

Bugs Catalog

AI/BL_Tokenizing_Machine/Kernel.h

Copyright © 2011, Reusable Software Research Group, The Ohio State University



```
// /*-----*\
// |   Abstract Instance : BL_Tokenizing_Machine_Kernel
// \*-----*/

#ifndef AI_BL_TOKENIZING_MACHINE_KERNEL
#define AI_BL_TOKENIZING_MACHINE_KERNEL 1

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

enumeration Token_Types
{
    KEYWORD,
    IDENTIFIER,
    CONDITION,
    WHITE_SPACE,
    COMMENT,
    ERROR
};

///-----

abstract_instance
class BL_Tokenizing_Machine_Kernel
{
public:

    /*!
    // some general operations on strings

    math definition LAST_OF (
        s: string of character
    ): string of character satisfies
    if s = empty_string
        then LAST_OF (s) = empty_string
    else there exists c: character, rest_of_s: string of character
        (s = rest_of_s * <c> and
        LAST_OF (s) = <c>)

    math definition ALL_BUT_LAST_OF (
        s: string of character
    ): string of character satisfies
    s = ALL_BUT_LAST_OF (s) * LAST_OF (s)

    math definition PREFIX (
```

```

    s_set: set of string of character
  ): set of string of character is
  {x: string of character where
    (there exists y: string of character
      (x * y is in s_set))}

// domain specific definitions

math definition IS_KEYWORD (
  s: string of character
): boolean is
s is in {"BEGIN", "DO", "ELSE", "END", "INSTRUCTION",
        "IF", "IS", "PROGRAM", "THEN", "WHILE"}

math definition IS_CONDITION_NAME (
  s: string of character
): boolean is
s is in {"next-is-empty", "next-is-not-empty",
        "next-is-enemy", "next-is-not-enemy",
        "next-is-friend", "next-is-not-friend",
        "next-is-wall", "next-is-not-wall",
        "random", "true"}

math definition IS_LETTER (
  c: character
): boolean is
c is in {'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i',
        'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r',
        's', 't', 'u', 'v', 'w', 'x', 'y', 'z', 'A',
        'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J',
        'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S',
        'T', 'U', 'V', 'W', 'X', 'Y', 'Z'}

math definition IS_DIGIT (
  c: character
): boolean is
c is in {'0', '1', '2', '3', '4', '5', '6', '7', '8', '9'}

math definition IS_IDENTIFIER (
  s: string of character
): boolean is
there exists c: character, rest: string of character
(s = <c> * rest and
 IS_LETTER (c) and
 for all x: character where (x is in elements (rest))
   (IS_LETTER (x) or IS_DIGIT (x) or x = '-')) and
not IS_KEYWORD (s) and
not IS_CONDITION_NAME (s)

math definition IS_WHITE_SPACE (
  s : string of character
): boolean is
s /= empty_string and
elements (s) is subset of {'\n', '\t', ' '}

```

```

math definition IS_COMMENT (
    s : string of character
): boolean is
first (s) = '#' and
'\n' is not in elements (s)

math definition OK_STRINGS: set of string of character is
{s: string of character where (IS_KEYWORD (s))} union
{s: string of character where (IS_IDENTIFIER (s))} union
{s: string of character where (IS_CONDITION_NAME (s))} union
{s: string of character where (IS_WHITE_SPACE (s))} union
{s: string of character where (IS_COMMENT (s))}

math definition IS_COMPLETE_TOKEN_TEXT (
    s: string of character
    c: character
): boolean is
(s is in OK_STRINGS and s * <c> is not in OK_STRINGS) or
(<c> is in PREFIX (OK_STRINGS) and
s * <c> is not in PREFIX (OK_STRINGS))

math definition WHICH_KIND (
    s: string of character
): integer satisfies
if IS_KEYWORD (s)
then
    WHICH_KIND (s) = KEYWORD
else if IS_IDENTIFIER (s)
then
    WHICH_KIND (s) = IDENTIFIER
else if IS_CONDITION_NAME (s)
then
    WHICH_KIND (s) = CONDITION
else if IS_WHITE_SPACE (s)
then
    WHICH_KIND (s) = WHITE_SPACE
else if IS_COMMENT (s)
then
    WHICH_KIND (s) = COMMENT
else
    WHICH_KIND (s) = ERROR

math definition REMOVE_SEPARATORS (
    s: string of string of character
): string of string of character satisfies
if s = empty_string
then
    REMOVE_SEPARATORS (s) = empty_string
else
    there exists t: string of character,
        rest: string of string of character
    (s = <t> * rest and
    if IS_WHITE_SPACE (t) or IS_COMMENT (t)

