数理解析研究所講究録
第1927巻 2014年 128-130

Towards Type System by Computer Algebra Systems in Programming Language (Extended Abstract)

Hisashi MIYASHITA
Cybernet Systems*

Tetsu YAMAGUCHI
Maplesoft†

Takashi IWAGAYA
Cybernet Systems‡

Gun EKI
Cybernet Systems§

Computer Algebra Systems (CAS) have been expanding the application area from fundamental science such as mathematics and physics to applied technologies such as mechanics, electronics, and nancial engineering. However, it is still under way to integrate CAS features with Programming Language (PL) features, and we presume it is because the differences between these two computation models are so large that we have not justi ed good motivations to bridge the gap.

We propose Hybrid System would be a common important challenge that the integration of CAS and PL requires since it typically requires CAS to describe continuous systems and PL to describe discrete systems. We are now trying to verify controller software with plant models in CAS by using our MapleSim modeling environment.

1 Introduction

Computer Algebra Systems (CAS) have been expanding the application area from fundamental science such as mathematics and physics to applied technologies such as mechanics, electronics, and nancial engineering. However, it is still under way to integrate CAS features with Programming Language (PL) features.

There have been a lot of researches to seek the possibility that computer systems may solve mathematical problems by exploiting the technologies of CAS and PL. Since CAS provides critical functionalities to translate, simplify, and solve mathematical expressions, mainly for proving theorems and solving logical conditions, many applications written in PL need to use them for such purposes. However, we have not yet successfully reached any common understanding on both computation models.

To the best of our knowledge, there have been at least three approaches to integrate CAS features on PL environment:

Believing Approach (Simple Integration) This approach [1] allows to use CAS functionalities from other programming environments. While we are able to use both features, we need to understand both programming models and translate the results of CAS to use them in the other environment under the trust in CAS. It may be a problem for more efficient and reliable computation. This approach is effective to use CAS as a solver because users have only to care about the truth of the results while the results are still unreliable.

himi@meadowy.org
†tetsuy@maplesoft.com
‡iwagaya@cybernet.co.jp
§g-eki@cybernet.co.jp

Autarkik Approach This approach reimplements CAS features in PL (or theorem prover) environment because proving mathematical theorems requires the full proofs for all of the algorithms in the environment, which is driven by Curry-Howard isomorphism [6] that ensures any typed programs be regarded as proofs. While this approach assures correctness of programs and their corresponding theorems, it is quite hard to implement efficient CAS algorithms with the proofs so that it would be too slow for mathematical computations [2].

Skeptical Approach Although Autarkik approach ensures that the results are proven to be correct, the computation without CAS is not efficient and lacking the important functionalities to manipulate mathematical expressions. In particular to the applications of systems of equations, it will be problematic for users to prove theorems. In Skeptical Approach, we can use CAS functionalities to transform mathematical expressions and then need to check the validity that the transformed expressions equal to the original expressions by using the theorem prover. Maple-Coq [2] adopts this approach for users to use both functions of Maple and Coq.

Still, all of the approaches are intended to use the functionalities of CAS and PL or optimize performance of theorem provers. We believe, however, we can invent innovations by fusing CAS and PL to open up new applications among the areas of mathematics, programming, and modeling. For example, if we can verify programs by using CAS functionalities, it will open up new possibilities to establish more efficient and productive programming environment.

We presume it is caused that the fundamental differences of the computation models between CAS and PL. In the algebraic view, CAS is based on ring or eld theories in their computation model in order to reduce or optimize input equations to efficient ones, while PL requires more primitive algebraic structures, e.g., typed lambda calculus is equivalent with Cartesian closed categories [3], which assumes much looser algebraic structure than ring and eld. Therefore, it is not typical to nd problems that require both levels of computation models. If some problems require exible computation models, they would use PL or category based theories, otherwise, they require only limited expressive power, they would use CAS for efficient computation.

The observations we discussed so far suggest that we need to nd out common important challenges for CAS and PL researchers to vigorously take up. In this paper, we propose *Hybrid Systems* as this challenge, which consists of CAS-based mathematical expressions and PL-based software. However, we believe that there would be other problems that needs both CAS and PL, but it should be future work.

2 Hybrid Systems

Hybrid system is a system consisting of elements behaving both continuously and discretely, which are typically called continuous and discrete systems, respectively. In many cases, continuous systems are physical systems, such as mechanical and electrical systems, while discrete systems are realized by using software.

Due to this characteristics, hybrid system is a motivating example that requires CAS to describe continuous systems in Differential Algebraic Equations (DAEs) and PL to describe discrete systems in software. Note that since physical systems in lumped element models can be described by using DAEs, which can be captured as ring algebra and thus we can exploit CAS to simulate such systems.

