5
Discussão sobre Otimização dos Modelos

Um dos problemas enfrentados pela verificação de modelos é a explosão de estados. Na literatura encontramos diversas técnicas como avaliação parcial, abstração e refinamento de modelos e a construção de grafos de decisão que tentam amenizar este problema. No caso da verificação de protocolos e algoritmos distribuídos, a explosão de estados está diretamente ligada ao tamanho da rede (número de nós) na qual se deseja verificar o protocolo.

Nos trabalhos que analisamos, o tamanho da rede na validação de protocolos é usualmente definido de forma empírica (WPP04, Lau02, CGMT04, BHE04). Ou seja, dada as propriedades que se desejam verificar, obtém-se um tamanho "razoável"para a rede através de simulações de cenários específicos. Com isso, espera-se que este tamanho de rede, idealmente mínimo, seja suficiente (valha para redes maiores) para a verificação da corretude do protocolo.

Por exemplo, na verificação do protocolo DSR apresentada em (Lau02), os autores argumentam que com cinco (5) nós, fixando dois (2) como origem e destino, se consegue representar todos os cenários "interessantes"para a verificação deste protocolo (figura 5.1).

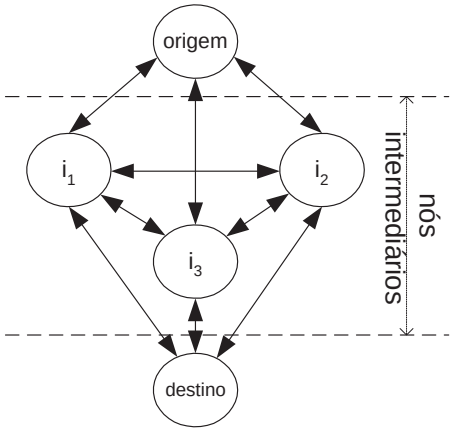

Figura 5.1: Modelo para a verificação do protocolo DSR

Na verificação do protocolo RDP apresentada em (BHE04), para analisarmos o comportamento do protocolo no hand-off[1], que ocorre quando a

[1]O hand-off ocorre no RDP sempre que uma estação móvel faz uma requisição e, antes de receber sua resposta, se transfere para outra agência

PUC-Rio - Certificação Digital Nº 0115606/CA

estação móvel se desloca após uma requisição e antes do recebimento da resposta, utilizamos o modelo apresentado na figura 5.2. Ou seja, uma rede com apenas uma (1) instância de cada módulo e duas (2) instâncias das estações base (MSS), foi suficiente para a verificarmos o protocolo RDP com respeito à garantia a entrega de mensagens. Como visto na seção 4.1.1, esta propriedade só não é válida para os cenários onde o agente móvel se desloca de forma incessante, impossibilitando o recebimento da mensagem.

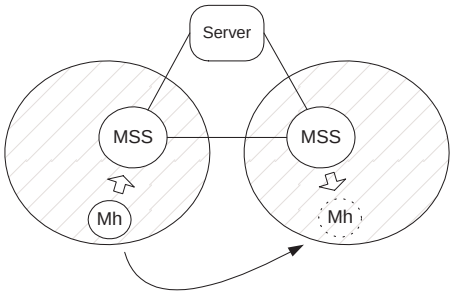

Figura 5.2: Modelo para a verificação do protocolo RDP

PUC-Rio - Certificação Digital Nº 0115606/CA

Em nossa arquitetura, a topologia utilizada por um protocolo pode ser fornecida de diversas formas, as quais foram descritas na seção 3.2.2. Nestas, de alguma forma especificamos o número ou intervalo do número de instâncias da rede em sua configuração inicial. Entretanto, podem ocorrer situações em que o número de instâncias fornecidas seja incompatível com a especificação a ser verificada. Por exemplo, seria incompatível modelarmos o problema dos filósofos com cinco (5) filósofos e apenas um (1) talher, sendo dois (2) o mínimo para que algum filósofo possa comer. Assim, propomos o conceito de modelo mínimo (MM), que definirá o menor número de instâncias necessárias para que se possa verificar uma especificação frente a uma certa propriedade. Além do limitante inferior, também discutimos como a rede é limitada superiormente.

