Confluence Analysis of Cognitive Models with Constraint Handling Rules

Daniel Gall, Ulm University

1 Introduction

This paper gives a brief summary of how logic programming (in the form of Constraint Handling Rules) can be used to formalize and analyze cognitive models. This mainly is a summary of the author's PhD topic.

Computational cognitive modeling

Computational cognitive modeling is an interdisciplinary field at the interface of psychology and computer science trying to explore the "essence of cognition (including motivation, emotion, perception, etc.) and various cognitive functionalities through developing detailed, process-based understanding by specifying corresponding computational models (in a broad sense) of representations, mechanisms, and processes." [Sun08]

State of the art

Currently, computational cognitive modeling architectures as well as the implementations of cognitive models are typically ad-hoc constructs. There are many variants of architectures to support modeling for a certain domain. The architectures lack a formalization from the computer science point of view. This impedes analysis of the underlying languages and the programmed models. It makes it hard to compare different implementation variants of the languages. It makes it hard to verify properties of the models.

Program Analysis with CHR

Constraint Handling Rules (CHR) is a declarative programming language originating from Constraint Logic Programming. Its strong relation to logic, supports analysis of CHR programs. Hence, there are many theoretical results of program analysis for CHR.

CHR as lingua franca

Furthermore, due to its elegance and analysis tools, many other rule-based formalisms and programming languages such as Production Rule Systems, Logical Algorithms, Business Rules, Term Rewriting Systems or Functional Programming have been embedded in CHR. CHR then serves as a common formalism to analyze and compare those languages and formalisms.

Objective

In this work, we present how cognitive models can be *formalized* and *analyzed* with the help of logic programming in form of CHR. We concentrate on confluence analysis of cognitive models in the popular cognitive architecture *Adaptive Control of Thought – Rational* [And+04].

2 Adaptive Control of Thought – Rational (ACT-R)

ACT-R is in its core a modular production rule system. The architecture is illustrated in Figure 1 It divides psychological knowledge into *declarative* and *procedural* knowledge.

Declarative knowledge

Declarative knowledge is represented as a network of so-called *chunks* that represent information units. Chunks are the central data structure of ACT-R. The network of chunks is a labeled graph where the nodes are the chunks. The chunks are typed and the types define the available connections to other chunks.

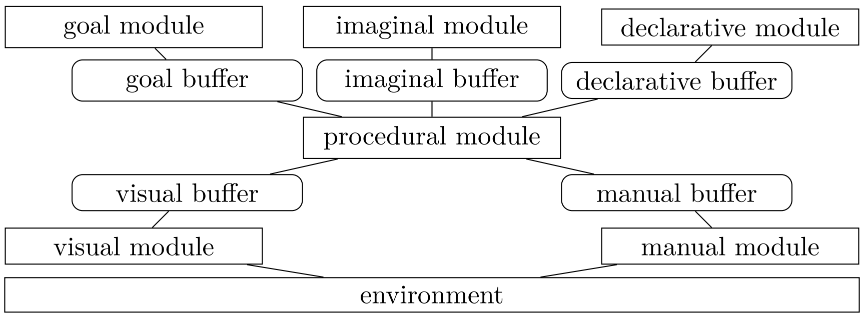

Figure 1: Modular architecture of ACT-R [And+04]

Procedural knowledge

Procedural knowledge is represented as if-then rules that match the current state of the working memory. The working memory consists the chunks in the *buffers* (c.f. Figure 1) and each buffer can hold at most one chunk at a time. The procedural module can request the *modules* to put a new chunk into their buffers.

Symbolic and sub-symbolic layers

The production rule system of ACT-R that implements a *symbolic* theory of cognition is enhanced by a *sub-symbolic layer* that adds features from connectionist models (e.g. neural networks) to model learning behavior, decision making and adaptation to different situations. For instance, the rules are enriched with a *utility value* that represents a rule priority and is adapted during the execution of the model. This allows the model to adapt the conflict resolution mechanism depending on prior experiences. A similar mechanism is the *activation* of chunks that allows for modeling latencies of recalls or even forgetting of facts.

