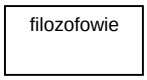
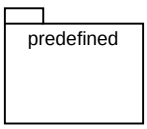

filozofowie


predefined

Package predefined

```
NEWTYPE Boolean
  LITERALS
    true,false;
  OPERATORS
    "not": Boolean          -> Boolean;
    "and": Boolean, Boolean -> Boolean;
    "or" : Boolean, Boolean -> Boolean;
    "xor": Boolean, Boolean -> Boolean;
    "=>" : Boolean, Boolean -> Boolean;
ENDNEWTYPE Boolean;

NEWTYPE Integer
  LITERALS
    NAMECLASS ('0':'9')+;
  OPERATORS
    "-"  : Integer          -> Integer;
    "+"  : Integer, Integer -> Integer;
    "-"  : Integer, Integer -> Integer;
    "*"  : Integer, Integer -> Integer;
    "/"  : Integer, Integer -> Integer;
    "mod": Integer, Integer -> Integer;
    "rem": Integer, Integer -> Integer;
    "<"  : Integer, Integer -> Boolean;
    ">"  : Integer, Integer -> Boolean;
    "<=" : Integer, Integer -> Boolean;
    ">=" : Integer, Integer -> Boolean;
    float: Integer          -> Real;
    fix  : Real             -> Integer;
ENDNEWTYPE Integer;

SYNTYPE Natural = Integer
  CONSTANTS >= 0
ENDSYNTYPE Natural;


NEWTYPE Real
  LITERALS
    NAMECLASS (('0':'9')+) OR (('0':'9')*'.'('0':'9')+);
  OPERATORS
    "-"  : Real          -> Real;
    "+"  : Real,Real     -> Real;
    "-"  : Real,Real     -> Real;
    "*"  : Real,Real     -> Real;
    "/"  : Real,Real     -> Real;
    "<"  : Real,Real     -> Boolean;
    ">"  : Real,Real     -> Boolean;
    "<=" : Real,Real     -> Boolean;
    ">=" : Real,Real     -> Boolean;
/* ASN.1 operator: */
    power: Integer, Integer -> Real;
ENDNEWTYPE Real;

NEWTYPE PId
  LITERALS
    null;
  OPERATORS
    unique! : PId -> PId;
ENDNEWTYPE PId;
```

```
NEWTYPE Character
  LITERALS
    NUL, SOH, STX, ETX, EOT, ENQ, ACK, BEL,
    BS,  HT,  LF,  VT,  FF,  CR,  SO,  SI,
    DLE, DC1, DC2, DC3, DC4, NAK, SYN, ETB,
    CAN, EM,  SUB, ESC, FS,  GS,  RS,  US,
    ' ', '!', '"', '#', '$', '%', '&', '''',
    '(', ')', '*', '+', ',', '-', '.', '/',
