Pips Database Manager

October 3, 2024

To analyze and transform code in a modular way, each PIPS pass uses and generates many objects related to different modules. These objects are called resources and different passes can, or at least should, only communicate with them. They are known by the name of the module they are related to, e.g. `foo`, and by the name of their kind, e.g. `CODE` or `PRECONDITIONS`.

The passes, a.k.a. phases, are declared together with the resources in `pipsmake-rc.tex`. Library `pipsmake` is responsible for the global consistency and for the pass isolation: passes do not have to know about the context because `pipsmake` ensures that they are called with all they need.

The PIPS database manager provides time stamps to support the `pipsmake` consistency scheme and disk transfers to provide long-term storage. Thanks to this, the compilation process can be broken into several steps, which is very useful for large applications and for interactions with other tools or with users.

The PIPS database manager, `pipsdbm`, was first designed with resource lists. When PIPS was used to analyze interprocedurally hundreds of modules, the list-based data structure did not scale up and was replaced by a two-level hash-table scheme providing a much faster access to any resource using its owner name and its name.

The new data structure is described in `pipsdbm_private.pdf`. However, the old data structures and API are still used by the client library, `pipsmake`.

1 Old Database Structures

Database = `name:string x directory:string x resources:resource*`

The domain `database` *is managed by the pipsdbm and preprocessor libraries to describe the current state of a PIPS execution. This Newgen domain contains its name (*`name`*), the directory in which it was created which is also know as* workspace*, and the information which has been computed for the different modules,* `resources`*. Each piece of information is called a* `resource`*. In reality, the name of the database and the name of the directory used are directly linked. The name of the directory is the name of the base with the string* .DATABASE *attached as a suffix.*

An element of type `resource` *is added to the list* `resources` *of each objet calculed for this program by the phases, analyses or transformations of Pips.*

The library which makes use of this data structure is ***`pipsdbm`****. It should be named* ***`database-util`*** *if a consistent file naming rule were used across PIPS development.*

Resource = name:string x owner_name:string x status x time:int x file_time:int

The domain resource *is used by pipsdbm to describe any piece of information which might be calculated by PIPS for a module or a program. For each bit of information, the following fields must be known: its resource kind name, (*name*), the function to whom this resource is related to (*owner_name*), whether it is present in memory or stored in file (*status*), its logical creation date (*time*), and, potentially, the creation date (Unix) of the corresponding file (*file_time*).*

The kind name *of the resource is in fact a type and could have been defined as an enumerated type. It is out of concern for the simple and generic nature of pipsdbm that we have chosen to define it as a character string. It is this* name *which permits a pipsdbm to select the proper C function for reading, writing or freeing a resource.*

At any given moment, each resource is identified in a unique manner by its resource kind name, ***name****, and the name of its so-called* owner, ***owner_name****.*

We discussed for some time the utility of having an owner_type to specify what the resource refers to: a program (e.g. entities, the symbol table), a module (most other resources), a loop, an instruction, etc. We decided against it since up to this date, we only have those resources which may be attached to programs or modules, and this information may be deduced from the kind name *of the resource. In fact, we only have a few resources which may be attached to a program; these are the entities (and the* ***user_file****, but they are largely poorly treated for the time being). We also have summary information related to the whole program, e.g. the program precondition.*

Status = memory:string + file:string

The domain status *is used by pipsdbm to know if the resource concerned may either reside in memory, in which case it may be found in memory, if it is needed for a processus, or on disc, if it is composed of a file which is normally never loaded in memory. In the latter case, the sub-domain* file *gives the file name. The file name must be relative to the workspace, if it is found in this workspace, and must be absolute, if it is found outside of the workspace, in order to allow the operations* ***mv*** *and* ***cp -pr*** *in the workspace. If the resource is in memory, the sub-domaine* memory *contains a pointer towards this resource.*

How do resources in a file and file resources differ? It cannot be seen from this description. The list of the latter group is explicitely managed somewherelsee in pipsdbm. File resources cannot be loaded into memory. They are accessed through other files named after the resource kind. For instance, the file PRINTED_FILE contains the name of the file containing the print-out of a function.

The pool of resources called database is often called the PIPS workspace in PIPS litterature. It has been called a database initially because a database can be memory resident. The storage of a database, i.e. a workspace, on disk is called a PIPS database or workspace. The file structure of a workspace is described elsewhere (Fabien?). See for instance the workspace slide in the PPOPP 2010 PIPS tutorial.