistema Verigraph, apresentando somente uma interface de linha de comando. Este projeto tem como objetivo geral:

1. a construção de uma interface gráfica para o módulo de verificação de modelos CTL, com a possibilidade de geração e navegação pelo espaço de estados (incluindo questões relacionadas ao leiaute do espaço de estados);
2. a extensão deste mesmo para suportar outras Lógicas Temporais (LTL e CTL*);
3. a melhoria da performance deste verificador de modelos.

Este trabalho se encontra em andamento: neste artigo revisamos a verificação de modelos para gramáticas de grafos e relatamos resultados iniciais sobre o projeto acima, em particular a conclusão do primeiro objetivo. A versão atual do Verigraph-GUI (mostrada nas Figuras 1 e 2) já permite realizar a simulação da aplicação de regras no sistema, a análise de pares críticos, a visualização e manipulação do espaço de estados e a verificação de fórmulas CTL sobre esse espaço de estados.

3. Espaço de estados de uma gramática de grafos

A Figura 1 mostra uma gramática de grafos que representa um sistema de trens, originalmente descrito em [Becker 2014]. Os tipos de nodos representam trens, passageiros e estações. As arestas com rótulo in representam a localização de passageiros a trens e estações, assim como a posição de trens em estações. As arestas com rótulo destiny apontam a estação de destino do passageiro. A gramática consiste de três regras: a regra *moveTrain* simula a movimentação do trem entre estações; *embarkPassenger* simula um passageiro em uma estação entrando no trem; *disembarkPassenger* simula um passageiro descendo do trem para sua estação de destino. O grafo inicial descreve uma situação com três estações conectadas de forma circular, um único trem e um passageiro. No sistema Verigraph-GUI as regras são descritas através de um único grafo com marcações específicas para os elementos deletados (marcados em azul) e criados (marcados em verde) pela regra, de forma semelhante à edição de gramática no sistema Groove [Rensink 2004].

Gramáticas de grafos têm como características serem modelos de execução baseada em regras de reescritas, normalmente levando a não-determinismo e concorrência na sua execução. Essa característica pode tornar difícil a compreensão sobre os comportamentos possíveis do sistema a partir da descrição da gramática.

O espaço de estados é um sistema de transições, isto é, um grafo que descreve todos os estados do sistema gerados pela aplicação sucessiva das regras de reescrita. Assim como no sistema Groove, o Verigraph-GUI considera como as propriedades de um

dado estado as regras que possuem *matches* naquele estado em particulador. Transições são anotadas somente com o nome da regra utilizada, mesmo que haja mais de um *match* possível para tal regra. Dado que a abordagem algébrica para gramáticas de grafos apresenta não-determinismo de representação, ao calcular o espaço de estados nós identificamos grafos isomorfos. Desta forma, definimos cada nodo do espaço de estados como sendo uma classe de equivalência de grafos isomorfos, ao invés de grafos concretos. Para exemplificar, a Figura 2 mostra o espaço de estados gerado para a gramática da Figura 1. O estado 0, marcado em azul na figura, corresponde à classe de equivalência que inclui o grafo inicial da gramática.

Nota-se que gramáticas de grafos podem gerar espaços de estados infinitos, e, mesmo quando são finitos, o tamanho destes pode apresentar um número muito grande de estados, fato normalmente conhecido como *explosão do espaço de estados*.

4. Verificação de Modelos CTL

A ideia geral da técnica de verificação de modelos é: dado um modelo (sistema de transições) M e uma fórmula em Lógica Temporal *f*, achar todos os estados de M que satisfazem *f* [Clarke 2008]. A execução dessa técnica é automática e permite verificar se um sistema satisfaz a propriedade especificada pela fórmula. Ela foi desenvolvida para procurar por erros de concorrência em programas paralelos, erros tipicamente difíceis de reproduzir e encontrar em testes. Verificação de modelos pode ser usada para analisar o espaço de estados gerado por gramáticas de grafos, que como mencionado na seção anterior podem apresentar um número muito grande de estados ou até mesmo ser infinitos.

