Automatic Dimensional Inference

Mitchell Wand*

College of Computer Science
Northeastern University
360 Huntington Avenue, 161CN
Boston, MA 02115, USA
wand@corwin.ccs.northeastern.edu

Patrick O'Keefe

ICAD, Inc.
1000 Massachusetts Avenue
Cambridge, MA 02139

1. While there have been a number of proposals to integrate dimensional analysis into existing compilers [1, 7, 8, 9], it appears that no one has made the easy observation that dimensional analysis fits neatly into the pattern of ML-style type inference [4, 5, 6]. In this paper we show how to add dimensions to the simply-typed lambda calculus, and we show that every typable dimension-preserving term has a principal type. The principal type is unique up to a choice of basis.

2. Let us fix a number N of dimensions. Then we can define a set of types by:

$$
\begin{aligned}
\langle\text{type}\rangle &::= \langle\text{type-var}\rangle \mid \langle\text{base-type}\rangle \mid \langle\text{type}\rangle \rightarrow \langle\text{type}\rangle \\
&::= Q(\langle\text{num-exp}\rangle_1, \ldots, \langle\text{num-exp}\rangle_N) \\
\langle\text{num-exp}\rangle &::= \langle\text{number}\rangle \mid \langle\text{num-var}\rangle \\
&::= \langle\text{num-exp}\rangle + \langle\text{num-exp}\rangle \\
&::= \langle\text{num-exp}\rangle - \langle\text{num-exp}\rangle \\
&::= \langle\text{number}\rangle * \langle\text{num-exp}\rangle
\end{aligned}
$$

Here we have added a new set of types $Q(n_1, \ldots, n_N)$, representing dimensional quantities. If we set $N = 3$, and denote the dimensions using MLT in the usual way, then $Q(n_1, \ldots, n_N)$ represents the dimension usually denoted $M^{n_1} L^{n_2} T^{n_3}$.

*Work supported by the National Science Foundation under grant number CCR-8801591. This paper should be cited as: Wand, M. and O'Keefe, P.M. "Automatic Dimensional Inference," *Computational Logic: in honor of J. Alan Robinson* (J.-L. Lassez and G. D. Plotkin, eds.), MIT Press, (1991), pp. 479–486.

Equality on number expressions is interpreted on the rationals, and equality on type expressions is interpreted symbolically up to equality of number expressions. Note that that a $\langle$num-exp$\rangle$ is just a linear combination of the number variables. This is why the last production is $\langle$number$\rangle *$ $\langle$num-exp$\rangle$ instead of $\langle$num-exp$\rangle * \langle$num-exp$\rangle$. For the unification algorithm, it is important to note that this property is preserved under substitution.

We can now state the types of some important constants. We follow [8] in using the names of units to denote dimensions:

$$\begin{aligned} \mathbf{kg} &: Q(1,0,0) \\ \mathbf{m} &: Q(0,1,0) \\ \mathbf{sec} &: Q(0,0,1) \end{aligned}$$

The intention is that **kg** denotes 1 kilogram, **m** denotes 1 meter, etc. Numerical constants are dimensionless (that is, of type $Q(0,0,0)$), so that $3 * \mathbf{sec}$ denotes the dimensional quantity "3 seconds." A full language proposal might, of course, be engineered so that the "$*$" is unnecessary.

The arithmetic operators have polymorphic types in this system, for example:

$$\begin{aligned} + &: (\forall i,j,k)(Q(i,j,k) \to Q(i,j,k) \to Q(i,j,k)) \\ * &: (\forall i,i',j,j',k,k')(Q(i,j,k) \to Q(i',j',k') \to Q(i+i',j+j',k+k')) \\ \sqrt{(-)} &: (\forall i,j,k)(Q(i,j,k) \to Q(0.5*i, 0.5*j, 0.5*k)) \end{aligned}$$

In general, any fixed exponent $(-)^n$ will have type $(\forall i,j,k)(Q(i,j,k) \to Q(n*i, n*j, n*k))$. We consider the case of general exponentiation and other transcendental functions below.

