How to build your own analysis: Specifying datatypes in the TYPE section

» Back to overview

The type system of FULA is a restricted version of the ML type system. In essence it is a monomorphic type system, i.e. the types of static functions must not contain type variables (there are some built-in functions with polymorphic types though). To assure an efficient implementation FULA demands that every type you use in the declaration of functions is either a built-in type or a type name defined in the TYPE section.

PAG/WWW provides the following basic datatypes that you can use in your analysis:

snum	a signed integer,
bool	boolean values *true* and *false*,
str	a literal string,
Label	a label in the *while* program to analyze,
Var	a variable in the *while* program to analyze,
Proc	a procedure in the *while* program to analyze,
Expression	an expression in the *while* program to analyze (this can be an aexpression or a bexpression),
edges	the type of an outgoing edge in TRANSFER functions.

In the TYPE section you can define additional datatypes by deriving new types from the basic types. PAG/WWW allows you to derive five kinds of datatypes: sets, lists, functions, lattices and cross products.

A type definition has the form *typename* = *typedefinition*
where *typedefinition* is one of:

set (Id)	Power set of elements of type Id. The resulting type is a complete lattice, ordered by set inclusion, with the empty set **{}** as the bottom element and **all** (the set including all elements of Id) as the top element.
list (Id)	The resulting type is a set of lists of elements of type Id. Example: If your *while* program has two variables x and y, then list(Var) is the set of lists { [], [x], [y], [x,y], [y,x], [x,x], [y,y], [x,x,y] ... } Since there is no order in this set, **list(Id) is not a complete lattice!**
Id_1 * ... * Id_n	The cross product of types Id_1 * ... * Id_n Example: Reaching definitions use pairs of variables and labels to store defining labels. The resulting type is a complete lattice if all subtypes are complete lattices.
Id_A **->** Id_B	The set of functions from Id_A to Id_B. Example: Constant propagation uses a function `Var -> snum` to store known values of variables. The resulting type is a complete lattice if type Id_B (the destination type of the function) is a complete lattice.
flat (Id)	Constructs a complete lattice from Id by adding a top element **top** and a bottom element **bot** such that for all elements x in Id: x < **top** and x > **bot**. Elements of Id are not comparable.
lift (Id)	Constructs a complete lattice from Id by adding a top element **top** and a bottom element **bot** such that for all elements x in Id: x < **top** and x > **bot**. **Id must be a complete lattice** and the order of Id is preserved, such that for each x,y: if x < y in ID then **bot** < x < y < **top** in lift(Id).

Important: PAG/WWW does not support **flat** and **lift** types over the same base type. That is if you have already defined a type `x = flat(a)` you cannot define a type `y = lift(a)` in the same analysis.

PAG/WWW predefines these commonly used complex types:

LabelSet	= set (Label)	Sets of *while* labels
VarSet	= set (Var)	Sets of *while* variables
ProcSet	= set (Proc)	Sets of *while* procedures
ExpressionSet	= set (Expression)	Sets of *while* expressions
ExpressionList	= list (Expression)	Lists of *while* expressions

Next step

- Specifying the framework of your analysis in the **PROBLEM** section

Contents

Search