Problems

ACT-R has a well-defined psychological theory and there are many domain-specific models using the architecture.However, there are some problems with the architecture: There is no formal operational semantics [GF15a; AGW14]. Therefore, there are no theoretical or practical results for computational analysis like termination, confluence or complexity.

3 Confluence Analysis of Cognitive Models

Confluence is the property that for a given start state, a program yields the same result independent of the rules chosen during the computation. This means that for all states that may lead to different successor states, there are computation paths that yield the same result (see Figure 2).

$$\begin{tikzcd}
 & S \arrow[dl, "*"'] \arrow[dr, "*"] & \\
S_1 \arrow[r, "*"'] & S' & S_2 \arrow[l, "*"]
\end{tikzcd}$$

Figure 2: Confluence diagram for a state S. In arbitrary many steps the states S_1 and S_2 can be reached. Those states join to the common state S'.

In ACT-R models, the conflict resolution mechanism chosen by the modeler eliminates any non-determinism in the choice of rules. However, the results depend on the chosen conflict resolution mechanism and its parameters as well as for instance data the model is presented during its execution.

On the other hand, the architecture of ACT-R typically leads to many rules with overlaps, i.e. where the conflict resolution mechanism becomes active at runtime. To gain control over the conflict resolution it is important to identify all possible rule conflicts and if they lead to different results.

Typically, cognitive models are not confluent by design. However, there are many rules in those models that should not interfere with each other. For instance, in many models artificial states are encoded in chunks that take care of a sequential execution of certain steps. To ensure model quality it is important to identify the rules that conflict with each other and to ensure that certain rules lead to deterministic (intermediate) states. Thereby, the modeler gains a higher level of control of the program flow and can ensure that the rule conflicts are desired, for instance because they are part of the learning process of the model.

4 Solution with Constraint Handling Rules

Use CHR analysis

CHR has a decidable confluence test that we want to use to implement a confluence test for ACT-R. The confluence test of CHR is a constructive approach that returns all sources of non-confluence in form of states that lead to differing final states.

General approach

The idea of our approach is illustrated in Figure 3. The main idea is to first develop an abstract formal semantics of ACT-R [GF15b; GF15a]. Then, a source-to-source transformation of ACT-R models to CHR programs can be developed [GF14; GF16]. The transformation has been shown to be sound and complete w.r.t. the formal semantics. Then the confluence test of CHR can be used.

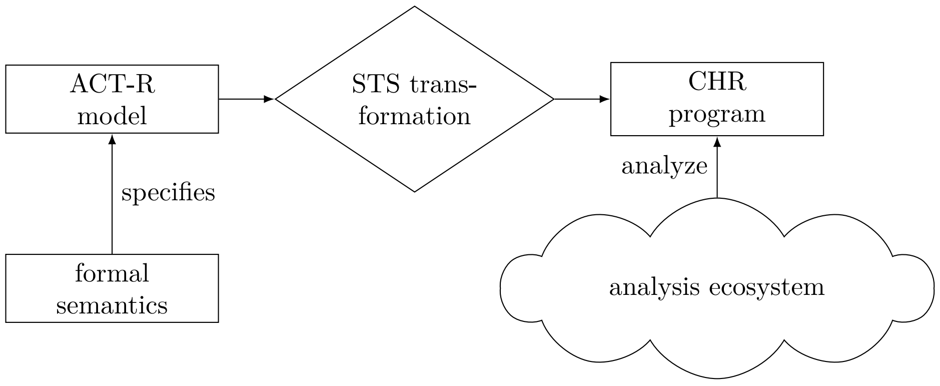

Figure 3: Overview of our approach

Formalization

The formal semantics is an abstract formalization of ACT-R that captures as many ACT-R implementations as possible. It abstracts from many details and technical artifacts from the various implementations [GF15b]. For instance, the semantics is independent of the conflict resolution mechanism by design, as we are interested in all possible rule overlaps that could appear at runtime as stated above. We have shown that including features from actual ACT-R implementations leads to a semantics that is an instance of our abstract formalization, i.e. every transition that is possible in the implementation is possible in the abstract semantics.

