

Bugs Catalog

AT/Program/Kernel.h

Copyright © 2008, Reusable Software Research Group, The Ohio State University



```
// /*-----*\
// |   Abstract Template : Program_Kernel
// \*-----*/

#ifndef AT_PROGRAM_KERNEL
#define AT_PROGRAM_KERNEL 1

///-----
/// Global Context -----
///-----

/*!
   #include "AT/Statement/Kernel.h"
!*/

///-----
/// Interface -----
///-----

abstract_template <
  concrete_instance class Statement
  /*!
    implements
      abstract_instance Statement_Kernel <Statement>
  !*/
  >
class Program_Kernel
{
public:
  /*!
    math definition IS_PRIMITIVE_INSTRUCTION (
      name: IDENTIFIER
    ): boolean is
      name is in {"move", "turnleft", "turnright", "infect", "skip"}

    math definition IS_LEGAL_USER_DEFINED_IDENTIFIER (
      name: IDENTIFIER
    ): boolean is
      name /= empty_string and
      not IS_PRIMITIVE_INSTRUCTION (name)

    math subtype CONTEXT is finite set of (
      name: IDENTIFIER
      body: STATEMENT
    )
  !*/

```

```

exemplar c
constraint
  for all name1, name2: IDENTIFIER, body1, body2: STATEMENT
  where ((name1, body1) is in c and (name2, body2) is in c)
    (if name1= name2 then body1 = body2) and
  for all name: IDENTIFIER, body: STATEMENT
  where ((name, body) is in c)
    (IS_LEGAL_USER_DEFINED_IDENTIFIER (name) and
     root (body).kind = BLOCK)

math definition IS_DEFINED_IN (
  c: CONTEXT
  name: IDENTIFIER
): boolean is
there exists body: STATEMENT
  ((name, body) is in c)

math subtype PROGRAM is (
  name: IDENTIFIER
  context: CONTEXT
  body: STATEMENT
)
exemplar p
constraint
  root (p.body).kind = BLOCK

math definition IS_INITIAL_PROGRAM (
  p: PROGRAM
): boolean is
p.name = empty_string and
p.context = empty_set and
IS_INITIAL_STATEMENT (p.body)
!*/

standard_abstract_operations (Program_Kernel);
!*/
  Program_Kernel is modeled by PROGRAM
  initialization
    ensures IS_INITIAL_PROGRAM (self)
!*/

procedure Swap_Name (
  alters Text& n
) is_abstract;
!*/
requires
  IS_IDENTIFIER (n)
ensures
  self.name = #n and
  n = #self.name and
  self.context = #self.context and
  self.body = #self.body
!*/

```

```

procedure Swap_Body (
    alters Statement& statement
) is_abstract;
/*!
requires
    root (statement).kind = BLOCK
ensures
    self.name = #self.name and
    self.context = #self.context and
    self.body = #statement and
    statement = #self.body
!*/

procedure Add_To_Context (
    consumes Text& n,
    consumes Statement& statement
) is_abstract;
/*!
requires
    IS_LEGAL_USER_DEFINED_IDENTIFIER (n) and
    not IS_DEFINED_IN (self.context, n) and
    root (statement).kind = BLOCK
ensures
    self.name = #self.name and
    self.context = #self.context union {(#n, #statement)} and
    self.body = #self.body
!*/

procedure Remove_From_Context (
    preserves Text n,
    produces Text& n_copy,
    produces Statement& statement
) is_abstract;
/*!
requires
    IS_DEFINED_IN (self.context, n)
ensures
    n_copy = n and
    self.name = #self.name and
    self.context = #self.context - {(n_copy, statement)} and
    self.body = #self.body
!*/

procedure Remove_Any_From_Context (
    produces Text& n,
    produces Statement& statement
) is_abstract;
/*!
requires
    self.context /= empty_set
ensures
    self.name = #self.name and
    self.context = #self.context - {(n, statement)} and
    self.body = #self.body

```

```
!*/  
  
function Boolean Is_In_Context (  
    preserves Text n  
    ) is_abstract;  
/*!  
    ensures  
        Is_In_Context = IS_DEFINED_IN (self.context, n)  
!*/  
  
function Integer Size_Of_Context () is_abstract;  
/*!  
    ensures  
        Size_Of_Context = |self.context|  
!*/  
  
};  
  
#endif // AT_PROGRAM_KERNEL
```

