Visão Geral Informal da Abd1, uma Linguagem para Programação de Raciocínios Abdutivos Dirigida a não Especialistas em Programação Lógica

Carlos Eduardo Andrade Oliveira[1], Osvaldo Luiz de Oliveira[1]

[1]Faculdade Campo Limpo Paulista (FACCAMP)
Rua Guatemala, 167, Jd. América – 13.231-230 – Campo Limpo Paulista – SP – Brasil
carlos.br@gmail.com, osvaldo@faccamp.br

Abstract. *The computational solutions proposed for abductive reasoning programming require extensive knowledge of logic programming and operational semantics of systems that extend logic programming to allow the calculation of abductions. Therefore, these solutions are unviable for non-experts in logic programming. This article presents an informal overview of Abd1, a language specific for abductive reasoning programming designed to be used by non-experts in logic programming.*

Resumo. *As soluções computacionais propostas para a programação de raciocínio abdutivo requerem do programador amplo conhecimento de programação lógica e da semântica operacional de sistemas que a estendem para permitir o cálculo abdutivo. Como consequência, estas soluções são inviáveis para não especialistas em programação lógica. Este artigo apresenta uma visão geral informal da Abd1, uma linguagem específica para programação de raciocínios abdutivos projetada para ser utilizada por não especialistas em programação lógica.*

1. Introdução

Abdução é um tipo de raciocínio no qual hipóteses são formuladas na tentativa de explicar fatos observados, por meio de uma teoria utilizada como fundamento (Josephson & Josephson, 1994). Abdução, em programação lógica (Denecker & Kakas, 2002), tem sido estudada no contexto da busca de soluções computacionalmente factíveis para muitos problemas, incluindo diagnóstico, entendimento de linguagem natural, assimilação de conhecimento, revisão de crenças e raciocínios na Web Semântica (Gavanelli *et al*., 2015). As soluções computacionais comumente propostas fazem uso de *framewoks* que estendem a programação lógica tradicional com sistemas de programação de restrições (*contraint logic programming systems*) (Jaffar & Maher, 1994). São exemplos de tais abordagens *Abductive Logic Programming* (ALP) (Kakas, Kowalski & Toni, 1993), *Abductive Constraint Logic Programming* (ACLP) (Kakas, Michael & Mourlas, 2000) e *Constraint Handling Rules* (CHR) (Abdennadher & Christiansen, 2000; Frühwirth, 1998). A programação de raciocínios abdutivos empregando as soluções existentes requer do programador grande esforço de aprendizagem sobre lógica, sobre a semântica operacional de cada uma das linguagens lógicas que são combinadas nos *frameworks* para abdução e, também, grande esforço de escrita de programas que reúnem simultaneamente linguagens de diferentes paradigmas

(e.g., regras, questões, execução sequencial de sentenças lógicas, execução com *backtracking*, recursão, *looping*, negação como falha, mecanismo de resolução de regras de restrição). Ou seja, as soluções para programação de raciocínios abdutivos existentes não estão ao alcance de um não especialista em programação lógica.

Ao contrário das soluções existentes, que servem a diferentes propósitos, a linguagem Abd1 foi projetada no cenário de uma pesquisa (Oliveira, 2016) que considera o desafio da programação de raciocínios abdutivos por não especialistas em programação lógica. A linguagem é baseada nas seguintes escolhas de projeto: (1) facilidade de aprendizagem da linguagem em vez grande capacidade expressiva; (2) emprego de um único paradigma de programação, o declarativo puro; (3) especificidade de propósito i.e., opção por uma linguagem específica para a programação de raciocínios abdutivos, em vez de uma linguagem que projetada para vários propósitos; (4) proximidade linguística com o domínio dos raciocínios abdutivos i.e., uso de termos do domínio dos raciocínios abdutivos tais como teoria, hipóteses, fatos etc.; e (5) simplicidade composicional de raciocínios, i.e., facilidade de composição de raciocínios complexos, a partir de raciocínios mais simples.

