UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE

CENTRO DE CIÊNCIAS EXATAS E DA TERRA

DEPARTAMENTO DE INFORMÁTICA E MATEMÁTICA APLICADA

PROGRAMA DE PÓS-GRADUAÇÃO EM SISTEMAS E COMPUTAÇÃO

PROVA AUTOMÁTICA DE SATISFATIBILIDADE MÓDULO TEORIA APLICADA AO MÉTODO B

Cláudia Fernanda O. K. Tavares

Natal / RN

Setembro de 2008

CLÁUDIA FERNANDA O. K. TAVARES

PROVA AUTOMÁTICA DE SATISFATIBILIDADE MÓDULO TEORIA APLICADA AO MÉTODO B

Dissertação apresentada em 27 de julho de 2007 e submetida ao Programa de Pós-Graduação em Sistemas e Computação do Departamento de Informática e Matemática Aplicada da Universidade Federal do Rio Grande do Norte como parte dos requisitos para a obtenção do grau de Mestre em Sistemas e Computação (MSc.).

Prof. Dr. David Boris Paul Déharbe
Orientador

Natal / RN
Setembro de 2008

Dedico este trabalho à mamãe, Ana Lígia, a papai, Orcines, e aos meus irmãos, Cláudia e Alexandre.

AGRADECIMENTOS

Durante o período pré-vestibular, a dúvida sobre que curso de graduação escolher era constante, de forma que cheguei a pensar em adiar por um ano esta escolha. Mas seria preciso fazê-la algum dia, então que fosse logo. Sem pensar muito, optei por Ciências da Computação. Apesar de gerar algumas insatisfações, a minha decisão logo foi aceita por todos.

No decorrer do curso, novas dúvidas surgiram. Era necessário saber se eu realmente tinha tomado a decisão certa. Pensei em desistir e mudar de curso algumas vezes, mas trocar o incerto pelo duvidoso parecia não valer a pena. Realmente não valia. A recompensa pelo esforço veio em dose dupla: colação de grau em Ciências da Computação e bolsa da CAPES para dar continuidade aos estudos no mestrado. Aproveitei o incentivo da bolsa e resolvi continuar a caminhada. Dois anos e meio de mestrado. Mais um turbilhão de dúvidas sobre a decisão tomada. De novo, a sensação de que persistir é melhor que desistir. Então, persisti.

Hoje, mesmo estando prestes a obter o título de Mestre em Sistemas e Computação, e já cursando um doutorado em Informática na França, financiado pela CAPES, muitas dúvidas sobre a permanência nesta área ainda me perseguem. Mas descobri que elas sempre existirão. Não por causa do curso, mas por minha causa. Por causa da mania de me questionar até sobre o inquestionável. As dúvidas aprenderam a conviver comigo e eu com elas. Posso dizer que, mais do que alegre, estou satisfeita com a minha escolha e com a insistência na não-desistência.

Parando mais uma vez para refletir, preciso agradecer imensamente a Deus que sempre me guiou e pedir que Ele sempre me acompanhe.

Gostaria de fazer, ainda, alguns agradecimentos:

A mamãe, Ana Lígia, e a papai, Orcines, o amor, carinho e dedicação destinados a mim, a compreensão da minha ausência em muitos momentos e a priorização da minha educação, independente de qualquer coisa.

A minha irmã Cláudia, o carinho e o apoio incondicional, e ao meu irmão Alexandre, que sempre acreditou no meu sucesso.

Ao prof^{o} David, a crença na realização deste trabalho, e à prof^{a} Anamaria, o auxílio nas pesquisas.

Aos demais professores do curso, os ensinamentos que permitiram a minha formação, e aos funcionários do DIMAp, a solicitude no cumprimento de suas funções.

Aos meus avós, tios, tias, primos e primas, a compreensão do meu nervosismo nesta reta final e a torcida pela minha vitória.

A Fabrício, o carinho e a ajuda em momentos cruciais, e também, o incentivo e apoio diante das dificuldades.

Aos meus amigos, presentes e ausentes, a fé depositada em mim e a força em vários momentos cruciais.

A todos os que foram e são, de alguma forma, importantes na minha vida.

Muito obrigada.

RESUMO

Este trabalho apresenta uma extensão do provador haRVey destinada à verificação de obrigações de prova originadas de acordo com o método B.

O método B de desenvolvimento de software abrange as fases de especificação, projeto e implementação do ciclo de vida do software. No contexto da verificação, destacam-se as ferramentas de prova Prioni, Z/EVES e Atelier-B/Click'n'Prove. Elas descrevem formalismos com suporte à checagem satisfatibilidade de fórmulas da teoria axiomática dos conjuntos, ou seja, podem ser aplicadas ao método B.

A checagem de SMT consiste na checagem de satisfatibilidade de fórmulas da lógica de primeira-ordem livre de quantificadores dada uma teoria decidível. A abordagem de checagem de SMT implementada pelo provador automático de teoremas haRVey é apresentada, adotando-se a teoria dos vetores que não permite expressar todas as construções necessárias às especificações baseadas em conjuntos. Assim, para estender a checagem de SMT para teorias dos conjuntos destacam-se as teorias dos conjuntos de Zermelo-Frankel (ZFC) e de von Neumann-Bernays-Gödel (NBG).

Tendo em vista que a abordagem de checagem de SMT implementada no haRVey requer uma teoria finita e pode ser estendida para as teorias não-decidíveis, a teoria NBG apresenta-se como uma opção adequada para a expansão da capacidade dedutiva do haRVey à teoria dos conjuntos. Assim, através do mapeamento dos operadores de conjunto fornecidos pela linguagem B a classes da teoria NBG, obtem-se uma abordagem alternativa para a checagem de SMT aplicada ao método B.

Área de Concentração: Métodos Formais
Palavras-chave: Método B; Obrigações de Prova; Verificação Formal; Teoria dos Conjuntos; Satisfatibilidade Módulo Teoria.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

Capítulo 1

Introdução

Métodos formais promovem o uso de técnicas matematicamente fundamentadas a fim de prover um quadro teórico rigoroso para o desenvolvimento de *software*. Em particular, a *especificação formal* é um processo baseado em fundamentos matemáticos para descrição de sistemas e suas propriedades a fim de facilitar o entendimento do sistema e detectar, previamente, falhas, inconsistências, ambigüidades e funcionalidades incompletas. A *verificação formal* tem como papel analisar se o software satisfaz as funcionalidades previstas na especificação. O *refinamento*, por sua vez, é uma técnica de transformação de especificações que tem como objetivo introduzir decisões de projeto e tornar as especificações mais próximas a códigos implementáveis por linguagens de programação.

O foco do refinamento são as mudanças ocorridas nas especificações baseadas na noção de substituição definida pelo cálculo das substituições, mas não necessariamente de equivalência. Ou seja, o cálculo de refinamentos não tem por objetivo gerar refinamentos equivalentes às especificações, mas sim gerar refinamentos que determinam que se um *software* é desenvolvido de acordo com o refinamento de sua especificação, então ele é válido em relação a sua especificação. A validade dos refinamentos é garantida através da verificação das obrigações de prova resultantes da aplicação das substituições. O processo de verificação é realizado através de *checagem de modelos* ou de *prova de teoremas*. A primeira técnica consiste em construir um modelo finito do sistema e verificar se ele

atende a todos os requisitos determinados pela especificação do sistema. Já na prova de teoremas, tanto o sistema quanto seus requisitos são expressos por meio de lógica matemática. Esta lógica é fornecida por um sistema formal, bem como um conjunto de axiomas e de regras de inferência, que são usados para provar as propriedades do sistema.

Um processo completo e rigoroso de desenvolvimento de *software* que combina especificação, verificação, refinamento e síntese é proporcionado pelo método B [1]. Ele é fundamentado sobre a base matemática composta pela lógica de primeira-ordem, teoria dos conjuntos e aritmética de inteiros. Nesse contexto, os desenvolvedores ficam responsáveis por fornecer uma especificação inicial, tomar decisões de projeto através de refinamentos e, possivelmente, interagir com um provador de teoremas para verificar a validade das obrigações de prova geradas e, conseqüentemente, garantir a correção da implementação assim construída.

Para o processo de verificação formal, neste trabalho, são apresentados formalismos com suporte à checagem de satisfatibilidade de fórmulas da teoria axiomática dos conjuntos. Cada formalismo é implementado por uma ferramenta de prova, facilitando a aplicação do mesmo à verificação das obrigações de prova geradas pelo método B. Como exemplo, tem-se a ferramenta Prioni [39] que integra o checador de modelos Alloy Analyzer [2] ao provador de teoremas Athena. Dessa forma, a utilização do Prioni para o método B requer a tradução das especificações em linguagem B para a linguagem Alloy. No contexto dos provadores de teoremas, destacam-se: o sistema Z-Eves [33] de suporte a especificação, verificação, refinamento e síntese de especificações em notação Z, precursora da notação B; e o sistema Atelier-B/Click'n'Prove [6, 13] que constitui o atual estado da arte para as ferramentas de suporte ao método B. Assim como ocorre com o Prioni, as especificações em linguagem B devem ser traduzidas para a linguagem Z a fim de permitir a utilização da ferramenta Z/EVES. Entretanto, o Atelier-B/Click'n'Prove pode ser diretamente aplicado na verificação das especificações B.

No campo da verificação formal por prova de teoremas, a checagem de satisfatibilidade de fórmulas livres de quantificadores a partir de uma teoria decidível constitui o

problema de checagem de *Satisfatibilidade Módulo Teoria* (SMT). Como exemplo, tem-se o provador automático de teoremas haRVey [21] que implementa uma extensão da abordagem de checagem de SMT para a lógica de primeira ordem e aplica-se tanto para teoria finitas decidíveis quanto não-decidíveis. O haRVey é um provador automático de teoremas para a lógica de primeira-ordem com igualdade, combinando técnicas de dedução automática [10], como resolução e paramodulação, além de procedimentos de decisão para fragmentos da aritmética. Tem como alvo a verificação de obrigações de prova geradas no processo de desenvolvimento de *software*. Em particular, provê um mecanismo de definição axiomática da semântica das diversas construções utilizadas nas especificações de *software*, tais como vetores, listas e conjuntos. Assim, a capacidade dedutiva do haRVey pode ser estendida à teoria dos conjuntos através da definição axiomática desta teoria.

Como opções de teorias axiomáticas dos conjuntos, destacam-se a teoria infinita dos conjuntos de Zermelo-Fräenkel (ZFC), que considera todos os objetos como conjuntos, e a teoria finita dos conjuntos de von Neumann-Bernays-Gödel (NBG), cujos objetos podem ser conjuntos ou classes. A teoria NBG é equivalente à ZFC, ou seja, todo teorema provado pro ZFC é também provado por NBG. Considerando-se que a checagem de SMT adotada requer uma axiomatização finita, a inclusão da teoria NBG ao haRVey possibilita a aplicação da checagem de SMT às obrigações de prova geradas pelo método B.

1.1 Motivação

Ferramentas de suporte para o método B têm sido desenvolvidas na indústria, mas não oferecem uma arquitetura aberta, como é o caso do Atelier-B e do B-Toolkit [4]. Algumas ferramentas de domínio público já foram publicadas pela academia nos últimos anos, como por exemplo, o Click'n'Prove, mas não com o mesmo nível de maturidade que as comerciais. Além disso, de maneira geral, as ferramentas existentes requerem do usuário um alto grau de conhecimento da lógica de primeira-ordem para guiar e controlar o

processo de verificação das obrigações de prova, dificultando assim sua utilização.

Em relação às ferramentas em fase de desenvolvimento, destaca-se o projeto Batcave [5] para a verificação automática de especificações originadas pelo método B, sendo desenvolvido no laboratório ConSiste [1] da UFRN [2]. Trata-se de uma extensão do *plugin* jBTools [26] que é incorporado ao editor de texto jEdit [27]. Neste contexto, o jBTools provê disponibilidade de símbolos matemáticos, verificação de sintaxe e de tipos, animação de especificação, checagem de modelo e síntese de código em linguagem Java e C#. Em complemento, o Batcave se responsabiliza pela geração das obrigações de prova e aciona o haRVey para verificá-las. Assim, a extensão da aplicabilidade do haRVey para a teoria axiomática dos conjuntos NBG proporciona uma melhor utilização deste por parte do Batcave.

1.2 Objetivos

Para atender às necessidades apresentadas na seção 1.1, este trabalho tem por objetivo fornecer uma alternativa para a verificação das obrigações de prova obtidas de acordo com o método B, ou seja, baseadas em construções de aritmética inteira e de teoria dos conjuntos. Mais especificamente, descreve-se uma axiomatização da teoria dos conjuntos NBG orientada pelos operadores de conjuntos da linguagem B. De posse deste conjunto de axiomas, obtém-se uma extensão do provador de teoremas haRVey que pode ser diretamente aplicado à checagem de satisfatibilidade das obrigações de prova módulo teoria NBG.

Uma vez que o projeto Batcave tem como objetivo desenvolver e disponibilizar um sistema de geração e de verificação de obrigações de prova para o método B, a integração do haRVey neste sistema caracteriza-se como um segundo objetivo deste trabalho. Nesta integração, as obrigações de prova geradas passam a ser verificadas através da checagem

[1] Disponível em: <http://www.consiste.dimap.ufrn.br>. Acesso em: 27 jan 2007.
[2] Disponível em: <http://www.ufrn.br>. Acesso em: 27 jan 2007.

de satisfatibilidade módulo teoria dos conjuntos NBG realizada pelo haRVey. Com isso, obtém-se um sistema de suporte a especificação, verificação, refinamento e síntese de especificações em notação B com as vantagens de possuir uma arquitetura aberta e ser automático e gratuito.

1.3 Organização do Trabalho

O trabalho está estruturado em sete capítulos, incluindo este. O capítulo 2 apresenta sucintamente o método B. Algumas abordagens de verificação formal aplicáveis ao método B e adotadas por outros provadores são descritas no capítulo 3. O capítulo 4 apresenta a abordagem de checagem de SMT implementada pelo provador automático de teoremas haRVey. A teoria dos conjuntos NBG, passível de aplicação a checagem de SMT, é descrita no capítulo 5, bem como a teoria dos conjuntos ZFC que a originou. O capítulo 6 apresenta uma reformulação da teoria dos conjuntos NBG baseada nos operadores disponíveis na linguagem B e incorporada à checagem de SMT fornecida pelo haRVey, cujos resultados também se encontram neste capítulo. Finalmente, no capítulo 7, são feitas algumas considerações finais sobre os assuntos focalizados.

Capítulo 2

Introdução ao Método B

Desenvolvido em 1985 por um dos criadores da notação Z [35, 15], Jean-Raymond Abrial, o método B de desenvolvimento de software abrange as fases de especificação, projeto e implementação do ciclo de vida do software. Cada uma dessas fases é vista como uma atividade que envolve provas matemáticas na justificativa de seus resultados. Assim, a coleção de todas as provas garante a correção do sistema.

A base matemática do método B consiste de lógica de primeira-ordem, aritmética de inteiros e teoria dos conjuntos. As construções básicas que relacionam tais conceitos são similares às da notação Z. Entretanto, as construções de estruturação são mais claras e próximas às construções das linguagens de programação modular imperativas, a fim de facilitar o entendimento e a usabilidade fora do meio acadêmico.

De acordo com o método B, à cada fase, são produzidas especificações que descrevem os requisitos de forma abstrata, expressando que comportamento é desejado dentro do domínio completo da aplicação, ao invés de como obter esse comportamento. Então, por serem abstratas, as especificações passam por uma seqüência de etapas de refinamento até obter uma implementação concreta. Durante esse processo, obrigações de prova são produzidas; se essas obrigações de prova são válidas, então os comportamentos da especificação e do *design* não são inconsistentes. Dessa forma, uma prova valida o

comportamento num domínio completo, e não num ponto específico.

Uma especificação B estrutura-se em módulos nomeados de acordo com seus níveis de abstração: *máquina*, *refinamento* ou *implementação*, do mais abstrato ao mais concreto. O processo de desenvolvimento inicia-se com uma ou mais máquinas, onde cada máquina pode ser composta por outras máquinas. As máquinas podem ser especificadas de forma mais concreta através de refinamentos até alcançar uma implementação. Ao provar a consistência de uma máquina, cada passo de refinamento deve provar sua correção em relação à máquina correspondente. Então, como a especificação da implementação constitui o último passo de refinamento, a implementação também deve manter sua correção em relação ao refinamento correspondente. E, caso ela esteja correta, os códigos executáveis (a implementação final) devem ser gerados. As implementações podem ser auto-suficientes ou podem depender dos serviços de outras máquinas que são refinadas separadamente até suas respectivas implementações.

2.1 A Linguagem B

A linguagem B é composta por camadas sobrepostas e inter-relacionadas. A camada mais interna consiste na lógica de primeira-ordem com igualdade. Ao redor dessa camada constrói-se a teoria dos conjuntos com tipos. A noção de substituição aplicada às camadas anteriores deriva a Linguagem de Substituição Generalizada (GSL). Finalmente, tem-se a notação de pseudo-programação denominada Notação de Máquina Abstrata (AMN) que faz uso das construções da GSL.

A partir da AMN, os sistemas são modelados como coleções de Máquinas Abstratas (AM) interdependentes que podem ser vistas como objetos, ou seja, instâncias de uma classe. Cada máquina encapsula dados e operações. O conjunto dos dados, representados por variáveis, constitui o estado da máquina que pode ser alterado através das operações.

Os dados são modelados por conceitos matemáticos, tais como conjuntos, relações,

funções, seqüências e árvores, e são restringidos por um conjunto de leis estáticas definidas por condições, denominado invariante. De maneira mais formal, o invariante é uma expressão de segurança ou de integridade que determina os estados válidos da máquina. Somente as operações modificam o estado de uma máquina, mas mantendo o invariante. Elas podem ter parâmetros e retornar resultados.

As operações têm um caráter declarativo e não-algorítmico que, em geral, não permite utilizar seqüenciamento ou iteração. Elas são especificadas através de construções da AMN derivadas de substituições da GSL que estabelecem o valor dos dados e das saídas ao término da operação. Tais substituições são generalizações daquelas modeladas pelo *cálculo da pré-condição mais fraca*[1] (*wp*-cálculo) de Edsger W. Dijkstra [8] cujas construções constituem-se em transformadores de predicados. Mais especificamente, o *wp*-cálculo define regras para transformar as pós-condições nas mais fracas pré-condições que as garantam. Ou seja, as pré-condições mais fracas são determinadas em função das pós-condições. Assim, sendo **x** uma variável, **E** uma expressão e **Q** a condição, a condição denotada por **[x := E]Q** é a pré-condição mais fraca sob a qual a substituição da variável **x** pela expressão **E**, denotada por **x := E**, estabelece a pós-condição **Q**, ou, em outras palavras, *wp*(**(x := E),Q**) = **[x := E]Q**.

A Tabela 2.1 especifica formalmente as construções da AMN derivadas da GSL. Os símbolos utilizados para condições são **P**, $\mathbf{P}_1$, ..., $\mathbf{P}_n$, **Q** e **I**; **S**, $\mathbf{S}_1$, ..., $\mathbf{S}_n$ são substituições; **C**, $\mathbf{C}_1$, ..., $\mathbf{C}_n$ representam conjuntos; **E** denota uma expressão, **f** representa uma função e **x**, **y** denotam variáveis quaisquer.

Com base no *wp*-cálculo, além das substituições, cada operação pode conter uma pré-condição que especifica as condições de aplicação daquela operação (em particular, informações de tipagem dos parâmetros). A pré-condição define o domínio da aplicação, ou seja, expressa a condição indispensável sem a qual a operação não pode ser invocada. Se uma operação é invocada quando sua pré-condição não é satisfeita, então os resultados são imprevisíveis.

[1] *weakest precondition*

CONSTRUÇÃO	REGRA
[*skip*]**Q**	**Q**
[BEGIN **S** END]**Q**	[**S**]**Q**
[PRE **P** THEN **S** END]**Q**	(**P** $\wedge$ [**S**]**Q**)
[IF **P** THEN $\mathbf{S}_1$ ELSE $\mathbf{S}_2$ END]**Q**	(**P** $\Rightarrow$ [$\mathbf{S}_1$]**Q**) $\wedge$ ($\neg$ **P** $\Rightarrow$ [$\mathbf{S}_2$]**Q**)
[IF **P** THEN **S** END]**Q**	[IF **P** THEN **S** ELSE *skip* END]**Q**
[IF $\mathbf{P}_1$ THEN $\mathbf{S}_1$ ELSEIF $\mathbf{P}_2$ THEN $\mathbf{S}_2$ ELSE $\mathbf{S}_3$ END]**Q**	[IF $\mathbf{P}_1$ THEN $\mathbf{S}_1$ ELSE (IF $\mathbf{P}_2$ THEN $\mathbf{S}_2$ ELSE $\mathbf{S}_3$ END) END]**Q**
[**x** := *bool*(**P**)]**Q**	[IF **P** THEN **x** := *true* ELSE **x** := *false* END]**Q**
[ANY **x** WHERE **P** THEN **S** END]**Q**	$\forall$ **x** $\bullet$ (**P** $\Rightarrow$ [**S**]**Q**) onde **x** não é livre em **Q**.
[**x** :$\in$ **C**]**Q**	[ANY **y** WHERE **y** $\in$ **C** THEN **x** := **y** END]**Q**
[**x** :**P**]**Q**	[ANY **y** WHERE [**x** := **y**]**P** THEN **x** := **y** END]**Q**
[CHOICE $\mathbf{S}_1$ OR $\mathbf{S}_2$ END]	[$\mathbf{S}_1$]**Q** $\wedge$ [$\mathbf{S}_2$]**Q**
[SELECT $\mathbf{P}_1$ THEN $\mathbf{S}_1$ ... ... WHEN $\mathbf{P}_n$ THEN $\mathbf{S}_n$ END]**Q**	($\mathbf{P}_1$ $\Rightarrow$ [$\mathbf{S}_1$]**Q**) $\wedge$ ... $\wedge$ ($\mathbf{P}_n$ $\Rightarrow$ [$\mathbf{S}_n$]**Q**)
[CASE **E** OF EITHER $\mathbf{C}_1$ THEN $\mathbf{S}_1$ OR ... ... OR $\mathbf{C}_n$ THEN $\mathbf{S}_n$ END]**Q**	[SELECT **E** $\in$ $\mathbf{C}_1$ THEN $\mathbf{S}_1$ ... ... WHEN **E** $\in$ **C** THEN $\mathbf{S}_n$ END]**Q**
[VAR **x** IN **S** END]**Q**	$\forall$ **x** $\bullet$ ([**S**]**Q**) onde **x** não é livre em **Q**.
[LET **x** BE **P** IN **S** END]**Q**	$\forall$ **x** $\bullet$ (**P** $\Rightarrow$ [**S**]**Q**) onde **x** não é livre em **Q**.
[$\mathbf{S}_1$; $\mathbf{S}_2$]**Q**	[$\mathbf{S}_1$] [$\mathbf{S}_2$]**Q**
[**f**(**x**) := **E**]**Q**	[**f** := **f** $\oplus$ {**x** $\mapsto$ **E**}]**Q**
[WHILE **P** DO **S** VARIANT **E** INVARIANT **I** END]**Q**	(**I** $\wedge \neg$ **P** $\Rightarrow$ **Q**) $\wedge$ (**I** $\Rightarrow$ **E** $\in \mathbb{N}$) $\wedge$ (**I** $\wedge$ **P** $\Rightarrow$ [**y** := **E**] [**S**]**E** < **y**) $\wedge$ (**I** $\wedge$ **P** $\Rightarrow$ [**S**]**I**) onde **y** não é livre em **Q**.

Tabela 2.1: Regras de reescrita da AMN derivadas das construções da GSL.

A especificação das operações pode ser total ou parcial. Uma especificação total caracteriza-se possuir a pré-condição *true*, ou seja, a operação pode ser invocada em qualquer estado da máquina e para qualquer valor dos parâmetros. Quando a especificação da operação é parcial, isso significa que ela possui uma pré-condição não-trivial e, portanto, a operação é definida apenas para estados e valores abrangidos pela pré-condição.

2.2 A Notação B

Para exemplificar a utilização do método B na modelagem de um sistema, será especificado um sistema de leilão. De maneira informal, esse sistema deve manter um histórico dos objetos que estão em leilão, bem como daqueles que já foram leiloados. Além disso, deve guardar também o maior lance recebido por cada objeto. Para iniciar o leilão, coloca-se um objeto na sessão corrente, relacionando-o ao lance mínimo permitido pelo leilão. Caso um objeto ainda esteja em leilão, ou seja, ainda não tenha sido leiloado, pode-se propor um lance maior do que os já propostos.

Uma máquina é formada por várias cláusulas. Toda máquina inicia com uma cláusula `MACHINE`. Essa cláusula comporta o nome da máquina e os seus parâmetros. Os parâmetros podem ser constantes do tipo números naturais ou conjuntos finitos não-vazios. Apesar de não ser determinado pela AMN, por conveção, adota-se o uso de identificadores com letras minúsculas para representar as constantes e identificadores com letras maiúsculas para representar os conjuntos. Por exemplo, para a máquina que especifica o leilão de objetos e recebe o conjunto dos objetos como parâmetro:

```
MACHINE leilao (OBJETOS)
```

Duas partes têm destaque: a nomeação de variáveis que representam o estado da máquina e a especificação dos tipos e das restrições sobre os valores das mesmas, através das cláusulas `VARIABLES` e `INVARIANT`, respectivamente. Por exemplo, a máquina `leilao` tem seu estado definido pela relação, denominada `sessao`, entre os objetos do leilão e todos os seus respectivos lances. Como cada objeto possui um lance mínimo cujo valor é maior que 0, a imagem de `sessao` é definida por $\mathbb{N}_1$. Além disso, o estado da máquina é definido também pelo conjunto (histórico) dos objetos já leiloados, denominado `leiloados`. Isso significa dizer que `leiloados` corresponde ao conjunto dos objetos que já foram colocados à leilão, ou seja, que pertencem ao domínio de `sessao`:

```
VARIABLES sessao, leiloados
```

`INVARIANT sessao` $\in$ `OBJETOS` $\leftrightarrow \mathbb{N}_1 \wedge$ `leiloados` $\subseteq$ `dom(sessao)`

A cláusula `CONSTANTS` realiza a declaração de constantes e a cláusula `PROPERTIES` permite impor restrições sobre o valor das mesmas. Assim, para um leilão, pode-se definir um lance mínimo permitido que será utilizado mais adiante como lance inicial para cada um dos objetos do leilão:

`CONSTANTS lance_min`

`PROPERTIES lance_min` $\in \mathbb{N}_1$

A cláusula `CONSTRAINTS`, tem um papel similar para os parâmetros da máquina. Para o leilão, deve-se garantir que existe ao menos um objeto a ser leiloado. Isso é feito através da restrição da cardinalidade do conjunto dos objetos:

`CONSTRAINTS card(OBJETOS)` $\in \mathbb{N}_1$

A cláusula `SETS` permite declarar conjuntos que podem ser usados para dar um tipo aos elementos da máquina. Tais conjuntos podem ser *adiados* ou *enumerados*. Os conjuntos adiados são aqueles cujos elementos são definidos apenas na implementação final. Em contrapartida, os conjuntos enumerados são definidos através da enumeração de seus elementos. Por convenção da AMN, um conjunto enumerado só contém os elementos declarados e estes são distintos dois a dois. Assim como aqueles recebidos como parâmetro da máquina, os conjuntos declarados em `SETS` são finitos, não-vazios e, por convenção, representados por identificadores com letras maiúsculas. Por exemplo, para um leilão, deve-se disponibilizar uma consulta a algum objeto a fim de saber se ele ainda está em leilão (`aberto`), se já foi leiloado (`fechado`) ou se não faz parte do leilão (`inexistente`). Os valores de retorno dessa consulta podem ser declarados a partir de um conjunto enumerado:

`SETS RESP = {aberto, fechado, inexistente}`

A cláusula ASSERTIONS é usada para expressar lemas que podem ser deduzidos do invariante. O papel desses lemas é o de prover subsídios adicionais para provar a correção da máquina. Por exemplo, para um leilão pode-se afirmar que os objetos já leiloados fazem parte do conjutno dos objetos oferecidos no leilão:

```
ASSERTIONS leiloados ⊆ OBJETOS
```

Por fim, as cláusulas INITIALISATION e OPERATIONS são usadas para especificar o valor inicial do estado e as operações possíveis, respectivamente. Por exemplo, no começo de um leilão são apresentados os objetos a serem leiloados e, portanto, ainda não há nem objetos leiloados e nem lances propostos:

```
INITIALISATION sessao, leiloados := ∅, ∅
```

Mas ao dar início ao leilão de um objeto é necessário definir seu lance mínimo através da operação iniciar. Ela recebe o objeto como parâmetro e o relaciona ao lance mínimo. Além disso, para exemplificar uma operação com retorno, pode-se destacar a operação consultar. Essa operação retorna o maior lance já proposto a um objeto oferecido no leilão, ou seja, a um objeto que pertence ao domínio da relação sessao:

```
OPERATIONS
  iniciar(obj)≙
    PRE obj ∈ OBJETOS \ dom(sessao)
    THEN sessao := sessao ∪ {obj ↦ lance_min}
    END;
  resp, m ⟵ consultar(obj) ≙
    PRE obj ∈ dom(sessao)
    THEN m := max(sessao[{obj}]) ||
         IF obj ∉ leiloados
         THEN resp := aberto
```

```
        ELSE resp := fechado
        END
    END
END
```

De maneira geral, as cláusulas utilizadas pelas máquinas são adotadas também para os refinamentos e as implementações. Mas, assim como as máquinas iniciam-se com uma cláusula `MACHINE`, os refinamentos iniciam-se com `REFINEMENT`, comportando o nome do refinamento e os seus parâmetros que devem ser os mesmos da máquina refinada. Tal máquina é referenciada através da cláusula `REFINES`.

A implementação inicia-se com a cláusula `IMPLEMENTATION`, seguida por `REFINES`. É importante salientar que a implementação deve apresentar uma especificação bem próxima do código final, portanto, seqüenciamento e iteração são permitidos neste nível de abstração. A especificação deve estar apta a ser executada no tempo permitido e com os recursos disponíveis pela linguagem final. E para isso ser possível, algumas restrições são impostas sobre os tipos de variáveis a serem usadas e as substituições aplicadas nas definições das operações. Assim sendo, uma implementação não possui a cláusula `VARIABLES` e seus dados devem ser encapsulados em máquinas importadas através da cláusula `IMPORTS`.

Ao importar uma máquina, a implementação deve fornecer valores para suprir os parâmetros dessa máquina, respeitando as suas restrições. Uma implementação tem um invariante determinando restrições sobre os valores de suas variáveis e, também, sobre a relação destas variáveis com as da máquina que foi refinada. Além disso, uma implementação tem um estado, um valor inicial e operações. Tais operações são idênticas às da máquina refinada em termos de quantidades e de tipos das entradas e saídas.

A Figura 2.1 apresenta o diagrama de dependências entre a máquina `leilao`, o refinamento desta máquina `leilao_ref`, a implementação da mesma `leilao_imp` e a máquina importada pela implementação `leilaoVar`, fornecendo as variáveis concretas.

As máquinas são representadas por um retângulo, o refinamento é representado por um hexágono e a implementação é representada por um trapézio. As especificações de cada módulo encontram-se no Apêndice A.

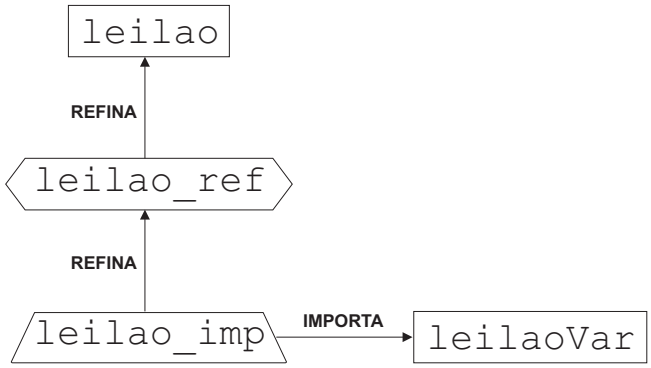

Figura 2.1: Diagrama de dependências de `leilao_imp`.

2.3 Obrigações de Prova

Cada máquina contém fundamentalmente três tipos de obrigações de prova. Essas obrigações de prova incluem sub-expressões comuns relativas aos diferentes elementos da máquina, como parâmetros, constantes, conjuntos e variáveis. A seguir tem-se a lista das sub-expressões:

- **`TIPO_PARAM`** especifica o tipo dos conjuntos parâmetros da máquina. Por convenção, são conjuntos de inteiros. Assim, se **`X`** é o único conjunto parâmetro de uma máquina, **`TIPO_PARAM`** é **`X`** $\in \mathbb{P}_1(\mathbb{Z})$;

- **`TIPO_CONJ`** especifica o tipo dos conjuntos declarados na cláusula `SETS`. Por convenção, tanto os conjuntos adiados quanto os enumerados são conjuntos de inteiros. Além disso, os conjuntos enumerados não possuem outros elementos e estes são distintos dois a dois. Assim, se **`X`** é um conjunto adiado e **`Y`** é um conjunto enumerado tal que **`Y = {a, b, c}`**, então **`TIPO_CONJ`** é:

$$\mathbf{X} \in \mathbb{P}_1(\mathbb{Z}) \land \mathbf{Y} \in \mathbb{P}_1(\mathbb{Z}) \land \mathbf{Y} = \{\mathbf{a},\mathbf{b},\mathbf{c}\} \land \mathbf{a} \neq \mathbf{b} \land \mathbf{a} \neq \mathbf{c} \land \mathbf{b} \neq \mathbf{c};$$

- **REST_PARAM** especifica as restrições adicionais sobre os parâmetros. É definido pelo usuário na cláusula `CONSTRAINTS`;
- **REST_CONST** especifica as restrições adicionais sobre as constantes e os conjuntos. É definido pelo usuário na cláusula `PROPERTIES`;
- **INV** especifica as restrições sobre as variáveis de estado. É definido pelo usuário através da cláusula `INVARIANT`.
- **ASSERT** especifica os lemas definidos na cláusula `ASSERTION`.
- **REST_TOTAL** abrevia a expressão **TIPO_PARAM** $\land$ **TIPO_CONJ** $\land$ **REST_PARAM** $\land$ **REST_CONST**, ou seja, representa todas as restrições sobre parâmetros, conjuntos declarados e constantes.

