Agda プラグイン機構

池上大介 †
Daisuke IKEGAMI

† 産業技術総合研究所システム検証研究センター/科学技術振興機構 CREST
AIST Research Center for Verification and Semantics/Japan Science and Technology CREST
ikegami-daisuke@aist.go.jp

Martin-Löf の型理論に基づく対話型定理証明支援系である Agda のプラグイン機構について述べる。ここで、プラグイン機構とは Agda の中で外部のツールを起動し、証明や型検査を外部のツールに代行させるためのインターフェースを言う。プラグイン機構により、モデル検査器や自動定理証明器と Agda を接続することができる。Agda とモデル検査器や自動定理証明システムを組み合わせた統合的検証環境の構築が本研究の目的であり課題である。

1 はじめに

対話型定理証明支援系 Agda のプラグイン機構について述べる。ここで、プラグイン機構とは Agda の中で外部のツールを起動し、証明や型検査を代行させるインターフェースを言う。Agda (図 1) は Martin-Löf の型理論 [8] に基づく、対話型の証明・プログラムエディタである。

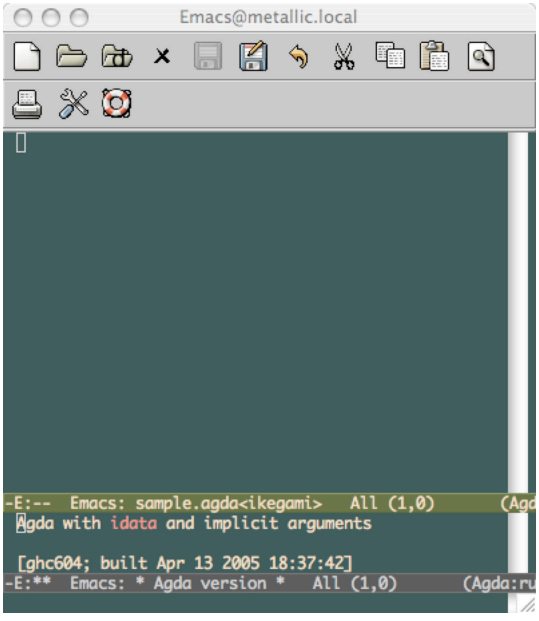

図 1: Agda

近年、プログラムの検証に対する技法として、定理証明/モデル検査が研究されている [9]。そこで、定理証明/モデル検査の長所を組み合わせた統合的な検証環境を作ることは自然な発想である。本研究では、定理証明器 Agda にプラグイン機構を追加した。これにより、モデル検査器や自動定理証明器と Agda を接続し、統合的な検証ツール (図 2) を作成することができる。Agda に基づく統合的検証環境の作成は今後の課題である。

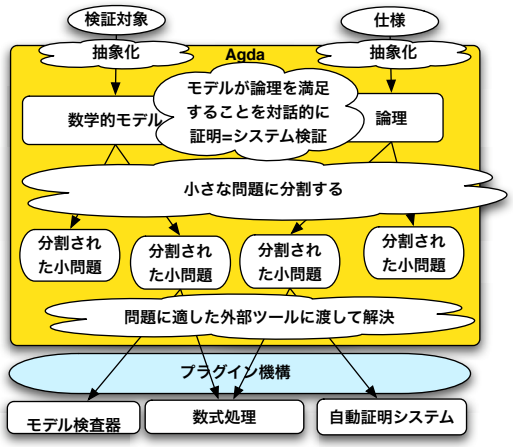

図 2: Agda を中核とする統合検証環境

第 2 節で Agda について説明する。その次の 3 節で Agda のプラグイン機構について述べる。第 4 節で関連研究を紹介し、本研究に関する考察を行う。最後にまとめと今後の課題を述べる。

2 対話型定理証明器 Agda とは

対話型定理証明器 Agda とは、対話型の証明・プログラムエディタである。Agda の証明/型検査は Martin-Löf の型理論に基づいている。Agda はスウ

ェーデン Chalmers 工科大学と産業技術総合研究所システム検証研究センターで共同開発が行われているオープンソース[1] のソフトウエアである。

Agda は Emacs エディタと協調して動作する。図 (3) は Agda の画面の説明である。Agda は Emacs

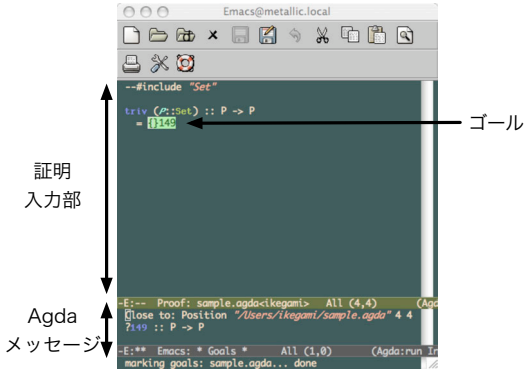

図 3: Agda (説明)