A Abd1 pode ser útil em situações que envolvem raciocínio abdutivo e o programador não é um especialista em programação lógica. Como exemplos destas situações estão o uso em modelagem computacional[1] com propósitos educacionais e o uso para programação de modelos de máquinas ou processos por um engenheiro, para posterior execução destes modelos por técnicos, com o objetivo de auxiliá-los no diagnóstico de falhas.

O restante deste artigo está organizado da seguinte forma. A Seção 2 apresenta os principais elementos da linguagem Abd1. A Seção 3 descreve as funcionalidades de um sistema desenvolvido para integrar a edição, a compilação e a execução de programas escritos em Abd1. Por fim, a Seção 4 apresenta conclusões e sugere trabalhos futuros.

2. A linguagem Abd1

Os elementos lógicos do projeto da linguagem Abd1 são apresentados formalmente em Oliveira (2016). Este artigo limita-se a discutir a linguagem Abd1 do ponto de vista de seu uso. Programas em Abd1 são compostos por teorias, fatos, raciocínios abdutivos e condições. Uma teoria define o conhecimento sobre um fenômeno e é declarada por meio de sentenças em Lógica Proposicional[2] na forma HF (Rodrigues, Oliveira & Oliveira, 2014). Fatos definem observações, sinais, sintomas, evidências, marcas etc.,

[1] Modelagem computacional refere-se à atividade de criar programas que estabelecem modelos para fenômenos reais ou abstratos. Na área da educação, modelagem computacional tem sido empregada como uma opção para implementar uma pedagogia construtivista que faz uso do computador não como uma ferramenta para ensinar, mas como um instrumento para um estudante aprender pela atividade de construir modelos. Embora não se refiram à programação de raciocínios abdutivos, Campbell & Oh (2015) e Helikar *et al*. (2015) são estudos recentes sobre modelagem computacional em educação.

[2] Conceitos da Lógica Proposicional (Nicoletti, 2010) são definidos de forma usual e não serão tratados neste artigo. A escolha da Lógica Proposicional como fundamento lógico da Abd1 deve-se à sua simplicidade, pelo menos quando comparada com outras lógicas como, por exemplo, a Lógica de Predicados.

para serem explicados por raciocínios abdutivos. Raciocínios abdutivos são compostos por uma ou mais teorias e um ou mais fatos e, opcionalmente, uma ou mais condições. Condições são usadas para definir contextos e circunstâncias para raciocínios abdutivos e, tal como na definição de uma teoria, uma condição é declarada por sentenças em Lógica Proposicional na forma HF.

Exemplo 1. (Um programa escrito na linguagem Abd1). O programa Abd1 da Figura 1 é um modelo para uma parte das falhas comumente apresentadas, quando um motor à combustão, de um automóvel típico, não dá partida. O programa declara duas teorias (linhas 1 a 9 e 10 a 17), um conjunto de fatos (linha 18), um raciocínio abdutivo (linhas 19 a 23) e um conjunto de condições (linhas 24 a 27). Linhas em branco podem ser introduzidas no programa com vistas a facilitar a sua leitura.

```
1   Theory teoriaFalhasPartidaMotor
2   {
3     bateriaSemCarga → motorNaoDaPartida ∧ luzesPainelApagadas ∧ motorArranqueNaoFazRuido ,
4     motorArranqueComDefeito → motorNaoDaPartida ∧ motorArranqueNaoFazRuido ,
5     falhasAlimentacao → motorNaoDaPartida ,

6     bateriaComCarga → ¬ bateriaSemCarga ,
7     motorArranqueFazRuido → ¬ motorArranqueNaoFazRuido ,
8     luzesPainelAcesas → ¬ luzesPainelApagadas
9   }

10  Theory teoriaFalhasAlimentacao
11  {
12    bombaCombustivelComDefeito → falhasAlimentacao ,
13    filtroCombustivelObstruido → falhasAlimentacao ,
14    condutorCombustivelRompido → falhasAlimentacao ,
15    tanqueCombustivelVazio → falhasAlimentacao ∧ marcadorMostraFaltaCombustivel ,

16    marcadorMostraExistenciaCombustivel → ¬ marcadorMostraFaltaCombustivel
17  }

18  Facts fatoMotorNaoDaPartida { motorNaoDaPartida }

19  AbductiveReasoning porQueMotorNaoDaPartida
20  <
21    teoriaFalhasPartidaMotor ,
22    fatoMotorNaoDaPartida
23  >

24  Conditions condicaoMotorArranqueFazRuido
25  {
26    motorArranqueFazRuido
27  }
```