```

```

        then
            REMOVE_SEPARATORS (s) = REMOVE_SEPARATORS (rest)
        else
            REMOVE_SEPARATORS (s) =
                <t> * REMOVE_SEPARATORS (rest))

math definition TOKENIZE_PROGRAM_TEXT (
    s: string of character
): string of string of character satisfies
if s = empty_string
then
    TOKENIZE_PROGRAM_TEXT (s) = empty_string
else
    there exists t, rest: string of character
        (s = t * rest and
         t /= empty_string and
         (if rest /= empty_string
          then
              there exists c: character,
                  rest_of_rest: string of character
                     (rest = <c> * rest_of_rest and
                      IS_COMPLETE_TOKEN_TEXT (t, c))) and
          (for all p, q: string of character, c: character
           where (t = p * <c> * q)
            (not IS_COMPLETE_TOKEN_TEXT (p, c))) and
          TOKENIZE_PROGRAM_TEXT (s) =
              <t> * TOKENIZE_PROGRAM_TEXT (rest))

math subtype BL_TOKENIZING_MACHINE_MODEL is (
    buffer: string of character
    ready_to_dispense: boolean
)
exemplar m
constraint
    m.ready_to_dispense =
        there exists c: character
            (LAST_OF (m.buffer) = <c> and
             IS_COMPLETE_TOKEN_TEXT (ALL_BUT_LAST_OF (m.buffer), c))
!*/

standard_abstract_operations (BL_Tokenizing_Machine_Kernel);
!*/
    BL_Tokenizing_Machine_Kernel is modeled by BL_TOKENIZING_MACHINE_MODEL
initialization ensures
    self.buffer = empty_string and
    self.ready_to_dispense = false
!*/

procedure Insert (
    preserves Character ch
) is_abstract;
!*/
requires
    self.ready_to_dispense = false

```

```

    ensures
        self.buffer = #self.buffer * <ch> and
        self.ready_to_dispense =
            IS_COMPLETE_TOKEN_TEXT (#self.buffer, ch)
    /*/

procedure Dispense (
    produces Text& token_text,
    produces Integer& token_kind
) is_abstract;
/*!
requires
    self.ready_to_dispense = true
ensures
    token_text = ALL_BUT_LAST_OF (#self.buffer) and
    token_kind = WHICH_KIND (token_text) and
    self.buffer = LAST_OF (#self.buffer) and
    self.ready_to_dispense = false
/*/

procedure Flush_A-Token (
    produces Text& token_text,
    produces Integer& token_kind
) is_abstract;
/*!
requires
    self.ready_to_dispense = false
ensures
    token_text = #self.buffer and
    token_kind = WHICH_KIND (token_text) and
    self.buffer = empty_string and
    self.ready_to_dispense = false
/*/

function Boolean Is_Ready_To_Dispense ( ) is_abstract;
/*!
    ensures
        Is_Ready_To_Dispense = self.ready_to_dispense
/*/

function Integer Size ( ) is_abstract;
/*!
    ensures
        Size = |self.buffer|
/*/

};

#endif // AI_BL_TOKENIZING_MACHINE_KERNEL

