Estudo de Técnicas Usadas em Verificação Formal de Modelos para Prevenção do Problema "Explosão de Estados"

Ana Silvia M. S. Amaral
INPE/Lac
anasil@lac.inpe.br

Eliane Martins
Unicamp/IC
eliane@ic.unicamp.br

N. L. Vijaykumar
INPE/Lac
vijay@lac.inpe.br

Resumo

Com o atual avanço tecnológico, aplicações das mais diversas áreas têm exigido desenvolvimento de sistemas de software cada vez mais complexos. Além disso há na indústria uma crescente demanda por metodologias que aumentem a confiança no projeto e construção de sistemas corretos. A estratégia tradicional de engenharia na construção de sistemas complexos é através do uso de modelos. Modelos são mais simples do que o sistema representado por eles, e auxiliam a entender e prever o comportamento do sistema.

Na verificação formal também se faz uso da criação de um modelo matemático do sistema. Então, as propriedades desejadas do sistema são especificadas numa forma concisa e sem ambigüidades através de uma linguagem e, por fim, através de um método de prova é verificado se o modelo satisfaz essas propriedades especificadas. Infelizmente, a modelagem de sistemas complexos por meio de máquinas de estado tem uma desvantagem inerente comumente conhecida como explosão de estad,s que vem a ser o foco deste trabalho.

Palavras-chave: *explosão de estados, máquinas finitas de estado, verificação formal de modelos*

1. Introdução

Com o atual avanço tecnológico, as aplicações têm exigido desenvolvimento de sistemas de software cada vez mais complexos. Esse crescimento da complexidade tem exigido metodologias e ferramentas mais poderosas e otimizadas para a realização dos testes, pois é crescente o surgimento de sistemas envolvendo riscos, tanto econômicos quanto de vidas humanas. Testes de sistemas grandes e complexos é uma atividade bastante difícil e cara, cuja importância no processo de desenvolvimento e manutenção do software não pode ser subestimada. No contexto deste artigo, sistemas complexos são também considerados reativos, pois são organizados como um conjunto de estados e transições entre esses estados, causadas por estímulos chamados eventos, tanto internos quanto externos.

Modelagem é uma maneira de se representar o comportamento de um sistema. A vantagem é que os modelos são mais simples, representam aspectos particulares de interesse, e seu custo é praticamente desprezível quando comparado ao custo total de desenvolvimento do sistema propriamente dito. No caso de sistemas reativos, as máquinas de estado são a escolha natural para a modelagem. Um problema com as máquinas de estado é a explosão no número de estados, que ocorre quando várias atividades independentes acontecem simultaneamente.

Statecharts são uma elegante extensão de máquinas de estado que permitem a representação de hierarquia, concorrência e sincronismo. *Statecharts* é uma das técnicas mais adequadas para a representação de sistemas reativos. Em [2], Binder faz o seguinte comentário: "*Statecharts* cumprem bem a sua tarefa. Eles ocultam complexidade, o que é útil no projeto e na comunicação entre desenvolvedores. Mas, não diminui a complexidade real do comportamento, e, portanto, não diminui o escopo do teste adequado, ou seja, para se ter um conjunto completo de teste, a máquina finita de estados implícita na representação usando *Statecharts* deve ser expandida".

Este trabalho faz parte de uma dissertação de mestrado cujo objetivo pincipal consiste em estender a ferramenta PerformCharts [15], que

calcula o desempenho de um sistema reativo representado em *Statecharts*, para que, através de outra ferramenta chamada Condado, permita também gerar automaticamente casos de testes para sistemas modelados em máquinas finitas de estados. Basicamente, a PerformCharts converte a especificação Statecharts de um sistema reativo e complexo para uma cadeia de Markov e essa cadeia é resolvida para obtenção de probabilidades limite. A cadeia de Markov é uma estrutura que consiste de um conjunto de estados e arcos de transição entre estes estados. A idéia básica é adaptar PerformCharts para modelar um sistema em Statecharts e desdobrar esse modelo em máquina finita de estados. Nesse desdobramento do modelo, é possível ocorrer a explosão de estados, a grande motivação deste estudo é a busca por técnicas para solucionar esse problema.