Figura 1. Um programa escrito na linguagem Abd1.

A sintaxe da Abd1 é simples e pode ser descrita informalmente assim: (1) uma teoria ou uma condição é um conjunto de sentenças da Lógica Proposicional na forma HF delimitadas pelos símbolos "{" e "}" precedidos, respectivamente, pelas palavras *Theory* ou *Conditions*, seguidas por um nome que identifica a teoria ou a condição. (2) Um fato é um conjunto de átomos da Lógica Proposicional delimitados pelos símbolos "{" e "}" precedidos pela palavra *Facts* seguida por um nome que identifica o conjunto de fatos. (3) A declaração de um raciocínio abdutivo inicia com a palavra *AbductiveReasoning* seguida por um nome para o raciocínio e pela declaração, delimitada pelos símbolos "<" e ">", de uma expressão

especificando nomes de teorias, de uma expressão especificando nomes de fatos e, opcionalmente, de uma expressão especificando nome de condições. Expressões que especificam nomes de teorias podem ser constituídas por apenas um nome de teoria ou pela especificação de nomes de duas ou mais teorias separadas pelo símbolo de união ("∪"). As expressões que especificam nomes de fatos e nomes de condições são descritas de forma similar às expressões que especificam nomes de teorias.

A teoria *teoriaFalhasPartidaMotor* declara possíveis falhas associadas à partida de um motor de automóvel, por meio de sentenças da linguagem da Lógica Proposicional, na forma HF. Por exemplo, a sentença da linha 4, declara algo que, em português, poderia ser traduzido aproximadamente como "se o motor de arranque está com defeito então o motor [a combustão] não dá partida e o motor de arranque não faz ruído [durante a ignição]". As outras sentenças desta teoria e da teoria *teoriaFalhasAlimentacao* e do conjunto de condições *condicaoMotorArranqueFazRuido* podem ser compreendidas de maneira similar.

A teoria *teoriaFalhasAlimentacao* é uma teoria "especializada" em falhas relacionadas ao sistema de alimentação do automóvel que, no programa, é descrito como sendo composto por tanque de combustível, bomba de combustível, filtro de combustível e condutores de combustível. A definição de diferentes teorias para capturar diferentes aspectos do fenômeno que está sendo modelado é um importante recurso da linguagem Abd1, pois permite especificar raciocínios para inferir hipóteses mais genéricas (por exemplo, por usar uma teoria mais genérica como a teoria *teoriaFalhasPartidaMotor*), ou inferir hipóteses que são explicações mais detalhadas para o fenômeno (por exemplo, por usar em um raciocínio uma teoria mais específica como a teoria *teoriaFalhasAlimentacao* ou a união desta teoria com a teoria *teoriaFalhasPartidaMotor*)[3].

O raciocínio abdutivo *porQueMotorNaoDaPartida* é definido pela teoria *teoriaFalhasPartidaMotor* e pelo fato *fatoMotorNaoDaPartida*, declarados no programa. Este raciocínio objetiva inferir hipóteses que explicam o fato do motor à combustão não dá partida, tendo como fundamento a teoria *teoriaFalhasPartidaMotor*.