```

Bugs Catalog

AI/BL Tokenizing Machine/Get.h

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



```
// /*-----*\
// |   Abstract Instance : BL-Tokenizing_Machine_Get
// \*-----*/

#ifndef AI_BL_TOKENIZING_MACHINE_GET
#define AI_BL_TOKENIZING_MACHINE_GET 1

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

#include "AI/BL-Tokenizing_Machine/Kernel.h"

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

abstract_instance
class BL-Tokenizing_Machine_Get :
    extends
        abstract_instance BL-Tokenizing_Machine_Kernel
{
public:

    /*!
    math definition IS_NEXT_TOKEN_AND_REMAINDER (
        in_str: string of character
        token_text: string of character
        c: character
        out_str: string of character
    ): boolean satisfies
    IS_NEXT_TOKEN_AND_REMAINDER (in_str, token_text, c, out_str) =
    (in_str = token_text * <c> * out_str and
    IS_COMPLETE_TOKEN_TEXT (token_text, c) and
    for all t2, out_str2: string of character, c2: character
        (if IS_NEXT_TOKEN_AND_REMAINDER (in_str, t2, c2, out_str2)
        then |token_text| <= |t2|))

    math definition IS_NEXT_NON_SEPARATOR_TOKEN_AND_REMAINDER (
        in_str: string of character
        token_text: string of character
        c: character
        out_str: string of character
    ): boolean satisfies
    (IS_NEXT_TOKEN_AND_REMAINDER (in_str, token_text, c, out_str) and
```

```

WHICH_KIND (token_text) /= WHITE_SPACE and
WHICH_KIND (token_text) /= COMMENT) or
(there exists t1, out_str1: string of character,
  c1: character
  (IS_NEXT_TOKEN_AND_REMAINDER (in_str, t1, c1, out_str1) and
  IS_NEXT_NON_SEPARATOR_TOKEN_AND_REMAINDER (
    <c1> * out_str1, token_text, c, out_str) and
  (WHICH_KIND (t1) = WHITE_SPACE or
  WHICH_KIND (t1) = COMMENT)))

math definition IS_REMAINDER_AFTER_LEADING_SEPARATORS (
  in_str: string of character
  out_str: string of character
): boolean satisfies
if there exists t, o: string of character, c: character
  (IS_NEXT_TOKEN_AND_REMAINDER (in_str, t, c, o) and
  (WHICH_KIND (t) = WHITE_SPACE or
  WHICH_KIND (t) = COMMENT))
then
  there exists t, o: string of character, c: character
  (IS_NEXT_TOKEN_AND_REMAINDER (in_str, t, c, o) and
  (WHICH_KIND (t) = WHITE_SPACE or
  WHICH_KIND (t) = COMMENT) and
  IS_REMAINDER_AFTER_LEADING_SEPARATORS (in_str, out_str) =
  IS_REMAINDER_AFTER_LEADING_SEPARATORS
  (<c> * o, out_str))
else
  IS_REMAINDER_AFTER_LEADING_SEPARATORS (in_str, out_str) =
  (in_str = out_str)

!*/

procedure Get_Next-Token (
  alters Character_IStream& str,
  produces Text& token_text,
  produces Integer& token_kind
) is_abstract;
!*/
requires
  str.is_open = true
ensures
  str.is_open = true and
  str.ext_name = #str.ext_name and
  (if there exists t, x: string of character, c: character
  (IS_NEXT_TOKEN_AND_REMAINDER (
    #self.buffer * #str.content, t, c, x))
  then
    there exists c: character
    (#self.buffer = <c> and
    IS_NEXT_TOKEN_AND_REMAINDER (
      #self.buffer * #str.content,
      token_text, c, str.content))
  else
    token_text = #self.buffer * #str.content and

```

```

        str.content = empty_string and
        self.buffer = empty_string) and
    self.ready_to_dispense = false and
    token_kind = WHICH_KIND (token_text)
!*/

procedure Get_Next_Non_Separator-Token (
    alters Character_IStream& str,
    produces Text& token_text,
    produces Integer& token_kind
) is_abstract;
!*/

requires
    str.is_open = true
ensures
    str.is_open = true and
    str.ext_name = #str.ext_name and
    (if there exists t, x: string of character, c: character
        (IS_NEXT_NON_SEPARATOR_TOKEN_AND_REMAINDER (
            #self.buffer * #str.content, t, c, x))
    then
        there exists c: character
            (self.buffer = <c> and
                IS_NEXT_NON_SEPARATOR_TOKEN_AND_REMAINDER (
                    #self.buffer * #str.content,
                    token_text, c, str.content))
    else
        self.buffer = empty_string and
        str.content = empty_string and
        IS_REMAINDER_AFTER_LEADING_SEPARATORS
            (#self.buffer * #str.content, token_text))
    self.ready_to_dispense = false and
    token_kind = WHICH_KIND (token_text)
!*/

};

#endif // AI_BL_TOKENIZING_MACHINE_GET

```

Last modified: Sat Aug 23 09:00:55 EDT 2008