Universidade Estadual de Londrina

WELLINGTON APARECIDO DELLA MURA

DETECÇÃO DE CONFLITOS EM CONTRATOS MULTILATERAIS

Londrina
2016

WELLINGTON APARECIDO DELLA MURA

DETECÇÃO DE CONFLITOS EM CONTRATOS MULTILATERAIS

Dissertação apresentada ao Programa de Mestrado em Ciências da Computação da Universidade Estadual de Londrina para obtenção do título de Mestre em Ciência da Computação.

Orientador: Prof. Dr. Adilson Luiz Bonifácio

Londrina
2016

Ficha de identificação da obra elaborada pelo autor, através do Programa de Geração Automática do Sistema de Bibliotecas da UEL

Della Mura, Wellington Aparecido.
Detecção de conflitos em contratos multilaterais / Wellington Aparecido Della Mura. - Londrina, 2016.
133 f. : il.

Orientador: Adilson Luiz Bonifácio.
Dissertação (Mestrado em Ciência da Computação) - Universidade Estadual de Londrina, Centro de Ciências Exatas, Programa de Pós-Graduação em Ciência da Computação, 2016.
Inclui bibliografia.

1. Contratos multilaterais - Tese. 2. Lógica Deôntica Relativizada - Tese. 3. Detecção de Conflitos - Tese. I. Bonifácio, Adilson Luiz. II. Universidade Estadual de Londrina. Centro de Ciências Exatas. Programa de Pós-Graduação em Ciência da Computação. III. Título.

WELLINGTON APARECIDO DELLA MURA

DETECÇÃO DE CONFLITOS EM CONTRATOS MULTILATERAIS

Dissertação apresentada ao Programa de Mestrado em Ciências da Computação da Universidade Estadual de Londrina para obtenção do título de Mestre em Ciência da Computação.

BANCA EXAMINADORA

Orientador: Prof. Dr. Adilson Luiz Bonifácio
Universidade Estadual de Londrina – UEL

Prof. Dr. Adenilson da Silva Simão
Universidade de São Paulo - USP

Prof. Dr. Daniel dos Santos Kaster
Universidade Estadual de Londrina – UEL

Prof. Dr. Evandro Baccarin
Universidade Estadual de Londrina – UEL

Londrina, 21 de setembro de 2016.

A minha esposa Jaciani pelo companheirismo, compreensão e carinho incondicionais.
A minha Família que sempre está ao meu lado.
Ao meu orientador pela dedicação, apoio e reconhecimento.

AGRADECIMENTOS

A Deus pela oportunidade de viver nesse mundo e permitir que eu deixe minha pequena contribuição para todos que virão depois de mim.

Ao meu orientador, Dr. Adilson Luiz Bonifácio, por todo apoio, compreensão e dedicação ao longo do mestrado que tanto contribuíram para meu crescimento profissional e pessoal.

Ao professor Dr. Evandro Baccarin, pelo constante auxílio nos assuntos relacionados a contratos multilaterais e pelos direcionamentos desde a minha qualificação.

Aos professores Dr. Daniel dos Santos Kaster e Dr. Adenilso da Silva Simão pelos direcionamentos e sugestões que tanto colaboraram para o enriquecimento deste trabalho.

À minha esposa Jaciani pela compreensão, força e amor nos momentos difíceis ao longo de tantos anos juntos e por essa nova e abençoada vida que está por vir.

A meus pais Maria Helena e Alberto pelas oportunidades e exemplo de trabalho e honestidade todos os dias da minha vida além do apoio que sempre me deram para buscar novos horizontes e melhorar cada dia mais.

Aos amigos Ederson Marcos Sgarbi, Carlos Eduardo Ribeiro (Biluka), Luiz Fernando Legore do Nascimento, José Reinaldo Merlin, André Luiz Oliveira, Thiago Adriano Coleti, Fabio de Sordi Júnior, Christian James de Castro Bussmann, Daniela de Freitas Guilhermino Trindade, Luiz Guilherme Sachs, Bruno Miguel Nogueira de Souza, André Luis Andrade Menolli, Glauco Carlos Silva, Ricardo Gonçalves Coelho e Ailton Sérgio Bonifácio, pelas tantas conversas produtivas sobre minha pesquisa, pelo apoio e sobretudo pela amizade de tantos anos.

A todos os colegas de trabalho do Centro de Ciências Tecnológicas da Universidade Estadual do Norte do Paraná pelo apoio ao longo de todo esse tempo de capacitação.

Aos funcionários do Departamento de Computação e da Secretaria de Pós-Graduação do Centro de Ciências Exatas da UEL pela ajuda e orientações ao longo do curso.

Enfim, agradeço a todos os amigos que contribuíram direta e indiretamente para a conclusão deste trabalho.

"Simplicity is a great virtue but it requires hard work to achieve it and education to appreciate it. And to make matters worse: complexity sells better."

Edsger W. Dijkstra

DELLA MURA, W. A.. **Detecção de conflitos em contratos multilaterais**. 133 p. Dissertação de Mestrado (Mestrado em Ciência da Computação) – Universidade Estadual de Londrina, Londrina–PR, 2016.

RESUMO

O conceito de contrato desempenha um papel importante no mundo dos negócios em que relações comerciais entre diferentes partes são ditadas por normas legais. Esse método de negociação tem sido cada vez mais utilizado por meio de transações eletrônicas, devido aos avanços tecnológicos e à globalização. Com isso, algumas dificuldades surgiram para se garantir a confiabilidade entre as partes interessadas na realização das negociações. Uma forma de se garantir propriedades e acordos em contratos eletrônicos é verificação automática e rigorosa com base em formalismos matemáticos. Alguns formalismos, como as lógicas deôntica e dinâmica, são empregados na representação de contratos eletrônicos. Modelos formais com suporte computacional permitem alcançar resultados mais precisos na verificação desses contratos. Em decorrência de tal êxito, estudos sobre contratos eletrônicos têm sido amplamente abordados na literatura. No entanto, novos desafios surgem nesse contexto, como por exemplo, a detecção de conflitos em contratos multilaterais, porém, poucos trabalhos têm lidado com essa forma contratual. Devido a isso, este trabalho propõe uma forma de representar adequadamente os contratos multilaterais por meio de formalismos e verificar automaticamente propriedades em contratos dessa natureza. Neste sentido, o trabalho apresenta a ferramenta RECALL – um verificador automático para detecção de conflitos em contratos multilaterais. Além disso, um estudo de caso real de venda de produtos via *Internet*, caracterizado por aspectos multilaterais, é modelado e verificado usando a ferramenta RECALL. Com a aplicação prática no contrato de vendas é possível fornecer uma prova de conceito sobre as funcionalidades da ferramenta desenvolvida.

Palavras-chave: Contratos Multilaterais. Lógica Deôntica Relativizada. Detecção de Conflitos.

DELLA MURA, W. A.. **Conflict detection on multiparty contracts**. 133 p. Master's Thesis (Master in Science in Computer Science) – State University of Londrina, Londrina–PR, 2016.

ABSTRACT

The notion of contracts has played an important role in business where trade relationships between different parties are dictated by legal rules. This concept has been used increasingly by electronic transactions due to technological advances and globalization. Therefore, new challenges have emerged to guarantee the reliability among stakeholders in electronic negotiations. Automatic verification based on formalisms lays the foundations to guarantee properties and agreements on electronic contracts. Some formalisms, such as deontic and dynamic logics, are used to represent electronic contracts. Formal models and computational support allow to attain more precise results on checking electronic contracts. The automatic verification of electronic contracts has arisen as a new challenge especially in the task of detecting conflicts in multi-party contracts. The problem of checking contracts has been largely addressed in the literature, but only few works have dealt with multi-party ones. This work proposes an approach to represent appropriately multi-party contracts by means of suitable formalisms and also to automatically verify properties in contracts of this nature. We present the RECALL tool, an automatic checker for finding conflicts on multi-party contracts. A real world case study characterized by multilateral aspects is also modeled and verified using the RECALL tool. Further, the practical application of our tool in a real world problem provides a proof of concept upon their functionalities.

Keywords: Multi-party contracts. Relativized Deontic Logic. Conflict Detection.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

CTD	*Contrary To Duty*
CTP	*Contrary To Prohibition*
CTL	*Computation Tree Logic*
$\mathcal{CL}$	*Contract Language*
DL	*Dynamic Logic*
LTL	*Linear Temporal Logic*
PDL	*Propositional Dynamic Logic*
$\mathcal{RCL}$	*Relativized Contract Language*
RDL	*Relativized Deontic Logic*
SDL	*Standard Deontic Logic*

SUMÁRIO

1 INTRODUÇÃO

Os avanços tecnológicos tornaram constante a interação entre pessoas e empresas através de sistemas computacionais. Essa relação tem colaborado para estabelecer novos desafios nas áreas de negociação, integração e interoperabilidade entre as organizações. Em geral, comunicações desse tipo estão sujeitas a situações de discordância entre as partes. Daí a necessidade de se obter formas de garantir os acordos estabelecidos de forma prática e confiável. A criação de acordos, ou simplesmente contratos, se torna fundamental neste cenário, quando uma transação é realizada entre as partes interessadas [3].

Os contratos têm papel importante no processo de negócio devido a necessidade de expressar o comprometimento dos participantes em relação aos seus direitos e deveres. Na área jurídica um contrato é definido como um acordo de vontades entre duas ou mais partes com o objetivo de atribuir, modificar, garantir, transferir ou extinguir direitos [4]. Os contratos são aplicados nas mais diversas áreas para garantir os direitos e deveres das partes envolvidas. Contudo, um contrato ainda pode apresentar inconsistências devido a sua descrição em linguagem natural tornando-o ambíguo ou conflitante e, possivelmente, levando a desentendimentos e quebras contratuais. Estes problemas motivam a formalização dos contratos com embasamento matemático, bem como sua automatização computacional, por meio do conceito de contratos eletrônicos [5]. Uma descrição formal de um contrato permite a verificação de forma mais confiável e automática, seja na análise, na negociação ou no seu monitoramento. Dentre essas representações mais precisas para contratos estão as lógicas modais, as lógicas temporais, a lógica deôntica e a lógica dinâmica [6, 7, 8]. Alguns trabalhos também utilizam XML [9] e outros modelos como as redes de Petri [3, 10].

A formalização dos contratos garante maior confiabilidade e exatidão em sua interpretação. No entanto, mesmo descrito formalmente, os contratos não estão livres de problemas, uma vez que em sua formulação inconsistências podem ser capturadas a partir de informações do mundo real. Por isso, a verificação automatizada de propriedades num contrato é muito importante para garantir que suas cláusulas sejam satisfeitas. A detecção de conflitos, uma das formas de verificação formal, procura por situações conflitantes entre as cláusulas de um contrato. Em geral, a detecção de conflitos é realizada antes da execução de um contrato para garantir que não existam cláusulas conflitantes, levando o contrato a uma situação de violação.

Este trabalho apresenta uma estratégia para a detecção automática de conflitos em contratos multilaterais. A proposta engloba uma forma de representação de contratos multilaterais baseada na linguagem de contrato $\mathcal{CL}$ [8] e nas lógicas voltadas para sistemas normativos baseados na Lógica Deôntica Relativizada [7]. O trabalha ainda implementa

a abordagem proposta e apresenta a aplicação prática dessa ferramenta num estudo de caso real de comércio eletrônico.

1.1 Motivação

Os contratos são garantias de que as regras de negócio, os direitos e deveres, sejam cumpridos. Na etapa de formulação do contrato é comum que todos os envolvidos tenham seus interesses e até normas próprias que devem ser incluídas na formulação do acordo multilateral. Porém, a combinação das normas de todos os participantes pode levar a uma situação de conflito no contrato como um todo. Um conflito normativo é uma situação em que as cláusulas de um contrato entram em contradição e forçam sua violação. A proibição e obrigação (ou permissão) de uma mesma ação num contrato, por exemplo, podem levá-lo a uma situação de conflito. Neste cenário, o contrato não pode ser satisfeito, pois todas as ações realizadas levam a uma violação das normas. No meio jurídico o conflito normativo é muito comum, e para contornar essa situação alguns mecanismos são aplicados para eliminar essas contradições como os critérios hierárquicos, cronológicos e de especialidade [11]. No entanto, um conflito detectado no processo de negociação de um contrato pode ser resolvido antes mesmo que este seja executado na prática. Dessa forma, a detecção de conflitos pode ser realizada utilizando técnicas de verificação formal sobre um formalismo adequado para descrever contratos.

O trabalho de Fenech [5] aborda a análise de contratos expressos em $\mathcal{CL}$ [8] e propõe um método de verificação de conflitos normativos em contratos bilaterais. Contudo, no caso de contratos multilaterais, a detecção de conflitos torna-se mais complexa, devido à necessidade de se determinar os indivíduos envolvidos nas cláusulas contratuais e os tipos de conflitos que podem ocorrer com estes indivíduos. Conforme já mencionado, a obrigação e a proibição de uma mesma ação para um determinado indivíduo é considerada um conflito, mas a proibição de uma ação para um indivíduo e a obrigação da mesma ação para outro pode não caracterizar um conflito.

Os conceitos presentes na $\mathcal{CL}$ são utilizados para definir os tipos de conflitos normativos desta abordagem. Como a $\mathcal{CL}$ é baseada na lógica deôntica padrão [12], onde os indivíduos do contrato não estão explícitos, não é possível determinar o responsável pela execução de uma ação. Porém, o trabalho de Herrestad e Krogh [7] propõe uma personalização de operadores deônticos para identificar os indivíduos de uma norma, possibilitando a representação de situações em que exista mais de um indivíduo no contrato.

Um trabalho que aborde a detecção de conflitos em contratos multilaterais contribui significativamente com área de contratos eletrônicos, pois os trabalhos conhecidos da literatura, na sua grande maioria, tem como objeto de estudo os contratos bilaterais. As propriedades bilaterais, em geral, são mais simples de lidar do que aquelas apresentadas

em cenários multilaterais. As implicações para a verificação automática quando se tem muitos indivíduos num contrato vão desde a dificuldade de se encontrar o responsável por uma violação [13] até o aumento da complexidade na forma de se representar estes contratos [7].

A Figura 1 mostra o escopo deste trabalho em relação as lógicas de especificação de contratos eletrônicos e a técnica de detecção de conflitos existentes na literatura. As elipses representam as lógicas envolvidas no trabalho enquanto que o retângulo indica a técnica de detecção de conflito para contratos bilaterais. Os retângulos hachurados em cinza indicam a proposta do trabalho com a relativização da $\mathcal{CL}$ e o mecanismo de detecção de conflitos para contratos multilaterais. As setas contínuas indicam uma lógica ou técnica da literatura enquanto que as setas tracejadas indicam a proposta desenvolvida neste trabalho.

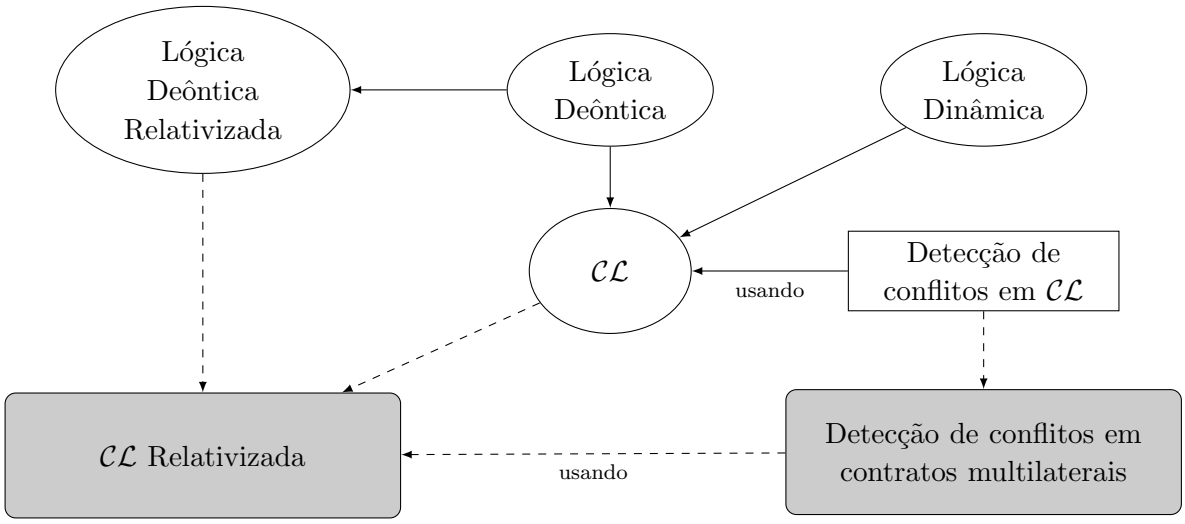

Figura 1 – Escopo do método proposto.

Este trabalho então estende o método de Fenech [14] para abranger a verificação de conflitos em contratos multilaterais baseada na lógica deôntica relativizada, proposta por Herrestad e Krogh [7], introduzida na $\mathcal{CL}$. O método é aplicado a um estudo de caso real na prática, baseado no contrato multilateral proposto por Daskalopulu [15], que modela as regras de comércio eletrônico de produtos via *Internet*.

1.2 Objetivos gerais e específicos

O objetivo geral deste trabalho é propor uma técnica de detecção de conflitos em contratos multilaterais. Existem técnicas de análise de conflitos em contratos bilaterais, onde a identificação dos indivíduos muitas vezes não é importante [5]. No entanto, não se tem conhecimento de uma abordagem de detecção de conflitos que leve em consideração a identificação dos responsáveis pelas ações num contrato.

Os objetivos específicos para se atingir o propósito deste trabalho são:

- **Adaptar a linguagem $\mathcal{CL}$ para suportar indivíduos:** a $\mathcal{CL}$ é baseada na lógica deôntica padrão, herdando as mesmas limitações quanto a identificação dos indivíduos na execução das ações. Ao agregar os conceitos deônticos relativizados [7], é possível determinar os indivíduos e suas ações deônticas.
- **Definir novos tipos de conflitos:** o tratamento dos indivíduos de maneira explícita faz surgir novos tipos de conflitos nos contratos multilaterais.
- **Adequar uma nova semântica de conflitos:** a extensão na semântica da linguagem de contratos é necessária para se determinar os indivíduos e suportar o novo conjunto de conflitos. Uma adaptação se faz necessária também no algoritmo de detecção de conflitos para contemplar os novos operadores relativizados.
- **Implementar o método proposto:** a solução de detecção de conflitos deve ser implementada com base na nova sintaxe e semântica da extensão da $\mathcal{CL}$. A ferramenta implementada deve proporcionar uma aplicação abrangente da técnica proposta.
- **Aplicar a ferramenta a um estudo de caso real:** a ferramenta desenvolvida deve ser aplicada num estudo de caso como prova de conceito do método proposto e demonstração de sua aplicabilidade em contratos reais.

1.3 Organização do Trabalho

A organização do trabalho obedece a estrutura a seguir. No Capítulo 2 é apresentado um referencial teórico sobre contratos convencionais e eletrônicos além de formalismos para a representação formal de sistemas e de contratos eletrônicos. O Capítulo 3 apresenta os trabalhos mais intimamente relacionados com a pesquisa apresentada neste documento. O Capítulo 4 apresenta o método proposto para detecção de conflitos, descrevendo a sintaxe da $\mathcal{CL}$ Relativizada, a técnica de detecção de conflitos multilaterais e um exemplo de sua aplicação. O Capítulo 5 apresenta a ferramenta que implementa o método proposto. Já o Capítulo 6 contextualiza o estudo de caso multilateral a ser verificado, sua modelagem através da representação proposta e aplicação da ferramenta sobre a modelagem obtida. Finalmente, o Capítulo 7 descreve as considerações finais do trabalho, em relação ao método proposto e aos problemas pendentes, bem como direções para trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta os conceitos relacionados a contratos convencionais e eletrônicos, como as características, problemas e desafios encontrados em suas aplicações. Algumas representações para sistemas e contratos eletrônicos também são descritas, fornecendo um conjunto de conceitos necessários para a solução dos problemas propostos neste trabalho.

2.1 Contratos

Desde os primórdios da sociedade moderna, os acordos, normas e regras regem o relacionamento entre as pessoas. Salomo [16] afirma que os contratos foram criados devido ao progresso da civilização e a necessidade de se garantir os interesses das pessoas. Um contrato pode, então, ser definido como sendo um conjunto de normas, ou cláusulas, que rege a negociação entre os indivíduos de uma sociedade. Os contratos são uma forma de manifestar e garantir as vontades dos indivíduos sobre um determinado assunto. Na área jurídica, segundo Souza [17], um contrato é uma declaração de vontade dentro da lei, capaz de produzir o efeito jurídico desejado pelo indivíduo que declarou tal vontade.

Os contratos podem ser classificados de acordo com a atribuição de responsabilidades. Os contratos unilaterais ocorrem quando apenas um indivíduo assume responsabilidades, no caso de doações e comodatos. Nos contratos bilaterais, dois indivíduos assumem responsabilidades. Já no caso de contratos multilaterais três ou mais indivíduos assumem responsabilidades [4].

A negociação de contratos entre indivíduos pode acarretar na combinação de cláusulas de interesse de cada envolvido. Essa combinação pode trazer diversos problemas, como é o caso dos conflitos normativos [11]. Um conflito normativo consiste numa contradição entre as normas[18]. Por exemplo, uma norma pode proibir uma certa ação enquanto a outra permite, ou ainda ter uma norma que proíbe determinada ação enquanto a outra obriga a execução desta ação. Essas situações podem ser demonstradas pela incompatibilidade entre as normas que obrigam e proíbem ou que permitem e proíbem simultaneamente [19]. A existência de conflitos faz com que o contrato se torne inválido, pois não há meio de satisfazê-lo sem violar ou desconsiderar alguma das normas estabelecidas [11]. Uma solução para este tipo de problema geralmente consiste em reconhecer a norma com maior hierarquia. Por exemplo, no caso de uma lei conflitar com a Constituição Federal, esta última deve prevalecer sobre a primeira. Ainda existe a possibilidade de se eliminar uma norma e considerar somente a outra, mas isso pode levar a uma situação discutível e nem sempre adequada [20].

No processo de negociação de contratos, em que são apresentadas as normas de todos os interessados, ao se deparar com um conflito, as soluções apresentadas podem não ser convenientes a nenhuma das partes. Por isso, no processo de negociação os interesses de todos os indivíduos devem ser levados em consideração, verificando se algum conflito normativo ocorre no momento da elaboração do contrato.

2.2 Contratos eletrônicos

Atualmente com a tecnologia da informação integrada em praticamente todos os âmbitos da sociedade, as companhias têm utilizado diversas ferramentas para auxiliar seus processos de negócio e interações com outras empresas. A troca de informações ou produtos, realização de processos entre as partes, relacionamentos a longo prazo e prestação de serviços [21] são apenas alguns exemplos. A contratação é um dos conceitos mais fundamentais da economia ocidental [6], e possui um papel importante tanto nos relacionamentos de negócios tradicionais quanto nos relacionamentos pelo meio eletrônico [22].

Um contrato eletrônico ou *"e-Contract"* é uma conversão de contratos convencionais, utilizados para firmar acordos entre pessoas ou entidades, para o meio digital com o propósito de utilizar a tecnologia como aliada na solução dos problemas encontrados neste contexto. Um contrato eletrônico possui as obrigações de cada participante, atividades exercidas e responsabilidades definidas para cada um dos indivíduos para satisfazer os termos e condições do contrato firmado [13].

Além da conversão de contratos legais, ainda há abordagens que tratam da utilização de contratos eletrônicos para formalizar regras de negócio em *workflows* [9], verificação de sistemas baseados em multi-agentes [23], verificação de arquiteturas baseadas em serviços e em nuvem [24], entre outras aplicações.

As áreas de aplicação dos contratos eletrônicos são as mais variadas, como meios legais, em que se converte ou substitui um contrato físico para o meio eletrônico, ou utilizando as técnicas de contratação eletrônica em sistemas, lógicas de negócio ou integração de serviços. A contratação eletrônica legal foca na criação de um documento contratual, que objetiva principalmente expressar a intenção das partes envolvidas [13]. Neste caso, as ferramentas para esta tarefa buscam informar as partes sobre os efeitos das disposições contratuais, auxiliar na verificação e execução do contrato, bem como acompanhar seu cumprimento.

A lei considera contratos como uma coleção de obrigações e o mesmo se aplica ao meio eletrônico. As pesquisas nesta área incluem métodos de inferência automática que facilitam a análise de problemas de forma prática. O propósito de um sistema de contratos eletrônicos legais é esclarecer e expandir uma afirmação de requerimentos incompleta ou imprecisa dentro de uma especificação mais precisa. A pesquisa de Pace, Prisacariu e

Schneider [24] baseia-se na utilização da lógica deôntica [12] para esse tipo de formalização, definindo os direitos e deveres entre outros conceitos legais complexos. Uma vantagem em comparação a contratos convencionais é o apoio computacional e automatizado dos contratos eletrônicos para que possam ser verificados, controlados ou monitorados de forma mais eficiente.

2.2.1 Topologia dos contratos eletrônicos

A topologia de um contrato eletrônico pode ser definida pelo número de participantes envolvidos e a complexidade das relações de direitos e deveres entre esses participantes. Cada participante do processo de contratação e execução do contrato é denominado indivíduo e pode representar as partes envolvidas num relacionamento de negócios, cliente ou vendedor numa transação de compra e venda, ou ainda o segurado num contrato de seguro automotivo [25].

O contrato bilateral tem como característica principal o acordo entre no máximo dois indivíduos. No caso do contrato multilateral, as cláusulas consistem num conjunto de acordos entre vários indivíduos para um objetivo comum. Como a classificação depende do número de envolvidos e de sua complexidade, as topologias relacionadas com este trabalho são descritas a seguir:

- **Contrato bilateral**: dois indivíduos estabelecem um contrato com obrigações de ambos. Um exemplo desse tipo de contrato ocorre numa prestação de serviço entre um provedor de internet e um consumidor [24], como representado na Figura 2. Neste contrato, o provedor se compromete a entregar a conexão ao cliente sob certas especificações. Por outro lado, o cliente tem as obrigações de usar apenas a velocidade contratada, pagar a mensalidade em dia e em caso de violação de alguma cláusula, pagar uma multa.

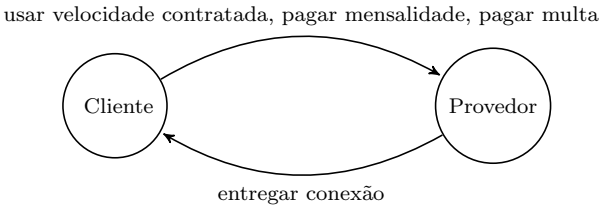

Figura 2 – Exemplo de contrato bilateral.

- **Contrato em cadeia**: envolve a participação de vários indivíduos, porém, em forma de sub-contratos. Um exemplo desse tipo de contrato é apresentado na Figura 3, onde a requisição de um produto é feita pelo cliente ao vendedor que, por sua vez, faz o pedido ao fornecedor que envia o produto ao vendedor, gerando assim um novo contrato. Uma característica interessante num contrato em cadeia é que ele pode

ser dividido em sub-contratos bilaterais, diminuindo assim sua complexidade sem perder informações importantes.

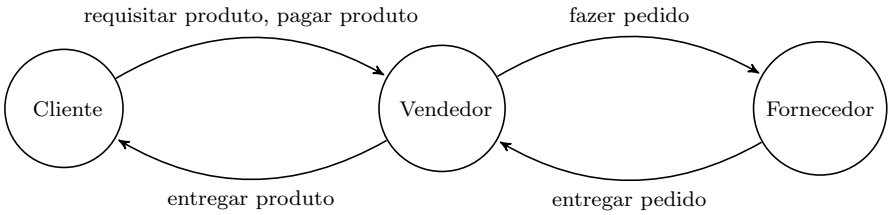

Figura 3 – Exemplo de contrato em cadeia.

- **Contrato multilateral**: de forma semelhante ao contrato em cadeia, possui a presença de dois ou mais indivíduos. Porém, os indivíduos podem interagir entre si, causando uma dependência mais complexa do que o modelo em cadeia. Um exemplo desse tipo de contrato é a execução de uma compra segura pela internet [15], que pode ser visualizada na Figura 4. O cliente realiza a compra do produto do vendedor e tem a obrigação de pagar o valor correspondente ao banco. Após o pagamento do produto, o vendedor envia o produto ao cliente por meio da transportadora e paga o valor do frete ao banco. Assim que o produto é entregue, o banco repassa o valor pago do produto para o vendedor e o valor do frete para a transportadora. Este processo, diferente do contrato em cadeia, não pode ser dividido num conjunto de contratos bilaterais sem perder detalhes do processo [13].

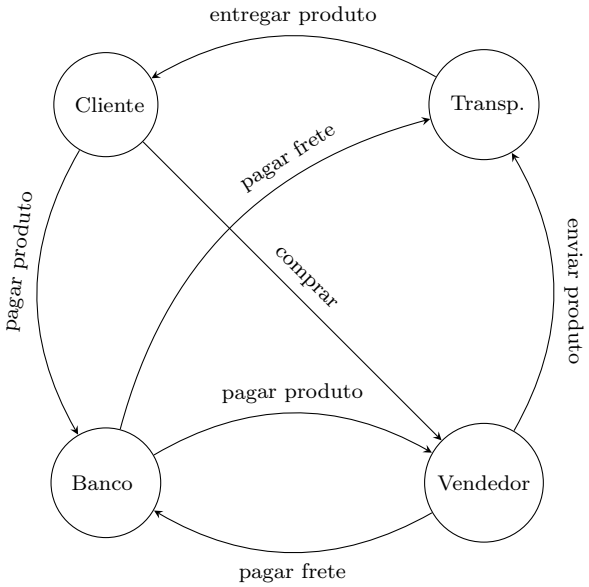

Figura 4 – Exemplo de contrato multilateral.

Os contratos eletrônicos mais simples são celebrados de forma bilateral [13]. Quando isso não ocorre, muitas vezes é possível dividir um contrato em vários contratos bilaterais.

Entretanto, essa divisão não pode ser realizada em contratos multilaterais sem a perda de informações importantes. Assim, um conjunto de contratos bilaterais não é capaz de expressar a complexidade do relacionamento entre os indivíduos de um contrato multilateral [3]. No processo de decomposição de um contrato multilateral num conjunto de contratos bilaterais ocorre a perda de informações e a diminuição de expressividade das normas do contrato.

Para exemplificar a dificuldade de separar um contrato multilateral em outros bilaterais, considera-se o exemplo da Figura 4. Ao separar o contrato de maneira bilateral, as dependências das ações devem ser levadas em conta. Na relação entre o vendedor e a transportadora, o pagamento do frete só pode ser realizado se a transportadora tiver entregue o produto ao cliente e este tenha notificado o banco sobre o recebimento. Uma discussão mais profunda sobre a impossibilidade de separação desse tipo de contrato é realizada na Seção 6.1 que trata do estudo de caso deste trabalho.

2.2.2 Verificação de contratos eletrônicos

Diversas pesquisas já foram realizadas sobre a área de contratos eletrônicos, em especial, na verificação formal desses contratos. A automatização da análise de um contrato, além de acelerar sua escrita, permite que propriedades desejadas sejam formalmente verificadas [5]. Algumas das técnicas e abordagens mais comuns em contratos eletrônicos são:

- **Negociação:** Baseado no processo de negociação de contratos legais, é um cenário em que pelo menos dois indivíduos visam alcançar um acordo que é aceitável por ambas as partes envolvidas. Normalmente, cada parte inicia a negociação oferecendo a solução preferida de seu interesse. A outra parte, caso não aceite, deve fazer contrapropostas para que ambos cheguem a um acordo. Este processo ocorre em contratos bilaterais ou multilaterais por meio de mediadores ou não [26].

- **Detecção de conflitos:** A detecção de conflitos [5] objetiva a eliminação de conflitos normativos de um contrato. Um conflito normativo é uma situação em que as normas se contradizem [19]. Esta situação invalida um contrato, levando a uma situação de violação [18]. A verificação de conflitos deve ocorrer antes da execução do contrato e, se for o caso, após sua negociação. Já existem estudos na área de detecção de conflitos bilaterais [14, 27], porém esta análise não é comum em contratos multilaterais.

- **Assinatura:** O problema da assinatura digital dos contratos ocorre posteriormente à etapa de negociação. Em geral, a assinatura de um contrato digital é mais complicada do que no modo convencional. O problema surge pelo fato de que nenhum indivíduo envolvido quer ser o primeiro a assinar o contrato. Pode ocorrer de que o outro

indivíduo se recuse a assinar depois de ter obtido a assinatura e o contrato do primeiro indivíduo [28]. Esse problema pode causar uma vantagem indesejável para um dos envolvidos [29, 30].

- **Monitoramento:** No processo de execução de um contrato, cada participante tem suas características particulares. O monitoramento ocorre durante a execução do contrato e busca verificar se as cláusulas são respeitadas pelo participantes. Um participante de um contrato pode executar as ações descritas numa cláusula ou então ignorá-las, levando a violação do contrato. Com o monitoramento da execução é possível tornar as transações e relacionamentos entre os indivíduos mais confiáveis, flexíveis, eficientes e aceitáveis. Dessa maneira tenta-se manter os benefícios entre todos os indivíduos mesmo na presença de violações [22]. Por esse motivo, um sistema de monitoramento de contratos precisa saber quais ações são aceitáveis ou esperadas de um indivíduo e, caso uma violação ocorra, quem é o responsável pela violação [22]. O problema de monitoramento já foi abordado tanto em contratos bilaterais [31] quanto em contratos multilaterais [32, 13].

2.3 Representação formal de sistemas

Em geral, um contrato representado por linguagem natural pode possuir ambiguidades. A utilização de formalismos para expressar contratos evita inconsistências e possibilita a verificação sistemática com apoio computacional. Por isso, uma das tarefas para se verificar contratos eletrônicos de forma automática é definir um formalismo adequado para expressá-los [14]. A seguir são descritas as lógicas normalmente utilizadas para representar sistemas e propriedades, bem como a lógica deôntica, indicada para representar sistemas normativos.

2.3.1 Lógica modal

A lógica modal é, estritamente falando, o estudo do comportamento dedutivo das expressões "é necessário que" e "é possível que", ou seja, sua principal característica é a capacidade de expressar necessidade e possibilidade [33]. O termo lógica modal descreve um grande número de lógicas na literatura que possuem características modais, como as lógicas epistêmicas [34], lógicas temporais [35] e lógicas deônticas [12]. Sendo assim, a lógica modal estende a lógica clássica com operadores modais que expressam não somente que uma proposição é verdadeira, mas que ela pode ser verdadeira ou necessariamente verdadeira [36].

A sintaxe da lógica modal é apresentada conforme segue:

$$\varphi ::= p \mid \top \mid \bot \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \varphi \rightarrow \varphi \mid \varphi \leftrightarrow \varphi \mid \Box\varphi \mid \Diamond\varphi \tag{2.1}$$

Seja p uma proposição atômica, $\top$ indica o valor verdadeiro e $\bot$ o valor falso. A lógica modal é baseada nos conectivos da Lógica Proposicional: negação $\neg$; conjunção $\wedge$; disjunção $\vee$; condicional $\rightarrow$; e bicondicional $\leftrightarrow$. Além dos operadores proposicionais, são acrescido dos operadores modais de necessidade, denotado por $\Box(p)$ e de possibilidade, denotado por $\Diamond(p)$ [36].

A semântica da lógica modal é definida considerando o conceito de mundos possíveis, sendo modelada por uma estrutura de Kripke [37]. Esta estrutura é denotada pela tupla $\mathcal{M} = \langle W, R, L\rangle$, onde W é o conjunto de mundos (ou estados) possíveis; $R \subseteq W \times W$ é a relação de acessibilidade entre os mundos, de modo que se $w, w' \in W$ e se $(w, w') \in R$, o mundo w' é acessível a partir de w; e a função L rotula as proposições válidas num determinado mundo, considerando que $w \in W$, se $p \in L(w)$, então p é verdadeiro no mundo w.

Dado um modelo $\mathcal{M} = \langle W, R, L\rangle$, a relação de satisfação $\models$ pode ser definida para cada mundo $w \in W$, conforme a Figura 5.

$$\mathcal{M}, w \models \neg p \iff p \notin L(w) \tag{2.2}$$
$$\mathcal{M}, w \models (p \wedge q) \iff p \in L(w) \text{ e } q \in L(w) \tag{2.3}$$
$$\mathcal{M}, w \models (p \vee q) \iff p \in L(w) \text{ ou } q \in L(w) \tag{2.4}$$
$$\mathcal{M}, w \models \Box(p) \iff \forall w' \in W \text{ tal que } (w, w') \in R \text{ então } p \in L(w') \tag{2.5}$$
$$\mathcal{M}, w \models \Diamond(p) \iff \exists w' \in W \text{ tal que } (w, w') \in R \text{ então } p \in L(w') \tag{2.6}$$

Figura 5 – Semântica da Lógica Modal.

O operador modal de necessidade, representado por $\Box(p)$ é satisfeito pelo modelo $\mathcal{M}$ se e somente se a proposição p é verdadeira em todos os mundos alcançáveis a partir do mundo w. Já o operador de possibilidade, denotado por $\Diamond(p)$, é satisfeito pelo modelo $\mathcal{M}$ desde que exista pelo menos um mundo alcançável a partir de w em que a proposição p seja verdadeira. Os demais conectores lógicos são determinados a partir da satisfação de $\mathcal{M}, w \models p$.

Um exemplo de modelo de Kripke pode ser dado conforme o grafo direcionado na Figura 6, considerando o modelo $\mathcal{M}$ com as definições a seguir:

$$W = \{w_0, w_1, w_2\} \tag{2.7}$$
$$R = \{(w_0, w_1), (w_0, w_2)\} \tag{2.8}$$
$$L(w_0) = \{p\}, L(w_1) = \{p, q\}, L(w_2) = \{q\} \tag{2.9}$$

Neste caso, considerando o mundo w_0 pode ser observado que $\mathcal{M}, w_0 \not\models \Box(p)$, uma vez que o mundo alcançável w_2 não é rotulado por p. Por outro lado, $\mathcal{M}, w_0 \models \Diamond(p \wedge q)$, pois $w_1 \models p \wedge q$, ou seja, existe um mundo alcançável a partir de w_0 que satisfaz p e q.

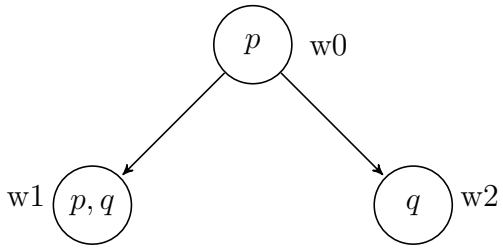

Figura 6 – Exemplo de modelo de Kripke para a lógica modal.

2.3.2 Lógica temporal

Uma fórmula lógica temporal, em geral, é avaliada sobre um sistema de transição [38] que modela uma especificação. Um sistema de transição é baseado num modelo de Kripke, conforme apresentado anteriormente. Uma fórmula lógica temporal é verdadeira ou falsa de acordo com o modelo sobre o qual é interpretada. Os modelos representam a evolução do sistema ao longo do tempo por meio do conjunto de estados e as fórmulas são avaliadas em cada estado do sistema. Por isso, a noção estática de verdade é substituída por uma noção dinâmica em que o sistema evolui de um estado para outro ao longo do tempo.

A lógica temporal pode ser dividida em dois tipos: linear e ramificada. Na abordagem linear o tempo é tratado como se cada momento tivesse apenas um único futuro possível, e assim as fórmulas são interpretadas como sequências lineares, ou caminhos. Já na abordagem ramificada, em cada estado pode ocorrer uma ramificação em vários futuros possíveis, dependendo da evolução do sistema, com uma representação em árvore.

Lógica temporal linear

A lógica temporal linear (ou LTL) possui conectivos que se referem ao futuro. Como futuro entende-se a evolução de um sistema de transição de estados. As fórmulas da LTL são interpretadas sobre uma sequência de estados chamada de *path* (caminho). Um *path* é uma sequência finita não vazia denotada por $\pi = \langle \pi_0, \pi_1, \ldots, \pi_{n-1} \rangle$ tal que os estados $\pi_0, \pi_1, \ldots, \pi_{n-1} \in W$ e $(\pi_i, \pi_{i-1}) \in R$ para todo $0 \leq i < n-1$. O tamanho de um *path* é denotado por $\mid \pi \mid = n$. Um *path* infinito é denotado por $\pi = \langle \pi_0, \pi_1, \pi_3, \ldots \rangle$ e seu tamanho é $\mid \pi \mid = \infty$. Um *path* é considerado maximal se não puder ser estendido com mais estados, e por isso, todo *path* infinito é maximal [39].