3. For our language, we choose the ordinary simply-typed lambda calculus with these types. For this language, we have the standard result that typability reduces to satisfiability:

Theorem 1 *Given any expression M in the untyped lambda calculus, any type t_0, and any set A_0 of type assumptions (associating types with identifiers) with $\operatorname{dom} A_0 = \operatorname{fv} M$, we can effectively construct a set E of equations such that for any type t and any set B of type assumptions with $\operatorname{dom} B = \operatorname{fv} M$, $B \vdash M : t$ iff there exists a substitution μ (associating types with type variables) such that*

$$(\mu \models E \wedge B = A_0\mu \wedge t = t_0\mu)$$

Here we model the polymorphism of $+$ and $*$ in the usual way, by ascribing to each occurrence of $+$ and $*$ a different instance (using fresh number variables) of its polymorphic type.

4. This result gives us a set of equations between type expressions and number expressions to be solved. To do this, we first solve the type equations, using the usual unification algorithm. Once all the type equations are eliminated, we are left with a set of equations between linear combinations of number variables. But these can be solved by linear algebra. There are three possible outcomes: (1) there may be no solution, meaning that there is no typing for the original expression; (2) there may be a unique solution, giving a unique set of dimensions; or (3) there may be infinitely many solutions. In this case, the usual algorithms of linear algebra give us a basis (a linear combination of type variables with rational coefficients) in which all solutions can be expressed. Hence we get a principal type, unique up to change of basis.

Theorem 2 *The language of type- and number-expressions has a decidable most general unification problem.*

Proof: The algorithm is in two parts. First, we solve the type equations, using ordinary substitution. Then we solve the resulting number equations. In order to utilize the result of the unification algorithm in this context, we need to state the invariant which it preserves. We therefore also must give the unification algorithm itself (in somewhat more detail than we might like) in order to show the small modifications necessary and in order to allow the reader to confirm that the invariant is preserved.

The algorithm has as its input a set E_0 of equations and has as its output either a most general unifier for E_0 or a signal that E_0 has no solutions. The algorithm has two variables, a set E of equations and a substitution σ. The algorithm will satisfy the following invariant:

$$\begin{array}{ll} (1) & (\forall\mu)(\mu \models E_0 \Rightarrow (\exists\nu)(\nu \models E \wedge \sigma \circ \nu = \mu)) \\ (2) & (\forall\mu)(\mu \models E \Rightarrow \sigma \circ \mu \models E_0) \end{array}$$

The algorithm is given as follows:

Initialization: Set $E = E_0$ and σ to be the identity substitution on $\mathrm{vars}(E_0)$.

Clearly, the initialization establishes the invariant.

Loop Step: If E contains no type equations, go to *Gaussian Elimination Step* below. Otherwise, choose a type equation from E and transform E according to the following case analysis, which is just the usual unification algorithm augmented by case (3):

(1) The equation is of the form $\langle$base-type$\rangle$ = $\langle$base-type$\rangle$. If the two sides are identical, delete the equation. If they are not identical, halt: there is no solution. Since there is no solution for E, by the first clause of the invariant there is no solution for E_0.

(2) The equation is of the form $t_1 \rightarrow t_2 = u_1 \rightarrow u_2$. Delete this equation from E and add the two equations $t_1 = u_1$ and $t_2 = u_2$. This is the usual step in the conventional case.

(3) The equation is of the form $Q(e_1, \ldots, e_N) = Q(e'_1, \ldots, e'_N)$. Delete this equation from E and add the number equations $e_i = e'_i$ for $i = 1, \ldots, N$.

(4) The equation is of the form $v = t$ or $t = v$, where v is a type variable. If v appears in t, halt: there is no solution. Otherwise, replace σ by $\sigma \circ [v \leftarrow t]$, delete the equation from E, and then substitute t for all occurrences of v in the resulting set of equations.

(5) Any other equation between type-expressions has no solution. Halt and report no solution.

This is the usual unification algorithm, and it is easy to see that each step preserves the invariant, and decreases the usual halting measure $\langle$number of variables in E, number of symbols in E $\rangle$.