Last modified: Mon Feb 26 17:03:05 EST 2007

Bugs Catalog

AT/Program/Generate_Code.h

Copyright © 2008, Reusable Software Research Group, The Ohio State University



```
// /*-----*\
// |   Abstract Template : Program_Generate_Code
// \*-----*/

#ifndef AT_PROGRAM_GENERATE_CODE
#define AT_PROGRAM_GENERATE_CODE 1

///-----
/// Global Context -----
///-----

#include "AT/Program/Kernel.h"
/*!
    #include "AT/Sequence/Kernel.h"
!*/

///-----
/// Interface -----
///-----

enumeration Instruction_Codes
{
    MOVE,
    TURNLEFT,
    TURNRIGHT,
    INFECT,
    SKIP,
    HALT,
    JUMP,
    JUMP_IF_NOT_NEXT_IS_EMPTY,
    JUMP_IF_NOT_NEXT_IS_NOT_EMPTY,
    JUMP_IF_NOT_NEXT_IS_WALL,
    JUMP_IF_NOT_NEXT_IS_NOT_WALL,
    JUMP_IF_NOT_NEXT_IS_FRIEND,
    JUMP_IF_NOT_NEXT_IS_NOT_FRIEND,
    JUMP_IF_NOT_NEXT_IS_ENEMY,
    JUMP_IF_NOT_NEXT_IS_NOT_ENEMY,
    JUMP_IF_NOT_RANDOM,
    JUMP_IF_NOT_TRUE
};

///-----

abstract_template <
    concrete_instance class Statement,
    /*!
```

```

        implements
            abstract_instance Statement_Kernel <Statement>
    /*/
    concrete_instance class Compiled_Program
    /*!
        implements
            abstract_instance Sequence_Kernel <Integer>
    /*/
    >
class Program_Generate_Code :
    extends
        abstract_instance Program_Kernel <Statement>
{
public:
    /*!
    math definition CALLS_INSTRUCTION (
        s: STATEMENT
        n: IDENTIFIER
    ): boolean satisfies
    there exists x: STATEMENT_LABEL
        (x = root (s) and
         if x.kind = CALL
         then
             CALLS_INSTRUCTION (s, n) =
                 (n = x.instruction)
         else
             CALLS_INSTRUCTION (s, n) =
                 there exists y: STATEMENT
                 where (y is in elements (children (s)))
                 (CALLS_INSTRUCTION (y, n)))

    math subtype NEW_INSTRUCTION is (
        name: IDENTIFIER
        body: STATEMENT
    )

    math definition HAS_A_CALLING_CYCLE (
        c: CONTEXT
    ): boolean is
    there exists v: NEW_INSTRUCTION, s: string of NEW_INSTRUCTION
        (for all i, j: NEW_INSTRUCTION
            where (<i> * <j> is substring of <v> * s * <v>)
            ({i, j} is subset of c and
             CALLS_INSTRUCTION (i.body, j.name)))

    math definition IS_CONSISTENT_WITH_CONTEXT (
        s: STATEMENT
        c: CONTEXT
    ): boolean satisfies
    there exists x: STATEMENT_LABEL
        (x = root (s) and
         if x.kind = CALL
         then
             IS_CONSISTENT_WITH_CONTEXT (s, c) =