Uma justificativa para a definição de um MM para uma especificação está na filosofia de LEP, cuja abstração tenta livrar o projetista de se preocupar com detalhes mais concretos da especificação. Por exemplo, protocolos como DSR e RDP tem o mesmo comportamento previsto em suas especificações, independentemente do número de nós na rede.

Originalmente projetamos MM para limitar o tamanho das redes geradas pela arquitetura. Ou seja, estávamos interessados em restringir a dimensão da rede em seu estágio inicial. Entretanto, esta idéia de MM também pode ser aplicada para ambientes dinâmicos, onde temos a criação e remoção de nós. Neste caso, o MM define sobre quais cenários (configurações da rede) uma dada propriedade pode ser verificada. Apesar disso, a apresentação que se segue visa fornecer informações à respeito da dimensão das redes necessárias

para a verificação de uma propriedade apenas para a definição da configuração inicial destas redes.

5.1 Modelo Mínimo

5.1.1 Definições

Seja M um módulo de uma especificação S, *Cmds* uma sequência de comandos em M, P um processo descrito por M, m um tipo de mensagem em S, *ch* um canal de mensagens, que em LEP são os próprios identificadores de elementos (pronomes), μ é um modelo de S, ϕ uma propriedade sobre S, $\mathrm{S}_f(\phi)$ o conjunto de subfórmulas de ϕ e *Pr(k)* o conjunto de pronomes parametrizados que requerem pelo menos k instâncias do respectivo pronome:

- Se P pode executar um comando de recebimento *ch?m* e *P'* pode executar um comando de envio *ch!m*, então *P(ch?m)||P'(ch!m)* representa um possível sincronismo entre P e *P'*; um possível sincronismo é chamado aqui de um **elo estático** de S;
- Se existe uma transição $ch?m \rightarrow Cmds$ num módulo M e $ch'!m' \in Cmds$, então $ch?m \rightarrow ch'!m'$ também é um **elo estático**;
- um **caminho** C é uma possível sequência de elos estáticos;
- C_ϕ é um possível caminho de execução para a propriedade ϕ se esta é válida em algum ponto de C; $C_{\cap\phi}$ é o conjunto de interseções de caminhos C_ϕ; as interseções são importantes pois definem os trechos dos caminhos necessários para se validar $_\phi$;
- S está num estado **proclesslock** com respeito à uma propriedade ϕ se $\exists P \in S, \exists ch?m \in C_{\cap\phi} / P(ch?m) \wedge \nexists Q \in S / Q(ch!m)$; ou seja, P pode esperar indefinidamente por uma mensagem m já que não há nenhum processo Q que possa se sincronizar com P;
- $\forall p \in S_f(\phi), p \in Pr(k)$, μ **atende** ϕ se $\{ \mathrm{p}_1, ..., \mathrm{p}_k \} \subseteq \mu$;
- um modelo mínimo μ_m para uma especificação S e uma propriedade ϕ é o menor modelo onde o estado **proclesslock** nunca ocorre e μ_m **atende** ϕ.

PUC-Rio - Certificação Digital Nº 0115606/CA

5.1.2 Cálculo do Modelo Mínimo

Este cálculo visa obter o número de instâncias e conexões necessárias de cada tipo de nó existente numa especificação em LEP, de forma que o modelo formado por essas instâncias possa validar a especificação e propriedade dadas. Tem como entradas a especificação e a propriedade a ser verificada. o termo "tipo de nó"deverá ser considerado aqui quando os nós em questão tiverem comportamentos distintos, como no caso de redes estruturadas (em termos LEP, significa termos mais de um (1) numa mesma especificação). Para as redes ad hoc, em geral, temos um único tipo de nó.

O algoritmo baseia-se na análise estática da especificação do protocolo. Nesta análise, deduzimos o número mínimo de nós na rede através da detecção das dependências entre comandos de envio e recebimento de mensagens, e da utilização de pronomes com parâmetros referentes à quantidade. Numa transição em LEP, os comandos que constituem a ação dependem da pré-condição associada (Pré-condição → Ação). No caso de termos um comando de recebimento como pré-condição, necessitamos de uma nova instância sincronizável, ou seja, que possa enviar a mensagem para que a pré-condição seja satisfeita.