の agda-mode を環境とする。agda-mode では、証明を入力するバッファと Agda のメッセージを表示するバッファの 2 つの上下のバッファに分けられる。

Agda の特徴のひとつとして対話的な証明の検査を行うことが挙げられる。ユーザは、上の証明を入力するバッファに証明したい命題を記述する。まだ証明されていない命題は、その右辺にゴールと呼ぶ「穴」がつくられる。ユーザは Agda と対話しながらゴールを埋めていき証明を完成させる。例として、3 段階のユーザと Agda の対話を図 4 に示した。ユー

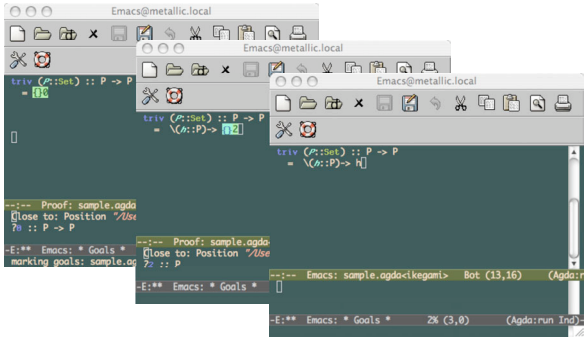

図 4: 対話的に証明を完成させる例

ザはゴールに証明の断片を入力し、Agda に問い合わせる。Agda はその断片が正しい推論ならば、次のゴールを生成する。逆に、ユーザが間違った推論をしたときは、Agda は間違っているというメッセージとその根拠を下のバッファに表示する。

3 Agda プラグイン機構とは

本節では Agda プラグイン機構について述べる。対話型定理証明支援系 Agda には現在本研究で作成したプラグイン機構と Martin Benke が作成したプラグイン機構の 2 種類のプラグイン機構が実装されている。前者の、本研究で作成したプラグイン機構は Agda 外部のツール (必ずしも Haskell で書かれているとは限らない、たとえば C や Java で書かれたツール) と接続するために作成した。一方、後者のプラグイン機構は、Agda に Haskell で書かれた機能を追加する目的で作成された。この 2 つのプラグイン機構により、Agda 本体のソースコードに手を加えることなく、Agda に機能を追加することが可能になる。

次に 2 種類のプラグイン機構のインターフェースについて説明する。本研究で作成したプラグインは、命題の対話的証明のゴールに “external” という予約語を入力することで、外部ツールの呼び出しを可能にする。文法は、以下の通りである。

```
external "pluginName [opts]" [args]
```

ここで、external の後ろに空白の後にダブルクオーテーションで囲まれたプラグイン名を記述する。プラグイン名の後ろに空白で区切られたオプション文字列を渡すことができる。また、ダブルクオーテーションで囲まれた文字列の後ろに空白で区切られた引数 (Agda で記述された項) を渡すことができる。

Agda は external で始まる文字列がゴールに与えられると、プラグインを駆動する。プラグインは True または False の二値と、さらに文字列を返すことが期待される。ここで、プラグインが返す文字列の書式は自由であり、ツールのログや、証明の反例などが期待される。Agda はこの文字列をユーザに告知するために Emacs の別のバッファ * Plugin Info * に表示する。

本研究で作成したプラグイン機構の活用例のひとつとして、NuSMV プラグイン (図 5) がある。NuSMV プラグインはモデル検査器 NuSMV [13] を Agda と接続する。NuSMV の機能の詳細については本論文の内容を越えるので説明しないが、NuSMV プラグインを用いることで Agda で行う命題の証明の中でモデル検査器 NuSMV を使うことができる。それに

