AlphaZ: Command Reference

October 20, 2015

Contents

1 AlphaZ Commands

This document provides the list of commands currently implemented in AlphaZ. The document is generated from a model specification of the commands located in `edu.csu.melange.alphaz.commands` plug-in. The implementation of commands described in this document can be found under `edu.csu.melange.alphaz.mde/src-gen`, and the script extension for compiler scripts are under `edu.csu.melange.alphaz.commands.scripts.mde/src-gen`.

The commands uses the following conventions:

- There is a single main implementation for each command. In this reference, all variations for convenience is listed, but it all calls the first method in each list in the end. Some specialized overloadings may change the data type. For example, String representation of affine functions may be parsed before passing along.
- boolean values are denoted as integers. true is equivalent to != 0.
- Some commands optionaly take a list of variables as inputs, indicated by the argument name `varList`. Multiple variables may be passsed as a comma delimited list, to apply the same command to all variables.
- `nodeID` is a unique ID given to each node in the AST, and can be seen using `PrintAST` command. The uniqueID is a multi-dimensional vector denoting the path from the root to the node where the values indicate that the x-th branch is taken. `nodeID` is used to specify a particular expression to apply the transformation, when there is no clear way to specify the target expression otherwise.

2 Basic

Commands for basic operations such as reading alphabets files and printing out different views of the program.

2.1 ASave

Outputs the program to the specified file with array syntax. When filename is not specified, saves as "original_filename-ASave.ab".

- `void ASave(Program program, String filename)`
- `void ASave(Program program)`

2.2 ASaveSystem

Outputs the result of AShow to the specified file. When filename is not specified, saves as "original_filename-ASave.ab".

- `void ASaveSystem(Program program, String system, String filename)`
- `void ASaveSystem(Program program, String system)`

2.3 AShow

Prints out the program using array notation. Prints out the entire program when target system is unspecified.

- `String AShow(Program program, String system)`
- `String AShow(Program program)`

2.4 Save

Saves the output of show to the specified file. When output filename is unspecified, saves to "original_filename-Show.ab".

- `void Save(Program program, String filename)`
- `void Save(Program program)`

2.5 SaveSystem

Saves the output of show to the specified file. When output filename is unspecified, saves to "original_filename-Show.ab".

- `void SaveSystem(Program program, String system, String filename)`
- `void SaveSystem(Program program, String system)`

2.6 Show

Pretty prints the program using alpha-purist syntax. Prints out the entire program when target system is unspecified.

- `String Show(Program program, String system)`
- `String Show(Program program)`

2.7 Normalize

Normalizes the program. Normalization rules are described in more detail at: `http://www.cs.colostate.edu/AlphaZ/wiki/doku.php?id=normalize`.

- `void Normalize(Program program)`

2.8 PrintAST

Prints out the AST of the program. Prints out the entire program when target system is unspecified.

- `String PrintAST(Program program, String system)`
- `String PrintAST(Program program)`

2.9 ReadAlphabets

Parses the given alphabets program and returns a Program object.

- `Program ReadAlphabets(String file)`

2.10 RenameSystem

Renames an AffineSystem.

- `void RenameSystem(Program program, String system, String newSystemName)`

2.11 RenameVariable

Renames a variable.

- `void RenameVariable(Program program, String system, String varName, String newVarName)`

2.12 RemoveUnusedVariables

Removes unused variables and equations. Unused variables are variables that is not used by the definition of output.

- `void RemoveUnusedVariables(Program program, String system)`
- `void RemoveUnusedVariables(Program program)`

3 Transformation

Transformations are commands that modify the program in some way.

3.1 CoB

Change of Basis transforms the domain of a variable to the image by the given function, while adding necessary dependence expressions to maintain the original semantics of the program. Details can be found at `http://www.cs.colostate.edu/AlphaZ/wiki/doku.php?id=change_of_basis`.

- `void CoB(Program program, String systemName, String varName, AffineFunction function)`
- `void CoB(Program program, String systemName, String varName, String function)`

3.2 ForceCoB

Force the change of basis even if the CoB is for input/output of a system.