A primeira obrigação de prova visa garantir que a máquina inicia em um estado válido, satisfazendo o invariante. Se **INIT** é a substituição especificada na cláusula `INITIALISATION`, essa obrigação de prova é:

$$\mathbf{REST\_TOTAL} \Rightarrow [\mathbf{INIT}]\mathbf{INV}$$

Para a máquina `leilao`, a obrigação de prova da inicialização é dada por:

$$\overbrace{\texttt{OBJETOS} \in \mathbb{P}_1(\mathbb{Z})}^{\texttt{TIPO\_PARAM}} \land \overbrace{\texttt{card(OBJETOS)} \in \mathbb{N}_1)}^{\texttt{REST\_PARAM}} \land \overbrace{\texttt{lance\_min} \in \mathbb{N}_1}^{\texttt{REST\_CONST}} \land$$

$$\overbrace{\underbrace{\texttt{RESP} \in \mathbb{P}_1(\mathbb{Z}) \land \texttt{RESP = \{aberto, fechado, inexistente\}} \land \texttt{aberto} \neq \texttt{fechado} \land \texttt{aberto} \neq \texttt{inexistente} \land \texttt{fechado} \neq \texttt{inexistente}}}^{\texttt{TIPO\_CONJ}}$$

$$\Rightarrow \overbrace{[\texttt{sessao, leiloados :=}\varnothing,\varnothing]}^{\texttt{INIT}}\ \texttt{sessao} \in \texttt{OBJETOS} \leftrightarrow \mathbb{N}_1 \land \texttt{leiloados} \subseteq \texttt{dom(sessao)}$$

Aplicando as regras do cálculo das substituições, a obrigação de prova é:

$$\texttt{REST\_TOTAL} \Rightarrow \varnothing \in \texttt{OBJETOS} \leftrightarrow \mathbb{N}_1 \wedge \varnothing \subseteq \texttt{dom(sessao)}$$

O segundo tipo de obrigação de prova visa estabelecer os lemas especificados na cláusula `ASSERTION`:

$$\textbf{REST\_TOTAL} \wedge \textbf{INV} \Rightarrow \textbf{ASSERT}$$

Aplicando-se à máquina `leilao`, tem-se:

$$\overbrace{\texttt{OBJETOS} \in \mathbb{P}_1(\mathbb{Z})}^{\texttt{TIPO\_PARAM}} \wedge \overbrace{\texttt{card(OBJETOS)} \in \mathbb{N}_1}^{\texttt{REST\_PARAM}} \wedge \overbrace{\texttt{lance\_min} \in \mathbb{N}_1}^{\texttt{REST\_CONST}} \wedge$$

$$\overbrace{\texttt{sessao} \in \texttt{OBJETOS} \leftrightarrow \mathbb{N}_1 \wedge \texttt{leiloados} \subseteq \texttt{dom(sessao)}}^{\texttt{INV}} \wedge$$

$$\overbrace{\underbrace{\texttt{RESP} \in \mathbb{P}_1(\mathbb{Z}) \wedge \texttt{RESP = \{aberto, fechado, inexistente\}} \wedge \texttt{aberto} \neq \texttt{fechado} \wedge \texttt{aberto} \neq \texttt{inexistente} \wedge \texttt{fechado} \neq \texttt{inexistente}}}^{\texttt{TIPO\_CONJ}}$$

$$\Rightarrow \overbrace{\texttt{leiloados} \subseteq \texttt{OBJETOS}}^{\texttt{ASSERT}}$$

Já o terceiro tipo tem como papel garantir que as operações da máquina preservam o invariante. Para cada operação, seja **PRE** e **SUBS** denotando, respectivamente, a pré-condição e a substituição dessa operação, a obrigação de prova associada é:

$$\textbf{REST\_TOTAL} \wedge \textbf{INV} \wedge \textbf{ASSERT} \wedge \textbf{PRE} \Rightarrow [\textbf{SUBS}]\textbf{INV}$$

A obrigação de prova da operação `iniciar` da máquina `leilao` é dada por:

$$\overbrace{\texttt{OBJETOS} \in \mathbb{P}_1(\mathbb{Z})}^{\texttt{TIPO\_PARAM}} \wedge \overbrace{\texttt{card(OBJETOS)} \in \mathbb{N}_1}^{\texttt{REST\_PARAM}} \wedge \overbrace{\texttt{lance\_min} \in \mathbb{N}_1}^{\texttt{REST\_CONST}} \wedge$$

$$\overbrace{\texttt{sessao} \in \texttt{OBJETOS} \leftrightarrow \mathbb{N}_1 \wedge \texttt{leiloados} \subseteq \texttt{dom(sessao)}}^{\texttt{INV}} \wedge$$

$$\overbrace{\texttt{leiloados} \subseteq \texttt{OBJETOS}}^{\texttt{ASSERT}} \land \overbrace{\texttt{obj} \in \texttt{OBJETOS} \setminus \texttt{dom(sessao)}}^{\texttt{PRE}} \land$$

$$\overbrace{\begin{array}{l}\texttt{RESP} \in \mathbb{P}_1(\mathbb{Z}) \land \texttt{RESP = \{aberto, fechado, inexistente\}} \land \\ \texttt{aberto} \neq \texttt{fechado} \land \texttt{aberto} \neq \texttt{inexistente} \land \\ \texttt{fechado} \neq \texttt{inexistente}\end{array}}^{\texttt{TIPO\_CONJ}}$$

$$\Rightarrow [\texttt{sessao := sessao} \cup \texttt{\{obj} \mapsto \texttt{lance\_min\}}]$$

$$\texttt{sessao} \in \texttt{OBJETOS} \leftrightarrow \mathbb{N}_1 \land \texttt{leiloados} \subseteq \texttt{dom(sessao)}$$

Essa obrigação de prova é então equivalente à seguinte fórmula:

$$\texttt{REST\_TOTAL} \land \texttt{INV} \land \texttt{ASSERT} \land \texttt{PRE}$$

$$\Rightarrow \texttt{sessao} \cup \texttt{\{obj} \mapsto \texttt{lance\_min\}} \in \texttt{OBJETOS} \leftrightarrow \mathbb{N}_1 \land \texttt{leiloados} \subseteq \texttt{dom(sessao)}$$

É importante destacar que obrigações de prova também são associadas aos refinamentos. Para o refinamento de algoritmo, elas garantem que as pré-condições não são fortalecidas e que as pós-condições não são enfraquecidas. Para o refinamento de dados, elas estabelecem uma relação entre as variáveis abstratas e concretas e garantem a preservação do invariante módulo essa relação de abstração/concretização. Por exemplo, dado um refinamento para a máquina `leilao`

```
REFINEMENT leilao_ref (OBJETOS)
REFINES leilao
```

pode-se aplicar um refinamento de dados à variável `sessao` através da variável `rsessao`. A nova variável deixa de ser uma relação e passa a ser uma função parcial, ou seja, para cada objeto, armazena apenas o lance mais alto já proposto e não mais o conjunto de todos os lances propostos. Isso é especificado da seguinte forma:

```
VARIABLES rsessao, rleiloados
INVARIANT rsessao ∈ OBJETOS ⇸ ℕ1 ∧ dom(rsessao) = dom(sessao) ∧
```

```
    ∀x • (x ∈ dom(rsessao) ⇒ rsessao(x) = max(sessao[{x}])) ∧
    rleiloados = leiloados
```

Uma vez que não se trata de um refinamento de algoritmo, as operações do refinamento são semelhantes às da máquina, mudando apenas os nomes das variáveis abstratas para as concretas e adequando as substituições. Por exemplo, para as operações `iniciar` e `consultar`, tem-se:

```
OPERATIONS
  iniciar(obj) ≙
    PRE obj ∈ OBJETOS \ dom(rsessao)
    THEN rsessao := rsessao ⊕ {obj ↦ lance_min}
    END;

  resp, m ⟵ consultar(obj) ≙
    PRE obj ∈ dom(rsessao)
    THEN m := rsessao(obj);
         IF obj ∉ rleiloados
         THEN resp := aberto
         ELSE resp := fechado
         END
    END
END
```

Seja $\mathbf{D}_{MR}$ a conjunção de **TIPO_PARAM** $\wedge$ **TIPO_CONJ** $\wedge$ **REST_CONST** da máquina e dos refinamentos anteriores e atual, $\mathbf{C}_M$ as restrições sobre os parâmetros da máquina original, $\mathbf{INIT}_{R-1}$ a inicialização do refinamento antecessor (que pode ser uma máquina), $\mathbf{INIT}_R$ a inicialização do refinamento e $\mathbf{INV}_R$ o invariante do refinamento. A obrigação de prova da inicialização de um refinamento visa garantir que, "dadas as restrições sobre os parâmetros da máquina original e as restrições de tipo sobre os parâmetros, conjuntos e constantes da máquina original até o refinamento atual, temos que a inicialização do

refinamento estabelece que não é o caso que a inicialização do refinamento antecessor provoca a quebra do invariante do refinamento atual", ou seja:

$$\mathbf{C}_M \wedge \mathbf{D}_{MR} \Rightarrow [\mathbf{INIT}_R]\neg\,[\mathbf{INIT}_{R-1}]\neg\,\mathbf{INV}_R$$

Para o refinamento `leilao_ref`, a obrigação de prova da inicialização é dada por:

$$\overbrace{\texttt{card(OBJETOS)} \in \mathbb{N}_1}^{C_M} \wedge$$

$$\overbrace{\begin{array}{l}\texttt{OBJETOS} \in \mathbb{P}_1(\mathbb{Z}) \wedge \texttt{lance\_min} \in \mathbb{N}_1 \wedge \\ \texttt{RESP} \in \mathbb{P}_1(\mathbb{Z}) \wedge \texttt{RESP = \{aberto, fechado, inexistente\}} \wedge \\ \texttt{aberto} \neq \texttt{fechado} \wedge \texttt{aberto} \neq \texttt{inexistente} \wedge \\ \underbrace{\texttt{fechado} \neq \texttt{inexistente}}\end{array}}^{D_{MR}}$$

$$\Rightarrow [\overbrace{\texttt{rsessao, rleiloados :=}\varnothing,\varnothing}^{INIT_R}] \neg [\overbrace{\texttt{sessao, leiloados :=}\varnothing,\varnothing}^{INIT_{R-1}}]$$

$$\neg\,(\texttt{rsessao} \in \texttt{OBJETOS} \nrightarrow \mathbb{N}_1 \wedge \texttt{dom(rsessao) = dom(sessao)} \wedge$$

$$\forall \texttt{x} \bullet (\texttt{x} \in \texttt{dom(rsessao)}$$

$$\Rightarrow \texttt{rsessao(x) = max(sessao[\{x\}])} \wedge$$

$$\texttt{rleiloados = leiloados)}$$

Essa obrigação de prova pode ser simplificada em:

$$C_M \wedge D_{MR} \Rightarrow \varnothing \in \texttt{OBJETOS} \nrightarrow \mathbb{N}_1 \wedge \texttt{dom}(\varnothing) = \texttt{dom}(\varnothing) \wedge$$

$$\forall \texttt{x} \bullet (\texttt{x} \in \texttt{dom}(\varnothing) \Rightarrow \varnothing(\texttt{x}) = \texttt{max}(\varnothing[\{\texttt{x}\}]))$$

Para cada operação sem saída de um refinamento, seja $\mathbf{D}_{MR}$ a conjunção das restrições **TIPO_PARAM** $\wedge$ **TIPO_CONJ** $\wedge$ **REST_CONST** da máquina e dos refinamentos anteriores e atual, $\mathbf{C}_M$ as restrições sobre os parâmetros da máquina original, $\mathbf{INV}_{MR}$ a conjunção do invariante da máquina e o invariante de cada refinamento anterior e atual, $\mathbf{INV}_R$ o invariante do refinamento, $\mathbf{PRE}_{MR-1}$ a conjunção da pré-condição da máquina e a pré-condição de cada refinamento anterior, $\mathbf{PRE}_R$ a pré-condição da operação do refinamento, $\mathbf{SUBS}_{R-1}$ a substituição da operação do refinamento antecessor e $\mathbf{SUBS}_R$ a substituição da operação do refinamento atual, a obrigação de prova da operação é dada por:

$$\mathbf{C}_M \ \wedge \ \mathbf{D}_{MR} \ \wedge \ \mathbf{INV}_{MR} \ \wedge \ \mathbf{PRE}_{MR-1} \ \Rightarrow \ \mathbf{PRE}_R \ \wedge \ [\mathbf{SUBS}_R] \neg \, [\mathbf{SUBS}_{R-1}] \neg \, \mathbf{INV}_R$$

Isso quer dizer que, “dadas as restrições sobre os parâmetros da máquina original, as restrições de tipo sobre os parâmetros, conjuntos e constantes, e também o invariante, desde a máquina original até o refinamento atual e a pré-condição da máquina original até a do refinamento antecessor, deve-se provar que a pré-condição da operação do refinamento atual é satisfeita e que esta operação estabelece que não é o caso que a operação do refinamento antecessor invalida o invariante do refinamento atual”. Ou seja, deve-se provar que as pré-condições da máquina e dos refianmentos anteriores não são fortalecidas no refinamento atual e que a pós-condição do refinamento anterior não é enfraquecida. Assim, o refinamento da operação `iniciar` é dado por:

$$\overbrace{\texttt{card(OBJETOS)} \in \mathbb{N}_1}^{\mathsf{C}_M} \ \wedge \quad \overbrace{\texttt{obj} \in \texttt{OBJETOS} \setminus \texttt{dom(sessao)}}^{\mathsf{PRE}_{MR-1}} \ \wedge$$

$$\overbrace{\begin{array}{l} \texttt{OBJETOS} \in \mathbb{P}_1(\mathbb{Z}) \wedge \texttt{lance\_min} \in \mathbb{N}_1 \ \wedge \\ \texttt{RESP} \in \mathbb{P}_1(\mathbb{Z}) \wedge \ \texttt{RESP = \{aberto, fechado, inexistente\}} \ \wedge \\ \texttt{aberto} \neq \texttt{fechado} \ \wedge \ \texttt{aberto} \neq \texttt{inexistente} \ \wedge \\ \underbrace{\texttt{fechado} \neq \texttt{inexistente}}_{} \end{array}}^{\mathsf{D}_{MR}} \ \wedge$$

$$\overbrace{\begin{array}{l} \texttt{sessao} \in \texttt{OBJETOS} \leftrightarrow \mathbb{N}_1 \ \wedge \ \texttt{leiloados} \subseteq \texttt{dom(sessao)} \ \wedge \\ \texttt{rsessao} \in \texttt{OBJETOS} \nrightarrow \mathbb{N}_1 \wedge \ \texttt{dom(rsessao) = dom(sessao)} \ \wedge \\ \forall \texttt{x} \bullet (\texttt{x} \in \texttt{dom(rsessao)} \Rightarrow \texttt{rsessao(x) = max(sessao[\{x\}])}) \ \wedge \\ \underbrace{\texttt{rleiloados = leiloados}}_{} \end{array}}^{\mathsf{INV}_{MR}}$$

$$\Rightarrow \overbrace{\texttt{obj} \in \texttt{OBJETOS} \setminus \texttt{dom(rsessao)}}^{\mathsf{PRE}_R} \ \wedge$$

$$\begin{array}{l} \quad \texttt{[rsessao := rsessao} \oplus \texttt{\{obj} \mapsto \texttt{lance\_min\}]} \\ \quad \neg \, \texttt{[sessao := sessao} \cup \texttt{obj} \mapsto \texttt{lance\_min]} \\ \quad\quad \neg \, (\texttt{rsessao} \in \texttt{OBJETOS} \nrightarrow \mathbb{N}_1 \wedge \ \texttt{dom(rsessao) = dom(sessao)} \ \wedge \\ \quad\quad\quad \forall \texttt{x} \bullet (\texttt{x} \in \texttt{dom(rsessao)} \Rightarrow \texttt{rsessao(x) = max(sessao[\{x\}])}) \ \wedge \\ \quad\quad\quad \texttt{rleiloados = leiloados)} \end{array}$$

Aplicando as regras do cálculo das substituições, a obrigação de prova é:

$$
\begin{array}{l}
\mathtt{C}_M \ \wedge\ \mathtt{D}_{MR} \ \wedge\ \mathtt{INV}_{MR} \ \wedge\ \mathtt{PRE}_{MR-1} \\
\Rightarrow\ \mathtt{PRE}_R \ \wedge\ \mathtt{rsessao} \oplus \{\mathtt{obj} \mapsto \mathtt{lance\_min}\} \in \mathtt{OBJETOS} \nrightarrow \mathbb{N}_1 \wedge \\
\quad \mathtt{dom(rsessao} \oplus \{\mathtt{obj} \mapsto \mathtt{lance\_min}\}) = \\
\quad \mathtt{dom(sessao} \cup \mathtt{obj} \mapsto \mathtt{lance\_min}) \ \wedge \\
\quad \forall \mathtt{x} \bullet (\mathtt{x} \in \mathtt{dom(rsessao} \oplus \{\mathtt{obj} \mapsto \mathtt{lance\_min}\}) \\
\qquad \Rightarrow (\mathtt{rsessao} \oplus \{\mathtt{obj} \mapsto \mathtt{lance\_min}\})(\mathtt{x}) \\
\qquad\quad = \mathtt{max((sessao} \cup \{\mathtt{obj} \mapsto \mathtt{lance\_min}\})[\{\mathtt{x}\}])
\end{array}
$$

Já para cada operação com saída de um refinamento, sendo a saída denotada por *y*, a obrigação de prova correspondente é:

$$
\begin{array}{c}
\mathbf{C}_M \ \wedge\ \mathbf{D}_{MR} \ \wedge\ \mathbf{INV}_{MR} \ \wedge\ \mathbf{PRE}_{MR-1} \\
\Rightarrow\ \mathbf{PRE'}_R \ \wedge\ [\mathbf{SUBS}_R] \neg\, [\mathbf{SUBS}_{R-1}] \neg\, (\mathbf{INV}_R \ \wedge\ (\mathbf{y'} = \mathbf{y}))
\end{array}
$$

onde $\mathbf{PRE'}_R$ é o mesmo que $\mathbf{PRE}_R$, mas com a saída renomeada para $\mathbf{y'}$. Ou seja, "tendo as restrições sobre os parâmetros da máquina original, as restrições de tipo sobre os parâmetros, conjuntos e constantes, e também o invariante, desde a máquina original até o refinamento atual e a pré-condição da máquina original até a do refinamento antecessor, deve-se provar que a pré-condição da operação do refinamento atual é satisfeita e que, substituindo cada uma de suas variáveis de saída por novas variáveis, esta operação estabelece que não é o caso que a operação do refinamento antecessor provoca a quebra do invariante do refinamento atual e a desigualdade entre as variáveis de saída correspondentes". Ou seja, deve-se provar que as pré-condições da máquina e dos refianmentos anteriores não são fortalecidas no refinamento atual e que a pós-condição do refinamento anterior não é enfraquecida, além de provar a correção do refinamento em relação às variáveis de saída da operação. Por exemplo, para o refinamento da operação `iniciar`, tem-se:

$$
\overbrace{\mathtt{card(OBJETOS)} \in \mathbb{N}_1}^{\mathtt{C}_M} \ \wedge \quad \overbrace{\mathtt{obj} \in \mathtt{dom(sessao)}}^{\mathtt{PRE}_{MR-1}} \ \wedge
$$

$$\overbrace{\underbrace{\texttt{OBJETOS} \in \mathbb{P}_1(\mathbb{Z}) \wedge \texttt{lance\_min} \in \mathbb{N}_1 \wedge \texttt{RESP} \in \mathbb{P}_1(\mathbb{Z}) \wedge \texttt{RESP} = \{\texttt{aberto}, \texttt{fechado}, \texttt{inexistente}\} \wedge \texttt{aberto} \neq \texttt{fechado} \wedge \texttt{aberto} \neq \texttt{inexistente} \wedge \texttt{fechado} \neq \texttt{inexistente}}}^{\texttt{D}_{MR}} \wedge$$

$$\overbrace{\underbrace{\texttt{sessao} \in \texttt{OBJETOS} \leftrightarrow \mathbb{N}_1 \wedge \texttt{leiloados} \subseteq \texttt{dom(sessao)} \wedge \texttt{rsessao} \in \texttt{OBJETOS} \nrightarrow \mathbb{N}_1 \wedge \texttt{dom(rsessao)} = \texttt{dom(sessao)} \wedge \forall \texttt{x} \bullet (\texttt{x} \in \texttt{dom(rsessao)} \Rightarrow \texttt{rsessao(x)} = \texttt{max(sessao[\{x\}])}) \wedge \texttt{rleiloados} = \texttt{leiloados}}}^{\texttt{INV}_{MR}}$$

$$\Rightarrow \texttt{PRE'}_R \wedge$$
$$[\texttt{m'} := \texttt{rsessao(obj)};$$
$$\texttt{IF obj} \notin \texttt{rleiloados THEN resp'} := \texttt{aberto}$$
$$\texttt{ELSE resp'} := \texttt{fechado END}]$$
$$\neg [\texttt{m} := \texttt{max(sessao[\{obj\}])};$$
$$\texttt{IF obj} \notin \texttt{leiloados THEN resp} := \texttt{aberto}$$
$$\texttt{ELSE resp} := \texttt{fechado END}]$$
$$\neg (\texttt{rsessao} \in \texttt{OBJETOS} \nrightarrow \mathbb{N}_1 \wedge \texttt{dom(rsessao)} = \texttt{dom(sessao)} \wedge$$
$$\forall \texttt{x} \bullet (\texttt{x} \in \texttt{dom(rsessao)} \Rightarrow \texttt{rsessao(x)} = \texttt{max(sessao[\{x\}])}) \wedge$$
$$\texttt{rleiloados} = \texttt{leiloados} \wedge (\texttt{m'} = \texttt{m} \wedge \texttt{resp'} = \texttt{resp}))$$

Essa obrigação de prova pode ser simplificada em:

$$\texttt{C}_M \wedge \texttt{D}_{MR} \wedge \texttt{INV}_{MR} \wedge \texttt{PRE}_{MR-1}$$
$$\Rightarrow \texttt{PRE'}_R \wedge$$
$$(\texttt{obj} \notin \texttt{rleiloados} \Rightarrow \texttt{rsessao(obj)} = \texttt{max(sessao[\{obj\}])}) \wedge$$
$$(\texttt{obj} \in \texttt{rleiloados} \Rightarrow \texttt{rsessao(obj)} = \texttt{max(sessao[\{obj\}]))}$$

Conforme dito anteriormente, assim como um refinamento, uma implementação tem a função de refinar. Portanto, as regras para verificação de inicialização e de operações com/sem saída de uma implementação com relação a um refinamento (ou a uma máquina)

são análogas àquelas de um refinamento com relação a uma máquina. Todas as obrigações de prova da máquina, do refinamento e da implementação do leilão estão descritas no Apêndice A.

A distância entre uma máquina e sua respectiva implementação pode ser muito grande, tornando cansativo o processo de geração e prova das obrigações de prova. Para intermediar esse processo, adota-se a criação de refinamentos sucessivos, onde se diz que a implementação refina o refinamento que, por sua vez, refina a máquina. Assim, provar a correção da implementação em relação ao refinamento e deste em relação à máquina corresponde a provar a correção da implementação diretamente em relação à máquina. E para sistemas mais complexos, pode-se utilizar vários refinamentos.

De posse das obrigações de prova geradas através do método B, o próximo passo a ser executado é a prova das mesmas. Para isso, deve-se utilizar ferramentas de apoio à verificação formal que implementem o formalismo do método B ou outros formalismos que possam ser adaptados a ele. Algumas das abordagens de verificação formal possíveis de serem aplicadas ao método B estão descritas no Capítulo 3.

Capítulo 3

Abordagens de Verificação Formal

Neste capítulo são apresentadas algumas abordagens de verificação formal aplicáveis ao método B de maneira direta, quando a linguagem B é a adotada, ou indireta, através de traduções simples entre linguagens. Isso só é possivel porque todas elas descrevem formalismos com suporte à checagem de satisfatibilidade de fórmulas da teoria axiomática dos conjuntos, que é um dos fundamentos da linguagem B. A teoria axiomática dos conjuntos é uma reformulação da teoria dos conjuntos na lógica de primeira-ordem.

A primeira abordagem descrita é aquela implementada pela ferramenta Prioni. Ela se destaca por integrar o checador de modelos Alloy Analyzer ao provador de teoremas Athena. Em seguida, tem-se a abordagem do provador de teoremas Z/Eves, que consiste em traduzir especificações em linguagem Z para fórmulas em lógica de primeira-ordem e prová-las, baseando-se no cálculo de predicados de primeira-ordem. De maneira similar, a abordagem implementada pelo provador de teoremas Atelier-B/Click'n'Prove, destinada à verificação de especificações B, também tem destaque. Uma característica em comum entre todas essas abordagens é que elas já foram implementadas por ferramentas de prova, facilitando sua aplicação ao método B. Ou seja, a prova de uma especificação B pode ser obtida através da tradução desta especificação em linguagem Alloy e posterior aplicação da ferramenta Prioni, em linguagem Z, seguida da aplicação de Z/Eves, ou, ainda, diretamente através da aplicação do Atelier-B/Click'n'Prove.

Existem, ainda, outras abordagens de verificação formal que também se baseiam na teoria axiomática dos conjuntos. Um exemplo é a linguagem 2LSC (*two-level syllogistic with cardinality*) descrita em [45]. Ela combina conjuntos com cardinalidade, ou seja, relaciona a aritmética à teoria dos conjuntos, e provê um procedimento de decisão para checar a satisfatibilidade de fórmulas 2LSC livre de quantificadores dentro de uma teoria infinita. Entretanto, devido à ausência de uma implementação e à alta complexidade computacional desse procedimento, a linguagem 2LSC não será considerada neste trabalho.

3.1 Prioni: Integração entre Checagem de Modelos e Prova de Teoremas

O artigo "Integrating Model Checking and Theorem Proving for Relational Reasoning" [39] apresenta uma técnica de checagem de satisfatibilidade de fórmulas de modelos infinitos através de modelos finitos. Assim, é identificada a classe de fórmulas da lógica relacional para as quais resolução sobre estruturas infinitas pode ser reduzida à resolução sobre estruturas finitas. Para tal, o artigo descreve a ferramenta Prioni que integra as duas abordagens de verificação formal: checagem de modelo e prova de teoremas através da resolução relacional. A entrada do Prioni são especificações escritas em uma linguagem declarativa de primeira-ordem baseada em relações (não há distinção entre relação e função) denominada *Alloy* [24, 25]. A Figura 3.1 apresenta a especificação de uma função recursiva que retorna o conjunto de todos os elementos de uma lista na linguagem Alloy.

A especificação inicia-se com a palavra `module` seguida pelo seu nome. A palavra `sig` introduz uma assinatura, ou seja, um conjunto de átomos, e cada uma delas possui declarações de campos (atributos) que introduzem relações. Tais campos são, por padrão, funções totais. Funções parciais e relações são diferenciadas através do uso dos modificadores `option` e `set`, respectivamente. A palavra `fun` introduz uma função que denota uma relação entre seus argumentos e o resultado e pode ser chamada em qualquer parte

de especificação. Para determinar que a função é a atual (corrente), utiliza-se a palavra `det`. A função `elms` tem um argumento `n` que é um subconjunto unitário de `No`. Para declarar um subconjunto genérico utiliza-se a palavra `set`. A variável `result` do corpo da função refere-se ao resultado da mesma. O operador "." representa composição relacional, onde `n.prox` representa o conjunto imagem de `n` mapeado pela função parcial `prox`. O operador pré-fixado "`*`" denota o fechamento transitivo-reflexivo, assim, as expressões `n.*prox` e `n.*prox.dado` denotam, respectivamente, o conjunto de todos os nós alcancáveis por `n` e o conjunto dos objetos desses nós. A palavra `assertion` introduz uma asserção, ou seja, uma afirmação lógica, a ser checada. O comando `check` checa a asserção dentro do escopo dado que, neste caso, são todas as listas com, no máximo, cinco nós.

As especificações Alloy são analisadas automaticamente pelo *Alloy Analyzer* (AA). O AA checa a validade das fórmulas Alloy para um determinado escopo finito, mas não para todos os escopos possíveis. O usuário deve informar o escopo que limita o domínio e o AA automaticamente traduz especificações Alloy para fórmulas de satisfatibilidade booleanas e utiliza provadores genéricos de satisfatibilidade (*off-the-shelf SAT solvers*) para encontrar interpretações que satisfazem as fórmulas. A checagem realizada pelo AA ocorre através de refutação, assim, se AA não encontra um contra-exemplo dentro do escopo informado, não há garantias de que o contra-exemplo não existe em um escopo maior. Então, para obter maior confiabilidade, o usuário deve re-executar o AA dentro

```
module Lista
sig Objeto {}
sig No {
  prox:  option No
  dado:  Objeto }
det fun elms(n:  No):  set Objeto {
  if (no n.prox) then result = n.dado
  else result = n.dado + elms(n.prox) }
assert Equivalencia {all n:  No | elms(n) = n.*prox.dado }
check Equivalencia for 5
```

Figura 3.1: Especificação de um leitor de listas em Alloy.

de um escopo maior, sucessivamente, enquanto o AA realiza a checagem em um tempo aceitável.

Para obter a prova completa de uma especificação, o Prioni integra o AA com o provador de teoremas *Athena* que adota uma linguagem denotacional de prova tipo-ω para lógica de primeira-ordem polimórfica com múltiplos tipos. Um sistema polimórfico de múltiplos tipos é aquele que permite que um objeto pertença a mais de um tipo [31]. Uma *linguagem denotacional de prova tipo-ω* (em inglês, *type-ω DPL*) é uma linguagem para apresentação e busca de provas que oferece fortes garantias de validade. As *type-ω DPLs* apresentam provas de uma maneira detalhada, em termos de regras de inferência expressas como *métodos* definidos pelo usuário. Tais métodos são "receitas de prova" que recebem argumentos e executam dinamicamente as deduções apropriadas [3].

Athena é um provador que oferece um alto grau de automação através do uso de métodos semelhantes às táticas dos provadores HOL [19] e Isabelle [32]. Ele adota a resolução através de dedução natural, facilitando a compreensão das provas e utiliza-se do provador Otter [40] para busca de provas. Além disso, oferece fortes garantias de confiabilidade.

A fim de possibilitar o uso efetivo do Athena na prova de especificações Alloy, o Prioni age em duas fases distintas. Inicialmente ele provê uma axiomatização da álgebra relacional em Athena, bem como uma biblioteca de lemas essenciais a este cálculo. A álgebra relacional é o conjunto de relações fechado sob operadores, ou seja, a aplicação dos operadores a uma ou mais relações produz uma nova relação. A axiomatização resultante em Athena representa relações como conjuntos de tuplas em uma teoria de conjuntos com tipos, finita e de primeira-ordem. As tuplas de pares ordenados são representadas através de uma estrutura polimórfica. Os conjuntos são polimórficos e seus tipos são determinados por um construtor de domínio. A igualdade de conjuntos e os demais operadores de conjuntos e de relação são definidos conforme a teoria dos conjuntos de primeira-ordem e, por não haver um operador de composição genérico (para relações cuja aridade seja maior que um), operadores de composição devem ser criados para cada tipo necessário de

composição de relações. Após a axiomatização, o Prioni provê uma tradução automática da especificação de Alloy para Athena, preservando o significado original da especificação, mas sem garantias de corretitude. A tradução da especificação da Figura 3.1 para a linguagem Athena é dada na Figura 3.2, onde o código Alloy é colocado em comentário através do simbolo "# ".

Na tradução para Athena, cada assinatura Alloy introduz um domínio como tipo utilizado através do construtor de domínio `(domain (Set-Of S))`. A construção `(Pair-Of S T)` representa tuplas de relações binárias do tipo `S` $\leftrightarrow$ `T`. Além disso, cada assinatura ou campo introduz um conjunto constante de tuplas, indicado por `declare`, cujos elementos são extraídos do domínio Athena apropriado. As restrições sobre as relações são declaradas globalmente através de asserções utilizando a palavra reservada `assert`, onde `is-fun` e `is-total-fun` denotam função (parcial) e função total, respectivamente. Cada "função" Alloy (declarada através da palavra chave `fun` introduz um símbolo de função Athena que deve ter seu tipo declarado seguido pela sua definição dada por `define`. O operador `(singleton ?x)` indica que `x` é um conjunto unitário. Os operadores `empty-def` e `subset-def` são definidos conforme a lógica de primeira-ordem. Os operadores de composição `comp-n-m` compõem relações de tipos $S_1 \times ... \times S_n$ e $T_1 \times ... \times T_m$, com $S_n = T_1$. Por fim, as asserções Alloy são traduzidas em obrigações de prova, onde o operador de fechamento transitivo-reflexivo é dado por `rtc`.

No exemplo, a função `elms` recebe um conjunto unitário `n` de elementos do tipo `No` como parâmetro e retorna um conjunto `result` de elementos do tipo `Objeto`, ou seja, retorna o conjunto composto pelo valor do nó relacionado à `n`. Trata-se de uma função recursiva cujo caso base ocorre quando `(empty? (comp-1-2 ?n prox))`, ou seja, quando a lista só possui um nó, então o resultado é o conjunto unitário composto pelo valor deste nó. Caso contrário, o resultado é o conjunto formado pela união do conjunto unitário composto pelo valor do nó atual e o conjunto resultante da aplicação da função `elms` ao próximo nó da lista.