```

```

        IS_PRIMITIVE_INSTRUCTION (x.instruction) or
        IS_DEFINED_IN (c, x.instruction)
    else
        IS_CONSISTENT_WITH_CONTEXT (s, c) =
            for all y: STATEMENT
                where (y is in elements (children (s)))
                    (IS_CONSISTENT_WITH_CONTEXT (y, c))

math definition IS_CONSISTENT (
    p: PROGRAM
): boolean satisfies
IS_CONSISTENT_WITH_CONTEXT (p.body, p.context) and
for all ni: NEW_INSTRUCTION
    where (ni is in p.context)
        (IS_CONSISTENT_WITH_CONTEXT (ni.body, p.context)) and
not HAS_A_CALLING_CYCLE (p.context)

math definition GENERATE_CODE (
    p: PROGRAM
): string of integer satisfies
if IS_CONSISTENT (p)
then
    GENERATE_CODE (p) =
        GENERATE_CODE_FOR_BLOCK (
            0, children (p.body), p.context) * <HALT>
else
    GENERATE_CODE (p) = empty_string

math definition GENERATE_CODE_FOR_BLOCK (
    start: integer
    body: string of STATEMENT
    context: CONTEXT
): string of integer satisfies
if body = empty_string
then
    GENERATE_CODE_FOR_BLOCK (
        start, body, context) = empty_string
else
    there exists s: STATEMENT, rest: string of STATEMENT
        (body = <s> * rest and
        GENERATE_CODE_FOR_BLOCK (start, body, context) =
            GENERATE_CODE_FOR_STATEMENT (start, s, context) *
            GENERATE_CODE_FOR_BLOCK (
                start + |GENERATE_CODE_FOR_STATEMENT (
                    start, s, context)|,
                rest, context))

math definition GENERATE_CODE_FOR_STATEMENT (
    start: integer
    stmt: STATEMENT
    context: CONTEXT
): string of integer satisfies
there exists x: STATEMENT_LABEL
    (x = root (stmt) and

```

```

if x.kind = BLOCK
then
    GENERATE_CODE_FOR_STATEMENT (start, stmt, context) =
        GENERATE_CODE_FOR_BLOCK (
            start, children (stmt), context)
else if x.kind = IF
then
    GENERATE_CODE_FOR_STATEMENT (start, stmt, context) =
        GENERATE_CODE_FOR_IF (
            start, x.test,
            first (children (stmt)),
            context)
else if x.kind = IF_ELSE
then
    GENERATE_CODE_FOR_STATEMENT (start, stmt, context) =
        GENERATE_CODE_FOR_IF_ELSE (
            start, x.test,
            first (children (stmt)),
            last (children (stmt)), context)
else if x.kind = WHILE
then
    GENERATE_CODE_FOR_STATEMENT (start, stmt, context) =
        GENERATE_CODE_FOR_WHILE (
            start, x.test,
            first (children (stmt)),
            context)
else if x.kind = CALL
then
    GENERATE_CODE_FOR_STATEMENT (start, stmt, context) =
        GENERATE_CODE_FOR_CALL (
            start, x.instruction, context)

```

```

math definition GENERATE_CODE_FOR_IF (
    start: integer
    test: integer
    body: STATEMENT
    context: CONTEXT
): string of integer satisfies
there exists str: string of integer
    (str = GENERATE_CODE_FOR_BLOCK (
        start + 2, children (body), context) and
    GENERATE_CODE_FOR_IF (start, test, body, context) =
        <GENERATE_TEST_CODE (test)> * <start + |str| + 2> *
        str)

```

```

math definition GENERATE_CODE_FOR_IF_ELSE (
    start: integer
    test: integer
    if_body: STATEMENT
    else_body: STATEMENT
    context: CONTEXT
): string of integer satisfies
there exists str1, str2: string of integer
    (str1 = GENERATE_CODE_FOR_BLOCK (
        start + 2, children (if_body), context) and

```

```

    str2 = GENERATE_CODE_FOR_BLOCK (
        start + |str1| + 4,
        children (else_body), context) and
    GENERATE_CODE_FOR_IF_ELSE (
        start, test, if_body, else_body, context) =
    <GENERATE_TEST_CODE (test)> * <start + |str1| + 4> *
    str1 * <JUMP> * <start + |str1| + |str2| + 4> * str2)

math definition GENERATE_CODE_FOR_WHILE (
    start: integer
    test: integer
    body: STATEMENT
    context: CONTEXT
): string of integer satisfies
there exists str: string of integer
    (str = GENERATE_CODE_FOR_BLOCK (
        start + 2, children (body), context) and
    GENERATE_CODE_FOR_WHILE (start, test, body, context) =
    <GENERATE_TEST_CODE (test)> * <start + |str| + 4> *
    str * <JUMP> * <start>)

math definition GENERATE_CODE_FOR_CALL (
    start: integer
    inst: IDENTIFIER
    context: CONTEXT
): string of integer satisfies
if IS_PRIMITIVE_INSTRUCTION (inst)
then
    GENERATE_CODE_FOR_CALL (start, inst, context) =
    <GENERATE_INSTRUCTION_CODE (inst)>
else
    there exists body: STATEMENT
    ((inst, body) is in context and
    GENERATE_CODE_FOR_CALL (start, inst, context) =
    GENERATE_CODE_FOR_BLOCK (
        start, children (body), context))

math definition GENERATE_INSTRUCTION_CODE (
    inst: IDENTIFIER
): integer satisfies
if inst = "move"
then
    GENERATE_INSTRUCTION_CODE (inst) = MOVE
else if inst = "turnleft"
then
    GENERATE_INSTRUCTION_CODE (inst) = TURNLEFT
else if inst = "turnright"
then
    GENERATE_INSTRUCTION_CODE (inst) = TURNRIGHT
else if inst = "infect"
then
    GENERATE_INSTRUCTION_CODE (inst) = INFECT
else if inst = "skip"
then
    GENERATE_INSTRUCTION_CODE (inst) = SKIP