Lógicas Temporais permitem fazer afirmações sobre o tempo. Um dos tipos de Lógica Temporal mais conhecidos é a *Computer Tree Logic* (CTL). Essa lógica foi criada com o objetivo de sintetizar esqueletos de sincronização [Clarke 2008] e suas expressões permitem descrever propriedades com base nas possíveis ramificações e rotas do espaço de estados do sistema. Uma formula em CTL é definida pela seguinte sintaxe:

$$\varphi ::= \bot \mid \top \mid p \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \varphi \Rightarrow \varphi \mid \varphi \iff \varphi \mid (\varphi) \mid$$

$$AX\varphi \mid AF\varphi \mid AG[\varphi U \varphi] \mid EX\varphi \mid EF\varphi \mid EG\varphi \mid E[\varphi_1 U \varphi_2]$$

onde p é uma formula atômica. No caso de espaços de estados de gramáticas de grafos, as fórmulas atômicas consistem dos nomes de regras passíveis de serem aplicadas sobre o estado em questão. Os operadores $\bot, \top, \neg, \vee, \wedge, \rightarrow$ e $\iff$ funcionam da mesma forma que em lógica proposicional clássica. Os operadores A e E são quantificadores e expressam *todos os caminhos satisfazem* e *existe algum caminho que satisfaça*, respectivamente. O operadores X, F, G e U são *operadores temporais* : X verifica se uma propriedade é satisfeita no próximo estado; F verifica se uma propriedade é satisfeita eventualmente em um caminho; G verifica se uma propriedade é satisfeita por todos os estados em um caminho; U verifica se, em um caminho, todos os estados satisfazem φ_1 até que a fórmula φ_2 seja satisfeita.

Na verificação de modelos, cada estado de um modelo é analisado para ver se ele satisfaz a fórmula CTL especificada pelo usuário. Como exemplo, a Figura 2, mostra o resultado para a fórmula $A[moveTrain\, U\, embarkPassenger]$. Essa formula especifica a propriedade de que o trem deve sempre poder se mover entre estações até que seja possível

