PIPS: Mechanism for the Automatic Consistency Management and Phase Activation (pipsmake)

François Irigoin
Pierre Jouvelot
Rémi Triolet

CAI, École des mines de Paris

October 17, 2019

Introduction

The interprocedural and interactive nature of PIPS make necessary the management of pieces of information linked to a particular module and phase, i.e. an analysis or a transformation. These pieces of information, called resources, may be reused by future phases as long as they are consistent, else they must be recomputed. This is true for any compiler but worse when interprocedural analyses and transformations since inter-module dependencies are created.

The consistency management and recomputation of these pieces of information, i.e. resources, could have been spread out in each of the transformations and analyses of PIPS. It seemed much better to centralize in one library the management of dependencies between phases and the maintenance of the coherence between different pieces of information, i.e. resources.

The `pipsmake` library offers two principal entry points: `make` and `apply`. The first one is used to request a particular piece of information in a consistent state, without having to worry about the calculation of all the different resources which are necessary for its calculation. The second one is used to apply a particular compiler pass, also known as a *rule*. A rule includes not only the name of the pass, but also the list of resources necessary for the execution of the pass and the list of resources generated by the pass. A resource is linked to a module and to a kind: for instance the `CODE` of function `foo`.

When a given resource must be found or recomputed, the set of rules are scanned to find out which rule produces the resource, and which secondary resources are necessary to compute the requested resource.

When several rules allow the calculation of a particular resource, an activation mechanism is used to define the default rule. This is indispensible to be able to treat the recursive calls produced by the successive manifestation of rules without having to request too much information concerning user partameters.

By default, the first rule producing a particular resource is considered activated. When several rules are available to produce the same resource, this resource must be the only one produced by this rule. In this manner it is coherent to activate rules dynamically.

A set of rules and particular resources may be statically defined in a file named `pipsmake.rc`. The library `pipsmake` contains the modules which perform the reading of such a file to initialize a a set of rules in memory and to write to disc a set of rules in a format compatible with their future read.

This is automatically made possible by certain interactive PIPS interfaces, on condition that the available alternative rules as well as the resource they produce have an alias name.

We present successively the structures of the data used to store in memory a set of derivation rules, virtual resources and real resources.

1 Set of derivation rules

`Makefile` = `rules:rule* x active_phases:string*`

The domaine `makefile` *is used by the high-level driver to describe between the different Pips phases. A* `Makefile` *is a list of rules (*`rule`*), each rule describing one of the Pips phases. In other words, the* `Makefile` *gives the list of phases active at the present instant* `active_phases`*. Remember that each type of resource may possibly be produced by different phases, but that only one phase is usable at any given instant.*

Note, the new fonctionalities for multiple resource production imply an ambiguity concerning the notion of active rule; active nature of the active rules being possible for a subset of rules which they produce (in particular in the case of patially cyclical rules).

2 Definition of a particular rule

`Rule` = `phase:string x required:virtual_resource* x produced:virtual_resource*`
`x preserved:virtual_resource* x modified:virtual_resource* x pre_transformation:virtual_re`
`x post_transformation:virtual_resource*`

The domain `rule` *permits the description of the actions of the phases of Pips on the resources managed by pips-db. Each phase requires that certain resources by available disponibles (*`required`*), it begins by executing potential transformations (*`pre_transformation`*), then produces one or several resources (*`produced`*), and modifies others (*`modified`*). The difference between the resources produced and and those modified permit the driver to successively manifest the phases on the correct order. Post processing phases are stored in* `post_transformation`

The transformation phases act on the code of the modules, which generally implies that the information which describe this module are lost. Yet, certain

among them do transformations which are so minor that certain descriptions are preserved (`preserved`*). This is notably the case of the privatization which all these descriptions preserved. Here is a list of the Pips phases.*

parser *syntaxic and calculation analysis of the control graph,*

linker *editing of links,*

proper-effects *calculation of the proper effects of the instructions,*

cumulated-effects *calculation of the cumulated effects of the instructions,*

usedef *calculation of the used-def chains and the def-use chains,*

privatizer *variable privatization ,*

dgkennedy *calculation of the dependency graph with the* niveaux de Kennedy*,*

dgwolfe *calculation of the dependency graph with the* vecteurs de direction de Wolfe*,*

3 Definition of a Virtual Resource

The virtual resources are the variables which may be instantiated in a real resource or in a list of real resources.

Virtual_resource = `name:string x owner`

The domain `virtual_resource` *permits the designation of a resource read or modifie'd by a phase descibing precicely in addition to the nature of the resource (*`datum`*) if the resource accessed is that which is attatched to the module, a program, to the modules accessed by the module to which the phase is apples or that which called it (*`owner`*). Here is the list of all the resources which Pips may be able to calculate.*

source *file source of a module Fortran; result of the initialization;*

code *module code with structured control graph; result of the control and parsing action;*

entities *program entites; resultat of the initialisation, of the parser and the linker;*

callees *modules callede directely by a module; result of the linker;*

proper-effects *proper effect of the instructions for a module; the term* propre *signifies that thg effets of the instruction blocs produced (loops, tests, ...) are not taken into account; result of proper-effects;*

cumulated-effects *cumulated effect of the instructions for a module; the term* cumulé *signifies that the effects of the instruction blocs prodiced (loops, tests, ...) are taken into account; result ofe cumulated-effects;*

sdfi summary data flow information *of a module; it is an abstract of the cumlulated effets of the module's instruction bloc; produce an abstraction of the effets consist of the elimination of the effets of the local variables of the module and, in th ecase of tables, the globalization of each effect erasing the expressions of indices in the process; the result of cumulated-effects;*

chains use-def *and* def-use chains *of a module; result of usedef;*

dgkennedy *dependancy graph with the* niveaux de Kennedy*;*

dgwolfe *dependancy graph with the* dependance direction vectors *(Wolfe);*

Owner = { program , module , main , callees , callers , all , select , compilation_unit }

The domain owner *permits the precise description of the virtual dependency rule concerning which modules are attached, the ressources read, written, produced or preserved. This could be the module itself (*module*), the modules called by the module to which the phase is applied (*callees*) or which it calls* caller*), or all the modules of the programme in question(*all*). The program (*program*) itself in fact caracterizes a particular workspace an so indirecty the set of modules no which we wish to perform work. The name of a pogram is generally not automatically derived from the code source because we can easily wish to derive several versions of the same sequential original code and give a different name to each different version.*

This supplementary attribute of the dependancies allows the top-level driver to manage multiple calls made necessary by the interprocedural nature of Pips and the elimination of the auto-recursion of the database manager.

select is a fake owner, to be used to select (or activate) rules from other pipsmake rules. Should only be used with the bang rules?

4 Real Resources

The real resources correspond to a particular set of data produced by a module or a particular program by a particular phase. The virtual resources take their value among these real resources, but these derivation rules of pipsmake are still generic and therefore still defined in terms of de virtual resources.

Real_resource = resource_name:string x owner_name:string

The domain real_resource *is a private domain for pipsmake which serves to enable the manifestation of a set of virtual resources for a given program and a module.*