Em Alloy, todos os valores são relações, não havendo distinção entre elementos e con-

juntos unitários. Entretanto, no exemplo de especificação resultante da tradução Alloy para Athena é necessário que haja essa distinção. Mas, especificá-la em linguagem Alloy pode ser um processo muito complexo. Para facilitar esse processo, o Prioni permite que a especificação Athena seja intercalada com expressões e fórmulas escritas (entre aspas

```
# sig Objeto {}
(domain Objeto-Dom)
(declare Objeto (Set-Of Objeto-Dom))

# sig No {
(domain No-Dom)
(declare No (Set-Of No-Dom))

# prox:  option No
(declare prox (Set-Of (Pair-Of No-Dom No-Dom)))

# dado:  Objeto }
(declare dado (Set-Of (Pair-Of No-Dom Objeto-Dom)))

(assert (is-fun prox))
(assert (is-total-fun dado))

# det fun elms(n:  No):  set Objeto {
(declare elms (-> (Set-Of No-Dom) (Set-Of Objeto-Dom)))
(define elms-def
  (forall ?n ?result
    (iff (= (elms ?n) ?result)
         (and (and (singleton ?n) (subset ?n No))

                   # if (no n.prox) then result = n.dado
              (and (if (empty?  (comp-1-2 ?n prox))
                     (= ?result (comp-1-2 ?n dado)))

                   # else result = n.dado + elms(n.prox) }
                   (if (not (empty?  (comp-1-2 ?n prox)))
                      (= ?result
                         (union (comp-1-2 ?n dado)
                                (elms (comp-1-2 ?n prox))))))))))
(assert elms-def)

# assert Equivalencia {all n:  No | elms(n) = n.*prox.dado }
(define Equivalencia
  (forall ?n (= (elms (singleton ?n))
                (comp-1-2 (comp-1-2 (singleton ?n) (rtc prox))
                          dado))))
```

Figura 3.2: Tradução da especificação Alloy para Athena.

duplas) em uma notação pré-fixada semelhante a Alloy, mas com suporte a tipos distintos como elementos e conjuntos unitários. Assim, as provas fornecidas pelo Prioni tornam-se mais compreensíveis para os usuários da linguagem Alloy do que aquelas que utilizam expressões Athena. Por exemplo, a asserção `Equivalencia` da Figura 3.2 estabelece uma igualdade entre conjuntos e para ser provada, deve-se verificar que `elms` é uma função válida e completa. Utilizando a notação "falso-Alloy", que contém operadores para a relação de pertinência (`in`) e de subconjunto (`subset`) e um operador de fechamento reflexivo-transitivo '`*`" pós-fixado, a asserção seria escrita através das expressões `ALL n | elms({n}) subset {n}.prox*.dado` e `ALL n | {n}.prox*.dado subset elms({n})`, respectivamente. Para o desenvolvimento da prova, o Prioni disponibiliza uma biblioteca de lemas comuns ao cálculo relacional e à teoria dos conjuntos escritos em "falso-Alloy".

Para o processo de prova da validade e da completude de `elms`, é possível introduzir regras de inferência atráves de expressões denominadas *métodos primitivos*. Mas nenhum mecanismo interno permite provar a validade desse métodos, assim, deve-se usá-los com moderação. Um exemplo de método primitivo pode ser a implementação do princípio da indução indicado em [39]. A prova por dedução é feita pelo provador Otter no prazo máximo de um minuto e falha caso P não seja deduzido a partir de $P_1, ..., P_n$ ou não seja uma consequência trivial dessa premissas. Caso contrário, a prova é instantaneamente encontrada e P é retornado com sucesso.

A fim de aplicar o Prioni à verificação de especificações B, é importante salientar que a linguagem Alloy trabalha com relações como conjuntos de tuplas e permite todas as operações padrões para manipulação de conjuntos, tais como união, interseção e diferença. Alloy também permite quantificação sobre as relações e quanto às constantes e funções usuais não suportadas diretamente, como $\varnothing$, *dom* e *range*, elas podem ser construídas através de restrições sobre as variáveis de relação. Dessa maneira, a tradução de uma especificação B para Alloy pode ser feita de maneira quase direta, adaptando os operadores não providos através de restrições e construções equivalentes, conforme descrito

em [16]. Em geral, uma especificação Alloy é dividida em várias seções. Uma delas destina-se a declaração e restrições de variáveis, que podem ser átomos, subconjuntos de domínios declarados ou relações. Uma outra seção pode ser utilizada para o invariante, restringindo os possíveis estados. Além disso, operações para modificar as variáveis são escritas em outra seção, onde se descreve a relação entre os estados das variáveis antes e depois de executada a operação. Todas essa facilidades permitem estender a aplicação da abordagem de verificação do Prioni a especificações B de maneira simples.

3.2 O Sistema Z/EVES

Originalmente, o sistema EVES [36, 41] foi desenvolvido para dar suporte à uma linguagem própria de especificação denominada Verdi, baseada no cálculo de predicados ordinários e na teoria dos conjuntos ZF de Zermelo-Fräenkel [46]. A combinação da capacidade de prova do sistema EVES com a notação Z resultou no sistema Z/EVES. O Z/EVES realiza a tradução da notação Z, definida pelo manual [35], para o cálculo de predicados de primeira-ordem suportado pelo EVES. Por ser reversível, essa tradução permite que o usuário só necessite de conhecimentos da notação Z.

A linguagem Z, fundamenta-se no cálculo de predicados de primeira-ordem e na teoria dos conjuntos ZF. Ela oferece o tipo básico dado pelo conjunto dos números inteiros $\mathbb{Z}$, a partir do qual pode-se definir novos tipos. Um tipo é um conjunto maximal de valores possíveis para variáveis deste tipo, ou seja um tipo é o maior conjunto S de valores para x tal que $x \in S$. Assim, é possível declarar variáveis de um único tipo T, representando um objeto que possui um dos valores definidos por T. Por exemplo, para especificar em Z o leilão descrito no Capítulo 2, deve-se introduzir o conjunto de todos os objetos do leilão e o conjunto de todas as mensagens de saída como um novo tipo básico e o lance inicial

comum a todos os objetos como uma constante tipada, ambos com escopo global:

$$[OBJETOS]$$

$$| \; lance\_min : \mathbb{N}_1$$

$$RESP ::= aberto \mid fechado \mid inexistente$$

Na notação Z, uma especificação é decomposta em módulos denominados *esquemas*. Eles são utilizados para descrever aspectos estáticos, como estados e invariantes que os restrigem, e dinâmicos, como as operações possíveis, a relação entre as entradas e as saídas das operações, as mudanças de estado que ocorrem e a relação entre os diferentes estados. Esquemas podem ser usados para estruturar e combinar estas descrições, agrupando partes da especificação, encapsulando-as e nomeando-as para reuso. As variáveis e o invariante da especificação são declarados dentro de um esquema que restringe o espaço de estado válido, portanto, possuem um escopo local. A parte superior (parte declarativa) de um esquema destina-se à declaração das variáveis tipadas, enquanto que a parte inferior (parte assertiva) destina-se à declaração do invariante ou do efeito da operação. O leilão tem seu estado definido pela relação entre os objetos do leilão e todos os seus respectivos lances, além do conjunto dos objetos já leiloados:

leilao

$$sessao : OBJETOS \leftrightarrow \mathbb{N}_1$$

$$leiloados : \mathbb{P}\ OBJETOS$$

$$leiloados \subseteq \operatorname{dom} sessao$$

Para esquemas que declaram mudanças de estado, na parte declarativa, faz-se referência ao nome do esquema que contém as variáveis de estado com o símbolo $'$ após o nome, para diferenciar os estados antes e após a operação. No início do leilão ainda não há lances propostos, portanto $sessao'$ é vazio e, consequentemente, $leiloados'$ também é.

Inicializacao

$leilao'$

$sessao' = \varnothing$

$leiloados' = \varnothing$

Para dar início ao leilão de um determinado objeto é necessário indicar seu lance mínimo:

OpIniciar

$\Delta leilao$

$obj? : OBJETOS$

$obj? \notin \mathrm{dom}\, sessao$

$sessao' = sessao \cup \{(obj? \mapsto lance\_min)\}$

$leiloados' = leiloados$

A declaração de $\Delta leilao$ abrevia a declaração $leilao, leilao'$ e indica que o esquema está descrevendo uma mudança de estado, ou seja, está introduzindo quatro novas variáveis que satisfazem o invariante implicitamente: *sessao*, *leiloados*, $sessao'$ e $leiloados'$. As duas primeiras variáveis indicam o estado do sistema antes da mudança, enquanto que as duas últimas indicam o estado do sistema após a mudança. O operador "?" indica uma variável (parâmetro) de entrada. Na parte assertiva do esquema tem-se a pré-condição da operação: o objeto a ser leiloado não pode já ter sido leiloado. Se a pré-condição é satisfeita, então a segunda linha é executada, adicionando o par $(obj?, lance\_min)$ a *sessao*. Caso contrário, os resultados são imprevisíveis.

Um exemplo de operação com saída é a operação *Consultar*:

Consultar

$\Xi leilao$

$obj? : OBJETOS$

$m! : \mathbb{N}_1$

$resp! : RESP$

$obj? \in \operatorname{dom} sessao$

$m! = max(sessao(\!| \{obj?\} |\!))$

if $obj? \notin leiloados$ **then** $resp! = aberto$ **else** $resp! = fechado$

A declaração de $\Xi leilao$ abrevia $leilao, leilao'$, com $sessao' = sessao$ e $leiloados' = leiloados$, e indica que o esquema não descreve mudanças de estado. Por convenção, o operador "!" indica uma variável de saída (retorno).

Uma prova no Z/EVES caracteriza-se por uma seqüência de passos de traduções da fórmula-alvo atual em outra fórmula logicamente equivalente, até que o predicado gerado seja *true*. Para o processo de prova, o Z/EVES disponibiliza comandos que variam dos automáticos até os direcionados pelo usuário. Além disso, há também uma variedade de formas de controlar as capacidades automáticas.

Por exemplo, o Z/EVES pode ser aplicado à especificação do leilão das seguintes formas:

1. checagem sintática e de tipos: é feita de forma direta e não necessita de interação com o usuário. O Z/EVES detecta e reporta os erros, permitindo corrigí-los imediatamente. Assim, se o invariante do esquema *leilao* fosse definido por $leiloados \subseteq \operatorname{ran} sessao$, o Z/EVES informaria que o tipo de $\operatorname{ran} sessao$ não é o mesmo da imagem da relação $\subseteq$;

2. checagem de domínio: o sistema deriva uma *condição de domínio* para cada parágrafo de uma especificação, garantindo que todas as suas expressões têm significado definido, ou seja, são bem formadas. Uma expressão não tem significado caso

aplique-se uma função parcial a um elemento que não se encontra no domínio da função (por exemplo, $\#\mathbb{Z}$) ou usando uma descrição definida de maneira inapropriada (por exemplo, $\mu\, x : \mathbb{Z} \bullet x \neq x$). Assim, se a operação *Consultar* possuísse apenas a pré-condição *true*, então o Z/EVES derivaria $obj? \in \mathrm{dom}\ sessao$ como condição de domínio a fim evitar a aplicação da função *sessao* a algum elemento $obj?$ que não pertencesse ao seu domínio;

3. checagem de consistência: para provar que uma especificação é consistente, deve-se provar que ela possui algum modelo que a satisfaça. A checagem deve ser feita para a inicialização do sistema e para as operações do mesmo. Para garantir a satisfatibilidade da inicialização, adiciona-se o teorema

   theorem InicializacaoEhConsistente

   $\exists\, leilao' \bullet Inicializacao$

   que descreve que deve haver um estado inicial que atenda ao invariante de estado e à inicialização do sistema.

   Para provar que cada operação respeita o invariante de estado do sistema, adicionam-se teoremas que estabeleçam que se o estado anterior à operação satisfaz o invariante de estado de sistema e a operação pode ser aplicada, então o estado após a operação também deve satisfazer o invariante. A consistência da operação *OpIniciar* é checada a partir do teorema

   theorem OpIniciarEhConsistente

   $\forall\, leilao \bullet \mathrm{pre}\ OpIniciar \Rightarrow (\exists\, leilao' \bullet OpIniciar \wedge leilao')$

4. expansão de esquema: para descrições muito compactas providas pelo cálculo de esquemas (*schema calculus*), o Z/EVES permite fazer expansão de esquemas e simplificação de resultados, facilitando a leitura e a utilização do provador através da tradução da especificação em predicados. Para expandir um esquema deve-se colocá-la como um alvo a ser provado. Então, para a expansão do esquema

Consultar pode-se adicionar o teorema

theorem ExpandirOpIniciar

OpIniciar

e aplicar o comando `prove by reduce`, que origina o predicado

$$
\begin{aligned}
&\mathit{sessao} \in \mathit{OBJETOS} \leftrightarrow \mathbb{N}_1 \\
&\land\ \mathit{leiloados} \in \mathbb{P}\,\mathit{OBJETOS} \\
&\land\ \mathit{leiloados} \in \mathbb{P}\,\mathrm{dom}\,\mathit{sessao} \\
&\land\ \mathit{obj}? \in \mathit{OBJETOS} \\
&\land\ \mathit{sessao}' = \mathit{sessao} \cup \{(\mathit{obj}?, \mathit{lance\_min})\} \\
&\land\ \mathit{leiloados}' = \mathit{leiloados} \\
&\land\ \neg\ \mathit{obj}? \in \mathrm{dom}\,\mathit{sessao}
\end{aligned}
$$

cujos esquemas referenciados são expandidos e simplificados;

5. cálculo de pré-condições: o sistema pode ser utilizado para calcular a pré-condição completa de um esquema. A exploração da pré-condição de um esquema consiste em testá-lo para todos os estados e com todas as entradas do domínio. Aplicando-se o cálculo de pré-condições obtém-se a pré-condição original adicionada de afirmações lógicas de que a entrada e o estado inicial são adequados. Para uma operação total (com pré-condição *true*) como a *Inicializacao*, o teorema

   theorem inicializacaoEhTotal

   $\forall\ \mathit{leilao} \bullet \mathrm{pre}\ \mathit{Inicializacao}$

   é facilmente provado. Em contrapartida, para explorar a pré-condição da operação *OpIniciar* deve-se introduzir o teorema

   theorem OpIniciar_Precondicao

   $\forall\ \mathit{leilao};\ \mathit{obj}? : \mathit{OBJETOS} \bullet \mathrm{pre}\ \mathit{OpIniciar}$

e aplicar o comando `prove by reduce`, que origina o predicado

$$
\begin{aligned}
&\mathit{sessao} \in \mathit{OBJETOS} \leftrightarrow \mathbb{N}_1 \\
&\land\ \mathit{leiloados} \in \mathbb{P}\,\mathit{OBJETOS} \\
&\land\ \mathit{leiloados} \in \mathbb{P}\,\mathrm{dom}\,\mathit{sessao} \\
&\land\ \mathit{obj}? \in \mathit{OBJETOS} \\
&\Rightarrow \neg\ \mathit{obj}? \in \mathrm{dom}\,\mathit{sessao}
\end{aligned}
$$

atestando que o valor que a pré-condição da operação é $\neg\ \mathit{obj}? \in \mathrm{dom}\,\mathit{sessao}$, ou seja, $\mathit{obj}? \notin \mathrm{dom}\,\mathit{sessao}$;

6. prova geral de teoremas: o sistema permite declarar e provar "teoremas de desafio" (*challenge theorems*), que são fatos que deveriam ser conseqüências da especificação, mas não são explicitados. Após a operação *OpIniciar*, o *lance_min* deve ser registrado como um lance dado ao objeto *obj*?. Isso pode ser checado através da prova do teorema

   theorem OpIniciar_Desafio

   $$\forall\ \mathit{OpIniciar};\ \mathit{obj}? : \mathit{OBJETOS} \bullet \mathit{sessao}'\,\mathit{obj}? = \mathit{lance\_min}$$

Após validar uma especificação, é possível refiná-la dentro desta mesma especificação a fim de torná-la mais concreta. A notação Z estabelece três tipos de obrigações de prova para validar refinamentos. As obrigações devem ser introduzidas como teoremas e provadas. Sejam **`Abs`** um sistema abstrato, $\mathbf{INIT}_{Abs}$ e $\mathbf{OP}_{Abs}$ seus esquemas de inicialização e de operação, respectivamente. Sejam, também, **`Conc`** um sistema concreto que refina **`Abs`**, $\mathbf{INIT}_{Conc}$ e $\mathbf{OP}_{Conc}$ seus esquemas de inicialização e de operação, respectivamente. Adotando-se o seguinte refinamento para a especificação do leilão, com esquema *leilao*, inicialização *Inicializacao* e operação *OpIniciar*:

leilao_ref

$$rsessao : OBJETOS \pfun \mathbb{N}_1$$

$$rleiloados : \mathbb{P}\ OBJETOS$$

$$rleiloados \subseteq \mathrm{dom}\ rsessao$$

Inicializacao_ref

$$leilao\_ref'$$

$$rsessao' = \varnothing$$

$$rleiloados' = \varnothing$$

OpIniciar_ref

$$\Delta leilao\_ref$$

$$obj? : OBJETOS$$

$$obj? \notin \mathrm{dom}\ rsessao$$

$$rsessao' = rsessao \oplus \{(obj? \mapsto lance\_min)\}$$

$$rleiloados' = rleiloados$$

e, tendo a relação **R**:

leilaoAbstracao

leilao

leilao_ref

$\mathrm{dom}\, sessao = \mathrm{dom}\, rsessao$

$leiloados = rleiloados$

$\forall x : OBJETOS$

$\bullet x \in \mathrm{dom}\, sessao \wedge sessao(\!|\ \{x\}\ |\!) \in \mathrm{dom}\, max \Rightarrow rsessaox = \max(sessao(\!|\ \{x\}\ |\!))$

a primeira obrigação de prova visa garantir que existe uma relação **R** entre os estados iniciais abstrato e concreto:

$$\forall \mathbf{Conc'} \bullet \mathbf{INIT}_{Conc} \Rightarrow (\exists \mathbf{Abs'} \bullet \mathbf{INIT}_{Abs} \wedge \mathbf{R'})$$

Então, a obrigação de prova da inicialização é dada por:

theorem InitRefinamento

$\forall\, leilao\_ref' \bullet Inicializacao\_ref \Rightarrow (\exists\, leilao' \bullet Inicializacao \wedge leilaoAbstracao')$

O segundo tipo de prova visa estabelecer que se uma operação abstrata pode ser aplicada, a sua operação concreta correspondente também pode:

$$\forall \mathbf{Abs;Conc} \bullet \mathrm{pre}\ \mathbf{OP}_{Abs} \wedge \mathbf{R} \Rightarrow \mathrm{pre}\ \mathbf{OP}_{Conc}$$

Aplicando ao leilão, tem-se:

theorem OpIniciarAbs1

$\forall\, leilao;\ leilao\_ref \bullet \mathrm{pre}\ OpIniciar \wedge leilaoAbstracao \Rightarrow \mathrm{pre}\ OpIniciar\_ref$

Já o terceiro tipo estabelece que se a pré-condição da operação abstrata é satisfeita então esta operação produz um resultado consistente com a operação concreta correspondente:

$$\forall \mathbf{Abs;\ Conc;\ Conc'} \bullet \mathrm{pre}\ \ \mathbf{OP}_{Abs} \land \mathbf{R} \land \mathbf{OP}_{Conc} \Rightarrow (\exists\ \mathbf{Abs'} \bullet \mathbf{OP}_{Abs} \land \mathbf{R'})$$

Para o leilão, essa obrigação corresponde a:

theorem OpIniciarAbs2

$\forall\, leilao;\ leilao\_ref;\ leilao\_ref'$

$\bullet\, \mathrm{pre}\ OpIniciar \land leilaoAbstracao \land OpIniciar\_ref$

$\Rightarrow (\exists\, leilao' \bullet OpIniciar \land leilaoAbstracao')$

Todas as operações abstratas do sistema devem ser verificadas em relação as suas correspondentes concretas. A verificação completa do sistema é obtida através da prova de todos os teoremas envolvidos. E sucessivos refinamentos podem ser aplicados até que se obtenha um modelo pronto para ser implementado.

Conforme dito no capítulo 2, a notação Z originou a notação B. Ambas as notações baseiam-se na lógica de primeira-ordem, aritmética de inteiros e teoria dos conjuntos. Além disso, ambas geram obrigações de prova necessárias para validar um sistema à cada passo de refinamento. A principal diferença entre as notações consiste na forma de estruturação, pois, enquanto a notação B adota o paradigma imperativo de programação, a notação Z adota a divisão em esquemas para estrutura e combinar as descrições dos objetos e suas propriedades. Assim, para converter uma especificação B para Z, deve-se destacar os conjuntos, tipos e constantes e declará-los globalmente. Em seguida, as variáveis e o invariante são declarados e definidos em um esquema principal e outros esquemas são criados para cada operação, separadamente, includindo a operação de inicialização. Portanto, o sistema Z/EVES pode ser adotado para a verificação de especificações B traduzidas em Z e a análise de seu funcionamento tem fundamental importância para o desenvolvimento de ferramentas de suporte ao método B.

3.3 Atelier-B e Click'n'Prove

O Atelier B é composto por um conjunto integrado de ferramentas que possibilitam o desenvolvimento de aplicações usando o método B. Ele auxilia os desenvolvedores na formalização de suas aplicações, executando automaticamente, tanto nas especificações quanto nos refinamentos e implementações, análises sintáticas, checagem de tipos, demonstração de obrigações de prova e geração de código-fonte. Além disso, alguns componentes extras, tais como tradutores para linguagens de programação (C, C++ ou ADA), editores matemáticos, bibliotecas de máquinas pré-definidas e várias ferramentas de análise de projeto são suportados. Assim, o Atelier B representa o estado da arte em verificação de especificações B.

O B4Free [7] é a versão gratuita (apenas para o meio acadêmico) do Atelier B e pode ser acessado por linha de comando ou através da integração da interface gráfica Click'n'Prove, que se executa sobre o XEmacs [44]. Também conhecido como "La Balbulette", o Click'n'Prove foi desenvolvido com o objetivo de facilitar o processo de prova interativa por parte do usuário, eliminando a necessidade de profundos conhecimentos matemáticos. Em contrapartida, o provador interativo do Click'n'Prove é limitado e não possui todas as funcionalidades presentes no provador interativo do Atelier-B, tais como a possibilidade de:

- definir novas regras de inferência, facilitando a prova de construções aritméticas cujo suporte é deficiente; e
- criar *scripts* de prova, permitindo a reaplicação de provas desenvolvidas pelo usuário a outras obrigações de prova.

O Click'n'Prove consiste em uma reestruturação dos aspectos externos do Atelier B e é composto pelos provadores **pb** ("provador do Atelier-B") e **pp** ("provador de predicados"). O provador **pb** foi desenvolvido exclusivamente para o Atelier-B e trabalha diretamente com a teoria dos conjuntos. Ele constitui-se de um grande número de regras

de inferência para geração de expressões matemáticas que utilizem o operador de pertinência "$\in$". O provador **pp** foi desenvolvido inicialmente de forma independente e logo depois incorporado ao Atelier-B para validar as regras de **pb**. Trata-se de um provador para o cálculo de predicados de primeira-ordem com igualdade (e um pouco de aritmética) e também de um tradutor da teoria dos conjuntos para esse cálculo.

O **pp** utiliza-se de um procedimento de decisão para o cálculo proposicional semelhante ao método Tableaux [22]. Dado que um *sequente* é um par ordenado $\langle \Gamma, \theta \rangle$, de listas finitas de fórmulas, onde uma das fórmulas de θ pode ser deduzida de alguma das hipóteses de Γ, o método Tableaux consiste na aplicação de regras simples que transformam gradualmente um sequente sem hipóteses em um ou mais sequentes com hipóteses formadas apenas por literais. Tais sequentes são gerados até que haja uma proposição atômica e sua correspondente negação entre as hipóteses, conforme mostra a Figura 3.3. Para o cálculo de predicados de primeira-ordem, o **pp** utiliza-se de um procedimento de

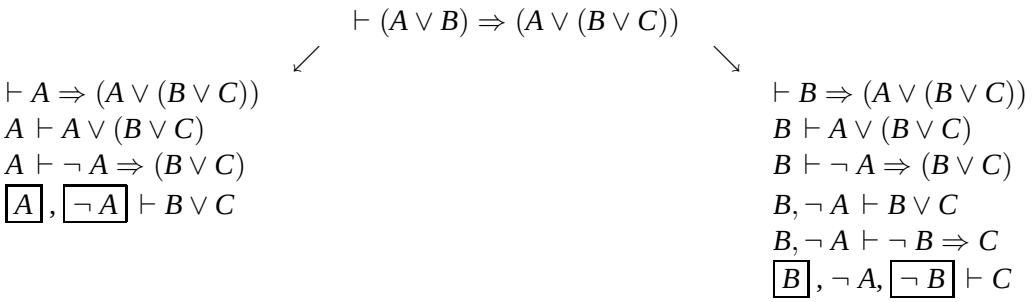

Figura 3.3: Exemplo de prova de uma proposição através do provador **pp** .

semi-decisão composto por duas fases. A primeira fase consiste em aplicar regras sobre um sequente que pode ser composto não somente por proposições atômicas como também por predicados universalmente quantificados, que são normalizados posteriormente. Isso ocorre até que a conclusão seja reduzida a FALSE e todos os literais contraditórios entre as hipóteses tenham sido gerados. A segunda fase consiste em buscar por contradições. Tais contradições resultam da instanciação daquelas hipóteses (do sequente) que são universalmente quantificadas. As instâncias são descobertas de forma sistemática por meio das hipóteses atômicas. Essa técnica é um caso especial da técnica do Otter [40] deno-

minada "conjunto de apoio" (*set of support*), onde o conjunto em questão é representado pelas suas hipóteses atômicas. As instâncias obtidas são movidas para o lado direito do sequente, que, atualmente, implica em FALSE. Após completar a segunda fase, o provador reinicia a primeira fase e assim sucessivamente. Um exemplo dessa técnica iterativa de prova é dado na Figura 3.4, onde na Fase 2 ocorre a instanciação das variáveis universalmente quantificadas x, y e z com x, y e y na hipótese $\forall(x,y,z) \bullet \neg\,(R_{xy} \wedge \neg\, R_{zx})$ resultante da Fase 1.

$\vdash \forall x \bullet ((\exists y \bullet R_{xy}) \Rightarrow \forall z \bullet R_{zx}) \Rightarrow \forall(x,y) \bullet (R_{xy} \Rightarrow R_{yx})$
FASE 1:
$\quad \hookrightarrow \forall(x,y,z) \bullet \neg\,(R_{xy} \wedge \neg\, R_{zx}), R_{xy}, \neg\, R_{yx} \vdash \mathsf{FALSE}$
$\quad$ FASE 2:
$\qquad \hookrightarrow \forall(x,y,z) \bullet \neg\,(R_{xy} \wedge \neg\, R_{zx}), R_{xy}, \neg\, R_{yx} \vdash \neg\,(R_{xy} \wedge \neg\, R_{yx}) \Rightarrow \mathsf{FALSE}$
$\qquad$ FASE 1:
$\qquad\quad \hookrightarrow \forall(x,y,z) \bullet \neg\,(R_{xy} \wedge \neg\, R_{zx}), \boxed{R_{xy}}, \neg\, R_{yx}, \boxed{\neg\, R_{xy}} \vdash \mathsf{FALSE}$
$\qquad\quad \hookrightarrow \forall(x,y,z) \bullet \neg\,(R_{xy} \wedge \neg\, R_{zx}), R_{xy}, \boxed{\neg\, R_{yx}}, \boxed{R_{yx}} \vdash \mathsf{FALSE}$

Figura 3.4: Exemplo de prova de um predicado de primeira-ordem através do provador **pp** .

Na tradução da teoria dos conjuntos para o cálculo de predicados de primeira-ordem com igualdade, o **pp** segue as construções determinadas no B-Book [1]. As expressões envolvendo os operadores sobre conjuntos, relações e funções são reduzidas ao máximo através de regras de reescrita, eliminando operadores complexos e produzindo fórmulas em lógica de primeira-ordem com os operadores de pertinência, construções de pares e de conjunto unitário, conjunto vazio e aritmética, conforme mostra a Figura 3.5.

Apesar de provar automaticamente expressões não-triviais, o **pp** apresenta quatro tipos de deficiências: *sensibilidade a hipóteses desnecessárias*, ou seja, pode falhar ou gastar muito tempo para provar uma expressão prefixada com muitas hipóteses não-utilizadas; *dificuldade em abstrair as expressões a serem provadas*, ou seja, pode gastar muito tempo para notar que uma certa expressão é um caso especial de outra mais simples; *dificuldade para gerenciar igualdades*, ou seja, pode falhar por não notar que duas expressões são iguais ou dois predicados são equivalentes; e *dificuldade para detectar estratégia de prova*, ou seja, por se tratar de um provador automático, o **pp** nem sempre detecta qual

$$f \in s \nrightarrow t \ \wedge \ a \subseteq t \ \wedge \ b \subseteq t \ \Rightarrow \ f^{-1}[a \cap b] \ = \ f^{-1}[a] \cap f^{-1}[b]$$

TRADUÇÃO:
$\forall(x,y) \bullet ((x,y) \in f \Rightarrow x \in s \wedge y \in t)$
$\wedge \ \forall(x,y,z) \bullet ((x,y) \in f \wedge (x,z) \in f \Rightarrow y = z)$
$\wedge \ \forall x \bullet (x \in a \Rightarrow x \in t)$
$\wedge \ \forall x \bullet (x \in b \Rightarrow x \in t)$
$\Rightarrow$
$\forall x \bullet (\exists y \bullet (x \in a \wedge x \in b \wedge (x,y) \in f) \Rightarrow \exists y \bullet (y \in a \wedge (x,y) \in f) \wedge \exists y \bullet (y \in b \wedge (x,y) \in f)) \wedge$
$\forall x \bullet (\exists y \bullet (y \in a \wedge (x,y) \in f) \wedge \exists y \bullet (y \in b \wedge (x,y) \in f) \Rightarrow \exists y \bullet (x \in a \wedge x \in b \wedge (x,y) \in f))$

Figura 3.5: Tradução de teoria dos conjuntos para predicados de primeira-ordem através do **pp** .

estratégia deve ser aplicada a uma expressão.

Para suprir as deficiências do **pp** o Click'n'Prove oferece uma interface proativa onde o usuário pode optar pelo modo automático ou interativo de provas. No primeiro caso, a árvore de prova é inteiramente construída de forma automática. No modo interativo o usuário deve decidir que regras aplicar à cada passo. Quando a árvore de provas é muito grande, seja em largura ou em profundidade, ela sofre uma simplificação onde somente a parte mais significativa dos sequentes é guardada. Essa nova árvore é apresentada na *janela de árvore* e o sequente corrente é mostrado por completo na *janela de sequente*.

A janela de sequente apresenta tanto as hipóteses quanto a conclusão do sequente em questão. E, em geral, um sequente apresenta muitas hipóteses. Assim, o sistema reestrutura as hipóteses através de uma hierarquia de maneira análoga à hierarquia usual de memórias. As hipóteses são classificadas nas seguintes áreas: *escondidas*, que não são visíveis na janelas; *localizadas*, que se tornam visíveis após busca em *escondidas*; *armazenadas*, que são visíveis mas não fazem mais parte da conclusão; e *selecionadas*, que são visíveis e são parte da conclusão. Com essa estruturação, os sequentes podem ser deslocados de uma área para outra através de botões de comandos disponíveis na janela de sequente. Além desses botões, muitos outros podem ser aplicados durante o processo de prova, conforme explicados em [13].

Aliando as técnicas automática e interativa de prova, o Click'n'Prove aumenta a sua

capacidade dedutiva dentro da lógica de primeira-ordem. As deficiências do provador **pp** são compensadas pela interface que disponibiliza os botões de comandos que são aplicáveis à prova. E, diferentemente dos demais provadores interativos, o Click'n'Prove apresenta uma interface dinâmica, guiada pelo processo de prova, onde, à cada passo desse processo, uma nova configuração de botões é apresentada, facilitando a interação do usuário para a prova de especificações B.

3.4 Síntese

Neste capítulo, foram apresentadas alternativas para o processo de verificação das obrigações de prova do método B. Para tal, destacamos três formalismos de suporte à checagem de satisfatibilidade módulo teoria dos conjuntos, bem como as ferramentas que implementam cada formalismo.

A partir da tradução de uma especificação B para a linguagem Alloy, é possível utilizar a ferramenta Prioni. Trata-se de uma ferramenta destinada à checagem de satisfatibilidade de fórmulas de modelos infinitos através de modelos finitos, onde estes últimos são verificados pelo checador de modelos Alloy Analyzer. Após a verificação dos modelos escritos em linguagem Alloy, o Prioni efetua a tradução dos mesmos para a linguagem Athena, linguagem de entrada do provador de teoremas de mesmo nome. Além disso, ele provê todos os lemas necessários para a prova, bem como a axiomatização do cálculo relacional, que constitui a base da linguagem Alloy. Em seguida, o provador Athena é acionado para efetuar a prova por dedução natural. No entanto, uma vez que a tradução de Alloy para Athena não possui garantias de corretitude, a prova final também não as possuirá. Além disso, apesar das publicações referentes à ferramenta Prioni, tanto Prioni quanto Athena não estão disponíveis para os usuários.

Considerada uma das principais ferramentas de suporte à notação Z, o Z/EVES constitui uma outra alternativa para a verificação de especificações em notação B, sendo esta

originária de Z. Traduzindo especificações B para Z, o Z/EVES pode ser aplicado para: (i) checagem sintática e checagem de tipos; (ii) checagem de domínio, originando uma condição de domínio para cada parágrafo; (iii) checagem de consistência, verificando se existe um modelo que satisfaz a especificação dada; (iv) expansão de esquema, expandindo o invariante; (v) cálculo de pré-condições; e (vi) prova geral de teoremas, provando lemas originados do invariante. Porém, apesar de disponibilizar uma vasta gama de opções de controle de prova, algumas provas requerem um alto grau de interatividade exigindo um bom nível de conhecimento da lógica de primeira ordem, por parte do usuário. A dificuldade de aplicação da ferramenta agrava-se pelo fato da mesma possuir uma arquitetura fechada e não disponibilizar um suporte ao usuário.