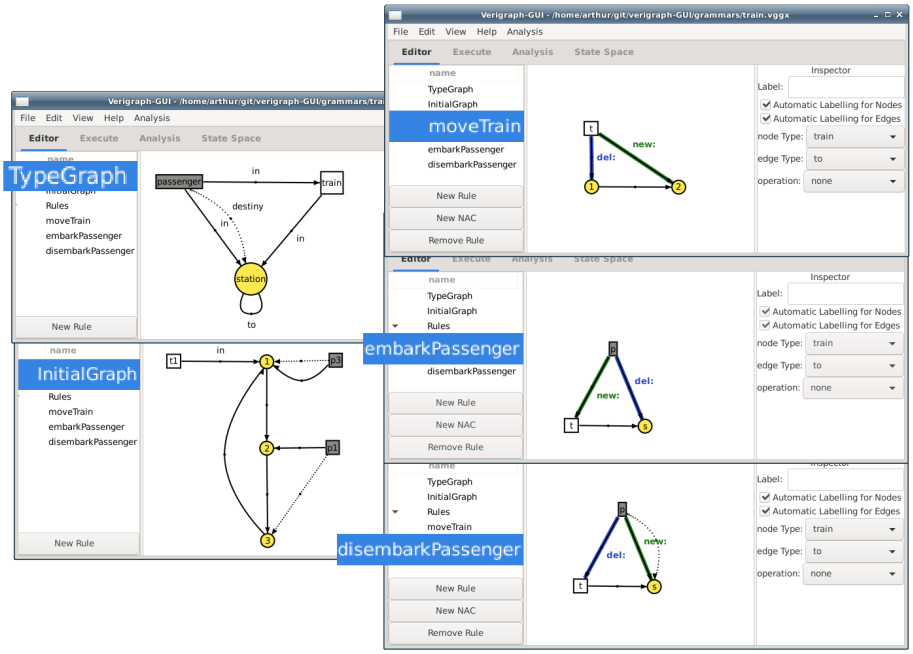

Figura 1. Gramática de exemplo especificada no Verigraph-GUI

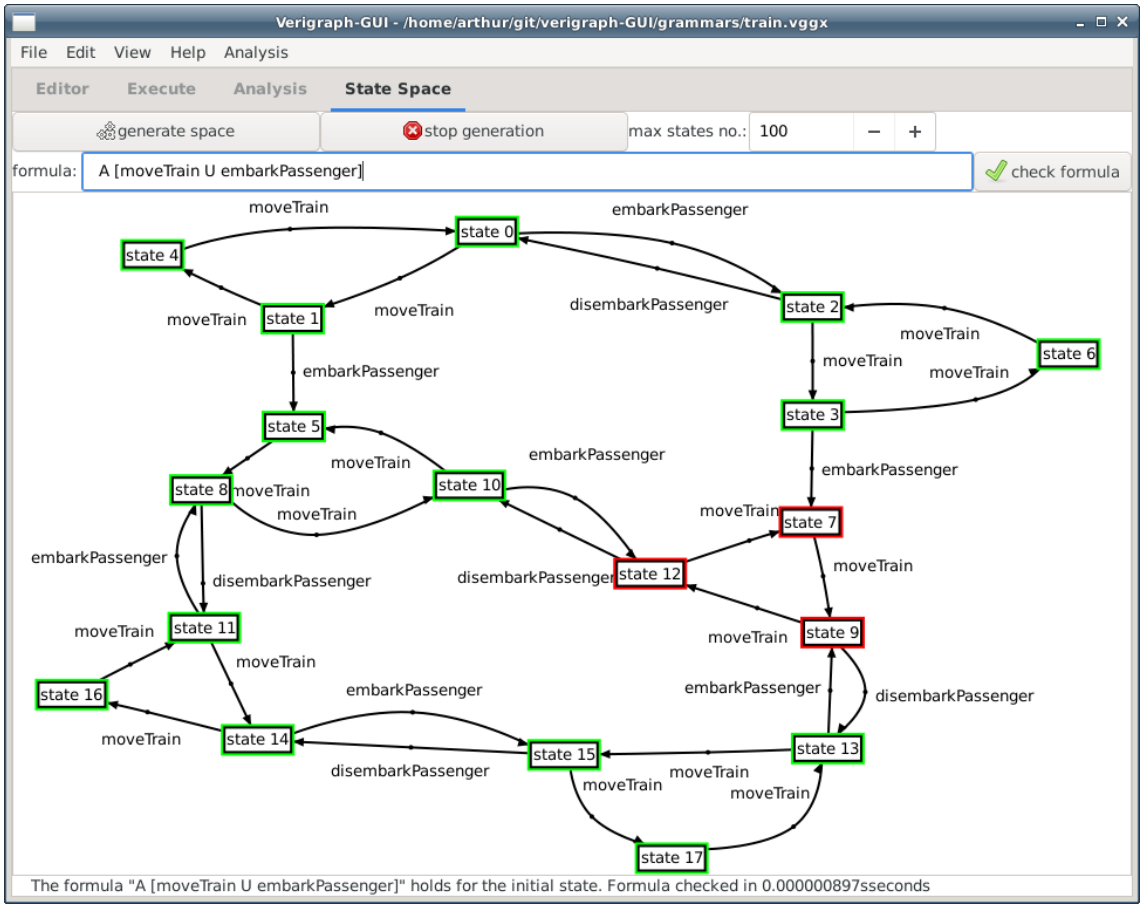

Figura 2. Espaço de estados gerado pela gramática de exemplo e resultado da verificação da fórmula CTL $A[moveTrain\ U\ embarkPassenger]$

que um passageiro embarque. Note que embora alguns estados não satisfaçam a fórmula, o estado 0 (estado inicial) o faz, sendo esta a verificação realizada pela ferramenta.

O algoritmo de satisfação de fórmulas CTL usado no Verigraph-GUI é descrito em [Huth and Ryan 2004], sendo utilizada a implementação feita em [Becker 2014]. O algoritmo funciona verificando quais estados satisfazem cada sub-fórmula de uma fórmula CTL ϕ. A avaliação das sub-fórmulas é feita em ordem crescente em relação aos seus tamanhos, deixando seus resultados disponíveis para serem usados nas sub-fórmulas maiores. Assim, por exemplo, a fórmula $A[moveTrain\ U\ embarkPassenger]$ tem o conjunto de sub-fórmulas $\{A[moveTrain\,U\,embarkPassenger], moveTrain, embarkPassenger\}$, e a execução do algoritmo encontra todos os estados que satisfazem $moveTrain$ e $embarkPassenger$ antes de verificar quais estados satisfazem $A[moveTrain\,U\,embarkPassenger]$.

5. Status do projeto e conclusões

Nesta seção, é relatado o estado atual do trabalho e os próximos passos a serem dados neste projeto.