Além do número de nós ser dado por essa relação de dependências, também obtemos informações quantitativas a partir dos pronomes parametrizados. Por exemplo, o pronome *any(k)*, seja num comando de envio, recebimento ou numa propriedade, requer que ao menos k instâncias do referente tipo sejam criadas. Neste caso, se definimos a fórmula <> any(3)!error (em algum momento quaisquer três (3) elementos enviam uma mensagem de erro), a rede necessariamente deverá conter pelo menos 3 nós. Do contrário, a propriedade sempre será inválida.

Para ilustramos estas dependências, apresentamos o algoritmo de eleição de um líder numa rede arbitrária em LEP (figura 5.3).

O conjunto de dependências para a especificação apresentada na figura 5.3 é mostrada na figura 5.4.

Considerando a propriedade <>*any?win* (eventualmente algum elemento recebe a mensagem *win*), identificamos no conjunto os locais onde a propriedade *any?win* ocorre (5.5).

Para a existência do comando de recebimento destacado na figura 5.5, precisamos inicialmente de uma (1) instância do módulo *candidate*, a qual eventualmente se sincronizará com este através de um comando de envio. Na figura 5.6, destacamos os nós (neste caso apenas um (1)) sincronizáveis com o nó *c?win*. Adicionamos ao modelo mínimo uma (1) instância referente a este

PUC-Rio - Certificação Digital Nº 0115606/CA

```
module candidate
        int my, p, count;
        init -> count = 0; my = this; neighbours!msg(my, count);
        this?win -> stop;
        this?msg(p, count) ->
                if ((p > my) or ((p == my) and (count < topology.size))) then
                        my = p;  count = count + 1;
                        neighbours!msg(my, count);
                else
                        if ((p == this) and (count > topology.size)) then
                                everyone!win;
                                stop;
                        endif
                endif
endmodule
```

Figura 5.3: Eleição de um líder numa rede arbitrária em LEP

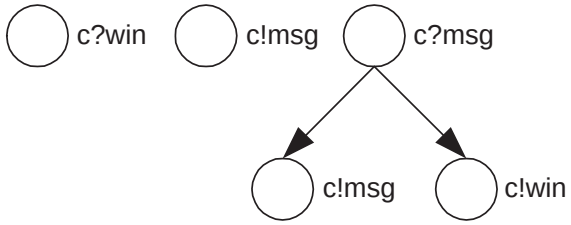

Figura 5.4: Conjunto de dependências para a especificação da figura 5.3

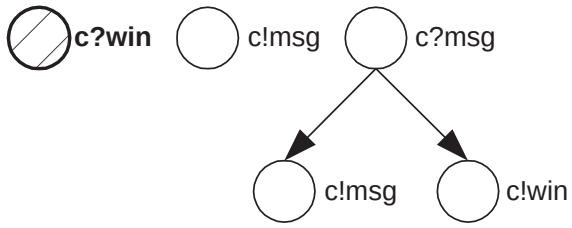

Figura 5.5: Ocorrência da propriedade *any?win* no conjunto em destaque

nó sincronizável encontrado.

Na sub-árvore do nó *c!win*, a raíz do nó é o comando de recebimento *c?msg*. Ou seja, devemos procurar no conjunto de nós os sincronizáveis com *c?msg* (nós 1 e 2 na figura 5.7).

Para o nó sincronizável 1, apenas mais uma (1) instância precisa ser criada, já que este não depende de nenhum outro comando. Entretanto, para o nó 2 há uma dependência *recursiva*. Ou seja, usando este caminho podemos criar um número indeterminado de instâncias. Como o objetivo é automatizar o processo, optamos pela dependência não-recursiva.