Uma vez que a aplicação do Prioni e do Z/EVES sobre as especificações B requer uma tradução de B para suas respectivas linguagens de entrada, é importante salientar que a especificação resultante deve preservar todas as características operacionais da especificação original. Portanto, a tradução pode tornar-se uma tarefa com alto custo de desenvolvimento. Assim, a aplicação direta da ferramenta Atelier-B ou do Click'n'Prove aparenta ser a melhor opção para o método B. O Atelier-B integra ferramentas para: (i) análise sintática e análise de tipos; (ii) demonstração e prova das obrigações de prova; e (iii) geração de código-fonte. Ele possui ainda sua versão B4Free, gratuita para o meio acadêmico, cuja interface gráfica é o Click'n'Prove constituído pelos provadores **pb** e **pp** . O **pp** apresenta uma série de deficiências em relação a prova de expressões não-triviais, enumeradas na seção 3.3, que, para serem supridas, requerem um bom nível de conhecimento da lógica de primeira ordem, assim como ocorre com Z/EVES.

Para diminuir a necessidade de recorrer a provadores interativos, faz-se necessária a possibilidade de aplicar técnicas de verificação automática para o método B como, por exemplo, a técnica de checagem de *Satisfatibilidade Módulo Teoria* (SMT) [12] apresentada no capítulo 4.

Capítulo 4

Satisfatibilidade Módulo Teoria

Assim como as ferramentas de verificação automática apresentadas no capítulo 3, muitas outras abordagens de verificação de *software* baseiam-se na geração e na prova de obrigações de prova. Isto quer dizer que, após gerar fórmulas da lógica de primeira-ordem, é necessário checar a satisfatibilidade destas fórmulas em uma dada teoria. Em particular, quando tais fórmulas são livres de quantificadores e a teoria é decidível, diz-se que é necessário checar sua *Satisfatibilidade Módulo Teoria* (SMT).

Este capítulo apresenta uma abordagem de checagem de SMT implementada pelo provador automático de teoremas haRVey. O haRVey recebe como entrada a axiomatização finita $Ax(\mathcal{A}_s^e)$ da teoria dos vetores $\mathcal{A}_s^e$ e uma fórmula de primeira-ordem a ser provada e realiza a prova através de superposição. Esse processo de checagem de SMT corresponde a uma das fases da técnica de prova para especificações baseadas em conjuntos descrita na seção 4.2.

4.1 Prova de Satisfatibilidade Módulo Teoria

A abordagem proposta no artigo “Satisfiability solving for software verification” [18] para checagem de satisfatibilidade de ϕ módulo $\mathcal{T}$ realiza-se através do cálculo de super-

posição. Esse artigo descreve um algoritmo correto e completo baseado no cálculo de superposição que é adotado por vários sistemas. O *cálculo da superposição* é definido como "um conjunto de regras que são derivadas da resolução e que são especialmente designadas para o tratamento de igualdade". O algoritmo é dado na Figura 4.1, cujas funções são:

*fol*2*prop***:** mapeia uma conjunção ϕ de literais básicos de primeira-ordem para uma fórmula proposicional ϕ^p;

*prop*2*fol***:** mapeia uma fórmula proposicional β^p para uma conjunção β de literais básicos de primeira-ordem;

*selec_atrib_total***:** retorna uma atribuição total para as variáveis proposicionais de ϕ^p;

$\mathcal{T} - satisfativel$**:** detecta se β é satisfatível em $\mathcal{T}$. Se sim, então retorna $(\texttt{sat}, \varnothing)$, senão retorna $(\texttt{insat}, \pi)$, onde π denota uma conjunção de literais básicos de primeira-ordem satisfatível em $\mathcal{T}$ e $\pi \subseteq \beta$.

```
function Bool+T(φ : formula livre de quantificadores)
1   φ^p ⟵ fol2prop(φ);
2   while Bool−satisfativel(φ^p) do
3     β^p ⟵ selec_atrib_total(φ^p);
4     (ρ, π) ⟵ T − satisfativel(prop2fol(β^p));
5     if ρ == sat then return sat
6     φ^p ⟵ φ^p ∧ ¬ fol2prop(π);
7   end while;
8   return insat;
end
```

Figura 4.1: Algoritmo simplificado de SMT

Em resumo, dada uma conjunção ϕ de literais básicos de primeira-ordem, ela é convertida em uma fórmula proposicional ϕ^p e suas atribuições verdadeiras são enumeradas e checadas em $\mathcal{T}$. O procedimento de satisfatibilidade em $\mathcal{T}$ retorna `sat`, caso seja encontrado um modelo, ou `insat`, caso contrário. Assim, seja $Ax(\mathcal{T})$ uma axiomatização finita de $\mathcal{T}$ composta por cláusulas equacionais e seja β uma conjunção de literais básicos. O procedimento de satisfatibilidade em $\mathcal{T}$ ocorre em duas fases:

FASE 1. ***Nivelamento de literais básicos*:** β' é o resultado do nivelamento de β obtido através da atribuição de uma nova constante à cada literal não-constante de β, de maneira que um literal nivelado seja uma igualdade da forma $f(c_1,\ldots,c_n) = c_{n+1}$ ou a negação de uma igualdade da forma $c_1 \neq c_2$, onde $c_1,\ldots,c_{n+1}$ são constantes e f é um símbolo de função n-ário.

FASE 2. ***Aplicação de superposição*:** regras do cálculo de superposição definidas em [9] são aplicadas exaustivamente às cláusulas de $Ax(\mathcal{T}) \wedge \beta'$. Alguns exemplos de regras são:

Superposição	*Paramodulação*	*Reflexão*
$\underline{l' = r' \;\; l = r}$	$\underline{l' = r' \;\; l \neq r}$	$\underline{s \neq t}$
$([r']l = r)\sigma$	$([r']l \neq r)\sigma$	$\bot$

onde $\bot$ denota a cláusula vazia, σ é o unificador mais genérico entre l' e um subtermo de l e os termos s e t são unificáveis. Um conjunto é dito *unificável* se possui um unificador. Uma substituição é denominada *unificador* de um conjunto de expressões se, e somente se, a instância de cada expressão, obtida a partir desta substituição, é idêntica às instâncias das outras expressões do conjunto. Um unificador σ de um conjunto de expressões é um unificador mais genérico se, e somente se, para cada unificador θ deste conjunto, há uma substituição λ tal que σ é a composição de θ e λ. A composição de θ e λ, onde $\theta = \{t_1/x_1,...,t_n/x_n\}$ e $\lambda = \{u_1/y_1,...,u_m/y_m\}$, é representada por $\theta \circ \lambda$ e é obtida do conjunto $\{t_1\,\lambda/x_1,...,t_n\,\lambda/x_n, u_1/y_1,...,u_m/y_m\}$, eliminando-se qualquer elemento $t_j\,\lambda/x_j$ tal que $t_j\,\lambda = x_j$ e qualquer elemento u_i/y_i tal que $y_i \in \{x_1,x_2,...,x_n\}$.

Por exemplo, pode-se adotar a teoria dos vetores $\mathcal{A}$, cuja assinatura é composta pelos símbolos de função binária `rd` (*leitura*), de função ternária `wr` (*escrita*) e um conjunto finito de símbolos de constantes (escrito em letras maiúsculas). Seja a axiomatização

$Ax(\mathcal{A})$ dada por:

$$\forall A, I, E \bullet (\mathtt{rd}(\mathtt{wr}(A, I, E), I) = E), \tag{4.1}$$

$$\forall A, I, J, E \bullet (I \neq J \Rightarrow \mathtt{rd}(\mathtt{wr}(A, I, E), J) = \mathtt{rd}(A, J)), \tag{4.2}$$

Seja β a conjunção de literais básicos

$$a = \mathtt{wr}(\mathtt{wr}(a, i, \mathtt{rd}(a, j)), j, \mathtt{rd}(a, i)) \wedge \mathtt{rd}(a, i) \neq \mathtt{rd}(a, j).$$

A FASE 1 gera o nivelamento β' composto pelos seguintes literais básicos nivelados:

$$c_1 = \mathtt{rd}(a, i) \tag{4.3}$$

$$c_2 = \mathtt{rd}(a, j) \tag{4.4}$$

$$c_3 = \mathtt{wr}(a, i, c_2) \tag{4.5}$$

$$c_4 = \mathtt{wr}(c_3, j, c_1) \tag{4.6}$$

$$a = c_4 \tag{4.7}$$

$$c_1 \neq c_2 \tag{4.8}$$

Na FASE 2, as regras do cálculo da superposição são aplicadas a $Ax(\mathcal{A})$, provando que β é $\mathcal{A}$-insatisfatível:

$c_1 = \mathtt{rd}(c_4, j)$	*Superposição* entre (4.1) e (4.6), com $\sigma = \{c_3/A, j/I, c_1/E\}$
$c_1 = \mathtt{rd}(a, j)$	*Reescrita* de $c_1 = \mathtt{rd}(c_4, j)$ por (4.7)
$c_1 = c_2$	*Superposição* entre $c_1 = \mathtt{rd}(a, j)$ e (4.4), com $\sigma = \varnothing$
$c_1 \neq c_1$	*Paramodulação* entre $c_1 = c_2$ e (4.8), com $\sigma = \varnothing$
$\bot$	*Reflexão* em $c_1 \neq c_1$

A abordagem de satisfatibilidade baseada no cálculo da superposição é *flexível*, uma vez que, para obter um novo procedimento de decisão para uma teoria basta adicionar os literais a serem provados e a axiomatização desta teoria. Em contrapartida, o cálculo da superposição possui limitações quando aplicado à aritmética. Para suprir tais limitações,

adota-se o esquema de combinação de Nelson e Oppen (N&O) [14] que combina prova de teoremas por saturação e raciocínio aritmético.

Uma teoria é *solidamente infinita* (em inglês, *stably infinite*) se qualquer fórmula satisfatível livre de quantificadores é satisfatível em um modelo de cardinalidade infinita. Dadas duas teorias $\mathcal{T}_1$ e $\mathcal{T}_2$ solidamente infinitas, cujas assinaturas são disjuntas, e seus respectivos procedimentos de satisfatibilidade, o esquema de combinação N &O permite resolver problemas de checagem de satisfatibilidade de uma conjunção Φ de literais básicos pertencentes a $\mathcal{T}_1 \cup \mathcal{T}_2$. Nele, os literais de Φ são *purificados*, ou seja, seus subtermos são nomeados por novas constantes, originando uma conjunção $\Phi_1 \wedge \Phi_2$ equisatisfatível à Φ, onde Φ_i contém apenas literais com símbolos de $\mathcal{T}_i$, para $i = 1, 2$. Dizer que α_1 é *equisatisfatível* à α_2 significa dizer que α_1 é satisfatível se, e somente se α_2 é satisfatível.

O esquema N&O consiste na troca de cláusulas elementares entre os procedimentos de cada teoria, onde uma *cláusula elementar* é aquela cujos literais são igualdades entre constantes e estas ocorrem em literais purificados pertencentes a ambas as teorias. Por exemplo, considera-se a teoria $\mathcal{T}$ obtida pela união da aritmética linear sobre os racionais LA($\mathcal{R}$) e da teoria dos vetores $\mathcal{A}$ com seus respectivos procedimentos de decisão. Dada a conjunção de literais purificados:

$$\mathtt{rd}(b, i) = e_2 \wedge \mathtt{wr}(a, i, e_1) \quad = \quad b \quad \wedge \tag{4.9}$$

$$e_1 + e_2 \quad \neq \quad 2e_2 \tag{4.10}$$

o literal (4.9) é enviado para o procedimento de decisão (por superposição) de $\mathcal{A}$, resultando em:

$\mathtt{rd}(b, i) = e_1$	*Superposição* entre (4.1) e $\mathtt{wr}(a, i, e_1) = b$, com $\sigma = \{a/A, i/I, e_1/E\}$
$e_1 = e_2$	*Reescrita* de $\mathtt{rd}(b, i) = e_1$ por $\mathtt{rd}(b, i) = e_2$

e a cláusula elementar derivada $e_1 = e_2$ é enviada junto com (4.10) para o procedimento de decisão de LA($\mathcal{R}$) que resulta em insatisfatibilidade. Então, N&O retorna a insatisfatibilidade da conjunção de (4.9) e (4.10).

A abordagem de checagem de SMT pode ser estendida paras as teorias não-decidíveis. Para elas, considera-se o algoritmo da Figura 4.1 como um algoritmo de semi-decisão. Outra extensão da abordagem de checagem de SMT destina-se às fórmulas de primeira-ordem quantificadas. Dado o conjunto dos axiomas $Ax(\mathcal{T})$ da teoria $\mathcal{T}$ e uma fórmula de primeira-ordem quantificada ϕ, é necessário transformar ϕ em uma fórmula básica ϕ_b através da susbtituição de cada sub-fórmula quantificada de ϕ por novos símbolos proposicionais e adicionar suas definições Δ a Ax, de modo que $Ax \wedge \phi$ é equisatisfatível à $Ax \wedge \delta \wedge \phi$. O algoritmo é dado na Figura 4.2, cujas funções são:

*fechamento_existencial***:** realiza fechamento existencial em todas as variáveis livres;

*minimiza_escopo***:** implementa, em sentido inverso, regras de transformação de fórmulas na forma prenex para mover os quantificadores pra dentro da fórmula o máximo possível;

*elimina_existencial***:** remove quantificadores existenciais através de skolemização;

*renomeia_e_define***:** remove quantificadores universais substituindo variáveis quantificadas por novos símbolos proposicionais e adiciona as novas definições aos axiomas.

function TransformacaoBasica(ϕ : *formula quantificada*)
  $\phi_0 \longleftarrow$ *fechamento_existencial*(ϕ);
  $\phi_1 \longleftarrow$ *minimiza_escopo*(ϕ_0);
  $\phi_2 \longleftarrow$ *elimina_existencial*(ϕ_1);
  $(\phi_3, \Delta) \longleftarrow$ *renomeia_e_define*(ϕ_2);
  return (ϕ_3, Δ);
end

Figura 4.2: Algoritmo de transformação de fórmulas quantificadas em fórmulas básicas

Por fim, a fórmula ϕ_b e a teoria axiomatizada $Ax \cup \Delta$ resultantes são enviadas para o algoritmo da Figura 4.1.

4.2 haRVey: Provando e Depurando Especificações Baseadas em Conjuntos

O artigo "Proving and debugging set-based specifications" [37] apresenta uma técnica de prova para especificações baseadas em conjuntos e é adotada a linguagem do método B para testá-la. A técnica divide-se em quatro fases: (i) *tradução de AM*, que são traduzidas para fórmulas em linguagem de primeira-ordem com igualdade; (ii) *geração de obrigações de provas*, obedecendo as regras do método B; (iii) *tradução das obrigações de prova*, que passam a ser representadas através da teoria decidível dos vetores com extensionalidade; e (iv) *prova das obrigações de prova*, onde os quantificadores são eliminados e as fórmulas são validadas (ou não) pelo provador haRVey.

A sintaxe da linguagem de entrada do haRVey é semelhante à da linguagem LISP [38]. Trata-se de uma linguagem não-tipada e que oferece os seguintes símbolos de operadores: *not* (negação), *and* (conjunção), *or* (disjunção), $->$ (implicação), $<->$ (equivalência), *forall* (quantificador universal) e *exists* (quantificador existencial). Os arquivos de entrada do haRVey são divididos em duas partes: teoria τ, que caracteriza o problema e define suas propriedades; e hipótese a ser provada ϕ. A Figura 4.3 apresenta um exemplo de arquivo de entrada do haRVey.

```
(
  (forall s1 s2 (<-> (equals s1 s2)
            (forall e (<-> (member e s1) (member e s2)))))
)
(->
  (equals s s)
)
```

Figura 4.3: Exemplo de arquivo de entrada haRVey.

Na fase de tradução de AM, a AM é traduzida para um fragmento da teoria dos conjuntos denominado SSET. Trata-se de uma teoria da lógica de primeira-ordem tipada com igualdade que considera elementos (`elem`) e conjuntos de elementos (`set`). SSET contém

o símbolo de constante $\varnothing$ (*conjunto vazio*) do tipo `set`, o símbolo de função unária $\{\cdot\}$ (*conjunto unitário*) do tipo `elem` $\rightarrow$ `set`, os símbolos de função binária $\cup$, $\cap$ e $\setminus$ (*união*, *interseção* e *diferença*) do tipo `set` $\times$ `set` $\rightarrow$ `set` e os símbolos de predicado binário $\in$ e $\subseteq$ (*pertinência* e *subconjunto*) dos tipos `elem` $\times$ `set` e `set` $\times$ `set`, respectivamente. Além disso, há as variáveis do tipo τ e símbolos de igualdade $=_\tau$ para cada $\tau \in$ {`elem`, `set`}. Por fim, SSET possui os seguintes axiomas adicionados à teoria da igualdade:

1. **Axioma do conjunto vazio**: $\forall E \bullet (\neg\, E \in \varnothing)$;
2. **Axiomas do conjunto unitário**: $\forall E \bullet (E \in \{E\})$ e $\forall E, F \bullet (e \neq F \Rightarrow \neg\, E \in \{F\})$;
3. **Axioma da união de conjuntos**: $\forall E, S_1, S_2 \bullet (E \in S_1 \cup S_2 \Leftrightarrow (E \in S_1 \vee E \in S_2))$;
4. **Axioma da interseção de conjuntos**: $\forall E, S_1, S_2 \bullet (E \in S_1 \cap S_2 \Leftrightarrow (E \in S_1 \wedge E \in S_2))$;
5. **Axioma do diferença de conjuntos**: $\forall E, S_1, S_2 \bullet (E \in S_1 \setminus S_2 \Leftrightarrow (E \in S_1 \wedge \neg\, E \in S_2))$;
6. **Axioma do subconjunto**: $\forall S_1, S_2 \bullet (S_1 \subseteq S_2 \Leftrightarrow \forall E \bullet (E \in S_1 \Rightarrow E \in S_2))$;
7. **Axioma da extensionalidade**: $\forall S_1, S_2 \bullet (S_1 = S_2 \Leftrightarrow \forall E \bullet (E \in S_1 \Leftrightarrow E \in S_2))$,

onde E, F são variáveis do tipo `elem` e S, S_1, S_2 são variáveis do tipo `set`.

Na fase de geração de obrigações de provas, as obrigações de provas da AMN traduzidas em linguagem SSET são geradas de acordo com as regras de substituições generalizadas suportadas pelo método B (apresentadas no capítulo 2).

Na fase de tradução das obrigações de prova, as obrigações de provas em linguagem SSET são traduzidas para a teoria dos vetores $\mathcal{A}_s^e$ que considera os tipos `value`, `index` e `array`. $\mathcal{A}_s^e$ contém os símbolos de função `wr` (*escrita*) e `rd` (*leitura*) dos tipos `array` $\times$ `index` $\times$ `value` $\rightarrow$ `array` e `array` $\times$ `index` $\rightarrow$ `value`, respectivamente. Além disso, $\mathcal{A}_s^e$ possui os seguintes axiomas adicionados à teoria da igualdade:

1. **Axioma da leitura-escrita em mesmos índice**: a leitura do índice I do vetor A resulta no valor E escrito neste mesmo índice do vetor A.

$$\forall A, I, E \bullet (\mathtt{rd}(\mathtt{wr}(A, I, E), I) = E);$$

2. **Axioma da leitura-escrita em índices diferentes**: o resultado da leitura do índice J do vetor A independe dos valores escritos em outros índices deste vetor.

$$\forall A, I, J, E \bullet (I \neq J \Rightarrow \mathtt{rd}(\mathtt{wr}(A, I, E), J) = \mathtt{rd}(A, J));$$

3. **Axioma da extensionalidade**: se a leitura de cada um dos índices I de dois vetores A e B resulta em um mesmo valor, então A e B são vetores iguais.

$$\forall A, B \bullet (\forall I \bullet (\mathtt{rd}(A, I) = \mathtt{rd}(B, I)) \Rightarrow A = B),$$

A tradução da linguagem SSET para a linguagem $\mathcal{A}_s^e$ baseia-se na utilização de função característica para a representação de conjuntos. Assim, cada conjunto pode ser convertido em um vetor de booleanos cujo índices são os elementos do conjunto e o valor é `TRUE`, se tal elemento pertence ao conjunto, ou é `FALSE`, caso contrário. Isso ocorre através da aplicação dos seguintes axiomas:

4. **Axioma do valor verdade**: `tt` e `ff` são valores distintos e representam os valores verdadeiro e falso.

$$\mathtt{tt} \neq \mathtt{ff};$$

5. **Axioma do vetor vazio**: determina o mapeamento do símbolo de constante $\varnothing$ para o símbolo de constante `mty` do tipo `array`.

$$\forall I \bullet (\mathtt{rd}(\mathtt{mty}, I) = \mathtt{ff});$$

6. **Axioma da união de vetores**: determina o mapeamento do termo básico $s_1 \cup s_2$, representado pelo símbolo de constante u, para os vetores $\widehat{s_1}$ e s_2 correspondentes à s_1 e s_2 do tipo `set`, respectivamente.

$$\forall I \bullet (\mathtt{rd}(u, I) = \mathtt{tt} \Leftrightarrow (\mathtt{rd}(\widehat{s_1}, I) = \mathtt{tt} \vee \mathtt{rd}(\widehat{s_2}, I) = \mathtt{tt}));$$

7. **Axioma da pertinência**: determina a criação de símbolos de constantes $\widehat{s}$ do tipo `array`.

$$\forall I \bullet (\mathtt{rd}(s, I) = \mathtt{tt} \vee \mathtt{rd}(s, I) = \mathtt{ff});$$

8. **Axioma do subconjunto**: determina o mapeamento do termo básico $s_1 \subseteq s_2$, representado pelo símbolo de predicado q, para os vetores $\widehat{s_1}$ e s_2 correspondentes à s_1 e s_2 do tipo `set`, respectivamente.

$$q \Leftrightarrow \forall I \bullet (rd(\widehat{s_1}, I) = \mathtt{tt} \Rightarrow rd(\widehat{s_2}, I) = tt),$$

Os tipos `elem` e `set` são mapeados para `index` e `array`, respectivamente, enquanto que `value` é interpretado como o conjunto usual de valores verdade. Termos básicos da forma $\{e_1, ..., e_n\}$ são mapeados para `wr`(...`wr` (`mty`, e_1, `tt`) ..., e_n, `tt`). Para os termos básicos da forma $s_1 \bowtie s_2$ (onde s_1, s_2 são do tipo `set` e $\bowtie$ é uma das funções binárias do tipo `set` $\times$ `set` $\rightarrow$ `set`), s_1 e s_2 são traduzidos para $\widehat{s_1}$ e $\widehat{s_2}$ do tipo `array` de maneira análoga àquela do axioma 6. Para cada símbolo de constante $\widehat{s}$ do tipo `array`, deve-se gerar uma instância do axioma 7. Os átomos básicos da forma $e \in s$ são mapeados para $rd(\widehat{s}, \widehat{e}) =$ `tt`. E, por fim, os termos básicos da forma $e_1 = e_2$ e $s_1 = s_2$ são traduzidos para $\widehat{e_1} = \widehat{e_2}$ e $\widehat{s_1} = \widehat{s_2}$.

Para provar uma fórmula ϕ da linguagem SSET, deve-se assumir o fato de que ϕ é satisfatível módulo SSET se e somente se $\widehat{\phi}$ é satisfatível módulo a teoria dos vetores $\mathcal{A}_s^e$. Assim, na fase de prova das obrigações de prova, cada fórmula a ser checada é negada e, então, tenta-se provar que a nova fórmula é insatisfatível módulo a teoria dos vetores $\mathcal{A}_s^e$. O processo de prova consiste em três passos:

PASSO 1. ***Eliminação de quantificadores*****:** todas as variáveis a quantificadas existencialmente são substituídas por novas constantes $\overline{a}$. Em seguida a fórmula é transformada para a forma normal prenex e cada sub-fórmula ψ quantificada mais externamente é substituída por um novo símbolo proposicional q e a fórmula $q \Leftrightarrow \psi$ é adicionada as axiomas de $\mathcal{A}_s^e$.

PASSO 2. ***Checagem de satisfatibilidade*:** o provador automático de teoremas haRVey é acionado para checar a insatisfatibilidade da fórmula. O haRVey é uma ferramenta baseada na combinação flexível e eficiente de um provador de SMT, usando diagramas de decisão binária (BDDs) e um motor de satisfatibilidade baseado no algoritmo de Davis e Putnam melhorado (DPLL) [29], com prova de teoremas por superposição. Os átomos básicos da fórmula são abstraídos para símbolos proposicionais e têm sua estrutura booleana correspondente representada por BDDs. A representação por BDDs gera a forma normal disjuntiva da fórmula e o haRVey checa a satisfatibilidade de cada disjunção separadamente. Ou seja, o haRVey possui um procedimento de decisão para a checagem da satisfatibilidade de uma conjunção de literais básicos capaz de retornar subconjuntos menores insatisfatíveis.

PASSO 3. ***Provisão de contra-exemplos*:** uma vez que o haRVey encontrou uma sub-fórmula insatisfatível, faz-se necessário construir um modelo da negação da mesma, para servir como contra-exemplo. Para isso utiliza-se a ferramenta CLPS [11]. Trata-se de um provador de restrições para conjuntos hereditariamente finitos com átomos. *Conjuntos hereditariamente finitos* (em inglês, *Hereditarily Finite Sets*) são conjuntos finitos, cujos elementos são finitos, os elementos dos elementos são finitos e assim sucessivamente, incluindo a teoria SSET. CLPS é baseado na construção da teoria dos conjuntos. Sendo assim, deve-se traduzir a fórmula de volta para uma conjunção de literais em SSET. E como o conjunto universal de uma AM é instanciado por algum conjunto finito, então CLPS pode encontrar a solução que se constitui na transição de uma instância de AM que leve a um estado onde o invariante é violado. Para estabelecer se tal estado é ou não alcançável pode-se utilizar uma ferramenta de animação ou um checador de modelo.

Para ilustrar o processo completo de prova, será adotado o exemplo da Figura 4.4 que apresenta a especificação **incorreta** da operação `setPronto` de um escalonador na linguagem B. A máquina `escalonador` possui um conjunto de processos, denominado `PID`. A variáveis de estados são representados pelos conjuntos `ativo`, `pronto`, `esperando`.

O invariante determina que tais conjuntos devem ser subconjuntos de `PID`, disjuntos dois a dois e que sempre deve existir no máximo um processo ativo. Na inicialização, todos os processos pertencem ao conjunto `esperando`, ou seja, eles estão em estado de espera. A operação `setPronto` consiste em desbloquear um processo `pr` que está em estado de espera e colocá-lo no estado de pronto ou de ativo. O erro ocorre devido ao fato da operação não retirar `pr` da lista de processos em espera, violando o invariante.

```
MACHINE escalonador
SETS PID
VARIABLES ativo, pronto, esperando
INVARIANT pronto ⊆ PID ∧ ativo ⊆ PID ∧ esperando ⊆ PID
        ∧ pronto ∩ esperando = ∅∧ pronto ∩ ativo = ∅
        ∧ ativo ∩ esperando = ∅∧ card(ativo) ≤ 1
INITIALISATION ativo, pronto := ∅, ∅ ‖ esperando := PID
OPERATIONS
  setPronto ≙
    BEGIN
      ANY pr WHERE pr ∈ esperando
      THEN /* esperando := esperando - {pr} ‖ */
           IF ativo ≠ ∅
           THEN pronto := pronto ∪ {pr}
           ELSE ativo := {pr}
           END
      END
    END
END
```

Figura 4.4: Especificação incorreta de (parte de) um escalonador em linguagem B.

Na fase de tradução da AM, os seguintes predicados são gerados para a operação e o invariante:

$P_{setPronto}$(a, p, e, a', p', e'):

∃pr • (pr ∈ e ∧ (if a ≠ ∅ then p' = p ∪ {pr} else a' = {pr})),

Inv(a, p, e):

p ⊆ pid ∧ e ⊆ pid ∧ a ⊆ pid

∧ p ∩ e = ∅∧ p ∩ a = ∅∧ a ∩ e = ∅∧ card(a) ≤ 1,

onde p, e, a abreviam as variáveis pronto, esperando, ativo, respectivamente,

`p', e', a'` denotam os valores de `p, e, a` após a execução, e `pid` é uma constante que representa `PID`.

Na fase de geração de obrigações de prova, deve-se provar que a operação preserva o invariante, checando a validade da fórmula:

$$\forall \texttt{a,p,e,a',p',e'} \bullet (\mathit{Inv}(\texttt{a,p,e}) \wedge P_{\mathit{setPronto}}(\texttt{a,p,e,a',p',e'}) \Rightarrow \mathit{Inv}(\texttt{a',p',e'}))$$

Essa obrigação de prova é traduzida para a teoria dos vetores durante a fase de tradução das obrigações de prova e gera a seguinte fórmula:

$$\forall \widehat{a},\widehat{p},\widehat{e},\widehat{a'},\widehat{p'},\widehat{e'} \bullet \overbrace{\underbrace{(q_0 \wedge q_1 \wedge q_2 \wedge \widehat{p}\_ \cap \_\widehat{e} = \texttt{mty} \wedge \widehat{p}\_ \cap \_\widehat{a} = \texttt{mty} \wedge \widehat{a}\_ \cap \_\widehat{e} = \texttt{mty} \wedge (\widehat{a} = \texttt{mty} \vee q_3)}}^{\mathit{Inv}(\widehat{a},\widehat{p},\widehat{e})} \wedge \overbrace{\exists \widehat{pr} \bullet (\texttt{rd}(\widehat{e},\widehat{pr}) = \texttt{tt} \wedge \texttt{if } \widehat{a} \neq \texttt{mty then } \widehat{p'} = \texttt{wr}(\widehat{p},\widehat{pr},\texttt{tt}) \texttt{ else } \widehat{a} = \texttt{wr}(\texttt{mty},\widehat{pr},\texttt{tt}))}^{P_{\mathit{setPronto}}(\widehat{a},\widehat{p},\widehat{e},\widehat{a'},\widehat{p'},\widehat{e'})} \Rightarrow \overbrace{\underbrace{q_4 \wedge q_5 \wedge q_6 \wedge \widehat{p'}\_ \cap \_\widehat{e'} = \texttt{mty} \wedge \widehat{p'}\_ \cap \_\widehat{a'} = \texttt{mty} \wedge \widehat{a'}\_ \cap \_\widehat{e'} = \texttt{mty} \wedge (\widehat{a'} = \texttt{mty} \vee q_7))}}^{\mathit{Inv}(\widehat{a'},\widehat{p'},\widehat{e'})}$$

onde as variáveis são renomeadas com o mesmo nome sobreposto pelo símbolo $\widehat{\ }$, as proposições são renomeadas para novas variáveis q_i, os conjuntos vazios são representados por `mty` e as conjunções também são renomeadas para $\widehat{x}\_ \cap \_\widehat{y}$. Além disso, a relação de pertinência é traduzida para a teoria dos vetores como uma operação de leitura, onde $x \in \mathit{set}$ equivale à $\texttt{rd}(\widehat{set},\widehat{x}) = \texttt{tt}$ e $x \notin \mathit{set}$ corresponde à $\texttt{rd}(\widehat{set},\widehat{x}) = \texttt{ff}$. As operações que modificam um conjunto são traduzidas em operações de escrita, onde adicionar um elemento x à um conjunto *set* corresponde à $\texttt{wr}(\widehat{set},\widehat{x},\texttt{tt})$ e remover um elemento x de um conjunto *set* corresponde à $\texttt{wr}(\widehat{set},\widehat{x},\texttt{ff})$.

Na fase de prova das obrigações de prova, o haRVey indica que a obrigação de prova não é válida e retorna o seguinte conjunto de literais:

$\{q_0, q_1, q_2, q_7,\ \widehat{a} = \texttt{mty},\ \widehat{e'} = \widehat{e}, \widehat{p'} = \widehat{p}, \widehat{a'} = i_{25},\ \texttt{rd}(\widehat{e},\texttt{pr}) = \texttt{tt},\ i_{17} = \texttt{mty},\ i_{14} = \texttt{mty},\ i_{11} = \texttt{mty},\ i_{35} \neq \texttt{mty}\},$

cuja associação entre literais originais da teoria dos conjuntos com os literais do haRVey é dada na Tabela 4.1, onde o literal destacado corresponde à violação do invariante.

HARVEY	SSET	HARVEY	SSET
q_0	$\texttt{a} \subseteq \texttt{pid}$	$\widehat{p'} = \widehat{p}$	$\texttt{p'} = \texttt{p}$
q_1	$\texttt{e} \subseteq \texttt{pid}$	$\widehat{a'} = i_{25}$	$\texttt{a'} = \{\texttt{pr}\}$
q_2	$\texttt{p} \subseteq \texttt{pid}$	$\texttt{rd}(\widehat{e},\texttt{pr}) = \texttt{tt}$	$\texttt{pr} \in \texttt{e}$
q_3	$\lvert\ \texttt{a}\ \rvert = 1$	$i_{17} = \texttt{mty}$	$\texttt{a} \cap \texttt{e} = \varnothing$
q_7	$\lvert\ \texttt{a'}\ \rvert = 1$	$i_{14} = \texttt{mty}$	$\texttt{p} \cap \texttt{e} = \varnothing$
$\widehat{a} = \texttt{mty}$	$\texttt{a} = \varnothing$	$i_{11} = \texttt{mty}$	$\texttt{p} \cap \texttt{a} = \varnothing$
$\widehat{e'} = \widehat{e}$	$\texttt{e'} = \texttt{e}$	$i_{35} \neq \texttt{mty}$	$\boxed{\texttt{a'} \cap \texttt{e'} \neq \varnothing}$

Tabela 4.1: Associações entre literais do haRVey e átomos da teoria dos conjuntos.

No último passo, o provador CLPS é acionado e, instanciando o conjunto universal `PID` com o conjunto unitário $\{p1\}$, retorna a seguinte solução:

$$\texttt{e} = \texttt{e'} = \texttt{a'} = \{\texttt{p1}\} \quad \text{e} \quad \texttt{a} = \texttt{p} = \texttt{p'} = \varnothing$$

Através de uma ferramenta de animação ou um checador de modelo, constata-se que a solução encontrada identifica um estado alcançável a partir do estado inicial. Assim, o usuário deve modificar a operação a fim impossibilitar a existência desse estado, ou seja, a existência de um processo *pr* pertencente aos conjuntos *ativo* e *esperando*, simultaneamente. Para corrigir a operação, basta descomentar a linha delimitada por `/*` e `*/` na Figura 4.4.