- `void ForceCoB(Program program, String systemName, String targetName, AffineFunction function)`
- `void ForceCoB(Program program, String systemName, String targetName, String function)`

3.3 Split

Splits a variable into two, where the domain of two variables are disjoint and union of them is equivalent to the original domain of the variable. If newName is not given, it creates a variable with prefix "_split" appended to the original name.

- `void Split(Program program, String systemName, String varName, String newName, String sepDomain)`
- `void Split(Program program, String systemName, String varName, String sepDomain)`

3.4 Merge

Merge two variables (of name var1Name and var2Name) into a single one. These variables must both be locals and must have disjoint domains. The name of the merge of these two variables is "newName".

- `void Merge(Program program, String systemName, String var1Name, String var2Name, String newName)`

3.5 Inline

Inlines the nth(number) reference to inlineEq in the definition of targetEq once. If the number is not specified, all references to inlineEq are inlined.

- `void Inline(Program program, String systemName, String targetEq, String inlineEq, int number)`
- `void Inline(Program program, String systemName, String targetEq, String inlineEq)`

3.6 Simplify

Simplifies the program in multiple ways. The domains are simplified by simplifyInContext (a.k.a. gist). When the domain of a reduction body is a single point, reduction is removed.

- `void Simplify(Program prog, String system)`
- `void Simplify(Program prog)`

3.7 createFreeScheduler

Takes an alphabetes program and converts it to a program that computes the fastest possible schedule (free shcedule). This transformation DOES NOT preserve the original semantics, but instead creates a new Program object. Implemented by Alex Klein as a class project in cs560@spring11

- `Program createFreeScheduler(Program program, String newProgPrefix)`
- `Program createFreeScheduler(Program program)`

3.8 SplitUnion

Replace an expression that has unions of polyhedra as its context domain with a case expression that splits the union into multiple disjiont polyhedra.

- `void SplitUnion(Program prog, String nodeID)`

3.9 ApplySTMap

Applies sequence of CoBs using STMap given for each variable. This is identical to the pre-processing step of ScheduledC code generator, where all variables are mapped to a common space so that ordering of iterations can be defined. Assumes correct schedule [TODO:run verifier before applying the transformation once the verifier is fixed] All STMaps and MemoryMaps given for the system will also be transformed appropriately.

- `void ApplySTMap(Program program, String systemName)`

3.10 UniformizeInContext

Attempts to uniformize all dependencies when possible.

- `void UniformizeInContext(Program prog, String system, int excludeInputs)`
- `void UniformizeInContext(Program prog)`
- `void UniformizeInContext(Program prog, String system)`

3.11 InlineForce

Force the inline action even if there is a self loop dependence

- `void InlineForce(Program program, String systemName, String targetEq, String inlineEq, int number)`
- `void InlineForce(Program program, String systemName, String targetEq, String inlineEq)`

3.12 InlineAll

Inline all the possible equation in a system when there is no self loop dependence

- `void InlineAll(Program program, String systemName, String inlineEq)`

3.13 InlineAllForce

Inline all the possible equations in a system even with self loop dependence once

- `void InlineAllForce(Program program, String systemName, String inlineEq)`

3.14 InlineSubSystem

Inline the subsystem used by the UseEquation specified by nodeID

- `void InlineSubSystem(Program program, String systemName, String label)`

3.15 OutlineSubSystem

Outline a list of equations of a given system. These equation are defined through a list of variable ("listEquations") that corresponds to the variable of a StandardEquation or to the output of a UseEquation. [V1] The created use equation will have no extension domain, and only variable expressions as inputs.

- `void OutlineSubSystem(Program program, String system, String listEquations)`

3.16 AddLocal

Adds a local variable (of name specified by "nameLocal") for the expression "expr" and replace each occurrence of this expression in the considered system. "expr" can be specified through a expression of the system (located in "nodeID"), or through an external expression "expr" whose list of indexes is "lInds".

- `void AddLocal(Program program, String system, String nameLocal, String nodeID)`
- `void AddLocal(Program program, String system, String nameLocal, String expr, String lInds)`

3.17 AddLocalUnique

Replace only the expression (specified by the location "nodeID") by a new local variable (of name "nameLocal").

- `void AddLocalUnique(Program program, String system, String nameLocal, String nodeID)`

3.18 DetectReductions

Detects simple reductions in the program and exposes as reduce expressions.