Um caso especial é a ocorrência do pronome *sender*. Como vimos, numa troca de mensagens o pronome *sender* faz referência ao remetente desta mensagem. Quando este ocorre em alguma cadeia de dependências, interpretamos como se um nó já criado estivesse sendo reaproveitado. Ou seja,

PUC-Rio - Certificação Digital Nº 0115606/CA

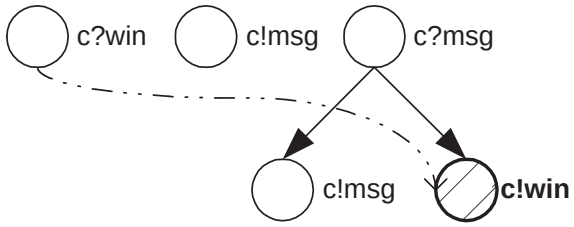

Figura 5.6: Nó sincronizável com o nó *c?win* em destaque

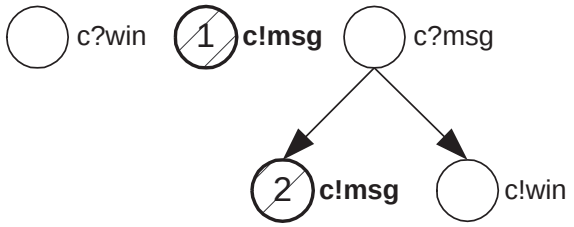

Figura 5.7: Nós sincronizáveis com o nó *c?msg*

neste caso não inserimos um novo nó no modelo mínimo.

Quando a quantidade de dependências existentes na especificação é pequena, o modelo mínimo tende a ser pequeno também e, eventualmente, incompatível com o protocolo e propriedade. Para estes casos, uma opção é manipularmos a propriedade. Por exemplo, poderíamos alterar a propriedade <> *any?win* para <> *any(5)?win*, o que nos obrigaría a ter pelo menos cinco (5) instâncias na rede.

Definições Seja S uma especificação em LEP, r um comando de recebimento de mensagem e c um comando de envio de mensagem:

(i) se r $\rightarrow$ c é um elo estático em S, então existe uma dependência entre c e r em S;

(ii) *Floresta de dependências* é o grafo resultante da união do conjunto de árvores de dependências r $\rightarrow$ c existentes em cada transição de S; os nós são os comandos de envio e recebimento e as arestas indicam as dependências;

(iii) uma dependência é *recursiva* se existe uma transição onde r e c são sincronizáveis;

Algoritmo Dada uma especificação S, uma propriedade p, o conjunto de subfórmulas sem quantificadores de p ($S_f(p)$) e o conjunto de nós na floresta de dependências de S referentes à $S_f(p)$, o número de nós necessários para se validar p em S é obtido da seguinte maneira:

PUC-Rio - Certificação Digital Nº 0115606/CA

1. Identificar agentes e objetos existentes na propriedade p e adicionar suas quantidades ao MM;

2. Construir a floresta F de dependências de S;

3. Identificar em F os nós onde ocorre alguma sub-fórmula de $S_f(p)$;

4. Para cada ocorrência de $S_f(p)$:

   (a) Adicionar ao MM um (1) nó do tipo indicado pela sub-fórmula, caso este ainda não exista no MM;

   (b) Caso esta ocorrência tenha ancestral em F que seja um comando de recebimento e não seja sincronizável com esta ocorrência (para evitar recursão indefinida):

      i. Buscamos em F os nós sincronizáveis a este comando de recebimento; para cada nó sincronizável, inserimos no MM o número de nós necessários para que haja a sincronização com o nó corrente (ou seja, desconsideramos o nó corrente dentro do modelo já formado); ainda para cada nó sincronizável, executamos à partir do item 4a;

      ii. Caso contrário, se é um comando de recebimento sem ancestrais, repetimos o item 4(b)i para este comando;

5. Ao final, após analisarmos todas as ocorrências de sub-fórmulas, o MM resultante é formado por todos os nós adicionados.

Nos estudos de caso verificados na tese, o modelo mínimo obtido do protocolo RDP para a propriedade *[] (mh!request → <>mh?response)* foi exatamente o ilustrado na figura 5.2, ou seja: duas (2) estações base (MSS), uma (1) estação móvel (Mh) e um (1) servidor (Server). Para o protocolo DSR, um modelo mínimo com apenas dois (2) nós foi obtido.

5.2 Limitante Superior

Complementando a seção anterior, nesta seção descreveremos de que maneira o tamanho dos modelos é limitado. Diferentemente do termo *modelo* utilizado em MM (referente ao tamanho das redes), aqui o objetivo é determinar um número máximo de estados visitados numa sequência de execução sem perda de generalidade. Ou seja, se uma propriedade não vale para um modelo, esta deve ser invalidada em algum traço de execução com este número máximo

PUC-Rio - Certificação Digital Nº 0115606/CA

de estados. Inversamente, se uma propriedade é válida, esta vale para todas as sequências limitadas por este número máximo de estados.