Apesar da eficiência do haRVey na verificação de especificações B, uma nova versão da ferramenta tem sido desenvolvida. O intuito dessa versão é ampliar a aplicabilidade

do haRVey para um grupo maior de construções da teoria dos conjuntos. Nela, adota-se a teoria axiomática dos conjuntos NBG para o processo de prova das obrigações de prova. Com isso, substitui-se a tradução de AMN para SSET pela tradução de AMN para NBG. Além disso, elimina-se a fase de tradução das obrigações de prova, resultando na checagem de satisfatibilidade modulo teoria NBG. A teoria dos conjuntos NBG é uma formulação alternativa da teoria ZFC e ambas são apresentadas no capítulo 5.

Capítulo 5

Teorias Axiomáticas dos Conjuntos

O capítulo 4 apresentou uma alternativa de verificação formal fornecida pelo provador haRVey. Trata-se de uma abordagem baseada na checagem de SMT para a teoria dos vetores $\mathcal{A}^e_\varsigma$. Uma vez que essa teoria não permite expressar todas as construções necessárias às especificações baseadas em conjuntos, como as especificações B, convém estender a checagem de SMT para teorias dos conjuntos a fim de aplicá-la à verificação das obrigações de prova do método B. Assim, duas teorias axiomáticas clássicas dos conjuntos são apresentadas neste capítulo: a teoria dos conjuntos de Zermelo-Fräenkel (ZFC) e a teoria dos conjuntos de von Neumann-Bernays-Gödel (NBG), equivalente à ZFC, mas com um conjunto finito de axiomas.

5.1 Teoria dos Conjuntos ZFC

A primeira axiomatização da teoria dos conjuntos amplamente aceita pelos matemáticos foi proposta por Ernest Zermelo em 1908 [46]. A teoria de Zermelo formula-se através da adição do símbolo relacional binário de pertinência "$\in$" ao cálculo de predicados com igualdade. Para Zermelo todos os objetos são conjuntos, ou seja, os conjuntos são definidos a partir de outros conjuntos. Assim, $X \in Y$ significa que o conjunto X é um elemento do conjunto Y.

Existem diversas formas de apresentação dos oito axiomas da teoria de Zermelo. Por exemplo, o *axioma do par* pode ser definido como "se X e Y são conjuntos, então existe um conjunto que contém apenas X e Y" e o *axioma do conjunto vazio* pode estar incluso no *axioma dos conjuntos elementares* ou pode não ser considerado um axioma. A formalização adotada neste trabalho apresenta a teoria dos conjuntos de Zermelo através dos seguintes axiomas:

1. **Axioma da extensionalidade**: se todo elemento de um conjunto X é também um elemento de Y e vice-versa, então $X = Y$.

$$\forall X, Y \bullet (\forall Z \bullet (Z \in X \leftrightarrow Z \in Y) \rightarrow X = Y)$$

2. **Axioma do conjunto vazio**: existe um conjunto que não possui elementos.

$$\exists X \bullet \forall Y \bullet Y \notin X$$

3. **Axioma das partes**: para todo conjunto X existe um conjunto $\mathbb{P}(X)$, o *conjunto das partes* de X, que contém todos os subconjuntos de X.

$$\forall X \bullet \exists Y \bullet \forall Z \bullet (Z \subseteq X \leftrightarrow Z \in Y)$$

4. **Axioma da união**: para todo conjunto X existe um conjunto $\bigcup X$, o *conjunto união* de X, que contém todos os elementos dos elementos de X.

$$\forall X \bullet \exists Y \bullet \forall Z \bullet (\exists V \bullet (Z \in V \wedge V \in X) \rightarrow Z \in Y)$$

5. **Axioma do par**: se X e Y são conjuntos, então existe um conjunto do qual X e Y são elementos.

$$\forall X, Y \bullet \exists Z \bullet (X \in Z \wedge Y \in Z)$$

6. **Axioma do infinito**: existe um conjunto X que contém o conjunto vazio como elemento e, para cada um de seus elementos Y ele contém o sucessor da forma $\{Y\}$.

$$\exists X \bullet (\varnothing \in X \wedge \forall Y \bullet (Y \in X \rightarrow \{Y\} \in X))$$

7. **Axioma da escolha**: para todo conjunto X existe uma função binária F que torna X bem ordenado. Ou seja, F é uma função de ordem em X e todo subconjunto não-vazio de X tem exatamente um elemento na imagem da função F.

$$\forall X \bullet (\forall Y \bullet (Y \in X \rightarrow Y \neq \varnothing) \rightarrow \exists F \bullet (F \; bem\_ordena \; X))$$

ou seja

$$\forall X \bullet (\forall Y \bullet (Y \in X \leftrightarrow Y \neq \varnothing) \rightarrow \exists F \bullet \forall Y \bullet (Y \in X \rightarrow \exists Z \bullet (Z \in Y \wedge F(Y) = Z)))$$

8. **Axioma da especificação** (ou *Axioma da separação*): seja $\varphi(Z)$ uma propriedade definida para todos os elementos de X, X possui um subconjunto X_φ contendo apenas os elementos Z de X para os quais $\varphi(Z)$ é verdadeiro.

$$\forall X \bullet \exists Y \bullet \forall Z \bullet (Z \in Y \leftrightarrow Z \in X \wedge \varphi(Z))$$

O axioma da especificação da teoria de Zermelo utiliza um conceito vago de "definitividade" (em inglês, *definiteness*) de uma propriedade. Para cada conjunto, há uma propriedade definida correspondente a ele e que caracteriza seus elementos. Mas nada se pode afimar sobre o inverso, ou seja, não há garantias de que existe um conjunto correspondente a cada propriedade definida para um conjunto. Para solucionar esse problema, Abraham Fräenkel [20] e Thoralf Skolem [34] definiram formalmente o conceito de "propriedade definida" como sendo qualquer propriedade que pode ser formulada através da lógica de primeira-ordem com os predicados adicionais de pertinência "$\in$" e de igualdade "$=$". A partir dessa definição, Fräenkel e Skolem formularam um novo axioma, denominado axioma da substituição:

9. **Axioma da substituição**: seja φ uma propriedade definida para alguns pares (x, y) no domínio X. Se para todo x existe um único y para o qual φ é verdadeiro então existe um conjunto Y que contém todos os y associados a algum x.

$$\forall X \bullet (\forall x \bullet (x \in X \rightarrow \exists! y \bullet \varphi(x,y)) \rightarrow \exists Y \bullet \forall y \bullet (y \in Y \leftrightarrow \exists x \bullet (x \in X \wedge \varphi(x,y))))$$

onde $\exists! y$ significa que "existe um único y".

A substituição do axioma da especificação pelo axioma da substituição e a adição do axioma da regularidade resultaram na teoria dos conjuntos de Zermelo-Fräenkel (ZF). O axioma da regularidade foi formulado por John Von Neumann [43] a fim de proibir a existência de conjuntos tais como $X = \{X\}$ ou um par de conjuntos $X = \{Y\}$ e $Y = \{X\}$:

10. **Axioma da regularidade** (ou Axioma da fundação): todo conjunto não-vazio X contém um elemento x, tal que X e x são conjuntos disjuntos.

$$\forall X \bullet (X \neq \varnothing \rightarrow \exists x \bullet (x \in X \wedge x \cap X = \varnothing))$$

O axioma da regularidade determina que um conjunto é criado por etapas. Na etapa 0 tem-se um conjunto de átomos. A etapa 1 é a coleção dos átomos e dos conjuntos de átomos da etapa 0. Por exemplo, se x e y são dois átomos, então a etapa 0 forma o conjunto $\{x,y\}$ e a etapa 1 forma o conjunto $\{x,y,\varnothing,\{x\},\{y\},\{x,y\}\}$. O conjunto da etapa 2 é constituído pelos elementos da etapa 1 e por todos os conjuntos formados por estes elementos. E assim sucessivamente. Dado um conjunto X_n formado através de n etapas, tal que $n \geq 0$, a aplicação do axioma da escolha a este conjunto resulta no conjunto X_0.

A existência de átomos na teoria ZF não levanta problemas conceituais, ao contrário do axioma da escolha que é polêmico pelo seu caráter não-construtivista. Isso significa que os resultados obtidos através da utilização do axioma da escolha envolvem conjuntos que não podem ser construídos explicitamente. Assim, quando uma teoria dos conjuntos requer o axioma da escolha, costuma-se acrescentar a letra "C" (de *Axiom of* ***C****hoice*) à sua sigla correspondente. Por exemplo, a teoria dos conjuntos ZF apresentada nesta seção também é conhecida como teoria dos conjuntos ZFC.

Enquanto os axiomas 2 até 6 consistem em axiomas de existência, o axioma 9, bem como o axioma 8, corresponde a um *axioma esquema* que deve ser instanciado por todas

os predicados possíveis. Dessa forma, os axiomas de existência asseguram a existência de conjuntos iniciais sobre os quais aplicam-se axiomas esquemas de maneira a formar novos conjuntos. Portanto, a teoria dos conjuntos ZFC não pode ser axiomatizada por um conjunto finito de axiomas.

5.2 Teoria dos Conjuntos NBG

Proposta inicialmente por von Neumann em 1920, modificada por Paul Bernays em 1937 e simplificada por Gödel em 1940, a teoria dos conjuntos de von Neumann-Bernays-Gödel (NBG) [30] consiste em uma formulação alternativa da teoria dos conjuntos ZFC. A teoria dos conjuntos NBG apresenta os mesmos resultados que a ZFC, mas com um conjunto finito de axiomas.

A teoria dos conjuntos NBG caracteriza-se por ser uma teoria com dois tipos básicos: conjuntos e classes. O conjuntos representam coleções de objetos (que podem ser conjuntos) e satisfazem os axiomas da teoria dos conjuntos ZFC. As classes representam coleções de conjuntos. Todos os conjuntos são classes, mas nem todas as classes são conjuntos. As *classes próprias* são classes que não são conjuntos e, portanto, não pertencem a outras classes. A relação de pertinência entre conjuntos e classes é definida por:

$$a \in s,$$

onde a denota um conjunto e s representa um conjunto ou uma classe. Adotando-se a convenção de que conjuntos são representados por letras minúsculas e classes são representadas por letras maiúsculas, a correspondência entre conjuntos e classes que possuem os mesmos elementos é definida através da função Rp:

$$\mathrm{Rp}(A, a) = \forall\, x \bullet (x \in a \leftrightarrow x \in A),$$

ou seja, um conjunto a representa uma classe A se todo elemento de a é também elemento de A e vice versa. Portanto, as classes próprias não possuem um conjunto que as represente.

A teoria dos conjuntos NBG é composta por dez axiomas, sendo seis referentes aos conjuntos e quatro referentes às classes:

1. **Axioma da extensionalidade**: se dois conjuntos a e b têm os mesmo elementos, então eles são iguais.

$$\forall x \bullet (x \in a \leftrightarrow x \in b) \leftrightarrow a = b)$$

2. **Axioma do par**: para quaisquer dois conjuntos x e y existe um conjunto cujos elementos são, exatamente, x e y.

$$\forall x, y \bullet \exists a \bullet a = x, y$$

3. **Axioma da união**: para todo conjunto x existe um conjunto $\bigcup x$ formado pelos elementos dos elementos de x.

$$\forall x \bullet \exists a \bullet \forall y \bullet (\exists z \bullet (z \in y \wedge y \in x) \rightarrow z \in a)$$

4. **Axioma das partes**: para todo conjunto x existe um conjunto $\mathbb{P}(x)$ que contém todos os subconjuntos de x.

$$\forall x \bullet \exists a \bullet \forall y \bullet (y \subseteq x \leftrightarrow y \in a)$$

5. **Axioma do infinito**: existe um conjunto x que contém o conjunto vazio como elemento e, para cada um de seus elementos y ele contém o sucessor da forma $\{y\}$.

$$\exists x \bullet (\varnothing \in x \wedge \forall y \bullet (y \in x \rightarrow \{y\} \in x))$$

6. **Axioma da regularidade**: todo conjunto não-vazio x contém um elemento y, tal que x e y são conjuntos disjuntos.

$$\forall x \bullet (x \neq \varnothing \rightarrow \exists y \bullet (y \in x \wedge y \cap x = \varnothing))$$

7. **Axioma da Extensionalidade**: se duas classes A e B têm os mesmos elementos, então elas são iguais.

$$\forall x \bullet (x \in A \leftrightarrow x \in B) \leftrightarrow A = B)$$

8. **Axioma do Tamanho**: para toda classe A existe um conjunto a, tal que $\mathrm{Rp}(A, a)$, se e somente se não existe uma bijeção entre A e a classe universo Ω formada por todos os conjuntos. Ou seja, todas as classes que possuem a mesma cardinalidade da classe universo ω são, de fato, classes e todas as outras classes são conjuntos.

$$\forall A \bullet \exists a \bullet (\mathrm{Rp}(A, a) \leftrightarrow \neg \ \exists \varphi \bullet \varphi \in A \rightarrowtail\!\!\!\!\!\rightarrow \Omega$$

9. **Axioma da Regularidade**: toda classe não-vazia A contém um conjunto a, tal que A e a são conjuntos disjuntos.

$$\forall A \bullet (A \neq \varnothing \rightarrow \exists a \bullet (a \in A \wedge a \cap A = \varnothing))$$

10. **Axioma da Formação de Classes**: para toda fórmula φ contendo classes livres de quantificadores existe uma classe A contendo apenas os conjuntos a para os quais $\varphi(a)$ é verdadeiro.

$$\forall \varphi \bullet \exists A \bullet \forall a \bullet (a \in A \leftrightarrow \varphi(a))$$

O axioma da Formação de Classes é um axioma esquema que pode ser expandido em um conjunto finito de axiomas através de sua aplicação aos axiomas de existência (axiomas de 2 até 5). Ou seja, esse axioma pode ser susbtituído por um número finito de casos particulares básicos, a partir dos quais é possível demonstrar todos os outros [23]. A partir de instanciações de φ no axioma da Formação de Classes é possível obter os seguintes axiomas:

- Axioma da classe vazia: $\varnothing \equiv \exists A \bullet \forall a \bullet (a \in A \leftrightarrow a \neq a)$

- Axioma da classe universal: $\Omega \equiv \exists A \bullet \forall a \bullet (a \in A \leftrightarrow a = a)$
- Axioma da interseção de classes: $X \cap Y \equiv \exists A \bullet \forall a \bullet (a \in A \leftrightarrow a \in X \land a \in Y)$
- Axioma da união de classes: $X \cup Y \equiv \exists A \bullet \forall a \bullet (a \in A \leftrightarrow a \in X \lor a \in Y)$
- Axioma da diferença de classes: $X \setminus Y \equiv \exists A \bullet \forall a \bullet (a \in A \leftrightarrow a \in X \land a \notin Y)$
- Axioma da classe complemento: $\overline{X} \equiv \exists A \bullet \forall a \bullet (a \in A \leftrightarrow a \notin X)$
- Axioma da classe das partes: $\mathbb{P}X \equiv \exists A \bullet \forall a \bullet (a \in A \leftrightarrow a \subseteq X)$
- Axioma do produto cartesiano: $X \times Y \equiv \exists A \bullet \forall a, b \bullet ((a, b) \in A \leftrightarrow a \in X \land b \in Y)$
- Axioma da imagem: $\operatorname{ran} R \equiv \exists B \bullet \forall b \bullet (b \in B \leftrightarrow \exists a \bullet (a \in A \land (a, b) \in R)$

A teoria dos conjuntos NBG também pode ser apresentada como uma teoria com um único tipo, onde os conjuntos diferenciam-se das classes através da definição “uma classe é um conjunto se ele é um elemento de outra classe”. Aliando-se a isso o fato de ser uma axiomatização finita e de grande poder de prova, a adoção da teoria dos conjuntos NBG na checagem de satisfatibilidade através do provador haRVey torna-se uma alternativa viável, eficiente e adequada à verificação de especificações formais.

Capítulo 6

Incorporando Teoria dos Conjuntos NBG ao haRVey

Para estender a capacidade dedutiva do haRVey à teoria dos conjuntos, é necessário haver uma definição axiomática da mesma. Tendo em vista que o haRVey implementa a abordagem de checagem de SMT descrita no capítulo 4, a teoria adotada deve ser representada por uma axiomatização finita. Portanto, a teoria NBG apresenta-se como uma opção adequada, uma vez que essa abordagem pode ser estendida para as teorias não-decidíveis.

Conforme dito na seção 5.2 do capítulo 5, o axioma da Formação de Classes da teoria dos conjuntos NBG pode ser expandido em um conjunto finito de axiomas. Estes representam os casos particulares básicos, resultantes da aplicação do axioma da Formação de Classes aos axiomas de existência, a partir dos quais é possível demonstrar todos os outros. A fim de aplicar a checagem de SMT às especificações em B, o conjunto dos axiomas básicos deve ser determinado com base nos operadores disponíveis na linguagem B. Ou seja, cada operador da linguagem B é considerado como uma classe que é definida através de um axioma. Como o método B reconhece apenas inteiros e conjuntos como tipos básicos, a teoria NBG é apresentada com um único tipo, onde todas as classes são conjuntos. Assim, os operadores sobre conjuntos são representados pelos seguintes axiomas de definição:

- **(Ax 1)** Definição de negação de pertinência

  $\forall e, s \bullet (\neg (e \in s) \leftrightarrow e \notin s)$

- **(Ax 2)** Definição de conjunto unitário

  $\forall e \bullet e \in \{e\}$

  $\forall e_1, e_2 \bullet (e_1 \in \{e_2\} \leftrightarrow e_1 = e_2)$

- **(Ax 3)** Definição de conjunto complemento

  $\forall e, s \bullet ((e \in \mathit{universe} \wedge e \notin s) \leftrightarrow e \in \overline{s})$

- **(Ax 4)** Definição de conjunto das partes

  $\forall s_1, s_2 \bullet (s_1 \in \mathbb{P}\, s_2 \leftrightarrow \forall e \bullet (e \in s_1 \rightarrow e \in s_2))$

- **(Ax 5)** Definição de conjunto vazio

  $\forall e \bullet e \notin \varnothing$

  $\mathbb{P}\, \varnothing = \{\varnothing\}$

- **(Ax 6)** Definição de subconjunto

  $\forall s_1, s_2 \bullet (s_1 \subseteq s_2 \leftrightarrow \forall e \bullet (e \in s_1 \rightarrow e \in s_2))$

- **(Ax 7)** Definição de negação de subconjunto

  $\forall s1, s2 \bullet (s1 \not\subseteq s2 \leftrightarrow \neg (s1 \subseteq s2))$

- **(Ax 8)** Definição de subconjunto próprio

  $\forall s_1, s_2 \bullet (s_1 \subset s_2 \leftrightarrow (s_1 \subseteq s_2 \wedge s_1 \neq s_2))$

- **(Ax 9)** Definição de negação de subconjunto próprio

  $\forall s1\ s2 \bullet (s1 \not\subset s2 \leftrightarrow (s1 \subseteq s2 \rightarrow s1 = s2))$

- **(Ax 10)** Definição de união

  $\forall e, s_1, s_2 \bullet (e \in (s_1 \cup e_2) \leftrightarrow (e \in s_1 \vee e \in s_2))$

- **(Ax 11)** Definição de interseção

  $\forall e, s_1, s_2 \bullet (e \in (s_1 \cap s_2) \leftrightarrow (e \in s_1 \wedge e \in s_2))$

- **(Ax 12)** Definição de diferença

  $\forall e, s_1, s_2 \bullet (e \in (s_1 \setminus s_2) \leftrightarrow (e \in s_1 \land e \notin s_2))$

- **(Ax 13)** Definição de par ordenado

  $\forall e_1, e_2, c1, c2 \bullet ((e_1, e_2) = (c1, c2) \leftrightarrow (e_1 = c1 \land e_2 = c2))$

  $\forall e_1, e_2 \bullet (e_1, e_2) = \{\{e_1\}\} \cup \{\{e_1\} \cup \{e_2\}\}$

- **(Ax 14)** Definição de projeção1

  $\forall e_1, e_2 \bullet \mathit{first}(e_1, e_2) = e_1$

- **(Ax 15)** Definição de projeção2

  $\forall e_1, e_2 \bullet \mathit{second}(e_1, e_2) = e_2$

- **(Ax 16)** Definição de produto cartesiano

  $\forall s_1, s_2, s_3 \bullet (s_3 \in s_1 \times s_2 \leftrightarrow \exists e_1, e_2 \bullet (e_1 \in s_1 \land e_2 \in s_2 \land s_3 = (e_1, e_2))))$

- **(Ax 17)** Definição de igualdade de conjuntos (axioma da extensionalidade)

  $\forall s1, s2 \bullet (s1 = s2 \leftrightarrow \forall e \bullet (e \in s1 \leftrightarrow e \in s2))$

- **(Ax 18)** Definição de desigualdade de conjuntos

  $\forall s1, s2 \bullet (s1 \neq s2 \leftrightarrow \neg (s1 = s2))$

Há ainda os axiomas que determinam a existência de certos conjuntos:

- **(Ax 19)** Axioma do conjunto universo

  $\exists \mathit{universe} \bullet \forall e \bullet e \in \mathit{universe}$

- **(Ax 20)** Axioma do par

  $\forall s_1, s_2 \bullet \exists s \bullet \forall e \bullet (e \in s \leftrightarrow (e \in s_1 \lor e \in s_2))$

- **(Ax 21)** Axioma do conjunto das partes

  $\forall s_1 \bullet \exists s_2 \bullet s_2 = \mathbb{P}\, s_1$

- **(Ax 22)** Axioma do conjunto união:

  $\forall s, e \bullet \exists s_1 \bullet (e \in s1 \leftrightarrow \exists s_2 \bullet (e \in s_2 \land s_2 \in s))$

- **(Ax 23)** Axioma do infinito

  $\exists s \bullet (\varnothing \in s \wedge (\forall e \bullet (e \in s \rightarrow \{e\} \in s)))$

- **(Ax 24)** Axioma da regularidade

  $\forall s \bullet (s \neq \varnothing \rightarrow \exists e \bullet (e \in s \wedge (e \cup s \neq \varnothing)))$

Com a axiomatização da teoria dos conjuntos NBG para o haRVey, expandindo sua capacidade dedutiva para esta teoria, a aplicação do mesmo não mais requer a tradução de uma AM para a linguagem SSET, geração das obrigações de prova SSET e posterior tradução destas obrigações para a teoria dos vetores, explicada na seção 4.2 do capítulo 4. Ao invés disso, pode-se aplicar a ferramenta Batcave diretamente sobre uma AM, a fim de obter as obrigaç¿øes de prova em linguagem de entrada do haRVey, já acrescida de novos símbolos de operadores, predicados e funções. Por convenção, adota-se os símbolos de constantes e_n e s_n, tal que $n \in \mathbb{N}$, para a representação de elementos e de conjuntos, respectivamente. A Tabela 6.1 apresenta os novos símbolos disponíveis no haRVey, onde pode-se destacar os símbolos de função *first* e *second* que denotam funções "não-puras", pois seus resultados são números inteiros. Embora números inteiros possam ser modelados por conjuntos, não o fazemos para tirar proveito de procedimentos de decisão para fragmentos da aritmética inteira.

MATEMÁTICA	haRVEY	MATEMÁTICA	haRVEY
$s_1 = s_2$	`(seteq s1 s2)`	$first(s)$	`(first s)`
$s_1 \neq s_2$	`(nseteq s1 s2)`	$second(s)$	`(second s)`
$e \in s$	`(in e s)`	$\mathbb{P}s$	`(power s)`
$e \notin s$	`(notin e s)`	$\mathbb{P}_1 s$	`(power1 s)`
$s_1 \subseteq s_2$	`(subseteq s1 s2)`	$\overline{s}$	`(compl s)`
$s_1 \nsubseteq s_2$	`(nsubseteq s1 s2)`	$s_1 \cup s_2$	`(cup s1 s2)`
$s_1 \subset s_2$	`(subset s1 s2)`	$s_1 \cap s_2$	`(cap s1 s2)`
$s_1 \not\subset s_2$	`(nsubset s1 s2)`	$s_1 \setminus s_2$	`(setminus s1 s2)`
(e_1, e_2)	`(ord_pair e1 e2)`	$\varnothing$	`emptyset`
$s_1 \times s_2$	`(cross s1 s2)`	$\{e\}$	`(unitset e)`

Tabela 6.1: Operadores em notação matemática e notação haRVey.

Para a checagem de SMT, o haRVey executa duas heurísticas, definidas no artigo "Satisfiability solving for software verification" [18] para redução do espaço de busca do

provador por superposição. Com isso, apenas o número mínimo de instâncias básicas dos axiomas é considerado. As duas heurísticas são:

- *Expansão de definição*: dada uma fórmula ϕ de primeira-ordem, todas as ocorrências de aplicações de predicados em ϕ são substituídas por instâncias das definições destes predicados e estas definições são retiradas do conjunto de axiomas a ser considerado;

- *Redução de teorias extensas*: a axiomatização é estruturada em módulos, denominados *teorias*, de maneira hierárquica, onde cada *teoria* define a semântica de um conjunto de símbolos através de seus respectivos axiomas. A declaração de utilização de símbolos definidos em outras teorias ocorre através da cláusula de importação explícita `extends`. Assim, dada uma fórmula ϕ de primeira-ordem, são computados os símbolos que ocorrem em ϕ e apenas os seus respectivos axiomas de definição são considerados, ou seja, os axiomas das teorias correspondentes a cada símbolo, bem como os axiomas das teorias estendidas.

Por exemplo: seja a teoria τ composta pelos 24 axiomas da teoria NBG para haRVey e a fórmula $\phi \equiv e \in \{e\} \cup s$. A expansão de definição substitui o predicado $\{e\} \cup s$ pela instância de sua definição em **(Ax 10)** resultando em $\phi \equiv e \in \{e\} \vee e \in s$. A redução de teorias extensas determina que apenas os axiomas referentes à definição do simbolo $\{\cdot\}$ devem ser considerados para a prova de ϕ. Portanto, visto que a definição de ϕ não se utiliza de outras definições, apenas **(Ax 2)** é considerado para o espaço de busca da prova de ϕ no provador por superposição.

Para tirar proveito da heurística de redução de teorias, a teoria NBG para haRVey é estruturada em módulos. Cada operador corresponde a um módulo contendo seus axiomas de definição. E os axiomas de existência são agrupados em um módulo único que é estendido por todos os outros. A axiomatização completa em linguagem haRVey encontra-se no Apêndice B.

6.1 Resultados Obtidos

A fim de avaliar as vantagens obtidas através da implementação da abordagem hierárquica para a axiomatização, realizou-se um experimento comparativo entre a prova de propriedades da teoria NBG para o haRVey utilizando a abordagem hierárquica e a abordagem monolítica. Para cada definição de operação foram selecionadas propriedades estabelecidas por tal operação, como, por exemplo, reflexividade (propriedade 9), simetria (propriedade 10), e transitividade (propriedade 11), para o operador de igualdade. Enquanto que na abordagem hierárquica os axiomas são estruturados em módulos separados, na abordagem monolítica todos os axiomas são agrupados em um único módulo, impossibilitando a aplicação da heurística de redução de teorias.

Os experimentos foram realizados em um Pentium IV 2.4GHz, rodando Linux (Ubuntu 5.10) com 512MB de RAM e 40GB de espaço de disco. Os resultados obtidos durante as provas na versão 5.10 do haRVey (incorporando a versão 0.91 - "Kanyam" - do provador E [17]) podem ser vistos nas Tabelas 6.2 e 6.3, onde foram considerados os critérios de *tempo* de duração da prova, número de *cláusulas* `#Cls` (incluindo iniciais e geradas) utilizadas para a prova e número de *passos* `#Ps` necessários para gerar a cláusula vazia (□), uma vez que a prova é feita por contradição. Nela, as propriedades 9, 16 e 22 destacam-se devido ao fato de que são trivialmente provadas somente com o uso da lógica proposicional, sem haver necessidade de iniciar uma prova na lógica de primeira-ordem e, portanto, nenhuma cláusula é utilizada e nenhum passo de prova é criado.

Os resultados obtidos demonstram que, com a axiomatização da teoria do conjuntos NBG orientada aos operadores de conjuntos da linguagem B, o haRVey é capaz de validar teoremas (que definem propriedades) indispensáveis a cada um destes operadores. Além disso, a axiomatização em ambas as abordagens, hierárquica e monolítica, são capazes de provar os mesmos teoremas. De maneira geral, a adoção da abordagem hierárquica reduz consideravelmente o tempo de prova e, principalmente, o número de cláusulas envolvidas na prova.

Tabela 6.2: Propriedades Provadas.

REF	PROPRIEDADE	PR. MONOLÍTICA			PR. HIERÁRQUICA		
		Tempo	#Cls	#Ps	Tempo	#Cls	#Ps
0	$\forall e, s \bullet (\{e\} \in \mathbb{P}s \leftrightarrow e \in s)$	0.280s	118	17	0.235s	65	23
1	$\forall e1, e2 \bullet (\{e1\} = \{e2\} \leftrightarrow e1 = e2)$	0.256s	109	17	0.219s	20	11
2	$\forall e \bullet \varnothing \neq \{e\}$	0.263s	91	14	0.175s	26	11
3	$\forall e \bullet \{e\} \neq \varnothing$	0.258s	92	14	0.165s	26	11
4	$\forall e, s \bullet (s \in \mathbb{P}\{e\} \leftrightarrow (s = \{e\} \lor s = \varnothing))$	0.365s	78	11	0.269s	41	11
5	$\forall s \bullet s \in \mathbb{P}s$	0.252s	93	11	0.132s	25	11
6	$\forall s1, s2 \bullet (\mathbb{P}s1 \in \mathbb{P}(\mathbb{P}s2) \leftrightarrow s1 \in \mathbb{P}s2)$	0.409s	141	17	0.261s	91	19
7	$\forall s1, s2, s3 \bullet ((s1 \in \mathbb{P}s2 \land s2 \in \mathbb{P}s3) \rightarrow s1 \in \mathbb{P}s3)$	0.252s	103	16	0.166s	58	16
8	$\forall e, s1, s2 \bullet ((e \in s1 \land s1 \in \mathbb{P}s2) \rightarrow e \in s2)$	0.271s	118	17	0.158s	52	18
9	$\forall s \bullet s = s$	0.268s	100	17	0.068s	-	-
10	$\forall s1, s2 \bullet (s1 = s2 \rightarrow s2 = s1)$	0.274s	109	17	0.207s	13	10
11	$\forall s1, s2, s3 \bullet ((s1 = s2 \land s2 = s3) \rightarrow s1 = s3)$	0.274s	109	17	0.226s	23	14
12	$\forall s1, s2 \bullet (s1 = s2 \rightarrow (s1 \in \mathbb{P}s2) \land s2 \in \mathbb{P}s1)$	0.476s	103	17	0.421s	47	16
13	$\forall s1, s2, s \bullet (s1 = s2 \leftrightarrow (\forall r1 \bullet (r1 \in s1 \rightarrow r1 \in s2) \land \forall r2 \bullet (r2 \in s2 \rightarrow r2 \in s1))$	0.637s	101	15	0.313s	18	12
14	$\forall s \bullet \varnothing \in \mathbb{P}s$	0.241s	76	10	0.152s	22	10
15	$\forall s \bullet (s = \varnothing \lor \exists e \bullet e \in s)$	0.242s	74	10	0.225s	7	6
16	$\forall s \bullet s \subseteq s$	0.265s	100	17	0.069s	-	-
17	$\forall s \bullet \varnothing \subseteq s$	0.241s	69	9	0.160s	26	6
18	$\forall s1, s2, s3 \bullet ((s1 \subseteq s2 \land s2 \subseteq s3) \rightarrow s1 \subseteq s3)$	0.271s	109	17	0.145s	21	15
19	$\forall s1, s2 \bullet (s1 \in \mathbb{P}_1 s2 \leftrightarrow (s1 \in \mathbb{P}s2 \land s1 \neq \varnothing))$	0.671s	110	17	0.467s	65	17
20	$\mathbb{P}_1 \varnothing = \varnothing$	0.268s	115	17	0.173s	50	17
21	$\forall e \bullet \mathbb{P}_1\{e\} = \{\{e\}\}$	0.254s	109	17	0.170s	55	17
22	$\forall s \bullet s \not\subset s$	0.273s	113	17	0.074s	-	-
23	$\forall s \bullet (\varnothing \subset s \leftrightarrow s \neq \varnothing)$	0.479s	73	9	0.337s	44	8
24	$\forall s \bullet s = s \cup s$	0.266s	109	17	0.221s	20	9
25	$\forall s \bullet s \cup s = s$	0.267s	109	17	0.216s	22	8
26	$\forall s1, s2 \bullet (s2 \subseteq s1 \rightarrow s1 \cup s2 = s1)$	0.267s	109	17	0.255s	90	13
27	$\forall s1, s2 \bullet (s2 \subseteq s1 \rightarrow s1 \cup s2 = s1)$	0.267s	109	17	0.255s	90	13
28	$\forall s \bullet s \cup \varnothing = s$	0.264s	109	17	0.254s	90	13
29	$\forall s \bullet \varnothing \cup s = s$	0.263s	109	17	0.255s	90	13
30	$\forall s1, s2 \bullet s1 \cup s2 = s2 \cup s1$	0.265s	109	17	0.222s	36	16
31	$\forall s1, s2, s3 \bullet (s1 \cup s2) \cup s3 = s1 \cup (s2 \cup s3)$	0.264s	109	17	0.230s	48	25
32	$\forall s1, s2, s3 \bullet ((s1 \cup s2) \in \mathbb{P}s3 \leftrightarrow (s1 \in \mathbb{P}s3 \land s2 \in \mathbb{P}s3))$	0.499s	110	17	0.324s	47	16
33	$\forall s1, s2 \bullet (s1 \cup s2 = \varnothing \leftrightarrow (s1 = \varnothing \land s2 = \varnothing))$	0.265s	109	17	0.160s	37	17
34	$\forall s1, s2 \bullet (\varnothing = s1 \cup s2 \leftrightarrow (s1 = \varnothing \land s2 = \varnothing))$	0.265s	109	17	0.158s	37	17
35	$\forall s1, s2, s3 \bullet s1 \cup (s2 \cup s3) = s2 \cup (s1 \cup s3)$	0.266s	109	17	0.229s	48	25
36	$\forall s1, s2, s3 \bullet ((s1 \in \mathbb{P}s2 \lor s1 \in \mathbb{P}s3) \rightarrow s1 \in \mathbb{P}(s1 \cup s3))$	0.353s	111	16	0.185s	74	16
37	$\forall e1, e2 \bullet (e1 \neq e2 \rightarrow \forall s \bullet (e2 \in (\{e1\} \cup s) \leftrightarrow e2 \in s))$	0.261s	103	18	0.199s	21	14
38	$\forall s1, s2 \bullet (s1 \subseteq s2 \rightarrow (s1 \cap s2) = s1)$	0.265s	109	17	0.248s	34	8
39	$\forall s1, s2 \bullet (s2 \subseteq s1 \rightarrow (s1 \cap s2) = s2)$	0.265s	109	17	0.249s	32	8
40	$\forall s \bullet \varnothing \cap s = \varnothing$	0.266s	109	21	0.156s	37	17
41	$\forall s \bullet s \cap \varnothing = \varnothing$	0.260s	101	14	0.176s	37	17
42	$\forall e, s \bullet ((e \in s \rightarrow \{e\} \cap s = \{e\}) \land (e \notin s \rightarrow \{e\} \cap s = \varnothing))$	0.359s	73	12	0.236s	33	12
43	$\forall e, s \bullet ((e \in s \rightarrow s \cap \{e\} = \{e\}) \land (e \in s \rightarrow s \cap \{e\} = \varnothing))$	0.358s	73	12	0.238s	33	12

Tabela 6.3: Propriedades Provadas (continuação).