- `void DetectReductions(Program prog, String system)`

3.19 Reduction

Transformations involving reduce expressions.

3.19.1 PermutationCaseReduce

Takes case expression inside reductions out side of the reduction. This transformation can be applied to three different targets, the entire program, an affine system, or an equation.

- `void PermutationCaseReduce(Program program, String systemName, String targetVar)`
- `void PermutationCaseReduce(Program program)`
- `void PermutationCaseReduce(Program program, String systemName)`

3.19.2 ReductionDecomposition

Decomposes a reduction spanning more than one dimensions to two reductions. Function f1 composed with f2 should match the original projection function. The target reduction is specified as the n-th occurrence in the rhs of an equation indexed from 0.

- `void ReductionDecomposition(Program program, String nodeID, String f1, String f2)`

3.19.3 SimplifyingReduction

Simplifies the specified reduction using the reuse specified. It does not check if the given reuse is correct.

- `void SimplifyingReduction(Program program, String system, String varName, String reuseVector)`
- `void SimplifyingReduction(Program program, String nodeID, String reuseVector)`

3.19.4 NormalizeReduction

Transforms reductions specified into normal form. The normal form of a reduction is when a reduce expression is the direct child of an equation.

- `void NormalizeReduction(Program program, String nodeID)`
- `void NormalizeReduction(Program program)`
- `void NormalizeReduction(Program program, String system, String equationName)`

3.19.5 FactorOutFromReduction

Factors out an operand of point-wise operations (specified with nodeID) from the reduction body. It checks if the operator is distributive over the reduction operator, but does not verify that the given expression is constant in context.

- `void FactorOutFromReduction(Program program, String nodeID)`

3.19.6 SplitReductionBody

Takes a nodeID of a reduce expression, reduce(op, f, expr), and transform it in to two reductions; reduce(op, f, D1 : expr) op reduce(op, f, D2 : expr) where D1 is the splitDomain, and D2 is the difference between the original expression domain of the reduction body and the splitDomain.

- `void SplitReductionBody(Program program, String nodeID, String splitDomain)`

3.19.7 TransformReductionBody

Given reduce expression, reduce(op, f, expr), transforms the reduction to reduce(op, f, image(Dexpr, T) : Tinv@expr). The domain is transformed to the image by T, and specified as restrict. All variable accesses are composed with the inverse of T, so that other variables are unchanged.

- `void TransformReductionBody(Program program, String nodeID, String T)`

3.19.8 SerializeReduction

Serializes a reduction using the dependencies inferred from the given schedule.

- `void SerializeReduction(Program program, String nodeID, String schedule)`
- `void SerializeReduction(Program program, String system, String var, String schedule)`

3.19.9 MergeReductions

Merges two reductions combined by a binary operator into one reduction (with case branches) if possible. It is possible if all the operators are the same and the projection functions matches.

- `void MergeReductions(Program program, String nodeID)`

4 CodeGen

Code generators from alphabets to other languages.

4.1 generateScheduledCode

Generates scheduledC code for a system using the TargetMapping specified for the system. Detailed options can be given through optional argument.

- `void generateScheduledCode(Program program, String system, CodeGenOptions options, String outDir, Boolean genVerifier)`
- `void generateScheduledCode(Program program, String system, String outDir)`
- `void generateScheduledCode(Program program, String system, String outDir, Boolean genVerifier)`
- `void generateScheduledCode(Program program, String system)`
- `void generateScheduledCode(Program program, String system, Boolean genVerifier)`
- `void generateScheduledCode(Program program, String system, CodeGenOptions options, String outDir)`

4.2 generateWriteC

Generates WriteC code for a system. TargetMapping is ignored except for MemoryMapping given for output variables. Detailed options can be given through optional argument.

- `void generateWriteC(Program program, String system, CodeGenOptions options, String outDir)`
- `void generateWriteC(Program program, String system, String outDir)`
- `void generateWriteC(Program program, String system)`

4.3 generateWrapper

Generates a wrapper code for the given sytem using the specified TargetMapping.

- `void generateWrapper(Program program, String system, CodeGenOptions options, String outDir)`
- `void generateWrapper(Program program, String system, String outDir)`
- `void generateWrapper(Program program, String system)`

4.4 generateVerificationCode

Generates code for verifying generated program. The code is generated using demand-driven code generator (WriteC).