O artigo está organizado da seguinte forma: a seção 2 discute, resumidamente, verificação formal de modelos. A seção 3 descreve a técnica de representação simbólica. A seção 4 a técnica *"on-the-fly"*. A seção 5, técnicas usadas para redução do modelo. A seção 6, técnica de decomposição do modelo. A seção 7 finaliza com algumas conclusões.

2. Verificação formal de modelos

Verificação formal de modelos é uma técnica automática cujo alvo são os sistemas reativos.

A principal razão para o uso de técnicas formais de modelagem de sistemas é que elas nos fornecem meios de análise de sistemas, tanto no nível de especificação quanto no de implementação. Também nos permitem verificar a conformidade da implementação com a especificação em diferentes níveis de abstração. Atualmente, existem várias ferramentas, comerciais ou não, que auxiliam o uso de formalismos.

Algoritmos para verificação formal de modelos oferecem uma abordagem automática e exaustiva para analisar completamente o sistema. Uma busca exaustiva no espaço de estados é realizada com o objetivo de verificar se o sistema é, realmente, um modelo de suas especificações. Sendo o modelo finito e os recuros disponíveis suficientes (memória, por exemplo) a verificação do modelo sempre termina descobrindo e apontando erros ou provando a corretude do sistema

Em particular, as técnicas de verificação formal de modelos fazem análise do comportamento de sistemas concorrentes com respeito a propriedades selecionadas, ou seja, um modelo finito do sistema é construído e verificado quanto a um conjunto de propriedades desejáveis. As especificações do modelo são tipicamente expressas em alguma lógica temporal ou como automata, o que deu origem a duas abordagens gerais usadas hoje na prática: verificação temporal do modelo e verificação pela teoria dos automata, respectivamente, ou seja, as técnicas são baseadas em como as propriedades são formalizadas [14]. No caso da especificação ser modelada como máquina finita de estados, o sistema pode ser comparado à especificação para determinar se o seu comportamento está em conformidade com a especificação.

Infelizmente, a modelagem de sistemas complexos como máquinas finitas de estados apresenta o problema de explosão de estados.

Várias técnicas têm sido propostas para se evitar a explosão de estados. Estes métodos estão classificados em 4 categorias[7], conforme a figura 1.

Figura 1. Técnicas para conter explosão de estados.

3. Representação simbólica

Em 1987, McMillan percebeu que uma representação simbólica de um sistema poderia ser usada para resolver o problema de explosão de estados [13] [14]. A representação simbólica foi baseada nos Diagramas Binários Ordenados de Decisão (OBDDs), estruturas de dados para representação de um conjunto de vetores de bit de mesmo tamanho ou, de maneira equivalente, uma fórmula booleana. OBDD é um grafo acíclico direcionado cujos nós têm zero ou extamente dois nós sucessores. Nós sem arcos de saída são rotulados com "F" ou "V". Todos os nós restantes são rotulados com uma variável , e seus arcos de saída são rotulados com "0" ou "1" .Somente o nó raiz não tem arco de entrada. A figura 2 mostra um OBDD que representa o conjunto {0011, 0111, 1011, 1100, 1101, 1110, 1111}, ou a fórmula (v_1 ^

v_2) v (v_3 ^ v_4). Um vetor v_1 v_2 v_3 v_4 pertence ao conjunto se e somente se o caminho correspondente com início na raiz até o final do OBDD termina em "V", sendo que nesse caminho o a rco de saída de cada nó v_i é selecionado de acordo com o valor de v_i.

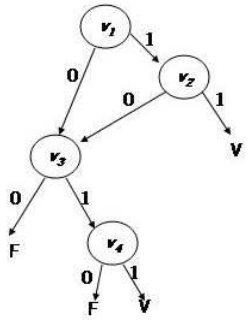

Figura 2. OBDD da fórmula (v_1 ^ v_2) v (v_3 ^v_4)

OBDDs são formados combinando-se quaisquer subárvores com o mesmo formato em uma única árvore e eliminando-se quaisquer nós cujos filhos da esquerda ou direita tenham o mesmo formato. Isto pode ser feito de forma eficiente de baixo para cima. A ordem em que as variáveis percorrem o OBDD é fixa. Assim, cada conjunto tem uma única representação OBDD e, também, testes das operações básicas de conjuntos: união, intersecção, complementação e equivalência podem ser realizadas de maneira eficiente com OBDDs reduzidos. Por outro lado, o tamanho de um OBDD (número de nós) pode depender fortemente da ordem das variáveis, e é difícil saber com antecipação se uma determinada ordem é boa.

