Applied Type System

Hongwei Xi

概要：Pure Type System フレームワーク (*PTS*) は型システムをデザイン/形式化する単純で一般的なアプローチを提供します。けれども依存型の存在を認めると、一般帰納、再帰型、作用 (例: 例外、参照、入出力)、などのような多くの実際のプログラミングの機能に *PTS* を適用させることが難しくなります。この論文では、実際のプログラミングの機能をサポートする型システムをたやすくデザイン/形式化できる、新しい Applied Type System (*ATS*) フレームワークを提案します。*ATS* の鍵となる突出した機能は、コンストラクトされて評価されるプログラムを含む動的な部分から、形作られて根拠となる型を含む静的な部分を完全に分離することにあります。この分離を用いると、*PTS* では許可されていましたが、プログラムが型の中に現われることは不可能になります。*ATS* の形式的な開発だけでなく、実用的なプログラミングのための型システムを作るフレームワークとして *ATS* を使ったいくつかの例も紹介します。
この翻訳の元論文は http://www.ats-lang.org/PAPER/ATS-types03.pdf です。

1. はじめに

Pure Type System フレームワーク (*PTS*) [Bar92] は型システムをデザイン/形式化する単純で一般的なアプローチを提供します。けれども依存型の存在を認めると、多くの実際のプログラミングの機能に *PTS* を適用させることが難しくなります。とりわけ、一般帰納 [CS87]、再帰型 [Men87]、作用 [HMST95]、例外 [HN88]、入出力が存在するとき *PTS* を使って純粋性を担保するためには、多大な努力が必要になることを私達は学びました。このような *PTS* の限界に対処するために、実際のプログラミングの機能をサポートする型システムをたやすくデザイン/形式化できる、新しい Applied Type System (*ATS*) フレームワークを提案します。*ATS* の鍵となる突出した機能は、コンストラクトされて評価されるプログラムを含む動的な部分から、形作られて根拠となる型を含む静的な部分を完全に分離することにあります。この分離は Dependent ML (DML) [XP99,Xi98] で開発された制限された依存型の成果に由来していて、参照や例外のような作用の存在下でも依存型を柔軟にサポートします。また、ガード型 (*guarded types*) と アサート型 (*asserting types*) という 2 つの新しい (馴染みのない) 型を導入することで、*ATS* では *PTS* よりも柔軟に効果的にプログラムの不変条件を捕捉することができることを示します。

ATS のデザインと形式化がこの論文の主な主張で、[Zen97,XP99,XCC03] と似たアイデアを使った研究成果です。*ATS* を使うと、*PTS* のある種生来の欠陥を乗り越えて、依存型の存在下で多くの一般的なプログラミングの機能をサポートする型システムをたやすく設計できます。私達は現在 *ATS* に基づいた型システムを持ち、(DML で開発されたような) 依存型のみではなく guarded recursive datatypes [XCC03] をもサポートする、型付き関数型プログラミング言語を設計/実装している最中です。Scheme が導入したようなアプローチ、つまり既存の言語に新しい言語構造を実装するような手法で、*ATS* を用いて多様な言語拡張をサポートできないか私達は探求しています。とりわけ、オブジェクト指向プログラミング [XCC03]、メタプログラミング [XCC03,CX03]、型クラス [XCC02] のようないくつかのプログラミングの機能をこの手法で扱えることを、私達は既に示しています。

論文の残りは次のような構成になっています。2 章では、*ATS* フレームワークの詳細な開発成果を示します。*ATS* で構成された汎用的な Applied Type System である ATS を形式化し、それから subject reduction と progress theorem を定義します。3 章では、*ATS* を拡張して、一般帰納、パターンマッチ、作用のような一般的で現実的なプログラミングの機能のいくつかに順応させます。4 章では、applied type system の興味深い例をいくつか示します。最後に、関連研究と開発の将来の可能性について紹介した後、結論を述べます。この論文はオンライン [Xi03] から入手できます。

2. Applied Type System

この章では Applied Type System (*ATS*) フレームワークの形式化を示します。ここでは *ATS* で形式化された型システムを表わすのに *applied type system* という用語を使います。この後の説明では、ATS を静的な要素 (statics) と動的な要素 (dynamics) から成る一般的な applied type system として定義します。直感的に、statics と dynamics はそれぞれ型とプログラムを扱っています。単純化のために、statics は簡単な純粋型付き言語であると仮定します。そしてこの言語の型を 種 (*sort*) という名前で呼ぶことにします。statics の項を 静的な項 (*static term*) と呼びます。また dynamics の項を 動的な項 (*dynamic term*) と呼びます。そして、特別な種 *type* の静的な項は dynamics の型として機能します。

2.1 Statics

xxx

謝辞

参考文献