Ref	Propriedade	Pr. Monolítica			Pr. Hierárquica		
		Tempo	#Cls	#Ps	Tempo	#Cls	#Ps
44	$\forall s1, s2 \bullet s1 \cap (s1 \cup s2) = s1$	0.268s	109	17	0.230s	27	8
45	$\forall s1, s2 \bullet (s1 \cup s2) \cap s1) = s1$	0.268s	109	17	0.228s	26	8
46	$\forall s \bullet s = s \cap s$	0.268s	109	17	0.217s	22	9
47	$\forall s \bullet s \cap s = s$	0.264s	109	17	0.217s	20	8
48	$\forall s1, s2 \bullet s1 \cap s2 = s2 \cap s1$	0.266s	109	17	0.227s	27	17
49	$\forall s1, s2, s3 \bullet (s1 \cap s2) \cap s3 = s1 \cap (2 \cap s3)$	0.263s	109	17	0.234s	44	27
50	$\forall s1, s2, s3 \bullet ((s1 \subseteq s3 \lor s2 \subseteq s3) \rightarrow (s1 \cap s2) \subseteq s3)$	0.266s	109	17	0.223s	28	13
51	$\forall s1, s2, s3(s1 \in \mathbb{P}(s2 \cap s3) \rightarrow (s1 \in \mathbb{P}s2 \land s1 \in \mathbb{P}s3))$	0.348s	111	16	0.314s	64	15
52	$\forall s1, s2, s3 \bullet (s1 \cap (s2 \cap s3) = s2 \cap (s1 \cap s3))$	0.267s	109	17	0.241s	44	27
53	$\forall s1, s2, s3 \bullet ((\mathbb{P}s1 \in \mathbb{P}s3 \lor \mathbb{P}s2 \in \mathbb{P}s3) \rightarrow (s1 \cap s2) \in s3)$	0.302s	145	17	0.206s	88	15
54	$\forall e, s1, s2 \bullet (e \in s2 \rightarrow ((\{e\} \cup s1) \cap s2 = \{e\} \cup (s1 \cap s2))$	0.269s	109	17	0.252s	56	29
55	$\forall e, s1, s2 \bullet (e \notin s2 \rightarrow ((\{e\} \cup s1) \cap s2 = s1 \cap s2))$	0.269s	109	19	0.252s	56	30
56	$\forall s \bullet s \setminus s = \varnothing$	0.267s	109	17	0.158s	37	17
57	$\forall s1, s2, s3 \bullet (s1 \setminus s2) \setminus s3 = s1 \setminus (s2 \cup s3)$	0.267s	109	17	0.258s	73	32
58	$\forall s1, s2, s3 \bullet ((s1 \setminus s2) \subseteq s3 \leftrightarrow s1 \subseteq (s2 \cup s3))$	0.265s	109	17	0.236s	41	22
59	$\forall s1, s2 \bullet (s1 \in \mathbb{P}s2 \rightarrow (s1 \setminus s2) = \varnothing)$	0.258s	109	17	0.159s	46	17
60	$\forall s \bullet \varnothing \setminus s = \varnothing$	0.263s	101	14	0.155s	37	17
61	$\forall s \bullet s \setminus \varnothing = s$	0.263s	109	17	0.156s	37	17
62	$\forall e, s \bullet ((e \in s \rightarrow \{e\} \setminus s = \varnothing) \land (e \notin s \rightarrow \{e\}))$	0.357s	73	12	0.234s	33	12
63	$\forall s1, s2, s3 \bullet (\mathbb{P}s1 \in \mathbb{P}s3 \rightarrow (s1 \setminus s2) \in s3)$	0.261s	127	17	0.201s	87	34
64	$\forall e, s1, s2 \bullet (e \in s2 \rightarrow (\{e\} \cup s1) \setminus s2 = s1 \setminus s2)$	0.270s	109	17	0.255s	73	31
65	$\forall e, s1, s2 \bullet (e \notin s2 \rightarrow (\{e\} \cup s1) \setminus s2 = \{e\} \cup (s1 \setminus s2))$	0.270s	109	19	0.261s	75	32
66	$\forall e1, e2, s1, s2 \bullet ((e1, e2) \in s1 \times s2 \leftrightarrow (e1 \in s1 \land e2 \in s2))$	0.267s	109	17	0.458s	38	17
67	$\forall s \bullet \varnothing \times s = \varnothing$	0.268s	113	17	0.183s	43	17
68	$\forall s \bullet s \times \varnothing = \varnothing$	0.266s	113	17	0.180s	43	17
69	$\forall s1, s2 \bullet (s1 \times s2 = \varnothing \leftrightarrow (s1 = \varnothing \lor s2 = \varnothing))$	0.508s	109	21	0.186s	46	17
70	$\forall s1, s2, s3 \bullet (s1 \cup (s2 \cap s3) = (s1 \cup s2) \cap (s1 \cup s3))$	0.267s	109	17	0.248s	58	25
71	$\forall s1, s2, s3 \bullet ((s1 \cap s2) \cup s3 = (s1 \cup s3) \cap (s2 \cup s3))$	0.269s	109	17	0.241s	58	35
72	$\forall s1, s2, s3 \bullet (s1 \cap (s2 \cup s3) = (s1 \cap s2) \cup (s1 \cap s3))$	0.268s	109	17	0.246s	57	34
73	$\forall s1, s2, s3 \bullet ((s1 \cup s2) \cap s3 = (s1 \cap s3) \cup (s2 \cap s3))$	0.270s	109	17	0.248s	57	34
74	$\forall s1, s2, s3 \bullet s1 \setminus (s2 \cup s3) = (s1 \setminus s2) \cap (s1 \setminus s3)$	0.268s	109	17	0.266s	89	39
75	$\forall s1, s2, s3 \bullet (s1 \cup s2) \setminus s3 = (s1 \setminus s3) \cup (s2 \setminus s3)$	0.268s	109	17	0.258s	82	36
76	$\forall s1, s2, s3 \bullet s1 \setminus (s2 \cap s3) = (s1 \setminus s2) \cup (s1 \setminus s3)$	0.270s	109	17	0.273s	89	39
77	$\forall s1, s2, s3 \bullet (s1 \cap s2) \setminus s3 = (s1 \setminus s3) \cap (s2 \setminus s3)$	0.272s	109	17	0.266s	84	37
78	$\forall s1, s2, s3 \bullet s1 \setminus (s2 \setminus s3) = (s1 \setminus s2) \cup (s1 \cap s3)$	0.269s	109	17	0.268s	88	39
79	$\forall s1, s2, s3 \bullet (s2 \subseteq s3 \rightarrow (s1 \cap s2 \subseteq s1 \cap s3))$	0.266s	109	17	0.166s	82	23
80	$\forall s1, s2, s3 \bullet (s2 \subseteq s3 \rightarrow (s1 \cup s2 \subseteq s1 \cup s3))$	0.274s	109	17	0.149s	37	21
81	$\forall s1, s2, s3 \bullet (s2 \subseteq s3 \rightarrow (s2 \setminus s1 \subseteq s3 \setminus s1))$	0.267s	109	17	0.288s	259	41
82	$\forall s1, s2, s3 \bullet (s2 \subseteq s3 \rightarrow (s1 \setminus s3 \subseteq s1 \setminus s2))$	0.266s	109	17	0.236s	205	26
83	$\forall s1, s2 \bullet s1 \subseteq (s1 \cup s2)$	0.263s	109	17	0.147s	25	9
84	$\forall s1, s2 \bullet (s1 \cap s2) \subseteq s1$	0.271s	109	17	0.142s	25	8
85	$\forall s1, s2 \bullet s1 \in \mathbb{P}(s1 \cup s2)$	0.266s	110	17	0.142s	35	16

Capítulo 7

Considerações Finais

Este trabalho apresentou um projeto para a expansão da sintaxe e da capacidade dedutiva do provador de teoremas haRVey aplicado à verificação de especificações B. Ao provador, foram adicionados axiomas definidos pela teoria dos conjuntos NBG, acrescentando-s novos símbolos e facilitando a prova de propriedades desta teoria, ou seja, adequando o haRVey ao método B, uma vez que a teoria dos conjuntos constitui uma das bases de B. Neste contexto, foram apresentadas neste trabalho: as principais características do método B; um resumo das principais abordagens de verificação formal aplicáveis a este método e implementadas por ferramentas de prova; a abordagem de checagem de SMT implementada pelo haRVey; e a teoria dos conjuntos NBG originária da teoria dos conjuntos ZFC.

A abordagem de checagem de SMT implementada pelo haRVey requer uma teoria finita, pois adotando-se uma teoria inifita, como por exemplo a teoria ZFC, o haRVey aplicaria as regras do cálculo de superposição a um conjunto infinito, aumentando a probabilidade do processo de prova não terminar. Assim, a teoria dos conjuntos NBG foi escolhida por tratar-se de uma axiomatização finita, possibilitando adotá-la na checagem de SMT do haRVey. Com isso, pode-se considerar, então, a possibilidade de utilizar o haRVey para a verificação das obrigações de prova originadas de acordo com o método B. Tais obrigações de prova podem ser diretamente geradas em linguagem de entrada do

haRVey através da ferramenta Batcave.

A teoria dos conjuntos NBG foi definida no haRVey com um único tipo, onde todas as classes são conjuntos, uma vez que a notação B só reconhece inteiros e conjuntos de inteiros. Além disso, o axioma de Formação de Classes foi substituído por dezoito novos axiomas. Cada operador da linguagem B foi considerado como uma classe definida por um axioma. Todos os axiomas foram estruturados adotando-se as abordagens hierárquica e monolítica. Com a realização de experimentos considerando propriedades básicas dos operadores de conjuntos, observou-se que a axiomatização NBG é capaz de prová-las com eficiência, principalmente quando esta axiomatização é definida de maneira hierárquica, o que leva à redução, de maneira geral, do espaço de busca e do tempo de prova.

A reformulação da teoria dos conjuntos NBG guiou-se pelos operadores presentes nas obrigações de prova geradas pelas regras do cálculo das substituições, que fundamenta o método B. Além dos operadores de conjuntos apresentados na Tabela 6.1 do capítulo 6, há ainda os operadores de relação, os operadores de função e os operadores de conjuntos "$\mathbb{F}$" (*subconjuntos finitos*), "$\mathbb{F}_1$" (*subconjuntos finitos e não-vazios*), "$\bigcup$" (*união generalizada*), "$\bigcap$" (*interseção generalizada*) e "$|\cdot|$" (*cardinalidade*) que ainda não foram axiomatizados. Isso ocorre devido à dificuldade em representá-los através de instanciações adequadas do axioma de Formação de Classes, sendo considerado parte dos trabalhos futuros.

Ainda como trabalhos futuros, tem-se a integração automática entre o Batcave e o haRVey. Atualmente, o gerador de obrigações de prova do Batcave é capaz de gerar obrigações, em linguagem genérica definida pelo projeto, para uma quantidade significativa (em torno de 80%) das construções de substituições do método B. Uma vez terminada a axiomatização dos operadores da notação B para o haRVey, a nova sintaxe da linguagem de entrada do haRVey poderá ser adotada pelo Batcave para geração das obrigações de prova nesta linguagem e, posterior verificação automática executada pelo haRVey. Assim, a integração Batcave-haRVey resultará em um sistema gratuito e de arquitetura aberta que fornece suporte a todas as fases do processo de desenvolvimento de *software* proporcio-

nado pelo método B.

Devido ao fato da teoria dos conjuntos NBG ser não-decidível, sua aplicação à checagem SMT no haRVey não apresenta resultados formais de decidibilidade. No entanto, com base nos resultados satisfatórios obtidos durante a prova de propriedades da teoria dos conjuntos, a incorporação da teoria dos conjuntos NBG ao haRVey apresenta-se com uma alternativa eficiente para a verificação de especificações B. Considerando-se que o haRVey provê um mecanismo de definição axiomática da semântica das diversas construções utilizadas nas especificações, a expansão do conjunto de operadores suportados pelo haRVey é facilmente obtida, diferentemente das outras abordagens de verificação existentes.

Apêndice A

Especificações e Obrigações de Prova

A.1 Especificação e Obrigações de Prova da Máquina

A especificação completa da máquina `leilao` é dada por:

```
MACHINE leilao (OBJETOS)
CONSTRAINTS card(OBJETOS) ∈ ℕ1
CONSTANTS lance_min
PROPERTIES lance_min ∈ ℕ1
SETS RESP = {aberto, fechado, inexistente}
VARIABLES sessao, leiloados
INVARIANT sessao ∈ OBJETOS ↔ ℕ1 ∧ leiloados ⊆ dom(sessao)
ASSERTIONS leiloados ⊆ OBJETOS
INITIALISATION sessao, leiloados := ∅, ∅
OPERATIONS
  iniciar(obj)≙
    PRE obj ∈ OBJETOS \ dom(sessao)
    THEN sessao := sessao ∪ {obj ↦ lance_min}
    END;
```

```
  propor(lance, obj) ≙
    PRE lance ∈ ℕ1 ∧ obj ∈ dom(sessao) \ leiloados ∧
        lance > max(sessao[{obj}])
    THEN sessao := sessao ∪ {obj ↦ lance}
    END;

  encerrar(obj) ≙
    PRE obj ∈ dom(sessao) \ leiloados
    THEN leiloados := leiloados ∪ {obj}
    END;

  resp, m ⟵ consultar(obj) ≙
    PRE obj ∈ dom(sessao)
    THEN m := max(sessao[{obj}]) ||
         IF obj ∉ leiloados
         THEN resp := aberto
         ELSE resp := fechado
         END
    END
END
```

As obrigações de prova da máquina `leilao` são definidas a seguir:

Inicialização:

```
  REST_TOTAL ⇒ [SUBS]INV
⇔
  TIPO_PARAM ∧ TIPO_CONJ ∧ REST_PARAM ∧ REST_CONST
  ⇒ [INIT]INV
⇔
  OBJETOS ∈ ℙ1(ℤ)∧ lance_min ∈ ℕ1
```

$\wedge$ RESP $\in \mathbb{P}_1(\mathbb{Z}) \wedge$ RESP = {aberto, fechado, inexistente}

$\wedge$ aberto $\neq$ fechado $\wedge$ aberto $\neq$ inexistente

$\wedge$ fechado $\neq$ inexistente $\wedge$ card(OBJETOS) $\in \mathbb{N}_1$

$\Rightarrow$ [sessao, leiloados := $\varnothing$, $\varnothing$] sessao $\in$ OBJETOS $\leftrightarrow \mathbb{N}_1$

$\wedge$ leiloados $\subseteq$ dom(sessao)

$\Leftrightarrow$

OBJETOS $\in \mathbb{P}_1(\mathbb{Z})$

$\Rightarrow \varnothing \in$ OBJETOS $\leftrightarrow \mathbb{N}_1 \wedge \varnothing \subseteq$ OBJETOS

Asserção:

REST_TOTAL $\wedge$ INV $\Rightarrow$ ASSERT

$\Leftrightarrow$

TIPO_PARAM $\wedge$ TIPO_CONJ $\wedge$ REST_PARAM $\wedge$ REST_CONST $\wedge$

INV $\Rightarrow$ ASSERT

$\Leftrightarrow$

OBJETOS $\in \mathbb{P}_1(\mathbb{Z}) \wedge$ lance_min $\in \mathbb{N}_1$

$\wedge$ RESP $\in \mathbb{P}_1(\mathbb{Z}) \wedge$ RESP = {aberto, fechado, inexistente}

$\wedge$ aberto $\neq$ fechado $\wedge$ aberto $\neq$ inexistente

$\wedge$ fechado $\neq$ inexistente $\wedge$ card(OBJETOS) $\in \mathbb{N}_1$

$\wedge$ sessao $\in$ OBJETOS $\leftrightarrow \mathbb{N}_1 \wedge$ leiloados $\subseteq$ dom(sessao)

$\Rightarrow$ leiloados $\subseteq$ OBJETOS

Operação `iniciar`:

REST_TOTAL $\wedge$ INV $\wedge$ ASSERT $\wedge$ PRE $\Rightarrow$ [SUBS]INV

$\Leftrightarrow$

TIPO_PARAM $\wedge$ TIPO_CONJ $\wedge$ REST_PARAM $\wedge$ REST_CONST $\wedge$

INV $\wedge$ ASSERT $\wedge$ PRE $\Rightarrow$ [SUBS]INV

$\Leftrightarrow$

OBJETOS $\in \mathbb{P}_1(\mathbb{Z}) \wedge$ lance_min $\in \mathbb{N}_1$

$\wedge$ RESP $\in \mathbb{P}_1(\mathbb{Z}) \wedge$ RESP = {aberto, fechado, inexistente}

$\wedge$ aberto $\neq$ fechado $\wedge$ aberto $\neq$ inexistente

$\wedge$ fechado $\neq$ inexistente $\wedge$ card(OBJETOS) $\in \mathbb{N}_1$

$\wedge$ sessao $\in$ OBJETOS $\leftrightarrow \mathbb{N}_1 \wedge$ leiloados $\subseteq$ dom(sessao)

$\wedge$ leiloados $\subseteq$ OBJETOS

$\wedge$ obj $\in$ OBJETOS $\setminus$ dom(sessao)

$\Rightarrow$ [sessao := sessao $\cup$ {obj $\mapsto$ lance_min}]

sessao $\in$ OBJETOS $\leftrightarrow \mathbb{N}_1 \wedge$ leiloados $\subseteq$ dom(sessao)

$\Leftrightarrow$

lance_min $\in \mathbb{N}_1 \wedge$ obj $\in$ OBJETOS $\setminus$ dom(sessao)

$\wedge$ sessao $\in$ OBJETOS $\leftrightarrow \mathbb{N}_1$

$\Rightarrow$ sessao $\cup$ {obj $\mapsto$ lance_min} $\in$ OBJETOS $\leftrightarrow \mathbb{N}_1$

Operação `propor`:

REST_TOTAL $\wedge$ INV $\wedge$ ASSERT $\wedge$ PRE $\Rightarrow$ [SUBS]INV

$\Leftrightarrow$

TIPO_PARAM $\wedge$ TIPO_CONJ $\wedge$ REST_PARAM $\wedge$ REST_CONST $\wedge$

INV $\wedge$ ASSERT $\wedge$ PRE $\Rightarrow$ [SUBS]INV

$\Leftrightarrow$

OBJETOS $\in \mathbb{P}_1(\mathbb{Z}) \wedge$ lance_min $\in \mathbb{N}_1$

$\wedge$ RESP $\in \mathbb{P}_1(\mathbb{Z}) \wedge$ RESP = {aberto, fechado, inexistente}

$\wedge$ aberto $\neq$ fechado $\wedge$ aberto $\neq$ inexistente

$\wedge$ fechado $\neq$ inexistente $\wedge$ card(OBJETOS) $\in \mathbb{N}_1$

$\wedge$ sessao $\in$ OBJETOS $\leftrightarrow \mathbb{N}_1 \wedge$ leiloados $\subseteq$ dom(sessao)

$\wedge$ leiloados $\subseteq$ OBJETOS

$\wedge$ lance $\in$ N1 $\wedge$ obj $\in$ dom(sessao) $\setminus$ leiloados

$\wedge$ lance > max(sessao[{obj}]

$\Rightarrow$ [sessao := sessao $\cup$ {obj $\mapsto$ lance}]

sessao $\in$ OBJETOS $\leftrightarrow \mathbb{N}_1 \wedge$ leiloados $\subseteq$ dom(sessao)

$\Leftrightarrow$

$$
\begin{array}{l}
\texttt{sessao} \in \texttt{OBJETOS} \leftrightarrow \mathbb{N}_1 \\
\wedge\ \texttt{lance} \in \texttt{N1} \wedge \texttt{obj} \in \texttt{dom(sessao)} \setminus \texttt{leiloados} \\
\Rightarrow\ \texttt{sessao} \cup \{\texttt{obj} \mapsto \texttt{lance}\} \in \texttt{OBJETOS} \leftrightarrow \mathbb{N}_1
\end{array}
$$

Operação `encerrar`:

$$
\begin{array}{l}
\texttt{REST\_TOTAL} \wedge \texttt{INV} \wedge \texttt{ASSERT} \wedge \texttt{PRE} \Rightarrow [\texttt{SUBS}]\texttt{INV} \\
\Leftrightarrow \\
\texttt{TIPO\_PARAM} \wedge \texttt{TIPO\_CONJ} \wedge \texttt{REST\_PARAM} \wedge \texttt{REST\_CONST} \wedge \\
\texttt{INV} \wedge \texttt{ASSERT} \wedge \texttt{PRE} \Rightarrow [\texttt{SUBS}]\texttt{INV} \\
\Leftrightarrow \\
\texttt{OBJETOS} \in \mathbb{P}_1(\mathbb{Z}) \wedge\ \texttt{lance\_min} \in \mathbb{N}_1 \\
\wedge\ \texttt{RESP} \in \mathbb{P}_1(\mathbb{Z}) \wedge\ \texttt{RESP = \{aberto, fechado, inexistente\}} \\
\wedge\ \texttt{aberto} \neq \texttt{fechado} \wedge \texttt{aberto} \neq \texttt{inexistente} \\
\wedge\ \texttt{fechado} \neq \texttt{inexistente} \wedge \texttt{card(OBJETOS)} \in \mathbb{N}_1 \\
\wedge\ \texttt{sessao} \in \texttt{OBJETOS} \leftrightarrow \mathbb{N}_1 \wedge\ \texttt{leiloados} \subseteq \texttt{dom(sessao)} \\
\wedge\ \texttt{leiloados} \subseteq \texttt{OBJETOS} \\
\wedge\ \texttt{obj} \in \texttt{dom(sessao)} \setminus \texttt{leiloados} \\
\Rightarrow\ [\texttt{leiloados := leiloados} \cup \{\texttt{obj}\}] \\
\qquad\qquad \texttt{sessao} \in \texttt{OBJETOS} \leftrightarrow \mathbb{N}_1 \wedge\ \texttt{leiloados} \subseteq \texttt{dom(sessao)} \\
\Leftrightarrow \\
\texttt{sessao} \in \texttt{OBJETOS} \leftrightarrow \mathbb{N}_1 \wedge\ \texttt{leiloados} \subseteq \texttt{OBJETOS} \\
\wedge\ \texttt{obj} \in \texttt{dom(sessao)} \setminus \texttt{leiloados} \\
\Rightarrow\ \texttt{leiloados} \cup \{\texttt{obj}\} \subseteq \texttt{dom(sessao)}
\end{array}
$$

Operação `consultar`:

$$
\begin{array}{l}
\texttt{REST\_TOTAL} \wedge \texttt{INV} \wedge \texttt{ASSERT} \wedge \texttt{PRE} \Rightarrow [\texttt{SUBS}]\texttt{INV} \\
\Leftrightarrow \\
\texttt{TIPO\_PARAM} \wedge \texttt{TIPO\_CONJ} \wedge \texttt{REST\_PARAM} \wedge \texttt{REST\_CONST} \wedge \\
\texttt{INV} \wedge \texttt{ASSERT} \wedge \texttt{PRE} \Rightarrow [\texttt{SUBS}]\texttt{INV}
\end{array}
$$

$\Leftrightarrow$

$\texttt{OBJETOS} \in \mathbb{P}_1(\mathbb{Z}) \wedge \texttt{lance\_min} \in \mathbb{N}_1$

$\wedge\ \texttt{RESP} \in \mathbb{P}_1(\mathbb{Z}) \wedge \texttt{RESP} = \{\texttt{aberto}, \texttt{fechado}, \texttt{inexistente}\}$

$\wedge\ \texttt{aberto} \neq \texttt{fechado} \wedge \texttt{aberto} \neq \texttt{inexistente}$

$\wedge\ \texttt{fechado} \neq \texttt{inexistente} \wedge \texttt{card(OBJETOS)} \in \mathbb{N}_1$

$\wedge\ \texttt{sessao} \in \texttt{OBJETOS} \leftrightarrow \mathbb{N}_1 \wedge\ \texttt{leiloados} \subseteq \texttt{dom(sessao)}$

$\wedge\ \texttt{leiloados} \subseteq \texttt{OBJETOS}$

$\wedge\ \texttt{obj} \in \texttt{dom(sessao)}$

$\Rightarrow$ [m := max(sessao[{obj}]) $\|$

IF obj $\notin$ leiloados THEN resp := aberto

ELSE resp := fechado END]

sessao $\in$ OBJETOS $\leftrightarrow \mathbb{N}_1 \wedge$ leiloados $\subseteq$ dom(sessao)

$\Leftrightarrow$

$\texttt{sessao} \in \texttt{OBJETOS} \leftrightarrow \mathbb{N}_1 \wedge\ \texttt{leiloados} \subseteq \texttt{dom(sessao)}$

$\wedge\ \texttt{obj} \in \texttt{dom(sessao)}$

$\Rightarrow\ (\texttt{obj} \notin \texttt{leiloados} \Rightarrow \texttt{sessao} \in \texttt{OBJETOS} \leftrightarrow \mathbb{N}_1)$

$\wedge\ (\texttt{obj} \in \texttt{leiloados} \Rightarrow \texttt{sessao} \in \texttt{OBJETOS} \leftrightarrow \mathbb{N}_1)$

A.2 Especificação e Obrigações de Prova do Refinamento

A seguir, tem-se o refinamento da máquina `leilao`:

```
REFINEMENT leilao_ref (OBJETOS)
REFINES leilao
VARIABLES rsessao, rleiloados
INVARIANT rsessao ∈ OBJETOS ⇸ ℕ1 ∧ dom(rsessao) = dom(sessao) ∧
   ∀x • (x ∈ dom(rsessao) ⇒ rsessao(x) = max(sessao[{x}])) ∧
   rleiloados = leiloados
INITIALISATION rsessao, rleiloados := ∅, ∅
OPERATIONS
```

```
  iniciar(obj) ≙
    PRE obj ∈ OBJETOS \ dom(rsessao)
    THEN rsessao := rsessao ⊕ {obj ↦ lance_min}
    END;

  propor(lance, obj) ≙
    PRE lance ∈ ℕ1 ∧ obj ∈ dom(rsessao) \ rleiloados ∧
        lance > rsessao(obj)
    THEN rsessao := rsessao ⊕ {obj ↦ lance}
    END;

  encerrar(obj) ≙
    PRE obj ∈ dom(rsessao) \ rleiloados
    THEN rleiloados := rleiloados ∪ {obj}
    END;

  resp, m ⟵ consultar(obj) ≙
    PRE obj ∈ dom(rsessao)
    THEN m := rsessao(obj);
         IF obj ∉ rleiloados
         THEN resp := aberto
         ELSE resp := fechado
         END
    END
END
```

As obrigações de prova associadas ao refinamento são:

Inicialização:

$$C_M \wedge D_{MR} \Rightarrow [SUBS_R] \neg [SUBS_{R-1}] \neg INV_R$$

$$\Leftrightarrow$$

$$C_M \wedge \ \texttt{TIPO\_PARAM}_{MR} \wedge \ \texttt{TIPO\_CONJ}_{MR} \wedge \ \texttt{REST\_CONST}_{MR}$$
$$\Rightarrow \ [\texttt{SUBS}_R] \neg \ [\texttt{SUBS}_{R-1}] \neg \ \texttt{INV}_R$$

$$\Leftrightarrow$$

$$\texttt{OBJETOS} \in \mathbb{P}_1(\mathbb{Z}) \wedge \ \texttt{lance\_min} \in \mathbb{N}_1$$
$$\wedge \ \texttt{RESP} \in \mathbb{P}_1(\mathbb{Z}) \wedge \ \texttt{RESP} = \{\texttt{aberto}, \texttt{fechado}, \texttt{inexistente}\}$$
$$\wedge \ \texttt{aberto} \neq \texttt{fechado} \wedge \texttt{aberto} \neq \texttt{inexistente}$$
$$\wedge \ \texttt{fechado} \neq \texttt{inexistente} \wedge \texttt{card(OBJETOS)} \in \mathbb{N}_1$$
$$\Rightarrow [\texttt{rsessao}, \texttt{rleiloados} := \varnothing, \varnothing] \ \neg \ [\texttt{sessao}, \texttt{leiloados} := \varnothing, \varnothing]$$
$$\neg \ (\texttt{rsessao} \in \texttt{OBJETOS} \nrightarrow \mathbb{N}_1 \wedge \ \texttt{dom(rsessao)} = \texttt{dom(sessao)}$$
$$\wedge \forall \texttt{x} \bullet (\texttt{x} \in \texttt{dom(rsessao)}$$
$$\Rightarrow \ \texttt{rsessao(x)} = \texttt{max(sessao[\{x\}])})$$
$$\wedge \ \texttt{rleiloados} = \texttt{leiloados})$$

$$\Leftrightarrow$$

$$\varnothing \in \texttt{OBJETOS} \nrightarrow \mathbb{N}_1 \wedge \ \texttt{dom}(\varnothing) = \texttt{dom}(\varnothing)$$
$$\wedge \forall \texttt{x} \bullet (\texttt{x} \in \texttt{dom}(\varnothing) \Rightarrow \varnothing(\texttt{x}) = \texttt{max}(\varnothing[\{\texttt{x}\}]))$$

Operação `iniciar`:

$$C_M \wedge \ D_{MR} \wedge \ \texttt{INV}_{MR} \wedge \ \texttt{PRE}_{MR-1} \Rightarrow \ \texttt{PRE}_R \wedge \ [\texttt{SUBS}_R] \neg \ [\texttt{SUBS}_{R-1}] \neg \ \texttt{INV}_R$$

$$\Leftrightarrow$$

$$C_M \wedge \ \texttt{TIPO\_PARAM}_{MR} \wedge \ \texttt{TIPO\_CONJ}_{MR} \wedge \ \texttt{REST\_CONST}_{MR} \wedge \ \texttt{INV}_{MR} \wedge \ \texttt{PRE}_{MR-1}$$
$$\Rightarrow \ \texttt{PRE}_R \wedge \ [\texttt{SUBS}_R] \neg \ [\texttt{SUBS}_{R-1}] \neg \ \texttt{INV}_R$$

$$\Leftrightarrow$$

$$\texttt{OBJETOS} \in \mathbb{P}_1(\mathbb{Z}) \wedge \ \texttt{lance\_min} \in \mathbb{N}_1$$
$$\wedge \ \texttt{RESP} \in \mathbb{P}_1(\mathbb{Z}) \wedge \ \texttt{RESP} = \{\texttt{aberto}, \texttt{fechado}, \texttt{inexistente}\}$$
$$\wedge \ \texttt{aberto} \neq \texttt{fechado} \wedge \texttt{aberto} \neq \texttt{inexistente}$$
$$\wedge \ \texttt{fechado} \neq \texttt{inexistente} \wedge \texttt{card(OBJETOS)} \in \mathbb{N}_1$$
$$\wedge \ \texttt{sessao} \in \texttt{OBJETOS} \leftrightarrow \mathbb{N}_1 \wedge \ \texttt{leiloados} \subseteq \texttt{dom(sessao)}$$
$$\wedge \ \texttt{rsessao} \in \texttt{OBJETOS} \nrightarrow \mathbb{N}_1 \wedge \ \texttt{dom(rsessao)} = \texttt{dom(sessao)}$$
$$\wedge \forall \texttt{x} \bullet (\texttt{x} \in \texttt{dom(rsessao)} \Rightarrow \texttt{rsessao(x)} = \texttt{max(sessao[\{x\}])})$$

$$
\begin{array}{l}
\quad \wedge\ \mathtt{rleiloados} = \mathtt{leiloados} \wedge \mathtt{obj} \in \mathtt{OBJETOS} \setminus \mathtt{dom(sessao)} \\
\quad \Rightarrow\ \mathtt{obj} \in \mathtt{OBJETOS} \setminus \mathtt{dom(rsessao)} \\
\qquad \wedge\ [\mathtt{rsessao} := \mathtt{rsessao} \oplus \{\mathtt{obj} \mapsto \mathtt{lance\_min}\}] \\
\qquad\quad \neg\, [\mathtt{sessao} := \mathtt{sessao} \cup \mathtt{obj} \mapsto \mathtt{lance\_min}] \\
\qquad\qquad \neg\, (\mathtt{rsessao} \in \mathtt{OBJETOS} \nrightarrow \mathbb{N}_1 \wedge\ \mathtt{dom(rsessao)} = \mathtt{dom(sessao)} \\
\qquad\qquad\quad \wedge\, \forall \mathtt{x} \bullet (\mathtt{x} \in \mathtt{dom(rsessao)} \\
\qquad\qquad\qquad\qquad \Rightarrow\ \mathtt{rsessao(x)} = \mathtt{max(sessao[\{x\}])}) \\
\qquad\qquad\quad \wedge\ \mathtt{rleiloados} = \mathtt{leiloados}) \\
\Leftrightarrow \\
\quad \mathtt{lance\_min} \in \mathbb{N}_1 \\
\quad \wedge\ \mathtt{sessao} \in \mathtt{OBJETOS} \leftrightarrow \mathbb{N}_1 \\
\quad \wedge\ \mathtt{rsessao} \in \mathtt{OBJETOS} \nrightarrow \mathbb{N}_1 \wedge\ \mathtt{dom(rsessao)} = \mathtt{dom(sessao)} \\
\quad \wedge\, \forall \mathtt{x} \bullet (\mathtt{x} \in \mathtt{dom(rsessao)} \Rightarrow \mathtt{rsessao(x)} = \mathtt{max(sessao[\{x\}])}) \\
\quad \wedge\ \mathtt{obj} \in \mathtt{OBJETOS} \setminus \mathtt{dom(sessao)} \\
\quad \Rightarrow\ \mathtt{obj} \in \mathtt{OBJETOS} \setminus \mathtt{dom(rsessao)} \\
\qquad \wedge\ \mathtt{rsessao} \oplus \{\mathtt{obj} \mapsto \mathtt{lance\_min}\} \in \mathtt{OBJETOS} \nrightarrow \mathbb{N}_1 \\
\qquad \wedge\ \mathtt{dom(rsessao} \oplus \{\mathtt{obj} \mapsto \mathtt{lance\_min}\}) \\
\qquad\quad = \mathtt{dom(sessao} \cup \mathtt{obj} \mapsto \mathtt{lance\_min}) \\
\qquad \wedge\, \forall \mathtt{x} \bullet (\mathtt{x} \in \mathtt{dom(rsessao} \oplus \{\mathtt{obj} \mapsto \mathtt{lance\_min}\}) \\
\qquad\qquad\quad \Rightarrow\ (\mathtt{rsessao} \oplus \{\mathtt{obj} \mapsto \mathtt{lance\_min}\})(\mathtt{x}) \\
\qquad\qquad\qquad = \mathtt{max((sessao} \cup \{\mathtt{obj} \mapsto \mathtt{lance\_min}\})[\{\mathtt{x}\}])
\end{array}
$$