O conjunto de condições, *condicaoMotorArranqueFazRuido*, estabelece um contexto especial para ser utilizado em raciocínios que vierem a ser formulados. Neste caso, tal condição é útil para realizar raciocínios que buscam hipóteses, por exemplo, para explicar o fato do motor não dar partida, no contexto do motor fazer ruído durante a ignição.

Sugere-se que sejam declarados no corpo do programa, durante a fase de concepção do programa, os raciocínios abdutivos, fatos e condições que o programador prevê que serão mais amplamente utilizados, entretanto a linguagem Abd1 foi projetada para permitir, também, a composição de raciocínios de forma interativa, por meio de "linha de comando", de maneira muito parecida com sistemas interativos que implementam Prolog (e.g., SWI-Prolog, SICStus Prolog). Assim, raciocínios podem ser compostos e executados de forma interativa. Exemplos da execução de raciocínios declarados no corpo do programa ou compostos diretamente em linha de comando, bem suas respectivas execuções, são apresentadas na Figura 2. O raciocínio apresentado na Figura 2(a) foi declarado no corpo do programa e os demais raciocínios apresentados na Figura 2 foram compostos diretamente na "linha de comando". Sintaticamente, estes últimos foram declarados pelos símbolos "<" e ">" delimitando, em ordem, nomes de teorias, nomes de fatos e, opcionalmente, nomes de condições que participam do raciocínio.

[3] Este mesmo recurso é útil também para enriquecer um modelo segundo teorias que referem a diferentes perspectivas de um fenômeno modelado. Por exemplo, uma teoria química sobre o funcionamento do motor à combustão, uma teoria dos custos dos componentes de um motor à combustão etc..

***Question**: porQueMotorNaoDaPartida .*
***Hypotheses**: { bateriaSemCarga , motorArranqueComDefeito , falhaAlimentacao }*
(a)

***Question**: < teoriaFalhasPartidaMotor , {motorNaoDaPartida }, condicaoMotorArranqueFazRuido > .*
***Hypotheses**: { falhaAlimentacao }*
(b)

***Question**: < teoriaFalhasPartidaMotor ∪ teoriaFalhasAlimentacao,*
{ motorNaoDaPartida }, condicaoMotorArranqueFazRuido > .
***Hypotheses**: { bombaCombustivelComDefeito , filtroCombustivelObstruido ,*
condutorCombustivelRompido , tanqueCombustivelVazio }
(c)

***Question**: < teoriaFalhasPartidaMotor ∪ teoriaFalhasAlimentacao ,*
{ motorNaoDaPartida , marcadorMostraExistenciaCombustivel } ,
condicaoMotorArranqueFazRuido > .
***Hypotheses**: { bombaCombustivelComDefeito ∧ marcadorMostraExistenciaCombustivel ,*
filtroCombustivelObstruido ∧ marcadorMostraExistenciaCombustivel ,
condutorCombustivelRompido ∧ marcadorMostraExistenciaCombustivel }
(d)

***Question**: < teoriaFalhasPartidaMotor , { motorNaoDaPartida , motorArranqueNaoFazRuido } > ∩*
< teoriaFalhasPartidaMotor , { motorNaoDaPartida , luzesPainelApagadas } > .
***Hypotheses**: { bateriaSemCarga }*
(e)

Figura 2. Exemplos de execuções de raciocínios empregando o programa apresentado na Figura 1.