    '0', '1', '2', '3', '4', '5', '6', '7',
    '8', '9', ':', ';', '<', '=', '>', '?',
    '@', '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', '{', '|', '}', '~', DEL;
      • /* '''' is an apostrophe, ' ' is a space, '~' is a tilde */
  OPERATORS
    chr  : Integer -> Character;
    num  : Character -> Integer;
    "<"  : Character, Character -> Boolean;
    "<=" : Character, Character -> Boolean;
    ">"  : Character, Character -> Boolean;
    ">=" : Character, Character -> Boolean;
ENDNEWTYPE Character;

NEWTYPE Charstring String (Character,'')
  ADDING LITERALS
       NAMECLASS '''' ((' ':'&') OR '''''' OR ('(':'~'))+ '''';
ENDNEWTYPE Charstring;

NEWTYPE Duration
  LITERALS
    NAMECLASS (('0':'9')+) OR (('0':'9')*'.'('0':'9')+);
  OPERATORS
    duration!: Real          -> Duration;
    "+"  : Duration, Duration -> Duration;
    "-"  : Duration           -> Duration;
    "-"  : Duration, Duration -> Duration;
    "*"  : Real,     Duration -> Duration;
    "*"  : Duration, Real     -> Duration;
    "/"  : Duration, Real     -> Duration;
    "<"  : Duration, Duration -> Boolean;
    ">"  : Duration, Duration -> Boolean;
    "<=" : Duration, Duration -> Boolean;
    ">=" : Duration, Duration -> Boolean;
ENDNEWTYPE Duration;

NEWTYPE Time
  LITERALS
    NAMECLASS (('0':'9')+) OR (('0':'9')*'.'('0':'9')+);
  OPERATORS
    time!: Duration       -> Time;
    "<"  : Time, Time     -> Boolean;
    "<=" : Time, Time     -> Boolean;
    ">"  : Time, Time     -> Boolean;
    ">=" : Time, Time     -> Boolean;
    "+"  : Duration, Time -> Time;
    "+"  : Time, Duration -> Time;
    "-"  : Time, Duration -> Time;
    "-"  : Time, Time     -> Duration;
ENDNEWTYPE Time;
```

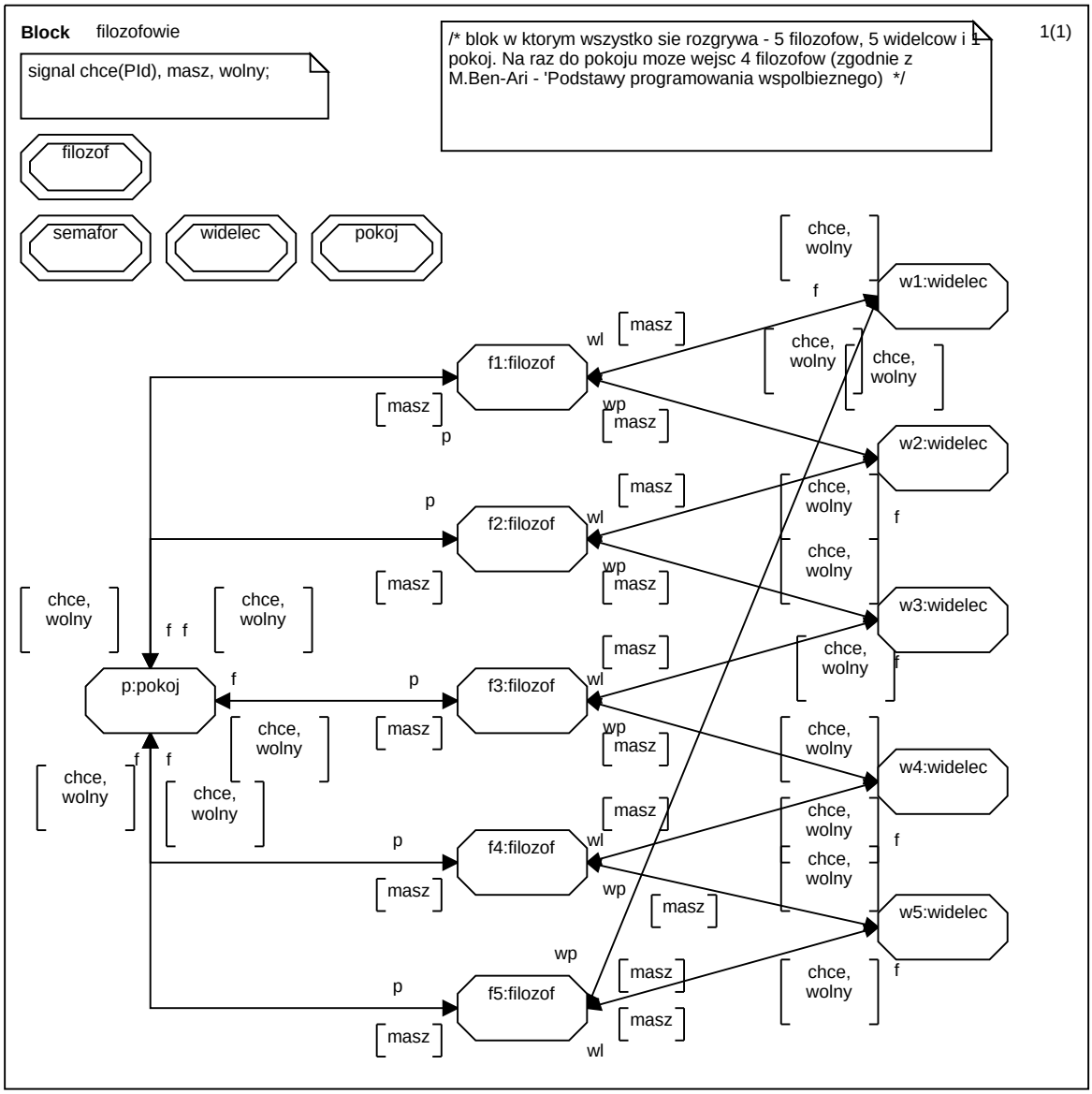
```
GENERATOR equality(TYPE item)
  OPERATORS
  "=" : equality, equality -> Boolean;
  "/=": equality, equality -> Boolean;
/*!Z105*/
 encode: equality -> Bitstring;
 encode: equality, Encoding -> Bitstring;
 decode: Bitstring -> equality;