- `void generateVerificationCode(Program program, String system, String outDir)`
- `void generateVerificationCode(Program program, String system)`

4.5 generateMakefile

Generates Makefile to compile generated code + wrapper.

- `void generateMakefile(Program program, String system, String outDir)`
- `void generateMakefile(Program program, String system)`

4.6 createCGOptionForWriteC

Creates instance of CodeGenOptions for WriteC using default values.

- `CodeGenOptions createCGOptionForWriteC()`

4.7 createCGOptionForScheduledC

Creates instance of CodeGenOptions for ScheduledC using default values.

- `CodeGenOptions createCGOptionForScheduledC()`

4.8 setCGOptionFlattenArrays

Specifies true/false (1/0) values for if the multi-dimensional arrays allocated should be flattened to 1D or not.

- `void setCGOptionFlattenArrays(CodeGenOptions CGoptions, int flatten)`

4.9 setCGOptionDisableNormalize_depreciated

Options to skip Normalize before generating code with the ScheduleC code generator. This is an option added as an workaround to scalability problems with Normalize, and will be removed in the future. Not normalizing may lead to significantly inefficient code.

- `void setCGOptionDisableNormalize_depreciated(CodeGenOptions options)`

4.10 createTiledCGOptionForScheduledC

Creates instance of CodeGenOptions for ScheduledC using default values for tiling.

- `TiledCodeGenOptions createTiledCGOptionForScheduledC()`

4.11 setTiledCGOptionOptimize

Applies optimizations to the tiled code generated using full-tile splitting of a selected statement group. Statement group is selected with heuristics that may not be accurate.

- `void setTiledCGOptionOptimize(TiledCodeGenOptions options, int optimize)`

4.12 generateScanC

code generator for a scan subsystem (a subsystem that contains only scan computations)

- `void generateScanC(Program program, String systemName, CodeGenOptions options, String outDir)`

5 Calculator

Calculator interface for directly manipulating polyhedral objects.

5.1 calc_readDomain

Reads domain in String and returns a Domain object.

- `Domain calc_readDomain(Domain paramDomain, String domain)`
- `Domain calc_readDomain(String domain)`

5.2 calc_readFunction

Reads affine function in String and returns an AffineFunction object.

- `AffineFunction calc_readFunction(Domain paramDomain, String function)`
- `AffineFunction calc_readFunction(String function)`

5.3 calc_inverse

Compute the inverse of the give affine function.

- `AffineFunction calc_inverse(AffineFunction function)`

5.4 calc_inverseInContext

Computes inverse of the given affine function, in the context of the given domain.

- `AffineFunction calc_inverseInContext(Domain domain, AffineFunction function)`

5.5 calc_compose

Returns a function, that computes function2@function1).

- `AffineFunction calc_compose(AffineFunction function1, AffineFunction function2)`

5.6 calc_intersection

Returns the intersection of two domains given.

- `Domain calc_intersection(Domain domain1, Domain domain2)`

5.7 calc_union

Returns the union of two domains given.

- `Domain calc_union(Domain domain1, Domain domain2)`

5.8 calc_join

Returns a function that computes function1@function2).

- `AffineFunction calc_join(AffineFunction function1, AffineFunction function2)`

5.9 calc_isEmpty

Returns true if the domain is empty.

- `Boolean calc_isEmpty(Domain domain)`

5.10 calc_isEquivalent

Returns true if the two domains/functions given are equivalent.

- `Boolean calc_isEquivalent(Domain domainA, Domain domainB)`
- `Boolean calc_isEquivalent(AffineFunction funcA, AffineFunction funcB)`

5.11 calc_image

Compute the image of the given domain by the given function.

- `Domain calc_image(AffineFunction function, Domain domain)`

5.12 calc_preImage

Compute the pre-image of the given domain by the given function.

- `Domain calc_preImage(AffineFunction function, Domain domain)`

5.13 calc_difference

Returns domainA / domainB.

- `Domain calc_difference(Domain domainA, Domain domainB)`

5.14 calc_simplifyInContext

Takes domain and context domain, and returns the domain with constraints that are redundant with the context removed.