We have shown that our abstract formulation is an instance of the semantics proposed in [AGW14] independently from our work. We have proposed some improvements of this semantics in [GF15a].

Relation of transformation and semantics

Due to the strong relation of CHR to logic and the power of logic programming, the source-to-source transformation is closely related to the formulation of the semantics which supported the soundness and completeness results. The source-to-source transformation to CHR immediately provides an implementation of the abstract formulation of ACT-R. The transformed CHR program matches the behavior of the cognitive model. The confluence analysis results of CHR can be used on that translation and transferred to the original model.

Invariant-based confluence

However, the confluence test of CHR cannot be used directly for the transformed models [GF17], as it also considers states as sources of non-confluence that can never appear in an ACT-R model. For instance, if a buffer holds two chunks at a time, this is not allowed in ACT-R, but nevertheless included in the confluence test. Therefore, an invariant has been defined that specifies the valid state space of ACT-R models. Then the results on invariant-based or observable confluence for CHR can be used [DSS07; Rai10]. In general, invariant-based confluence is undecidable depending on the invariant. It has been shown that in the case of the ACT-R invariant, the confluence test remains decidable.

5 Lessons Learned

Generalization of the approach

Due to the role of CHR as a lingua franca of declarative programming, the approach can be generalized to many other formalisms and languages. Since there are numerous analysis tools for CHR, the overhead of developing a source-to-source transformation and proving soundness and completeness quickly pays. The many available embeddings of other rule-based approaches show that CHR is suitable for a use as an analytical platform. In [RF11] the approach has been executed successfully for Graph Transformation Systems.

CHR implementation

As a result of its close relation to first-order logic, CHR has proven as a useful language to implement a formally specified operational semantics of a production rule system and arguably many other declarative languages and formalisms. The ACT-R implementation of CHR is close to the formal semantics and can be seen as a direct implementation of it.

Abstraction

This implementation is a model of the procedural core of ACT-R that still abstracts from the actual implementations. For instance, the requests to modules are simply represented by built-in constraints that get a certain input and yield the result the corresponding ACT-R module would yield. This corresponds to the modularity of the ACT-R cognitive theory.

Modularity of confluence test

However, for this reason module requests appear to be an obstruction to confluence, as in general one has to assume that two identical requests may yield the different results. The more knowledge of the particular modules is available, the better gets the confluence test. However, even without such knowledge, the confluence test will identify such rules as possible sources of non-confluence and the modeler can consider the problem in the implementation. For instance, the test may unveil side conditions such as that the request to the declarative memory has to be deterministic (i.e. there is only one matching chunk for the request in every case) in order to maintain confluence.

6 Conclusion

Logic programming and its strengths in formal semantics and program analysis can enable program analysis in the field of cognitive modeling. The modularity of logic programs enables an isolated view on the procedural core of computational cognitive models. Implementations can still be expressed as instances of those abstract formalizations. The analysis methods of CHR can be used to analyze such models. Although this paper concentrates on confluence analysis, the results on operational equivalence of CHR are directly available by using the formulated invariant on ACT-R translations.

For the future, we want to specifically implement the refined semantics of ACT-R that is used in implementations using our formalizations on the basis of our work in [GF15a; GF18]. There we already have shown that the refinements can be seen as a further refinement of our formulation of the abstract semantics. The basic ideas of how the implementational details can be implemented in CHR are shown in [GF14; Gal13]. Furthermore, we are interested in how concepts of ACT-R and other cognitive architectures can be integrated in a clean way to CHR without inheriting the technical artifacts of the original implementations. This could yield a sound, formal, declarative and clean framework for implementing cognitive models.

References

[AGW14] Rebecca Albrecht, Michael Gießwein, and Bernd Westphal. "Towards formally founded ACT-R simulation and analysis". In: *Proceedings of the 12th Biannual conference of the German cognitive science society (Gesellschaft für Kognitionswissenschaft).* Vol. 15 (Suppl. 1). Cognitive Processing. Springer, 2014, pp. 27–28.