O raciocínio da Figura 2(a) está declarado no corpo do programa e calcula as hipóteses que explicam o fato *motorNaoDaPartida* por meio da teoria *teoriaFalhasPartidaMotor*. A Figura 2(b) é um exemplo de um raciocínio realizado sob certo contexto. Especificamente, tal raciocínio é o mesmo da Figura 2(a) só que no contexto do motor de arranque fazer ruído durante a ignição, o que é declarado pela condição *condicaoMotorArranqueFazRuido*. Pode-se observar que o contexto altera significativamente o conjunto resultante de hipóteses. O raciocínio da Figura 2(c) é o mesmo da Figura 2(b), só que objetiva buscar explicações mais detalhadas sobre quais são as possíveis falhas de alimentação. Para isto, a teoria especializada em falhas de alimentação, *teoriaFalhasAlimentacao*, é unida à teoria *teoriaFalhasPartidaMotor*, para possibilitar a inferência de hipóteses mais detalhadas para o fato *motorNaoDaPartida*. Pode-se observar que o resultado do raciocínio são quatro hipóteses sobre possíveis falhas na alimentação. Este recurso de composição de uma base de conhecimento maior pela união de teorias, é especialmente importante para modelagem e é possível na linguagem Abd1 porque não existe ordem das sentenças dentro de uma teoria (teoria é conjunto). Isto não é possível em linguagens como, por exemplo, Prolog porque a ordem das sentenças nesta última tem efeito sobre a semântica do programa. O raciocínio da Figura 2(d) é o mesmo da Figura 2(c), exceto pela inclusão de um novo fato: o marcador do painel do automóvel mostra a existência de combustível, o que é descrito pela adição da proposição *marcadorMostraExistenciaCombustivel* ao conjunto de fatos da Figura 2(c). Em Abd1, raciocínios são conjuntos de hipóteses, assim a linguagem permite a escrita de raciocínios complexos a partir de expressões envolvendo a união, a intersecção ou a diferença entre raciocínios mais simples. O raciocínio da Figura 2(e) ilustra este recurso, ao descrever um raciocínio composto pela intersecção de dois raciocínios mais simples em que, um deles infere hipóteses que explicam os fatos do motor não dar partida e o motor de arranque não fazer ruído durante a ignição e, o outro, infere hipóteses do motor não dar partida e as luzes do painel do automóvel estarem apagadas. Em outras palavras, o raciocínio composto por estes dois raciocínios mais simples infere possíveis hipóteses que explicam, simultaneamente, os dois raciocínios mais simples.

■

Um raciocínio abdutivo infere possíveis hipóteses para explicar um conjunto de fatos. Alguns estudos (e.g., Caroprese & Trubitsyna, 2014) sugerem que algumas hipóteses podem explicar melhor o conjunto de fatos do que outras. Embora este artigo não trate disto, a Abd1 permite especificar raciocínios que utilizam heurísticas para seleção de boas hipóteses. A gramática completa e a semântica da linguagem Abd1 estão propostas em Oliveira (2016).

3. Ambiente Integrado de Edição, Compilação e Execução de Programas

Um ambiente integrado de edição, compilação e execução de programas (Figura 3), nomeado AIDEP-Abd1, acrônimo para "Ambiente Integrado de Desenvolvimento e Edição de Programas escritos em Abd1", foi desenvolvido para Windows[4] e está disponível para *download* em `http://abd1.gear.host/`. O ambiente é composto por: (1) área de edição de programas (aba *Edit*), (2) área de execução de raciocínios (aba *Execute*) e (3) área de acompanhamento de nomes declarados no programa (caixa *Used names*).