[1] http://sourceforge.net/projects/agda/ でソースが公開されている。

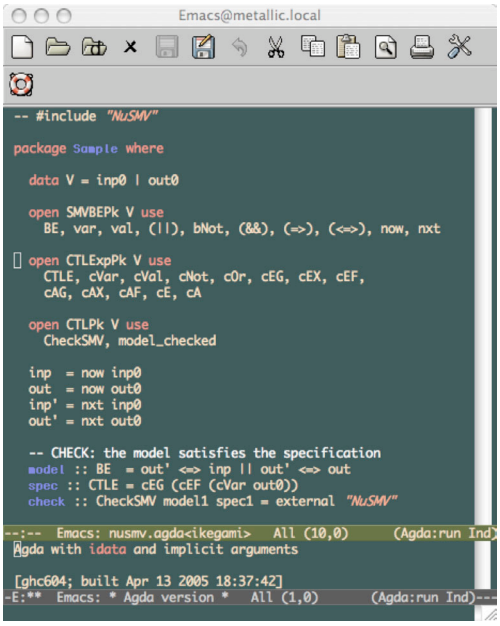

図 5: NuSMV プラグイン

より、証明を人間が記述する代わりに、状態全探索によって反例がみつからないことをモデル検査器を用いて自動的に検査することができる。

もうひとつのプラグインとして FOL プラグイン (図 6) がある。FOL プラグインは執筆時現在開発中

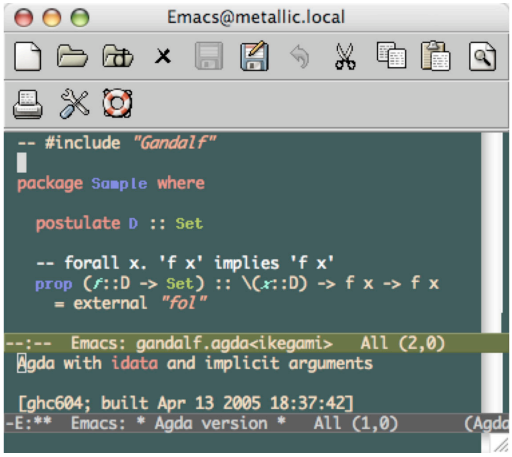

図 6: FOL プラグイン

である。FOL プラグインは一階述語論理の自動定理証明器と Agda を接続することを目的としている。一階述語論理の自動定理証明器はいくつか存在する (Gandalf [10], Paradox [14], etc.)。Agda とこれらの自動定理証明器を接続するために、プラグイン機構とともに Koen Classen が開発している翻訳器 Santa を用いる。Santa は一階述語論理の一般的な記述から、各ツールの入力ファイルを生成する翻訳フィルタである。Agda と自動定理証明については、関連研究を次の節で述べる。

一方、Benke のプラグイン機構の活用例のひとつとして、Agsy プラグインがある。Agsy プラグインは Agda の対話的証明支援において、一部の証明の自動化を行うものである。具体的には Control + ! (agda-auto) を打鍵すると、Agsy プラグインによる自動証明プログラムが起動し、証明が見つかったときはその証明をゴールに埋め込むものである。Agsy プラグインは Agda のソースコードに含まれている。

4 関連研究と考察

本節では関連研究を紹介するとともに、本研究に対する考察を行う。

4.1 Martin-Löf 型理論と、一階述語論理の自動定理証明

Andreas と Coquand と Norell は Martin-Löf 型理論に基づく型検査器と、一階述語論理の自動定理証明器を接続することを提案している [1]。一階述語論理の充足可能性を調べる問題は決定不能だが、にもかかわらず自動証明器を作成する研究が近年なされている [11]。対話型定理証明器は、証明の人手による記述量が長くなるのが問題である。自動定理証明器が証明の記述を代行することができれば、その問題は解決される。そのために Andreas らは、Martin-Löf の型理論を一階述語論理と健全であるように拡張し、さらに一階述語論理の自動定理証明器 gandalf [10] と接続できる型検査器のプロトタイプを作成した。

Andreas らの Martin-Löf 型理論の拡張は、定理証明器 Agda にも適用できる。本研究で作成した Agda のプラグイン機構を用いて一階述語論理の自動定理証明器と Agda を接続することにより、証明の記述量を軽減することができると期待される。

4.2 Coq の Maple mode

Coq [12] はフランスの Inria で開発されている対話的な定理証明器である。Darahaye と Mayero は Coq と数式処理システム Maple を接続するためのインターフェースを作成した [4,5]。数式処理システムの多くは定理証明のための機能を持っていない。Maple version 4. も定理証明のための機能を持って