```

```

math definition GENERATE_TEST_CODE (
    test: integer
): integer satisfies
if test = NEXT_IS_EMPTY
then
    GENERATE_TEST_CODE (test) = JUMP_IF_NOT_NEXT_IS_EMPTY
else if test = NEXT_IS_NOT_EMPTY
then
    GENERATE_TEST_CODE (test) = JUMP_IF_NOT_NEXT_IS_NOT_EMPTY
else if test = NEXT_IS_WALL
then
    GENERATE_TEST_CODE (test) = JUMP_IF_NOT_NEXT_IS_WALL
else if test = NEXT_IS_NOT_WALL
then
    GENERATE_TEST_CODE (test) = JUMP_IF_NOT_NEXT_IS_NOT_WALL
else if test = NEXT_IS_FRIEND
then
    GENERATE_TEST_CODE (test) = JUMP_IF_NOT_NEXT_IS_FRIEND
else if test = NEXT_IS_NOT_FRIEND
then
    GENERATE_TEST_CODE (test) = JUMP_IF_NOT_NEXT_IS_NOT_FRIEND
else if test = NEXT_IS_ENEMY
then
    GENERATE_TEST_CODE (test) = JUMP_IF_NOT_NEXT_IS_ENEMY
else if test = NEXT_IS_NOT_ENEMY
then
    GENERATE_TEST_CODE (test) = JUMP_IF_NOT_NEXT_IS_NOT_ENEMY
else if test = RANDOM
then
    GENERATE_TEST_CODE (test) = JUMP_IF_NOT_RANDOM
else if test = TRUE
then
    GENERATE_TEST_CODE (test) = JUMP_IF_NOT_TRUE
!*/

procedure Generate_Code (
    produces Compiled_Program& cp
) is_abstract;
!*/
preserves self
ensures
    if IS_CONSISTENT (self)
then
        cp = GENERATE_CODE (self)
!*/

};

#endif // AT_PROGRAM_GENERATE_CODE