Operação `propor`:

$$
\begin{array}{l}
\quad \mathtt{C}_M \wedge\ \mathtt{D}_{MR} \wedge\ \mathtt{INV}_{MR} \wedge\ \mathtt{PRE}_{MR-1} \Rightarrow\ \mathtt{PRE}_R \wedge\ [\mathtt{SUBS}_R] \neg\, [\mathtt{SUBS}_{R-1}] \neg\, \mathtt{INV}_R \\
\Leftrightarrow \\
\quad \mathtt{C}_M \wedge\ \mathtt{TIPO\_PARAM}_{MR} \wedge\ \mathtt{TIPO\_CONJ}_{MR} \wedge\ \mathtt{REST\_CONST}_{MR} \wedge\ \mathtt{INV}_{MR} \wedge\ \mathtt{PRE}_{MR-1} \\
\quad \Rightarrow\ \mathtt{PRE}_R \wedge\ [\mathtt{SUBS}_R] \neg\, [\mathtt{SUBS}_{R-1}] \neg\, \mathtt{INV}_R \\
\Leftrightarrow \\
\quad \mathtt{OBJETOS} \in \mathbb{P}_1(\mathbb{Z}) \wedge\ \mathtt{lance\_min} \in \mathbb{N}_1 \\
\quad \wedge\ \mathtt{RESP} \in \mathbb{P}_1(\mathbb{Z}) \wedge\ \mathtt{RESP} = \{\mathtt{aberto}, \mathtt{fechado}, \mathtt{inexistente}\}
\end{array}
$$

$$
\begin{array}{l}
\quad \wedge\ \texttt{aberto} \neq \texttt{fechado} \wedge \texttt{aberto} \neq \texttt{inexistente} \\
\quad \wedge\ \texttt{fechado} \neq \texttt{inexistente} \wedge \texttt{card(OBJETOS)} \in \mathbb{N}_1 \\
\quad \wedge\ \texttt{sessao} \in \texttt{OBJETOS} \leftrightarrow \mathbb{N}_1 \wedge\ \texttt{leiloados} \subseteq \texttt{dom(sessao)} \\
\quad \wedge\ \texttt{rsessao} \in \texttt{OBJETOS} \nrightarrow \mathbb{N}_1 \wedge\ \texttt{dom(rsessao)} = \texttt{dom(sessao)} \\
\quad \wedge \forall \texttt{x} \bullet (\texttt{x} \in \texttt{dom(rsessao)} \Rightarrow \texttt{rsessao(x)} = \texttt{max(sessao[\{x\}])}) \\
\quad \wedge\ \texttt{rleiloados} = \texttt{leiloados} \wedge\ \texttt{lance} \in \mathbb{N}_1 \\
\quad \wedge\ \texttt{obj} \in \texttt{dom(sessao)} \setminus \texttt{leiloados} \wedge\ \texttt{lance} > \texttt{max(sessao[\{obj\}]} \\
\quad \wedge\ \texttt{obj} \in \texttt{dom(rsessao)} \setminus \texttt{rleiloados} \wedge\ \texttt{lance} > \texttt{rsessao(obj)} \\
\quad \Rightarrow\ \texttt{lance} \in \mathbb{N}_1 \wedge\ \texttt{obj} \in \texttt{dom(rsessao)} \setminus \texttt{rleiloados} \\
\qquad \wedge\ \texttt{lance} > \texttt{rsessao(obj)} \\
\qquad \wedge\ [\texttt{rsessao} := \texttt{rsessao} \oplus \{\texttt{obj} \mapsto \texttt{lance}\}] \\
\qquad\quad \neg\, [\texttt{sessao} := \texttt{sessao} \cup \{\texttt{obj} \mapsto \texttt{lance}\}] \\
\qquad\qquad \neg\, (\texttt{rsessao} \in \texttt{OBJETOS} \nrightarrow \mathbb{N}_1 \wedge\ \texttt{dom(rsessao)} = \texttt{dom(sessao)} \\
\qquad\qquad\quad \wedge \forall \texttt{x} \bullet (\texttt{x} \in \texttt{dom(rsessao)} \\
\qquad\qquad\qquad\qquad \Rightarrow\ \texttt{rsessao(x)} = \texttt{max(sessao[\{x\}])}) \\
\qquad\qquad\quad \wedge\ \texttt{rleiloados} = \texttt{leiloados}) \\
\Leftrightarrow \\
\quad \texttt{sessao} \in \texttt{OBJETOS} \leftrightarrow \mathbb{N}_1 \\
\quad \wedge\ \texttt{rsessao} \in \texttt{OBJETOS} \nrightarrow \mathbb{N}_1 \wedge\ \texttt{dom(rsessao)} = \texttt{dom(sessao)} \\
\quad \wedge \forall \texttt{x} \bullet (\texttt{x} \in \texttt{dom(rsessao)} \Rightarrow \texttt{rsessao(x)} = \texttt{max(sessao[\{x\}])}) \\
\quad \wedge\ \texttt{rleiloados} = \texttt{leiloados} \wedge\ \texttt{obj} \in \texttt{dom(sessao)} \setminus \texttt{leiloados} \\
\quad \wedge\ \texttt{lance} > \texttt{max(sessao[\{obj\}]} \wedge\ \texttt{obj} \in \texttt{dom(rsessao)} \setminus \texttt{rleiloados} \\
\quad \Rightarrow\ \texttt{rsessao} \oplus \{\texttt{obj} \mapsto \texttt{lance}\} \in \texttt{OBJETOS} \nrightarrow \mathbb{N}_1 \\
\qquad \wedge\ \texttt{dom(rsessao} \oplus \{\texttt{obj} \mapsto \texttt{lance}\}) \\
\qquad\quad = \texttt{dom(sessao} \cup \{\texttt{obj} \mapsto \texttt{lance}\}) \\
\qquad \wedge \forall \texttt{x} \bullet (\texttt{x} \in \texttt{dom(rsessao} \oplus \{\texttt{obj} \mapsto \texttt{lance}\}) \\
\qquad\qquad \Rightarrow\ (\texttt{rsessao} \oplus \{\texttt{obj} \mapsto \texttt{lance}\})(\texttt{x}) \\
\qquad\qquad\quad = \texttt{max((sessao} \cup \{\texttt{obj} \mapsto \texttt{lance}\})[\{\texttt{x}\}])
\end{array}
$$

Operação `encerrar`:

$$
\begin{array}{l}
\quad \mathtt{C}_M \wedge\ \mathtt{D}_{MR} \wedge\ \mathtt{INV}_{MR} \wedge\ \mathtt{PRE}_{MR-1} \Rightarrow\ \mathtt{PRE}_R \wedge\ [\mathtt{SUBS}_R] \neg\, [\mathtt{SUBS}_{R-1}] \neg\, \mathtt{INV}_R \\
\Leftrightarrow \\
\quad \mathtt{C}_M \wedge\ \mathtt{TIPO\_PARAM}_{MR} \wedge\ \mathtt{TIPO\_CONJ}_{MR} \wedge\ \mathtt{REST\_CONST}_{MR} \wedge\ \mathtt{INV}_{MR} \wedge\ \mathtt{PRE}_{MR-1} \\
\quad \Rightarrow\ \mathtt{PRE}_R \wedge\ [\mathtt{SUBS}_R] \neg\, [\mathtt{SUBS}_{R-1}] \neg\, \mathtt{INV}_R \\
\Leftrightarrow \\
\quad \mathtt{OBJETOS} \in \mathbb{P}_1(\mathbb{Z}) \wedge\ \mathtt{lance\_min} \in \mathbb{N}_1 \\
\quad \wedge\ \mathtt{RESP} \in \mathbb{P}_1(\mathbb{Z}) \wedge\ \mathtt{RESP} = \{\mathtt{aberto}, \mathtt{fechado}, \mathtt{inexistente}\} \\
\quad \wedge\ \mathtt{aberto} \neq \mathtt{fechado} \wedge\ \mathtt{aberto} \neq \mathtt{inexistente} \\
\quad \wedge\ \mathtt{fechado} \neq \mathtt{inexistente} \wedge\ \mathtt{card}(\mathtt{OBJETOS}) \in \mathbb{N}_1 \\
\quad \wedge\ \mathtt{sessao} \in \mathtt{OBJETOS} \leftrightarrow \mathbb{N}_1 \wedge\ \mathtt{leiloados} \subseteq \mathtt{dom}(\mathtt{sessao}) \\
\quad \wedge\ \mathtt{rsessao} \in \mathtt{OBJETOS} \nrightarrow \mathbb{N}_1 \wedge\ \mathtt{dom}(\mathtt{rsessao}) = \mathtt{dom}(\mathtt{sessao}) \\
\quad \wedge \forall \mathtt{x} \bullet (\mathtt{x} \in \mathtt{dom}(\mathtt{rsessao}) \Rightarrow \mathtt{rsessao}(\mathtt{x}) = \mathtt{max}(\mathtt{sessao}[\{\mathtt{x}\}])) \\
\quad \wedge\ \mathtt{rleiloados} = \mathtt{leiloados} \wedge\ \mathtt{obj} \in \mathtt{dom}(\mathtt{sessao}) \setminus \mathtt{leiloados} \\
\quad \wedge\ \mathtt{obj} \in \mathtt{dom}(\mathtt{rsessao}) \setminus \mathtt{rleiloados} \\
\quad \Rightarrow\ \mathtt{obj} \in \mathtt{dom}(\mathtt{rsessao}) \setminus \mathtt{rleiloados} \\
\qquad \wedge\ [\mathtt{rleiloados} := \mathtt{rleiloados} \cup \{\mathtt{obj}\}] \\
\qquad\quad \neg\, [\mathtt{leiloados} := \mathtt{leiloados} \cup \{\mathtt{obj}\}] \\
\qquad\qquad \neg\, (\mathtt{rsessao} \in \mathtt{OBJETOS} \nrightarrow \mathbb{N}_1 \wedge\ \mathtt{dom}(\mathtt{rsessao}) = \mathtt{dom}(\mathtt{sessao}) \\
\qquad\qquad\quad \wedge \forall \mathtt{x} \bullet (\mathtt{x} \in \mathtt{dom}(\mathtt{rsessao}) \\
\qquad\qquad\qquad\quad \Rightarrow \mathtt{rsessao}(\mathtt{x}) = \mathtt{max}(\mathtt{sessao}[\{\mathtt{x}\}])) \\
\qquad\qquad\quad \wedge\ \mathtt{rleiloados} = \mathtt{leiloados}) \\
\Leftrightarrow \\
\quad \mathtt{sessao} \in \mathtt{OBJETOS} \leftrightarrow \mathbb{N}_1 \\
\quad \wedge\ \mathtt{rsessao} \in \mathtt{OBJETOS} \nrightarrow \mathbb{N}_1 \wedge\ \mathtt{dom}(\mathtt{rsessao}) = \mathtt{dom}(\mathtt{sessao}) \\
\quad \wedge \forall \mathtt{x} \bullet (\mathtt{x} \in \mathtt{dom}(\mathtt{rsessao}) \Rightarrow \mathtt{rsessao}(\mathtt{x}) = \mathtt{max}(\mathtt{sessao}[\{\mathtt{x}\}])) \\
\quad \wedge\ \mathtt{rleiloados} = \mathtt{leiloados} \wedge\ \mathtt{obj} \in \mathtt{dom}(\mathtt{sessao}) \setminus \mathtt{leiloados} \\
\quad \wedge\ \mathtt{obj} \in \mathtt{dom}(\mathtt{rsessao}) \setminus \mathtt{rleiloados} \\
\quad \Rightarrow\ \mathtt{rleiloados} \cup \{\mathtt{obj}\} = \mathtt{leiloados} \cup \{\mathtt{obj}\}
\end{array}
$$

Operação `consultar`:

$C_M \wedge D_{MR} \wedge INV_{MR} \wedge PRE_{MR-1} \Rightarrow PRE_R \wedge [SUBS'_R] \neg [SUBS_{R-1}] \neg (INV_R \wedge y' = y)$

$\Leftrightarrow$

$C_M \wedge TIPO\_PARAM_{MR} \wedge TIPO\_CONJ_{MR} \wedge REST\_CONST_{MR} \wedge INV_{MR} \wedge PRE_{MR-1}$
$\Rightarrow PRE_R \wedge [SUBS'_R] \neg [SUBS_{R-1}] \neg (INV_R \wedge y' = y)$

$\Leftrightarrow$

$OBJETOS \in \mathbb{P}_1(\mathbb{Z}) \wedge lance\_min \in \mathbb{N}_1$
$\wedge RESP \in \mathbb{P}_1(\mathbb{Z}) \wedge RESP = \{aberto, fechado, inexistente\}$
$\wedge aberto \neq fechado \wedge aberto \neq inexistente$
$\wedge fechado \neq inexistente \wedge card(OBJETOS) \in \mathbb{N}_1$
$\wedge sessao \in OBJETOS \leftrightarrow \mathbb{N}_1 \wedge leiloados \subseteq dom(sessao)$
$\wedge rsessao \in OBJETOS \nrightarrow \mathbb{N}_1 \wedge dom(rsessao) = dom(sessao)$
$\wedge \forall x \bullet (x \in dom(rsessao) \Rightarrow rsessao(x) = max(sessao[\{x\}]))$
$\wedge rleiloados = leiloados \wedge obj \in dom(sessao)$
$\Rightarrow obj \in dom(rsessao)$
$\wedge$ [m' := rsessao(obj);
IF obj $\notin$ rleiloados THEN resp' := aberto
ELSE resp' := fechado END]
$\neg$ [m := max(sessao[{obj}]);
IF obj $\notin$ leiloados THEN resp := aberto
ELSE resp := fechado END]
$\neg$ (rsessao $\in$ OBJETOS $\nrightarrow \mathbb{N}_1 \wedge$ dom(rsessao) = dom(sessao)
$\wedge \forall x \bullet (x \in dom(rsessao)$
$\Rightarrow rsessao(x) = max(sessao[\{x\}]))$
$\wedge$ rleiloados = leiloados
$\wedge$ m' = m $\wedge$ resp' = resp)

$\Leftrightarrow$

$sessao \in OBJETOS \leftrightarrow \mathbb{N}_1 \wedge leiloados \subseteq dom(sessao)$
$\wedge rsessao \in OBJETOS \nrightarrow \mathbb{N}_1 \wedge dom(rsessao) = dom(sessao)$
$\wedge \forall x \bullet (x \in dom(rsessao) \Rightarrow rsessao(x) = max(sessao[\{x\}]))$

```
  ∧ rleiloados = leiloados ∧ obj ∈ dom(sessao)
  ⇒ obj ∈ dom(rsessao)
    ∧ (obj ∉ rleiloados ⇒ rsessao(obj) = max(sessao[{obj}]))
    ∧ (obj ∈ rleiloados ⇒ rsessao(obj) = max(sessao[{obj}]))
```

A.3 Especificação e Obrigações de Prova da Implementação

O refinamento `leilao_ref` pode ser refinado através da seguinte implementação:

```
IMPLEMENTATION leilao_imp (OBJETOS)
REFINES leilao_ref
VALUES lance_min = 100
IMPORTS leilaoVar (OBJETOS)
INVARIANT sessaoVar = rsessao ∧ leiloadosVar = rleiloados
OPERATIONS
  iniciar(obj) ≙
    VAR st IN
      st ⟵ em_sessao(obj)
      IF obj ∈ OBJETOS ∧ st = FALSE
      THEN set_sessao(lance_min, obj)
      ELSE skip
      END
    END;
  propor(lance, obj) ≙
    VAR st1, st2, nn IN
      st1 ⟵ em_sessao(obj);
      st2 ⟵ nao_leiloado(obj);
      nn ⟵ get(obj);
```

```
      IF st1 = TRUE ∧ st2 = TRUE
      THEN
        IF lance > nn
        THEN set_sessao(lance, obj)
        ELSE skip
        END
      END
    END;
  encerrar(obj) ≙
    VAR st1, st2 IN
      st1 ⟵ em_sessao(obj); st2 ⟵ nao_leiloado(obj);
      IF st1 = TRUE ∧ st2 = TRUE
      THEN set_leiloados(obj)
      ELSE skip
      END
    END;
  resp, m ⟵ consultar(obj) ≙
    VAR st1, st2 IN
      st1 ⟵ em_sessao(obj); st2 ⟵ nao_leiloado(obj);
      IF st1 = TRUE
      THEN m ⟵ get(obj);
        IF st2 = TRUE
        THEN resp := aberto
        ELSE resp := fechado
        END
      ELSE m := 0; resp := inexistente
      END
    END
END
```

A especificação da máquina importada pela implementação `leilao_imp` é dada por:

```
  MACHINE leilaoVar (OBJETOS)
  VARIABLES sessaoVar, leiloadosVar
  INVARIANT sessaoVar ∈ OBJETOS ⇸ ℕ1 ∧ leiloadosVar ⊆
dom(sessaoVar)
  INITIALISATION sessaoVar, leiloadosVar := ∅, ∅
  OPERATIONS
    set_sessao(lance, obj) ≙
      PRE obj ∈ OBJETOS ∧ lance ∈ ℕ1
      THEN sessaoVar := sessaoVar ⊕ {obj ↦ lance_min}
      END;

    set_leiloados(obj) ≙
      PRE obj ∈ dom(sessaoVar)
      THEN leiloadosVar := leiloadosVar ∪ {obj}
      END;

    resp ⟵ em_sessao(obj) ≙
      PRE obj ∈ OBJETOS
      THEN resp := bool(obj ∈ dom(sessaoVar))
      END;

    resp ⟵ nao_leiloado(obj) ≙
      PRE obj ∈ OBJETOS
      THEN resp := bool(obj ∉ leiloadosVar)
      END;

    resp ⟵ get(obj) ≙
      PRE obj ∈ dom(sessaoVar)
      THEN resp := sessaoVar(obj)
      END
```

```
END
```

Para a verificação da implementação são geradas as seguintes obrigações de prova:

Inicialização:

$$
\begin{array}{l}
\mathtt{C}_M \wedge\ \mathtt{D}_{MI} \wedge\ \mathtt{D}_{imp} \Rightarrow\ [\mathtt{SUBS}_{imp}\ ;\ \mathtt{SUBS}_I] \neg\, [\mathtt{SUBS}_R] \neg\, \mathtt{INV}_I \\
\Leftrightarrow \\
\mathtt{C}_M \wedge\ \mathtt{TIPO\_PARAM}_{MI} \wedge\ \mathtt{TIPO\_CONJ}_{MI} \wedge\ \mathtt{REST\_CONST}_{MI} \wedge \\
\mathtt{TIPO\_PARAM}_{imp} \wedge\ \mathtt{TIPO\_CONJ}_{imp} \wedge\ \mathtt{REST\_CONST}_{imp} \\
\Rightarrow\ [\mathtt{SUBS}_{imp}\ ;\ \mathtt{SUBS}_I] \neg\, [\mathtt{SUBS}_R] \neg\, \mathtt{INV}_I \\
\Leftrightarrow \\
\mathtt{OBJETOS} \in \mathbb{P}_1(\mathbb{Z}) \wedge\ \mathtt{lance\_min} \in \mathbb{N}_1 \\
\wedge\ \mathtt{RESP} \in \mathbb{P}_1(\mathbb{Z}) \wedge\ \mathtt{RESP} = \{\mathtt{aberto}, \mathtt{fechado}, \mathtt{inexistente}\} \\
\wedge\ \mathtt{aberto} \neq \mathtt{fechado} \wedge\ \mathtt{aberto} \neq \mathtt{inexistente} \\
\wedge\ \mathtt{fechado} \neq \mathtt{inexistente} \wedge\ \mathtt{card}(\mathtt{OBJETOS}) \in \mathbb{N}_1 \\
\Rightarrow\ [\mathtt{sessaoVar}, \mathtt{leiloadosVar} := \varnothing, \varnothing] \\
\quad \neg\, [\mathtt{rsessao}, \mathtt{rleiloados} := \varnothing, \varnothing] \\
\qquad \neg\, (\mathtt{sessaoVar} = \mathtt{rsessao} \wedge\ \mathtt{leiloadosVar} = \mathtt{rleiloados}) \\
\Leftrightarrow \\
\varnothing = \varnothing
\end{array}
$$

Operação `iniciar`:

$$
\begin{array}{l}
\mathtt{C}_M \wedge\ \mathtt{D}_{MI} \wedge\ \mathtt{D}_{imp} \wedge\ \mathtt{INV}_{MI} \wedge\ \mathtt{PRE}_{MR} \Rightarrow\ \mathtt{PRE}_I \wedge\ [\mathtt{SUBS}_I] \neg\, [\mathtt{SUBS}_R] \neg\, \mathtt{INV}_I \\
\Leftrightarrow \\
\mathtt{C}_M \wedge\ \mathtt{TIPO\_PARAM}_{MI} \wedge\ \mathtt{TIPO\_CONJ}_{MI} \wedge\ \mathtt{REST\_CONST}_{MI} \wedge \\
\mathtt{TIPO\_PARAM}_{imp} \wedge\ \mathtt{TIPO\_CONJ}_{imp} \wedge\ \mathtt{REST\_CONST}_{imp} \wedge\ \mathtt{INV}_{MI} \wedge\ \mathtt{PRE}_{MR} \\
\Rightarrow\ \mathtt{PRE}_I \wedge\ [\mathtt{SUBS}_I] \neg\, [\mathtt{SUBS}_R] \neg\, \mathtt{INV}_I \\
\Leftrightarrow \\
\mathtt{OBJETOS} \in \mathbb{P}_1(\mathbb{Z}) \wedge\ \mathtt{lance\_min} \in \mathbb{N}_1 \\
\wedge\ \mathtt{RESP} \in \mathbb{P}_1(\mathbb{Z}) \wedge\ \mathtt{RESP} = \{\mathtt{aberto}, \mathtt{fechado}, \mathtt{inexistente}\}
\end{array}
$$

$\wedge$ aberto $\neq$ fechado $\wedge$ aberto $\neq$ inexistente

$\wedge$ fechado $\neq$ inexistente $\wedge$ card(OBJETOS) $\in \mathbb{N}_1$

$\wedge$ sessao $\in$ OBJETOS $\leftrightarrow \mathbb{N}_1 \wedge$ leiloados $\subseteq$ dom(sessao)

$\wedge$ rsessao $\in$ OBJETOS $\nrightarrow \mathbb{N}_1 \wedge$ dom(rsessao) = dom(sessao)

$\wedge \forall x \bullet$ (x $\in$ dom(rsessao) $\Rightarrow$ rsessao(x) = max(sessao[{x}]))

$\wedge$ rleiloados = leiloados $\wedge$ sessaoVar = rsessao

$\wedge$ leiloadosVar = rleiloados $\wedge$ obj $\in$ OBJETOS $\setminus$ dom(sessao)

$\wedge$ obj $\in$ OBJETOS $\setminus$ dom(rsessao)

$\Rightarrow$ [VAR st

IN st $\longleftarrow$ em_sessao(obj);

IF obj $\in$ OBJETOS $\wedge$ st = FALSE

THEN set_sessao(lance_min, obj)

ELSE skip END

END]

$\neg$ [rsessao := rsessao $\oplus$ {obj $\mapsto$ lance_min}]

$\neg$ (sessaoVar = rsessao $\wedge$ leiloadosVar = rleiloados)

$\Leftrightarrow$

lance_min $\in \mathbb{N}_1$

$\wedge$ sessao $\in$ OBJETOS $\leftrightarrow \mathbb{N}_1$

$\wedge$ rsessao $\in$ OBJETOS $\nrightarrow \mathbb{N}_1 \wedge$ dom(rsessao) = dom(sessao)

$\wedge \forall x \bullet$ (x $\in$ dom(rsessao) $\Rightarrow$ rsessao(x) = max(sessao[{x}]))

$\wedge$ sessaoVar = rsessao $\wedge$ obj $\in$ OBJETOS $\setminus$ dom(sessao)

$\wedge$ obj $\in$ OBJETOS $\setminus$ dom(rsessao) $\wedge$ obj $\in$ OBJETOS lance_min $\in \mathbb{N}_1$

$\wedge$ obj $\notin$ dom(sessaoVar)

$\Rightarrow$ sessaoVar $\oplus$ {obj $\mapsto$ lance_min}

= rsessao $\oplus$ {obj $\mapsto$ lance_min}

Operação `propor`:

$C_M \wedge D_{MI} \wedge D_{imp} \wedge INV_{MI} \wedge PRE_{MR} \Rightarrow PRE_I \wedge [SUBS_I] \neg [SUBS_R] \neg INV_I$

$\Leftrightarrow$

$$
\begin{array}{l}
\quad \mathtt{C}_{M} \wedge\ \mathtt{TIPO\_PARAM}_{MI} \wedge\ \mathtt{TIPO\_CONJ}_{MI} \wedge\ \mathtt{REST\_CONST}_{MI} \wedge \\
\quad \mathtt{TIPO\_PARAM}_{imp} \wedge\ \mathtt{TIPO\_CONJ}_{imp} \wedge\ \mathtt{REST\_CONST}_{imp} \wedge\ \mathtt{INV}_{MI} \wedge\ \mathtt{PRE}_{MR} \\
\quad \Rightarrow\ \mathtt{PRE}_{I} \wedge\ [\mathtt{SUBS}_{I}] \neg\, [\mathtt{SUBS}_{R}] \neg\, \mathtt{INV}_{I} \\
\Leftrightarrow \\
\quad \mathtt{OBJETOS} \in \mathbb{P}_1(\mathbb{Z}) \wedge\ \mathtt{lance\_min} \in \mathbb{N}_1 \\
\quad \wedge\ \mathtt{RESP} \in \mathbb{P}_1(\mathbb{Z}) \wedge\ \mathtt{RESP} = \{\mathtt{aberto}, \mathtt{fechado}, \mathtt{inexistente}\} \\
\quad \wedge\ \mathtt{aberto} \neq \mathtt{fechado} \wedge\ \mathtt{aberto} \neq \mathtt{inexistente} \\
\quad \wedge\ \mathtt{fechado} \neq \mathtt{inexistente} \wedge\ \mathtt{card(OBJETOS)} \in \mathbb{N}_1 \\
\quad \wedge\ \mathtt{sessao} \in \mathtt{OBJETOS} \leftrightarrow \mathbb{N}_1 \wedge\ \mathtt{leiloados} \subseteq \mathtt{dom(sessao)} \\
\quad \wedge\ \mathtt{rsessao} \in \mathtt{OBJETOS} \nrightarrow \mathbb{N}_1 \wedge\ \mathtt{dom(rsessao)} = \mathtt{dom(sessao)} \\
\quad \wedge \forall \mathtt{x} \bullet (\mathtt{x} \in \mathtt{dom(rsessao)} \Rightarrow \mathtt{rsessao(x)} = \mathtt{max(sessao[\{x\}])}) \\
\quad \wedge\ \mathtt{rleiloados} = \mathtt{leiloados} \wedge\ \mathtt{sessaoVar} = \mathtt{rsessao} \\
\quad \wedge\ \mathtt{leiloadosVar} = \mathtt{rleiloados} \wedge\ \mathtt{lance} \in \mathbb{N}_1 \\
\quad \wedge\ \mathtt{obj} \in \mathtt{dom(sessao)} \setminus \mathtt{leiloados} \\
\quad \wedge\ \mathtt{obj} \in \mathtt{dom(rsessao)} \setminus \mathtt{rleiloados} \\
\qquad \Rightarrow [\mathtt{VAR\ st1, st2, nn} \\
\qquad\quad \mathtt{IN\ st1} \longleftarrow \mathtt{em\_sessao(obj);} \\
\qquad\qquad \mathtt{st2} \longleftarrow \mathtt{nao\_leiloado(obj);} \\
\qquad\qquad \mathtt{nn} \longleftarrow \mathtt{get(obj);} \\
\qquad\qquad \mathtt{IF\ st1 = TRUE} \wedge \mathtt{st2 = TRUE} \\
\qquad\qquad \mathtt{THEN\ IF\ lance > nn} \\
\qquad\qquad\quad \mathtt{THEN\ set\_sessao(lance, obj)} \\
\qquad\qquad\quad \mathtt{ELSE\ skip\ END} \\
\qquad\qquad \mathtt{ELSE\ skip\ END} \\
\qquad\quad \mathtt{END}] \\
\qquad\ \neg\, [\mathtt{rsessao} := \mathtt{rsessao} \oplus \{\mathtt{obj} \mapsto \mathtt{lance}\}] \\
\qquad\quad \neg\, (\mathtt{sessaoVar = rsessao} \wedge\ \mathtt{leiloadosVar = rleiloados}) \\
\Leftrightarrow \\
\quad \mathtt{sessao} \in \mathtt{OBJETOS} \leftrightarrow \mathbb{N}_1 \\
\quad \wedge\ \mathtt{rsessao} \in \mathtt{OBJETOS} \nrightarrow \mathbb{N}_1 \wedge\ \mathtt{dom(rsessao)} = \mathtt{dom(sessao)}
\end{array}
$$

$\wedge\, \forall x \bullet (x \in \mathrm{dom}(rsessao) \Rightarrow rsessao(x) = \max(sessao[\{x\}]))$

$\wedge\ sessaoVar = rsessao \wedge obj \in OBJETOS \setminus \mathrm{dom}(sessao)$

$\wedge\ obj \in OBJETOS \setminus \mathrm{dom}(rsessao) \wedge obj \in OBJETOS$

$\wedge\ obj \in \mathrm{dom}(sessaoVar) \wedge obj \notin leiloadosVar$

$\wedge\ lance > sessaoVar(obj) \wedge lance \in \mathbb{N}_1$

$\Rightarrow\ sessaoVar \oplus \{obj \mapsto lance\}$

$= rsessao \oplus \{obj \mapsto lance\}$

Operação `encerrar`:

$C_M \wedge\ D_{MI} \wedge\ D_{imp} \wedge\ INV_{MI} \wedge\ PRE_{MR} \Rightarrow\ PRE_I \wedge\ [SUBS_I] \neg\, [SUBS_R] \neg\, INV_I$

$\Leftrightarrow$

$C_M \wedge\ TIPO\_PARAM_{MI} \wedge\ TIPO\_CONJ_{MI} \wedge\ REST\_CONST_{MI} \wedge$

$TIPO\_PARAM_{imp} \wedge\ TIPO\_CONJ_{imp} \wedge\ REST\_CONST_{imp} \wedge\ INV_{MI} \wedge\ PRE_{MR}$

$\Rightarrow\ PRE_I \wedge\ [SUBS_I] \neg\, [SUBS_R] \neg\, INV_I$

$\Leftrightarrow$

$OBJETOS \in \mathbb{P}_1(\mathbb{Z}) \wedge\ lance\_min \in \mathbb{N}_1$

$\wedge\ RESP \in \mathbb{P}_1(\mathbb{Z}) \wedge\ RESP = \{aberto, fechado, inexistente\}$

$\wedge\ aberto \neq fechado \wedge aberto \neq inexistente$

$\wedge\ fechado \neq inexistente \wedge \mathrm{card}(OBJETOS) \in \mathbb{N}_1$

$\wedge\ sessao \in OBJETOS \leftrightarrow \mathbb{N}_1 \wedge\ leiloados \subseteq \mathrm{dom}(sessao)$

$\wedge\ rsessao \in OBJETOS \nrightarrow \mathbb{N}_1 \wedge\ \mathrm{dom}(rsessao) = \mathrm{dom}(sessao)$

$\wedge\, \forall x \bullet (x \in \mathrm{dom}(rsessao) \Rightarrow rsessao(x) = \max(sessao[\{x\}]))$

$\wedge\ rleiloados = leiloados \wedge sessaoVar = rsessao$

$\wedge\ leiloadosVar = rleiloados \wedge obj \in \mathrm{dom}(sessao) \setminus leiloados$