いないため、計算の正しさを数式処理システムのみでは確かめることができない。例えば [5] では、Maple version 4. がある間違った推論「$a = 0$ から $1 = 0$ を導く」ことを紹介している。Maple version 4. では宣言 $a = 0$ の後に、両辺に $1/a$ をかけることで誤った式 $1 = 0$ を得ることができる。一方、定理証明システム Coq では $a = 0$ のときに $1/a$ が定義できないことを証明できる。その一方で、Coq は計算に時間がかかることが問題である。実際、Darahaye らは $1000 * 1000$ が Coq ではメモリー不足で計算できないことを紹介している。Maple では $1000 * 1000$ は 1 秒以内に計算できる。Darehaye らは Coq の中に Maple とのインターフェースを構成し、Coq で行う証明の中で Maple を呼ぶことができることを示した。これにより、Maple に不足している証明機能を補い、かつ Coq に不足している計算機能を追加することができる。

定理証明器 Agda においても計算に時間がかかることは Coq と同様である。実際、筆者の計算機では $1000 * 1000$ は Agda では 5 分以上を必要とした[2]。本研究で構築した定理証明器 Agda のプラグイン機構を、数式処理システムと接続するために用いれば、Coq と Maple で得られた知見と同様の知見が得られると考えられる。さらに、Agda の特徴である「プログラムと証明が同じ」という性質は、アルゴリズムの検証と実装を同時に行うために有利であろうと思われる。数式処理システムと Agda を接続するのは今後の課題である。

5 まとめと今後の課題

本稿では、以下の項目を述べた:

- プログラムの検証において、対話型の定理証明器 Agda とその他の検証技法 (自動証明/モデル検査) を組み合わせた統合的な検証手法を提案した。
- 対話型定理証明器 Agda のプラグイン機構について述べた。

現在の Agda のプラグイン機構は、Benke のプラグイン機構と作者のプラグイン機構について両方とも Agda の内部文法に関する知識が必要である。しかし、Agda の内部文法は複雑なので、文法の解説文書や簡易に翻訳するためのインターフェースを作成することが今後の課題である。

もうひとつの課題は、証明の理由をツールが返すことができるようにすることである。そのためには、ツールが返す証明の理由のオブジェクトを Agda の証明オブジェクトに翻訳する必要がある。異なるツール間の証明オブジェクトの変換は今後の研究対象である。

[2] そこで計算を打ち切ったため、計算が終了したかどうかは確認していない

謝辞

対話型定理証明器 Agda のプラグイン機構作成にあたりまして、スウェーデン Chalmers 工科大学の Catarina Coquand, Peter Dybjer, Benke Marcin ならびに産業技術総合研究所の武山誠氏に多大な指導および貢献をいただきましたことを深く感謝いたします。また、一階述語論理の自動定理証明器の作成にあたり、スウェーデン Chalmers 工科大学の Koen Claessen, Grégoire Hamon, Ulf Norell の各氏に多大な指導および貢献をいただきましたことをあわせて深く感謝いたします。

参考文献

[1] A. Abel, T. Coquand and U. Norell, Connecting a logical framework to a first-order logic prover (extended version). Technical report, Department of Computing Science, Chalmers University of Technology, Gothenburg, Sweden, 2005.

[2] Agda and Cover Project, `http://www.coverproject.org/AgdaPage/`

[3] L. Augustsson, Cayenne — a simple functional language with a powerful type system. `http://www.md.chelmers.se/~augustss/cayenne/`

[4] D. Delahaye and M. Mayero, A maple mode for Coq. `http://pauillac.inria.fr/coq/contribs/MapleMode.html`

[5] D. Delahaye and M. Mayero, Dealing with algebraic expressions over a field in Coq using Maple. In Journal of Symbolic Computation, special issue on the integration of automated reasoning and computer algebra systems, pp. 569–592, Vol. 39, No. 5, May 2005.

[6] T. Hallgren, Alfa — a successor of the proof editor ALF. `http://www.cs.chalmers.se/~hallgren/Alfa/`

[7] L. Magnusson and B. Nordstr m. The ALF proof editor and its proof engine. In Types for Proofs and Programs, volume 806 of LNCS, Springer, pages 213-237, Nijmegen, 1994.

[8] Bengt Nordström, Kent Petersson and Jan Smith, Martin-Löf type theory. In Handbook of Logic in

Computer Science, volume 5. Oxford University Press, Oct. 2000.

[9] K. Schneider, Verification of reactive systems – formal methods and algorithms, Springer, 2003.

[10] Tanel Tammet, Gandalf – an automated theorem proving system. `http://deepthought.ttu.se/it/gandalf/`

[11] The CADE ATP System Competition `http://www.cs.miami.enu/~tptp/CASC/`

[12] The Coq proof assistant `http://coq.inria.fr/`

[13] NuSMV – a new symbolic model checker `http://nusmv.irst.itc.it/`

[14] Paradox – a first-order logic model finder `http://www.math.chalmers.se/~koen/paradox/`