[And+04] John R. Anderson, Daniel Bothell, Michael D. Byrne, Scott Douglass, Christian Lebiere, and Yulin Qin. "An Integrated Theory of the Mind". In: *Psychological Review* 111.4 (2004), pp. 1036–1060. ISSN: 0033-295X.

[DSS07] Gregory J Duck, Peter J Stuckey, and Martin Sulzmann. "Observable confluence for constraint handling rules". In: *Logic Programming.* Springer, 2007, pp. 224–239.

[Gal13] Daniel Gall. "A Rule-Based Implementation of ACT-R Using Constraint Handling Rules". In: *Master Thesis, Ulm University* (2013).

[GF14] Daniel Gall and Thom Frühwirth. "Exchanging Conflict Resolution in an Adaptable Implementation of ACT-R". In: *Theory and Practice of Logic Programming* 14 (Special Issue 4-5 2014), pp. 525–538. ISSN: 1475-3081.

[GF15a] Daniel Gall and Thom Frühwirth. "A refined operational semantics for ACT-R: investigating the relations between different ACT-R formalizations". In: *Proceedings of the 17th International Symposium on Principles and Practice of Declarative Programming, Siena, Italy, July 14-16, 2015.* Ed. by Moreno Falaschi and Elvira Albert. ACM, 2015, pp. 114–124. ISBN: 978-1-4503-3516-4.

[GF15b] Daniel Gall and Thom W. Frühwirth. "A Formal Semantics for the Cognitive Architecture ACT-R". In: *Logic-Based Program Synthesis and Transformation - 24th International Symposium, LOPSTR 2014, Canterbury, UK, September 9-11, 2014. Revised Selected Papers.* Ed. by Maurizio Proietti and Hirohisa Seki. Vol. 8981. Lecture Notes in Computer Science. Springer, 2015, pp. 74–91. ISBN: 978-3-319-17821-9.

[GF16] Daniel Gall and Thom Frühwirth. "Translation of Cognitive Models from ACT-R to Constraint Handling Rules". In: *Rule Technologies. Research, Tools, and Applications: 10th International Symposium, RuleML 2016, Stony Brook, NY, USA, July 6-9, 2016. Proceedings.* Ed. by Jose Julio Alferes, Leopoldo Bertossi, Guido Governatori, Paul Fodor, and Dumitru Roman. Springer International Publishing, 2016, pp. 223–237. ISBN: 978-3-319-42019-6.

[GF17] Daniel Gall and Thom Frühwirth. "A Decidable Confluence Test for Cognitive Models in ACT-R". In: *Rules and Reasoning.* Ed. by Stefania Costantini, Enrico Franconi, William Van Woensel, Roman Kontchakov, Fariba Sadri, and Dumitru Roman. Cham: Springer International Publishing, 2017, pp. 119–134. ISBN: 978-3-319-61252-2.

[GF18] D. Gall and T. Frühwirth. “An Operational Semantics for the Cognitive Architecture ACT-R and its Translation to Constraint Handling Rules”. In: *ArXiv e-prints, accepted for publication in the ACM Transactions on Computational Logic* (05/2018). arXiv: `1702.01606 [cs.LO]`.

[Rai10] Frank Raiser. “Graph Transformation Systems in Constraint Handling Rules: Improved Methods for Program Analysis”. PhD thesis. Germany: Ulm University, 2010.

[RF11] Frank Raiser and Thom Frühwirth. “Analysing Graph Transformation Systems Through Constraint Handling Rules”. In: *Theory Practice of Logic Programming* 11.1 (01/2011), pp. 65–109. ISSN: 1471-0684.

[Sun08] Ron Sun. “Introduction to Computational Cognitive Modeling”. In: *The Cambridge Handbook of Computational Psychology.* Ed. by Ron Sun. New York: Cambridge University Press, 2008, pp. 3–19.