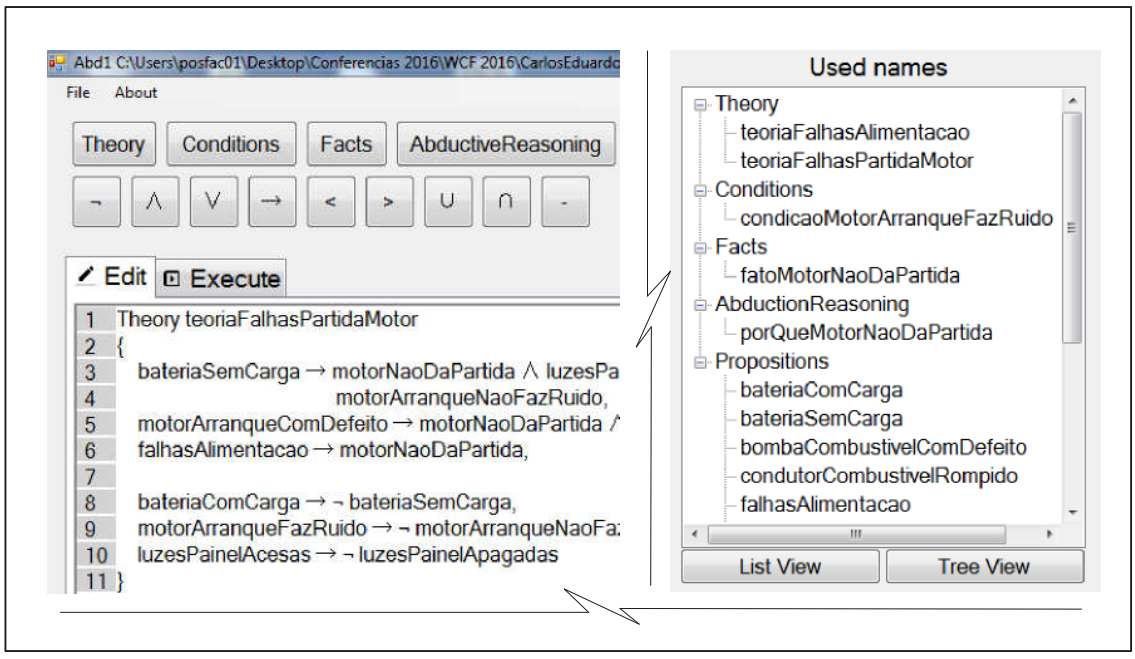

Figura 3. Ambiente integrado de edição, compilação e execução de programas Abd1.

A edição de programas pode ser feita digitando-se livremente textos na área de edição de programas (aba *Edit*). *Templates* para as principais construções sintáticas da Abd1 podem ser obtidos acionando-se os botões *Theory*, *Conditions*, *Facts* e *AbductiveReasoning*, disponíveis acima da aba *Edit*. Símbolos especiais, não

[4] Todo AIDEP-Abd1 (editor, compilador, interface, autômato executor de raciocínios etc.) foi desenvolvido em Microsoft .NET usando a linguagem C#. O autômato executor de raciocínios utiliza, para inferência das hipóteses abdutivas, o algoritmo Peirce proposto em Rodrigues, Oliveira & Oliveira (2014).

disponíveis em teclados convencionais também podem ser obtidos acionando-se botões acima da aba *Edit*.

A área de execução de raciocínios (aba *Execute*) apresenta um sistema interativo onde raciocínios podem ser digitados e executados. A Figura 2 contém exemplos destas interações em que, o sistema solicita a digitação de um raciocínio (*Question*), o raciocínio é digitado e, ao ser solicitado a execução do raciocínio, o sistema responde com um conjunto de possíveis hipóteses abdutivas (*Hypotheses*).

A área de acompanhamento de nomes expõe na caixa *Used names* os nomes das teorias, condições, fatos, raciocínios abdutivos etc. declarados no programa. Este é um importante recurso para o programador administrar os nomes criados e também facilita a digitação de nomes nas áreas de edição e de execução de raciocínios: um *click* duplo sobre um nome na caixa *Used names* faz com o nome seja escrito no corpo do programa (aba *Edit*) ou na linha de execução (aba *Excute*).

4. Conclusões e Trabalhos Futuros

Um programa na linguagem Abd1 é escrito por meio dos conceitos de teoria, condições, fatos e raciocínios abdutivos. Diferentes teorias podem ser propostas para diferentes perspectivas sobre as quais um fenômeno pode ser entendido. A possibilidade de reunir em um raciocínio duas ou mais teorias cria facilidade para investigar um fenômeno sob concepções que se contrapõem ou se combinam de modo a oferecer visões que variam em generalidade, em abrangência e em conceitos. O emprego de condições permite facilmente compor algo frequente em investigações que requerem raciocínios abdutivos: a presença ou a ausência de certos contextos ou circunstâncias.