O modelo finito de estados de um sistema pode ser expresso em termos de OBBDs: cada estado é representado por uma atribuição de valores booleanos ao conjunto de variáveis de estado associados ao modelo. Se as variáveis de estado não forem binárias, mas pertencerem a um domínio finito D, então uma codificação binária apropriada em D pode ser aplicada. A relação de transição pode então ser expressa como uma função Booleana aplicada a dois conjuntos de variáveis: um conjunto representando o estado atual e o outro representando o novo estado. Esta função é representada como um OBDD.

Concluindo, a representação do sistema passa a ser, portanto, do tamanho da representação OBDD, o que permite a verificação de sistemas (hardware, em particular) que são muitas ordens de magnitude maiores.

4. Verificação "*on-the-fly*"

Segundo a técnica **análise de alcançabilidade**, uma exploração exaustiva de todos os estados e transições de um sistema deve ser executada. Mas, uma busca exaustiva no sistema global é normalmente impossível em sistemas não triviais. Técnicas *on-the-fly* [7] são baseadas no fato de que muitos estados do automata do sistema podem ser gerados somente quando necessário; o automata da propriedade é construído e usado para direcionar a construção do automata do sistema. Alguns estados do sistema podem nunca ser gerados. Uma busca em profundidade pode ser usada para uma exploração *on-the-fly* do sistema, reduzindo de maneira substancial a capacidade necessária de memória.

Para uma busca em profundidade no grafo, a necessidade de memória é relativa ao caminho atual sendo explorado. Esse tipo de busca permite redução de memória enquanto ainda garante exploração exaustiva do espaço de estados. Entretanto, como poderá ocorrer regeneração de estados já visitados, poderá haver um crescimento dramático do tempo necessário para realizar a verificação. Por outro lado, para grafos de alcançabilidade grandes, pode ser impossível armazenar todos os estados. Vários métodos têm sido propostos na tentativa de se conseguir um bom ponto de equilíbrio entre essas duas estratégias. Por exemplo, além de armazenar o estado corrente, pode-se criar uma cache restrita de estados visitados selecionados. Quando a cache estiver totalmente ocupada, estados mais antigos são gradualmente substituídos pelos mais novos de acordo com diferentes políticas que podem ser adotadas. A eficiência desta técnica depende do tamanho da cache e, também, do tamanho do espaço de estados.

Quando o tamanho do problema impossibilita uma verificação exaustiva, a técnica *bit-state hashing* ou *supertrace* [15] é usada para se executar uma busca parcial no espaço de estados. Estados visitados são armazenados numa tabela *hash* H, cujo tamanho depende da memória disponível. Para cada estado s, um único bit com endereço h(s) é usado, onde h é uma função hash que retorna um bit de endereço em H. Se o bit no endereço h(s) tem valor igual a um, então o algoritmo de busca assume que o endereço s já foi visitado.

Tradicionalmente, análise de alcançabilidade tem sido usada com sucesso na detecção de erros do tipo bloqueio *(deadlock)* ou código não exercitado. Entretanto, a aplicabilidade dos algoritmos de

análise de alcançabilidade tem sido estendida com o desenvolvimento de técnicas de verificação do modelo onde a representação da propriedade também é representada em máquinas de estado

Uma vantagem da verificação do tipo *on-the-fly* é que ela necessita prosseguir somente até que um erro seja detectado, neste caso um contra exemplo é gerado a fim de auxiliar o projetista na correção do erro. Esta estratégia é, portanto, vantajosa nos primeiros estágios do projeto, quando a tendência é o sistema conter um número maior de erros.

5. Redução

Técnicas de redução são baseadas na construção parcial ou de uma abstração do espaço de estados de um programa preservando totalmente a capacidade de se provar propriedades de interesse.