No processo de verificação, um verificador de modelos gera todos os traços de execução possíveis de um modelo para uma propriedade (figura 5.8).

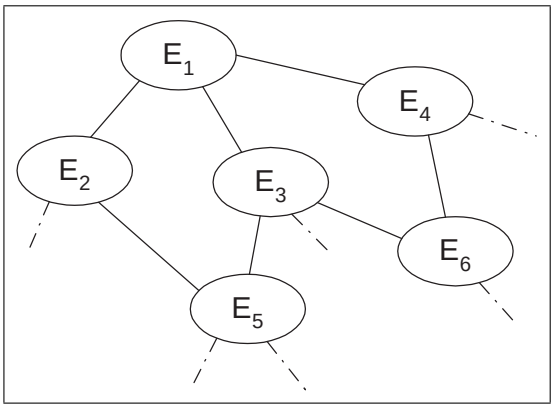

Figura 5.8: Traços de um modelo gerados por um verificador de modelos

No nosso caso, as transições entre os estados E_k (mundos) da figura 5.8 ocorrem em função: das movimentações na rede; das trocas de mensagens entre os processos, e; das mudanças de estado locais de cada processo. Assim, o MM para uma especificação e uma propriedade elimina destes traços os estados onde a verificação da propriedade não é possível pela quantidade insuficiente de nós.

Além disso, estamos interessados em determinar a dimensão deste modelo (conjunto de estados da figura 5.8), ou seja, o número estados que seja suficiente para verificarmos uma propriedade sobre este modelo. Para tal, nos baseamos no conceito de filtração (Gol87). A filtração de um modelo M, referente à uma fórmula ϕ, produz um modelo finito M_f de no máximo $|S_f(\phi)|$ mundos, onde $S_f(\phi)$ é o conjunto de sub-fórmulas de ϕ. Assim, para as lógicas temporais como LTL e CTL que tem a Propriedade do Modelo Finito (*FMP - Finite Model Property*), uma fórmula é válida em M_f se, e somente se, esta vale para o seu modelo M original.

5.2.1 Definições

(i) Um modelo de Kripke é uma quádrupla M=(S,S_0,R,L) sobre um conjunto AP de proposições atômicas, onde S é o conjunto de estados finito, $S_0 \subseteq$S é o conjunto de estados iniciais, R$\subseteq$S$\times$S uma relação de transição total e L:S$\rightarrow 2^{AP}$ é a função que rotula cada estado com o conjunto de proposições atômicas que valem neste estado.

PUC-Rio - Certificação Digital Nº 0115606/CA

(ii) Seja μ um modelo de Kripke e ϕ uma propriedade em CTL ou LTL então, pela FMP:

$$\mu \models \phi \text{ sss } \mu_f \models \phi, |\mu_f| \leq 2^{|S_f(\phi)|}$$

, onde μ_f é a filtração do modelo μ.

(iii) Seja S uma especificação fornecida em LEP, *Top* um tipo de topologia adotada por S e ϕ_{LEP} uma propriedade para o modelo de S em LEP, $\mu_m(S, Top, \phi_{LEP})$ é o modelo mínimo de S sobre *Top* e ϕ_{LEP}.

Como μ_f é o modelo filtrado de μ, poderíamos encontrar nos estados de μ_f todas as formações de redes, incluindo as redes com quantidade insuficiente de nós, as quais podem não permitir a correta verificação da especificação. Para evitar estes cenários indesejáveis, forçamos para que o modelo filtrado μ_f só contenha redes maiores que o modelo mínimo μ_m, ou seja:

$$\forall E_k \in \mu_f, |E_k| \geq |\mu_m|$$

, onde E_k é um estado de mu_f e $|E_k|$ se refere ao tamanho da rede no estado k.