Em Abd1 teorias, fatos e condições são tratados como conjuntos e o resultado de um raciocínio, as hipóteses, também formam um conjunto. Isto é conveniente porque traz simplicidade conceitual para o programa que a emprega e, também, porque possibilita a descrição de raciocínios mais complexos a partir de operações (operações sobre conjuntos) envolvendo outros raciocínios.

A linguagem Abd1 foi desenvolvida dentro de um contexto de pesquisa sobre raciocínios abdutivos. Finalmente, existe um ambiente que a implementa e ela pode ser empregada em estudos teóricos e experimentais que (1) busquem compreender melhor as necessidades linguísticas de sistemas para modelagem de raciocínios abdutivos por não especialistas em programação lógica, e (2) avaliem a efetividade da Abd1 dentro de um contexto potencial de uso.

Referências

Abdennadher, S. & Christiansen, H. (2000) An Experimental CLP Platform for Integrity Constraints and Abduction, In H. Larsen, J. Kacprzyk, S. Zadrozny, T. Andreasen & T. Christiansen (eds.), *Proceedings of the FQAS* 2000, Flexible Query Answering Systems, pp. 141–152.

Campbell, T. & Oh, P. (2015) Engaging Students in Modeling as an Epistemic Practice of Science, *Journal of Science Education and Technology*, v. 24, n. 2, pp. 125–131.

Caroprese, L. & Trubitsyna, I. (2014) A Measure of Arbitrariness in Abductive Explanations. *Theory and Practice of Logic Programming*, v. 14, n. 4–5, pp. 1–25.

Denecker, M. & Kakas, A. C. (2002) Abduction in Logic Programming. In A. C. Kakas & F. Sadri (eds.), *Computational Logic: logic programming and beyond: essays*, LNCS 2407, Springer-Verlag, pp. 402–436.

Frühwirth, T. (1998) Theory and Practice of Constraint Handling Rules, *The Journal of Logic Programming*, v. 37, n. 3, 95–138.

Gavanelli, M., Lamma, E., Riguzzi, F., Bellodi, E., Zese, R. & Cota, G. (2015) An Abductive Framework for Datalog Ontologies. In M. De Vos, T. Eiter, Y. Lierler & F. Toni (eds.), *Proceedings of the Technical Communications of the 31st International Conference on Logic Programming* (ICLP 2015), pp. 1–13.

Helikar, T., Cutucache, C. E., Dahlquist, L. M., Herek, T. A., Larson, J. J. & Rogers, J. A. (2015), Integrating Interactive Computational Modeling in Biology Curricula, *PLOS Computational Biology*, v. 11, n. 3, e1004131.

Jaffar, J. & Maher, M. J. (1994) Constraint Logic Programming: a survey, *Journal of Logic Programming*, v. 19–20, pp. 503–581.

Josephson, J. R. & Josephson, S. G. (1994) *Abductive Inference: Computation, Philosophy, Technology*. Cambridge University Press.

Kakas, A. C., Kowalski, R. A. & Toni, F. (1993) Abductive Logic Programming, *Journal of Logic and Computation*, v. 2, n. 6, pp. 719–770.

Kakas, A. C., Michael, A. & Mourlas, C. (2000) ACLP: Abductive Constraint Logic Programming, *Journal of Logic Programming*, v. 44, n. 1–3, pp. 129–177.

Nicoletti, M. C. (2010) *A Cartilha da Lógica*, EdUFSCar.

Oliveira, C. E. A. (2016). *Abd1: um protótipo de linguagem especifica para programação de raciocínios abdutivos*, Dissertação de mestrado, PMCC Faccamp, Campo Limpo Paulista.

Rodrigues, F., Oliveira, C. E. A. & Oliveira, O. L. (2014) Peirce: an algorithm for abductive reasoning operating with a quaternary reasoning framework. *Research in Computer Science*, v. 82, pp. 53–66.