Nesta seção serão discutidas as principais estratégias para a redução do espaço de estados.

5.1. Redução de ordem parcial

Uma técnica deste tipo tem como objetivo evitar a geração de partes redundantes do diagrama de estados. Segundo a maioria das técnicas de verificação formal de modelos, concorrência é modelada por meio de intercalação (*interleaving)*, que é o principal fator que contribui para a explosão de estados [8] [10]. Redução de ordem parcial é baseada no fato de que intercalação é ineficiente em muitos casos, pois o resultado final da execução de um conjunto de ações concorrentes é frequentemente independente da ordem de ocorrência das mesmas, o que levou vários pesquisadores a desenvolverem métodos avançados baseados no espaço de estados, onde somente a sequência de um conjunto de ações concorrentes seja executado. Como resultado, evita-se a geração desnecessária de todos as intercalações possíveis, enquanto se preserva as propriedades desejadas.

Métodos de redução de ordem parcial realizam uma busca seletiva na estrutura do sistema. Para cada estado s alcançado durante a busca, eles processam um subconjunto T do conjunto de transições habilitadas em s, e exploram somente transições contidas em T. As técnicas principais propostas na literatura para identificar esses subconjuntos, são baseadas no algoritmo de formação de:

1. conjuntos persistentes: um conjunto persistente T para um estado s é composto de transições habilitadas em s, com a seguinte característica: qualquer transição que seja alcançável do estado s a partir de execuções de transições exclusivamente não contidas em T sendo, portanto, **independente** (isto é, não interage ou afeta) das transições em T.

2. conjuntos dormentes (*sleep)*: conjunto de transições habilitadas em s, mas que não deverão ser executadas a partir do estado s, por já terem sido investigadas em estados passados. Considere duas transições t_1 e t_2 habilitadas e independentes entre si em um estado s e não pertencentes ao conjunto dormentes. Caso t1 seja investigada primeiro, quando t2 for investigada t1 é colocada em dormentes. Transições dormentes só serão acordadas em certas situações, como por exemplo, a ocorrência de uma transição dependente da mesma. O conjunto dormentes pode ser usado para melhorar o método já descrito, que faz uso de uma cache de estados, e também com conjuntos persistentes a fim de reduzir o espaço de estados a ser explorado. Quando a técnica de conjunto persistente não consegue selecionar transições independentes em um estado, conjunto de dormentes (sleep) pode evitar a exploração de múltiplas intercalações dessas transições.

5.2. Minimização do modelo

A tarefa de verificação consiste em estabelecer que um sistema S satisfaz alguma propriedade f. Consideremos alguma equivalência semântica R que preserva a propriedade f e, uma máquina de estado mínima S' tal que (S, S') $\in$ R. Então S satisfaz f se e somente se S' satisfaz f. Dizemos que S' é o quociente de S em relação a R. O processo de construção de S' a partir de S é chamado minimização. Se R reflete a aplicação de uma abstração a S, então S' contém menos estados do que S. A técnica de análise da máquina de estados minimizada correspondente pode aumentar de maneira significativa o tamanho dos sistemas passíveis de serem analisados com uma mesma disponibilidade de recursos. Minimização deve obter o diagrama minimizado S' de estados sem antes ter que gerar o diagrama completo do sistema.

Considere um sistema descrito por uma expressão composta que agrupa máquinas de estados individuais. Tal expressão reflete uma organização específica dos componentes do sistema em uma estrutura hierárquica. Minimização Composicional executa a minimização em partes, partindo do nível mais baixo para o mais alto. As expressões de composição de cada nível definem

quais máquinas de estado devem fazer parte da composição a fim de obter máquinas de estado dos subsistemas de cada nível. O resultado de cada composição deve ser minimizado.