O módulo de verificação de modelos havia sido implementado no Verigraph com uma interface textual [Becker 2014]. Neste trabalho foi desenvolvida uma GUI para este módulo no Verigraph-GUI, que pode ser vista na figura 2. Essa GUI, assim como as GUIs de outros módulos do Verigraph-GUI, foi projetada utilizando a ferramenta Glade, que permite a construção de GUIs de forma visual. Ela permite gerar e visualizar o espaço de estados da gramática, e também especificar uma formula CTL e visualizar os estados que a satisfazem. Sobre o que foi feito no desenvolvimento dessa GUI:

- O algoritmo de geração do espaço de estados implementado no trabalho de [Becker 2014] foi modificado para:
  - Fazer a utilização de busca em largura ao invés de busca em profundidade;
  - Limitar a busca a um número máximo de estados (a busca anteriormente era limitada à um nível máximo de profundidade)
  - Rodar em paralelo com a geração da visualização do espaço de estados. Isso permite que o usuário pare a geração do espaço de estados mantendo o modelo gerado até o momento.
- O algoritmo de satisfação implementado em [Becker 2014] foi integrado, permitindo que uma fórmula CTL especificada pelo usuário seja verificada caso o espaço de estados tenha sido gerado.
- Para visualização do espaço de estados,
  - É usado um algoritmo de leiaute que organiza os nodos do espaço de estado em níveis, gerando uma estrutura de diagrama. Os níveis são organizados de forma que o primeiro nível contenha apenas o estado inicial, e os outros níveis contenham os estados que possam ser alcançados apenas a partir de um estado do nível anterior. O leiaute gerado por esse algoritmo para a gramática da Figura 1 pode ser visto na Figura 3.
  - Essa estrutura é desenhada utilizando os módulos de renderização e de definição de *callbacks* para o *canvas* (área principal da janela), os quais são também utilizado nos módulos de edição e simulação. O resultado do

algoritmo de satisfação da formula CTL é usado para definir cores de destaque para os nodos do grafo: verde se a fórmula é satisfeita, e vermelho caso contrário.