Uma fórmula LTL é composta de proposições atômicas e gerada conforme segue:

$$\varphi ::= \top \mid \bot \mid p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \varphi \rightarrow \varphi \mid X\varphi \mid F\varphi \mid G\varphi \mid \varphi U \varphi' \tag{2.10}$$

Além dos identificadores das proposições, os valores lógicos avaliados como verdadeiro e falso são representados pelos símbolos $\top$ e $\bot$, respectivamente. Os operadores lógicos $\neg$, $\wedge$, $\vee$ e $\rightarrow$ têm o mesmo significado da lógica proposicional convencional. Já os

operadores X, F, G e U, são usados para especificar situações temporais de uma fórmula. A expressão $X\varphi$ significa que a proposição φ é verdadeira no próximo estado. Já a expressão $F\varphi$ representa que φ é verdadeira em algum estado futuro da computação enquanto $G\varphi$ indica que φ é verdadeira em todos os estados do caminho de computação. Por fim, a expressão $\varphi U \varphi'$ diz que a proposição φ deve ser verdadeira em todos os estados até que φ' seja verdadeira.

Uma fórmula α em LTL é satisfeita se existe um modelo $\mathcal{M}$ e um *path* π tal que $\mathcal{M}, \pi \models \varphi$. A relação de satisfação $\models$ consiste em verificar se o *path* π satisfaz a fórmula φ no modelo $\mathcal{M}$. Um *path* é formado por uma sequência de estados s e suas posições são indexadas a partir de π_1 até π_n onde n é a quantidade de posições de π. Dado o *path* π, um caminho no modelo $\mathcal{M}$, a relação de satisfação em LTL [1] é descrita na Figura 7.

$$\pi \models \top \tag{2.11}$$
$$\pi \not\models \bot \tag{2.12}$$
$$\pi \models p \iff p \in L(\pi_1) \tag{2.13}$$
$$\pi \models \neg\varphi \iff \pi \not\models \varphi \tag{2.14}$$
$$\pi \models \varphi \wedge \varphi' \iff \pi \models \varphi \text{ e } \pi \models \varphi' \tag{2.15}$$
$$\pi \models \varphi \vee \varphi' \iff \pi \models \varphi \text{ ou } \pi \models \varphi' \tag{2.16}$$
$$\pi \models \varphi \rightarrow \varphi' \iff \pi \models \varphi \text{ sempre que } \pi \models \varphi' \tag{2.17}$$
$$\pi \models X\varphi \iff \pi_2 \models \varphi \tag{2.18}$$
$$\pi \models G\varphi \iff \forall i \geq 1 \mid \pi_i \models \varphi \tag{2.19}$$
$$\pi \models F\varphi \iff \exists i \geq 1 \mid \pi_i \models \varphi \tag{2.20}$$
$$\pi \models \varphi U \varphi' \iff \exists i \geq 1 \mid \pi_i \models \varphi' \text{ e } \forall j, 1 \leq j < i \mid \pi_j \models \varphi \tag{2.21}$$

Figura 7 – Semântica da LTL.

As definições (2.11) e (2.12) da Figura 7 indicam as proposições verdadeiras e falsas, respectivamente. Já a definição (2.13) indica que a proposição p é válida no primeiro estado do *path*. Os conectores da lógica proposicional são definidos nas equações (2.14) a (2.17). O operador X, definido em (2.18), considera a segunda posição do *path* para verificar a fórmula φ, indicando assim o próximo estado. As definições (2.19) e (2.20) representam os operadores G e F que indicam, respectivamente, que a fórmula φ é verdadeira em todos as posições de π ou em pelo menos uma posição de π. Por fim, a definição (2.21) representa o operador U, indicando que em alguma posição π^i, a proposição φ' é verdadeira e que em todas as posições que antecedem π^i, a fórmula φ é verdadeira.

Várias propriedades relevantes podem ser verificadas com fórmulas LTL, tais como, de segurança, em que o sistema é livre de *deadlocks*, e de progresso, em que toda requisição realizada é atendida. Num sistema especificado com as proposições *ocupado*, *requisitado*, *iniciado*, *pronto*, *habilitado*, por exemplo, podem existir as seguintes propriedades:

- o sistema não pode ter *iniciado* e ainda não estar *pronto* para atender requisições:

$$G\neg(iniciado \wedge \neg\, pronto) \tag{2.22}$$

- em todo sistema, se é *requisitado* algum recurso, o requisitante deve ser *respondido*:

$$G(requisitado \rightarrow F\ respondido) \tag{2.23}$$

- um processo é *habilitado* infinitas vezes em todo o caminho de computação:

$$GF\ habilitado \tag{2.24}$$

- um elevador subindo para o segundo andar não muda sua direção quando houver passageiros que desejam ir para o quinto andar:

$$G(andar2 \wedge subindo \wedge pressionadoBotao5 \rightarrow (subindoUandar5)) \tag{2.25}$$

Os exemplos acima mostram que a LTL consegue expressar diversas propriedades importantes em sistemas de transição de estados. Contudo, a LTL não consegue expressar a quantificação dos caminhos de computação. Os cenários a seguir, por exemplo, não podem ser expressos em LTL.

- De um estado qualquer é possível chegar a um estado de reinicialização? Ou seja, existe um caminho partindo de todos os estados para um estado que satisfaça a reinicialização?
- Um elevador pode permanecer parado no $3^{\underline{o}}$ andar com as portas fechadas? Ou seja, a partir do estado em que se encontra no $3^{\underline{o}}$ andar, há um caminho ao longo do qual ele permanece lá?

Lógica de computação em árvore

A lógica de computação em árvore (CTL) modela o futuro em forma de ramificações. A CTL permite a quantificação dos vários caminhos de execução de um sistema [40]. A computação da CTL é representada por uma árvore de estados e não apenas uma sequência como na LTL. Cada caminho na árvore representa uma execução possível do sistema. A árvore em si representa todas as execuções possíveis, como pode ser visto na Figura 8. À esquerda é representado o modelo e à direta, a árvore de computação e seus nós, representando o estado corrente e o nível de profundidade da execução, indicando as mudanças de estados. [41].

A sintaxe de construção de fórmulas CTL estende a LTL e adiciona os operadores existencial e universal conforme segue:

$$\begin{aligned}&\varphi ::= \top \mid \bot \mid p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \varphi \rightarrow \varphi \mid \\ &AX\varphi \mid EX\varphi \mid AF\varphi \mid EF\varphi \mid AG\varphi \mid EG\varphi \mid A[\varphi U \varphi] \mid E[\varphi U \varphi]\end{aligned} \tag{2.26}$$

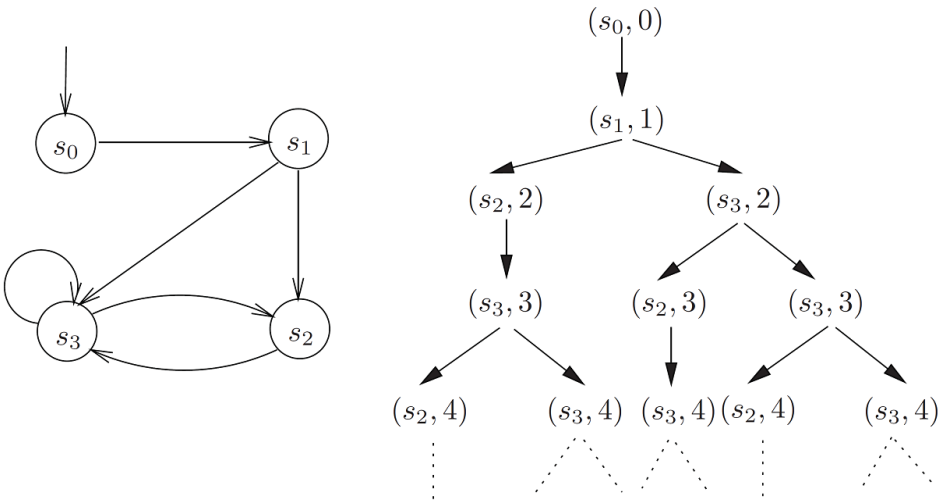

Figura 8 – Exemplo de árvore de computação num sistema de transição [1].

Os conectivos temporais da CTL são usados sempre aos pares. O primeiro conectivo da fórmula é um quantificador, A (inevitavelmente) ou E (provavelmente), enquanto que o segundo é usado como apresentado na LTL. Os operadores temporais E e A expressam propriedades para algum ou todos os caminhos de computação que se iniciam em determinado estado. O operador E denota o quantificador existencial de caminho e o operador A denota o quantificador universal de caminhos [41]. Alguns exemplos de fórmulas CTL podem ser vistas a seguir:

- é possível chegar a um estado em que o sistema está *iniciado* mas não *pronto*?

$$EF(iniciado \wedge \neg pronto) \tag{2.27}$$

- em qualquer caminho e em todo este caminho, se algum recurso é *requisitado*, ele é *reconhecido* em algum momento?

$$AG(requisitado \rightarrow AF\ reconhecido) \tag{2.28}$$

- um determinado processo está *habilitado* infinitamente em cada caminho de computação.

$$AG(AF\ habilitado) \tag{2.29}$$

As fórmulas em CTL são interpretadas sobre os caminhos a partir de estados de um sistema de transição [41]. Cada *path* na árvore é entendido como uma computação possível no modelo. A raiz da árvore representa o estado inicial da computação. Dado o modelo $\mathcal{M} = (W, R, L)$, a relação de satisfação $\mathcal{M}, s \models \varphi$ pode ser dada indutivamente conforme é mostrado na Figura 9, tal que $\mathcal{M}$ é um modelo de transição, $s \in W$ e φ é uma fórmula em CTL.

$$\mathcal{M}, s \models \top \tag{2.30}$$
$$\mathcal{M}, s \not\models \bot \tag{2.31}$$
$$\mathcal{M}, s \models p \iff p \in L(s) \tag{2.32}$$
$$\mathcal{M}, s \models \neg\varphi \iff \mathcal{M}, s \not\models \varphi \tag{2.33}$$
$$\mathcal{M}, s \models \varphi \wedge \varphi' \iff \mathcal{M}, s \models \varphi \text{ e } \mathcal{M}, s \models \varphi' \tag{2.34}$$
$$\mathcal{M}, s \models \varphi \vee \varphi' \iff \mathcal{M}, s \models \varphi \text{ ou } \mathcal{M}, s \models \varphi' \tag{2.35}$$
$$\mathcal{M}, s \models \varphi \rightarrow \varphi' \iff \mathcal{M}, s \not\models \varphi \text{ ou } \mathcal{M}, s \models \varphi' \tag{2.36}$$
$$\mathcal{M}, s \models AX\varphi \iff \forall s_1 \mid s \rightarrow s_1 \text{ com } \mathcal{M}, s_1 \models \varphi \tag{2.37}$$
$$\mathcal{M}, s \models EX\varphi \iff \exists s_1 \mid s \rightarrow s_1 \text{ com } \mathcal{M}, s_1 \models \varphi \tag{2.38}$$
$$\mathcal{M}, s \models AG\varphi \iff \forall \pi = s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow \cdots \mid s_1 = s \text{ e } \forall s_i \in \pi \text{ com } \mathcal{M}, s_i \models \varphi \tag{2.39}$$
$$\mathcal{M}, s \models EG\varphi \iff \exists \pi = s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow \cdots \mid s_1 = s \text{ e } \forall s_i \in \pi \text{ com } \mathcal{M}, s_i \models \varphi \tag{2.40}$$
$$\mathcal{M}, s \models AF\varphi \iff \forall \pi = s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow \cdots \mid s_1 = s \text{ e } \exists s_i \in \pi \mid \mathcal{M}, s_i \models \varphi \tag{2.41}$$
$$\mathcal{M}, s \models EF\varphi \iff \exists \pi = s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow \cdots \mid s_1 = s \text{ e } \exists s_i \in \pi \mid \mathcal{M}, s_i \models \varphi \tag{2.42}$$
$$\mathcal{M}, s \models A[\varphi U \varphi'] \iff \forall \pi = s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow \cdots \mid s_1 = s \; \exists s_i \in \pi \mid \mathcal{M}, s_i \models \varphi' \tag{2.43}$$
$$\text{e } \forall j < i \text{ com } \mathcal{M}, s_j \models \varphi \tag{2.44}$$
$$\mathcal{M}, s \models E[\varphi U \varphi'] \iff \exists \pi = s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow \cdots \mid s_1 = s \; \exists s_i \in \pi \mid \mathcal{M}, s_i \models \varphi' \tag{2.45}$$
$$\text{e } \forall j < i \text{ com } \mathcal{M}, s_j \models \varphi \tag{2.46}$$

Figura 9 – Semântica da CTL.

Na CTL algumas propriedades mais complexas podem ser expressadas aninhando-se os quantificadores existencial e universal. Por exemplo, a propriedade citada anteriormente em que para todo caminho de computação sempre é possível retornar para o estado inicial, pode ser representada como $AG\ (EF\ (inicio))$. A interpretação é que em todo estado do sistema (G), para todo caminho de computação (A), existe a possibilidade (E) de eventualmente retornar ao início $(F\ inicio)$ [42].

Outro exemplo de expressão que não pode ser descrita com LTL é: "Um elevador pode permanecer parado no 3º andar com as portas fechadas?". Esta expressão é descrita em CTL da seguinte maneira:

$$AG(andar3 \wedge parado \wedge portaFechada \rightarrow EG(andar3 \wedge parado \wedge portaFechada)) \tag{2.47}$$

2.3.3 Lógica dinâmica proposicional

A Lógica Dinâmica (DL) é um sistema formal para raciocínio sobre programas, proposta por Pratt [43]. Programas são definidos como uma sequência de ações para computar um dados que mudam o valor de uma variável e consequentemente alteram o resultado de fórmulas proposicionais. Esta lógica é utilizada na formalização de especificações de correção e provas rigorosas para que essas especificações sejam satisfeitas por um programa em particular. Existem outras atividades que se enquadram nas aplicações

da DL, como: determinar a equivalência dos programas; comparar o poder expressivo de várias construções de programação; ou sintetizar programas a partir de especificações [44].

A lógica dinâmica pode ser entendida como uma união da lógica de predicados, da lógica modal e da álgebra de eventos regulares. A diferença principal da lógica dinâmica comparada com a lógica proposicional é que o valor verdade não é estático, assim como ocorre na lógica temporal. Na lógica de predicados, o valor de uma fórmula φ é determinado por uma avaliação das suas variáveis sobre alguma estrutura. Por exemplo, a avaliação de φ é considerada imutável quando $x = 2$ e φ indica que "x é par". Já na lógica dinâmica existem construções sintáticas explícitas chamadas programas, cujo papel principal é mudar os valores das variáveis, mudando os valores verdade das fórmulas. Por exemplo, o programa $x := x + 1$ sobre os números naturais altera o valor verdade da fórmula φ, já que agora, "x é impar" [45].

A Lógica Dinâmica Proposicional (PDL), proposta por Fischer e Ladner [2], é baseada na ideia original da DL, associando um programa α com operadores modais de necessidade [ ] e possibilidade $\langle\ \rangle$. A fórmula $[\alpha]p$ significa que sempre que o programa α terminar, deve fazê-lo num estado que satisfaça a proposição p. Um estado é a representação de um instante de tempo que descreve a realidade, associando um valor a cada variável do programa. Já o operador modal de possibilidade $\langle\alpha\rangle p$ representa que há uma computação de α que termina num estado que satisfaça a proposição p. A equivalência entre os operadores de necessidade e possibilidade, pode ser denotada por: $\langle\alpha\rangle p \equiv \neg[\alpha]\neg p$.

Há dois conjuntos de símbolos presentes no sistema formal da PDL. O primeiro representa o conjunto de fórmulas atômicas denotado por Φ. O segundo representa o conjunto dos programas atômicos Σ, considerados indivisíveis numa linguagem de programação. Dessa forma, entende-se que a PDL se abstrai dos conceitos da computação e estuda a interação pura entre os programas e as proposições [44].

As fórmulas da Lógica Dinâmica descrevem propriedades que ocorrem após a execução de um programa. A fórmula $\varphi = [\alpha \cup \beta]p$, por exemplo, indica que sempre que o programa α ou o programa β são executados a proposição p é verdadeira no estado seguinte. Por outro lado, a fórmula $\varphi' = \langle(\alpha;\beta)^*\rangle p$ indica que há uma sequência de execuções alternadas de α e β que podem alcançar um estado em que a proposição p é verdadeira.

As fórmulas da PDL são formadas por operadores da lógica modal [33] associadas a estruturas algébricas. A sintaxe da Lógica Dinâmica Proposicional é definida recursivamente conforme segue:

$$\varphi ::= \top \mid \bot \mid p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \varphi \rightarrow \varphi \mid \varphi \leftrightarrow \varphi \mid [\alpha]\varphi \mid \langle\alpha\rangle\varphi \tag{2.48}$$

$$\alpha ::= a \mid 0 \mid 1 \mid \alpha;\beta \mid \alpha \cup \beta \mid \alpha^* \mid ?\varphi \tag{2.49}$$

Os operadores proposicionais $\neg$, $\wedge$, $\vee$, $\rightarrow$ e $\leftrightarrow$ são definidos da sua maneira usual. O conjunto de modalidades e demais operadores da PDL são apresentados na Tabela 1 [46].

Mod.	**Descrição**	**Significado**
0	Violação	Aborte o sistema
1	Qualquer ação	É permitido executar qualquer ação
$\alpha;\beta$	Composição	Primeiro faça α e depois faça β
$\varphi?$	Teste lógico	Se φ então prossiga senão aborte o sistema
$\alpha \cup \beta$	União	Faça α ou β de forma não determinística e não exclusiva
α^*	Iteração	Repita α uma quantidade não determinada de vezes
$[\alpha]\varphi$	Necessidade	Após α ocorrer, φ será verdadeiro
$\langle\alpha\rangle\varphi$	Possibilidade	Após α ocorrer, talvez φ será verdadeiro

Tabela 1 – Sintaxe da Lógica Dinâmica Proposicional.

Os conjuntos dos programas Σ e das fórmulas Φ são definidos indutivamente pelas regras a seguir [2]:

1. Programas:
   a) Programas atômicos, 0 e 1 são programas;
   b) Se α e β são programas e φ é uma fórmula, então $\alpha;\beta$, $\alpha \cup \beta$, α^* e $\varphi?$ são programas.
2. Fórmulas:
   a) Fórmulas atômicas, $\top$ e $\bot$ são fórmulas;
   b) Se p e q são fórmulas e α é um programa, então $p \wedge q$, $\neg p$, $[\alpha]p$ e $\langle\alpha\rangle p$ são fórmulas.

Na Figura 10 são apresentadas algumas aplicações desta lógica em estruturas mais comuns encontrados na maioria dos algoritmos.

$$\textbf{skip} \iff 1? \tag{2.50}$$
$$\textbf{fail} \iff 0? \tag{2.51}$$
$$\textbf{if } \varphi_1 \to \alpha_1 \mid \cdots \mid \varphi_n \to \alpha_n \textbf{ fi} \iff \varphi_1?;\alpha_1 \cup \cdots \cup \varphi_n?;\alpha_n \tag{2.52}$$
$$\textbf{if } \varphi \textbf{ then } \alpha \textbf{ else } \beta \iff \varphi?;\alpha \cup \neg\varphi?;\beta \tag{2.53}$$
$$\textbf{while } \varphi \textbf{ do } \alpha \iff (\varphi?;\alpha)^*;\neg\varphi? \tag{2.54}$$
$$\textbf{repeat } \alpha \textbf{ until } \varphi \iff \alpha;(\neg\varphi?;\alpha)^*;\varphi? \tag{2.55}$$

Figura 10 – Aplicações da Lógica Dinâmica na representação de programas.

A semântica dos operadores da PDL, proposta por Fischer e Ladner [2], é representada pela estrutura $\mathcal{A} = \langle W, \pi, \rho\rangle$, tal que:

- W é o conjunto de estados (ou mundos) do programa.

- $\pi : \Phi \to 2^W$ é uma função que provê uma interpretação das fórmulas atômicas, e $w \in \pi(p)$ indica que a fórmula p é verdadeira no estado w.

- $\rho : \Sigma \to 2^{W \times W}$ retorna uma interpretação sobre os programas atômicos onde $(u, v) \in \rho(\alpha)$ significa que existe uma execução do programa α iniciando no estado u e terminando no estado v.

Todos os programas e fórmulas são estendidos indutivamente pelas funções ρ e π conforme pode ser visto na Figura 11. A verificação se uma fórmula p é verdadeira num estado w do programa é dada pela relação $\mathcal{A}, w \models p$. Uma fórmula p é válida num modelo $\mathcal{A}$ se para todo estado $w \in W$, $\mathcal{A}, w \models p$. Além disso, a fórmula p será satisfeita num modelo $\mathcal{A}$, se existe um mundo w tal que $\mathcal{A}, w \models p$.

$$\rho(0) = \emptyset \tag{2.56}$$
$$\rho(\alpha; \beta) = \rho(\alpha) \circ \rho(\beta) \text{ (Composição de relações)} \tag{2.57}$$
$$\rho(\alpha \cup \beta) = \rho(\alpha) \cup \rho(\beta) \tag{2.58}$$
$$\rho(\alpha^*) = \rho(\alpha)^* \tag{2.59}$$
$$\rho(p?) = \{(w, w) : w \in \pi(p)\} \tag{2.60}$$
$$\pi(\top) = W \tag{2.61}$$
$$\pi(\bot) = \emptyset \tag{2.62}$$
$$\pi(p \wedge q) = \pi(p) \cup \pi(q) \tag{2.63}$$
$$\pi(\neg p) = W \setminus \pi(p) \tag{2.64}$$
$$\pi(\langle\alpha\rangle p) = \{w \in W : \exists v((w, v) \in \rho(\alpha) \text{ e } v \in \pi(p)\} \tag{2.65}$$
$$\pi([\alpha]p) = \{w \in W : \forall v((w, v) \in \rho(\alpha) \to v \in \pi(p))\} \tag{2.66}$$

Figura 11 – Semântica da Lógica Dinâmica.

Num exemplo genérico, seja a estrutura $\mathcal{A} = \langle W, \pi, \rho\rangle$ em PDL dada por $W = \{w_0, w_1\}$, $\pi = \emptyset$, $\rho(\alpha) = \{(w_0, w_0)\}$, $\rho(\beta) = \{(w_0, w1)\}$ e $\Sigma = \{\alpha, \beta\}$, representada graficamente pelo modelo da Figura 12. Nessa estrutura $\mathcal{A}$ é possível verificar que a partir do estado b, a fórmula $\varphi = [A^*](\langle A\rangle\top \wedge \langle B\rangle[A \cup B]\ \bot)$ é satisfeita, denotado por $\mathcal{A}, b \models \varphi$.

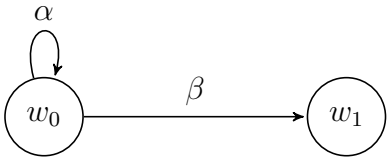

Figura 12 – Exemplo gráfico de um modelo em PDL [2].

Nota-se que a PDL foi projetada para descrever e verificar propriedades de programas e sistemas sequenciais. Uma aplicação importante da PDL foi proposta por Meyer [47],

dando origem a uma variante da PDL equivalente à Lógica Deôntica [12]. Nesta abordagem, o conceito de ação foi introduzido dentro da Lógica Deôntica como uma variação da noção de programas na PDL.

2.3.4 Lógica deôntica

A lógica deôntica tem como objetivo primário a expressão de conceitos legais e éticos [12]. Na área da Ciência da Computação, esta lógica tornou-se interessante para capturar o comportamento de sistemas. O nome dado à lógica deôntica vem da palavra grega *déon*, que expressa necessidade ou o que é preciso. Esta lógica pode ser entendida como a lógica das normas, expressando o que é obrigatório ou permitido [48]. A lógica deôntica é um tipo especial de lógica modal que aborda os conceitos normativos, sistemas de normas e o raciocínio sobre estas normas. Os conceitos normativos incluem deveres, possibilidades e impossibilidades, representados respectivamente, pela obrigação, permissão e proibição de ações [5].

O operador da lógica deôntica $P(\alpha)$ representa que a ação α é permitida. Uma ação α não permitida é denominada proibida e representada por $F(\alpha)$. Já a negação de uma ação α proibida é considerada obrigatória, denotada por $O(\alpha)$ [12]. Muitos estudos apontam a lógica deôntica como uma ramificação da lógica modal e seus conceitos de obrigação, permissão e proibição, entendidos como as modalidades de necessidade ($\Box$), possibilidade ($\Diamond$) e impossibilidade ($\neg\Diamond$) [49].

A literatura possui algumas variações da lógica deôntica de acordo com a percepção de seus autores. A mais utilizada é a lógica deôntica padrão (ou SDL), proposta por von Wright [12], com uma descrição modal baseada em estruturas de Kripke [37]. A SDL também foi a primeira lógica deôntica a ser especificada axiomaticamente e o seu esquema de regras e axiomas e definido na Figura 13.

$$(K) \qquad O(\varphi \to \psi) \to (O\varphi \to O\psi) \tag{2.67}$$
$$(D) \qquad \neg(O\varphi \wedge O\neg\varphi) \tag{2.68}$$
$$(N) \qquad \frac{\varphi}{O\varphi} \tag{2.69}$$
$$(P) \qquad P\varphi \leftrightarrow \neg O\neg\varphi \tag{2.70}$$
$$(F) \qquad F\varphi \leftrightarrow O\neg\varphi \tag{2.71}$$
$$(MP) \qquad \frac{\varphi, \varphi \to \psi}{\psi} \tag{2.72}$$
$$(PL) \qquad \text{todas as tautologias da lógica proposicional} \tag{2.73}$$

Figura 13 – Axiomas da Lógica Deôntica Padrão.

O axioma (K) representa a relação de obrigação entre implicações. O axioma (D)

não permite que existam conflitos entre as normas, determinando que se uma ação é obrigatória, esta não pode ser então proibida. A regra de necessidade modal, representada por (N), define que se uma ação φ é verdadeira, consequentemente $O(\varphi)$ também é considerada verdadeira. Além disso, são definidos os operadores de permissão (P) e de proibição (F) baseados no operador de obrigação. Por fim, o axioma (PL) indica que todas as tautologias da lógica proposicional são válidas na SDL e o axioma (MP) representa o *modus ponens* na SDL.

A semântica dos operadores da SDL, como as outras lógicas modais, é baseada na noção de mundos possíveis e pode ser representada por uma estrutura baseada em Kripke denotada por $M = \langle W, R, L \rangle$ onde:

- W é o conjunto de mundos possíveis;
- R é a relação de acessibilidade $R \subseteq W \times W$ que contém os mundos possíveis e alcançáveis a partir de um mundo w;
- L é a função de valoração que informa se uma proposição p existe no mundo w.

Seja um modelo de Kripke $\mathcal{M}$ e um mundo w, a relação de satisfação $\models$ dos operadores deônticos é definida conforme a Figura 14. A relação de satisfação do operador de obrigação O é baseada no operador de necessidade $(\Box)$ da lógica modal. Por isso, a fórmula $O(\alpha)$ é válida num mundo w se em todos os mundos acessíveis a partir de w, a proposição α é válida. Da mesma forma, o operador de permissão P tem semântica similar ao operador modal de possibilidade $(\Diamond)$ e o operador de proibição F é válido somente se α não é válida em todos os mundos acessíveis por w.

$$\mathcal{M}, w \models O(\alpha) \iff \forall w' \text{ tal que } (w, w') \in R \text{ e } \alpha \in L(w') \tag{2.74}$$
$$\mathcal{M}, w \models P(\alpha) \iff \exists w' \text{ tal que } (w, w') \in R \text{ e } \alpha \in L(w') \tag{2.75}$$
$$\mathcal{M}, w \models F(\alpha) \iff \forall w' \text{ tal que } (w, w') \in R \text{ e } \alpha \notin L(w') \tag{2.76}$$

Figura 14 – Semântica da Lógica Deôntica.

As expressões escritas em SDL podem ser interpretadas textualmente de duas maneiras: "É obrigatório que..." seguido de uma sentença descritiva; ou "É obrigatório a..." seguido de um verbo que representa uma ação ou atividade. Nesta perspectiva, as expressões da SDL podem representar dois tipos de lógica deôntica: a que expressa uma situação e outra que expressa uma ação [50]. A lógica deôntica que se preocupa com situações e relacionamentos é chamada de *ought-to-be* ou "deve ser", já a lógica que se preocupa com o que deve ser feito, é chamada de *ought-to-do* ou "deve fazer". A diferença entre as duas visões vai depender do tipo de norma que precisa ser descrita. No caso de

contratos eletrônicos, que tratam em sua grande maioria a ação realizada pelos indivíduos, a abordagem *ought-to-do* é a mais indicada.

A abordagem proposta por Meyer [47] apresenta algumas melhorias na SDL ao realizar a redução da lógica deôntica para a lógica dinâmica proposicional. A principal mudança realizada é a formalização do conceito de ação e o uso de expressões *ought-to-do* permitindo que as ações sejam expressas formalmente. Os três operadores deônticos, que contribuem para expressar o conceito *ought-to-do*, são representados na lógica dinâmica conforme a Figura 15.

$$F(\alpha) \iff [\alpha]V \tag{2.77}$$
$$P(\alpha) \iff \neg F\alpha \quad (\text{ou } \langle\alpha\rangle\neg V) \tag{2.78}$$
$$O(\alpha) \iff F(\overline{\alpha}) \quad (\text{ou } [\overline{\alpha}]V) \tag{2.79}$$

Figura 15 – Representação da Lógica Deôntica pela Lógica Dinâmica.

Uma ação na abordagem em questão tem o mesmo significado de um programa em PDL, ou seja, pode ser executada e mudar o valor de uma proposição. A proibição de uma ação é reduzida a um operador dinâmico de necessidade em PDL onde a execução da ação α leva o sistema a um estado de violação representado por V. A partir desta redução da proibição são definidos a permissão, como a negação da proibição e a obrigação como a proibição da negação de α.

A lógica deôntica foi proposta originalmente para definir estruturas normativas da lei e possibilitar a sua verificação. No entanto, essa lógica pode ser aplicada nas mais diversas áreas onde deseja-se verificar o comportamento de sistemas. Alguns exemplos de aplicações da lógica deôntica são [51]:

- **Tolerância a falhas em sistemas:** nenhum sistema de computador é livre de falhas e muitas vezes é recomendado que seja definido o que acontece quando ocorre uma anomalia no sistema. Esta técnica é conhecida como tratamento de exceções, onde o sistema tolerante a falhas pode levar a um comportamento que garanta seu funcionamento mesmo depois de um problema ocorrido.
- **Normas de comportamento de usuários:** normalmente nenhum sistema está pronto para tratar todos os tipos de comportamentos dos usuários, pois suas ações são muitas vezes imprevisíveis. O tratamento de erros nos sistemas utilizando a lógica deôntica pode ajudar a controlar os erros causados pelos usuários já que é possível determinar o comportamento aceito e com isso indicar as ações do sistema.
- **Especificação de políticas:** com a lógica deôntica é possível descrever o comportamento dos componentes dentro das organizações. Assim sendo, a definição das

regras e políticas das organizações precisam definir o que pode ou não ser realizado e o que ocorre quando uma norma não é respeitada. Neste caso, a lógica deôntica pode ser utilizada para definir políticas não ambíguas e indicar as consequências de uma violação.

- **Especificação de normas legais:** a representação da lei é uma área vasta e muito explorada da lógica deôntica mesmo antes de sua automação. Desta maneira, atos e leis são formalizados e manipulados de acordo com as regras de alguma lógica deôntica. A formalização das normas através da lógica permite que sistemas sejam capazes de auxiliar advogados e juízes nas mais diversas tarefas que envolvem leis e normas.

Nota-se que essas aplicações envolvem algum tipo de especificação de normas e comportamento. Nenhuma das aplicações listadas, no entanto, precisa implementar a especificação lógica deôntica em forma executável num computador. Mesmo que essa ideia seja interessante, principalmente no caso de sistemas tolerantes a falhas e automação legal, ainda é utilizada apenas na especificação e não na implementação. No caso da contratação eletrônica, é possível descrever um contrato utilizando lógica deôntica e aplicar técnicas de verificação formal [24], monitoramento [31] ou detecção de conflitos [52].

2.4 A linguagem de contratos $\mathcal{CL}$

A linguagem para contratos, denominada $\mathcal{CL}$ [53], foi desenvolvida para a representação formal de contratos legais, serviços *web*, interfaces e protocolos de comunicação. Essa linguagem é expressiva o suficiente para capturar o comportamento de contratos e permitir sua verificação formal. Alguns estudos baseados nessa linguagem incluem verificação por meio de *model checking* [24], detecção de conflitos [52] e monitoramento de contratos [31]. A $\mathcal{CL}$ é baseada em ações síncronas (concorrentes), nos princípios da lógica deôntica para definir normas [12], e na lógica dinâmica proposicional para representar as ações num sistema [44]. A parte deôntica da $\mathcal{CL}$ possui os operadores de permissão, obrigação e proibição, bem como a compensação sobre a violação de obrigações ou proibições. Já a parte dinâmica expressa as cláusulas válidas após a execução de alguma ação. Uma característica importante da $\mathcal{CL}$ é a captura de conceitos e propriedades naturais encontradas em contratos legais e eletrônicos buscando evitar os principais paradoxos deônticos [54].

2.4.1 Sintaxe da $\mathcal{CL}$

A construção de fórmulas na linguagem $\mathcal{CL}$ é baseada nos operadores das lógicas proposicional, dinâmica e deôntica. A sintaxe da $\mathcal{CL}$ é definida pela gramática descrita na Figura 16, onde $\mathcal{C}$ é um contrato ou uma de suas cláusulas.

$$\mathcal{C} := \phi \mid O_{\mathcal{C}}(\alpha) \mid P(\alpha) \mid F_{\mathcal{C}}(\alpha) \mid \mathcal{C} \wedge \mathcal{C} \mid [\beta]\mathcal{C} \mid \bot \tag{2.80}$$
$$\alpha := a \mid \mathbf{0} \mid \mathbf{1} \mid \alpha \times \alpha \mid \alpha \cdot \alpha \mid \alpha + \alpha \tag{2.81}$$
$$\beta := a \mid \mathbf{0} \mid \mathbf{1} \mid \beta \times \beta \mid \beta \cdot \beta \mid \beta + \beta \mid \beta^* \mid \varphi? \tag{2.82}$$
$$\varphi := \phi \mid \mathbf{0} \mid \mathbf{1} \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \neg\varphi \tag{2.83}$$

Figura 16 – Gramática da $\mathcal{CL}$.

As cláusulas podem conter os operadores lógicos, como na lógica proposicional, representados pela conjunção $\wedge$ e pelo valor lógico falso, denotado por $\bot$. As constantes proposicionais são denotadas por ϕ. Uma constante proposicional descreve uma situação como por exemplo "o produto foi entregue" ou "o banco repassou o valor do frete".

Uma ação é considerada uma tarefa, ou programa da PDL, obrigatória, proibida ou permitida, de acordo com a modalidade deôntica associada. As ações executadas dentro dos operadores deônticos são representadas por α. Já as ações β são utilizadas nos operadores dinâmicos juntamente com o teste lógico, denotado por φ. As ações deônticas consistem num conjunto de ações básicas (ou atômicas) denotado por $\mathcal{A}_{\mathcal{B}}$. Além disso, existem os tipos especiais de ações que não pertencem ao conjunto $\mathcal{A}_{\mathcal{B}}$, representadas por $\mathbf{0}$ e $\mathbf{1}$. A ação $\mathbf{0}$ representa uma violação num contrato e a ação $\mathbf{1}$, chamada ação de salto (*skip*), representa a execução de qualquer ação do conjunto $\mathcal{A}_{\mathcal{B}}$ e significa que a ação executada, em particular, não é importante.

As ações dentro das fórmulas podem ser combinadas pelos operadores: $+$, quando há uma escolha entre as ações; $\cdot$, quando existe uma sequência de ações; e $\times$, quando ocorrem ações síncronas (ou concorrentes). O conjunto de ações compostas $\mathcal{A}_{\mathcal{D}}$ é composto por ações básicas, ações especiais $\mathbf{0}$ e $\mathbf{1}$, e pela combinação de operadores. O conjunto de ações concorrentes, denotado por $\mathcal{A}_{\mathcal{B}}^{\times}$, é formado por ações compostas que utilizam apenas o operador $\times$ e obtidas a partir da combinação das ações básicas. Na $\mathcal{CL}$, as ações atômicas de $\mathcal{A}_{\mathcal{B}}$ são representadas por símbolos e, intuitivamente, executadas por um indivíduo. Nesta linguagem, os indivíduos são incorporados nas ações, seguindo o conceito da lógica deôntica padrão [24].

As modalidades deônticas $O_{\mathcal{C}}(\alpha)$, $F_{\mathcal{C}}(\alpha)$, $P(\alpha)$ representam respectivamente os operadores de obrigação, proibição e permissão da lógica deôntica padrão, onde α é uma ação e $\mathcal{C}$ representa a penalidade ou reparação quando a cláusula é violada após a execução da ação. Note que não existe nenhum tipo de reparação nas permissões já que estas são facultativas e não implicam numa violação. Esse conceito de reparação, representa as cláusulas contrárias ao dever (CTD ou *contrary-to-duty*) e contrárias à proibição (CTP ou *contrary-to-prohibition*). As cláusulas CTD representam situações em que há uma violação de obrigação primária e implica em executar uma obrigação secundária. Essa

obrigação secundária é uma reparação ou penalidade da violação gerada pela obrigação primária. O mesmo ocorre para o CTP que trata da violação de uma proibição [55].

Obrigações ou proibições sem reparações, $O_{\perp}(\alpha)$ e $F_{\perp}(\alpha)$, também podem ser representadas, onde o símbolo $\perp$ indica a ausência de reparação, ou seja, que o contrato será violado. Obrigações (ou proibições) sem reparações são chamadas de categóricas, pois não podem ser violadas de forma alguma. Para facilitar a sintaxe, as obrigações categóricas podem ser representadas por $O(\alpha)$ e, do mesmo modo, as proibições como $F(\alpha)$.

Os operadores deônticos da $\mathcal{CL}$ são aplicados apenas sobre as ações, ao contrário da SDL que permite a aplicação das modalidades deônticas tanto em ações quanto em situações. Já os operadores dinâmicos são aplicados sobre as ações dinâmicas β. Na expressão $[\beta]O(\alpha)$, por exemplo, após β ser executada, α é obrigatoriamente executada. A lógica dinâmica proposicional na $\mathcal{CL}$ também proporciona a interação entre as ações e as fórmulas. Obrigações condicionais (ou permissões e proibições) podem ser expressas de duas formas: $[\beta]O(\alpha)$, onde após β ser executada, obrigatoriamente α deve ser executada; $[C?]O(\alpha)$, simula a implicação $C \rightarrow O(\alpha)$. A sequência $\varphi?.\alpha$ indica uma ação de guarda, onde α é executada se φ é verdadeira.

2.4.2 A semântica da $\mathcal{CL}$

A semântica da $\mathcal{CL}$ captura o comportamento da execução de uma sequência de ações de forma que a cláusula do contrato seja respeitada. Esta semântica é utilizada na técnica de monitoramento proposta por Kyas, Prisacariu e Schneider [31], onde as sequências de ações, chamadas de *traces*, que violam um contrato são identificadas. Um *trace* σ é definido por uma sequência ordenada de ações concorrentes e a ação especial **1** (*skip*), como $a_0, a_1, \ldots$, onde $a_i \in \mathcal{A}_{\mathcal{B}}^{\times}$, $i \geq 0$. O conjunto de ações concorrentes é obtido a partir da combinação das ações básicas usando o conector de concorrência $\times$. Sejam as ações básicas $a, b, c \in \mathcal{A}_{\mathcal{B}}$, as ações concorrentes são dadas por $\mathcal{A}_{\mathcal{B}}^{\times} = \{a, b, c, a \times b, b \times c, a \times c, a \times b \times c\}$.