No processo acima descrito, o diagrama de estados dos subsistemas intermediários é construído por meio de análise de alcançabilidade. Essa abordagem de composição incremental e redução é normalmente chamada análise de alcançabilidade composicional (CRA). Este tipo de abordagem é capaz de localizar o efeito de alterações, assim é particularmente indicada para análise de sistemas que evoluem. Quando o sistema sofre alterações, somente os subsistemas afetados necessitam ser novamente analisados. As técnicas do tipo CRA podem ser combinadas com representação simbólica. Técnicas do tipo *on-the-fly* podem ser usadas na última etapa do CRA, quando o diagrama global de estados do sistema é explorado. Nem sempre as máquinas intermediárias de estado têm um tamanho menor do que o diagrama global de estados do sistema, pois os subsistemas podem conter muitas seqüencias de execução desnecessárias dentro do contexto do sistema final. Esse fenômeno é conhecido como explosão intermediária de estados. Um tipo de solução para esse problema é através do uso de interfaces. Uma interface é um processo que representa um conjunto de seqüencias de execução autorizadas que podem ser executadas pelo subsistema no ambiente específico.

5.3. Abstração

Quando o modelo inicial é complexo demais para ser analisado de forma razoável então um modelo pode ser construído omitindo-se detalhes irrelevantes e sucessivamente analisados. Esta estratégia parte da hipótese de que se o modelo abstrato está de acordo com as especificações, o modelo incial também estará, mas se o modelo abstrato não satisfaz as especificações não significa que o modelo inicial não satisfaz. Isso significa que cuidados devam ser tomados para se criar um modelo abstrato que preserve a especificação. Várias técnicas de abstração têm sido propostas em [1] [5] [6] [12]. A maioria das estratégias de redução do espaço de estados de sistemas é baseada em algum tipo de abstração do sistema em análise. De fato, a técnica de minimização composicional também é vista como um tipo de técnica de abstração, pois abstrai detalhes do comportamento do sistema com base na descrição da estrutura do sistema e na especificação da interação entre seus componentes.

No caso de programas com comportamento dependente de dados, [12] propõe usar verificação formal de modelos em aproximações de seu espaço de estados, quando os mesmos são grandes demais, ou até infinito. Aproximações são baseadas no mapeamento de conjuntos de faixa de valores que as variáveis do programa podem assumir em conjuntos de valores abstratos. Elas são construídas diretamente do texto do programa, sem antes construir o sistema de transição original.

Outros tipos de abordagens de abstração incluem exploração de simetrias no sistema para a geração do espaço de estados [11] e para a verificação formal de modelos [5]. Em geral, técnicas de abstração para programas com comportamento dependente de dados são menos aplicáveis a sistemas concorrentes, onde, como mencionado, o foco está nas interações entre os processos.

6. Modularização

Verificação composicional usa o conceito de modularização, ou seja, explora a decomposição natural de um sistema complexo em componentes menores. As propriedades dos componentes do sistema são verificadas antes. Essas propriedades são então combinadas para a dedução de propriedades do sistema global. Uma questão que surge é que, freqüentemente, propriedades de subsistemas são satisfeitas somente quando condições específicas são impostas ao seu ambiente. Uma abordagem proposta para tratar essa questão é usar processos de interface que modelam o ambiente do subsistema. Em geral, decompor propriedades do sistema global em propriedades locais de seus subsistemas é uma tarefa complicada. Além disso, deve-se provar que tais decomposições estejam corretas, isto é, que a satisfação das propriedades locais dos subsistemas implicam na satisfação de algumas propriedades globais do sistema. A técnica necessita ser suportada por ferramentas automatizadas a um nível alto a fim de se tornar amplamente utilizado por engenheiros de software. Como reporta Kurshan em [12], “encontrar heurísticas úteis para determinar decomposições de propriedades do sistema global em propriedades locais de subsistemas é um dos problemas mais importantes em aberto no campo”.

7. Conclusões

Este trabalho apresentou um estudo de técnicas usadas em verificação formal de modelos para prevenção do problema **explosão de estados**. No caso de sistemas não triviais, se medidas preventivas não forem tomadas, a ocorrência da explosão de estados é praticamente certa. Algumas formas básicas de prevenção, consistem em: abstrair informações complexas do modelo, que não afetem o comportamento do sistema; ou, baseado na estratégia **dividir para conquistar** usar algum tipo de técnica de decomposição do sistema em modelos menores, onde, um modelo conterá algumas informações e deixará de conter outras que poderão estar em outro modelo para serem testadas; uso de representação simbólica; explorar o sistema de forma parcial; usar técnicas *on-the-fly* que são interessantes nos primeiros estágios do projeto, quando a tendência é conter um número maior de erros.