- `Domain calc_simplifyInContext(Domain domain, Domain context)`

6 Analysis

Analyses are commands that give information about the program, without modifying the program.

6.1 BuildPRDG

Constructs a PRDG for the specified AffineSystem. Input variables are excluded from the PRDG default. To override this option, set the optional argument noInput to 0.

- `PRDG BuildPRDG(Program program, String systemName, int noInputs)`
- `PRDG BuildPRDG(Program program, String systemName)`

6.2 ExportPRDG

Exports the given PRDG as a dot file.

- `void ExportPRDG(PRDG prdg, String filename)`

6.3 CheckProgram

Performs uniqueness and completeness check of the program. The program is a valid alphabets program, if it passes this check. Details of the check can be found at `http://www.cs.colostate.edu/AlphaZ/wiki/doku.php?id=check_program`.

- `void CheckProgram(Program program)`

6.4 VerifyTargetMapping

Verifies the target mapping given to a system. The third input takes "NONE", "MIN", "MAX" to control the verbosity.

- `void VerifyTargetMapping(Program program, String system, String verbose)`

6.5 CheckSystem

CheckSystem is CheckProgram applied to the specified system.

- `void CheckSystem(Program program, String system)`

6.6 Farkas1DScheduler

Farkas mono-dimensional scheduler.

- `List<ScheduledStatement> Farkas1DScheduler(PRDG prdg)`

6.7 FarkasMDScheduler

Farkas multi-dimensional scheduler. Uses ISL implementation.

- `List<ScheduledStatement> FarkasMDScheduler(PRDG prdg)`

6.8 PlutoScheduler

Pluto scheduler. Uses implementation in ISL.

- `List<ScheduledStatement> PlutoScheduler(PRDG prdg)`

7 TargetMapping

Commands for specifying Target Mapping.

7.1 setSpaceTimeMap

Specifies a space time mapping of a variable at a certain level. By default (when there is no value for level), the space-time map is set up for the first level of targetMapping.

- `void setSpaceTimeMap(Program program, String system, int level, String varList, AffineFunction stMap)`
- `void setSpaceTimeMap(Program program, String system, int level, String varList, String stMap)`
- `void setSpaceTimeMap(Program program, String system, String varList, String stMap)`

7.2 setMemoryMap

Specifies a memory map for a variable. Some variables may share the same meory map and memory space, then they can be set up using one command by providing a list of variables.

- `void setMemoryMap(Program program, String system, String varList, String memorySpace, AffineFunction memoryMap, String modFactors)`
- `void setMemoryMap(Program program, String system, String varList, String memorySpace, String memoryMap, String modFactor)`
- `void setMemoryMap(Program program, String system, String varList, AffineFunction memoryMap, String modFactors)`
- `void setMemoryMap(Program program, String system, String varList, AffineFunction memoryMap)`
- `void setMemoryMap(Program program, String system, String varList, String memorySpace, AffineFunction memoryMap)`
- `void setMemoryMap(Program program, String system, String varList, String memorySpace, String memoryMap)`
- `void setMemoryMap(Program program, String system, String varList, String memoryMap)`

7.3 setMemorySpace

Specifies the memory space for a variable. Normally, each variable has a separate memory space. However, some variables may share the same memory space, and this can be set up once by providing a list of variable.

- `void setMemorySpace(Program program, String system, String space, String varList)`

7.4 setStatementOrdering

Specifies the ordering of the statements (alphabets variables) in the generated code. This is similar to adding an additional ordering dimensions to the last dimension of space-time mapping, and providing ordering information. AlphaZ provides an alternative to such specification by allowing the user to specify partial orderings between statements. A total order is deduced at the time of code generation.

- `void setStatementOrdering(Program program, String system, String predecessor, String successor)`

7.5 listSpaceTimeMaps

Lists all SpaceTime maps that have been specified for a system.

- `void listSpaceTimeMaps(Program prog, String system, int level)`
- `void listSpaceTimeMaps(Program prog, String system)`

7.6 listMemoryMaps

Lists all memory mappings specified for a system.

- `void listMemoryMaps(Program prog, String system)`

7.7 setSchedule

Applies schedules found by a scheduler (such as Farkas scheduler) to target mapping.