Um *trace* é, formalmente, definido por um mapeamento $\sigma : \mathbb{N} \rightarrow \mathcal{A}_{\mathcal{B}}^{\times} \cup \{\mathbf{1}\}$, tal que dado $i \in \mathbb{N}$, σ retorna as ações concorrentes de $\mathcal{A}_{B}^{\times}$ da posição i do *trace*. O tamanho do *trace* é denotado por $|\sigma|$. Um *trace* infinito que possui apenas ações de salto **1** a partir de uma determinada posição i é considerado finito de tamanho i. Um *trace* vazio é denotado por ε. Os elementos do *trace* são representados por $\sigma(i)$, onde i é uma posição. Para expressar um *subtrace* finito utiliza-se $\sigma(i..j)$, onde i é o início e j é o fim, e para um *trace* infinito basta informar o seu início, onde $\sigma(i...)$ representa um *subtrace* iniciado em i. A operação de concatenação de dois *traces* σ' e σ'' é denotada por $\sigma'\sigma''$ e só é possível se e somente se σ' é finito. O conteúdo de uma posição i de um *trace* concatenado é definido por $\sigma'\sigma''(i) = \sigma'(i)$ se $i <\mid \sigma' \mid$ e $\sigma'\sigma''(i) = \sigma''(i- \mid \sigma' \mid)$ para $i \geq \mid \sigma' \mid$.

A semântica baseada em *traces* infinitos da $\mathcal{CL}$ é definida parcialmente na Figura 17. A relação de satisfação $\models$ é aplicada sobre o par ordenado $(\sigma, \mathcal{C})$, onde σ é um

trace e $\mathcal{C}$ é um contrato, e $\sigma \models \mathcal{C}$ significa que σ é satisfeito em $\mathcal{C}$. Caso contrário, o *trace* viola o contrato, denotado por $\sigma \not\models \mathcal{C}$. Para ilustrar a semântica da $\mathcal{CL}$, tome o operador $O_{\mathcal{C}}(\alpha_\times)$ na Equação (2.88). A obrigação de executar α é satisfeita se $\sigma(0)$ tem a ação $\alpha_\times$. Caso contrário ocorre uma violação e o restante do trace deve satisfazer a reparação $\mathcal{C}$, denotada por $\sigma(1\dots) \models \mathcal{C}$. A semântica completa sobre os operadores da $\mathcal{CL}$ pode ser vista na Figura 58 do Anexo A.

$$\sigma \models \top \tag{2.84}$$
$$\sigma \not\models \bot \tag{2.85}$$
$$\sigma \models \mathcal{C}_1 \wedge \mathcal{C}_2 \iff \sigma \models \mathcal{C}_1 \text{ e } \sigma \models \mathcal{C}_2 \tag{2.86}$$
$$\sigma \models [\alpha_\times]\mathcal{C} \iff (\alpha_\times \subseteq \sigma(0) \text{ e } \sigma(1\dots) \models \mathcal{C}) \text{ ou } \alpha_\times \not\subseteq \sigma(0) \tag{2.87}$$
$$\sigma \models O_{\mathcal{C}}(\alpha_\times) \iff \alpha_\times \subseteq \sigma(0) \text{ ou } \sigma(1\dots) \models \mathcal{C} \tag{2.88}$$
$$\sigma \models F_{\mathcal{C}}(\alpha_\times) \iff \alpha_\times \not\subseteq \sigma(0) \text{ ou } (\alpha_\times \subseteq \sigma(0) \text{ e } \sigma(1\dots) \models \mathcal{C}) \tag{2.89}$$

Figura 17 – Semântica de *traces* infinitos da $\mathcal{CL}$ (parcial).

A seguir um exemplo simples de contrato que ilustra a semântica da $\mathcal{CL}$:

> "*Se o cliente exceder o limite de banda da internet então ele deve pagar uma multa, ou ele deve atrasar o pagamento mas notificar o provedor via e-mail. Em caso de quebra das duas cláusulas de penalidade, o cliente deve pagar em dobro*" [56].

Este exemplo pode ser representado em $\mathcal{CL}$ como:

$$[e]O_{O_\perp(p\cdot p)}(p + a \times n), \tag{2.90}$$

onde e significa exceder o limite da banda, p é pagar o valor proposto, a é atrasar o pagamento e n representa a notificação ao provedor.

Alguns *traces* que respeitam e violam este contrato são descritos a seguir:

- $\sigma = \{e, p\}$; representa "exceder o limite de banda e pagar por isso", respeitando o contrato, pois respeita a obrigação principal.
- $\sigma = \{e, a, p, p\}$; representa "exceder o limite de banda então atrasar o pagamento e pagar o dobro". Também respeita o contrato, pois ao exceder o limite de banda, atrasa o pagamento mas não notifica provedor e paga em dobro.
- $\sigma = \{p, p, p\}$; representa "Pagar três vezes" e respeita o contrato, pois, desde que o *trace* não inicie com a ação e, não existe obrigação.

- Os traces $\sigma = \{e, e, e\}$ (constantemente excedendo o limite) e $\sigma = \{e, a, a\}$ (excedendo o limite e constantemente atrasando o pagamento) são exemplos de violação do contrato.

Devido à $\mathcal{CL}$ ser baseada na PDL e na SDL, algumas propriedades e composições importantes destas duas lógicas também são observadas nesta linguagem de contratos, conforme são descritas nas Figuras 57 e 56 do Anexo A. As propriedades e decomposições da $\mathcal{CL}$ são utilizadas pelo algoritmo de detecção de conflitos descrito na Seção 3.1. Este algoritmo auxilia o processo de detecção, bem como o monitoramento de contratos, com base nos conflitos e violações definidos em $\mathcal{CL}$. As definições sobre conflito e violação são descritas a seguir:

- **Relação de conflito:** é uma relação simétrica e irreflexiva sobre as ações básicas de $\mathcal{A}_\mathcal{B}$ denotada por $\# \subseteq \mathcal{A}_\mathcal{B} \times \mathcal{A}_\mathcal{B}$. Essa relação é comumente usada em contratos legais onde duas ações não podem ocorrer ao mesmo tempo. Num exemplo simples, uma ação "dirigir" conflita, segundo as leis de trânsito, com as ações "beber" e "falar ao celular". A equação dessa relação é então dada por $a_\# b \rightarrow a \times b = \mathbf{0} \mid \forall a, b \in \mathcal{A}_\mathcal{B}$, onde a representa a ação "dirigir" e b representa "beber". Da mesma forma, a equação $c_\# a$ determina que a ação "falar ao celular", representada por c, conflita com "dirigir". Note que não há transitividade na relação de conflito, não ocorrendo um conflito com "falar ao telefone" e "beber", por exemplo.

- **Violação de contrato:** só ocorre com operadores de Obrigação (O) ou de Proibição (F). Como no operador de Permissão (P) a ação é facultativa, não há garantia de sua execução. Quando uma ação obrigatória não é executada ou quando uma ação proibida é executada, ocorre uma violação. No entanto, de acordo com os conceitos de penalidade e reparação, descritos anteriormente, uma compensação pode ser imposta para evitar uma violação em todo contrato. A $\mathcal{CL}$ define um conceito de complemento de ação que representa a violação sobre uma obrigação. O complemento de uma ação é a execução de qualquer outra ação ou sequência de ações do conjunto de ações básicas $\mathcal{A}_\mathcal{B}$, denotado por $\overline{\alpha}$, e obtido pela função $\overline{f} : \mathcal{A}_\mathcal{D} \rightarrow \mathcal{A}_\mathcal{D}$. Seja um conjunto de ações básicas $\mathcal{A}_\mathcal{B} = \{a, b\}$, alguns exemplos de complementos são: $\overline{a} = b$ é qualquer ação diferente de a, portanto, a única alternativa é b. Já no caso de $\overline{a \cdot b} = b + a \cdot a$, o complemento das ações a e b em sequência pode ser a ação b ou a ação a duas vezes seguidas.

2.4.3 Aplicações da $\mathcal{CL}$

Com a representação de contratos em $\mathcal{CL}$ é possível realizar a verificação formal destas especificações de várias maneiras. Algumas formas de se aplicar a $\mathcal{CL}$ são apresentadas a seguir:

- **Verificação formal de contratos.** Uma abordagem proposta por Pace, Prisacariu e Schneider [24] especifica contratos em $\mathcal{CL}$ e transforma as fórmulas para uma variação do μ-*calculus* denominada $\mathcal{C}\mu$. A técnica se baseia nos seguintes passos:

  1. Traduzir o contrato convencional para $\mathcal{CL}$;
  2. Traduzir sintaticamente as especificações em $\mathcal{CL}$ para $\mathcal{C}\mu$;
  3. Obter manualmente um modelo de Kripke das fórmulas $\mathcal{C}\mu$;
  4. Traduzir o modelo para a linguagem de uma ferramenta para *model checking*, no caso, o NuSMV;
  5. Realizar a verificação do modelo;
  6. Interpretar o contra-exemplo, se existir, como uma cláusula $\mathcal{CL}$ e ajustar o contrato.

  Por se tratar de uma primeira aplicação da técnica, alguns passos são manuais, principalmente a construção do modelo e a interpretação do contra-exemplo.

- **Monitoramento de contratos em tempo de execução.** Na abordagem de monitoramento de contratos proposta por Kyas, Prisacariu e Schneider [31], o monitor do contrato é um autômato finito com saída gerado automaticamente usando a semântica de *traces* da $\mathcal{CL}$. O monitoramento consiste nas seguintes etapas:

  1. Definir uma semântica baseada em *traces* para a $\mathcal{CL}$;
  2. Obter, com base na $\mathcal{CL}$, um *Alternating Büchi Automata* que aceita seus *traces*;
  3. Criar um *Büchi Automata* não determinístico do autômato gerado no passo anterior;
  4. Obter o monitor do contrato criando uma máquina de *Moore* que tem como entrada a fita de ações realizadas e como saída informa se o contrato foi violado ou não.

- **Detecção de conflitos em contratos.** Com o objetivo de auxiliar o processo de negociação, esta abordagem proposta por Fenech [5] realiza a análise num contrato em $\mathcal{CL}$ buscando conflitos normativos, onde uma cláusula sobrepõe outra, invalidando o contrato. A análise é realizada com a semântica de *traces*, convertendo o contrato num autômato para procurar cláusulas conflitantes. O autômato obtido além de permitir a análise de conflitos ainda pode ser utilizado para o monitoramento do contrato. Essa abordagem realiza as seguintes tarefas:

  1. Traduzir os contratos convencionais para $\mathcal{CL}$;
  2. Obter as decomposições das cláusulas do contrato;

3. Criar um autômato representando as cláusulas a partir das decomposições do contrato;
4. Buscar nos estados do autômato os conflitos normativos.

Além dessas aplicações existem outras pesquisas realizadas com a $\mathcal{CL}$, como a tradução automática de contratos baseado processamento de linguagem natural [57] e técnicas para a modelagem visual de contratos [58].

3 TRABALHOS RELACIONADOS

Uma das abordagens para verificação de conflitos em contratos escritos em $\mathcal{CL}$ [53], baseada na lógica deôntica padrão [12], foi proposta por Fenech [14]. No entanto, essa proposta não permite a identificação das responsabilidades sobre uma determinada ação no contrato. Já o trabalho de Herrestad e Krogh [7] propõe uma extensão da lógica deôntica padrão, a lógica deôntica relativizada, para que os envolvidos no contrato sejam identificados individualmente. A lógica deôntica relativizada personaliza os operadores deônticos, permitindo que cenários mais complexos com a participação de vários indivíduos sejam descritos de forma precisa. Porém, não há conhecimento de uma extensão que compreenda as demais características da $\mathcal{CL}$, como o conceito de penalidade/compensação ou algum mecanismo para evitar paradoxos deônticos.

As propriedades em contratos bilaterais, em geral, são mais simples que as propriedades apresentadas em contratos multilaterais, onde dois ou mais indivíduos estão envolvidos. As dificuldades nos contratos multilaterais estão relacionadas principalmente com a representação do contrato [7] e a identificação de responsáveis pelas violações [13]. Por isso, a detecção de conflitos em contratos multilaterais se torna mais complexa, exigindo que as partes envolvidas sejam identificadas para que determinados tipos de conflitos sejam detectados.

A seguir são apresentados os trabalhos que estão mais intimamente relacionados com esta pesquisa. Inicialmente é apresentada a técnica de detecção de conflitos em contratos descritos em $\mathcal{CL}$ e posteriormente técnicas de relativização da lógica deôntica.

3.1 Análise de conflitos em contratos bilaterais

Os conflitos normativos estão presentes tanto em contratos convencionais quanto em contratos eletrônicos. A condição de conflito em contratos caracteriza-se quando suas cláusulas não podem ser satisfeitas. Se uma cláusula $\mathcal{C}_1$, por exemplo, proíbe uma ação α e uma cláusula $\mathcal{C}_2$ obriga a execução dessa mesma ação α, torna-se impossível a satisfação do contrato, pois ao atender uma cláusula, a outra é violada. Diante deste cenário, foi proposto por Fenech [5] um método de detecção de conflitos em contratos eletrônicos especificados na linguagem de contratos $\mathcal{CL}$.

3.1.1 Definição de conflitos

Um contrato possui conflitos quando ocorre uma contradição entre normas. Essa situação pode ser representada pela incompatibilidade entre as normas de proibição com normas de permissão ou obrigação [5].

Os conflitos podem ser verificados tanto entre operadores deônticos conflitantes sobre uma mesma ação quanto entre ações distintas, situação na qual estas ações seguem o princípio da exclusão mútua, podendo ser executadas ao mesmo tempo. Essa relação de conflito é representada em $\mathcal{CL}$ pelo símbolo #. Desta maneira, a exclusão entre as ações α e β pode ser representada por $\alpha\#\beta$.

As situações de conflito e ações conflitantes que podem ocorrer em um contrato são definidas da seguinte forma:

- Obrigar e proibir a mesma ação: $O(\alpha) \wedge F(\alpha)$
- Permitir e proibir a mesma ação: $P(\alpha) \wedge F(\alpha)$
- Obrigar ações conflitantes: $O(\alpha) \wedge O(\beta)$, com $\alpha\#\beta$
- Obrigar e permitir ações conflitantes: $O(\alpha) \wedge P(\beta)$, com $\alpha\#\beta$.

Existe também um outro tipo de conflito, chamado conflito fraco, em que a violação pode ser evitada. Na fórmula $O(\alpha + \beta) \wedge F(\alpha)$, por exemplo, existe uma escolha na obrigação em executar a ação α ou a ação β. Caso seja escolhida a ação α, um conflito vai ocorrer com a proibição de α, mas se a ação β é executada, o conflito não se caracteriza.

A detecção de conflitos em $\mathcal{CL}$ tem como base as técnicas de verificação de modelos (*model checking*) [38], onde propriedades desejadas podem ser verificadas no modelo que especifica o contrato. A propriedade que define um contrato livre de conflitos pode ser descrita como: *"dada qualquer sequência de ações que não viola o contrato, a execução do contrato não termina num estado que ocorre um conflito"*. Para todas os caminhos de computação válidos no modelo, nenhum destes caminhos pode chegar a um estado conflitante. A semântica de *traces* da $\mathcal{CL}$ é então adaptada por Fenech [14] para possibilitar essa verificação.

3.1.2 A semântica de *traces* para detecção de conflitos

Com base na semântica convencional de *traces* da $\mathcal{CL}$, descrita na Seção 2.4, uma extensão é proposta por Fenech [5] de forma que as informações deônticas sejam preservadas. A extensão inclui a informação deôntica no *trace*, informando quais obrigações, permissões e proibições são válidas em uma determinada posição e a possibilidade da semântica aceitar *traces* finitos que satisfazem o contrato.

A partir dessa semântica é possível obter automaticamente um autômato que aceite os *traces* do contrato, incluindo as informações necessárias para a detecção de conflitos. Para isso, existe o *trace* que contém a informação deôntica do contrato e o *trace* formado com o conjunto das ações do contrato. Para o *trace* deôntico, um contrato com o alfabeto de ações básicas $\mathcal{A}_{\mathcal{B}}$, incorpora um alfabeto D_a, formado por operadores deônticos O_a,

F_a e P_a para cada ação $a \in \mathcal{A}_\mathcal{B}$. Além disso, se α é um conjunto de ações concorrentes, denota-se $O_\alpha = \{O_a,\ \forall a \in \alpha\}$.

O *trace* deôntico é denotado por σ_d e cada posição consiste de um conjunto de elementos de 2^{D_a}. Essa definição é necessária para diferenciar uma composição de escolha entre ações com uma composição de ações concorrentes. Por exemplo, o contrato $C' = O(a+b) \wedge F(b)$ tem um *trace* deôntico $\sigma'_d = \langle\{\{O_a, O_b\}, \{F_b\}\}\rangle$ que não está em conflito. Por outro lado, o contrato $C'' = O(a \times b) \wedge F(b)$ com o *trace* $\sigma''_d = \langle\{\{O_a\}, \{O_b\}, \{F_b\}\}\rangle$ está em conflito, pois obriga a execução tanto de a quanto de b ao mesmo tempo em que proíbe a execução da ação b.

A semântica é expressa pela relação de satisfação $\sigma,\ \sigma_d\ \models\ \mathcal{C}$, onde $\mathcal{C}$ é um contrato em $\mathcal{CL}$, σ é o *trace* finito de ações concorrentes de $\mathcal{A}^{\times}_{\mathcal{B}}$ e σ_d é o *trace* finito de informações deônticas formado por um conjunto dos elementos de 2^{D_a}. A relação de satisfação $\sigma, \sigma_d \models \mathcal{C}$ é considerada bem formada quando os *traces* têm o mesmo tamanho, ou seja, $\mid \sigma \mid = \mid \sigma_d \mid$, onde $\mid \sigma \mid$ é a função que retorna o tamanho do *trace*. Uma relação de satisfação bem formada é necessária pois cada posição de σ está relacionada com a mesma posição em σ_d. No caso de *traces* vazios, ou seja, $\mid \sigma \mid = 0$ e $\mid \sigma_d \mid = 0$, o contrato nunca é violado [14]. A concatenação de dois *traces* finitos é denotada por $\sigma'_d; \sigma''_d$. A união de dois *traces*, dada pelo operador $\cup$, é definida como $\sigma_d \cup \sigma'_d = \sigma_d(0) \cup \sigma'_d(0); \sigma_d(1) \cup \sigma'_d(1); \ldots; \sigma_d(n) \cup \sigma'_d(n)$, assumindo que $\mid \sigma_d \mid = \mid \sigma'_d \mid$ e considerando que $\sigma_d(i)$ é a posição i do *trace* σ_d. Esta operação de união é importante para representar a semântica de conjunção de cláusulas do contrato.

Para exemplificar o funcionamento da semântica, considere o contrato $\mathcal{C} = [a]O(b) \wedge [b]F(b)$. Um dos *traces* que satisfazem o contrato $\sigma = \langle\{a\}, \{b\}\rangle$ em conjunto com o *trace* deôntico $\sigma_d = \langle\emptyset, \{O_b\}\rangle$, resultam em $\sigma, \sigma_d \models \mathcal{C}$. Os *traces* informados satisfazem o contrato $\mathcal{C}$, pois em $\sigma(0)$ é executada a ação a sem a informação deôntica, $\sigma_d(0) = \emptyset$. Em $\sigma(1)$ é executada a ação b que satisfaz a informação deôntica presente em $\sigma_d(1)$. Por outro lado, os *traces* $\sigma = \langle\{a \times b\}, \emptyset, \emptyset\rangle$ e $\sigma_d = \langle\emptyset, \{\{O_b\}, \{F_b\}\}, \emptyset\rangle$ levam o contrato a uma situação de conflito. Neste caso existe um conflito pois a ação b é obrigatória e proibida ao mesmo tempo. O simbolo $\emptyset$ presente em uma posição do *trace* representa a ausência de informação deôntica no caso de σ_d ou de uma ação a ser executada, no caso de σ. No *trace* deôntico σ_d, esta situação ocorre quando o contrato é iniciado por um operador dinâmico de necessidade, como em $[a]O(b)$, pois neste momento a obrigação ainda não é válida. A semântica completa da relação de satisfação dos operadores da $\mathcal{CL}$ é apresentada na Figura 59 no Anexo A.

3.1.3 Algoritmo de detecção de conflitos

A análise de um contrato em $\mathcal{CL}$ é baseada na busca por *traces* com a ocorrência de um conflito entre os operadores deônticos em alguma posição de σ_d. O processo ocorre

em duas etapas: a geração de um autômato que especifica um contrato $\mathcal{C}$ em $\mathcal{CL}$, cuja linguagem aceita corresponde aos *traces* determinados pela semântica do contrato; e a aplicação do algoritmo de detecção de conflitos sobre o autômato utilizando as regras de um contrato livre de conflitos. Um contrato está livre de conflitos se para todo σ e σ_d tal que $\sigma, \sigma_d \models \mathcal{C}$, todo elemento $D \in \sigma_d(i)$, para $0 \leq i \leq \mid \sigma_d \mid$, não esteja em conflito com qualquer outra informação deôntica em $\sigma_d(i)$. Seja D' um elemento de $\sigma_d(i)$, D não está em conflito com D' se existir pelo menos um elemento $d \in D$ tal que:

$$\forall D'(D' \neq \{F_a\} \text{ e } (D' \neq \{P_b\} \text{ e } a\#b) \text{ e } D' \subseteq \{O_a \mid a \in A\} \rightarrow \exists O_b \in D' \mid (a,b) \notin \#), \text{ com } d = O_a \tag{3.1}$$

$$\nexists d' \in D' \mid d' = P_a \text{ ou } d' = O_a, \text{ com } d = F_a \tag{3.2}$$

$$\nexists d' \in D' \mid d' = F_a \text{ ou } (d' = O_b \text{ e } a\#b), \text{ com } d = P_a \tag{3.3}$$

A análise de um contrato em $\mathcal{CL}$ consiste em verificar se o autômato que especifica o contrato está livre de conflitos. O algoritmo de verificação de conflitos procura por estados do autômato com informação deôntica conflitante.

Seja um contrato $\mathcal{C}$, com o alfabeto de ações $\mathcal{A}_\mathcal{B}$ e o alfabeto deôntico D_a, o autômato A que modela este contrato é dado por

$$A(\mathcal{C}) = \langle S, \mathcal{A}_\mathcal{B}^\times, s_0, T, V, l, \delta \rangle, \tag{3.4}$$

onde:

- S é o conjunto de estados;
- $\mathcal{A}_\mathcal{B}^\times$ é o conjunto de ações concorrentes obtidas do alfabeto de ações $\mathcal{A}_\mathcal{B}$;
- s_0 é o estado inicial;
- $T \subseteq S \times \mathcal{A}_\mathcal{B}^\times \times S$ é a relação das transições rotuladas pelas ações concorrentes entre os estados;
- V é o estado que representa a violação do contrato;
- $l : S \rightarrow \mathcal{CL}$ é uma função de rotulação dos estados com as cláusulas em $\mathcal{CL}$; e
- $\delta : S \rightarrow 2^{D_a}$ é uma função de rotulação de estados com um conjunto de informações deônticas.

Uma execução (ou sequência de estados) é aceita por um autômato quando nenhum dos estados da execução é V. De forma parecida, um autômato aceita uma palavra w, que consiste em uma sequência de ações, se nenhuma dessas ações é o rótulo de uma transição que contenha o estado V. Esta condição é expressa como $ACEITA(A(\mathcal{C}), w)$.

O Algoritmo 1 constrói o autômato $A(\mathcal{C})$ da seguinte forma:

1. Cria o estado s_0 onde $l(s_0) = C$, com todas as cláusulas do contrato;

2. Recebe o estado s_0, corrente, como parâmetro e verifica três casos:

   a) Se $l(s) = 1$, então o contrato é satisfeito e não restam cláusulas a serem processadas. O estado atual s é marcado como SAT (satisfação), e uma transição é criada para este estado s;

   b) Se $l(s) = 0$, então há uma violação no contrato, o estado atual s é marcado como V (violação), e uma transição é criada para este estado; ou

   c) Caso contrário, a função $f(l(s), \alpha)$ é executada para cada elemento do alfabeto de ações, e para cada retorno, representado pela cláusula $\mathcal{C}'$, da função f é verificado se existe um estado s' que represente este retorno, ou seja, $l(s') = \mathcal{C}'$. Se existir este estado s', uma transição de s para s' é criada com o rótulo α. Caso contrário, um novo estado s' é criado com a rotulação obtida pela função $f_d(\mathcal{C}')$, como descrito na Figura 18, e uma transição de s para este novo s' é criada com o rótulo α. Em seguida, o Algoritmo 1 é invocado para o estado s' recursivamente.

3. O autômato está completo ao término das chamadas recursivas.

Algoritmo 1: Construção do autômato do contrato segundo Fenech

Algoritmo: $f_c(s)$
Entrada: Um estado com a cláusula $\mathcal{C}$ do contrato.
Saída: Estados do automato $\mathcal{A}(\mathcal{C})$.
se $l(s) = 1$ **então**
  $T \leftarrow T \cup (s, 1, s)$;
senão se $l(s) = 0$ **então**
  $V \leftarrow s$;
  $T \leftarrow T \cup (V, 1, V)$;
senão
  para cada $\alpha \in \mathcal{A}_{\mathcal{B}}^{\times}$ **faça**
    se $\exists s' \in S \mid l(s') = f(l(s), a)$ **então**
      $T \leftarrow T \cup (s, \alpha, s')$;
    senão
      novo s';
      $l(s') = f(l(s), a)$;
      $S \leftarrow S \cup s'$;
      $T \leftarrow T \cup (s, \alpha, s')$;
      $\delta(s') \leftarrow f_d(l(s'))$;
      $f_c(s')$;
    fim
  fim
fim

A criação dos estados do autômato é realizada com base na função de decomposição $f : \mathcal{CL} \times \mathcal{A}_{\mathcal{B}}^{\times} \rightarrow \mathcal{CL}$, definida na Figura 60 do Anexo A. Seja φ uma fórmula em $\mathcal{CL}$ e uma ação $\alpha \in \mathcal{A}_{\mathcal{B}}^{\times}$, a função f retorna a cláusula a ser verificada no próximo passo de execução da fórmula. Para ilustrar, assuma a fórmula $[b]O(b)$ e a ação b. Ao aplicar a função $f([b]O(b), b)$ seu retorno é $O(b)$.

A função f utiliza o operador binário /, descrito na Figura 61 do Anexo A, para tratar uma sequência de ações concorrentes de acordo com a leitura do *trace*. A partir de uma sequência de ações compostas α e uma ação φ, essa função retorna o resultado da execução de φ na sequência. A fórmula $(a \cdot b)/a$, por exemplo, retorna b, enquanto $((a \cdot b) + (a \cdot c))/a$ resulta em $b + c$.

$$f_d(C_1 \wedge C_2) = \{f_d(C_1)\} \cup \{f_d(C_2)\} \tag{3.5}$$
$$f_d(O(\alpha_{\times})) = \{O_{a_{\times}}\} \tag{3.6}$$
$$f_d(F(\alpha_{\times})) = \{F_{a_{\times}}\} \tag{3.7}$$
$$f_d(P(\alpha_{\times})) = \{P_{a_{\times}}\} \tag{3.8}$$
$$f_d(O(\alpha + \alpha')) = f_d(O(\alpha)) \cup f_d(O(\alpha')) \tag{3.9}$$
$$\text{em outros casos} = \emptyset \tag{3.10}$$

Figura 18 – Definição da função de rotulação deôntica f_d.

Além da análise de conflitos, o autômato ainda pode ser utilizado no monitoramento, na simulação, na análise de alcançabilidade das cláusulas conflitantes ou na verificação de cláusulas supérfluas [5].

3.2 Lógica Deôntica Relativizada

A relativização da lógica deôntica [7] é a principal alteração da SDL para que os indivíduos envolvidos em uma modalidade deôntica possam ser identificados. Essa modificação permite a especificação de cláusulas e a detecção de conflitos em contratos dessa natureza. Porém, não se conhece trabalhos que abordem a detecção de conflitos em contratos multilaterais usando a Lógica Deôntica Relativizada.

O objetivo da lógica deôntica é expressar os conceitos éticos e normativos das leis sobre os indivíduos. Porém, na lógica deôntica clássica, as expressões são escritas sem revelar a identidade dos indivíduos envolvidos. Na lógica deôntica padrão (SDL), a expressão $O(\alpha)$ significa que é obrigatório executar a ação α, onde a obrigação é imposta a algum indivíduo ou é genérica, direcionada a todos os possíveis indivíduos. Esta falta de personificação das ações torna a aplicação da SDL limitada tanto na análise do mundo real, em situações morais ou legais, quanto na concepção de sistemas multi-agente [59].

Neste cenário é comum existir sentenças sem a menção do indivíduo, como por exemplo em "é obrigatório pagar seus impostos em dia". Sentenças que expressam situações, tais como "é obrigatório que João pague seus impostos em dia", também são desejáveis. Um tipo de relativização de operadores deônticos foi proposto por Herrestad e Krogh [7] para explicitar o indivíduo responsável pela execução de uma determinada ação.

A relativização da lógica deôntica especifica quais indivíduos estão relacionados a um certo operador deôntico. Os operadores deônticos são considerados relativizados, quando apontam o indivíduo responsável por executar uma ação, ou não relativizados, quando não definem esse indivíduo [59]. Esta extensão da lógica deôntica preocupa-se com as conexões entre suas modalidades (obrigações, permissões ou proibições) que são vinculadas a indivíduos específicos e as modalidades impessoais [50]. Vários tipos de modalidades vinculando indivíduos específicos, podem ser obtidos conforme segue:

- **Modalidade geral (ou genérica):** modalidade relativa a todos os indivíduos;
- **Modalidade impessoal (ou não especificada):** quando a modalidade não menciona o indivíduo vinculado;
- **Modalidades pessoais (ou relativizadas):** modalidade relacionada a um único indivíduo; e
- **Modalidade direcionada:** quando a modalidade especifica o indivíduo que deve executar a ação e o indivíduo beneficiado (ou requerente) desta ação.

A seguir serão descritos os operadores relativizados e direcionados, uma forma de reparar uma modalidade direcionada e a semântica dos operadores relativizados.

3.2.1 Operadores deônticos relativizados e direcionados

A relativização da lógica deôntica é obtida ao adicionar operadores capazes de definir o portador da modalidade relacionada à execução de certa ação. O conjunto de todos os indivíduos é denotado por I e cada indivíduo portador da obrigação ou permissão é referido pelo nome ou por um índice $i, j, k, ... \in I$. Seja o indivíduo i e uma ação α, a sentença ${}_iO(\alpha)$ expressa que o indivíduo i é obrigado a fazer α, e ${}_iP(\alpha)$ significa que o indivíduo i tem permissão para executar a ação α [7].

Portanto, para nomear as obrigações e permissões definidas para um certo indivíduo, pode-se utilizar o termo "Obrigação relativizada ao portador" e "Permissão relativizada ao portador" respectivamente. Esta relativização possibilita expressar situações mais detalhadas com a lógica deôntica, como por exemplo, num comércio eletrônico, onde

o vendedor i é obrigado a *entregar* um produto e o cliente j é obrigado a *pagar* por esse produto. Essa sentença pode ser descrita por ${}_iO(entregar) \wedge {}_jO(pagar)$.

Uma característica típica das obrigações presentes em contratos é sua execução por um indivíduo, o portador da obrigação, para outro indivíduo, chamado de contra-parte ou requerente [60]. O termo obrigação direcionada está relacionado ao portador da obrigação que executa a ação e ao portador do direito de receber essa ação [61]. Para entender essa necessidade, suponha o vendedor i e o cliente j, num contrato em que i deve entregar o produto para j. Dessa cláusula podemos extrair duas sentenças, uma em que i deve entregar o produto para j e outra que j deve receber o produto de i. Nesta construção fica clara que as sentenças convergem para a mesma situação: ${}_iO(entregar) \wedge {}_jO(receber)$. Além disso, existe o direcionamento da ação *entregar* onde a obrigação de executá-la pertence ao cliente e o direito de recebê-la é do vendedor.

Uma obrigação direcionada é entendida como a união de duas obrigações relativizadas, em que a primeira ${}_iO(\alpha)$ indica que o indivíduo i deve executar a ação α e a segunda $O_j(\alpha)$ expressa que o indivíduo j deve receber a execução da ação α. A sentença ${}_iO_j(\alpha)$ representa uma obrigação direcionada [7].

Dessa forma, a sentença ${}_iO_j(\alpha) \iff {}_iO(\alpha) \wedge O_j(\alpha)$ indica que o indivíduo i é obrigado a executar a ação α em favor ao indivíduo j, se e somente se, i é obrigado a executar a ação α e j é obrigado a ser beneficiado pela ação α. Do ponto de vista legal, o indivíduo j é considerado no operador O_j como o requerente da ação.

3.2.2 Reparação ou penalidade de uma modalidade direcionada

Nas obrigações direcionadas, quando ocorre uma violação, não existe ação a ser tomada a não ser a invalidação do contrato. Tan e Thoen [60] aplicam as obrigações direcionadas a problemas decorrentes do comércio eletrônico e propõem uma melhoria neste conceito, adicionando uma penalidade em caso de violação pelo indivíduo portador da obrigação. A abordagem de Tan e Thoen prevê penalidades ao indivíduo que violou o contrato e dá poderes ao requerente de executar alguma ação legal, que pode ser outra ação de compensação ou penalidade.

Para que isso seja possível, é necessária a substituição da expressão $O_j(\alpha)$, presente na sentença direcionada, por uma noção diferente de contra-partida. O primeiro passo é adicionar a implicação $\Rightarrow_S$, em que determinada proposição α implica em outra proposição β sob o sistema normativo S, ou seja, $\alpha \Rightarrow_S \beta$. Mais precisamente, um sistema normativo S, representa as possibilidades de um sistema, definindo o que é legal e ilegal. Sistemas normativos diferentes podem diferir no comportamento, relacionamento e no resultado da execução de ações. Dessa forma, S é um sistema de normas de um contrato que relaciona as proposições informadas [62].

Em conjunto com o conectivo condicional $\Rightarrow_S$, é preciso definir uma ação de penalidade que represente a possibilidade de iniciar uma ação contra o portador da obrigação i. Diante dessa necessidade é introduzida a proposição la_i, interpretada como uma ação legal contra i. Formalmente, a definição de obrigação direcionada é representada, considerando um sistema normativo S, os indivíduos i e j e a ação α, como:

$$ {}_iO^S_j(\alpha) \equiv {}_iO(\alpha) \wedge (\neg\, \alpha \Rightarrow_S {}_jP(la_i)) \tag{3.11} $$

Esta definição diz que o indivíduo i é obrigado, sob o sistema normativo S, a executar a ação α para o indivíduo j, se e somente se, o indivíduo i executa a ação α, mas caso não execute esta ação, o sistema normativo S permite que o indivíduo j inicie uma ação legal la contra i. Esta permissão para executar uma ação legal tem a intenção de dar poderes ao requerente, mas não garante que a ação legal seja executada. A substituição da expressão $O_j(\alpha)$ por $(\neg\, \alpha \Rightarrow_S {}_jP(la_i))$ explicita o papel do requerente j. Dessa forma, é possível obter uma definição operacional do papel de requerente da ação, em que este tem o poder de penalizar o portador da obrigação por não cumprir a norma.

3.2.3 Semântica dos operadores relativizados

A formalização dos operadores relativizados e direcionados pode ser dada pela estrutura baseada no modelo de Kripke $\mathcal{M} = \langle \mathcal{W}, I, \mathcal{R}, P \rangle$, onde:

- $\mathcal{W}$ representa o conjunto de mundos $\mathcal{W} = \{w, w', \dots\}$;
- I denota o conjunto de indivíduos;
- $\mathcal{R}$ é o conjunto de funções $\{\mathcal{R}_1, \mathcal{R}_2, \dots, \mathcal{R}_n\}$ para cada indivíduo $i_1, i_2, \dots, i_n \in I$;
- $\mathcal{R}_i$ representa uma função que retorna os mundos alcançáveis a partir do mundo atual w (mundos deonticamente perfeitos) dado por $\mathcal{R}_i : \mathcal{W} \to 2^{\mathcal{W}}$;
- P é a função de valoração padrão que associa valores booleanos as sentenças presentes nos mundos de W;
- $[\![\alpha]\!]$ representa o conjunto de todos os mundos onde α é verdadeira.

A condição de satisfação dos operadores relativizados, considerando um modelo $\mathcal{M}$ a partir do mundo w, é definida na Figura 19.

$$ \mathcal{M}, w \models {}_iO(\alpha) \iff \mathcal{R}_i(w) \subseteq [\![\alpha]\!] \tag{3.12} $$
$$ \mathcal{M}, w \models {}_iP(\alpha) \iff \mathcal{R}_i(w) \cap [\![\alpha]\!] \neq \emptyset \tag{3.13} $$

Figura 19 – Semântica da Lógica Deôntica Relativizada.

A semântica dos operadores ${}_iO$ e ${}_iP$ possui os mesmos axiomas do operador de obrigação (O) da SDL, desde que a obrigação impessoal $O(\alpha)$ faça referência apenas ao mesmo indivíduo. Já o axioma (D) da SDL para ${}_iO$, denotado por $\neg({}_iO(\alpha) \wedge {}_iO(\neg\alpha))$ e que se refere à ausência de conflitos, é garantido pela seguinte restrição:

$$\mathcal{R}_i(w) \neq \emptyset, \forall \mathcal{R}_i \in \mathcal{R} \wedge \forall w \in \mathcal{W}. \tag{3.14}$$

No entanto, Herrestad e Krogh julgam o axioma (D) controverso [7], pois assume que não há conflito entre deveres. Esta regra não representa situações do mundo real em que podem existir conflitos numa mesma norma ou entre uma norma e alguma lei, código moral ou promessa. Como o ideal é um sistema de axiomas e de regras coerentes, deve existir uma relação entre os indivíduos e seus deveres sem obrigações conflitantes. Uma obrigação conflitante pode ser caracterizada por ${}_iO(\alpha) \wedge {}_jO(\neg\alpha)$, onde um conflito de exclusão mútua é caracterizado, pois enquanto i é obrigado a realizar a ação α, o indivíduo j é obrigado a evitar que a ação α seja realizada [50].

A ausência de conflitos é garantida se todas as obrigações, para todos os indivíduos, podem ser realizadas em conjunto e se a permissão de outro indivíduo jamais esteja em conflito com estas obrigações. Considerando estas regras, não deve existir uma situação em que um indivíduo tem a obrigação de executar uma ação α e outro indivíduo seja obrigado a impedir a execução de α. Este cenário pode ser especificado por $\neg({}_iO(\alpha) \wedge \exists_{j \in I}\ {}_jO(\neg\alpha))$, definindo que um indivíduo i é obrigado a executar uma ação α, o que torna necessário que todos os outros indivíduos não sejam obrigados a não executar essa ação α. Assim, sempre que uma ação é obrigatória a algum indivíduo, a mesma também é permitida a todos os indivíduos, dado pela expressão ${}_iO(\alpha) \rightarrow \forall_{j \in I}\ {}_jP(\alpha)$. Diante da expressão descrita, é desejável um sistema deôntico completo e coerente onde todas as ações obrigatórias também sejam permitidas para todos os indivíduos.

Com a semântica dos operadores de obrigação e permissão relativizadas é possível formalizar as obrigações e permissões gerais, como em $\forall i \in I,\ {}_iO(\alpha)$ e $\forall i \in I,\ {}_iP(\alpha)$; e impessoais, em $\exists i \in I,\ {}_iO(\alpha)$ e $\exists i \in I,\ {}_iP(\alpha)$.

4 UM MÉTODO PARA DETECÇÃO DE CONFLITOS EM CONTRATOS MULTILATERAIS

A tarefa de se evitar conflitos normativos é extremamente importante e necessária na área de contratação eletrônica. Conforme apresentado no Capítulo 3, alguns trabalhos foram propostos para tratar a verificação de conflitos em contratos, porém não se tem conhecimento de trabalhos que lidem com a detecção de conflitos em contratos multilaterais [63, 25]. Dessa forma, este capítulo propõe um método para detecção automática de conflitos normativos em contratos multilaterais.

O método proposto consiste numa lógica de contratos baseada na $\mathcal{CL}$ com características da lógica deôntica relativizada para especificar contratos multilaterais; e em uma extensão do algoritmo de detecção de conflitos proposto por Fenech [14]. A detecção de conflitos em contratos bilaterais proposta por Fenech [14] consiste de duas etapas: a construção de um autômato que representa o contrato descrito em $\mathcal{CL}$; e o algoritmo de detecção de conflitos sobre o autômato. Este trabalho estende o mecanismo proposto por Fenech para suportar a análise de contratos multilaterais, onde a identificação dos participantes é essencial no processo de detecção de conflitos. A extensão da proposta ocorre tanto na representação dos contratos quanto no algoritmo de busca por conflitos sobre o autômato que representa a extensão do formalismo.

O processo de detecção de conflitos em contratos multilaterais necessita, primeiramente, que o formalismo adotado seja capaz de representar os indivíduos envolvidos e suas relações. Os conflitos em contratos dessa natureza devem ser interpretados de maneira distinta da técnica original proposta por Fenech [14]. As regras de execução das ações num contrato muitas vezes recaem sobre indivíduos específicos, e não globalmente como na proposta clássica.

Com relação a representação de contratos multilaterais e detecção de conflitos dessa natureza, este trabalho propõe uma extensão da $\mathcal{CL}$ que compreende:

1. a extensão da sintaxe da $\mathcal{CL}$ para suportar o conceito de relativização;
2. a definição de uma semântica sobre as mudanças propostas aos operadores da $\mathcal{CL}$;
3. a adaptação do algoritmo de construção do autômato baseado na nova semântica;
4. a adaptação do algoritmo de detecção de conflitos.

Com o intuito de manter o método proposto autocontido neste Capítulo, algumas propriedades e construções sintáticas da $\mathcal{CL}$ que se aplicam na extensão serão apresentados novamente quando necessárias.

4.1 Extensão da sintaxe da $\mathcal{CL}$

No trabalho proposto por Herrestad e Krogh [7], os operadores da lógica deôntica são relativizados para englobar a identificação dos participantes envolvidos num contrato. Além do operador global, como definido na lógica deôntica convencional, é possível que um indivíduo específico, relacionado a um operador, seja identificado quando uma ação é executada, ou ainda que ambos, o executor e o recebedor de uma ação, sejam identificados. Já a $\mathcal{CL}$ clássica é baseada na lógica deôntica padrão e na lógica dinâmica, juntamente com a noção de penalidade nas obrigações e proibições.

Com base no trabalho de Herrestad e Krogh [7], a relativização é estendida para os operadores deônticos e dinâmicos da $\mathcal{CL}$. Seja $\mathcal{C}$ um contrato derivado pela gramática da Figura 20, onde $\mathcal{I}$ é o conjunto de indivíduos do contrato, $\mathcal{D} = \{O, P, F\}$ é o conjunto de operadores deônticos, $\mathcal{R} = \{g, i, i_{\frown}j \mid i, j \in \mathcal{I}\}$ define os tipos de relativizações possíveis, e α uma ação relacionada ao operador d, com $d \in \mathcal{D}$. A fórmula ${}_{g}d(\alpha)$ indica que a ação α deve ser executada por todos os indivíduos de $\mathcal{I}$ sob o operador deôntico d. Já a fórmula ${}_{i}d(\alpha)$ indica que o indivíduo $i \in \mathcal{I}$ deve executar a ação α relacionada ao operador d. Por fim, ${}_{i_{\frown}j}d(\alpha)$ indica que o indivíduo i deve executar a ação α associada para o indivíduo j conforme d. Com o propósito de simplificar a notação, o símbolo g que representa um operador global é suprimido na sintaxe. Já para os operadores dinâmicos, ${}_{g}[\alpha]\mathcal{C}$ indica que após a execução da ação α por todos os indivíduos, o contrato $\mathcal{C}$ entra em vigor. O operador dinâmico relativizado ${}_{i}[\alpha]\mathcal{C}$ indica que $\mathcal{C}$ é válido se o indivíduo i executar α. Por último, ${}_{i_{\frown}j}[\alpha]\mathcal{C}$ indica que $\mathcal{C}$ entra em vigor se i executar α para j.

A sintaxe relativizada da $\mathcal{CL}$, chamada de $\mathcal{RCL}$, é descrita conforme a gramática da Figura 20. Um contrato $\mathcal{C}$ pode então ser derivado por obrigações, $\mathcal{C}_O$, permissões, $\mathcal{C}_P$, proibições, $\mathcal{C}_F$, e operadores dinâmicos, $\mathcal{C}_D$. Os operados lógicos de conjunção, $\wedge$, de disjunção exclusiva, $\oplus$, e de disjunção, $\vee$, são aplicados conforme definição convencional. Além disso, os operadores deônticos de obrigação e proibição são acompanhados do mecanismo de penalidade em caso de violação. Na fórmula $iO_{\mathcal{C}'}(\alpha)$, a penalidade $\mathcal{C}'$ passa a vigorar caso o indivíduo i não execute a ação α. Note que $\mathcal{C}'$ é uma cláusula em $\mathcal{RCL}$.

Na gramática da $\mathcal{RCL}$, as ações associadas a operadores deônticos são representadas por α e as ações de operadores dinâmicos por β. Estas ações podem ser compostas por vário operadores. O operador $+$ representa a escolha entre duas ações, $\times$ representa a execução concorrente das ações, $\cdot$ indica a ordem de execução entre as ações, além das ações especiais $\mathbf{0}$ e $\mathbf{1}$ que representam, respectivamente, violação do contrato e a execução de qualquer ação. Um operador dinâmico também possibilita a execução iterativa, indicada pelo operador unário $*$, de uma determinada ação, bem como a negação de uma ação, operador $\overline{\alpha}$, que permite a execução de qualquer ação com exceção de α.

Na linguagem $\mathcal{CL}$, a disjunção exclusiva $\oplus$ é inserida para evitar alguns paradoxos

$$\mathcal{C} ::= \mathcal{C}_O \mid \mathcal{C}_P \mid \mathcal{C}_F \mid \mathcal{C} \wedge \mathcal{C} \mid \mathcal{C}_D \mid \top \mid \bot \tag{4.1}$$
$$\mathcal{C}_O ::= O_{\mathcal{C}}(\alpha) \mid {}_iO_{\mathcal{C}}(\alpha) \mid {}_{i\frown j}O_{\mathcal{C}}(\alpha) \mid \mathcal{C}_O \oplus \mathcal{C}_O \tag{4.2}$$
$$\mathcal{C}_P ::= P(\alpha) \mid {}_iP(\alpha) \mid {}_{i\frown j}P(\alpha) \mid \mathcal{C}_P \oplus \mathcal{C}_P \tag{4.3}$$
$$\mathcal{C}_F ::= F_{\mathcal{C}}(\alpha) \mid {}_iF_{\mathcal{C}}(\alpha) \mid {}_{i\frown j}F_{\mathcal{C}}(\alpha) \mid \mathcal{C}_F \vee \mathcal{C}_D\mathcal{C}_F \tag{4.4}$$
$$\mathcal{C}_D ::= [\beta]\mathcal{C} \mid {}_i[\beta]\mathcal{C} \mid {}_{i\frown j}[\beta]\mathcal{C} \tag{4.5}$$
$$\alpha ::= 0 \mid 1 \mid a \mid \alpha \times \alpha \mid \alpha \cdot \alpha \mid \alpha + \alpha \tag{4.6}$$
$$\beta ::= 0 \mid 1 \mid a \mid \beta \times \beta \mid \beta \cdot \beta \mid \beta + \beta \mid \overline{\beta} \mid \beta^* \tag{4.7}$$

Figura 20 – Gramática da $\mathcal{RCL}$.

deônticos [53] e possui o mesmo significado proveniente da lógica proposicional. Contudo, é importante analisar o comportamento deste operador em cada caso. Além do operador $\oplus$ ser irrelevante numa permissão, uma vez que sua execução é opcional, também surgem alguns problemas paradoxais com sua aplicação. Na fórmula ${}_{i\frown j}O_{{}_{i\frown j}F(\alpha)}(\mathbf{0}) \oplus {}_{i\frown j}O_{{}_{i\frown j}P(\beta)}(\mathbf{0})$, onde a ação $\mathbf{0}$ representa uma violação, as penalidades de ambas as obrigações passam a vigorar ainda sob a influência da disjunção exclusiva, resultando na fórmula ${}_{i\frown j}F(\alpha) \oplus {}_{i\frown j}P(\beta)$, que além de indesejada, não pode ser derivada pela gramática proposta da $\mathcal{RCL}$ (Vide Figura 20). Fenech [14] refere-se ao operador $\oplus$ como um problema em aberto, que permite a especificação de cláusulas sem significado claro. Em alguns casos, como nas obrigações, é possível dispensar o uso do operador $\oplus$ da fórmula ${}_{i\frown j}O(\alpha) \oplus_{i\frown j} O(\beta)$ através da composição ${}_{i\frown j}O(\alpha + \beta) \wedge_{i\frown j} F(\alpha \times \beta)$. Mesmo com os possíveis problemas provenientes do uso da disjunção exclusiva, optou-se por manter esse operador para garantir a expressividade e compatibilidade da $\mathcal{RCL}$ em relação à $\mathcal{CL}$ clássica, permitindo que uma fórmula em $\mathcal{CL}$ também possa ser verificada pelo método proposto.

As propriedades deônticas observadas na $\mathcal{CL}$, conforme descritas na Seção 2.4, se propagam para os operadores relativizados, uma vez que um operador global é definido sobre todos os indivíduos do contrato. Uma obrigação global pode ser expressa pela conjunção da obrigação relativizada da mesma ação para todos os indivíduos do contrato, representada por $O(\alpha) \iff \bigwedge_{\forall i \in \mathcal{I}} {}_iO(\alpha)$. Por isso, as derivações da Figura 21 são propagadas para todos os operadores da $\mathcal{RCL}$.

As decomposições e equivalências entre as fórmulas em $\mathcal{CL}$ [64], e consequentemente na extensão proposta, têm um papel importante na definição da semântica e no algoritmo de detecção de conflitos. A Figura 22 apresenta as decomposições da $\mathcal{RCL}$. Estas equivalências são baseadas na relação entre as lógicas dinâmica e deôntica proposta por Meyer [47]. Tanto as decomposições quanto as equivalências apresentadas seguem as derivações baseadas na abrangência dos operadores globais e direcionados da mesma maneira que as propriedades relativizadas da Figura 21.

$$
\begin{aligned}
&{}_iO_{\mathcal{C}}(\alpha) \implies {}_iP(\alpha) && (4.8)\\
&{}_iP(\alpha) \implies \neg\, {}_iF_{\mathcal{C}}(\alpha) && (4.9)\\
&{}_iO_{\mathcal{C}}(\alpha) \implies \neg\, {}_iF_{\mathcal{C}}(\alpha) && (4.10)\\
&{}_iF(\alpha) \implies {}_iF(\alpha \times \beta),\ \forall \beta \in \mathcal{A}_{\mathcal{B}} && (4.11)
\end{aligned}
$$

Figura 21 – Propriedades da $\mathcal{RCL}$.

$$
\begin{aligned}
&{}_iO_{\mathcal{C}}(\alpha) \iff {}_i[\overline{\alpha}]\mathcal{C} && (4.12)\\
&{}_iF_{\mathcal{C}}(\alpha) \iff {}_i[\alpha]\mathcal{C} && (4.13)\\
&{}_iP(\alpha) \iff {}_i[\alpha]\top && (4.14)\\
&{}_iO_{\mathcal{C}}(\alpha \times \beta) \iff {}_iO_{\mathcal{C}}(\alpha) \wedge\ {}_iO_{\mathcal{C}}(\beta) && (4.15)\\
&{}_iO_{\mathcal{C}}(\alpha \cdot \beta) \iff {}_iO_{\mathcal{C}}(\alpha) \wedge\ {}_i[\alpha]\ {}_iO_{\mathcal{C}}(\beta) && (4.16)\\
&{}_iO_{\mathcal{C}}(\alpha + \beta) \iff ({}_iO_{\mathcal{C}}(\alpha) \wedge\ {}_iO_{\mathcal{C}}(\beta)) \oplus\ {}_iO_{\mathcal{C}}(\alpha) \oplus\ {}_iO_{\mathcal{C}}(\beta) && (4.17)\\
&{}_iF_{\mathcal{C}}(\alpha \times \beta) \iff {}_iF_{\mathcal{C}}(\alpha) \wedge\ {}_iF_{\mathcal{C}}(\beta) && (4.18)\\
&{}_iF_{\mathcal{C}}(\alpha \cdot \beta) \iff {}_iF_{\mathcal{C}}(\alpha) \vee\ {}_i[\alpha]\ {}_iF_{\mathcal{C}}(\beta) && (4.19)\\
&{}_iF_{\mathcal{C}}(\alpha + \beta) \iff {}_iF_{\mathcal{C}}(\alpha) \wedge\ {}_iF_{\mathcal{C}}(\beta) && (4.20)\\
&{}_iP(\alpha \times \beta) \iff {}_iP(\alpha) \wedge\ {}_iP(\beta) && (4.21)\\
&{}_iP(\alpha \cdot \beta) \iff {}_iP(\alpha) \wedge\ {}_i[\alpha]\ {}_iP(\beta) && (4.22)\\
&{}_iP(\alpha + \beta) \iff {}_iP(\alpha) \wedge\ {}_iP(\beta) && (4.23)\\
&{}_i[\alpha \times \beta]\mathcal{C} \iff {}_i[\alpha]\mathcal{C} \wedge\ {}_i[\beta]\mathcal{C} && (4.24)\\
&{}_i[\alpha \cdot \beta]\mathcal{C} \iff {}_i[\alpha]\ {}_i[\beta]\mathcal{C} && (4.25)\\
&{}_i[\alpha + \beta]\mathcal{C} \iff {}_i[\alpha]\mathcal{C} \wedge\ {}_i[\beta]\mathcal{C} && (4.26)\\
&{}_i[\alpha^*]\mathcal{C} \iff \mathcal{C} \wedge\ {}_i[\alpha]\ {}_i[\alpha^*]\mathcal{C} && (4.27)
\end{aligned}
$$

Figura 22 – Decomposições da $\mathcal{RCL}$.

4.2 Semântica relativizada da $\mathcal{CL}$

A linguagem de contratos $\mathcal{CL}$ possui duas definições de semântica: a semântica completa, que trata das ramificações da computação num modelo normativo, e a semântica de *traces*, que estuda os caminhos de computação que satisfazem um contrato.

A semântica completa, proposta por Prisacariu e Schneider[8], define uma estrutura normativa baseada nos mundos de Kripke [37]. A relação de satisfação num contrato é definida de duas formas:

1. satisfação da relação $K, s \models \mathcal{C}$, onde o contrato $\mathcal{C}$ é satisfeito no modelo K a partir do estado s; e

2. a validade da relação $K \models \mathcal{C}$, onde o contrato $\mathcal{C}$ é válido em todos os estados do modelo K.

No entanto, a semântica completa não incorpora o tratamento de ações compostas, tais como $\alpha + \beta, \alpha \cdot \beta$ ou $\alpha \times \beta$, limitando a expressão de contratos mais complexos. Já a semântica de *traces* [56] contém uma sequência de ações, chamada *trace*, que satisfaz as fórmulas de um contrato. Esta semântica pode ser utilizada para o monitoramento de contratos [31] e para a detecção de conflitos em contratos [14], através da geração dos modelos que os representam.

A semântica da Lógica Deôntica Relativizada [7] é baseada no modelo de Kripke estendido, que possui informações sobre os indivíduos associados aos operadores deônticos. Esta semântica abrange o tratamento dos operadores deônticos relativizados, porém não possui um método de geração de um modelo a partir de uma fórmula. Assim sendo, a proposta deste trabalho é estender a semântica de *traces* baseada no trabalho de Fenech [14] usando a abordagem de Herrestad e Krogh [7] para tratar os novos operadores adicionados à sintaxe da $\mathcal{RCL}$.

A semântica de *traces* da $\mathcal{CL}$ proposta por Fenech [14] é definida pela sequência de ações, chamada de *trace*, executadas num contrato de forma que as cláusulas especificadas sejam satisfeitas. Junto ao *trace* de ações também são armazenadas informações sobre os operadores deônticos, chamado de *trace* deôntico. Nesta abordagem, o conteúdo do *trace* de ações é composto de ações relativizadas, que consiste na ação relacionada ao operador e os indivíduos que executam e recebem esta ação.

Uma ação relativizada é definida como uma tupla $a_r = \langle i, \alpha, j \rangle$, onde $i, j \in \mathcal{I}$ são os indivíduos que executam e recebem a ação, respectivamente, e α é uma ação básica do conjunto $\mathcal{A}_\mathcal{B}$ de todas as ações básicas do contrato. O conjunto das ações relativizadas é obtido pela combinação das ações e dos indivíduos envolvidos no contrato, dado por $\mathcal{A}_r = \{\mathcal{I} \times \mathcal{A}_\mathcal{B} \times \mathcal{I}\}$. Já o conjunto de ações relativizadas concorrentes é definido por $\mathcal{A}_r^2 = 2^{\mathcal{A}_r} - \emptyset$, onde cada ação é formada pela combinação das ações relativizadas que ocorrem ao mesmo tempo.

Um *trace* de ações é, formalmente, definido por $\sigma : \mathbb{N} \to \mathcal{A}_r^2$, onde σ retorna as ações concorrentes da posição $i \in \mathbb{N}$ do *trace*. Seja $\sigma = \alpha_0, \alpha_1, \ldots$ um *trace*, com $\alpha_i \in \mathcal{A}_r^2$, $i \geq 0$, temos que $\sigma(i) = \alpha_i$. O comprimento de um *trace* é dado por $\mid \sigma \mid$ enquanto que o *trace* vazio é denotado por ε. Já um *subtrace* pode ser representado por $\sigma(i..j)$, onde i é a posição de início e j a posição de fim do *trace*, enquanto que um *trace* infinito $\sigma(i...)$ representa um *subtrace* iniciado em i. A operação de concatenação de dois *traces* σ' e σ'' segue como a convencional, denotada por $\sigma'\sigma''$.

Em relação a proposta de Fenech [14], o *trace* deôntico da $\mathcal{RCL}$ tem um aumento considerável nas representações devido à relativização. A representação dessas modalidades deônticas sobre as ações de um contrato é dada por $\mathcal{M} = \{ \ {}_rd_\alpha \mid r \in \mathcal{R}, d \in \mathcal{D}, \alpha \in \mathcal{A}_\mathcal{B}\}$. Assim, um *trace* deôntico é denotado por $\sigma_d : \mathbb{N} \to 2^\mathcal{M}$, onde cada posição de σ_d retorna um conjunto de combinações das representações de modalidades deônticas, com

$\sigma_d(i) \in 2^{\mathcal{M}} \mid i \geq 0$. A representação $2^{\mathcal{M}}$ no *trace* deôntico distingue as conjunções e disjunções entre modalidades deônticas. Considere os contratos $\mathcal{C} = {}_{i\curvearrowright j}O(a) \wedge {}_{i\curvearrowright j}O(b) \wedge {}_{i\curvearrowright j}F(b)$ e $\mathcal{C}' = {}_{i\curvearrowright j}O(a+b) \wedge {}_{i\curvearrowright j}F(b)$. Os *traces* deônticos obtidos por $\mathcal{C}$ e $\mathcal{C}'$ são, respectivamente, $\sigma_d = \langle\{{}_{i\curvearrowright j}O_a\}, \{{}_{i\curvearrowright j}O_b\}, \{{}_{i\curvearrowright j}F_b\}\rangle$, indicando um conflito entre os operadores, e $\sigma'_d = \langle\{{}_{i\curvearrowright j}O_a,\ {}_{i\curvearrowright j}O_b\}, \{{}_{i\curvearrowright j}F_b\}\rangle$, onde não há conflito, já que existe uma possibilidade de escolher entre as obrigações de a e b, sem violar a proibição de b. A união de dois *traces* deônticos é definida por $\sigma_d \cup \sigma'_d = \sigma_d(0) \cup \sigma'_d(0); \sigma_d(1) \cup \sigma'_d(1); \ldots; \sigma_d(n) \cup \sigma'_d(n)$. Esta operação de união é importante para representar a conjunção de cláusulas do contrato, como pode ser visto na Equação (4.30) da Figura 23.

Numa modalidade deôntica global, que impacta em todos os indivíduos, o *trace* de ações σ deve possuir uma ação relativizada concorrente que represente todos os indivíduos executando a referida ação desta modalidade deôntica. Uma fórmula global $O(\alpha)$ gera um *trace* de ações $\sigma(0) \subseteq \{\langle x, \alpha, y\rangle \mid y \in \mathcal{I}, \forall x \in \mathcal{I}\}$. No caso da modalidade relativizada ${}_iO(\alpha)$, é gerado o *trace* $\sigma(0) \subseteq \{\langle i, \alpha, x\rangle \mid x \in \mathcal{I}\}$. Por fim na modalidade direcionada ${}_{i\curvearrowright j}O(\alpha)$, a ação esperada no *trace* é $\sigma(0) \subseteq \{\langle i, \alpha, j\rangle \mid i, j \in \mathcal{I}\}$.

A semântica dos operadores proposta neste trabalho é, consequentemente baseada no trabalho de Fenech [14]. A relação de satisfação $\sigma, \sigma_d \models \mathcal{C}$ é definida para determinar se o *trace* de ações σ e o *trace* deôntico σ_d satisfazem o contrato $\mathcal{C}$. A semântica de *traces* para a $\mathcal{RCL}$ é apresentada na Figura 23, considerando que $\mathcal{C}$ é uma cláusula do contrato, $\alpha \in \mathcal{A}_r$ é uma ação relativizada e $i, j \in \mathcal{I}$ são indivíduos do contrato. A semântica das fórmulas envolvendo ações compostas de escolha, concorrência e sequência são obtidas usando as decomposições apresentadas na Seção 4.1.

As modificações na semântica não alteram o processamento dos *traces* de ações em relação à abordagem de Fenech [14]. A inclusão de informação extra sobre os indivíduos envolvidos na ação relativizada mantém a relação de satisfação proposta por Fenech. Na semântica original é verificado se uma ação de modalidade deôntica está contida numa determinada posição do *trace* de ações, $\alpha \subseteq \sigma(0)$. Na semântica da $\mathcal{RCL}$ é preciso verificar também o executor e o recebedor da ação, respectivamente, para que o *trace* de ações e o *trace* deôntico satisfaçam a modalidade direcionada. Na fórmula $\sigma, \sigma_d \models_{i\curvearrowright j} O(\alpha)$, é preciso que exista pelo menos uma ação relativizada em $\sigma(0)$ em que o indivíduo i executa a ação α para o indivíduo j, formalmente dado por $\exists\varphi \in \sigma(0)$ tal que $\varphi = \langle i, \alpha, j\rangle$. Por outro lado, no caso de uma modalidade dinâmica direcionada, $\sigma, \sigma_d \models_{i\curvearrowright j} [\alpha]\mathcal{C}$, o *trace* deôntico é vazio e a verificação do contrato ocorre apenas sobre o *trace* de ações.

As demais modalidades da $\mathcal{RCL}$ seguem a mesma intuição proposta para a $\mathcal{CL}$ na definição da semântica. Por isso, suas representações foram omitidas, mas podem ser obtidas através das decomposições apresentadas na Figura 22 da Seção 4.1.

$\sigma, \sigma_d \not\models \mathcal{C}$ se $\mid \sigma \mid \neq \mid \sigma_d \mid$ (4.28)
$\sigma, \sigma_d \models \mathcal{C}$ se $\mid \sigma \mid = 0$ e $\mid \sigma_d \mid = 0$ (4.29)
$\sigma, \sigma_d \models \mathcal{C}_1 \wedge \mathcal{C}_2$ se $\sigma, \sigma'_d \models \mathcal{C}_1$ e $\sigma, \sigma''_d \models \mathcal{C}_2$ e $\sigma_d = \sigma'_d \cup \sigma''_d$ (4.30)
$\sigma, \sigma_d \models O_{\mathcal{C}}(\alpha)$ se $O_\alpha \in \sigma_d(0)$ e $((\forall i \in \mathcal{I}, \exists \varphi \in \sigma(0), x \in \mathcal{I} \mid \varphi = \langle i, \alpha, x \rangle$ e (4.31)
$\sigma(1\ldots), \sigma_d(1\ldots) \models \top)$ ou $(\sigma(1\ldots), \sigma_d(1\ldots) \models \mathcal{C}))$ (4.32)
$\sigma, \sigma_d \models {}_iO_{\mathcal{C}}(\alpha)$ se ${}_iO_\alpha \in \sigma_d(0)$ e $((\exists \varphi \in \sigma(0), x \in \mathcal{I} \mid \varphi = \langle i, \alpha, x \rangle$ e (4.33)
$\sigma(1\ldots), \sigma_d(1\ldots) \models \top)$ or $(\sigma(1\ldots), \sigma_d(1\ldots) \models \mathcal{C}))$ (4.34)
$\sigma, \sigma_d \models {}_{i\frown j}O_{\mathcal{C}}(\alpha)$ se ${}_{i\frown j}O_\alpha \in \sigma_d(0)$ e $((\exists \varphi \in \sigma(0) \mid \varphi = \langle i, \alpha, j \rangle$ e (4.35)
$\sigma(1\ldots), \sigma_d(1\ldots) \models \top)$ ou $(\sigma(1\ldots), \sigma_d(1\ldots) \models \mathcal{C}))$ (4.36)
$\sigma, \sigma_d \models F_{\mathcal{C}}(\alpha)$ se $F_\alpha \in \sigma_d(0)$ e $((\exists i \in \mathcal{I}, \exists \varphi \in \sigma(0), x \in \mathcal{I} \mid \varphi = \langle i, \alpha, x \rangle$ e (4.37)
$\sigma(1\ldots), \sigma_d(1\ldots) \models \mathcal{C})$ ou $(\sigma(1\ldots), \sigma_d(1\ldots) \models \top))$ (4.38)
$\sigma, \sigma_d \models {}_iF_{\mathcal{C}}(\alpha)$ se ${}_iF_\alpha \in \sigma_d(0)$ e $((\exists \varphi \in \sigma(0), x \in \mathcal{I} \mid \varphi = \langle i, \alpha, x \rangle$ e (4.39)
$\sigma(1\ldots), \sigma_d(1\ldots) \models \mathcal{C})$ or $(\sigma(1\ldots), \sigma_d(1\ldots) \models \top))$ (4.40)
$\sigma, \sigma_d \models {}_{i\frown j}F_{\mathcal{C}}(\alpha)$ se ${}_{i\frown j}F_\alpha \in \sigma_d(0)$ e $((\exists \varphi \in \sigma(0) \mid \varphi = \langle i, \alpha, j \rangle$ e (4.41)
$\sigma(1\ldots), \sigma_d(1\ldots) \models \mathcal{C})$ ou $(\sigma(1\ldots), \sigma_d(1\ldots) \models \top))$ (4.42)
$\sigma, \sigma_d \models P_{\mathcal{C}}(\alpha)$ se $P_\alpha \in \sigma_d(0)$ e $\sigma(1\ldots), \sigma_d(1\ldots) \models \top$ (4.43)
$\sigma, \sigma_d \models {}_iP_{\mathcal{C}}(\alpha)$ se ${}_iP_\alpha \in \sigma_d(0)$ e $\sigma(1\ldots), \sigma_d(1\ldots) \models \top$ (4.44)
$\sigma, \sigma_d \models {}_{i\frown j}P_{\mathcal{C}}(\alpha)$ se ${}_{i\frown j}P_\alpha \in \sigma_d(0)$ e $\sigma(1\ldots), \sigma_d(1\ldots) \models \top$ (4.45)
$\sigma, \sigma_d \models [\alpha]\mathcal{C}$ se $(\forall i \in \mathcal{I}, \exists \varphi \in \sigma(0), x \in \mathcal{I} \mid \varphi = \langle i, \alpha, x \rangle$ e $\sigma(1\ldots), \sigma_d(1\ldots) \models \mathcal{C})$ (4.46)
ou $(\forall i \in \mathcal{I}, x \in \mathcal{I}, \nexists \varphi \in \sigma(0) \mid \varphi = \langle i, \alpha, x \rangle)$ (4.47)
$\sigma, \sigma_d \models {}_i[\alpha]\mathcal{C}$ se $(\exists \varphi \in \sigma(0), x \in \mathcal{I} \mid \varphi = \langle i, \alpha, x \rangle$ e $\sigma(1\ldots), \sigma_d(1\ldots) \models \mathcal{C})$ ou (4.48)
$(x \in \mathcal{I}, \nexists \varphi \in \sigma(0) \mid \varphi = \langle i, \alpha, x \rangle)$ (4.49)
$\sigma, \sigma_d \models {}_{i\frown j}[\alpha]\mathcal{C}$ se $(\exists \varphi \in \sigma(0) \mid \varphi = \langle i, \alpha, j \rangle$ e $\sigma(1\ldots), \sigma_d(1\ldots) \models \mathcal{C})$ ou (4.50)
$(\nexists \varphi \in \sigma(0) \mid \varphi = \langle i, \alpha, j \rangle)$ (4.51)

Figura 23 – Semântica da $\mathcal{RCL}$.

4.3 Algoritmo de construção do autômato

O processo de detecção de conflitos em contratos multilaterais baseado na $\mathcal{RCL}$ consiste na construção do autômato que representa os *traces* que satisfazem o contrato. O algoritmo proposto é uma adaptação do algoritmo proposto por Fenech [14]. O algoritmo leva em conta os novos operadores deônticos para que os critérios de busca detectem os conflitos relacionados a identificação partindo dos indivíduos.

Na abordagem de Fenech [14] o autômato que representa o contrato é completamente construído de forma que todos os *traces* do contrato sejam satisfeitos. Uma adaptação para o algoritmo da $\mathcal{RCL}$ é a definição do critério de parada quando um conflito é encontrado. A estrategia pode tornar o algoritmo menos custoso na prática, pois o processo de construção do autômato, que ocorre pelas decomposições do contrato, pode ser interrompido (Veja Subseção 4.4).

A definição do autômato $\mathcal{A}$ que modela o contrato $\mathcal{C}$ é definido por

$$\mathcal{A}(\mathcal{C}) = \langle S, \mathcal{A}_r^2, \mathcal{M}, \mathcal{I}, s_0, T, V, l, \delta \rangle \tag{4.52}$$

onde:

- S é o conjunto de estados do autômato;
- $\mathcal{A}_r^2$ é o conjunto de ações relativizadas concorrentes;
- $\mathcal{M}$ é o conjunto de rótulos deônticos;
- $\mathcal{I}$ é o conjunto de indivíduos;
- s_0 é o estado inicial;
- $T \subseteq S \times \mathcal{A}_r^2 \times S$ é a relação de transições rotuladas;
- V é o estado de violação do contrato;
- $l : S \to \mathcal{C}$ é a função de rotulação dos estados com as decomposições do contrato; e
- $\delta : S \to 2^{\mathcal{M}}$ é a função de rotulação dos estados com informações deônticas.

Uma sequência de ações relativizadas concorrentes do conjunto $\mathcal{A}_r^2$, que define um *trace* de ações σ, é uma palavra da linguagem aceita pelo autômato construído. O Algoritmo 2 descreve o processo de construção através da função $f : \mathcal{C} \times \mathcal{A}_r^2 \to \mathcal{C}$, que recebe um contrato e um conjunto de ações relativizadas, e retorna a decomposição resultante da execução dessas ações. Observe que neste algoritmo, o conjunto de ações concorrentes é substituído pelo conjunto de ações relativizadas concorrentes devido a relativização dos operadores e a modificação do *trace* de ações.

A Figura 24 mostra a aplicação da função f sobre os operadores deônticos e dinâmicos, ambos relativizados, onde $\varphi \in \mathcal{A}_r^2$ é uma ação relativizada. Note que a função f completa considera ainda os operadores deônticos de proibição e operadores dinâmicos sobre ações compostas conforme a sintaxe e as regras de decomposição definidas na Seção 4.1.

A função f, no trabalho de Fenech [14], lida apenas com operadores globais e ações básicas. A redefinição de f abrange esse comportamento conforme descrito na Equação (4.56) da Figura 24. Já a Equação (4.58) ilustra o tratamento da função f sobre uma obrigação direcionada, um comportamento que não pode ser capturado pela abordagem clássica. Neste caso a função f recebe os parâmetros ${}_{i\curvearrowright j}O_{\mathcal{C}}(\alpha)$ e φ, um contrato e um conjunto de ações relativizadas, respectivamente. Se a ação na forma $\langle i, \alpha, j \rangle$ existir no conjunto φ, a obrigação é satisfeita, e a função f retorna $\top$. Se a obrigação não é satisfeita, a penalidade $\mathcal{C}$ entra em vigor.

$$f(\top, \varphi) = \top \tag{4.53}$$
$$f(\bot, \varphi) = \bot \tag{4.54}$$
$$f(\mathcal{C}_1 \wedge \mathcal{C}_2, \varphi) = f(\mathcal{C}_1, \varphi) \wedge f(\mathcal{C}_2, \varphi) \tag{4.55}$$
$$f(O_{\mathcal{C}}(\alpha), \varphi) = \begin{cases} \top & \text{se } \forall i \in \mathcal{I}, \exists a_r \in \varphi \mid a_r = \langle i, \alpha, x \rangle, \text{ com } x \in \mathcal{I} \\ \mathcal{C} & \text{em qualquer outro caso.} \end{cases} \tag{4.56}$$
$$f({}_iO_{\mathcal{C}}(\alpha), \varphi) = \begin{cases} \top & \text{se } \exists a_r \in \varphi \mid a_r = \langle i, \alpha, x \rangle, \text{ com } i, x \in \mathcal{I} \\ \mathcal{C} & \text{em qualquer outro caso.} \end{cases} \tag{4.57}$$
$$f({}_{i\curvearrowright j}O_{\mathcal{C}}(\alpha), \varphi) = \begin{cases} \top & \text{se } \exists a_r \in \varphi \mid a_r = \langle i, \alpha, j \rangle, \text{ com } i, j \in \mathcal{I} \\ \mathcal{C} & \text{em qualquer outro caso.} \end{cases} \tag{4.58}$$
$$f(F_{\mathcal{C}}(\alpha), \varphi) = \begin{cases} \mathcal{C} & \text{se } \exists i \in \mathcal{I}, \exists a_r \in \varphi \mid a_r = \langle i, \alpha, x \rangle, \text{ com } x \in \mathcal{I} \\ \top & \text{em qualquer outro caso.} \end{cases} \tag{4.59}$$
$$f({}_iF_{\mathcal{C}}(\alpha), \varphi) = \begin{cases} \mathcal{C} & \text{se } \exists a_r \in \varphi \mid a_r = \langle i, \alpha, x \rangle, \text{ com } i, x \in \mathcal{I} \\ \top & \text{em qualquer outro caso.} \end{cases} \tag{4.60}$$
$$f({}_{i\curvearrowright j}F_{\mathcal{C}}(\alpha), \varphi) = \begin{cases} \mathcal{C} & \text{se } \exists a_r \in \varphi \mid a_r = \langle i, \alpha, j \rangle, \text{ com } i, j \in \mathcal{I} \\ \top & \text{em qualquer outro caso.} \end{cases} \tag{4.61}$$
$$f(P_{\mathcal{C}}(\alpha), \varphi) = \top \tag{4.62}$$
$$f({}_iP_{\mathcal{C}}(\alpha), \varphi) = \top \tag{4.63}$$
$$f({}_{i\curvearrowright j}P_{\mathcal{C}}(\alpha), \varphi) = \top \tag{4.64}$$
$$f([\alpha]\mathcal{C}, \varphi) = \begin{cases} \mathcal{C} & \text{se } \forall i \in \mathcal{I}, \exists a_r \in \varphi \mid a_r = \langle i, \alpha, x \rangle, \text{ com } x \in \mathcal{I} \\ \top & \text{em qualquer outro caso.} \end{cases} \tag{4.65}$$
$$f({}_i[\alpha]\mathcal{C}, \varphi) = \begin{cases} \mathcal{C} & \text{se } \exists a_r \in \varphi \mid a_r = \langle i, \alpha, x \rangle, \text{ com } i, x \in \mathcal{I} \\ \top & \text{em qualquer outro caso.} \end{cases} \tag{4.66}$$
$$f({}_{i\curvearrowright j}[\alpha]\mathcal{C}, \varphi) = \begin{cases} \mathcal{C} & \text{se } \exists a_r \in \varphi \mid a_r = \langle i, \alpha, j \rangle, \text{ com } i, j \in \mathcal{I} \\ \top & \text{em qualquer outro caso.} \end{cases} \tag{4.67}$$

Figura 24 – Função de decomposição f para a construção de $\mathcal{A}(\mathcal{C})$.

Algoritmo 2: *construirAutomato*(s)

```
input  : Um estado s ∈ S do autômato A(C)
output: O autômato A(C)
begin
    if buscarConflitos (s) then
        um conflito foi encontrado no estado s;
    else if l(s) = ⊤ then
        T ← T ∪ (s, ⊤, s);
    else if l(s) = ⊥ then
        V ← s;
        T ← T ∪ (V, ⊥, V);
    else
        for α ∈ A_r^2 do
            C' ← f(l(s), α);
            if ∃s' ∈ S | l(s') = C' then
                T ← T ∪ (s, α, s');
            else
                S ← S ∪ s';
                l(s') ← C';
                T ← T ∪ (s, α, s');
                δ(s') ← f_d(C');
                construirAutomato (s');
            end
        end
    end
    return A(C);
end
```

No Algoritmo 2 também é definida outra função auxiliar de rotulação deôntica dada por $f_d : \mathcal{C} \to 2^{\mathcal{M}}$. Cada estado do autômato é rotulado com as respectivas informações deônticas representadas no *trace* deôntico σ_d do contrato decomposto. A função de rotulação deôntica também é redefinida para capturar adequadamente o comportamento dos novos operadores relativizados e direcionados. A redefinição de f_d é apresentada na Figura 25. A função recebe um contrato com operadores deônticos e retorna um subconjunto de $2^{\mathcal{M}}$, indicando os operadores deônticos e suas respectivas ações presentes no contrato decomposto. Quando um contrato é formado apenas por operadores dinâmicos, sem a informação deôntica, a função retorna o conjunto vazio. Vale ressaltar que as decomposições já apresentadas podem ser aplicadas para a definição completa de f_d. A informação deôntica obtida com f_d é associada ao estado em que o contrato se decompôs através da função δ e utilizada pelo Algoritmo 3 na detecção por situações de conflito num contrato (Veja a Seção 4.4).

O Algoritmo 2 começa pelo estado inicial s_0 do autômato com o contrato completo constrói o autômato $\mathcal{A}(\mathcal{C})$ da seguinte forma:

1. A função recursiva *construirAutomato* constrói o autômato a partir do estado s;

2. A função *buscarConflitos* verifica se o estado s não possui um conflito. Se um conflito não é detectado em s, três casos podem ocorrer:

   a) Se $l(s) = \top$, então o contrato representado em s é satisfeito e não restam cláusulas a serem processadas;

   b) Caso contrário, se $l(s) = \bot$, então há uma violação no contrato e o estado atual s é marcado como V (violação);

   c) Caso contrário, para cada elemento do alfabeto de ações relativizadas $\mathcal{A}_r^2$, a função $f(l(s), \alpha)$ retorna uma decomposição do contrato $\mathcal{C}'$. Se existir um estado s' que represente o contrato $\mathcal{C}'$, ou seja, $l(s') = \mathcal{C}'$, então uma transição de s para s' é rotulada com α. Caso contrário, um novo estado s' é gerado, representando o contrato $\mathcal{C}'$, com a rotulação obtida pela função $f_d(\mathcal{C}')$ e uma transição de s para s' é rotulada com α. A função *construirAutomato*(s') é chamada recursivamente para o estado s'.

3. O algoritmo termina após a decomposição completa do contrato ou caso um conflito seja encontrado.

$$f_d(C_1 \wedge C_2) = \{f_d(C_1)\} \cup \{f_d(C_2)\} \tag{4.68}$$
$$f_d({}_{i\curvearrowright j}O(\alpha)) = \{{}_{i\curvearrowright j}O_\alpha\} \tag{4.69}$$
$$f_d({}_{i\curvearrowright j}O(\alpha \cdot \beta)) = f_d({}_{i\curvearrowright j}O(\alpha) \wedge {}_{i\curvearrowright j}[\alpha]\, {}_{i\curvearrowright j}O(\beta)) \tag{4.70}$$
$$f_d({}_{i\curvearrowright j}O(\alpha + \beta)) = f_d({}_{i\curvearrowright j}O(\alpha)) \cup f_d({}_{i\curvearrowright j}O(\beta)) \tag{4.71}$$
$$f_d({}_{i\curvearrowright j}O(\alpha \times \beta)) = \{f_d({}_{i\curvearrowright j}O(\alpha))\} \cup \{f_d({}_{i\curvearrowright j}O(\beta))\} \tag{4.72}$$
$$f_d({}_{i\curvearrowright j}F(\alpha)) = \{{}_{i\curvearrowright j}F_\alpha\} \tag{4.73}$$
$$f_d({}_{i\curvearrowright j}F(\alpha \cdot \beta)) = f_d({}_{i\curvearrowright j}F(\alpha) \wedge {}_{i\curvearrowright j}[\alpha]\, {}_{i\curvearrowright j}F(\beta)) \tag{4.74}$$
$$f_d({}_{i\curvearrowright j}F(\alpha + \beta)) = \{f_d({}_{i\curvearrowright j}F(\alpha))\} \cup \{f_d({}_{i\curvearrowright j}F(\beta))\} \tag{4.75}$$
$$f_d({}_{i\curvearrowright j}F(\alpha \times \beta)) = \{f_d({}_{i\curvearrowright j}F(\alpha))\} \cup \{f_d({}_{i\curvearrowright j}F(\beta))\} \tag{4.76}$$
$$f_d({}_{i\curvearrowright j}P(\alpha)) = \{{}_{i\curvearrowright j}P_\alpha\} \tag{4.77}$$
$$f_d({}_{i\curvearrowright j}P(\alpha \cdot \beta)) = f_d({}_{i\curvearrowright j}P(\alpha) \wedge {}_{i\curvearrowright j}[\alpha]\, {}_{i\curvearrowright j}P(\beta)) \tag{4.78}$$
$$f_d({}_{i\curvearrowright j}P(\alpha + \beta)) = f_d({}_{i\curvearrowright j}P(\alpha)) \cup f_d({}_{i\curvearrowright j}P(\beta)) \tag{4.79}$$
$$f_d({}_{i\curvearrowright j}P(\alpha \times \beta)) = \{f_d({}_{i\curvearrowright j}P(\alpha))\} \cup \{f_d({}_{i\curvearrowright j}P(\beta))\} \tag{4.80}$$
$$f_d({}_{i\curvearrowright j}[\alpha]\mathcal{C}) = \emptyset \tag{4.81}$$
$$f_d({}_{i\curvearrowright j}[\alpha \cdot \beta]\mathcal{C}) = \emptyset \tag{4.82}$$
$$f_d({}_{i\curvearrowright j}[\alpha + \beta]\mathcal{C}) = \emptyset \tag{4.83}$$
$$f_d({}_{i\curvearrowright j}[\alpha \times \beta]\mathcal{C}) = \emptyset \tag{4.84}$$

Figura 25 – Função f_d para rotulação deôntica.

4.4 Algoritmo de detecção de conflitos

Os conflitos em contratos, geralmente, são caracterizados pela impossibilidade da satisfação simultânea de suas cláusulas. Uma das formas de detecção de conflitos especificados em $\mathcal{RCL}$ é através do modelo de autômato gerado pelo Algoritmo 2 da Subseção 4.3. O método de detecção adapta o algoritmo da abordagem clássica, com o tratamento das novas interpretações de conflitos considerando a relativização dos operadores deônticos. Os possíveis cenários de conflito tratados são caracterizados pela ocorrência de:

1. uma ação entre operadores deônticos de obrigação e proibição;
2. uma ação entre operadores deônticos de proibição e permissão;
3. ações pré-definidas como conflitantes entre operadores de obrigação;
4. ações pré-definidas como conflitantes entre um operador de permissão e de obrigação.

Uma das alterações com relação ao algoritmo original de Fenech consiste numa avaliação entre operadores deônticos relativizados quando estes estão relacionados a uma mesma ação. A abordagem clássica caracteriza como conflito a ocorrência de uma mesma ação entre operadores deônticos. Entretanto, se os operadores são relativizados e associados a uma mesma ação executada por indivíduos distintos, o conflito não se concretiza. A fórmula ${}_iO(\alpha) \wedge {}_jF(\alpha)$, onde a ação α é obrigada e proibida ao mesmo tempo com indivíduos distintos associados aos operadores, ilustra este cenário onde a violação não ocorre e o contrato pode ser satisfeito. Similarmente, a execução de uma ação permitida e proibida ao mesmo tempo com indivíduos distintos não caracteriza um conflito, como ocorre na abordagem clássica.

Note que um operador global deôntico que representa uma obrigação sobre todos os indivíduos sempre está em conflito com os operadores relativizados de proibição. Neste caso, o conflito pode ser caracterizado quando uma obrigação global e uma proibição relativizada sobre uma mesma ação estão presentes numa conjunção. A fórmula $O(\alpha) \wedge {}_iF(\alpha)$ ilustra este cenário de conflito, também representada pela especificação equivalente $\forall x \in \mathcal{I},\ {}_xO_{\mathcal{C}}(\alpha) \wedge {}_iF(\alpha)$. Similarmente, um conflito ocorre no caso de uma permissão e uma proibição sobre uma mesma ação, quando ao menos uma das modalidades é global.

Nos cenários onde duas ações são pré-definidas como conflitantes, a detecção também ocorre de maneira distinta da original [14] na presença das relativizações. Na abordagem de Fenech o conflito entre ações é definido pela relação $\# \subseteq \mathcal{A}_{\mathcal{B}} \times \mathcal{A}_{\mathcal{B}}$, onde $\alpha \# \beta$ indica que as ações α e β não podem ocorrer concorrentemente. Com a relativização e consequente identificação dos indivíduos envolvidos na execução das ações, a relação de conflito agora é denotada por $\# \subseteq \mathcal{A}_r \times \mathcal{A}_r$, para que um conflito predefinido entre ações relativizadas possa ser representado.

No caso de ações predefinidas como conflitantes, um conflito só é efetivamente caracterizado quando as ações são executadas pelo mesmo indivíduo especificado na relativização. Para que duas ações jamais sejam executadas por um único indivíduo, a relação de conflito pode ser definida de duas formas: a relação de conflito global, quando as ações em conflito não podem ser executadas concorrentemente quaisquer que sejam os indivíduos que as executem; e a relação de conflito relativizada, quando as ações conflitantes não podem ser executadas pelo mesmo indivíduo.

A relação de conflito global é denotada por $\#_g \subseteq \mathcal{A}_\mathcal{B} \times \mathcal{A}_\mathcal{B}$ e mantém a mesma interpretação da relação original proposta para $\mathcal{CL}$. A relação de conflito global entre ações $\alpha \#_g \beta$, com $\alpha, \beta \in \mathcal{A}_\mathcal{B}$, indica que essas ações não podem ocorrer de forma concomitante independente do indivíduo executor. A forma equivalente da relação de conflito global é dada por $\forall i, j \in \mathcal{I}, \langle i, \alpha, x\rangle \# \langle j, \beta, y\rangle$ com $x, y \in \mathcal{I}$.

Já numa relação de conflito relativizada, quando as ações são executadas por indivíduos distintos, o conflito não se caracteriza. A relação de conflito relativizada é denotada por $\#_r \subseteq \mathcal{A}_\mathcal{B} \times \mathcal{A}_\mathcal{B}$. Uma relação de conflito relativizada entre ações $\alpha \#_r \beta$, com $\alpha, \beta \in \mathcal{A}_\mathcal{B}$, indica que as duas ações não podem ser executadas de forma concomitante por um mesmo indivíduo executor. A forma equivalente dessa relação de conflito é dada por $\alpha \#_r \beta \iff \forall i \in \mathcal{I}, \langle i, \alpha, x\rangle \# \langle i, \beta, y\rangle$ com $x, y \in \mathcal{I}$.

Com os cenários de conflitos definidos, o algoritmo de detecção busca por conflitos de operadores deônticos associados a uma mesma ação ou a ações distintas que estejam numa relação de conflito. O Algoritmo 3 descreve os passos da detecção de conflitos no autômato $\mathcal{A}(\mathcal{C})$ a partir de um estado $s \in S$ e das informações deônticas retornadas pela função $\delta(s)$. Para cada conjunto de rótulos deônticos $\mathcal{D} \in \delta(s)$, um estado s possui um conflito entre operadores caso exista algum elemento $d \in D$ que esteja em conflito com algum elemento d' de um conjunto $\mathcal{D}' \in \delta(s) - \mathcal{D}$.

Algoritmo 3: $buscarConflitos(s)$

```
input  : Um estado s do autômato A(C)
output: Uma situação de conflito.
begin
    for D ∈ δ(s) do
        for D' ∈ δ(s) − {D} do
            if ∃d ∈ D | f_#(d) ∩ D' ≠ ∅ then
                return Conflito entre d e f_#(d) ∩ D';
            end
        end
    end
    return Nenhum Conflito;
end
```

O conjunto de operadores deônticos em conflito com outro operador é obtido pela função $f_{\#} : \mathcal{M} \rightarrow 2^{\mathcal{M}}$, onde dado um operador deôntico a função retorna os operadores em conflito com este operador. Para cada $d \in \mathcal{D}$, o algoritmo verifica a existência de algum elemento de $\mathcal{D}'$ no conjunto de operadores retornado por $f_{\#}(d)$. Os retornos da função $f_{\#}$ define definidos na Figura 26, onde $\mathcal{R} = \{g,\ _i,\ _{i \curvearrowright j}\}$ são os tipos de relativizações, $\alpha \in \mathcal{A}_{\mathcal{B}}$ é uma ação básica e $i, j, x, y \in \mathcal{I}$ são os indivíduos envolvidos no contrato.

$$f_{\#}(O_\alpha) = \{{}_rF_\alpha \in \mathcal{M} \mid r \in \mathcal{R}, \forall x, y \in \mathcal{I} \text{ com } i = x, j = y\} \cup \{{}_rd_\beta \in \mathcal{M} \mid r \in \mathcal{R}, \quad (4.85)$$
$$d \in \{O, P\}, \forall(\alpha, \beta) \in (\#_g \cup \#_r) \text{ com } \beta \in \mathcal{A}_{\mathcal{B}}, \forall x, y \in \mathcal{I} \text{ com } i = x, j = y\}$$
$$f_{\#}(P_\alpha) = \{{}_rF_\alpha \in \mathcal{M} \mid r \in \mathcal{R}, \forall x, y \in \mathcal{I} \text{ com } i = x, j = y\} \cup \{{}_rO_\beta \in \mathcal{M} \mid r \in \mathcal{R}, \quad (4.86)$$
$$\forall(\alpha, \beta) \in (\#_g \cup \#_r),\ \text{com } \beta \in \mathcal{A}_{\mathcal{B}}, \forall x, y \in \mathcal{I} \text{ com } i = x, j = y\}$$
$$f_{\#}(F_\alpha) = \{{}_rd_\alpha \in \mathcal{M} \mid r \in \mathcal{R}, d \in \{O, P\}, \forall x, y \in \mathcal{I} \text{ com } i = x, j = y\} \quad (4.87)$$
$$f_{\#}({}_xO_\alpha) = \{{}_rF_\alpha \in \mathcal{M} \mid r \in \mathcal{R}, \forall y \in \mathcal{I} \text{ com } i = x, j = y\} \cup \quad (4.88)$$
$$\{{}_rd_\beta \in \mathcal{M} \mid r \in \mathcal{R}, d \in \{O, P\}, \forall(\alpha, \beta) \in \#_g \text{ com } \beta \in \mathcal{A}_{\mathcal{B}}, \forall x, y \in \mathcal{I} \text{ com } i = x, j = y\}$$
$$\cup \{{}_rd_\beta \in \mathcal{M} \mid r \in \mathcal{R}, d \in \{O, P\}, \forall(\alpha, \beta) \in \#_r \text{ com } \beta \in \mathcal{A}_{\mathcal{B}}, \forall y \in \mathcal{I} \text{ com } j = y\}$$
$$f_{\#}({}_xP_\alpha) = \{{}_rF_\alpha \in \mathcal{M} \mid r \in \mathcal{R}, \forall y \in \mathcal{I} \text{ com } i = x, j = y\} \cup \{{}_rO_\beta \in \mathcal{M} \mid r \in \mathcal{R}, \quad (4.89)$$
$$\forall(\alpha, \beta) \in \#_g \text{ com } \beta \in \mathcal{A}_{\mathcal{B}}, \forall x, y \in \mathcal{I} \text{ com } i = x, j = y\} \cup \{{}_rO_\beta \in \mathcal{M} \mid r \in \mathcal{R},$$
$$\forall(\alpha, \beta) \in \#_r \text{ com } \beta \in \mathcal{A}_{\mathcal{B}}, \forall y \in \mathcal{I} \text{ com } j = y\}$$
$$f_{\#}({}_xF_\alpha) = \{{}_rd_\alpha \in \mathcal{M} \mid r \in \mathcal{R}, d \in \{O, P\}, \forall y \in \mathcal{I} \text{ com } i = x, j = y\} \quad (4.90)$$
$$f_{\#}({}_{x \curvearrowright y}O_\alpha) = \{{}_rF_\alpha \in \mathcal{M} \mid r \in \mathcal{R},\ \text{com } i = x, j = y\} \cup \{{}_rd_\beta \in \mathcal{M} \mid r \in \mathcal{R}, \quad (4.91)$$
$$d \in \{O, P\}, \forall(\alpha, \beta) \in \#_g \text{ com } \beta \in \mathcal{A}_{\mathcal{B}}, \forall x, y \in \mathcal{I} \text{ com } i = x, j = y\} \cup \{{}_rd_\beta \in \mathcal{M} \mid r \in \mathcal{R},$$
$$d \in \{O, P\}, \forall(\alpha, \beta) \in \#_r \text{ com } \beta \in \mathcal{A}_{\mathcal{B}}, \forall y \in \mathcal{I} \text{ com } j = y\}$$
$$f_{\#}({}_{x \curvearrowright y}P_\alpha) = \{{}_rF_\alpha \in \mathcal{M} \mid r \in \mathcal{R},\ \text{com } i = x, j = y\} \cup \{{}_rO_\beta \in \mathcal{M} \mid r \in \mathcal{R}, \quad (4.92)$$
$$\forall(\alpha, \beta) \in \#_g \text{ com } \beta \in \mathcal{A}_{\mathcal{B}}, \forall y \in \mathcal{I} \text{ com } i = x, j = y\} \cup \{{}_rO_\beta \in \mathcal{M} \mid r \in \mathcal{R}, \forall(\alpha, \beta) \in \#_r$$
$$\text{com } \beta \in \mathcal{A}_{\mathcal{B}}, \forall y \in \mathcal{I} \text{ com } j = y\}$$
$$f_{\#}({}_{x \curvearrowright y}F_\alpha) = \{{}_rd_\alpha \in \mathcal{M} \mid r \in \mathcal{R}, d \in \{O, P\},\ \text{com } i = x, j = y\} \quad (4.93)$$

Figura 26 – Função $f_{\#}$ para avaliação de conflitos.

Para ilustrar a aplicação da função $f\#$, considere a avaliação do operador de obrigação relativizado ${}_{x \curvearrowright y}O_\alpha$ na Equação (4.91) da Figura 26. O conjunto de operadores conflitantes com ${}_{x \curvearrowright y}O_\alpha$ é formado pela união de três subconjuntos de $\mathcal{M}$:

- operadores associados a uma mesma ação, denotado por $\{{}_rF_\alpha \in \mathcal{M} \mid r \in \mathcal{R},\ \text{com } i = x, j = y\}$, retornam todas as proibições de α executadas por x e recebidas por y.
- operadores associados a ações conflitantes globalmente com α, denotado por $\{{}_rd_\beta \in \mathcal{M} \mid r \in \mathcal{R}, d \in \{O, P\}, \forall(\alpha, \beta) \in \#_g, \forall x, y \in \mathcal{I} \text{ com } i = x, j = y\}$, retornam todas as obrigações e permissões relacionadas a uma ação β para todo par (α, β) que esteja numa relação de conflito global.

- operadores associados a ações conflitantes numa relativização com α, denotado por $\{_r d_\beta \in \mathcal{M} \mid r \in \mathcal{R}, d \in \{O, P\}, \forall(\alpha, \beta) \in \#_r, \forall y \in \mathcal{I} \text{ com } j = y\}$, retornam todas as obrigações e permissões relacionadas a uma ação β para todo par (α, β) que esteja numa relação de conflito relativizada onde o indivíduo executor é x.

4.5 Aplicação do método

Um exemplo simples de aplicação da técnica proposta pode ser observada num contrato de compra e venda onde é necessário a identificação dos indivíduo envolvidos. O contrato é descrito conforme segue:

> *(1) Após o cliente pagar o vendedor, o vendedor é obrigado a entregar o produto para o cliente; (2) após o vendedor entregar o produto para o cliente, o vendedor não pode entregar o produto para o cliente novamente.*

A fórmula $i \curvearrowright j[a]\ {}_{j\curvearrowright i}O(b) \wedge\ {}_{j\curvearrowright i}[b]\ {}_{j\curvearrowright i}F(b)$, onde $i, j \in \mathcal{I}$ e $a, b \in \mathcal{A}_\mathcal{B}$, especifica o descrição do contrato utilizando $\mathcal{RCL}$. Os indivíduos são representados por i (cliente) e j (vendedor); as ações por a (pagar) e b (entregar o produto).

O conjunto de ações relativizadas do contrato é dado por $\mathcal{A}_r = \{(i, a, j),\ (i, b, j),\ (j, a, i),\ (j, b, i),\ (i, a, i),\ (i, b, i),\ (j, a, j),\ (j, b, j)\}$. Observe que o conjunto $\mathcal{A}_r$ representa todas as possíveis relativizações entre as ações do contrato. No entanto, várias ações não refletem um contrato do mundo real, tal como (i, a, i) onde um mesmo indivíduo executa e recebe uma ação. Num contrato são raras as ocorrências desse tipo de comportamento, permitindo uma redução considerável das ações relativizadas As ações concorrentes relativizadas $\mathcal{A}_r^2$ são obtidas a partir do conjunto de ações relativizadas e apresentadas na Tabela 2.

Em seguida, o autômato que representa o contrato $\mathcal{C}$ é construído por meio do Algoritmo 2. O autômato $\mathcal{A}(\mathcal{C})$ resultante é ilustrado na Figura 27. O estado inicial s_0 representa o contrato, onde $l(s_0) = i \curvearrowright j[a]\ {}_{j\curvearrowright i}O(b) \wedge\ {}_{j\curvearrowright i}[b]\ {}_{j\curvearrowright i}F(b)$. A partir de s_0 todas as ações descritas na Tabela 2 são avaliadas pela função de decomposição $f(\mathcal{C}, \alpha)$, produzindo os subcontratos associados aos respectivos estados, s_1, s_2, s_3 e s_4:

$$
\begin{aligned}
l(s_1) &= {}_{j\curvearrowright i}O(b), && \text{com } \alpha \in \{1, 5, 6, 11\};\\
l(s_2) &= {}_{j\curvearrowright i}F(b), && \text{com } \alpha \in \{4, 9, 10, 14\};\\
l(s_3) &= {}_{j\curvearrowright i}O(b) \wedge\ {}_{j\curvearrowright i}F(b), && \text{com } \alpha \in \{7, 12, 13, 15\};\\
l(s_4) &= \top, && \text{com } \alpha \in \{2, 3, 8\}.
\end{aligned}
$$

Note que a execução de qualquer uma das ações, 2, 3 ou 8, torna o contrato satisfeito, pois nenhum dos operadores dinâmicos são disparados, levando o contrato ao

Código	Ações Relativizadas Concorrentes
1	(i, a, j)
2	(i, b, j)
3	(j, a, i)
4	(j, b, i)
5	$(i, a, j) \times (i, b, j)$
6	$(i, a, j) \times (j, a, i)$
7	$(i, a, j) \times (j, b, i)$
8	$(i, b, j) \times (j, a, i)$
9	$(i, b, j) \times (j, b, i)$
10	$(j, a, i) \times (j, b, i)$
11	$(i, a, j) \times (i, b, j) \times (j, a, i)$
12	$(i, a, j) \times (i, b, j) \times (j, b, i)$
13	$(i, a, j) \times (j, a, i) \times (j, b, i)$
14	$(i, b, j) \times (j, a, i) \times (j, b, i)$
15	$(i, a, j) \times (i, b, j) \times (j, a, i) \times (j, b, i)$

Tabela 2 – Ações relativizadas concorrentes.

estado de satisfação s_4. Em cada um dos estados s_1, s_2 e s_3, criados a partir de uma decomposição do contrato, é obtido um conjunto de rótulos deônticos gerados pela função f_d, que repassados para δ são utilizados na detecção de conflitos. Caso algum impasse seja detectado o algoritmo de construção para e retorna o conflito encontrado.

A decomposição a partir de s_0 resulta no subcontrato representado em s_1. O Algoritmo 3 então realiza a verificação do estado s_1 que possui os operadores deônticos obtidos com a função $\delta(s_1) = \{\{_{j\curvearrowright i}O_b\}\}$. Como $_{j\curvearrowright i}O_b$ é o único operador deôntico de s_1 a verificação para e nenhum conflito é encontrado.

Na sequência, o estado s_1 é decomposto nos seguintes subcontratos:

$$l(s_4) = \top, \qquad \text{com } \alpha \in \{4, 7, 9, 10, 12, 13, 14, 15\};$$
$$l(s_5) = \bot, \qquad \text{com } \alpha \in \{1, 2, 3, 5, 6, 8, 11\}.$$

Como não há decomposição a partir dos estados s_4 e s_5, o algoritmo para, ambos são verificados e nenhum conflito é encontrado. De maneira similar ao estado s_1, s_2 é verificado e nenhum conflito é detectado. O estado s_2 então é decomposto nos seguintes subcontratos:

$$l(s_4) = \top \qquad \text{com } \alpha \in \{1, 2, 3, 5, 6, 8, 11\};$$
$$l(s_5) = \bot \qquad \text{com } \alpha \in \{4, 7, 9, 10, 12, 13, 14, 15\}.$$

Em ambos os casos de decomposição, o estado s_5 representa uma violação, sendo rotulado com o valor $\bot$, indicando que o subcontrato decomposto não é satisfeito com a ação executada.

Já no estado s_3 o algoritmo de detecção encontra um conflito entre os operadores e o algoritmo de construção do autômato para, marcando o estado s_3 como conflitante. O conflito é detectado pelo Algoritmo 3 através dos seguintes passos:

1. os operadores deônticos de s_3 são obtidos pela função $\delta(s_3) = \{\{{}_{j\curvearrowright i}O_b\}, \{{}_{j\curvearrowright i}F_b\}\}$;
2. para cada elemento $\mathcal{D} \in \delta(s_3)$, onde $\mathcal{D} = \{\ {}_{j\curvearrowright i}O_b\}$, é verificado se há um conflito com outro elemento $\mathcal{D}' \in (\delta(s_3) - \{\mathcal{D}\})$;
3. para cada elemento $d \in \mathcal{D}$ é obtido, por meio da função $f_{\#}$, o conjunto de operadores em conflito com d. No caso de $d = {}_{j\curvearrowright i}O_b$, os operadores que representam um conflito é dado por $f_{\#}(d) = \{\ {}_{j\curvearrowright i}F_b,\ {}_{j}F_b, F_b\}$.
4. ao verificar se d está em conflito com algum outro elemento de $\delta(s_3)$, uma vez que $\mathcal{D}' = \{\ {}_{j\curvearrowright i}F_b\}$, temos que $f_{\#}(\ {}_{j\curvearrowright i}O_b) \cap \mathcal{D}' = {}_{j\curvearrowright i}F_b$ e assim um conflito entre ${}_{j\curvearrowright i}O_b$ e ${}_{j\curvearrowright i}F_b$ é detectado.

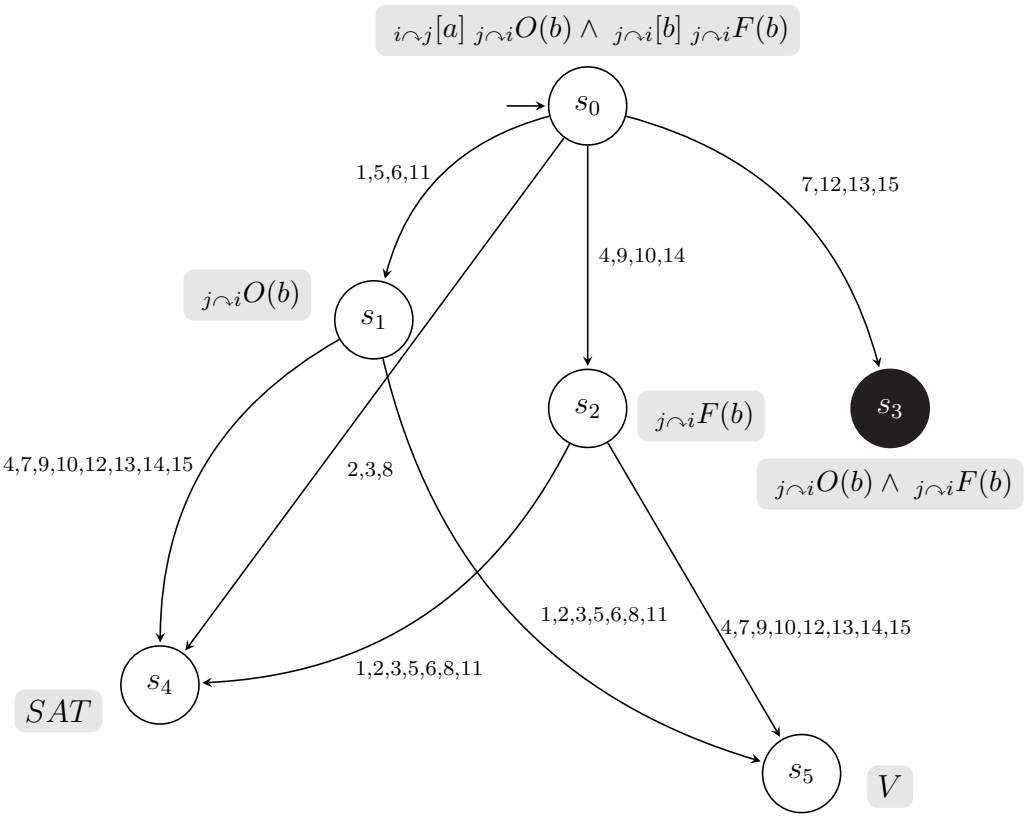

Figura 27 – Autômato construído a partir do contrato $\mathcal{C}$.

Observe que o algoritmo de detecção de conflitos avalia cada estado criado pelo algoritmo de construção do autômato, procurando situações de conflito nas decomposições do contrato. Na decomposição do estado s_3, que possui o contrato ${}_{j\curvearrowright i}O(b) \wedge\ {}_{j\curvearrowright i}F(b)$, é encontrado um conflito entre a obrigação e a proibição relativizadas. Consequentemente, o algoritmo para e determina que o contrato possui um conflito, marcando o estado s_3 como conflitante.

Note a diferença entre o estado s_3, que possui um conflito entre os operadores do contrato, e o estado s_5, que indica a execução de uma ação que viola o contrato. Essa ação leva o contrato a um estado onde as cláusulas são violadas. Na construção do autômato que representa o contrato, todas as transições de um estado com conflito vão para o estado de violação, e a partir deste estado com conflito não há possibilidade do contrato ser satisfeito. Entretanto, como o objetivo do algoritmo é encontrar um conflito, o processamento para antes que essa decomposição para a violação seja executada.

5 A FERRAMENTA DE DETECÇÃO DE CONFLITOS

O método de detecção de conflitos proposto foi desenvolvido através do paradigma orientado a objetos. A análise e projeto da ferramenta foram realizados com o suporte da linguagem UML [65] e a linguagem de programação escolhida para o desenvolvimento foi Java [66], devido principalmente a sua característica multiplataforma. A arquitetura da ferramenta proposta, denominada *RECALL*[1], é apresentada na Figura 28. Os retângulos representam os módulos da ferramenta, os retângulos inclinados indicam os arquivos de entrada e saída, enquanto as setas denotam as mensagens trocadas entre os módulos e os rótulos indicam os dados repassados na operação.

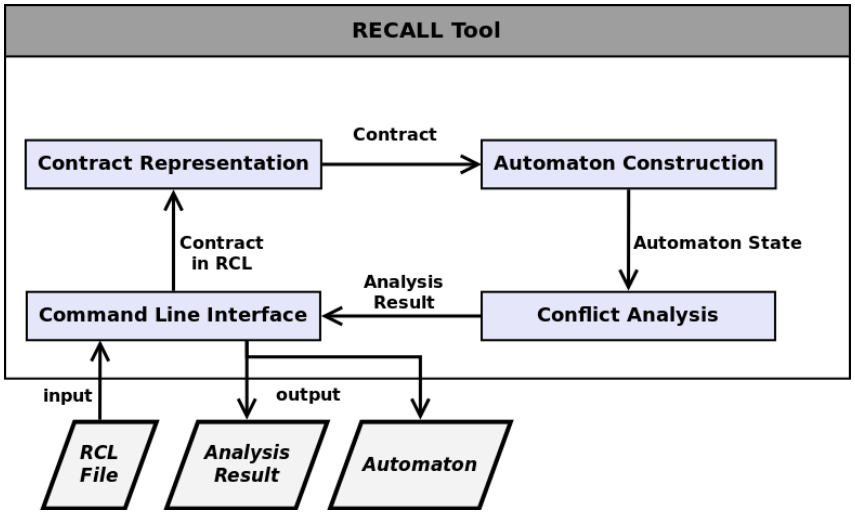

Figura 28 – Arquitetura proposta da ferramenta.

A ferramenta proposta possui um módulo com uma interface por linha de comando (*Command Line Interface*) que recebe as configurações da análise e o arquivo que descreve o contrato, e em seguida retorna o resultado da análise. O módulo *Contract Representation* extrai o contrato do arquivo descrito em $\mathcal{RCL}$ e verifica se o contrato é bem formado, conforme a sintaxe proposta na Seção 4.1. O autômato que representa o contrato é obtido pelo módulo *Automaton Construction*, implementado de acordo com o Algoritmo de Construção e a Função de Decomposição (Veja Seção 4.3) Já o módulo *Conflict Analysis* realiza a busca por conflitos no contrato decomposto usando as funções auxiliares para extração de informações deônticas f_d e para a avaliação de conflitos $f_{\#}$, descritos na Seção 4.4.

[1] RECALL é um acrônimo de **R**elativiz**E**d **C**ontr**A**ct **L**anguage ana**L**izer ou Analisador da Linguagem de Contratos Relativizada.

5.1 Análise e projeto

A fase de análise e projeto da ferramenta foi realizada com suporte dos diagramas provenientes da UML. A estrutura dos módulos foram descritos pelos diagramas de classes e de pacotes enquanto que, para a especificação comportamental e integração dos módulos, foram utilizados os diagramas de sequência.

O diagrama da Figura 29 descreve os pacotes que compõem o projeto da ferramenta RECALL conforme a arquitetura apresentada na Figura 28. A leitura dos contratos descritos em $\mathcal{RCL}$ é realizada pelas classes do pacote *parser*. No pacote *model.contracts* estão presentes as classes relacionadas com a representação dos contratos, operadores e modalidades deônticas e dinâmicas, enquanto o pacote *model.actions* contém as ações básicas e compostas de um contrato. O autômato gerado a partir dos algoritmos de construção é representado pelas classes do pacote *model.automata*. Já o pacote *algorithms* possui as classes que representam os módulos *Automaton Construction* e *Conflict Analysis* que por sua vez representam as funcionalidades dos algoritmos de construção de autômato e busca por conflitos. Por fim, os pacotes *run* e *util* contém, respectivamente, as classes da interface por linha de comando e classes auxiliares para a execução dos algoritmos.

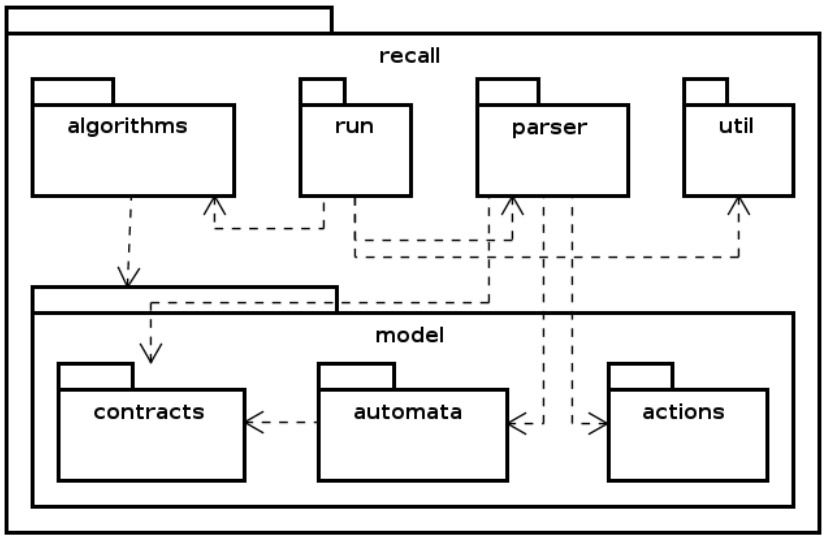

Figura 29 – Diagrama de pacotes do projeto da ferramenta.

Com base na arquitetura e no diagrama de pacotes apresentados, as subseções a seguir descrevem o projeto de cada módulo proposto, bem como o comportamento e a interação entre suas classes.

5.1.1 Representação dos contratos

Para que um contrato seja adequadamente representado na ferramenta, foi desenvolvido um analisador sintático capaz de interpretar um arquivo bem formado conforme

a sintaxe da $\mathcal{RCL}$ e retornar um objeto com a estrutura de dados correspondente às definições deste contrato. A Figura 30 apresenta o diagrama de classes do pacote *parser*, responsáveis pelo funcionamento do analisador sintático. A classe *ContractLoader* encapsula o processo de reconhecimento do contrato em $\mathcal{RCL}$ de um arquivo ou de uma variável.

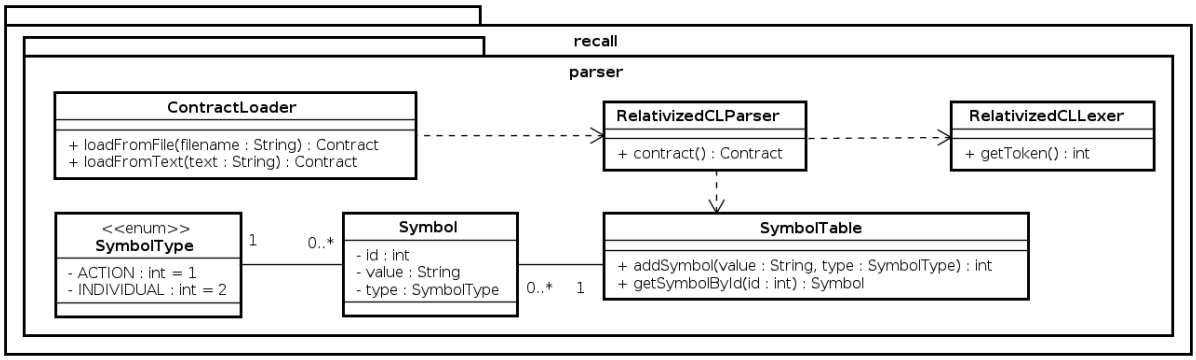

Figura 30 – Diagrama de classes do analisador sintático para $\mathcal{RCL}$.

Ao interpretar um arquivo de contrato, o analisador sintático retorna um objeto da classe *Contract* que possui um conjunto de cláusulas representado pelo atributo *clauses* e os conjuntos de conflitos globais e relativizados indicados pelos atributos *globalConflicts* e *relativizedConflicts*, respectivamente. O diagrama de classes do contrato, presente na Figura 31, engloba as classes responsáveis por representar as cláusulas deônticas e dinâmicas, os conflitos predefinidos e as ações básicas e compostas do contrato.

Cada cláusula do contrato, representada pela classe abstrata *Clause*, possui o atributo *relativizationType* que define o tipo de relativização. Eventualmente, uma cláusula pode conter informações sobre o executor e o recebedor da ação envolvida, respeitando os atributos *sender* e *receiver*. O atributo *action* contém a ação relacionada com a cláusula e o atributo *value* determina se a cláusula já foi avaliada pelo processo de construção do autômato. Fórmulas mais complexas podem ser expressas pela composição de cláusulas definida pelo atributo *composition*.

As modalidades deônticas são representadas pela classe *DeonticClause* especializada de *Clause*. O atributo *action* herdado de *clause* representa a ação que está relacionada à modalidade deôntica. Os atributos *deonticType* e *penalty* determinam o tipo de modalidade deôntica e a cláusula de penalidade, quando aplicável. Já a modalidade dinâmica, classe *DynamicClause*, considera o atributo *action* como o gatilho do operador de modo a tornar ativa a cláusula do atributo *clause*.

De maneira semelhante, as ações básicas e compostas são representadas pelas classes *BasicAction* e *ComposedAction* que implementam a interface *Action*. Quando uma ação está relacionada com modalidades deônticas pode ser composta pelos operadores de concorrência, sequência e escolha. No caso de uma modalidade dinâmica, a ação composta

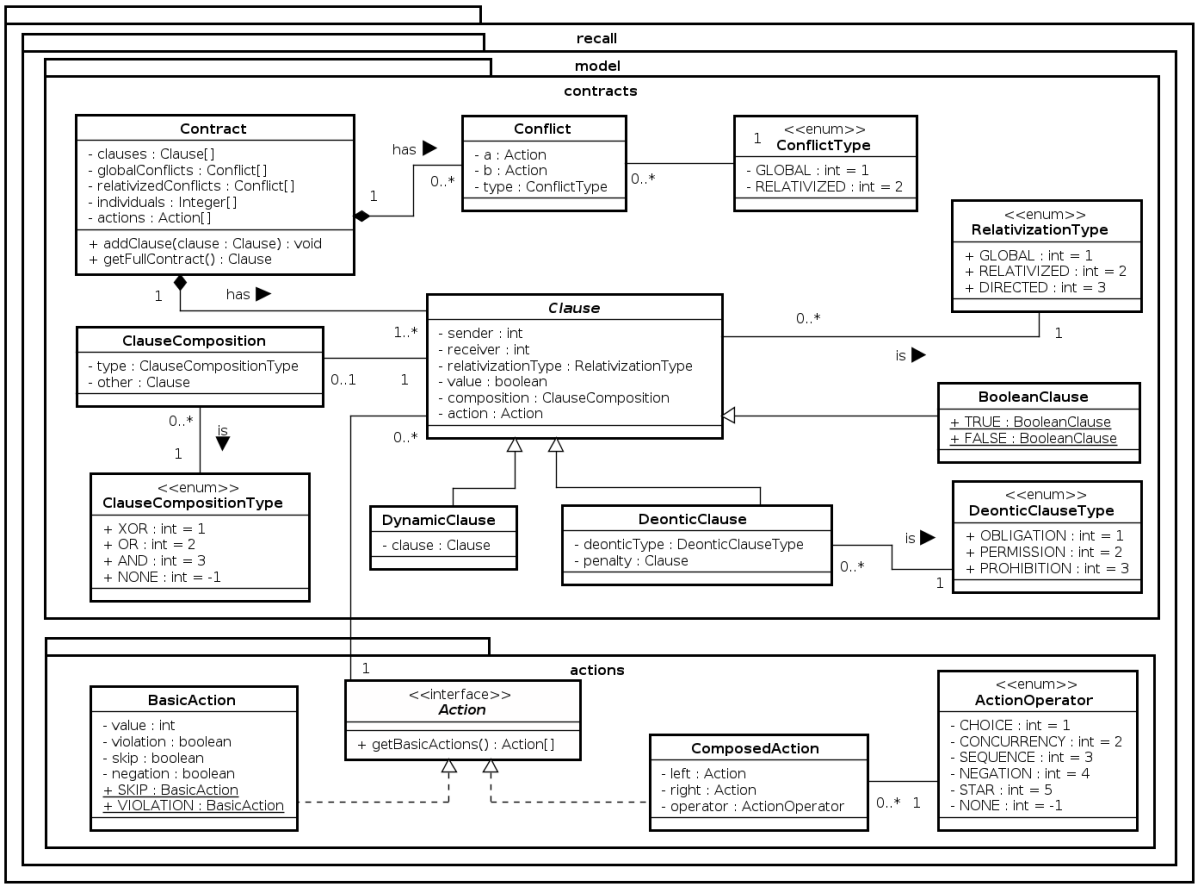

Figura 31 – Diagrama de classes para representação de um contrato.

pode conter além dos operadores já mencionados, a negação da ação ou a sua repetição não determinística. Em ambos os casos, a composição é determinada pela classe *ComposedAction*, que contém o operador e as ações. Essa definição de operadores para a composição é representada pela enumeração *ActionOperator* e a restrição dos operadores é realizada pelo analisador sintático ao reconhecer a linguagem proveniente de um arquivo $\mathcal{RCL}$.

A interação entre as classes responsáveis pela análise sintática e as classes que representam um contrato pode ser acompanhada pelo diagrama de sequência da Figura 32. A atividade do componente Analisador Sintático se inicia com a classe principal *Main* através do método *loadFromFile* da classe *ContractLoader*. A partir dessa chamada, os analisadores sintático e léxico iniciam o processamento do arquivo contendo o contrato. O analisador léxico, classe *RelativizedCLLexer*, faz a leitura do arquivo e repassa ao analisador sintático cada novo símbolo encontrado. Caso o símbolo seja uma ação ou um indivíduo, o analisador sintático adiciona a informação na tabela de símbolos, classe *SymbolTable*. Caso contrário, o símbolo é uma clásula, e assim deve ser encaminhada para a lista de cláusulas do contrato por meio do método *addClause*.

Ao final do processamento um objeto da classe *Contract* está preenchido com as cláusulas do contrato e pode ser repassado ao módulo de Construção do Autômato.

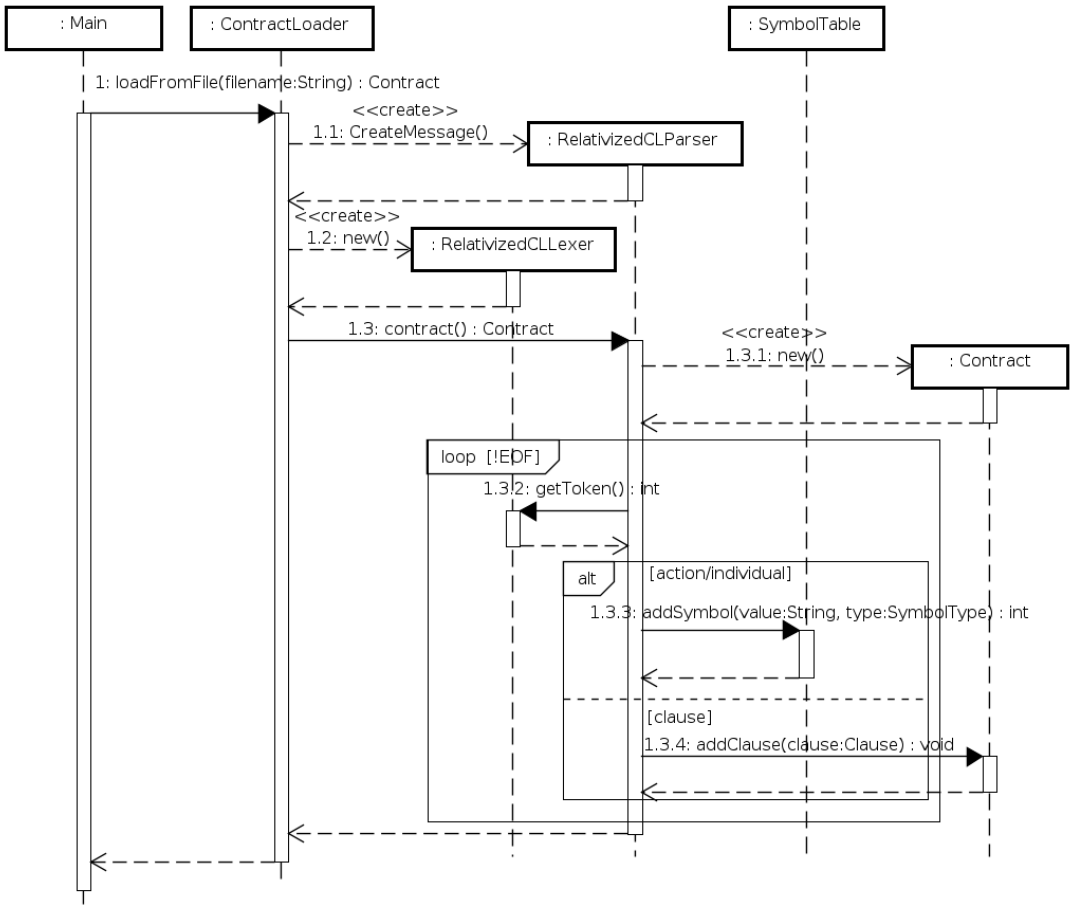

Figura 32 – Diagrama de sequência do Analisador Sintático.

5.1.2 Construção do Autômato

O módulo de construção do autômato é representado na Figura 33. As classes representam um autômato e as funcionalidades do módulo de construção juntamente com os algoritmos necessários.

A Função de Decomposição é implementada pelo método *decompose*, da classe *ClauseDecomposer*, que recebe como parâmetros uma cláusula e um conjunto de ações relativizadas retornando uma cláusula resultante. O métodos auxiliares *decomposeDeontic* e *decomposedDynamic* são responsáveis por decompor as cláusulas deônticas e dinâmicas. No caso de cláusulas que possuem ações compostas, o método *processComposedActions* aplica as decomposições necessárias retornando novas cláusula equivalentes formadas apenas por ações atômicas.

A classe *AutomataConstructor* implementa o algoritmo de construção do autômato com o método *constructAutomaton* executado a partir do estado inicial do autômato. O resultado do processamento do contrato pela classe *AutomataConstructor* é um autômato que representa o contrato. O contrato é definido no diagrama da Figura 33 pela classe *Automaton* do pacote *model.automata*, que possui os atributos que contém os conjuntos

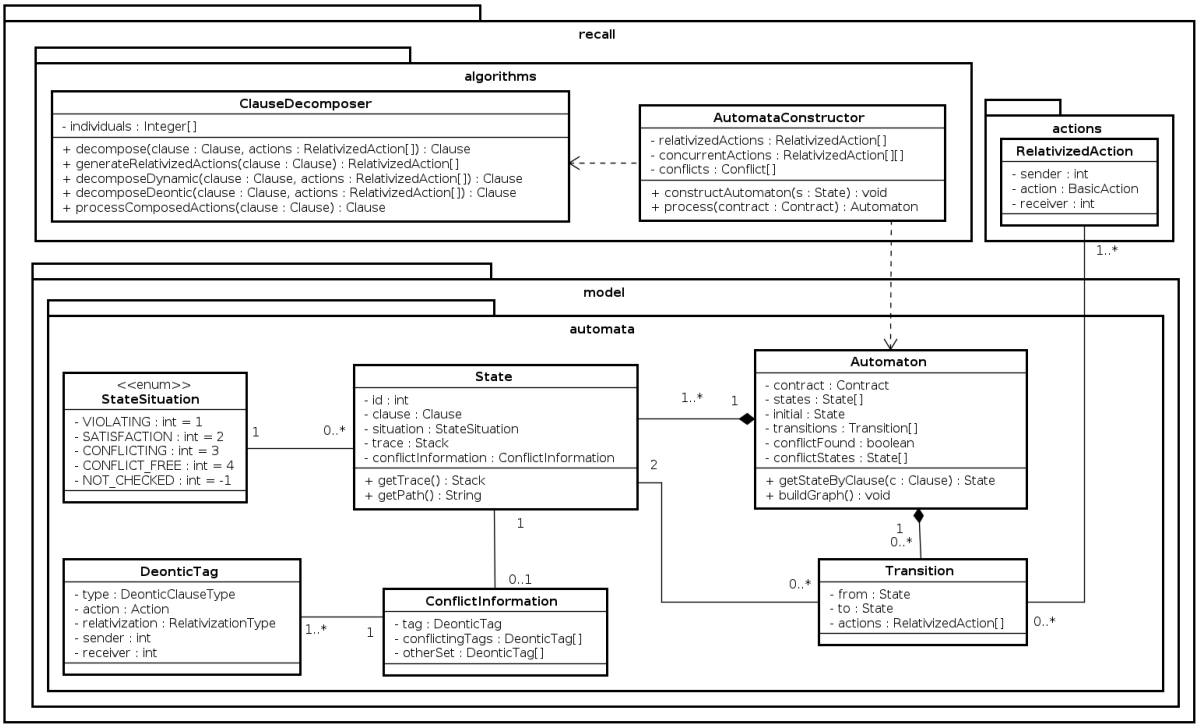

Figura 33 – Classes de representação e construção do autômato.

de estados e transições. As transições possuem os atributos *from* e *to* para determinar os estados de origem e destino, assim como o atributo *actions* com um conjunto de ações relativizadas representando as ações concorrentes utilizadas na decomposição de um próximo estado. A classe *RelativizedAction* denota uma ação relativizada que possui além de uma ação básica, o indivíduo executor e recebedor da ação. Um estado do autômato é representado pela classe *State* que possui a cláusula decomposta, além das informações de satisfação ou violação. Caso um estado possua cláusulas em conflito, o atributo *conflictInformation* possui os detalhes da verificação realizada.

O diagrama de sequência da Figura 34 mostra o comportamento do construtor do autômato. A construção segue os passos definidos no algoritmo de construção onde o contrato é decomposto de acordo com o conjunto de ações possível de ser executado e o resultado da decomposição é adicionado a um novo estado do autômato. A cada novo estado o resultado das decomposições, uma nova decomposição é realizada até que não existam mais decomposições possíveis.

5.1.3 Detecção de conflitos

O módulo analisador de conflitos funciona de forma concomitante a construção do autômato. A cada novo estado criado na construção, uma busca por situações conflitantes também é realizada. O diagrama de classes da Figura 35 representa o algoritmo de busca por conflitos e as suas funções auxiliares.

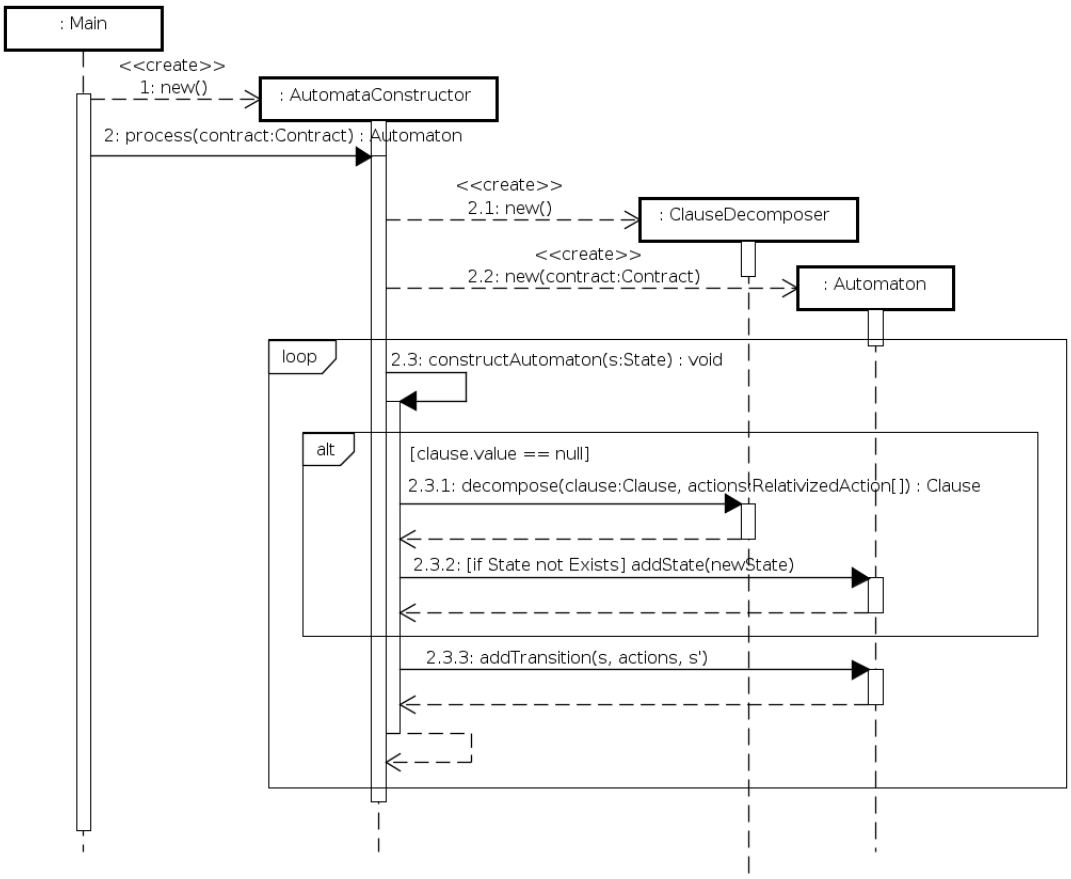

Figura 34 – Diagrama de sequência do Construtor do Autômato.

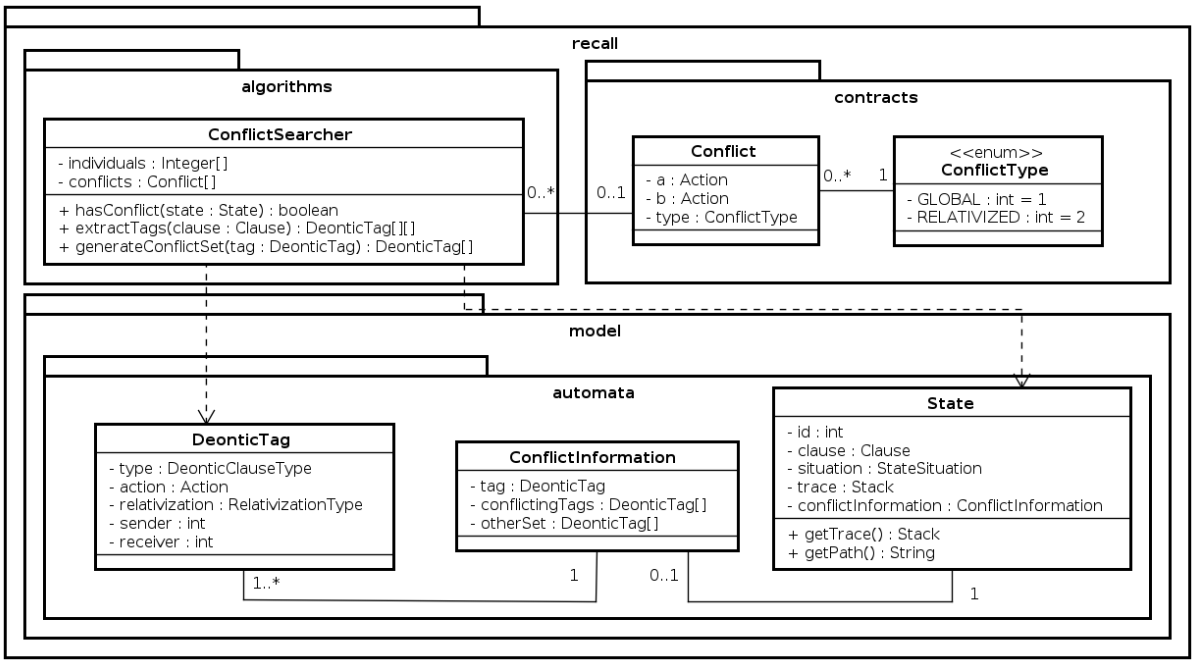

Figura 35 – Diagrama de classes da detecção de conflitos.

O algoritmo de detecção de conflitos é implementado pelo método *hasConflict* da classe *ConflictSearcher*. Este método recebe um estado, objeto da classe *State*, e verifica se há um conflito. Se for caso, o estado é marcado como conflitante, no atributo *situation*, e as cláusulas conflitantes são armazenadas em *conflictInformation*.

Para auxiliar a detecção de conflitos, o método *extractTags* retorna a lista de marcadores deônticos conforme definido na função f_d. Os marcadores deônticos representados pela classe *DeonticTag* contêm as informações relacionadas ao operador deôntico, ação e tipo de relativização das modalidades analisadas. A principal diferença entre as classes *DeonticClause*, do pacote *model.contracts*, e *DeonticTag*, em *model.automata*, é que esta última possui apenas informações necessárias para a análise de conflitos, como o operador deôntico e a ação relacionada, descartando outras características mais complexas como penalidades, operadores entre ações e cláusulas compostas. Já a função $f_{\#}$, implementada pelo método *generateConflictSet*, retorna os possíveis conflitos que podem ocorrer em relação ao marcador deôntico informado. Para os conflitos relacionados à mesma ação, são utilizadas as definições descritas na Seção 4.4. Já no caso dos conflitos pré-definidos, considera-se o atributo *conflicts*, para o qual são gerados os conjuntos de marcadores em conflito para cada marcador obtido na análise de conflitos.

O diagrama de sequência da Figura 36 representa a interação da classe *ConflictSearcher* com o construtor do autômato e demais classes de apoio. Ao criar uma nova decomposição do contrato analisado, a classe *ConflitcSearcher* verifica se existe um conflito na fórmula. O método *hasConflict* da classe *ConflitcSearcher* invoca o método *extractTags* obtendo os marcadores deônticos da fórmula. Para cada marcador da fórmula são verificados os possíveis conflitos com o método *generateConflictSet* e em caso afirmativo o estado é marcado como conflitante.

5.2 Desenvolvimento

De acordo com a análise e projeto da ferramenta, descrito na seção anterior, a ferramenta foi implementada na linguagem Java [66]. O paradigma orientado a objetos permite a extensão e melhoria da ferramenta de forma mais fácil e organizada, além da integração com possíveis outras ferramentas [67]. A escolha da linguagem Java para a implementação se deve a alguns aspectos, tais como: ser orientada a objetos, independente de plataforma, robusta e amplamente difundida [68]. Um ambiente integrado de desenvolvimento, *Netbeans IDE* [69], auxiliou a construção da ferramenta, fornecendo um editor próprio para a linguagem, além de ferramentas de depuração, documentação e produção de código executável.

Algumas bibliotecas de código aberto também foram adotadas para auxiliar o desenvolvimento da ferramenta. Para interpretar o contrato escrito em $\mathcal{RCL}$ foi utilizada a

biblioteca *ANTLR* [70] que, a partir de especificações gramaticais, gera os analisadores léxico e sintático em código Java, permitindo assim sua integração com os outros componentes da ferramenta. A implementação do interpretador de parâmetros em linha de comando contou com a ajuda da biblioteca *Apache Commons CLI* [71]. Já os componentes de construção do autômato e análise de conflitos foram implementados com suporte da biblioteca *Guava* [72], que provê funções otimizadas para operações em conjuntos como intersecção, união e o conjunto das partes.

Após a análise do contrato, o autômato gerado é exportado para um arquivo em formato DOT[73]. Assim, o autômato pode ser visualizado com as ferramentas disponíveis na biblioteca *Graphviz* [74]. Tanto a linguagem DOT quanto a ferramenta *Graphiviz* possuem implementações multiplataforma de código aberto e são frequentemente adotados para a criação e visualização de grafos em diversas áreas da computação [75].

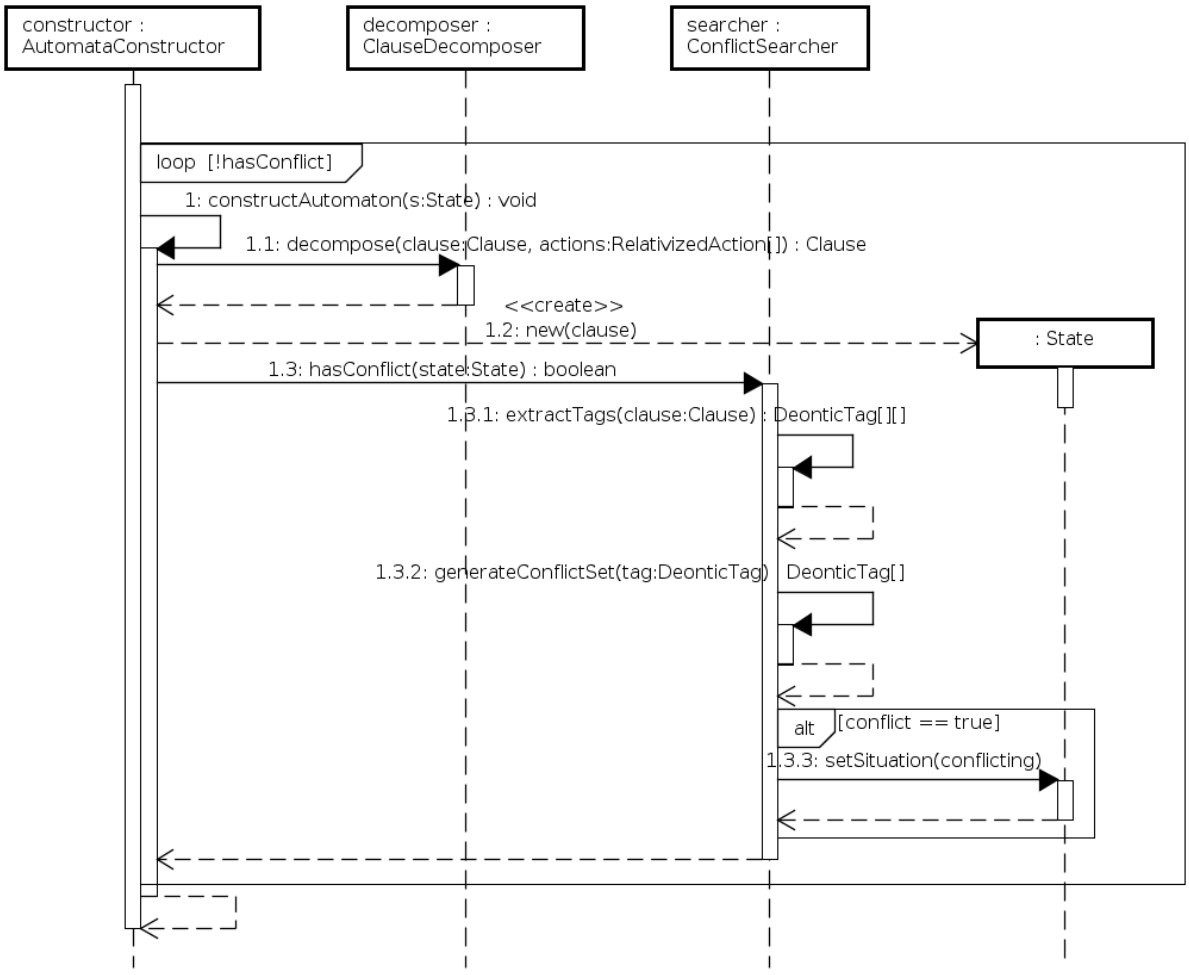

Figura 36 – Diagrama de sequência para a detecção de conflitos.

5.2.1 Descrição geral

Por meio da gramática da $\mathcal{RCL}$, descrita na Seção 4.1, a ferramenta *ANTLR* [70] gera o código-fonte em Java de classes com as funções do analisador léxico, classe *RelativizedCLLexer*, e sintático, classe *RelativizedCLParser*. A partir dessas classes básicas foram desenvolvidas as classes *ContractLoader*, que lê o contrato a partir do arquivo de especificação, e *SymbolTable*, que contém a lista de identificadores encontrada na análise sintática do contrato. Cada identificador é uma ação básica se for do tipo *SymbolType.ACTION*, ou um indivíduo, caso seja *SymbolType.INDIVIDUAL*.

Em seguida, a construção do autômato que representa o contrato é realizada conforme o código da Figura 37, que implementa o algoritmo de construção. Na linha (4) do código de construção cada estado é verificado em busca de conflitos antes de iniciar sua decomposição. Esta verificação permite a interrupção do processo caso um conflito seja encontrado, evitando decomposições desnecessárias, conforme se observa nas linhas (7) a (9). Um contrato é livre de conflitos se todas as suas decomposições estão livres de conflito. Quando uma situação conflitante é encontrada o processo de verificação pode ser encerrado com a apresentação de um contraexemplo. O contraexemplo revela exatamente a sequência de decomposições que levam o contrato a uma situação de conflito. Contudo, caso seja necessária a verificação completa do contrato em busca de todos os conflitos existentes, é possível configurar a ferramenta para que a construção só termine quando o contrato for completamente decomposto.

```
1  void constructAutomaton(State s) {
2   Clause c1 = s.getClause();
3   if (c1.getValue() == null) {
4    if (searcher.hasConflict(s))
5     automaton.setConflictFound(true);
6    for (Set<RelativizedAction> a : generateActions(c1)) {
7     if ((automaton.isConflictFound() && !config.isContinueOnConflict())
8     || s.getSituation() == StateSituation.conflicting)
9      return;
10    Clause c2 = decomposer.decompose((Clause) c1.clone(), a);
11    State s1 = automaton.getStateByClause(c2);
12    if (s1 != null) {
13     automaton.getTransitions().add(new Transition(s, s1, a));
14    } else {
15     s1 = new State(c2);
16     automaton.getStates().add(s1);
17     automaton.getTransitions().add(new Transition(s, s1, a));
18     constructAutomaton(s1);
19     s1.getTrace().add(new Transition(s, s1, a));
20    }
21   }
22  } else
23   s.setSituation(c1.getValue()?StateSituation.satisfaction :StateSituation.violating);
24 }
```

Figura 37 – Código-fonte do método *constructAutomaton*.

Conforme pode ser visualizado na linha (10) da Figura 37, a construção do autômato depende das decomposições realizadas pela classe *ClauseDecomposer*, aqui representado pelo objeto *decomposer*. Esta classe implementa, por meio do método *decompose*, a função de decomposição f que recebe uma cláusula e um conjunto de ações relativizadas para retornar a cláusula resultante após a execução das ações informadas. O método *decompose* realiza a decomposição através de métodos específicos para cada modalidade. A decisão de tratar cada operador deôntico e dinâmico separadamente se deve a maior facilidade em testar e corrigir o código desenvolvido. Sendo assim, existem os métodos privados *decomposeDeontic* e *decomposeDynamic* para tratar respectivamente das decomposições de modalidades deônticas e do operador dinâmico.

A busca por conflitos é realizada a cada nova decomposição pelo método *hasConflict*, da classe *ConflictSearcher*, conforme descrito na Figura 38. O método *extractTags* na linha (3) implementa a função de extração de marcadores deônticos f_d (Veja Seção 4.4). Este método recebe uma cláusula e retorna os conjuntos de marcadores deônticos de acordo com a avaliação da cláusula. Cada elemento desse conjunto de marcadores, e.g. d_i, é verificado em relação a outros elementos dos outros conjuntos, e.g. d_j, em busca de potenciais conflitos.

```
1  boolean hasConflict(State state) {
2   Clause clause = state.getClause();
3   Set<Set<DeonticTag>> delta = extractTags(clause);
4   for (Set<DeonticTag> d1 : delta)
5    for (Set<DeonticTag> d2 : Sets.difference(delta, Sets.newHashSet(d1)))
6     for (DeonticTag d : d1) {
7      Set<DeonticTag> conflictSet = generateConflictSet(d);
8      Set<DeonticTag> inter = Sets.intersection(conflictSet, d2);
9      if (!inter.isEmpty()) {
10      state.setSituation(StateSituation.CONFLICTING);
11      state.setConflictInformation(new ConflictInformation(d,inter,d2));
12      return true;
13     }
14    }
15   state.setSituation(StateSituation.CONFLICTFREE);
16   return false;
17  }
```

Figura 38 – Código-fonte do método *hasConflict*.

A função $f\#$ retorna os conflitos possíveis entre operadores relacionados a d, para todo $d \in d_i$, e para qualquer $d' \in d_j$, onde $d_j \neq d_i$. Um conflito ocorre quando pelo menos um elemento de d_i está em conflito com algum outro elemento de d_j. Neste caso, o método *hasConflict*, linhas (10) a (12), adiciona um marcador no estado analisado com a situação *StateSituation.CONFLICTING* e armazena no atributo *conflictInformation* os objetos *DeonticTag* conflitantes. Com esta informação é possível gerar o *trace* que resulte num conflito para ser retornado pela ferramenta após o processamento.

5.2.2 Estratégias de otimização

A implementação da ferramenta RECALL seguiu as especificações do método proposto no Capítulo 4, conforme análise e projeto descritos na Seção 5.1. Contudo, testes sobre a ferramenta ao longo do processo de desenvolvimento permitiram a identificação de algumas estratégias de otimização sobre a construção do autômato que representa o contrato analisado. Essas estratégias para melhorar a eficiência da ferramenta, considerando memória e tempo de processamento, contam com: a ordenação das ações nas decomposições do contrato; a poda de ações relativizadas desnecessárias; e a poda de indivíduos irrelevantes em operadores relativizados e globais.

Ordenação das ações nas decomposições

Na análise de um contrato, onde $\mathcal{I}$ é o conjunto de indivíduos e $\mathcal{A}_{\mathcal{B}}$ é o conjunto de ações básicas, o conjunto de ações relativizadas $\mathcal{A}_r$ e o conjunto de ações concorrentes $\mathcal{A}_r^2$ devem ser obtidos pelo cálculo das combinações de indivíduos e ações. Na fase de construção do autômato, as decomposições são obtidas de acordo com as ações concorrentes de $\mathcal{A}_r^2$. A primeira estratégia de verificação consiste então em avaliar primeiramente os subconjuntos com o maior número de ações concorrentes em $\mathcal{A}_r^2$. Essa ordenação decrescente no número de ações aumenta a possibilidade de se encontrar mais rapidamente os potenciais conflitos, uma vez que um número maior de ações sendo executadas simultaneamente aumenta a probabilidade de se ocorrer um conflito. De fato, a natureza de contratos multilaterais mostra que grande parte das decomposições com conflitos são obtidas por meio da execução concorrente de ações relativizadas. Uma combinação de ações, sejam estas ações concorrentes ou ações executadas numa conjunção de cláusulas distintas, tendem, na prática, a gerar mais conflitos em relação a execução independente de ações relativizadas.

Porém, essa abordagem não é eficiente em casos onde a situação de conflito ocorre devido a não execução de uma determinada ação. Por exemplo, em penalidades de obrigações ou proibições onde a execução de um conjunto das ações relativizadas concorrentes não provocam uma situação de conflito, já que as cláusulas não são violadas e, por consequência, não são ativadas. Num outro cenário, a decomposição do contrato com apenas uma ação relativizada pode ativar uma penalidade, gerando uma decomposição conflitante. Como o objetivo principal da ferramenta é detectar os conflitos existentes num contrato multilateral, normalmente obtidos pela combinação de vários contratos, optou-se então pela estratégia que potencializa a detecção de um conflito quanto antes possível durante o processo.

Poda de ações relativizadas desnecessárias

Ao considerar as etapas do método proposto, nota-se que a complexidade de construção do autômato depende mais do conjunto de ações básicas e indivíduos do que do

número de cláusulas presentes no contrato. O cálculo da combinação das ações relativizadas concorrentes cresce exponencialmente de acordo com o número de ações básicas e indivíduos do contrato. Várias dessas combinações não refletem os cenários de análise de um contrato na prática, e muitos estados e transições do autômato são construídos desnecessariamente. Para ilustrar essa situação, considere um contrato com $|\mathcal{I}| = 4$ e $|\mathcal{A}_{\mathcal{B}}| = 3$. O conjunto de ações relativizadas obtido é $|\mathcal{A}_r| = 48$ e as ações concorrentes, que refletem todas as possibilidades de combinação, é de $|\mathcal{A}_r^2| = 2^{48} - 1$. Fica claro que não apenas a ordenação das ações é importante, mas também a necessidade de contornar a explosão combinatória, eliminando casos que representam cenários irreais.

A estratégia realiza uma poda nas ações relativizadas, conforme o comportamento da função de decomposição, implementada pela classe *ClauseDecomposer*. O comportamento dessa função mostra que cada modalidade em nível atômico, sem qualquer operador entre as ações, pode resultar em duas possibilidades de decomposição, a satisfação ou a violação. A função de decomposição sempre retorna para qualquer modalidade deôntica sua satisfação ou violação, e de uma modalidade dinâmica são obtidas a satisfação ou uma nova cláusula ativada pelo gatilho dinâmico. Com base nesse comportamento, a estratégia de poda seleciona dinamicamente as ações relativizadas da decomposição corrente durante o processo de análise do contrato. Para cada modalidade da cláusula é possível extrair a ação que viola uma proibição, satisfaz uma obrigação ou ativa o gatilho de uma modalidade dinâmica. Assim apenas as ações relativizadas relacionadas ao subcontrato da decomposição são consideradas, enquanto que as ações irrelevantes da cláusula são desconsideradas na avaliação. Nas modalidades dinâmicas apenas as ações de gatilho são computadas, descartando o restante da cláusula. Já nas modalidades deônticas, as penalidades são descartadas na avaliação, uma vez que a cláusula só deve ser considerada quando a obrigação (ou proibição) relacionada é violada.

Essa estratégia de poda diminui consideravelmente a quantidade de ações relativizadas, ainda mantendo a construção de todos os estados relevantes das decomposições do contrato. A garantia de que o contrato é completamente avaliado está relacionada ao fato de que a partir das ações extraídas de cada cláusula, pelo menos as ações responsáveis pela satisfação dessas cláusulas são obtidas; no caso de obrigações, permissões e da modalidade dinâmica; ou pela violação das cláusulas, no caso de proibições. A partir da combinação dessas ações extraídas das cláusulas é possível obter todas as situações de satisfação e violação dessas cláusulas.

Para ilustrar o processo de poda, considere o contrato $\mathcal{C} = {}_{i\frown j}O(a) \wedge_{j\frown i} [b]_{j\frown k}F(c)$ com duas cláusulas numa conjunção. A ação relativizada $\langle i, a, j\rangle$ é extraída da cláusula ${}_{i\frown j}O(a)$, que garante a satisfação dessa obrigação, e a partir da cláusula ${}_{j\frown k}[b]_{j\frown k}F(c)$, obtém-se a ação $\langle j, b, k\rangle$, que dispara o gatilho dinâmico. Com essas duas ações, são obtidos os conjuntos $\mathcal{A}_r = \{\langle i, a, j\rangle, \langle j, b, k\rangle\}$ e $\mathcal{A}_r^2 = \{\{\langle i, a, j\rangle\}, \{\langle j, b, k\rangle\}, \{\langle i, a, j\rangle, \langle j, b, k\rangle\}\}$. Ao

processar esse conjunto de ações concorrentes é possível gerar todas as decomposições do contrato com um número reduzido de ações relativizadas concorrentes através da poda em relação ao método proposto inicialmente, onde todas as combinações entre ações e indivíduos eram levadas em conta. O conjunto de ações obtido sem essa estratégia de poda teria para o contrato $\mathcal{C}$, com o conjunto $\mathcal{A}_{\mathcal{B}} = \{a, b, c\}$ e o conjunto $\mathcal{I} = \{i, j, k, l\}$, o conjunto $\mid \mathcal{A}_r \mid = 36$. Mesmo com a aplicação da poda todas as decomposições são obtidas corretamente, visto que o complemento de cada ação é composto pelos elementos restantes do conjunto $\mathcal{A}_r^2$, assegurando tanto a satisfação quanto a violação (ou ativação do gatilho) das modalidades.

Poda de indivíduos irrelevantes

Outro fator de impacto na eficiência da ferramenta foi o número de operadores globais num contrato multilateral. Conforme descrito na semântica da $\mathcal{RCL}$ (Ver Seção 4.2), um operador global equivale a relativização dos operadores com todos os indivíduos do contrato. No caso de uma fórmula com obrigação global, sua satisfação ocorre quando todos os indivíduos do contrato executam a ação relacionada. Por exemplo, um subcontrato $\mathcal{C}_1 = O(\alpha) \wedge {}_{i \frown j} F(\beta)$ de um contrato maior, onde o conjunto de indivíduos é $\mathcal{I} = \{i, j, k, l\}$, equivale a $\bigwedge_{i \in \mathcal{I}} {}_{i} O(\alpha) \wedge {}_{i \frown j} F(\beta)$. Neste caso, todos os indivíduos de $\mathcal{I}$ devem executar, na prática, a ação mesmo que no subcontrato não conste os indivíduos k e l.

Assim uma nova estratégia para melhorar a eficiência da ferramenta foi desenvolvida para as decomposições de contratos, relacionada agora ao conjunto de indivíduos. Ao longo da construção do autômato que representa o contrato, cada subcontrato é avaliado como um novo contrato em cada estado do autômato. Por isso, a avaliação em cada estado passou a gerar as ações relativizadas apenas com os indivíduos relacionados ao subcontrato do estado corrente. No caso do contrato $\mathcal{C}_1$, por exemplo, é possível decompor equivalências para os cenários onde todos os indivíduos são considerados, apenas com os indivíduos i e j.

5.3 Aplicação da ferramenta

A ferramenta RECALL foi desenvolvida com o objetivo de analisar contratos multilaterais expressos em $\mathcal{RCL}$. Tanto a ferramenta quanto seu código fonte estão disponíveis em <http://recallcontracts.github.io> sob licença GNU GPL [76].

A verificação de um contrato com suporte da ferramenta RECALL requer, primeiramente, sua especificação em $\mathcal{RCL}$. O arquivo de contrato é organizado em duas partes: conflitos predefinidos e cláusulas. A Figura 39 apresenta um exemplo genérico de contrato.

Os conflitos predefinidos devem ser declarados na seção `conflict` dentro das subseções `global` e `relativized` para ações globalmente conflitantes ou relativizados quando

```
conflict {
    global { (a, b), (c, d) };
    relativized { (e, f), (e, a) };
};
[e]({j,k}O(f) ^ P(a) ^ {k}[a.b]({i,j}O(e&f)));
{j,i}F(c) _/{j}O(d)/_ ^ P(b) ^ {i,k}[a]({k}[b]({j,i}P(h))));
```

Figura 39 – Exemplo de contrato em $\mathcal{RCL}$ no formato RECALL.

o conflito depende do executor e recebedor da ação, respectivamente. Ambos os conjuntos são opcionais e podem então ser omitidos. Em seguida, todas as cláusulas do contrato devem ser especificadas conforme a gramática da $\mathcal{RCL}$ separadas por ponto-e-virgula. A interpretação dessas cláusulas separadas por ponto e vírgula é de uma operação de conjunção entre todos os componentes.

A relativização dos operadores deônticos são definidos da seguinte forma:

1. Relativizados: quando apenas um indivíduo é especificado. e.g. `{i}O(a)`, onde o indivíduo i é obrigado a executar a ação a.

2. Direcionados: quando dois indivíduos são especificados. e.g. `{i,j}O(a)`, onde o indivíduo i é obrigado a executar a ação a para o indivíduo j.

3. Globais: quando a especificação dos indivíduos é omitida, e.g. `O(a)` onde todos os indivíduos do contrato são obrigados a executar a ação a.

As modalidades deônticas são representadas por `O`, `P`, e `F`, respectivamente, operadores de obrigação, permissão e proibição. As penalidades são descritas entre os delimitadores `_/` e `/_` após a cláusula associada. As ações e indivíduos podem ser definidos por qualquer sequência que comece com letras maiúsculas ou minúsculas seguidos de letras, números e *underscore* (`_`). A sintaxe da RECALL suporta comentários de linha, utilizando o prefixo `//` e comentários em bloco com várias linhas, iniciando com `/*` e terminando com `*/`. Os símbolos dos operadores sobre as ações e cláusulas da $\mathcal{RCL}$ são definidos na Tabela 3. Além disso, na Tabela 4, são listados alguns exemplos de construção de fórmulas em $\mathcal{RCL}$ e a respectiva especificação no formato $\mathcal{RCL}$ para a ferramenta RECALL.

A execução da ferramenta sobre o arquivo de especificação do contrato é realizada por linha de comando como:

```
$recall contract.rcl [options]
```

Símbolo $\mathcal{RCL}$	**Sintaxe RECALL**
Disjunção Exclusiva ($\oplus$)	`-`
Disjunção ($\wedge$)	`|`
Conjunção ($\vee$)	`^`
Ações concorrentes ($\times$)	`&`
Escolha de ações ($+$)	`+`
Sequência de ações ($\cdot$)	`.`
Sequência indefinida de ações (a^*)	`a*`
Negação de uma ação ($\overline{a}$)	`!a`

Tabela 3 – Símbolos da $\mathcal{RCL}$.

Descrição	Cláusula em $\mathcal{RCL}$	Cláusula na RECALL
Obrigação e Proibição relativizadas	${}_iO(a) \wedge \ {}_jF(a)$	`{i}O(a)^{j}F(a)`
Modalidade dinâmica direcionada	${}_{i \frown j}[a]_{i \frown j}O(b)$	`{i,j}[a]({i,j}O(b))`
Obrigação com penalidade	${}_{j \frown k}O(a)_{i \frown j F(a)}$	`{j,k}O(a)_/{i,j}F(a)/_`

Tabela 4 – Exemplos de cláusulas em $\mathcal{RCL}$ para RECALL.

O arquivo do contrato `contract.rcl` é passado como parâmetro, assim como as opções de execução conforme a Tabela 5.

Parâmetro	**Descrição**
`-c`	Detecta todos os conflitos construindo o autômato completamente.
`-g <filename>`	Exporta o autômato para um arquivo específico em formato DOT. Caso o nome do arquivo seja omitido, o arquivo DOT é exportado com o nome do arquivo do contrato.
`-v`	Exibe informações adicionais de análise no arquivo de *log* e na saída padrão.
`-h`	Exibe a lista de parâmetros válidos na ferramenta, uma breve descrição e exemplos de aplicação.

Tabela 5 – Parâmetros de linha de comando da ferramenta.

Após a execução da ferramenta sobre um contrato, o veredito pode retornar que o contrato está livre de conflitos, conforme a Figura 40, ou que ao menos um conflito foi detectado, conforme a Figura 41. Na ocorrência de um conflito é também retornado o estado em que ocorreu, o tipo de conflito e o *trace* que leva o contrato ao conflito.

```
./recall example.rcl -g -c
Analysing contract in example.rcl
CONFLICT-FREE: The analyzed contract is conflict-free.
```

Figura 40 – Execução de um contrato livre de conflitos.

```
./recall example1.rcl -g -c
Analysing contract in example1.rcl
CONFLICT: A conflict was found in the contract.
Conflict found in state (s1)
Conflict: F(j,a,i) conflicts with O(j,a,i)
Trace: (s1)<-T2-(s0)
Stacktrace:
(s1) - {j,i}P(a) AND {j,i}O(a)
<T2> - (i, b, j), (j, a, i)
(s0) - {j,i}[a]({j,i}F(a)) AND {i,j}[b]({j,i}O(a))
```

Figura 41 – Execução de um contrato que possui conflitos.

O autômato obtido na análise é exportado no formato DOT e pode ser visualizado no *Graphiviz* [74]. O objetivo da exportação do autômato é facilitar a visualização do contrato e suas propriedades, além de possibilitar futuras integrações com outras ferramentas.

6 ESTUDO DE CASO

O estudo de caso apresentado neste trabalho aborda um modelo de negócio para venda de produtos via *Internet*, baseado no trabalho de Daskalopulu[15]. Daskalopulu propôs uma representação formal de contratos multilaterais através de autômatos e redes de Petri, para verificação de propriedades especificadas em CTL, tais como alcançabilidade e completude dos contratos [1]. Numa outra direção, o estudo de caso é, no presente trabalho, tratado através de modalidades deônticas e submetido a um método de detecção de conflitos em contratos dessa natureza.

O contrato analisado consiste, basicamente, num sistema de compra e venda de produtos onde potenciais fraudes podem ocorrer no processo. Os envolvidos no contrato, além do comprador e vendedor, são instituições financeiras que trabalham como mediadoras dos pagamentos entre os indivíduos, e transportadoras, responsáveis pelas entregas. A seguir são apresentados o cenário completo do contrato e as características que o tornam um contrato multilateral modelado em $\mathcal{RCL}$. Na sequência é realizada a verificação formal do contrato pela ferramenta RECALL e as ações corretivas sobre os potenciais conflitos encontrados.

6.1 Descrição do cenário

As transações eletrônicas são cada vez mais comuns nos dias atuais. Embora existam várias leis de defesa do consumidor, contratos mal formulados ainda podem existir permitindo fraudes ou mal-entendidos entre as partes envolvidas. O cenário se torna ainda mais complexo quando o modelo de negócios de um comércio eletrônico tem interdependências entre as várias partes envolvidas. Essas interdependências, em geral, são necessárias para expressar com precisão a complexidade de um contrato. Por exemplo, no caso de um contrato de venda via *Internet*, não é possível garantir que ambas as partes cumprirão o acordo necessário sobre o pagamento e entrega. O vendedor pode entregar um produto para o comprador antes que este tenha pago o valor que lhe cabe. Em seguida, o vendedor fica desprotegido e corre o risco de ser prejudicado, uma vez que o comprador pode não pagar o produto. Em contrapartida, se o comprador pagar o produto antes de recebê-lo, então o vendedor poderia não realizar a entrega, fato que também viola o acordo. Em ambos os casos uma das partes poderia ficar desprotegida levando a uma quebra de contrato, seja por falta de pagamento ou pelo não recebimento das mercadorias.

Com o objetivo de evitar possíveis violações contratuais, várias empresas de comércio eletrônico utilizam uma instituição financeira para intermediar suas transações. No estudo de caso realizado, todas as transações financeiras são intermediadas por uma

instituição bancária. Dessa forma, o comprador realiza o pagamento através de um banco. Após a entrega do produto, o respectivo valor é transferido para o vendedor. Essa prática evita que o comprador seja lesado, uma vez que o banco pode reembolsá-lo caso o produto não seja entregue.

O contrato deste estudo de caso é apresentado em alto nível na Figura 42, onde são descritas as transações que expressam acordos entre as partes envolvidas no processo de compra e venda. A transação é, basicamente, composta por um comprador que realiza a aquisição de produtos de algum vendedor em específico. Uma transportadora deve então entregar o produto ao comprador, enquanto o banco intermedeia as transações financeiras. Além do acordo entre as partes também são consideradas as regras internas dos envolvidos. Neste cenário, a transportadora define uma regra interna onde um produto só deve ser entregue se o frete já tenha sido pago pelo vendedor. Outra regra interna é difundida pelo banco que, para evitar fraudes, proíbe a realização dos pagamentos caso não receba as devidas notificações.

A interação entre as partes envolvidas do contrato são representadas na Figura 43. Note que o grafo não expressa completamente o contrato do estudo de caso uma vez que as modalidades não podem ser representadas com esse modelo. O significado pretendido é representar graficamente os indivíduos, suas relações e ações associadas, a fim de facilitar a compreensão. Com base nessa representação, é possível mostrar que o estudo de caso é de fato caracterizado por um contrato multilateral. Os nós representam os indivíduos e as arestas direcionadas indicam as ações executadas pelo indivíduo do nó de origem para o indivíduo representado pelo nó de destino. Duas ou mais ações na aresta, separados por vírgula, significam que os indivíduos estão relacionados por ações distintas. Todos os indivíduos são referidos por símbolos que os representam no contrato conforme a Tabela 6. Da mesma forma, as ações relacionadas ao contrato são apresentados na Tabela 7.

Individuo	**Símbolo**
buyer (Comprador)	b
seller (Vendedor)	s
bank (Banco)	k
carrier (Transportadora)	c

Tabela 6 – Lista de indivíduos.

O contrato apresentado não pode ser dividido em contratos bilaterais sem que informações relevantes sobre o modelo de negócio sejam perdidas. Essencialmente, um contrato multilateral se distingue de um bilateral uma vez que este último não é capaz de modelar inter-relações entre vários indivíduos simultaneamente. Portanto, contratos bilaterais podem modelar relacionamentos somente entre dois ou mais indivíduos quando estes estão numa relação encadeada [77]. Para comprovar essa afirmação, um subconjunto das relações do contrato na Figura 43 é modelado por contratos bilaterais na Figura 44.

[**Contrato de compra e venda**]

1. O **Comprador** realiza a compra de um produto do **Vendedor**.
2. O **Comprador** é obrigado a pagar o valor do produto para o **Banco**.
3. O **Banco** deve enviar uma notificação sobre o pagamento do produto para o **Vendedor**.
4. Após o **Banco** notificar o pagamento ao **Vendedor**, este é obrigado a enviar o produto pela **Transportadora** e pagar o valor referente ao frete desse envio ao **Banco**.
5. A **Transportadora** deve entregar o produto ao **Comprador**.
6. Após a entrega do produto, o **Comprador** é obrigado a informar o **Banco** sobre o recebimento do produto e a **Transportadora** deve notificar o **Vendedor** que o produto foi entregue ao **Comprador**.
7. Quando o **Vendedor** é notificado sobre a entrega deve então notificar o **Banco** para liberar o pagamento do valor do frete à **Transportadora**.
8. Quando o **Comprador** notifica o **Banco** sobre o recebimento do produto foi recebido, o **Banco** realiza o pagamento do valor do produto ao **Vendedor**.
9. O **Banco** deve pagar o valor do frete à **Transportadora** somente após o **Vendedor** realizar o pagamento do valor especificado.

[**Regras internas do Banco**]

10. O **Banco** é proibido de pagar o **Vendedor** enquanto não receber uma notificação do **Comprador** confirmando o recebimento do produto.
11. O **Banco** é proibido de liberar o pagamento do frete para a **Transportadora** até que o **Vendedor** notifique o **Banco**.

[**Regras internas da Transportadora**]

12. A **Transportadora** é proibida de entregar o produto até que o **Vendedor** tenha pago o frete.

Figura 42 – Contrato de venda em alto nível.

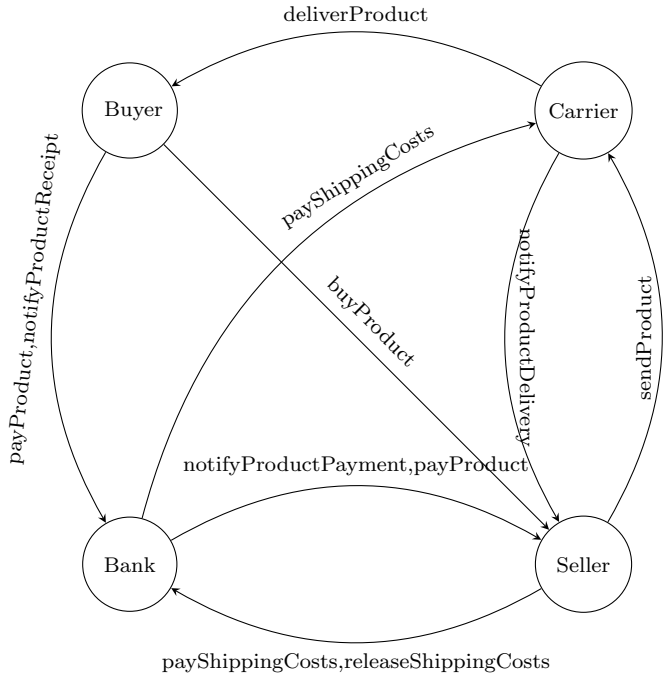

Figura 43 – Grafo representando os relacionamentos do contrato de compra e venda.

ação	**descrição**
buyProduct	Comprar o produto
payProduct	Pagar o valor do produto
notifyProductPayment	Notificar o pagamento do produto
sendProduct	Enviar o produto
deliverProduct	Entregar o produto
notifyProductReceipt	Notificar o recebimento do produto
notifyProductDelivery	Notificar a entrega do produto
payShippingCosts	Pagar o frete
releaseShippingCosts	Liberar o pagamento do frete

Tabela 7 – Lista de ações do contrato.

Note que o contrato da Figura 44 modela a relação entre o banco e o vendedor. Da mesma forma, as relações entre o comprador, o banco e a transportadora estão na Figura 45. Os contratos bilaterais não compartilham informações entre si e também não são capazes de assegurar a ordem de execução de uma ação em relação a outras ações presentes em contratos bilaterais distintos. Por isso, quando o contrato de vendas é modelado por um conjunto de contratos bilaterais, indivíduos de subcontratos distintos não estão diretamente relacionados. Sendo assim, a expressividade do modelo de negócio não pode ser capturada de forma adequada. Fica claro que a transportadora não tem a obrigação de notificar o vendedor sobre a entrega do produto. Portanto, não se pode garantir que o vendedor irá notificar o banco para liberar o valor do frete para a transportadora. Logo, a transportadora pode ser prejudicada. Essas situações tornam os contratos instáveis, pois todas as relações e condições previstas podem não ser corretamente modeladas.

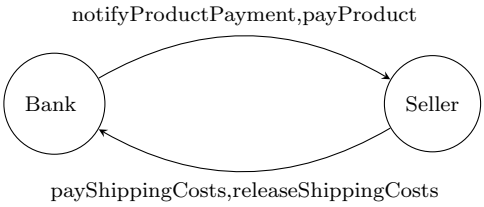

Figura 44 – Relacionamento bilateral entre Banco e Vendedor.

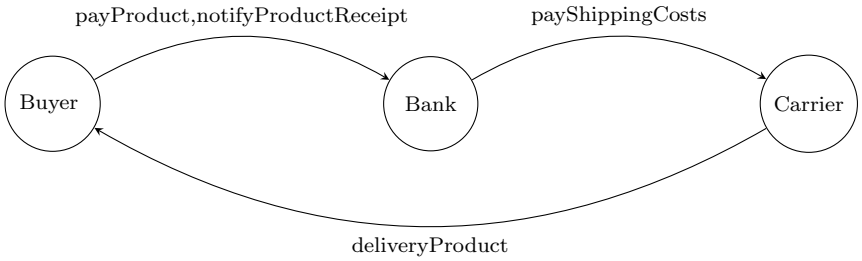

Figura 45 – Relacionamento bilateral entre Comprador, Banco e Transportadora.

6.2 Verificação do contrato

O estudo de caso foi especificado na linguagem $\mathcal{RCL}$ para que possa ser verificado automaticamente. A especificação do contrato é descrita na Figura 46. Na linha (19), por exemplo, é descrita a especificação $\mathcal{RCL}$ para a cláusula descrita na linha (10) da Figura 42 na Seção 6.1. O contrato foi então submetido a ferramenta RECALL. Ao final do processo, a ferramenta retornou a análise apresentada na Figura 47.

```
1  {b,s}[buyProduct](
2   {b,k}O(payProduct)^
3   {b,k}[payProduct](
4    {k,s}O(notifyProductPayment)^
5    {k,s}[notifyProductPayment](
6     {s,c}O(sendProduct)^
7     {s,k}O(payShippingCosts)^
8     {s,k}[payShippingCosts](
9      {s,c}[sendProduct](
10      {c,b}O(deliverProduct) ^
11      {c,b}[deliverProduct](
12       {b, k}O(notifyProductReceipt) ^
13       {c, s}O(notifyProductDelivery) ^
14       {b,k}[notifyProductReceipt]({k,c}O(payProduct)) ^
15       {c, s}[notifyProductDelivery](
16        {s, k}O(liberateShippingCosts) ^
17        {s, k}[liberateShippingCosts]({k,c}O(payShippingCosts))
18 )))))));
19 {b,k}[(!notifyDelivery)*]({k,s}F(payProduct));
20 {s,k}[(!liberateShippingCosts)*]({k,c}F(payShippingCosts));
21 {s,c}[(!payShippingCosts)*]({c,b}F(deliverProduct));
```

Figura 46 – A especificação em $\mathcal{RCL}$ do contrato de compra e venda (Parcial).

O veredito sobre a ocorrência de conflitos no contrato é apresentado na linha (2) da Figura 47. Caso um conflito não seja encontrado num determinado contrato, a mensagem indicaria que *"o contrato é livre de conflitos"*. As linhas (4) e (5) descrevem maiores

detalhes sobre um conflito detectado. Um *trace* do autômato é também mostrado na linha (7) como uma sequência de estados e transições que leva o contrato para um estado de conflito. Outros detalhes também são fornecidos pela ferramenta, como todas as decomposições associadas a cada estado do autômato e as ações realizadas ao longo das transições presentes no *trace*. Tais informações estão disponíveis na íntegra na Figura 51 do Apêndice A.

```
1  Analysing contract in study-conflicting.rcl
2  [CONFLICT] A conflict was found in the analyzed contract.
3  ------------------------------------------------------------
4  Conflict found in state (s23)
5  Conflict: F(c,deliverProduct,b) conflicts with [O(c,deliverProduct,b)]
6  ------------------------------------------------------------
7  Trace: (s23)<--T223--(s22)<--T224--(s21)<--T225--(s20)<--T226--(s19)<--T227--(s0)
```

Figura 47 – A saída da ferramenta RECALL após a análise (Parcial).

A informação presente na linha (4) da Figura 47 mostra as cláusulas que estão em conflito. A primeira cláusula especifica que a transportadora é proibida de entregar o produto para o comprador. A segunda cláusula diz que a transportadora é obrigada a entregar o produto para o comprador. Logo, existe um impasse entre as regras. Na sequência o *trace* completo fornece todas as ações executadas e qual sequência realizada no contrato que o conduziu a situação conflitante. Observe que, após o processamento e decomposição do contrato no estado inicial (s_0), de acordo com a ação na transição (t_{227}), é obtido o estado (s_{19}). Da mesma forma, decomposições sucessivas são realizadas até atingir o estado em conflito (s_{23}).

O autômato correspondente a análise realizada é fornecido pela ferramenta, conforme é mostrado parcialmente na Figura 48. Cada estado obtido no autômato contém sua respectiva decomposição do contrato e uma transição para este estado representa o conjunto de ações relativizadas que foi executado para se obter tal decomposição. Os estados não preenchidos representam decomposições intermediárias obtidas após avaliar suas respectivas cláusulas. Um estado especial é utilizado para representar uma violação do contrato quando a fórmula avaliada retorna *false*. Outro estado especial indica a satisfação do contrato, onde a fórmula avaliada é reduzida a *true*. Finalmente, o último estado especial, destacado em cinza na Figura 48, indica uma decomposição conflitante, como visto na linha (5) da Figura 47.

O autômato de saída tem o número de transições bastante reduzido devido às estratégias de otimização, como a ordenação das ações concorrentes a serem avaliadas e a poda de ações desnecessárias, apresentados na Subseção 5.2.2. Essa redução, contudo, mantém a expressividade do autômato com relação as decomposições avaliadas, uma vez que o número de estados permanece o mesmo. A representação gráfica completa do autômato obtido do estudo de caso é apresentada na Figura 52 do Apêndice A.

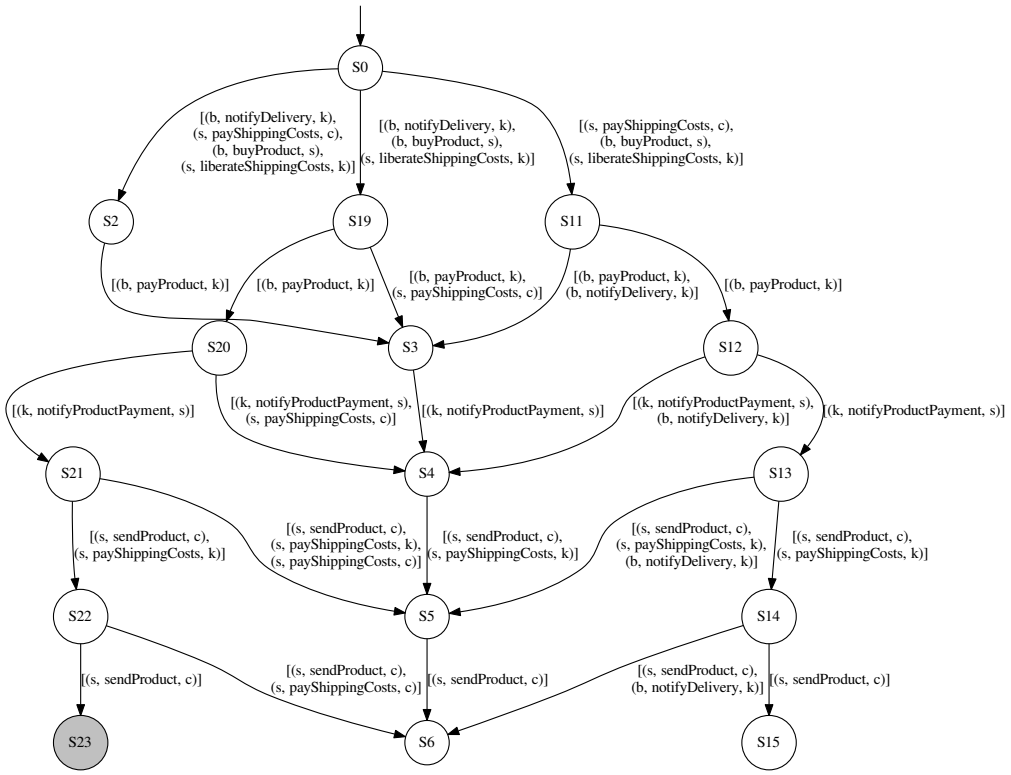

Figura 48 – Autômato parcial da análise sobre o estudo de caso.

6.3 Contrato revisado

De acordo com as informações fornecidas na análise retornada pela ferramenta RECALL, é possível verificar que a regra interna da transportadora está em conflito com o restante do contrato (Veja linha (21) da Figura 46). Esta regra interna estabelece que a transportadora pode enviar o produto somente após o pagamento do frete. Por outro lado, no contrato original, espera-se que o produto seja entregue pela transportadora ao comprador antes que o pagamento do frete seja liberado pelo banco. Portanto uma incompatibilidade ocorre na regra interna da transportadora, onde o pagamento do frete realizado pelo vendedor era esperado pela transportadora antes que o produto fosse enviado ao comprador. Já a transportadora considera que, em sua regra interna, o pagamento do frete seja realizado pelo vendedor, ao invés do banco, para então entregar o produto.

O resultado da análise de verificação permite que o conflito existente seja resolvido mudando algumas cláusulas no contrato original. Em primeiro lugar, uma nova cláusula é incluída indicando que o banco agora deve notificar a transportadora sobre o pagamento do frete pelo vendedor. Outra alteração no contrato foi realizada especificamente na regra interna da transportadora. A transportadora, agora, deve levar em conta a notificação do banco como garantia de pagamento do frete.

Dessa forma, as especificações do contrato são reescritas seguindo as alterações propostas a fim de evitar a situação de conflito. As cláusulas (5) e (12) são alteradas na descrição do contrato conforme a Figura 49.

[Contrato de compra e venda]

5. O **Banco** deve notificar a **Transportadora** sobre o pagamento do frete e após o **Banco** atestar o pagamento, a **Transportadora** é obrigada a entregar o produto para o **Comprador**.

[Regras internas da Transportadora]

12. A **Transportadora** é proibida de entregar o produto até que o **Banco** notifique-a de que o **Vendedor** pagou o valor do frete.

* Note que as demais cláusulas do contrato original permanecem inalteradas.

Figura 49 – Alteração do contrato conforme os ajustes propostos.

O novo contrato em $\mathcal{RCL}$ com as alterações propostas é apresentado na Figura 50. A cláusula específica em que o banco deve notificar a transportadora sobre o pagamento do frete é descrita na linha (9). A transportadora então considera a notificação do banco como garantia para liberar a entrega do produto, como especificado na linha (21). As linhas (1)-(8) e (10)-(20) do contrato original permanecem inalteradas.

```
1  {b,s}[buyProduct](
2   {b,k}O(payProduct)^
3   {b,k}[payProduct](
4    {k,s}O(notifyProductPayment)^
5    {k,s}[notifyProductPayment](
6     {s,c}O(sendProduct)^
7     {s,k}O(payShippingCosts)^
8     {s,k}[payShippingCosts](
9      {k,c}O(notifyShippingPayment) ^ {s,c}[sendProduct](
10      {c,b}O(deliverProduct) ^
11      {c,b}[deliverProduct](
12       {b, k}O(notifyProductReceipt) ^
13       {c, s}O(notifyProductDelivery) ^
14       {b,k}[notifyProductReceipt]({k,c}O(payProduct)) ^
15       {c, s}[notifyProductDelivery](
16        {s, k}O(liberateShippingCosts) ^
17        {s, k}[liberateShippingCosts]({k,c}O(payShippingCosts))
18 )))))));
19 {b,k}[(!notifyDelivery)*]({k,s}F(payProduct));
20 {s,k}[(!liberateShippingCosts)*]({k,c}F(payShippingCosts));
21 {k,c}[(!notifyShippingPayment)*]({c,b}F(deliverProduct));
```

Figura 50 – Versão corrigida do contrato de vendas.

Após as alterações, a nova versão do contrato é submetida novamente a ferramenta RECALL. Desta vez, a análise resultante indicou um contrato livre de conflitos. Após esses ajustes, uma vez que o banco agora é obrigado a notificar a transportadora sobre o

pagamento do frete, o contrato é executado com garantia de segurança, sem a possibilidade de perdas ou danos para algumas das partes.

Por uma questão de completude em relação à divisão de contratos multilaterais em bilaterais, é importante observar que mesmo a versão mais recente do contrato de vendas não pode ser capturado por um conjunto de contratos bilaterais conforme argumentos já apresentados, na Seção 6.1, sobre sua versão anterior.

7 CONCLUSÃO

Este trabalho apresentou um método de análise de contratos multilaterais, onde conflitos ou propriedades importantes podem ser verificadas. A abordagem é baseada na extensão de dois aspectos importantes de um processo de verificação de contratos multilaterais: a representação dos contratos; e a detecção de conflitos nestes contratos.

A linguagem de representação proposta para os contratos multilaterais foi derivada da linguagem de contratos $\mathcal{CL}$, proposta por Prisacariu e Schneider[53], adotando os conceitos sobre relativização proposta por Herrestad e Krogh[7]. Essa linguagem estendida, denominada $\mathcal{RCL}$, permite que obrigações, proibições e permissões de ações atômicas ou compostas por operadores de escolha, concorrência ou sequência, sejam expressas. Os pré-requisitos de uma cláusula também podem ser expressos em $\mathcal{RCL}$, especificando a noção de gatilhos num operador dinâmico. A relativização desses operadores determinam o envolvimento dos indivíduos relacionados com as cláusulas, indicando o responsável pela execução de uma ação e o indivíduo beneficiado por essa ação.

O processo de detecção de conflitos foi estendido com base na proposta de Fenech, Pace e Schneider[14]. A extensão incorporou a avaliação de conflitos sobre operadores deônticos e dinâmicos relativizados. O método proposto foi implementado dando origem a ferramenta RECALL. Essa ferramenta é capaz de analisar automaticamente contratos escritos em $\mathcal{RCL}$ e retornar informações importantes de análise sobre potenciais conflitos presentes em suas cláusulas.

Neste trabalho um estudo de caso foi ainda aplicado a ferramenta RECALL: um contrato multilateral de compra e venda de produtos pela *Internet*. O contrato de compra e venda de produtos foi especificado em $\mathcal{RCL}$ e verificado pela ferramenta RECALL. A ferramenta detectou um conflito importante entre as cláusulas do contrato que pôde então ser corrigido com base nas informações retornadas pela ferramenta. Com a aplicação do estudo de caso à ferramenta foi possível também obter uma prova de conceito do método proposto, bem como das funcionalidades da ferramenta desenvolvida.

Como trabalho futuro pretende-se estender a $\mathcal{RCL}$ de forma que seja possível expressar grupos de indivíduos específicos em cláusulas ou penalidades. O agrupamento de indivíduos e sua relação com os operadores deônticos e dinâmicos deve ser bem definido, tanto na sintaxe quando na semântica da $\mathcal{RCL}$, para que as propriedades da linguagem sejam propagadas corretamente na extensão. Numa outra direção, pretende-se aplicar os autômatos resultantes da análise realizada pela ferramenta no monitoramento de contratos [31], ou na negociação automática de contratos por meio do produto de autômatos [78]. Por fim, espera-se que melhorias sejam realizadas na ferramenta RECALL, principalmente

relacionadas a interação com o usuário para facilitar a visualização das análises através do autômato e dos *traces* resultantes, bem como a inclusão de um editor gráfico de contratos multilaterais em $\mathcal{RCL}$.

REFERÊNCIAS

[1] HUTH, M.; RYAN, M. *Logic in Computer Science*. [S.l.]: Cambridge University Press, 2004. ISBN 9780521543101.

[2] FISCHER, M. J.; LADNER, R. E. Propositional dynamic logic of regular programs. *Journal of Computer and System Sciences*, v. 18, n. 2, p. 194 – 211, 1979. ISSN 0022-0000. Disponível em: <http://www.sciencedirect.com/science/article/pii/0022000079900461>.

[3] XU, L. A multi-party contract model. *ACM SIGecom Exchanges*, v. 5, p. 10, 2004.

[4] MIRANDA, M. B. Teoria geral dos contratos. In: *Revista Virtual Direito Brasil*. [S.l.: s.n.], 2008. v. 2, n. 2, p. 15.

[5] FENECH, S. *Conflict Analysis of Deontic Contracts*. mestrado — Department of Computer Science and Artificial Intelligence - University of Malta, março 2008.

[6] LEE, R. M. A logic model for electronic contracting. *Decision Support Systems*, n. 4, p. 27–44, March 1988.

[7] HERRESTAD, H.; KROGH, C. Deontic logic relativised to bearers and counterparties. In: *Proceedings of the Anniversary of Anthology in Computers and Law*. [S.l.]: Tano A.S., 1995. p. 453–522.

[8] PRISACARIU, C.; SCHNEIDER, G. A dynamic deontic logic for complex contracts. *The Journal of Logic and Algebraic Programming*, v. 81, n. 4, p. 458 – 490, 2012. ISSN 1567-8326. Disponível em: <http://www.sciencedirect.com/science/article/pii/S1567832612000197>.

[9] KOETSIER, M.; GREFEN, P.; VONK, J. Contracts for cross-organizational workflow management. In: BAUKNECHT, K.; MADRIA, S.; PERNUL, G. (Ed.). *Electronic Commerce and Web Technologies*. Springer Berlin Heidelberg, 2000, (Lecture Notes in Computer Science, v. 1875). p. 110–121. ISBN 978-3-540-67981-3. Disponível em: <http://dx.doi.org/10.1007/3-540-44463-7_10>.

[10] AALST, W. M. P. van der et al. Multiparty contracts: Agreeing and implementing interorganizational processes. *Comput. J.*, v. 53, n. 1, p. 90–106, 2010.

[11] AUSÍN, T. *Entre la Lógica y el Derecho – Paradojas y conflictos normativos*. Barcelona, 2005. Disponível em: <http://books.google.com.br/books?id=5WD2u1u9qjUC>.

[12] von Wright, G. H. Deontic logic. In: *Mind*. [S.l.]: Oxford University Press, 1951, (A quarterly review of psychology and philosophy, 237). p. 1–15.

[13] XU, L. *Monitoring Multi-party contracts for e-business*. Tese (Doutorado) — Faculty of Economics and Business Administration of Tilburg University, 2004.

[14] FENECH, S.; PACE, G.; SCHNEIDER, G. Automatic conflict detection on contracts. In: LEUCKER, M.; MORGAN, C. (Ed.). *Theoretical Aspects of Computing - ICTAC 2009*. Springer Berlin Heidelberg, 2009, (Lecture Notes in Computer Science, v. 5684). p. 200–214. ISBN 978-3-642-03465-7. Disponível em: <http://dx.doi.org/10.1007/978-3-642-03466-4_13>.

[15] DASKALOPULU, A. Model checking contractual protocols. *CoRR*, cs.SE/0106009, 2001. Disponível em: <http://arxiv.org/abs/cs.SE/0106009>.

[16] SALOMO, J. *Contratos de prestação de serviços: manual teórico e prático*. Juarez de Oliveira, 2005. ISBN 9788574535302. Disponível em: <http://books.google.com.br/books?id=hvGdc5PXIUYC>.

[17] SOUZA, M. C. J. d. *Contratos Eletrônicos*. MAUAD, 1997. (Série Jurídica, 2). Disponível em: <http://books.google.com.br/books?id=r7ptQFFK4V4C&printsec=frontcover&hl=pt-BR>.

[18] BOBBIO, N. *Teoria do Ordenamento Jurídico*. 1ª edição. ed. Brasília: Editora UnB, 2006.

[19] OCHOA, C. H. *Conflictos normativos*. México: Universidad Nacional Autónoma de México, 2003.

[20] LIMA, S. P. T. S.; SANZ, W. C. Análise lógico-deôntica de conflitos normativos no âmbito jurídico. Sociedade Brasileira para o Progresso da Ciência, 2011. Disponível em: <http://www.sbpcnet.org.br/livro/63ra/conpeex/pivic/trabalhos/SAULO_PA.PDF>.

[21] ANGELOV, S.; GREFEN, P. Requirements on a b2b e-contract language. In: *BETA publicatie*. [S.l.]: Eindhoven : Technische Universiteit Eindhoven, 2005, (BETA publicatie : working papers, 140). p. 46.

[22] XU, L.; JEUSFELD, M. A. A concept for monitoring of electronic contracts. In: *In The 2003 IEEE Conference on E-Commerce (CEC'03). IEEE Computer*. [S.l.]: Society Press, 2003. p. 584–600.

[23] UDUPI, Y. B.; SINGH, M. P. Contract enactment in virtual organizations: A commitment-based approach. In: *Proceedings of the 21st National Conference on Artificial Intelligence - Volume 1*. AAAI Press, 2006. (AAAI'06), p. 722–727. ISBN 978-1-57735-281-5. Disponível em: <http://dl.acm.org/citation.cfm?id=1597538.1597654>.

[24] PACE, G.; PRISACARIU, C.; SCHNEIDER, G. Model checking contracts – a case study. In: NAMJOSHI, K. et al. (Ed.). *Automated Technology for Verification and Analysis*. Springer Berlin Heidelberg, 2007, (Lecture Notes in Computer Science, v. 4762). p. 82–97. ISBN 978-3-540-75595-1. Disponível em: <http://dx.doi.org/10.1007/978-3-540-75596-8_8>.

[25] ANGELOV, S.; GREFEN, P. *B2B eContract Handling - A Survey of Projects, Papers and Standards*. [S.l.], 2001.

[26] BOSSE, T. et al. Automated formal analysis of human multi-issue negotiation processes. *Multiagent Grid Syst.*, IOS Press, Amsterdam, The Netherlands, The Netherlands, v. 4, n. 2, p. 213–233, abr. 2008. ISSN 1574-1702. Disponível em: <http://dl.acm.org/citation.cfm?id=1402618.1402622>.

[27] ANGELOV, K.; CAMILLERI, J. J.; SCHNEIDER, G. A framework for conflict analysis of normative texts written in controlled natural language. In: *The Journal of Logic and Algebraic Programming*. [S.l.: s.n.], 2013. p. 45.

[28] CHADHA, R.; KRAMER, S.; SCEDROV, A. Formal analysis of multi-party contract signing. In: *Computer Security Foundations Workshop, 2004. Proceedings. 17th IEEE*. [S.l.: s.n.], 2004. p. 266–279. ISSN 1063-6900.

[29] KORDY, B.; RADOMIROVIC, S. Constructing optimistic multi-party contract signing protocols. In: *Computer Security Foundations Symposium (CSF), 2012 IEEE 25th*. [S.l.: s.n.], 2012. p. 215–229. ISSN 1940-1434.

[30] GARAY, J. A.; MACKENZIE, P. D. Abuse-free multi-party contract signing. In: *Proceedings of the 13th International Symposium on Distributed Computing*. London, UK, UK: Springer-Verlag, 1999. p. 151–165. ISBN 3-540-66531-5. Disponível em: <http://dl.acm.org/citation.cfm?id=645956.675948>.

[31] KYAS, M.; PRISACARIU, C.; SCHNEIDER, G. Runtime monitoring of electronic contracts. In: *In ATVA08, LNCS*. [S.l.]: Springer-Verlag, 2008.

[32] HVITVED, T.; KLAEDTKE, F.; ZăLINESCU, E. A trace-based model for multiparty contracts. *The Journal of Logic and Algebraic Programming*, v. 81, n. 2, p. 72 – 98, 2012. ISSN 1567-8326. Disponível em: <http://www.sciencedirect.com/science/article/pii/S156783261100021X>.

[33] CHELLAS, B. F. *Modal Logic: an introduction*. [S.l.]: Cambridge University Press, 1980.

[34] MEYER, J.-J. C. Epistemic logic. In: GOBLE, L. (Ed.). *The blackwell guide to philosophical logic*. [S.l.]: Blackwell, 2001. cap. 9.

[35] PNUELI, A. The temporal logic of programs. In: *FOCS*. [S.l.]: IEEE Computer Society, 1977. p. 46–57.

[36] BLACKBURN, P.; BENTHEM, J. van; WOLTER, F. *Handbook of Modal Logic*. [S.l.]: Elsevier Science, 2006. (Studies in Logic and Practical Reasoning). ISBN 9780080466668.

[37] KRIPKE, S. A. A completeness theorem in modal logic. *Journal of Symbolic Logic*, Association for Symbolic Logic, v. 24, n. 1, p. 1–14, 03 1959. Disponível em: <http://projecteuclid.org/euclid.jsl/1183733464>.

[38] CLARKE, E. M.; JR, O. G.; PELED, D. A. *Model Checking*. London, England: The MIT Press, 2000. 314 p.

[39] MULLER-OLM, M.; SCHMIDT, D.; STEFFEN, B. Model-Checking: A Tutorial Introduction. p. 330–354, 1999.

[40] GUERRA, P. T.; WASSERMANN, R. Revision of ctl models. In: KURI-MORALES, A.; SIMARI, G. (Ed.). *Advances in Artificial Intelligence – IBERAMIA 2010*. Springer Berlin Heidelberg, 2010, (Lecture Notes in Computer Science, v. 6433). p. 153–162. ISBN 978-3-642-16951-9. Disponível em: <http://dx.doi.org/10.1007/978-3-642-16952-6_16>.

[41] KATOEN, J. *Concepts, Algorithms, and Tools for Model Checking*. Inst. für Mathematische Maschinen und Datenverarbeitung, 1999. (Arbeitsberichte des Instituts für Mathematische Maschinen und Datenverarbeitung). Disponível em: <http://books.google.com.br/books?id=ka7AtgAACAAJ>.

[42] BAIER, C.; KATOEN, J.-p. *Principles of Model Checking*. first. London, England: The MIT Press, 2008. 994 p. ISBN 9780262026499.

[43] PRATT, V. R. Semantical considerations on floyd-hoare logic. In: *FOCS*. [S.l.]: IEEE Computer Society, 1976. p. 109–121.

[44] HAREL, D.; KOZEN, D.; TIURYN, J. Dynamic logic. In: *Handbook of Philosophical Logic*. [S.l.]: MIT Press, 1984. p. 497–604.

[45] KOZEN, D. A completeness theorem for kleene algebras and the algebra of regular events. *Information and Computation*, p. 366–390, 1994.

[46] BALBIANI, P. Propositional dynamic logic. In: ZALTA, E. N. (Ed.). *The Stanford Encyclopedia of Philosophy*. Winter 2008. [S.l.: s.n.], 2008.

[47] MEYER, J. J. C. A different approach to deontic logic: Deontic logic viewed as a variant of dynamic logic. *Notre Dame Journal of Formal Logic*, v. 29, n. 1, p. 109–136, 1987.

[48] GOMES, N. G. Um panorama da lógica deôntica. *Kriterion: Revista de Filosofia*, scielo, v. 49, p. 9 – 38, 00 2008. ISSN 0100-512X. Disponível em: <http://www.scielo.br/scielo.php?script=sci_arttext&pid=S0100-512X2008000100002&nrm=iso>.

[49] HILPINEN, R. Deontic logic. In: GOBLE, L. (Ed.). *The blackwell guide to philosophical logic*. [S.l.]: Blackwell, 2001. cap. 8.

[50] ROYAKKERS, L. *Extending Deontic Logic for the Formalisation of Legal Rules*. Springer, 1998. (Law and Philosophy Library). ISBN 9780792349822. Disponível em: <http://books.google.com.br/books?id=kUMC4ShT3lQC>.

[51] WIERINGA, R.; MEYER, J.-J. Applications of deontic logic in computer science: A concise overview. In: *Deontic Logic in Computer Science: Normative System Specification*. [S.l.]: John Wiley & Sons, 1993. p. 17–40.

[52] FENECH, S.; PACE, G. J.; SCHNEIDER, G. Clan: A tool for contract analysis and conflict discovery. In: LIU, Z.; RAVN, A. P. (Ed.). *7th International Symposium on Automated Technology for Verification and Analysis (ATVA'09)*. Macao, China: Springer, 2009. (Lecture Notes in Computer Science, v. 5799), p. 90–96. ISBN 978-3-642-04760-2.

[53] PRISACARIU, C.; SCHNEIDER, G. A formal language for electronic contracts. In: *In FMOODS'07, volume 4468 of LNCS*. [S.l.]: Springer, 2007. p. 174–189.

[54] MCNAMARA, P. Deontic logic: Challenges to standard deontic logics. 2010. The Stanford Encyclopedia of Philosophy. Disponível em: <http://plato.stanford.edu/entries/logic-deontic/>.

[55] PRAKKEN, H.; SERGOT, M. Contrary-to-duty obligations. *Studia Logica*, v. 57, p. 91–115, 1996.

[56] KYAS, M.; PRISACARIU, C.; SCHNEIDER, G. *Run-time Monitoring of Electronic Contracts–theoretical results*. [S.l.], 2008.

[57] MONTAZERI, S. M.; ROY, N.; SCHNEIDER, G. From Contracts in Structured English to CL Specifications. In: *5th International Workshop on Formal Languages and Analysis of Contract-Oriented Software (FLACOS'11)*. Málaga, Spain: [s.n.], 2011. (EPTCS, v. 68), p. 55–69. ISSN 2075-2180. Disponível em: <http://dx.doi.org/10.4204/EPTCS.68.6>.

[58] MARTINEZ, E. et al. A Model for Visual Specification of e-Contracts. In: *The 7th IEEE International Conference on Services Computing (IEEE SCC'10)*. Miami, USA: IEEE Computer Society, 2010. p. 1–8. ISBN 978-0-7695-4126-6. Disponível em: <http://dx.doi.org/10.1109/SCC.2010.32>.