We are now trying to establish embedded software veri cation in hybrid systems by exploiting control-loop structure of such systems. We illustrate the schema of the veri cation in Figure 1. Since the most of the hybrid systems are forming control loops where controllers by software control plants by multi-domain physical elements such as mechanics and electronics, we can formalize the speci cation of the control software by using the formal speci cation of plant models.

We have been developing MapleSim [4] modeling environment to describe multi-domain plant models in formalized mathematical equations, and thus we are expecting to derive preconditions and postconditions of controller software in such equations. As stated in [5], these conditions can be translated into type system in PL and thus CAS is expected to be used for type checking of such software.

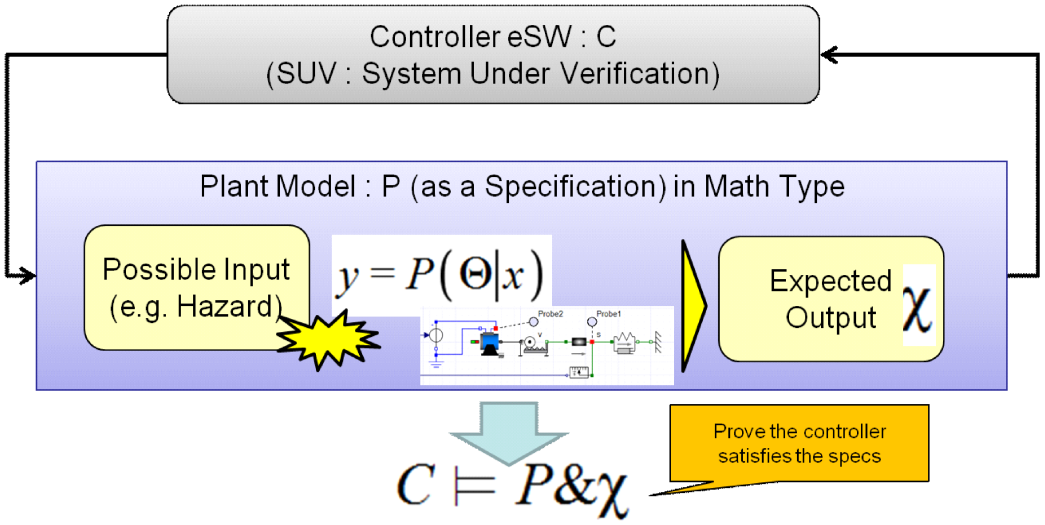

Figure 1: Verifying controller software by using plant models

3 Concluding Remarks

As we have discussed, the existing researches suggest that while the integration of CAS and PL is proposed to enrich the functionalities and optimize their computation, we have not yet established "killer applications" that critically need such integration.

We presume that this is caused by the chasm between algebraic structures between CAS, based on ring and eld theories, and PL, based on the looser algebraic structures such as category theory. Since it is a fundamental difference in their computation models, convincing reasons why we need the integration between CAS and PL are not so apparent.

Therefore, identifying valuable challenges that require both of the computation models would be an important next step to justify such integration.

We propose hybrid systems consisting of discrete (PL) and continuous (CAS) systems for this challenge, and now trying to use plant models in CAS to describe formal speci cations of controller software in control-loop systems. By using our MapleSim systems, we expect to reduce such plant models in DAE to mathematical equations and then verify controller software to establish new type systems for software.

References

[1] Andrew Adams, Martin Dunstan, Hanne Gottliebsen, Tom Kelsey, Ursula Martin, and Sam Owre. Computer algebra meets automated theorem proving: Integrating Maple and PVS. In *Theorem proving in higher order logics*, pages 27–42. Springer, 2001.

[2] David Delahaye and Micaela Mayero. Dealing with algebraic expressions over a eld in Coq using Maple. *J. Symb. Comput.*, 39(5):569–592, May 2005.

[3] Gérard Huet. Cartesian closed categories and lambda-calculus. In *Combinators and Functional Programming Languages*, pages 123–135. Springer, 1986.

[4] Maplesoft. MapleSim, http://www.maplesoft.com/products/maplesim/.

[5] Aleksandar Nanevski, Greg Morrisett, and Lars Birkedal. Hoare type theory, polymorphism and separation. *Journal of Functional Programming*, 18(5-6):865–911, 2008.

[6] Morten Heine Sørensen and Pawel Urzyczyn. *Lectures on the Curry-Howard isomorphism*, volume 149. Elsevier, 2006.