- `void setSchedule(Program prog, String system, List<ScheduledStatement> schedules)`

7.8 setParallel

Specify the parallel dimensions for a system. The nth dimension (specified by parallelDims, start with zero) with the specified ordering prefix to be parallel.

- `void setParallel(Program program, String system, int level, String orderingPrefix, String parallelDims)`
- `void setParallel(Program program, String system, String orderingPrefix, String parallelDims)`

7.9 CreateSpaceTimeLevel

Create a spacetimelevel data stucture for level (level)

- `void CreateSpaceTimeLevel(Program program, String system, int level)`

7.10 setOrderingDimensions

specify which dimension is the ordering dimension for each level of the targetMapping. Dimension starts with zero. If no value is specified for the targetMapping level, it is the first level by default.

- `void setOrderingDimensions(Program program, String system, int level, String dims)`
- `void setOrderingDimensions(Program program, String system, int level, int dim)`
- `void setOrderingDimensions(Program program, String system, String dims)`
- `void setOrderingDimensions(Program program, String system, int dim)`

7.11 setSpaceTimeMapForMemoryAllocation

set space time map for the memory allocation statement for the input/output of the use equation. isInput – 0: for nTh input of the use equaiton isInput – 1: for nTh output of the use equation

- `void setSpaceTimeMapForMemoryAllocation(Program program, String system, String label, int isInput, int num, AffineFunction stMap)`
- `void setSpaceTimeMapForMemoryAllocation(Program program, String system, String label, int isInput, int num, String stMap)`

7.12 setSpaceTimeMapForValueCopy

set the space time map for the value copy statement for input/output of a use equation

- `void setSpaceTimeMapForValueCopy(Program program, String system, String label, int isInput, int num, AffineFunction stMap)`
- `void setSpaceTimeMapForValueCopy(Program program, String system, String label, int isInput, int num, String stMap)`

7.13 setSpaceTimeMapForMemoryFree

set the space time map for the memory free statement for the input/ouput of a useEquation. isInput – 0, for the input of the useEquation isInput – 1, for the output of the useEquation

- `void setSpaceTimeMapForMemoryFree(Program program, String system, String label, int isInput, int num, AffineFunction stMap)`
- `void setSpaceTimeMapForMemoryFree(Program program, String system, String label, int isInput, int num, String stMap)`

7.14 setSpaceTimeMapForUseEquationOptimization

set the space time map for the memory allocation, value copy, memory allocation statement for the input/output of the useEuqation. isInput – 0: for the input of the useEquation isInput – 1: for the output of the useEquation

- `void setSpaceTimeMapForUseEquationOptimization(Program program, String system, String label, int isInput, int num, AffineFunction stMapForMemoryAllocation, AffineFunction stMapForValueC AffineFunction stMapForMemoryFree)`
- `void setSpaceTimeMapForUseEquationOptimization(Program program, String system, String label, int isInput, int num, String stMapForMemoryAllocation, String stMapForValueCopy, String stMapForMemoryFree)`

7.15 setMemorySpaceForUseEquationOptimization

set memory space for the input/ouput of the useEquation

- `void setMemorySpaceForUseEquationOptimization(Program program, String system, String label, int isInput, int num, String spaceName)`

7.16 setBandForTiling

configure a band of continuous dimensions for tiling.

- `void setBandForTiling(Program program, String system, String bandName, int levels, String orderingPrefix, int startDim, int endDim)`
- `void setBandForTiling(Program program, String system, String bandName, int levels, int startDim, int endDim)`

7.17 setSubTilingWithinBand

Configure the subtiling specification for a band. The band is identified by the name. There are two types of tiling type: sequential and openmp wavefront. Represented with “sequential” and “wavefront” separately.

- `void setSubTilingWithinBand(Program program, String system, String bandName, int level, int startDim, int endDim, String tilingType)`
- `void setSubTilingWithinBand(Program program, String system, String bandName, int level, String tilingType)`

7.18 setDefaultDTilerConfiguration

Set up the default configuration for dtiler: apply parametric one level tiling to all the dimensions from [start to end].

- `void setDefaultDTilerConfiguration(Program program, String system, int startDim, int endDim, String tilingType)`
- `void setDefaultDTilerConfiguration(Program program, String system, String tilingType)`