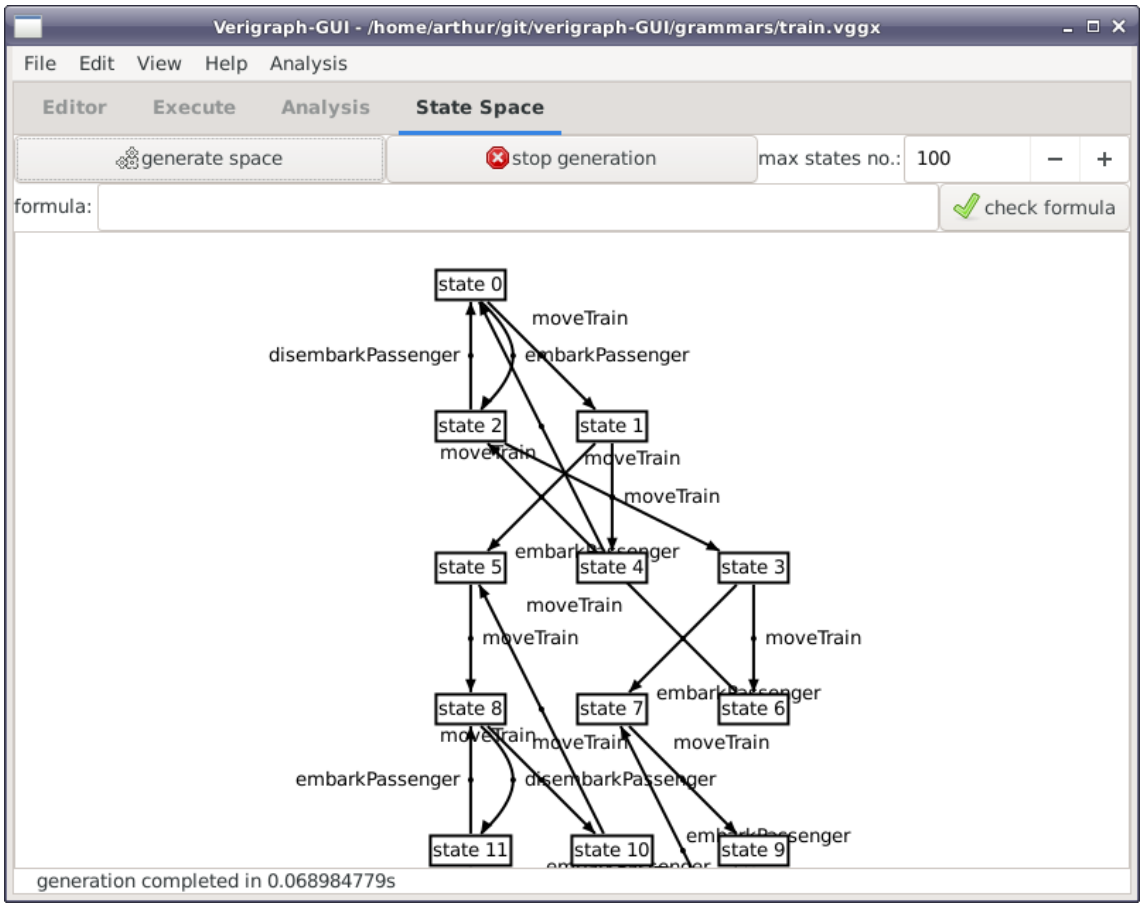

Figura 3. Leiaute gerado durante a construção do espaço de estados. O leiaute mostrado na Figura 2 é o resultado da organização deste. Nesta figura, os nodos não têm sombreamento porque a verificação não foi executada (foi somente executada a geração do espaço de estados).

Atualmente está em progresso a avaliação da performance do algoritmo de verificação de modelos através de *benchmarks*. Um fator importante a ser considerado para a performance do verificador de modelos é a geração do espaços de estados. Por conta de cada estado ser considerado uma classe de equivalência de grafos, a verificação de isomorfismo de grafos atualmente realizada é custosa e pretendemos propor melhorias nesse sentido. Pretendemos também:

- Melhorar a visualização do espaço de estados, pois atualmente não é possível visualizar qual classe de equivalência de grafos isomorfos um estado representa. Uma possibilidade é integrar o espaço de estados com o módulo de simulação, de forma que quando um estado seja selecionado, esse estado se torne o estado atual do simulador (similar ao sistema Groove [Rensink 2004]).
- Estender o verificador de modelos para aceitar outras lógica temporais, em particular LTL e CTL* [Huth and Ryan 2004]. Essas lógicas permitem expressar propriedades diferentes:

- Diferentemente de CTL, LTL não permite verificar pela existência de um caminho. LTL permite expressar propriedades sobre os caminhos de forma universal. Isso porque LTL não faz uso de quantificadores (A, E). No entanto, LTL permite selecionar um subconjunto de caminhos descrevendo-os com uma fórmula, algo que não é possível de fazer em CTL pois os operadores temporais em CTL são sempre associados a quantificadores.
- CTL* é uma lógica temporal que combina as expressividades de CTL e LTL. Isso é feito removendo as restrições de que todo operador temporal deve ser associado a um quantificador. O lado negativo da lógica CTL* é que sua verificação é mais custosa computacionalmente.

Referências

Azzi, G. G., Bezerra, J. S., Ribeiro, L., Costa, A., Rodrigues, L. M., and Machado, R. (2018). *The Verigraph System for Graph Transformation*, pages 160–178. Springer International Publishing, Cham.

Becker, T. R. (2014). VeriGraph: A Tool For Model Checking Graph Grammars. Monografia (Bacharel em Ciência da Computação), UFRGS (Universidade Federal do Rio Grande do Sul), Rio Grande do Sul, Brazil.

Clarke, E. M. (2008). *25 Years of Model Checking: History, Achievements, Perspectives*. Springer Berlin Heidelberg, Berlin, Heidelberg.

Ehrig, H., Ehrig, K., Prange, U., and Taentzer, G. (2006). *Fundamentals of Algebraic Graph Transformation (Monographs in Theoretical Computer Science. An EATCS Series)*. Springer-Verlag, Berlin, Heidelberg.

Fuchs, A. L., Machado, R., and Ribeiro, L. (2019). Verigraph-gui: A gui for the verigraph tool. In *Anais WEIT 2019*, pages 268–277. Universidade de Passo Fundo (UPF).

Huth, M. and Ryan, M. (2004). *Logic in computer science - modelling and reasoning about systems (2. ed.)*.

Rensink, A. (2004). The groove simulator: A tool for state space generation. In Pfaltz, J. L., Nagl, M., and Böhlen, B., editors, *Applications of Graph Transformations with Industrial Relevance*, pages 479–485, Berlin, Heidelberg. Springer Berlin Heidelberg.