```

Last modified: Mon Feb 26 17:03:05 EST 2007

Bugs Catalog

AT/Program/Parse.h

Copyright © 2008, Reusable Software Research Group, The Ohio State University



```
// /*-----*\
// |   Abstract Template : Program_Parse
// \*-----*/

#ifndef AT_PROGRAM_PARSE
#define AT_PROGRAM_PARSE 1

///-----
/// Global Context -----
///-----

#include "AT/Program/Kernel.h"
/*!
    #include "AT/Statement/Parse.h"
!*/

///-----
/// Interface -----
///-----

abstract_template <
    concrete_instance class Statement
    /*!
        implements
            abstract_instance Statement_Kernel <Statement>
    !*/
    >
class Program_Parse :
    extends
        abstract_instance Program_Kernel <Statement>
{
public:
    /*!
        math definition INSTRUCTION_TO_STRING_OF_TOKENS (
            name: IDENTIFIER
            body: STATEMENT
        ): string of string of character is
        <"INSTRUCTION"> * <name> * <"IS"> *
        STATEMENT_TO_STRING_OF_TOKENS (body) *
        <"END"> * <name>

        math definition CONTEXT_TO_STRING_OF_TOKENS (
            c: CONTEXT
        ): string of string of character satisfies
        if c = empty_set
```

```

    then
        CONTEXT_TO_STRING_OF_TOKENS (c) = empty_string
    else
        there exists name: IDENTIFIER, body: STATEMENT
            ((name, body) is in c and
             CONTEXT_TO_STRING_OF_TOKENS (c) =
                 INSTRUCTION_TO_STRING_OF_TOKENS (name, body) *
                 CONTEXT_TO_STRING_OF_TOKENS (c - {(name, body)}))

math definition PROGRAM_TO_STRING_OF_TOKENS (
    p: PROGRAM
): string of string of character is
<"PROGRAM"> * <p.name> * <"IS"> *
CONTEXT_TO_STRING_OF_TOKENS (p.context) *
<"BEGIN"> * STATEMENT_TO_STRING_OF_TOKENS (p.body) *
<"END"> * <p.name>

math definition IS_AN_INPUT_REP (
    s: string of character
    p: PROGRAM
): boolean is
PROGRAM_TO_STRING_OF_TOKENS (p) =
    REMOVE_SEPARATORS (TOKENIZE_PROGRAM_TEXT (s))
!*/

procedure Parse (
    alters Character_IStream& str
) is_abstract;
!*/
produces self
requires
    str.is_open = true
ensures
    if there exists p: PROGRAM, s, t: string of character
        (#str.content = s * t and
         IS_AN_INPUT_REP (s, p))
    then
        str.is_open = true and
        str.ext_name = #str.ext_name and
        there exists u: string of character
            (#str.content = u * str.content and
             IS_AN_INPUT_REP (u, self))
!*/

};

#endif // AT_PROGRAM_PARSE

```

Last modified: Sat Aug 23 09:00:20 EDT 2008

Bugs Catalog

AT/Program/Pretty_Print.h

Copyright © 2008, Reusable Software Research Group, The Ohio State University



```
// /*-----*\
// |   Abstract Template : Program_Pretty_Print
// \*-----*/

#ifndef AT_PROGRAM_PRETTY_PRINT
#define AT_PROGRAM_PRETTY_PRINT 1

///-----
/// Global Context -----
///-----

#include "AT/Program/Kernel.h"
/*!
    #include "AT/Statement/Pretty_Print.h"
!*/

///-----
/// Interface -----
///-----

abstract_template <
    concrete_instance class Statement
    /*!
        implements
            abstract_instance Statement_Kernel <Statement>
    !*/
>
class Program_Pretty_Print :
    extends
        abstract_instance Program_Kernel <Statement>
{
public:
    /*!
        math definition DISPLAY_INSTRUCTION (
            n: IDENTIFIER
            b: STATEMENT
            i: integer
        ): string of character is
        MAKE_A_STRING (i, ' ') * "INSTRUCTION " * n * " IS\n" *
        PRETTY_DISPLAY (b, i+4) *
        MAKE_A_STRING (i, ' ') * "END " * n * "\n"

        math definition DISPLAY_CONTEXT (
            c: CONTEXT
            i: integer
```

```

): string of character satisfies
if c = empty_set
then
    DISPLAY_CONTEXT (c, i) = empty_string
else
    there exists name: IDENTIFIER, body: STATEMENT
        ((name, body) is in c and
        DISPLAY_CONTEXT (c, i) =
            DISPLAY_INSTRUCTION (name, body, i) * "\n" *
            DISPLAY_CONTEXT (c - {(name, body)}, i))

math definition PRETTY_DISPLAY (
    p: PROGRAM
    i: integer
): string of character is
MAKE_A_STRING (i, ' ') * "PROGRAM " * p.name * " IS\n\n" *
DISPLAY_CONTEXT (p.context, i+4) *
MAKE_A_STRING (i, ' ') * "BEGIN\n" *
PRETTY_DISPLAY (p.body, i+4) *
MAKE_A_STRING (i, ' ') * "END " * p.name * "\n"
!*/

procedure Pretty_Print (
    alters Character_OStream& out,
    preserves Integer indentation_factor
) is_abstract;
!*/
preserves self
requires
    out.is_open = true
ensures
    out.is_open = true and
    out.ext_name = #out.ext_name and
    out.content =
        #out.content * PRETTY_DISPLAY (self, indentation_factor)
!*/

};

#endif // AT_PROGRAM_PRETTY_PRINT

```

Last modified: Thu Jan 11 16:58:48 EST 2007