Enfim, não existe uma técnica perfeita, ou seja, eficiente para todos os tipos de sistemas. Combinar técnicas, na maioria das vezes, traz bons resultados.

O trabalho em andamento consiste em adaptar uma ou mais dessas técnicas a fim de minimizar o problema da explosão de estados durante o processo de desdobramento de um modelo em Statecharts para máquinas finitas de estados estendidas.

8. Referências

[1] Bharadwaj, R., Heitmeyer, C., "Verifying SCR Requirements Specifications Using State Exploration", in *Proc. Of the 1st ACM Sigplan Workshop on Automated Analysis of Software* (AAS'97), Paris, France, January 1997, pp. 9-23. R. Cleaveland and D. Jackson, Eds.

[2] Binder, R. V., "TESTING OBJECT-ORIENTED SYSTEMS Models, Patterns, And Tools", Addison-Wesley, USA, 2001.

[3] Clarke, E.M., Grumberg, O. & Long, D, "Model Checking and Abstraction", *ACM Transactions on Programming Languages and Systems (ACM-TOPLAS)*, vol. 16(5), pp. 1512-1542, September 1994.

[4] Clarke, E.M., Grumberg, O. & Long, D, "Model Checking ", vol. 152, *Springer-Verlag Nato ASI series F*, 1996.

[5] Clarke, E.M., Filkorn, T. & Jha, S., "Exploiting Symmetry in Temporal Logic Model Checking", *Formal Methods in System Design*, vol. 9, pp. 77-104, 1996.

[6] Cousot, P., Cousot, R., "Refining Model Checking by Abstract Interpretation ", *Journal of Automated Software Engineering, special issue on Automated Analysis of Software*, vol. 6(1), pp. 69-95, January 1999, R. Cleaveland and D. Jackson, Eds.

[7] Dimitra, G., "Model Checking for Concurrent Software Architectures", *Ph.D. Thesis Imperial College of Science, Technology and Medicine, University of London, Department of Computing*, 1999.

[8] Godefroid, P., Wolper, P., "Using Partial Orders for the Efficient Verification of Deadlock Freedom and Safety Proprierties", in *Proc. of the 3th International Workshop on Computer-Aided Verification (CAV'91)*, Aalborg, Denmark, July 1991. Lecture Notes in Computer Science 575, pp. 332-342, K. G. Larsen and A. Skou, Eds.

[9] Holzmann, G. J., Godefroid, P., & Pirottin, D., "Coverage Preserving Reduction Strategies for Reachability Analysis", in *Proc. of the IFIP WG 6.1 International Symposium on Protocol Specification, Testing and Verification (PSTV'92)*, Orlando, Florida, June 1992, pp. 349-364.

[10] Holzmann, G. J. & Peled, D. "An Improvement in Formal Verification", in *Proc. of the 7th Conference on Formal Description Techniques (FORTE'94)*, Bern, Switzerland, October, 1994, pp. 177-194.

[11] Ip, C. & Dill, D., "Better verification through symmetry", in *Proc. of the 11th IFIP WG10.2 International Conference on Computer Hardware Description Languages and their Applications – CHDL'93*, Ottawa, Ontario, Canada, April 1993. D. Agnew, L. J. M. Claesen, and R. Camposano, Eds.

[12] Kurshan, R. P., Merrit, M., Orda, A. & Sachs, S. R.: "A Structural Linearization Principle for Processes", *Formal Methods in System Design 5*, 1994, pp. 227-244.

[13] McMilan, K. L., *Symbolic Model Checking: An approach to the State Space Explosion*, Kluwer Academic Publishers, 1993.

[14] Valmari, A., "The State Space Problem", in *Proc. of Petri Nets I: Basic Models*, 1998. Lecture Notes in Computer Science 1491, pp. 429-528. Springer-Verlag, Ed.

[15] Vijaykumar, N.L.; Carvalho, S.V.; Abdurahiman, V. On proposing Statecharts to specify Performance Models. *International Transactions in Operational Research (ITOR)*, 9(3), pp.321-336, May 2002.