Quanto às propriedades, ϕ_{LEP} pode ser vista como uma propriedade abstrata, pois contém os pronomes de LEP. Para utilizarmos a FMP precisamos torná-la mais concreta, compatível com o modelo analisado. Ou seja, precisamos instanciar os pronomes existentes na propriedade com os constituintes do modelo. Para tal nos baseamos nas seguinte regras, supondo Φ o conjunto de fórmulas atômicas e $S_f(\phi)$ o conjunto de sub-fórmulas de ϕ:

- $\phi \in \Phi, |\phi| = 1$
- $\phi \in \{none, everyone\}, |\phi| = t,\ t$ é o tamanho da rede
- $\phi = v, |\phi| = n_v,\ v$ é uma variável na propriedade e n_v é o número de instâncias do tipo de v
- $\phi = any(k), |\phi| = k$
- $\phi \notin \Phi \cup \{none, everyone, any(k)\}, |\phi| = \Sigma|S_f(\phi)|$

Ou seja, quando a fórmula é um pronome parametrizado, seu grau é dado pelo número de instâncias que o pronome referencia. Se a fórmula é o pronome *none* ou *everyone*, o grau depende do tamanho da rede. Se é uma variável, o grau é dado pelo número de instâncias do tipo da variável. Caso só tenhamos um tipo de nó, o grau de uma variável é o tamanho da rede, assim como o pronome *everyone*. Finalmente, quando é composta por sub-fórmulas, o grau da fórmula é dado pela soma dos graus das sub-fórmulas.

PUC-Rio - Certificação Digital Nº 0115606/CA

5.2.2 Aplicação

Para ilustrarmos como obtemos o limite superior do modelo de uma especificação em LEP, consideraremos a especificação do ataque combinado apresentada na figura 5.9:

```
topology is Star(commander:1,soldier:1..6);
module commander
        init -> everyone!agree;
        this?yes(any(3)) -> everyone!consensus;
        this?no -> everyone!cancel;
endmodule
module soldier
        this?agree -> sender!yes;
        this?agree -> sender!no;
        this?cancel ->
        this?consensus ->
endmodule
```

Figura 5.9: Algoritmo de consenso especificado em LEP

Consideremos a propriedade $<>p!consensus$ para análise, a qual questiona se em algum momento um processo p (neste caso, o comandante) envia uma mensagem ordenando o ataque (mensagem *consensus*). Para esta propriedade, temos duas (2) sub-fórmulas ($p!consensus, <> p!consensus$). Com isso, podemos concluir erroneamente que o modelo filtrado tem tamanho menor ou igual a quatro (4, que é igual a $2^{|S_f(<>p!consensus)|}$. Entretanto, vemos pela especificação que a transição onde ocorre o envio da mensagem *consensus* tem como pré-condição o comando de recebimento *this?yes(any(3))*. Naturalmente o modelo tem que refletir esta pré-condição. Ou seja, o significado da propriedade $<> p!consensus$ de fato é uma conjunção de todas fórmulas necessárias para que esta ocorra. Assim, teríamos:

$$<> (p!agree \text{ and } <> (p_{1..3}!yes \text{ and } <> p!consensus)).$$

Ou seja, considerando que as mensagens *agree* e *consensus* só podem ser enviadas por um processo (comandante), e $p_{1..3}$ referem-se à três (3) dos seis (6) soldados, temos:

- $|<> p!consensus| = 2$
- $|<> p_{1..3}!yes| = |<> (p_1!yes \; and \; p_2!yes \; and \; p_3!yes)| = \binom{6}{3}$
- $|<> p!agree \; and \; (<> p_{1..3}!yes \; and \; (<> p!consensus))| = 3 + \binom{6}{3} + 2$

PUC-Rio - Certificação Digital Nº 0115606/CA

Assim, a dimensão do modelo para a verificação da especificação do ataque coordenado é limitada por $2^{5+\binom{6}{3}}$, ou seja, 2^{125}. Este valor representa o tamanho do conjunto de todas as simulações da especificação do ataque coordenado para a propriedade dada. Ou seja, quanto maior é este valor, maior é o esforço do verificador de modelos para concluir a verificação. Entretanto, neste valor não estão sendo consideradas as otimizações feitas pelos verificadores que tornam mais eficiente o processo de verificação.

PUC-Rio - Certificação Digital Nº 0115606/CA