Gaussian Elimination Step: At this point we have a set of equations E between number expressions. This is a set of linear equations over the number variables, which can be solved by linear algebra. If the set has no solutions, then halt and report no solution. If the set has solutions, then linear algebra gives a substitution mapping the dependent variables to linear combinations of the independent variables (the basis). Every choice of values for the basis yields a solution to the equations, and every solution is of this form. Hence this gives a most general solution to the original equations, unique up to choice of basis. *QED*

5. For example, consider

$$\lambda xyz.(x - y * (z - 0.5 * sec)^2)$$

From this we can deduce that $z : T$, $y : M^\alpha L^\beta T^\gamma$, and $x : M^\alpha L^\beta T^{\gamma+2}$, so the entire term has type

$$(\forall \alpha, \beta\, \gamma)(M^{\alpha}L^{\beta}T^{\gamma+2} \to M^{\alpha}L^{\beta}T^{\gamma} \to T \to M^{\alpha}L^{\beta}T^{\gamma+2})$$

6. What about exponentiation and other transcendental functions? General exponentiation does not appear to fit neatly into our scheme, because it yields dependent types for the exponents. However, the fundamental theorem of dimensional analysis, the so-called Π theorem [2, 3], states that any dimensionally meaningful physical formula can be put in the form

$$C \times f(\Pi_1, \ldots \Pi_n)$$

where C is a constant, f is an arbitrary dimensionless function, and the Π_i are dimensionless products of physical quantities. Thus an exponential, if it appears in a physically meaningful formula, must be applied to a dimensionless quantity. Therefore it is adequate to assign the exponential function the type

$$e^{(-)} : Q(0,0,0) \to Q(0,0,0)$$

without restricting the physical expressiveness of the resulting system Similarly, the trigonometric functions may safely be assigned the dimensionless type

$$\sin : Q(0,0,0) \to Q(0,0,0)$$

7. We have stated the result in terms of the simply-typed lambda calculus, but it is easily adapted to include the ML **let** rule: to type **let** $x = N$ **in** M under type assumptions A, find the most general type of N and then universally quantify over all the type and number variables which appear in the type of N but not in A; the resulting type scheme is then associated with x in the type assumptions for M.

A more interesting extension is to add *local* dimensions. We add to the lambda-calculus a new construct $\mathbf{newdim}(M)$, with the following type inference rule:

$$\frac{A' \vdash M : Q(\overbrace{0, \ldots, 0}^{N}, 1) \to Q(n_1, \ldots, n_N, 0)}{A \vdash \mathbf{newdim}(M) : Q(n_1, \ldots, n_N)}$$

where A' is A with all the numerical types changed to add a trailing 0, and the type inference for M is done with one new dimension.

The intention is that M receives a unit quantity of the new dimension. For example, in

$$\mathbf{newdim}(\lambda\, apple.N)$$

apple denotes 1 apple in its scope. Thus

$$\mathbf{newdim}(\lambda\, apple.\, \mathbf{newdim}(\lambda\, orange.(2 * apple + 3 * orange)))$$

is illegal, as one cannot add apples and oranges. The restriction that M return 0 as its last dimension prevents the new dimension from escaping from its scope.

References

[1] Baldwin, G. "Implementation of Physical Units," *SIGPLAN Notices 22:8* (August, 1987), 45–50.

[2] Bridgman, P. W., *Dimensional Analysis*, Yale University Press, 1922.

[3] Buckingham, E., *Phys. Rev. 4* (1914), 345.

[4] Cardelli, L. "Basic Polymorphic Typechecking," *Polymorphism Newsletter 2*,1 (Jan, 1985). Also appeared as Computing Science Tech. Rep. 119, AT&T Bell Laboratories, Murray Hill, NJ.

[5] Damas, L., and Milner, R. "Principal Type-Schemes for Functional Programs," *Conf. Rec. Ninth Ann. ACM Symp. on Principles of Programming Languages* (1982) 207–212.

[6] Milner, R. "A Theory of Type Polymorphism in Programming," *J. Comp. & Sys. Sci. 17* (1978), 348–375.

[7] Dreiheller, A., Moerschbacher, M., and Mohr, B., "Programming Pascal with Physical Units," *SIGPLAN Notices 21:12*, (December, 1986), 114–123.

[8] Hilfinger, P.N., "An Ada Package for Dimensional Analysis," *ACM TOPLAS 10* (1988), 189–203.

[9] Maenner, R., "Strong Typing and Physical Units," *SIGPLAN Notices 21:3* (March, 1986), 11–20.