$\wedge\ obj \in \mathrm{dom}(rsessao) \setminus rleiloados$

$\Rightarrow$ [VAR st1, st2

IN st1 $\longleftarrow$ em_sessao(obj);

st2 $\longleftarrow$ nao_leiloado(obj);

IF st1 = TRUE $\wedge$ st2 = TRUE

THEN set_leiloados(obj)

$\texttt{ELSE skip END}$

$\texttt{END]}$

$\neg\,\texttt{[rleiloados := rleiloados} \cup \texttt{\{obj\}]}$

$\neg\,\texttt{(sessaoVar = rsessao} \wedge \texttt{leiloadosVar = rleiloados)}$

$\Leftrightarrow$

$\texttt{OBJETOS} \in \mathbb{P}_1(\mathbb{Z})$

$\wedge\ \texttt{sessao} \in \texttt{OBJETOS} \leftrightarrow \mathbb{N}_1 \wedge \texttt{leiloados} \subseteq \texttt{dom(sessao)}$

$\wedge\ \texttt{rsessao} \in \texttt{OBJETOS} \nrightarrow \mathbb{N}_1 \wedge \texttt{dom(rsessao) = dom(sessao)}$

$\wedge\ \forall \texttt{x} \bullet (\texttt{x} \in \texttt{dom(rsessao)} \Rightarrow \texttt{rsessao(x) = max(sessao[\{x\}]))}$

$\wedge\ \texttt{rleiloados = leiloados} \wedge \texttt{sessaoVar = rsessao}$

$\wedge\ \texttt{leiloadosVar = rleiloados} \wedge \texttt{obj} \in \texttt{dom(sessao)} \setminus \texttt{leiloados}$

$\wedge\ \texttt{obj} \in \texttt{dom(rsessao)} \setminus \texttt{rleiloados} \wedge \texttt{obj} \in \texttt{OBJETOS}$

$\Rightarrow\ \texttt{leiloadosVar} \cup \texttt{\{obj\} = rleiloados} \cup \texttt{\{obj\}}$

Operação `consultar`:

$\texttt{C}_M \wedge \texttt{D}_{MI} \wedge \texttt{D}_{imp} \wedge \texttt{INV}_{MI} \wedge \texttt{PRE}_{MR}$

$\Rightarrow\ \texttt{PRE}_I \wedge \texttt{[SUBS'}_I\texttt{]}\neg\,\texttt{[SUBS}_R\texttt{]}\neg\,\texttt{(INV}_I \wedge \texttt{y' = y)}$

$\Leftrightarrow$

$\texttt{C}_M \wedge \texttt{TIPO\_PARAM}_{MI} \wedge \texttt{TIPO\_CONJ}_{MI} \wedge \texttt{REST\_CONST}_{MI} \wedge$

$\texttt{TIPO\_PARAM}_{imp} \wedge \texttt{TIPO\_CONJ}_{imp} \wedge \texttt{REST\_CONST}_{imp} \wedge \texttt{INV}_{MI} \wedge \texttt{PRE}_{MR}$

$\Rightarrow\ \texttt{PRE}_I \wedge \texttt{[SUBS'}_I\texttt{]}\neg\,\texttt{[SUBS}_R\texttt{]}\neg\,\texttt{(INV}_I \wedge \texttt{y' = y)}$

$\Leftrightarrow$

$\texttt{OBJETOS} \in \mathbb{P}_1(\mathbb{Z}) \wedge \texttt{lance\_min} \in \mathbb{N}_1$

$\wedge\ \texttt{RESP} \in \mathbb{P}_1(\mathbb{Z}) \wedge \texttt{RESP = \{aberto, fechado, inexistente\}}$

$\wedge\ \texttt{aberto} \neq \texttt{fechado} \wedge \texttt{aberto} \neq \texttt{inexistente}$

$\wedge\ \texttt{fechado} \neq \texttt{inexistente} \wedge \texttt{card(OBJETOS)} \in \mathbb{N}_1$

$\wedge\ \texttt{sessao} \in \texttt{OBJETOS} \leftrightarrow \mathbb{N}_1 \wedge \texttt{leiloados} \subseteq \texttt{dom(sessao)}$

$\wedge\ \texttt{rsessao} \in \texttt{OBJETOS} \nrightarrow \mathbb{N}_1 \wedge \texttt{dom(rsessao) = dom(sessao)}$

$\wedge\ \forall \texttt{x} \bullet (\texttt{x} \in \texttt{dom(rsessao)} \Rightarrow \texttt{rsessao(x) = max(sessao[\{x\}]))}$

$\wedge\ \texttt{rleiloados = leiloados} \wedge \texttt{sessaoVar = rsessao}$

$\wedge$ leiloadosVar = rleiloados $\wedge$ obj $\in$ dom(sessao)

$\wedge$ obj $\in$ dom(rsessao)

$\Rightarrow$ [VAR st1, st2

IN st1 $\longleftarrow$ em_sessao(obj);

st2 $\longleftarrow$ nao_leiloado(obj);

IF st1 = TRUE

THEN m' $\longleftarrow$ get(obj);

IF st2 = TRUE

THEN resp' := aberto

ELSE resp' := fechado END

ELSE m' := 0; resp' := inexistente END

END]

$\neg$ [IF obj $\notin$ rleiloados THEN resp := aberto

ELSE resp := fechado END]

$\neg$ (sessaoVar = rsessao $\wedge$ leiloadosVar = rleiloados

$\wedge$ m' = m $\wedge$ resp' = resp)

$\Leftrightarrow$

OBJETOS $\in \mathbb{P}_1(\mathbb{Z}) \wedge$ lance_min $\in \mathbb{N}_1$

$\wedge$ RESP $\in \mathbb{P}_1(\mathbb{Z}) \wedge$ RESP = {aberto, fechado, inexistente}

$\wedge$ aberto $\neq$ fechado $\wedge$ aberto $\neq$ inexistente

$\wedge$ fechado $\neq$ inexistente $\wedge$ card(OBJETOS) $\in \mathbb{N}_1$

$\wedge$ sessao $\in$ OBJETOS $\leftrightarrow \mathbb{N}_1 \wedge$ leiloados $\subseteq$ dom(sessao)

$\wedge$ rsessao $\in$ OBJETOS $\nrightarrow \mathbb{N}_1 \wedge$ dom(rsessao) = dom(sessao)

$\wedge \forall$x $\bullet$ (x $\in$ dom(rsessao) $\Rightarrow$ rsessao(x) = max(sessao[{x}]))

$\wedge$ rleiloados = leiloados $\wedge$ sessaoVar = rsessao

$\wedge$ leiloadosVar = rleiloados $\wedge$ obj $\in$ dom(sessao)

$\wedge$ obj $\in$ dom(rsessao)

$\Rightarrow$ obj $\in$ OBJETOS

$\wedge$ (obj $\in$ dom(sessaoVar) $\wedge$ obj $\notin$ leiloadosVar

$\Rightarrow$ sessaoVar(obj) = rsessao(obj))

$$
\begin{aligned}
&\wedge\ (obj \in dom(sessaoVar) \wedge obj \in leiloadosVar \\
&\quad \Rightarrow sessaoVar(obj) = rsessao(obj) \wedge aberto = fechado) \\
&\wedge\ (obj \in dom(sessaoVar) \wedge obj \in leiloadosVar \\
&\quad \wedge\ obj \notin rleiloados) \\
&\quad \Rightarrow sessaoVar(obj) = rsessao(obj) \wedge fechado = aberto) \\
&\wedge\ (obj \notin dom(sessaoVar) \wedge obj \notin rleiloados \\
&\quad \Rightarrow 0 = rsessao(obj) \wedge inexistente = aberto) \\
&\wedge\ (obj \notin dom(sessaoVar) \wedge obj \in rleiloados \\
&\quad \Rightarrow 0 = rsessao(obj) \wedge inexistente = fechado)
\end{aligned}
$$

Apêndice B

Teoria Axiomática dos Conjuntos NBG para haRVey

```
((theory set
  (extends)
  (axioms
    ;;definição de conjunto unitário
    (forall e (in e (unitset e)))
    ;;definição de conjunto unitário
    (forall e1 e2 (<-> (in e1 (unitset e2)) (= e1 e2)))
    ;;axioma do conjunto universo
    (exists universe (forall e (in e universe)))
    ;;axioma do par
    (forall s1 s2
      (exists s
        (forall r (<-> (in r s) (or (in r s1) (in r s2)))))))
  ;;propriedades
  (prove unitset-subset
    (forall e s (<-> (in (unitset e) (power s)) (in e s))))
  (prove unitset-equal-unitset
```

```
   (forall e1 e2
     (<-> (seteq (unitset e1) (unitset e2)) (= e1 e2))))
 (prove emptyset-equal-unitset
   (forall e (nseteq emptyset (unitset e))))
 (prove unitset-equal-emptyset
   (forall e (nseteq (unitset e) emptyset)))
 (prove in-powerset-unitset
   (forall e s
     (<-> (in s (power (unitset e)))
          (or (seteq s (unitset e)) (seteq s emptyset)))))
);;end of theory set

(theory not_member
 (extends set)
 (axioms
   (forall e s (<-> (not(in e s)) (notin e s))))
);;end of theory not_member

(theory powerset
 (extends set)
 (axioms
   (forall s1 s2 e
     (<-> (in s1 (power s2)) (-> (in e s1) (in e s2)))))
 ;;propriedades
 (prove in-powerset-self
   (forall s (in s (power s))))
 (prove powerset-subset
   (forall s1 s2
     (<-> (in (power s1) (power (power s2)))
          (in s1 (power s2))))))
```

```
 (prove in-powerset-transitive
   (forall s1 s2 s3
     (-> (and (in s1 (power s2)) (in s2 (power s3)))
         (in s1 (power s3)))))
 (prove in-subset
   (forall e s1 s2
     (-> (and (in e s1) (in s1 (power s2))) (in e s2))))
);;end of theory powerset

(theory equality
 (extends set)
 (axioms
   ;;axioma da extensionalidade
   (forall s1 s2
     (<-> (seteq s1 s2) (forall e (<-> (in e s1) (in e s2))))))
 ;;propriedades
 (prove equality-symmetric
   (forall s (seteq s s)))
 (prove equality-reflexive
   (forall s1 s2 (-> (seteq s1 s2) (seteq s2 s1))))
 (prove equality-transitive
   (forall s1 s2 s3
     (-> (and (seteq s1 s2) (seteq s2 s3)) (seteq s1 s3))))
 (prove extensionality2
   (forall s1 s2
     (<-> (seteq s1 s2)
          (and (in s1 (power s2)) (in s2 (power s1))))))
 (prove extensionality3
   (forall s1 s2 s
     (<-> (seteq s1 s2)
```

```
          (and (forall r1 (-> (in r1 s1) (in r1 s2)))
               (forall r2 (-> (in r2 s2) (in r2 s1)))))))
);;end of theory equality

(theory inequality
 (extends set)
 (axioms
   (forall s1 s2 (<-> (nseteq s1 s2) (not(seteq s1 s2)))))
);;end of theory inequality

(theory emptyset
 (extends set not_member powerset equality)
 (axioms
   ;;axioma do conjunto vazio
   (forall e (notin e emptyset))
   (seteq (power emptyset) (unitset emptyset)))
 ;;propriedades
 (prove emptyset-subset
   (forall s (in emptyset (power s))))
 (prove non-emptyset-has-member
   (forall s (or (seteq s emptyset) (exists e (in e s)))))
);;end of theory emptyset

(theory subset
 (extends set)
 (axioms
   (forall s1 s2
     (<-> (subseteq s1 s2)
          (forall e (-> (in e s1) (in e s2))))))
 ;;propriedades
```

```
 (prove subset-self
   (forall s (subseteq s s)))
 (prove emptyset-set-subset
   (forall s (subseteq emptyset s)))
 (prove subset-transitive
   (forall s1 s2 s3
     (-> (and (subseteq s1 s2) (subseteq s2 s3))
          (subseteq s1 s3))))
);;end of theory subset

(theory not_subset
 (extends set not_member)
 (axioms
   (forall s1 s2 (<-> (nsubseteq s1 s2) (not(subseteq s1 s2)))))
);;end of theory not_subset

(theory not_emptyset_powerset
 (extends set powerset emptyset equality inequality subset)
 (axioms
   (forall s1 s2
        (<-> (seteq s1 (power1 s2))
             (and (subseteq s1 (power s2))
                  (nseteq s1 emptyset)))))
 ;;propriedades
 (prove in-power1
   (forall s1 s2
     (<-> (in s1 (power1 s2))
          (and (in s1 (power s2)) (nseteq s1 emptyset)))))
 (prove power1-emptyset
   (seteq (power1 emptyset) emptyset))
```

```
 (prove power1-unitset
   (forall e
     (seteq (power1 (unitset e)) (unitset (unitset e)))))
);;end of theory not_emptyset_powerset

(theory proper_subset
 (extends set equality inequality subset)
 (axioms
   (forall s1 s2
     (<-> (subset s1 s2)
          (and (subseteq s1 s2) (nseteq s1 s2)))))
 ;;propriedades
 (prove proper_subset-self
   (forall s (nsubset s s)))
 (prove emptyset-set-proper_subset
   (forall s (<-> (subset emptyset s) (nseteq s emptyset))))
);;end of theory proper_subset

(theory not_proper_subset
 (extends set equality subset)
 (axioms
   (forall s1 s2
     (<-> (nsubset s1 s2) (-> (subseteq s1 s2) (seteq s1 s2)))))
);;end of theory not_proper_subset

(theory union
 (extends set)
 (axioms
   (forall e s1 s2
     (<-> (in e (cup s1 s2)) (or (in e s1) (in e s2))))
```

```
;;propriedades
(prove union-self-right
  (forall s (seteq s (cup s s))))
(prove union-self-left
  (forall s (seteq (cup s s) s)))
(prove union-subset-left
  (forall s1 s2 (-> (subseteq s1 s2) (seteq (cup s1 s2) s2))))
(prove union-subset-right
  (forall s1 s2 (-> (subseteq s2 s1) (seteq (cup s1 s2) s1))))
(prove union-emptyset-right
  (forall s (seteq (cup s emptyset) s)))
(prove union-emptyset-left
  (forall s (seteq (cup emptyset s) s)))
(prove union-commutes
  (forall s1 s2 (seteq (cup s1 s2) (cup s2 s1))))
(prove union-associates
  (forall s1 s2 s3
    (seteq (cup (cup s1 s2) s3) (cup s1 (cup s2 s3)))))
(prove union-subset
  (forall s1 s2 s3
    (<-> (in (cup s1 s2) (power s3))
         (and (in s1 (power s3)) (in s2 (power s3))))))
(prove union-equal-emptyset-left
  (forall s1 s2
    (<-> (seteq (cup s1 s2) emptyset)
         (and (seteq s1 emptyset) (seteq s2 emptyset)))))
(prove union-equal-emptyset-right
  (forall s1 s2
    (<-> (seteq emptyset (cup s1 s2))
         (and (seteq s1 emptyset) (seteq s2 emptyset)))))
```

```
 (prove union-permutes
   (forall s1 s2 s3
     (seteq (cup s1 (cup s2 s3)) (cup s2 (cup s1 s3)))))
 (prove subset-union
   (forall s1 s2 s3
     (-> (or (in s1 (power s2)) (in s1 (power s3)))
          (in s1 (power (cup s2 s3))))))
 (prove in-union
   (forall e1 e2
     (-> (!= e1 e2)
          (forall s
            (<-> (in e2 (cup (unitset e1) s)) (in e2 s))))))
);;end of theory union

(theory intersection
 (extends set)
 (axioms
   (forall e s1 s2
     (<-> (in e (cap s1 s2)) (and (in e s1) (in e s2)))))
 ;;propriedades
 (prove intersection-subset-left
   (forall s1 s2 (-> (subseteq s1 s2) (seteq (cap s1 s2) s1))))
 (prove intersection-subset-right
   (forall s1 s2 (-> (subseteq s2 s1) (seteq (cap s1 s2) s2))))
 (prove intersection-emptyset-left
   (forall s (seteq (cap emptyset s) emptyset)))
 (prove intersection-emptyset-right
   (forall s (seteq (cap s emptyset) emptyset)))
 (prove unitset-intersection
   (forall e s
```

```
    (seteq (cap (unitset e) s)
           (ite (in e s) (unitset e) emptyset))))
(prove intersection-unitset
  (forall e s
    (seteq (cap s (unitset e))
           (ite (in e s) (unitset e) emptyset))))
(prove intersection-union-left
  (forall s1 s2 (seteq (cap s1 (cup s1 s2)) s1)))
(prove intersection-union-right
  (forall s1 s2 (seteq (cap (cup s1 s2) s1) s1)))
(prove intersection-self-right
  (forall s (seteq s (cap s s))))
(prove intersection-self-left
  (forall s (seteq (cap s s) s)))
(prove intersection-commutes
  (forall s1 s2 (seteq (cap s1 s2) (cap s2 s1))))
(prove intersection-associates
  (forall s1 s2 s3
    (seteq (cap (cap s1 s2) s3) (cap s1 (cap s2 s3)))))
(prove intersection-subset
  (forall s1 s2 s3
    (-> (or (subseteq s1 s3) (subseteq s2 s3))
        (subseteq (cap s1 s2) s3))))
(prove subset-intersection
  (forall s1 s2 s3
    (-> (in s1 (power (cap s2 s3)))
        (and (in s1 (power s2)) (in s1 (power s3))))))
(prove intersection-permutes
  (forall s1 s2 s3
    (seteq (cap s1 (cap s2 s3)) (cap s2 (cap s1 s3)))))
```

```
 (prove intersection-result
   (forall s1 s2 s3
     (-> (or (in (power s1) (power s3))
          (in (power s2) (power s3))) (in (cap s1 s2) s3))))
 (prove compute-intersection1
   (forall e s1 s2
     (-> (in e s2)
          (seteq (cap (cup (unitset e) s1) s2)
                 (cup (unitset e) (cap s1 s2))))))
 (prove compute-intersection2
   (forall e s1 s2
     (-> (notin e s2)
          (seteq (cap (cup (unitset e) s1) s2) (cap s1 s2)))))
);;end of theory intersection

(theory difference
 (extends set not_member)
 (axioms
   (forall e s1 s2
     (<-> (in e (setminus s1 s2))
          (and (in e s1) (notin e s2)))))
 ;;propriedades
 (prove difference-self
   (forall s (seteq (setminus s s) emptyset)))
 (prove difference-difference
   (forall s1 s2 s3
     (seteq (setminus (setminus s1 s2) s3)
            (setminus s1 (cup s2 s3)))))
 (prove difference-subset
   (forall s1 s2 s3
```

```
    (<-> (subseteq (setminus s1 s2) s3)
         (subseteq s1 (cup s2 s3)))))
 (prove difference-superset
  (forall s1 s2
   (-> (in s1 (power s2)) (seteq (setminus s1 s2) emptyset))))
 (prove difference-emptyset-left
  (forall s (seteq (setminus emptyset s) emptyset)))
 (prove difference-emptyset-right
  (forall s (seteq (setminus s emptyset) s)))
 (prove unitset-difference
  (forall e s
   (seteq (setminus (unitset e) s)
          (ite (in e s) emptyset (unitset e)))))
 (prove difference-result
  (forall s1 s2 s3
   (-> (in (power s1) (power s3)) (in (setminus s1 s2) s3))))
 (prove compute-difference1
  (forall e s1 s2
   (-> (in e s2)
        (seteq (setminus (cup (unitset e) s1) s2)
               (setminus s1 s2)))))
 (prove compute-difference2
  (forall e s1 s2
   (-> (notin e s2)
        (seteq (setminus (cup (unitset e) s1) s2)
               (cup (unitset e) (setminus s1 s2))))))
);;end of theory difference

(theory ordered_pair
 (extends set union equality)
```

```
 (axioms
   ;;(e1, e2) = e1, e1,e2
   (forall e1 e2
     (seteq (ord_pair e1 e2)
            (cup (unitset (unitset e1))
                 (unitset (cup (unitset e1) (unitset e2))))))
   (forall e1 e2 c1 c2
     (<-> (seteq (ord_pair e1 e2)
          (ord_pair c1 c2)) (and (= e1 c1) (= e2 c2)))))
)

(theory projection1
 (extends set ordered_pair)
 (axioms
   (forall e1 e2 (= (first (ord_pair e1 e2)) e1)))
)

(theory projection2
 (extends set ordered_pair)
 (axioms
   (forall e1 e2 (= (second (ord_pair e1 e2)) e2)))
)

(theory cross_product
 (extends set equality ordered_pair)
 (axioms
   (forall p s1 s2
     (<-> (in p (cross s1 s2))
          (exists e1 e2
            (and (in e1 s1) (in e2 s2)
```

```
                (seteq p (ord_pair e1 e2)))))))
 ;;propriedades
 (prove tuple-in-cross
   (forall e1 e2 s1 s2
     (<-> (in (ord_pair e1 e2)
          (cross s1 s2)) (and (in e1 s1) (in e2 s2)))))
 (prove cross-emptyset-left
   (forall s (seteq (cross emptyset s) emptyset)))
 (prove cross-emptyset-right
   (forall s (seteq (cross s emptyset) emptyset)))
 (prove cross-equal-emptyset
   (forall s1 s2
     (<-> (seteq (cross s1 s2) emptyset)
          (or (seteq s1 emptyset) (seteq s2 emptyset)))))
);;end of theory ordered_pair

(theory complement
 (extends set)
 (axioms
   ;;axioma da classe complemento
   (forall e s
     (<-> (and (in e universe) (notin e s)) (in e (compl s)))))
);;end of theory complement

(theory nbg
 (extends set equality inequality not_member emptyset union)
 (axioms
   ;;axioma das partes
   (forall s (exists s1 (seteq s1 (power s))))
   ;;axioma da união
```

```
   (forall s e
     (exists s1 (<-> (in e s1) (exists s2 (and (in e s2) (in s2
s))))))
   ;;axioma do infinito
   (exists s
     (and (in emptyset s)
          (forall e (-> (in e s) (in (unitset e) s)))))
   ;;axioma da regularidade
   (forall s
     (-> (nseteq s emptyset)
          (exists e
            (and (in e s) (nseteq (cup e s) emptyset))))))
 );;end of theory nbg
)


(prove distribute-cup-over-cap-right
  (forall s1 s2 s3
  (seteq (cup s1 (cap s2 s3)) (cap (cup s1 s2) (cup s1 s3)))))

(prove distribute-cup-over-cap-left
  (forall s1 s2 s3
    (seteq (cup (cap s1 s2) s3) (cap (cup s1 s3) (cup s2 s3)))))

(prove distribute-cap-over-cup-right
  (forall s1 s2 s3
    (seteq (cap s1 (cup s2 s3)) (cup (cap s1 s2) (cap s1 s3)))))

(prove distribute-cap-over-cup-left
  (forall s1 s2 s3
```

```
   (seteq (cap (cup s1 s2) s3) (cup (cap s1 s3) (cap s2 s3)))))

(prove distribute-difference-over-cup-right
  (forall s1 s2 s3
    (seteq (setminus s1 (cup s2 s3))
           (cap (setminus s1 s2) (setminus s1 s3)))))

(prove distribute-difference-over-cup-left
  (forall s1 s2 s3
    (seteq (setminus (cup s1 s2) s3)
           (cup (setminus s1 s3) (setminus s2 s3)))))

(prove distribute-difference-over-cap-right
  (forall s1 s2 s3
    (seteq (setminus s1 (cap s2 s3))
           (cup (setminus s1 s2) (setminus s1 s3)))))

(prove distribute-difference-over-cap-left
  (forall s1 s2 s3
    (seteq (setminus (cap s1 s2) s3)
           (cap (setminus s1 s3) (setminus s2 s3)))))

(prove distribute-difference-over-difference-right
  (forall s1 s2 s3
    (seteq (setminus s1 (setminus s2 s3))
           (cup (setminus s1 s2) (cap s1 s3)))))

(prove subset-intersection
  (forall s1 s2 s3
    (-> (subseteq s2 s3) (subseteq (cap s1 s2) (cap s1 s3)))))
```

```
(prove subset-union
  (forall s1 s2 s3
    (-> (subseteq s2 s3) (subseteq (cup s1 s2) (cup s1 s3)))))

(prove subset-difference1
  (forall s1 s2 s3
    (-> (subseteq s2 s3)
         (subseteq (setminus s2 s1) (setminus s3 s1)))))

(prove subset-difference2
  (forall s1 s2 s3
    (-> (subseteq s2 s3)
         (subseteq (setminus s1 s3) (setminus s1 s2)))))

(prove subset-over-union
  (forall s1 s2 (subseteq s1 (cup s1 s2))))

(prove subset-over-intersection
  (forall s1 s2 (subseteq (cap s1 s2) s1)))

(prove powerset-over-union
  (forall s1 s2 (in s1 (power (cup s1 s2)))))
```

Referências Bibliográficas

[1] Jean Raymond Abrial. *The B Book - Assigning Programs to Meanings*. Cambridge University Press, 1996.

[2] Alloy. *Alloy Community*. Disponível em: <http://allow.mit.edu>. Acesso em: 07 set 2008.

[3] Konstantine Arkoudas. *Type-omega DPLs*. 2001. Disponível em: <http://www.cag.lcs.mit.edu/~kostas/papers/ndlOmega.pdf>. Acesso em: 07 set 2008.

[4] B-Core. *The B-Toolkit*. Disponível em: <http://www.b-core.com/btoolkit.html>. Acesso em: 07 set 2008.

[5] Project Batcave. *Batcave - B AuTomated Computer-Aided Verification Environent*. Disponível em: <http://www.consiste.dimap.ufrn.br/projetos/batcave>. Acesso em: 07 set 2008.

[6] ClearSy. *Atelier b - version 3.6*. Disponível em: <http://www.atelierb.societe.com>. Acesso em: 07 set 2008.

[7] ClearSy. *B4free*. Disponível em: <http://www.b4free.com/index.php>. Acesso em: 07 set 2008.

[8] Edsger W. Dijkstra. *A Discipline of Programming*. Prentice-Hall Series in Automatic Computation. Prentice-Hall, 1976.

[9] Robert Nieuwenhuis e Albert Rubio. Paramodulation-based theorem proving. In *Handbook of Automated Reasoning*. p. 371-443. 2001. Disponível em: <http://citeseer.ist.psu.edu/nieuwenhuis01paramodulationbased.html>. Acesso em: 07 set 2008.

[10] John Alan Robinson e Andrei Voronkov, editor. *Handbook of Automated Reasoning (in 2 volumes)*. Elsevier e MIT Press, 2001.

[11] Fabrice Bouquet e Bruno Legeard e Fabien Peureux. CLPS-B - a constraint solver for b. In *TACAS '02: Proceedings of the 8th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, p. 188-204, Londres, Reino Unido, 2002. Springer-Verlag.

[12] S. Ranise e C. Tinelli. Satisfiability modulo theories (pre-print). *TRENDS and CONTROVERSIES–IEEE Magazine on Intelligent Systems*, 21(6), p. 71-81. 2006. Disponível em: <http://www.loria.fr/~ranise/pubs/IEEE-pp.pdf>. Acesso em: 07 set 2008.

[13] J.-R. Abrial e D. Cansell. *Click'n'Prove - Interactive Proofs Within Set Theory*. 2003. Versão 23. Disponível em: <http://www.loria.fr/~cansell/sld.balbu.pdf>. Acesso em: 07 set 2008.

[14] Greg Nelson e Derek C. Oppen. Simplification by cooperating decision procedures. *ACM Trans. Program. Lang. Syst.*, 1(2), p. 245-257. 1979. Disponível em: <http://portal.acm.org/citation.cfm?id=357079>. Acesso em: 07 set 2008.

[15] Jim Woodcock e Jim Davies (Author). *Using Z: Specification, Refinement, and Proof*. Prentice-Hall International Series in Computer Science, 1996.

[16] Leonid Mikhailov e Michael Butler. An approach to combining B and Alloy. In *ZB'2002 – Formal Specification and Development in Z and B*, p. 140-161. 2002. Disponível em: <http://eprints.ecs.soton.ac.uk/archive/00006232>. Acesso em: 07 set 2008.

[17] E-Prover. The E Equational Theorem Prover. Disponível em: <http://www4.informatik.tu-muenchen.de/~schulz/WORK/eprover.html>. Acesso em: 07 set 2008.

[18] D. Deharbe e S. Ranise. *Satisfiability Solving for Software Verification*. 2006. Disponível em: <http://www.loria.fr/~ranise/pubs/sttt-submitted.pdf>. Acesso em: 07 set 2008.

[19] M. J. C. Gordon e T. F. Melham. *Introduction to HOL: A theorem proving environment for higher order logic*. Cambridge University Press., Cambridge, Inglaterra, 1993.

[20] Abraham A. Fräenkel. The notion of "definite" and the independence os the axiom of choice. In *Jean van Heijenoort [42]*.

[21] haRVey. *haRVey - A cocktail of thories*. Disponível em: <http://harvey.loria.fr>. Acesso em: 07 set 2008.

[22] R Hähnle. Tableaux and related methods. In *A. Robinson e A. Voronkov, Handbook of Automated Reasoning*. Elsevier Science, p. 100-178. 2001. Disponível em: <http://www.consiste.dimap.ufrn.br/~david/hbar_files/hbar-tableaux.ps>. Acesso em: 07 set 2008.

[23] Carlos Ivorra. *La Axiomática de la Teoría de Conjuntos*. Disponível em: <http://www.uv.es/~ivorra/Libros/Axiomas.pdf>. Acesso em: 07 set 2008.

[24] Daniel Jackson. Micromodels of software: Modelling and analysis with alloy, 1991. Disponível em: <http://alloy.mit.edu/alloy2website/reference-manual.pdf>. Acesso em: 07 set 2008.

[25] Daniel Jackson. *Software Abstraction*. MIT Press, 2006.

[26] B. Tatibouët e A. Hammad J.C. Voisinet. jBTools: An experimental platform for the formal B method. In *Principles and Practice of Programming in Java (PPPJ'02)*, p. 137-140. 2002.

[27] jEdit. *jEdit - Programmer's Text Editor*. Disponível em: <http://www.jedit.org>. Acesso em: 07 set 2008.

[28] Antti-Juhani Kaijanaho. The formal method known as B and a sketch for its implementation. Master's thesis, University of Jyväskylä, 2002. Disponível em: <citeseer.ist.psu.edu/kaijanaho02formal.html>. Acesso em: 07 set 2008.

[29] George Logemann e Donald Loveland Martin Davis. A machine program for theorem-proving. *Communications of ACM*, 5(7), p. 394-397. 1962.

[30] Elliot Mendelson. *An introduction to mathematical logic*. Londres: Chapman & Hall, 4. ed, 1997.

[31] D. Nazareth. *A Polymorphic Sort System for Axiomatic Specification Languages*. 1995. Disponível em: <http://citeseer.ist.psu.edu/655092.html>. Acesso em: 07 set 2008.

[32] Lawrence C. Paulson. *Isabelle: A Generic Theorem Prover*. Springer, 1994. LNCS 828.

[33] Mark Saaltink. The Z/EVES system. In M. G. Hinchey e D. Till J. P. Bowen, editor, *ZUM'97: Z Formal Specification Notation*, volume 1212 of *Lecture Notes in Computer Science*. Springer-Verlag, p. 72-85. 1997. Disponível em: <http://citeseer.ist.psu.edu/102914.html>. Acesso em: 07 set 2008.

[34] Thoralf Skolem. Some remarks on axiomatized set theory. In *Jean van Heijenoort [42]*.

[35] J. M. Spivey. *The Z notation: a reference manual*. Prentice-Hall, Inc., Upper Saddle River, EUA, 1989. Disponível em: <http://spivey.oriel.ox.ac.uk/mike/zrm/zrm.pdf>. Acesso em: 07 set 2008.

[36] Dan Craigen *et al*. EVES: An overview. In *VDM '91: Proceedings of the 4th International Symposium of VDM Europe on Formal Software Development-Volume I*, p. 389-405, Londres, Reino Unido, 1991. Springer-Verlag.

[37] Jean-François Couchot *et al*. Proving and debugging set-based specifications. In Ana Cavalcanti e Patrícia Machado, editor, *VI Workshop de Métodos Formais (WMF'2003)*, p. 137-151. 2003. Best Paper Award. Disponível em: <http://www.consiste.dimap.ufrn.br/~david/files/wmf03a.pdf>. Acesso em: 07 set 2008.

[38] John McCarthy *et al*. *LISP 1.5 Programmer's Manual*. Cambridge, EUA, 1965. Disponível em: <http://www.softwarepreservation.org/projects/LISP/book/LISP%201.5%20 Programmers%20Manual.pdf>. Acesso em: 07 set 2008.

[39] Konstantine Arkoudas *et al*. Integrating model checking and theorem proving for relational reasoning. In *Seventh International Seminar on Relational Methods in Computer Science (RelMiCS 2003)*, volume 3015 of *Lecture Notes in Computer Science (LNCS)*, Malente, Alemanha, p. 21-33. 2003. Disponível em: <http://www.cag.lcs.mit.edu/~kostas/RELMICS2003Details.html>. Acesso em: 07 set 2008.

[40] L. Wos *et al*. *Automated Reasoning: Introduction and Applications*. McGraw-Hill, 2. ed, 1992.

[41] Sentot Kromodimoeljo *et al*. The EVES system. In *Functional Programming, Concurrency, Simulation and Automated Reasoning: International Lecture Series 1991-1992, McMaster University, Hamilton, Canadá*, p. 349-375, Londres, Reino Unido, 1993. Springer-Verlag.

[42] Jean van Heijenoort. *From Frege to Gödel: A Source Book in Mathematical Logic, 1879-1931*. Harvard University Press, Cambridge, EUA, 1967.

[43] John von Neumann. *An axiomatization of set theory*. 1925.

[44] XEmacs. *XEmacs: The next generation of Emacs*. Disponível em: <http://www.xemacs.org/>. Acesso em: 07 set 2008.

[45] Calogero G. Zarba. Combining sets with cardinals. *Journal of Automated Reasoning*, 34(1), 2005. Disponível em: <http://www.springerlink.com/index/N64G106101GV4271.pdf>. Acesso em: 07 set 2008.

[46] Ernst Zermelo. Investigations in the foundations of set theory I. In *Jean van Heijenoort [42]*.

Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo