Compilação de Código Certificado para Modelos Reo*

Erick Simas Grilo[1], **Bruno Lopes**[1]

[1]Instituto de Computação – Universidade Federal Fluminense (UFF)
Niterói – RJ – Brasil

simas_grilo@id.uff.br, bruno@ic.uff.br

Abstract. *Critical Systems are systems which require high reliability and are present in a wide variety of domains. Should they fail, serious problems can occur, resulting in financial loss and even deaths. Standard software engineering techniques does not suffice in guaranteeing the required level of reliability. Reo is a graphical modelling language based in coordination which focuses on model system interaction by means of common primitives in distributed systems. Constraint Automata are defined as primitive formal semantics for Reo, providing means to reason about and prove properties of Reo models. The present work describes a formalization of Constraint Automata in Coq proof assistant, regarding its main formalisms, including a compositional operation.*

Resumo. *Sistemas críticos são sistemas que necessitam de alta confiabilidade e estão presente nos mais variados domínios. Em caso de falha, tais sistemas podem provocar problemas sérios, causando prejuízos financeiros e até morte. A engenharia de* software *padrão não é o suficiente para garantir o nível requerido de confiabilidade. Reo é uma linguagem gráfica de modelagem baseada em coordenação cujo foco é em modelar a interação de sistemas por meio do uso de primitivas comuns em sistemas distribuídos.* Constraint Automata *são definidos como a semântica formal primitiva para Reo, providenciando meios formais para raciocinar sobre e provar propriedades acerca de modelos Reo. O presente trabalho descreve uma formalização de Constraint Automata no assistente de provas Coq, onde os principais formalismos da teoria são formalizados, junto de uma operação de composição.*

Sistemas críticos são sistemas cuja falha pode resultar em mortes, destruição significativa, alta perda financeira ou dano ambiental [Knight 2002]. Em suma, sistemas que precisam de um alto nível de confiabilidade. Técnicas padrão de engenharia de *software* não são projetadas para lidar com sistemas não tolerantes a falhas. Apenas testes, por exemplo, não garantem que o sistema possui a confiabilidade requerida ou as propriedades desejadas. Nestes casos, tais garantias necessitam de provas de que requisitos são cumpridos e/ou de que a corretude do sistema seja provada formalmente.

Sistemas formais compõem um *background* teórico e implementado (e.g. *software*) capaz de modelar e raciocinar sobre sistemas, garantindo (matematicamente) que os requisitos são atendidos e que sistemas se comportam conforme o esperado. O uso de sistemas formais para certificar a linha 1 do metrô de Paris, França [Gerhart et al. 1994], culminou na construção de um sistema de metrô totalmente automatizado, eliminando

*Os autores agradacem ao CNPq e à FAPERJ pelo suporte dado ao projeto

a necessidade de construir uma nova linha de metrô e consequentemente economizando milhões de euros.

O uso de sistemas lógicos para modelar e raciocinar sobre sistemas parece ser uma abordagem promissora por permitirem a prova matemática de tais propriedades em um ambiente computacional [Grilo and Lopes 2018]. Assistentes de prova [Loveland 2014], como Coq [Dowek et al. 1992] levam à possibilidade de automatizar a verificação de tais sistemas e fornecer código certificado. A base teórica leva a ver provas como programas e programas como provas o que permite transformar uma prova de que os requisitos são atendidos em um modelo em um código (i.e. um programa) certificado.

O Coq é um dos mais proeminentes assistentes de prova. Ele lida com uma linguagem de alto nível para modelar, provar e fornecer código certificado automaticamente, além de possuir uma linguagem de táticas usada no processo de prova facilmente extensível.

Reo [Arbab 2004] é uma linguagem formal gráfica de modelagem de coordenação baseada em canais (conectores — vide Figura 1) para a modelagem e verificação de sistemas. Concebida como uma linguagem cujo propósito é conectar sistemas já existentes no âmbito da engenharia de *software* baseada em componentes [Kokash and Arbab 2009], onde novos sistemas são projetados considerando sistemas já existentes, Reo modela a interação de tais sistemas, de acordo com o resultado desejado. Em Reo, canais complexos são construídos composicionalmente de canais mais simples. Tais canais são chamados de conectores e compõem o núcleo de Reo. É por meio desses conectores que Reo integra instâncias de diferentes sistemas em um sistema baseado em componentes.

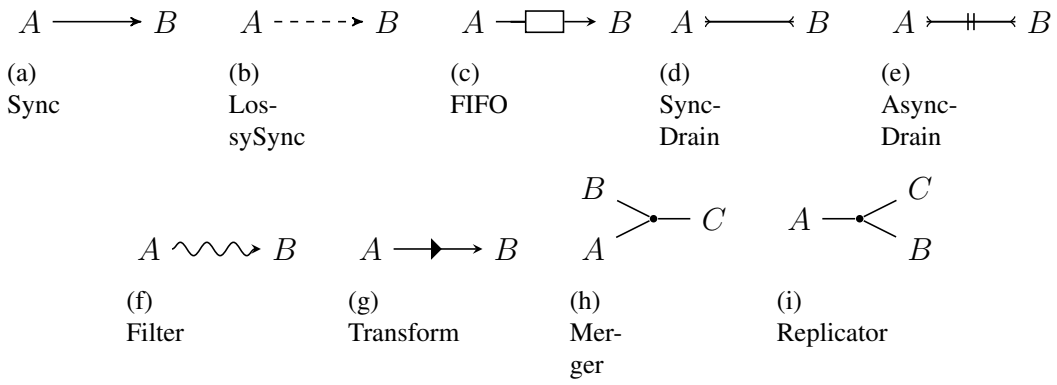

Figura 1. Conectores canônicos em Reo

Até onde é de conhecimento dos autores, há apenas um trabalho que visa a implementação da semântica formal de Reo no Coq [Zhang et al. 2016]. Tal implementação, por formalizar diretamente o comportamento de conectores Reo, possibilita apenas o raciocínio nos canais formalizados, não permitindo a criação de novos canais definidos pelo usuário. Neste mesmo trabalho, a composição de conectores mais complexos é definida por meio da conjunção de conectores, não fazendo uso de uma operação como a definida por [Arbab 2006]. Dessa forma, [Zhang et al. 2016] acabam por formalizar Reo no Coq não fazendo uso das definições e operações definidas pela semântica formal criada pelos criadores de Reo [Arbab 2006].

Constraint Automata [Arbab 2006] é o formalismo mais básico que denota

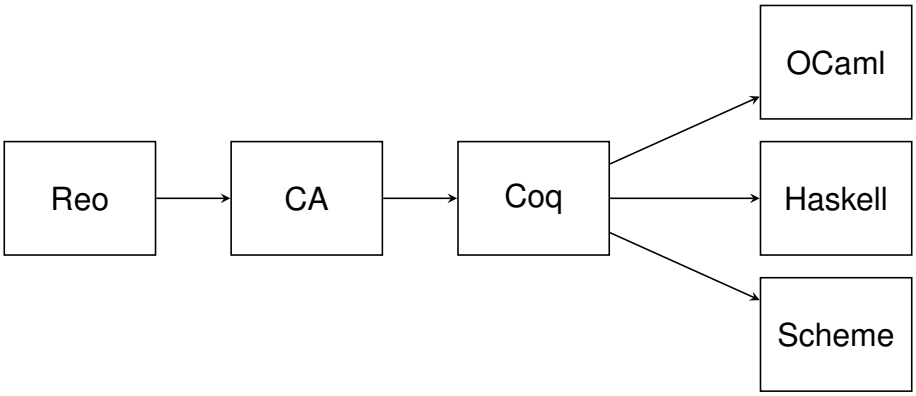

Figura 2. Sequência de execução do *framework*

semântica formal para Reo. Trata-se de um modelo de autômatos onde as transições são dependentes de proposições lógicas acerca de dados vistos nas portas do autômato, que por sua vez denotam nós em Reo, representando pontos dos conectores Reo pelos quais passam fluxos de dados. Portanto, transições neste autômato são reguladas por portas conectadas e os itens de dados vistos nestas portas. Cada nó em Reo pode ser visto como uma instância de um *software* modelado. Para cada conector Reo canônico há um *constraint automaton* associado [Kokash and Arbab 2009].

O presente trabalho trata da construção de um compilador certificado para modelos Reo. Este compilador foi implementado por meio da formalização construtiva de *Constraint Automata* no Coq. São formalizadas as principais definições referentes a este formalismo, uma operação de composição de autômatos. Autômatos que modelam os conectores (elencados na Figura 1) Reo também são formalizados. A Figura 2 ilustra o passo a passo da utilização do *framework* obtido. A partir de um modelo Reo, é gerado um correspondente em *constraint automaton* no Coq, possibilitando a obtenção de código certificado para uma dentre as três linguagens a seguir: OCaml, Haskell e Scheme.

A formalização de cada *constraint automaton* correspondente aos conectores Reo é também realizada. Toda a implementação pode ser encontrada em `https://github.com/simasgrilo/CACoq`. A Figura 3 denota um exemplo que modela o Alternating Bit Protocol (ABP), um protocolo que implementa a comunicação segura entre duas entidades. Um ente transmite uma mensagem intermitentemente até receber a confirmação do recebimento. O Listing 1 exemplifica o código em Scheme certificado extraído do ABP modelado em Reo, contendo o autômato correspondente resultante da composição dos conectores Reo canônicos envolvidos.

Listing 1. Trecho do modelo do ABP compilado para Haskell

```
1  transformCaBehavior :: List (Prod (Prod (List AbpPorts) (DC AbpPorts Nat))(List ()))
2  transformCaBehavior = Cons (Pair (Pair (Cons A (Cons B Nil)) (TrDc transformFunction A B)) (Cons _ Nil)) Nil
3
4  lossySyncCaBehavior :: List(Prod (Prod (List AbpPorts) (DC AbpPorts Nat))(List ()))
5  lossySyncCaBehavior =
6  Cons (Pair (Pair (Cons B (Cons C Nil)) (EqDc B C)) (Cons _ Nil)) (Cons(Pair (Pair (Cons B Nil) TDc)
7      (Cons _ Nil)) Nil)
8
9  lossySync2CaBehavior :: List (Prod (Prod (List AbpPorts) (DC AbpPorts Nat)) (List ()))
10 lossySync2CaBehavior =
11 Cons (Pair (Pair (Cons D (Cons E Nil)) (EqDc D E)) (Cons _ Nil)) (Cons (Pair (Pair (Cons D Nil) TDc)
12     (Cons _ Nil)) Nil)
13
14 filterCaBehavior :: List (Prod (Prod (List AbpPorts) (DC AbpPorts Nat)) (List ()))
```

```
15  filterCaBehavior = Cons (Pair (Pair (Cons E (Cons F Nil)) (AndDc (EqDc E B) (EqDc E F))) (Cons _ Nil))
16  (Cons (Pair (Pair (Cons E Nil) (NegDc (EqDc E B))) (Cons _ Nil)) Nil)
17
18  resultingPaAbp :: ConstraintAutomata () AbpPorts Nat
19  resultingPaAbp =
20  unsafeCoerce buildPA abpPortsEq
21  (prod_eqdec (\_ _ -> transformStateEq) (\_ _ -> lossySyncStateEq))
22  (prod_eqdec (\_ _ -> lossySyncState2Eq) (\_ _ -> filterState3Eq))
23  filterLossySyncProduct filterLossySyncProduct2
```

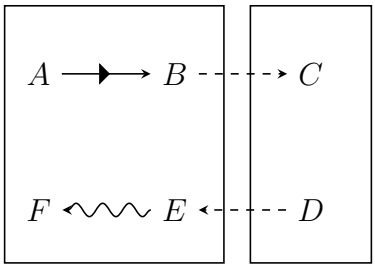

Figura 3. Modelagem do Alternating Bit Protocol em Reo

O resultado obtido neste trabalho é um compilador de modelos Reo para *Constraint Automata* em um sistema como o Coq. Tal formalização permite a prova de propriedades acerca de modelos Reo, tais como propriedades sobre o fluxo de dados em um conector Reo ou que determinadas configurações só são atingidas em certos casos. Por utilizar o Coq, o ferramental de extração de código certificado deste assistente de provas pode ser aplicado a fim de gerar código certificado do modelo para outras linguagens.

Referências

Arbab, F. (2004). Reo: a channel-based coordination model for component composition. *Mathematical Structures in Computer Science*, 14(3):329–366.

Arbab, F. (2006). Coordination for component composition. *Electronic Notes in Theoretical Computer Science*, 160:15 – 40. Proceedings of the International Workshop on Formal Aspects of Component Software (FACS 2005).

Dowek, G., Felty, A., Herbelin, H., Huet, G., Murthy, C., Parent, C., Paulin-Mohring, C., and Werner, B. (1992). *The COQ Proof Assistant: User's Guide: Version 5.6*. INRIA.

Gerhart, S., Craigen, D., and Ralston, T. (1994). Case study: Paris metro signaling system. *IEEE Software*, 11(1):32–28.

Grilo, E. and Lopes, B. (2018). Formalization and certification of software for smart cities. In *International Joint Conference on Neural Networks (IJCNN)*, pages 662–669. IEEE.

Knight, J. C. (2002). Safety critical systems: challenges and directions. In *Proceedings of the 24th International Conference on Software Engineering*, pages 547–550. ACM.

Kokash, N. and Arbab, F. (2009). *Formal Behavioral Modeling and Compliance Analysis for Service-Oriented Systems*, pages 21–41. Springer Berlin Heidelberg, Berlin, Heidelberg.

Loveland, D. W. (2014). *Automated Theorem Proving: a logical basis*. Elsevier.

Zhang, X., Hong, W., Li, Y., and Sun, M. (2016). Reasoning about connectors in coq. In *International Workshop on Formal Aspects of Component Software*, pages 172–190. Springer.