[59] KROGH, C.; HERRESTAD, H. *Individuals Obligations*. 1994.

[60] TAN, Y.-h.; THOEN, W. A logical model of directed obligations and permissions to support electronic contracting in electronic commerce. *International Journal of Electronic Commerce*, v. 3, n. Formal aspects of digital commerce, p. 87 – 104, 1998.

[61] HERRESTAD, H.; KROGH, C. Obligations Directed from Bearers and to Counterparties. p. 210–218, 1995.

[62] AGOTNES, T. et al. On the logic of normative systems. *International Joint Conference on Artificial Intelligence*, 2007.

[63] HVITVED, T. A survey of formal languages for contracts. In: FLACOS'10 (Ed.). *Fourth Workshop on Formal Languages and Analysis of Contract–Oriented Software*. [s.n.], 2010. p. 29–32. Disponível em: <http://www.diku.dk/hjemmesider/ansatte/hvitved/publications/hvitved10flacosb.pdf>.

[64] PRISACARIU, C.; SCHNEIDER, G. *CL: A Logic for Reasoning about Legal Contracts - Semantics*. [S.l.], 2008.

[65] BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. *UML: guia do usuário*. CAMPUS - RJ, 2006. ISBN 9788535217841. Disponível em: <https://books.google.com.br/books?id=ddWqxcDKGF8C>.

[66] ORACLE. *Java SE Development Kit 8*. Acesso em: 01 de outubro de 2015. Disponível em: <http://www.oracle.com/technetwork/pt/java/javase/>.

[67] GAMMA, E. et al. *Design Patterns: Elements of Reusable Object-oriented Software*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1995. ISBN 0-201-63361-2.

[68] CASS, S. *The 2015 Top Ten Programming Languages*. Acesso em: 31 de janeiro de 2016. Disponível em: <http://spectrum.ieee.org/computing/software/the-2015-top-ten-programming-languages>.

[69] ORACLE. *Netbeans IDE 8.0.1*. Acesso em: 01 de outubro de 2015. Disponível em: <https://netbeans.org/>.

[70] PARR, T. *ANTLR 4.5*: Another tool for language recognition. Acesso em: 10 de dezembro de 2015. Disponível em: <http://www.antlr.org/>.

[71] The Apache Software Foundation. *Apache Commons CLI 1.3.1*. Acesso em: 15 de dezembro de 2015. Disponível em: <https://commons.apache.org/proper/commons-cli>.

[72] GOOGLE. *Guava 1.8*: Google core libraries for java. Acesso em: 10 de dezembro de 2015. Disponível em: <https://github.com/google/guava>.

[73] GANSNER, E. *Dot*: graph description language. Acesso em: 06 de janeiro de 2016. Disponível em: <http://www.graphviz.org/content/dot-language>.

[74] GANSNER, E. *Graphviz 2.38*: Graph visualization software. Acesso em: 06 de janeiro de 2016. Disponível em: <http://www.graphviz.org/>.

[75] GANSNER, E. R.; NORTH, S. C. An open graph visualization system and its applications to software engineering. *SOFTWARE - PRACTICE AND EXPERIENCE*, v. 30, n. 11, p. 1203–1233, 2000.

[76] Free Software Foundation. *GNU General Public License*. 2007. Disponível em: <https://www.gnu.org/licenses/gpl-3.0.html>.

[77] HAUGEN, B. Multi-party electronic business transactions. *http://www.supplychainlinks.com/MultiPartyBusinessTransactions.PDF*, 2002.

[78] FLOOD, M. D.; GOODENOUGH, O. R. Contract as automaton: The computational representation of financial agreements. In: *The Office of Financial Research*. Social Science Research Network, 2015. Disponível em: <http://papers.ssrn.com/sol3/papers.cfm?abstract_id=2648460>.

Apêndices

APÊNDICE A – RESULTADOS DO ESTUDO DE CASO

Os resultados do estudo de caso descrito no Capítulo 6 são apresentados agora de maneira completa e detalhada. O contrato submetido a ferramenta RECALL apresentou um conflito como pode ser observado na saída da ferramenta na Figura 51. O conflito encontrado no estado ($s_2$3) do autômato que representa o contrato especifica a proibição de entrega do produto, `{c,b}F(deliverProduct)`, e ao mesmo tempo a obrigação de se entregar esse produto, `{c,b}O(deliverProduct)`. Além das informações contidas no *trace* de saída, a ferramenta RECALL oferece uma representação visual do autômato do contrato. As técnicas de poda da ferramenta reduziram significativamente a quantidade de transições no autômato, e uma versão mais compacta, considerando apenas a primeira transição criada entre cada par de estados, pode ser visualizada na Figura 52. Note que todos os estados continuam presentes no autômato construído.

```
1  Analysing contract in study−conflicting.rcl
2  [CONFLICT] A conflict was found in the analyzed contract.
3  −−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
4  Conflict found in state (s23)
5  Conflict: F(c,deliverProduct,b) conflicts with [O(c,deliverProduct,b)]
6  −−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
7  Trace: (s23)<−−T223−−(s22)<−−T224−−(s21)<−−T225−−(s20)<−−T226−−(s19)<−−T227−−(s0)
8  −−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
9  Stacktrace:
10 (s23) − {s,c}[!payShippingCosts∗]({c,b}PROHIBITION(deliverProduct)_/F/_) AND {c,b}[
       deliverProduct]({c,s}[notifyProductDelivery]({s,k}[liberateShippingCosts]({k,c}OBLIGATION(
       payShippingCosts)_/F/_) AND {s,k}OBLIGATION(liberateShippingCosts)_/F/_) AND {b,k}[
       notifyProductReceipt]({k,c}OBLIGATION(payProduct)_/F/_) AND {c,s}OBLIGATION(
       notifyProductDelivery)_/F/_ AND {b,k}OBLIGATION(notifyProductReceipt)_/F/_) AND {c,b}
       OBLIGATION(deliverProduct)_/F/_
11 <T223> − [(s, sendProduct, c)]
12 (s22) − {s,c}[!payShippingCosts∗]({c,b}PROHIBITION(deliverProduct)_/F/_) AND {s,c}[sendProduct
       ]({c,b}[deliverProduct]({c,s}[notifyProductDelivery]({s,k}[liberateShippingCosts]({k,c}OBLIGATION(
       payShippingCosts)_/F/_) AND {s,k}OBLIGATION(liberateShippingCosts)_/F/_) AND {b,k}[
       notifyProductReceipt]({k,c}OBLIGATION(payProduct)_/F/_) AND {c,s}OBLIGATION(
       notifyProductDelivery)_/F/_ AND {b,k}OBLIGATION(notifyProductReceipt)_/F/_) AND {c,b}
       OBLIGATION(deliverProduct)_/F/_)
13 <T224> − [(s, sendProduct, c), (s, payShippingCosts, k)]
14 (s21) − {s,c}[!payShippingCosts∗]({c,b}PROHIBITION(deliverProduct)_/F/_) AND {s,k}[
       payShippingCosts]({s,c}[sendProduct]({c,b}[deliverProduct]({c,s}[notifyProductDelivery]({s,k}[
       liberateShippingCosts]({k,c}OBLIGATION(payShippingCosts)_/F/_) AND {s,k}OBLIGATION(
       liberateShippingCosts)_/F/_) AND {b,k}[notifyProductReceipt]({k,c}OBLIGATION(payProduct)_/
       F/_) AND {c,s}OBLIGATION(notifyProductDelivery)_/F/_ AND {b,k}OBLIGATION(
       notifyProductReceipt)_/F/_) AND {c,b}OBLIGATION(deliverProduct)_/F/_)) AND {s,k}
       OBLIGATION(payShippingCosts)_/F/_ AND {s,c}OBLIGATION(sendProduct)_/F/_
15 <T225> − [(k, notifyProductPayment, s)]
16 (s20) − {s,c}[!payShippingCosts∗]({c,b}PROHIBITION(deliverProduct)_/F/_) AND {k,s}[
       notifyProductPayment]({s,k}[payShippingCosts]({s,c}[sendProduct]({c,b}[deliverProduct]({c,s}[
       notifyProductDelivery]({s,k}[liberateShippingCosts]({k,c}OBLIGATION(payShippingCosts)_/F/_)
       AND {s,k}OBLIGATION(liberateShippingCosts)_/F/_) AND {b,k}[notifyProductReceipt]({k,c}
       OBLIGATION(payProduct)_/F/_) AND {c,s}OBLIGATION(notifyProductDelivery)_/F/_ AND {b,
       k}OBLIGATION(notifyProductReceipt)_/F/_) AND {c,b}OBLIGATION(deliverProduct)_/F/_))
       AND {s,k}OBLIGATION(payShippingCosts)_/F/_ AND {s,c}OBLIGATION(sendProduct)_/F/_)
       AND {k,s}OBLIGATION(notifyProductPayment)_/F/_
17 <T226> − [(b, payProduct, k)]
18 (s19) − {s,c}[!payShippingCosts∗]({c,b}PROHIBITION(deliverProduct)_/F/_) AND {b,k}[payProduct
       ]({k,s}[notifyProductPayment]({s,k}[payShippingCosts]({s,c}[sendProduct]({c,b}[deliverProduct]({c,s
       }[notifyProductDelivery]({s,k}[liberateShippingCosts]({k,c}OBLIGATION(payShippingCosts)_/F/_)
       AND {s,k}OBLIGATION(liberateShippingCosts)_/F/_) AND {b,k}[notifyProductReceipt]({k,c}
       OBLIGATION(payProduct)_/F/_) AND {c,s}OBLIGATION(notifyProductDelivery)_/F/_ AND {b,
       k}OBLIGATION(notifyProductReceipt)_/F/_) AND {c,b}OBLIGATION(deliverProduct)_/F/_))
       AND {s,k}OBLIGATION(payShippingCosts)_/F/_ AND {s,c}OBLIGATION(sendProduct)_/F/_)
       AND {k,s}OBLIGATION(notifyProductPayment)_/F/_) AND {b,k}OBLIGATION(payProduct)_/F
       /_
19 <T227> − [(b, notifyDelivery, k), (b, buyProduct, s), (s, liberateShippingCosts, k)]
20 (s0) − {b,k}[!notifyDelivery∗]({k,s}PROHIBITION(payProduct)_/F/_) AND {s,k}[!
       liberateShippingCosts∗]({k,c}PROHIBITION(payShippingCosts)_/F/_) AND {s,c}[!
       payShippingCosts∗]({c,b}PROHIBITION(deliverProduct)_/F/_) AND {b,s}[buyProduct]({b,k}[
       payProduct]({k,s}[notifyProductPayment]({s,k}[payShippingCosts]({s,c}[sendProduct]({c,b}[
       deliverProduct]({c,s}[notifyProductDelivery]({s,k}[liberateShippingCosts]({k,c}OBLIGATION(
       payShippingCosts)_/F/_) AND {s,k}OBLIGATION(liberateShippingCosts)_/F/_) AND {b,k}[
       notifyProductReceipt]({k,c}OBLIGATION(payProduct)_/F/_) AND {c,s}OBLIGATION(
       notifyProductDelivery)_/F/_ AND {b,k}OBLIGATION(notifyProductReceipt)_/F/_) AND {c,b}
       OBLIGATION(deliverProduct)_/F/_)) AND {s,k}OBLIGATION(payShippingCosts)_/F/_ AND {s,
       c}OBLIGATION(sendProduct)_/F/_) AND {k,s}OBLIGATION(notifyProductPayment)_/F/_)
       AND {b,k}OBLIGATION(payProduct)_/F/_)
21 −−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
```

Figura 51 – *Trace* de análise do estudo de caso.

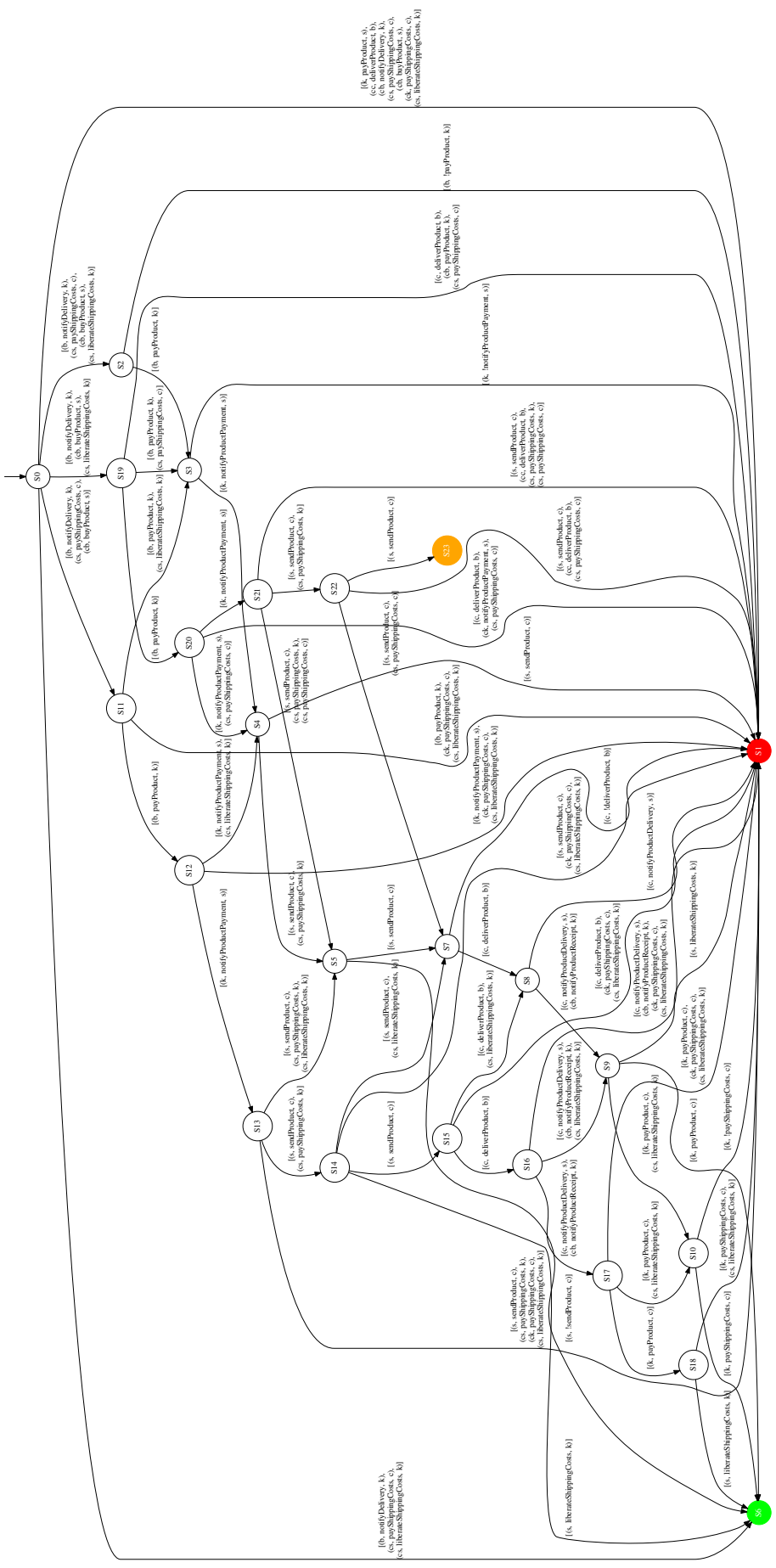

Figura 52 – Autômato obtido do estudo de caso.

A Figura 53 mostra o contrato completo com as correções após o conflito detectado (Veja Seção 6.3). O resultado da ferramenta sobre o contrato corrigido é mostrado na Figura 54. Neste caso o veredito da ferramenta foi um contrato livre de conflitos. Adicionalmente, na Figura 55 é apresentado o autômato resultante da análise.

```
{b,s}[buyProduct](
 {b,k}O(payProduct)^
 {b,k}[payProduct](
  {k,s}O(notifyProductPayment)^
  {k,s}[notifyProductPayment](
   {s,c}O(sendProduct)^
   {s,k}O(payShippingCosts)^
   {s,k}[payShippingCosts](
    {k,c}O(notifyShippingPayment) ^ {s,c}[sendProduct](
     {c,b}O(deliverProduct) ^
     {c,b}[deliverProduct](
      {b, k}O(notifyProductReceipt) ^
      {c, s}O(notifyProductDelivery) ^
      {b,k}[notifyProductReceipt]({k,c}O(payProduct)) ^
      {c, s}[notifyProductDelivery](
       {s, k}O(liberateShippingCosts) ^
       {s, k}[liberateShippingCosts]({k,c}O(payShippingCosts))
))))))));
{b,k}[(!notifyDelivery)*]({k,s}F(payProduct));
{s,k}[(!liberateShippingCosts)*]({k,c}F(payShippingCosts));
{k,c}[(!notifyShippingPayment)*]({c,b}F(deliverProduct));
```

Figura 53 – Contrato corrigido.

```
Analysing contract in study-conflict-free.rcl
Processing contract ...
Completed in 300ms
[CONFLICT-FREE] The analyzed contract is conflict-free.
```

Figura 54 – A saída da ferramenta RECALL sobre o contrato corrigido.

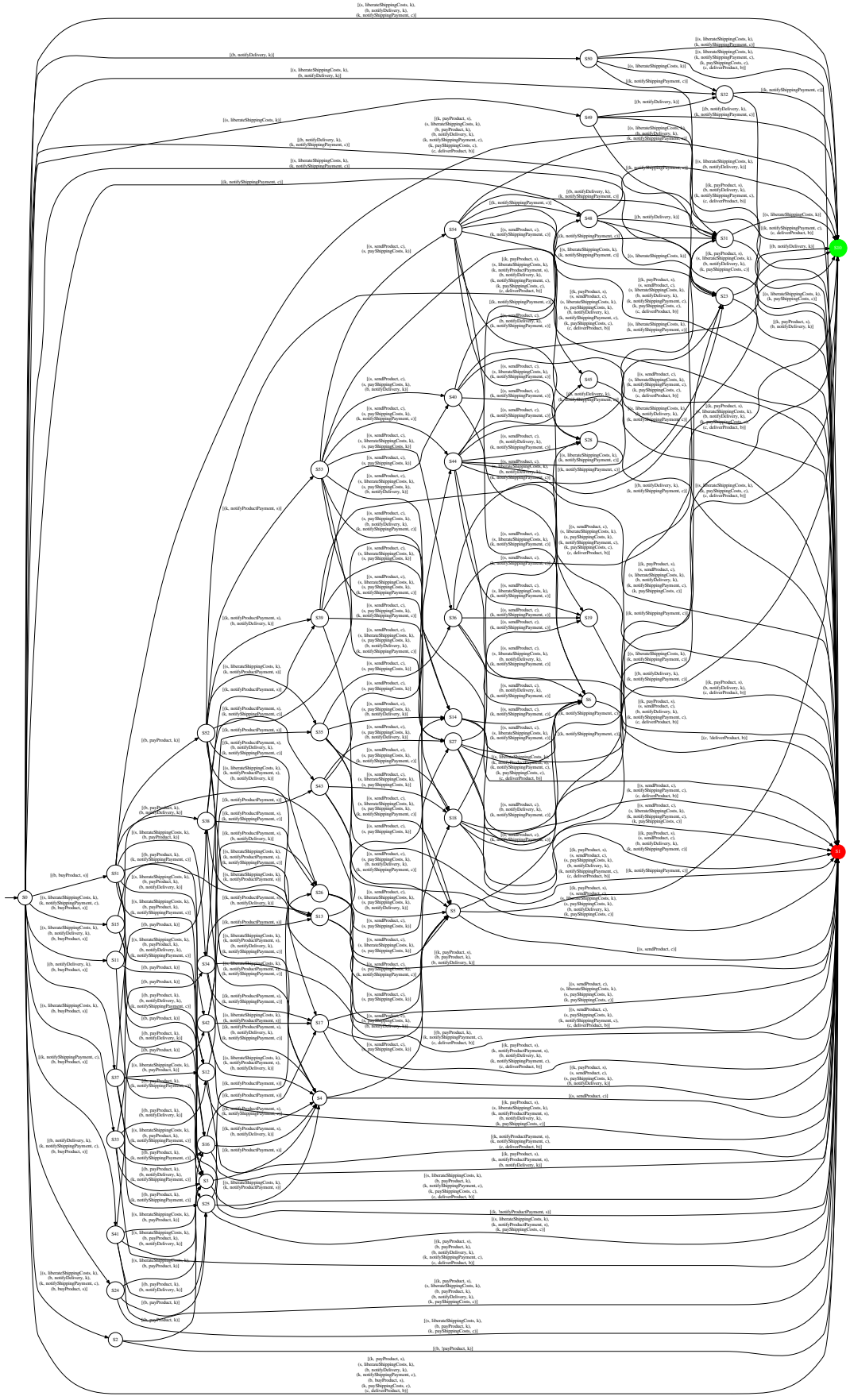

Figura 55 – Autômato obtido do estudo de caso.

Anexos

ANEXO A – A LINGUAGEM $\mathcal{CL}$

A seguir são apresentadas informações detalhadas sobre a semântica e as propriedades da $\mathcal{CL}$ para monitoramento e detecção de conflitos. O Apêndice A.1 mostra as definições para a semântica dos operadores da $\mathcal{CL}$ baseada em *traces* infinitos, aplicada ao monitoramento e verificação de contratos bilaterais. Já o Apêndice A.2 apresenta a semântica de *traces* finitos da $\mathcal{CL}$ utilizada para a detecção de conflitos bilaterais e as funções auxiliares propostas por Fenech, Pace e Schneider[14].

A.1 Propriedades e semântica de *traces* infinitos da $\mathcal{CL}$

A linguagem de contratos $\mathcal{CL}$ [8] foi inspirada nas lógicas deôntica [12] e dinâmica [49] para representação de contratos bilaterais [53] e seu monitoramento [31]. As propriedades da linguagem auxiliam na construção de algoritmos para a verificação formal de contratos descritos em $\mathcal{CL}$. A Figura 56 apresentada as equivalências entre fórmulas da $\mathcal{CL}$ e a Figura 57 apresenta as propriedades da lógica deôntica que também estão presentes na $\mathcal{CL}$. Já a semântica de *traces* infinitos da $\mathcal{CL}$, utilizada para monitoramento, é apresentada na Figura 58.

$$O_{\mathcal{C}}(\alpha) \wedge O_{\mathcal{C}}(\beta) \iff O_{\mathcal{C}}(\alpha \times \beta) \tag{A.1}$$
$$[\alpha_{\times}]\mathcal{C} \Rightarrow [\alpha_{\times} \times \alpha'_{\times}]\mathcal{C} \tag{A.2}$$
$$[\alpha_{\times}]\mathcal{C}_1 \wedge [\alpha'_{\times}]\mathcal{C}_2 \Rightarrow [\alpha_{\times} \times \alpha'_{\times}]\mathcal{C}_1 \wedge \mathcal{C}_2 \tag{A.3}$$
$$F(\alpha) \Rightarrow F(\alpha \times \beta) \tag{A.4}$$
$$F(\alpha \times \beta) \iff F(\alpha) \wedge F(\beta) \tag{A.5}$$
$$F(\alpha + \beta) \iff F(\alpha) \wedge F(\beta) \tag{A.6}$$
$$F(\alpha \cdot \beta) \iff F(\alpha) \vee [\alpha]F(\beta) \tag{A.7}$$
$$O(\alpha + \beta) \iff O(\alpha) \oplus O(\beta) \tag{A.8}$$
$$O(\alpha \cdot \beta) \iff O(\alpha) \wedge [\alpha]O(\beta) \tag{A.9}$$
$$P(\alpha \times \beta) \iff P(\alpha) \wedge P(\beta) \tag{A.10}$$
$$P(\alpha + \beta) \iff P(\alpha) \oplus P(\beta) \tag{A.11}$$
$$P(\alpha \cdot \beta) \iff P(\alpha) \wedge \langle\alpha\rangle P(\beta) \tag{A.12}$$

Figura 56 – Decomposições da $\mathcal{CL}$.

$$O(\alpha) \to P(\alpha) \tag{A.13}$$
$$P(\alpha) \to \neg F(\alpha) \tag{A.14}$$
$$F(\alpha) \to \neg P(\alpha) \tag{A.15}$$
$$\text{se } \alpha = \beta \text{ então } O_{\mathcal{C}}(\alpha) \leftrightarrow O_{\mathcal{C}}(\beta) \tag{A.16}$$

Figura 57 – Propriedades da $\mathcal{CL}$.

$$\sigma \models \top \tag{A.17}$$
$$\sigma \not\models \bot \tag{A.18}$$
$$\sigma \models \mathcal{C}_1 \to \mathcal{C}_2 \iff \sigma \models \mathcal{C}_1 \text{ então } \sigma \models \mathcal{C}_2 \tag{A.19}$$
$$\sigma \models \mathcal{C}_1 \wedge \mathcal{C}_2 \iff \sigma \models \mathcal{C}_1 \text{ e } \sigma \models \mathcal{C}_2 \tag{A.20}$$
$$\sigma \models \mathcal{C}_1 \vee \mathcal{C}_2 \iff \sigma \models \mathcal{C}_1 \text{ ou } \sigma \models \mathcal{C}_2 \tag{A.21}$$
$$\sigma \models \mathcal{C}_1 \oplus \mathcal{C}_2 \iff (\sigma \models \mathcal{C}_1 \text{ e } \sigma \not\models \mathcal{C}_2) \text{ ou } (\sigma \not\models \mathcal{C}_1 \text{ e } \sigma \models \mathcal{C}_2) \tag{A.22}$$
$$\sigma \models [\alpha_\times]\mathcal{C} \iff (\alpha_\times \subseteq \sigma(0) \text{ e } \sigma(1\dots) \models \mathcal{C}) \text{ ou } \alpha_\times \not\subseteq \sigma(0) \tag{A.23}$$
$$\sigma \models [\beta \cdot \beta']\mathcal{C} \iff \sigma \models [\beta][\beta']\mathcal{C} \tag{A.24}$$
$$\sigma \models [\beta + \beta']\mathcal{C} \iff \sigma \models [\beta]\mathcal{C} \text{ ou } \sigma \models [\beta']\mathcal{C} \tag{A.25}$$
$$\sigma \models [\beta^*]\mathcal{C} \iff \sigma \models \mathcal{C} \text{ e } \sigma \models [\beta][\beta^*]\mathcal{C} \tag{A.26}$$
$$\sigma \models [\mathcal{C}_1?]\mathcal{C}_2 \iff \sigma \not\models \mathcal{C}_1 \text{ ou } (\sigma \models \mathcal{C}_1 \text{ e } (\sigma \models \mathcal{C}_2) \tag{A.27}$$
$$\sigma \models O_{\mathcal{C}}(\alpha_\times) \iff \alpha_\times \subseteq \sigma(0) \text{ ou } \sigma(1\dots) \models \mathcal{C} \tag{A.28}$$
$$\sigma \models O_{\mathcal{C}}(\alpha \cdot \alpha') \iff \sigma \models O_{\mathcal{C}}(\alpha) \text{ e } \sigma \models [\alpha]O_{\mathcal{C}}(\alpha') \tag{A.29}$$
$$\sigma \models O_{\mathcal{C}}(\alpha + \alpha') \iff \sigma \models O_{\perp}(\alpha) \text{ ou } \sigma \models O_{\perp}(\alpha') \text{ ou } \sigma \models [\overline{\alpha + \alpha'}]\mathcal{C} \tag{A.30}$$
$$\sigma \models F_{\mathcal{C}}(\alpha_\times) \iff \alpha_\times \not\subseteq \sigma(0) \text{ ou } (\alpha_\times \subseteq \sigma(0) \text{ e } \sigma(1\dots) \models \mathcal{C}) \tag{A.31}$$
$$\sigma \models F_{\mathcal{C}}(\alpha \cdot \alpha') \iff \sigma \models F_{\perp}(\alpha) \text{ ou } \sigma \models [\alpha]F_{\mathcal{C}}(\alpha') \tag{A.32}$$
$$\sigma \models F_{\mathcal{C}}(\alpha + \alpha') \iff \sigma \models F_{\mathcal{C}}(\alpha) \text{ e } \sigma \models F_{\mathcal{C}}(\alpha') \tag{A.33}$$
$$\sigma \models [\overline{\alpha_\times}]\mathcal{C} \iff (\alpha_\times \not\subseteq \sigma(0) \text{ e } \sigma(1\dots) \models \mathcal{C}) \text{ ou } \alpha_\times \subseteq \sigma(0) \tag{A.34}$$
$$\sigma \models [\overline{\alpha \cdot \alpha'}]\mathcal{C} \iff \sigma \models [\overline{\alpha}]\mathcal{C} \text{ e } \sigma \models [\alpha][\overline{\alpha'}]\mathcal{C} \tag{A.35}$$
$$\sigma \models [\overline{\alpha + \alpha'}]\mathcal{C} \iff \sigma \models [\overline{\alpha}]\mathcal{C} \text{ ou } \sigma \models [\overline{\alpha'}]\mathcal{C} \tag{A.36}$$

Figura 58 – Semântica de *traces* infinitos da $\mathcal{CL}$.

A.2 Semântica da $\mathcal{CL}$ para detecção de conflitos

A partir da semântica de *traces* infinitos da $\mathcal{CL}$, Fenech[5] propôs uma extensão que preserva as informações deônticas necessárias para a detecção de conflitos, conforme apresentada na Figura 59. O autômato que especifica um contrato $\mathcal{C}$ em $\mathcal{CL}$ é obtido através das decomposições de $\mathcal{C}$ retornadas pela função $f : \mathcal{CL} \times \mathcal{A}_{\mathcal{B}}^{\times} \to \mathcal{CL}$, definida na Figura 60. O operador binário $/$, descrito na Figura 61, auxilia a função de decomposição f, tratando uma sequência de ações concorrentes.

$$\sigma, \sigma_d \not\models C \iff | \sigma | \neq | \sigma_d | \tag{A.37}$$
$$\sigma, \sigma_d \models \top \iff | \sigma | = 0 \text{ ou } (\sigma_d(0) = \emptyset \text{ e } \sigma(1\dots), \sigma_d(1\dots) \models \top) \tag{A.38}$$
$$\sigma, \sigma_d \not\models \bot \tag{A.39}$$
$$\sigma, \sigma_d \models C_1 \wedge C_2 \iff \sigma, \sigma'_d \models C_1 \text{ e } \sigma, \sigma''_d \models C_2 \text{ e } \sigma_d = \sigma'_d \cup \sigma''_d \tag{A.40}$$
$$\sigma, \sigma_d \models C_1 \oplus C_2 \iff | \sigma | = 0 \text{ ou } (\sigma, \sigma_d \models C_1 \text{ e } \sigma, \sigma_d \not\models C_2) \text{ ou } (\sigma, \sigma_d \models C_2 \text{ e } \sigma, \sigma_d \not\models C_1) \tag{A.41}$$
$$\sigma, \sigma_d \models [\alpha]C \iff | \sigma | = 0 \text{ ou } (\sigma_d(0) = \emptyset) \text{ e } ((\alpha \subseteq \sigma(0) \text{ e } \sigma(1\dots), \sigma_d(1\dots) \models C) \text{ ou } (\alpha \not\subseteq \sigma(0))) \tag{A.42}$$
$$\sigma, \sigma_d \models [\beta \cdot \beta']C \iff \sigma, \sigma_d \models [\beta][\beta']C \tag{A.43}$$
$$\sigma, \sigma_d \models [\beta + \beta']C \iff \sigma, \sigma_d \models [\beta]C \wedge [\beta']C \tag{A.44}$$
$$\sigma, \sigma_d \models [\beta^*]C \iff \sigma, \sigma_d \models C \wedge [\beta][\beta^*]C \tag{A.45}$$
$$\sigma, \sigma_d \models [\overline{\alpha}]C \iff \sigma_d(0) = \emptyset) \text{ e } ((\alpha \not\subseteq \sigma(0) \text{ e } \sigma(1\dots), \sigma_d(1\dots) \models C) \text{ ou } (\alpha \subseteq \sigma(0))) \tag{A.46}$$
$$\sigma, \sigma_d \models [\overline{\alpha \cdot \alpha'}]C \iff \sigma, \sigma_d \models [\overline{\alpha}]C \text{ e } \sigma, \sigma_d \models [\alpha][\overline{\alpha'}]C \tag{A.47}$$
$$\sigma, \sigma_d \models [\overline{\alpha + \alpha'}]C \iff \sigma_d(0) = \emptyset \text{ e } (\sigma, \sigma_d \models [\overline{\alpha}]C \text{ ou } \sigma, \sigma_d \models [\overline{\alpha'}]C) \tag{A.48}$$
$$\sigma, \sigma_d \models O_C(\alpha) \iff | \sigma | = 0 \text{ ou } (\sigma_d(0) = O_\alpha \text{ e } ((\alpha \subseteq \sigma(0) \text{ e } \sigma(1\dots), \sigma_d(1\dots) \models \top) \text{ ou } \sigma(1\dots), \sigma_d(1\dots) \models C) \tag{A.49}$$
$$\sigma, \sigma_d \models O_C(\alpha \cdot \alpha') \iff \sigma, \sigma_d \models O_C(\alpha) \wedge [\alpha]O_C(\alpha') \tag{A.50}$$
$$\sigma, \sigma_d \models O_C(\alpha + \alpha') \iff \sigma, \sigma'_d \models O_\bot(\alpha) \text{ ou } \sigma, \sigma''_d \models O_\bot(\alpha') \text{ ou } (\sigma_d(0) = (\sigma'_d(0) \cup \sigma''_d(0)) \text{ e } \sigma(1\dots), \sigma_d(1\dots) \models [\overline{\alpha \cdot \alpha'}]C) \tag{A.51}$$
$$\sigma, \sigma_d \models F_C(\alpha) \iff | \sigma | = 0 \text{ ou } (\sigma_d(0) = F_\alpha \text{ e } ((\alpha \not\subseteq \sigma(0) \text{ e } \sigma(1\dots), \sigma_d(1\dots) \models \top) \text{ ou } (\alpha \subseteq \sigma(0) \text{ e } \sigma(1\dots), \sigma_d(1\dots) \models C)) \tag{A.52}$$
$$\sigma, \sigma_d \models F_C(\alpha \cdot \alpha') \iff \sigma, \sigma_d \models F_\bot(\alpha) \text{ ou } [\alpha]F_C(\alpha') \tag{A.53}$$
$$\sigma, \sigma_d \models F_C(\alpha + \alpha') \iff \sigma, \sigma'_d \models F_C(\alpha) \wedge F_C(\alpha') \tag{A.54}$$
$$\sigma, \sigma_d \models P(\alpha) \iff | \sigma | = 0 \text{ ou } (\sigma_d(0) = P_\alpha \text{ e } \sigma(1\dots), \sigma_d(1\dots) \models \top) \tag{A.55}$$
$$\sigma, \sigma_d \models P(\alpha \cdot \alpha') \iff \sigma, \sigma_d \models P(\alpha) \wedge [\alpha]P(\alpha') \tag{A.56}$$
$$\sigma, \sigma_d \models P(\alpha + \alpha') \iff \sigma, \sigma_d \models P(\alpha) \wedge P(\alpha') \tag{A.57}$$

Figura 59 – Semântica da $\mathcal{CL}$ para detecção de conflitos.

$$f(1,\varphi) = 1 \tag{A.58}$$

$$f(0,\varphi) = 0 \tag{A.59}$$

$$f(C_1 \wedge C_2,\varphi) = f(C_1,\varphi) \wedge f(C_2,\varphi) \tag{A.60}$$

$$f(C_1 \oplus C_2,\varphi) = \begin{cases} 1 & \text{se } (f(C_1,\varphi)=1 \wedge f(C_2,\varphi)=0) \vee \\ & (f(C_1,\varphi)=0 \wedge f(C_2,\varphi)=1) \\ 0 & \text{se } (f(C_1,\varphi)=1 \wedge f(C_2,\varphi)=1) \vee \\ & (f(C_1,\varphi)=1 \wedge f(C_2,\varphi)=1) \\ f(C_1,\varphi) \oplus f(C_2,\varphi) & \text{em todos os outros casos.} \end{cases} \tag{A.61}$$

$$f([\alpha_\times]C,\varphi) = \begin{cases} C & \text{se } \alpha_\times \subseteq \varphi \\ 1 & \text{em todos os outros casos.} \end{cases} \tag{A.62}$$

$$f([\beta \cdot \beta']C,\varphi) = f([\beta][\beta']C,\varphi) \tag{A.63}$$

$$f([\beta + \beta']C,\varphi) = f([\beta]C \wedge [\beta']C,\varphi) \tag{A.64}$$

$$f([\beta^*]C,\varphi) = f(C \wedge [\beta][\beta^*]C,\varphi) \tag{A.65}$$

$$f(O_c(\alpha_\times),\varphi) = \begin{cases} 1 & \text{se } \alpha_\times \subseteq \varphi \\ C & \text{em todos os outros casos.} \end{cases} \tag{A.66}$$

$$f(O_c(\alpha \cdot \alpha'),\varphi) = f(O_C(\alpha) \wedge [\alpha]O_C(\alpha')) \tag{A.67}$$

$$f(O_c(\alpha + \alpha'),\varphi) = \begin{cases} 1 & \text{se } f(O_0(\alpha),\varphi)=1 \vee f(O_0(\alpha'),\varphi)=1 \\ C & \text{se } f(O_0(\alpha),\varphi)=0 \vee f(O_0(\alpha'),\varphi)=0 \\ O_C(\alpha+\alpha'/\varphi), & \text{em todos os outros casos.} \end{cases} \tag{A.68}$$

$$f(F_c(\alpha_\times),\varphi) = \begin{cases} C & \text{se } \alpha_\times \subseteq \varphi \\ 1 & \text{em todos os outros casos.} \end{cases} \tag{A.69}$$

$$f(F_c(\alpha \cdot \alpha'),\varphi) = f([\alpha]F_C(\alpha')) \tag{A.70}$$

$$f(F_c(\alpha + \alpha'),\varphi) = f(F_C(\alpha) \wedge F_C(\alpha') \tag{A.71}$$

$$f([\overline{\alpha_\times}]C,\varphi) = \begin{cases} C & \text{se } \alpha_\times \not\subseteq \varphi \\ 1 & \text{em todos os outros casos.} \end{cases} \tag{A.72}$$

$$f([\overline{\alpha \cdot \alpha'}]C,\varphi) = f([\overline{\alpha'}][\overline{\alpha'}]C,\varphi) \tag{A.73}$$

$$f([\overline{\alpha + \alpha'}]C,\varphi) = \begin{cases} C & \text{se } f([\overline{\alpha}]C,\varphi)=C \vee f([\overline{\alpha'}]C,\varphi)=C \\ 1 & \text{se } f([\overline{\alpha}]C,\varphi)=1 \wedge f([\overline{\alpha'}]C,\varphi)=1 \\ f([\overline{\alpha+\alpha'/\varphi}]C & \text{em todos os outros casos.} \end{cases} \tag{A.74}$$

Figura 60 – Função de decomposição da $\mathcal{CL}$.

$$\alpha/\varphi = \emptyset \text{ se } \varphi = \alpha, \text{ senão } \mathbf{0} \tag{A.75}$$
$$(\mathbf{0} \cdot \alpha)/\varphi = \mathbf{0} \tag{A.76}$$
$$(\mathbf{1} \cdot \alpha)/\varphi = \alpha \tag{A.77}$$
$$(\alpha \cdot \alpha')/\varphi = (\alpha/\varphi) \cdot \alpha' \tag{A.78}$$
$$(\alpha + \alpha')/\varphi = \alpha/\varphi + \alpha'/\varphi \tag{A.79}$$

Figura 61 – Operador "/" que auxilia a função de decomposição.

TRABALHOS PUBLICADOS PELO AUTOR

O artigo abaixo foi publicado pelo autor durante o programa de mestrado. O trabalho descreve o método de detecção de conflitos proposto com base na linguagem estendida $\mathcal{RCL}$.

- Wellington Aparecido Della Mura, Adilson Luiz Bonifácio, **Devising a conflict detection method for multi-party contracts**, 34th International Conference of the Chilean Computer Science Society, SCCC 2015, Santiago, Chile, November 9-13, 2015, http://dx.doi.org/10.1109/SCCC.2015.7416574, (Qualis CC 2012, B3)

Atualmente, outros trabalhos estão sendo escritos para serem apreciados em conferências e periódicos especializados da área. Estes trabalhos devem englobar a implementação da ferramenta RECALL, sua aplicação num estudo de caso real (Vide Capítulo 6), além de experimentos práticos para analisar a eficiência da ferramenta em termos de processamento e memória, bem como sua escalabilidade com relação ao tamanho dos contratos.