 decode: Bitstring, Encoding -> equality;
/*!Z105END*/
ENDGENERATOR;


GENERATOR ordered(TYPE item)
  OPERATORS
    "<"  : ordered, ordered -> Boolean;
    ">"  : ordered, ordered -> Boolean;
    "<=" : ordered, ordered -> Boolean;
    ">=" : ordered, ordered -> Boolean;
ENDGENERATOR;

GENERATOR String(TYPE Itemsort LITERAL emptystring)
  /* Strings are "indexed" from one */
  LITERALS
    emptystring;
  OPERATORS
    mkstring : Itemsort                -> String;
    length   : String                  -> Integer;
    first    : String                  -> Itemsort;
    last     : String                  -> Itemsort;
    "//"     : String, String          -> String;
    extract! : String, Integer         -> Itemsort;
    modify!  : String, Integer, Itemsort -> String;
    substring: String, Integer, Integer  -> String;
ENDGENERATOR String;

GENERATOR Powerset(TYPE Itemsort)
  LITERALS
    empty;
  OPERATORS
    "in"     : Itemsort, Powerset -> Boolean;
    incl     : Itemsort, Powerset -> Powerset;
    del      : Itemsort, Powerset -> Powerset;
    "<"      : Powerset, Powerset -> Boolean;
    ">"      : Powerset, Powerset -> Boolean;
    "<="     : Powerset, Powerset -> Boolean;
    ">="     : Powerset, Powerset -> Boolean;
    "and"    : Powerset, Powerset -> Powerset;
    "or"     : Powerset, Powerset -> Powerset;
ENDGENERATOR Powerset;



GENERATOR Array(TYPE Index, TYPE Itemsort)
  OPERATORS
    make!   : Itemsort             -> Array;
    modify! : Array, Index, Itemsort -> Array;
    extract!: Array, Index         -> Itemsort;
ENDGENERATOR Array;
```

```
/*!Z105*/ /* Don't change this line */
/* ASN.1 types */
SYNTYPE
  IA5String = Charstring
ENDSYNTYPE;

SYNTYPE
  NumericString = Charstring (from ("0".."9"))
ENDSYNTYPE;

SYNTYPE
  Printablestring = Visiblestring
ENDSYNTYPE;

SYNTYPE
  Visiblestring = Charstring (from
("A".."Z"|"a".."z"|"0".."9"|'''','(',')','+',',','-','.','/',':','=','?'))
ENDSYNTYPE;

NEWTYPE Graphicstring
  inherits Charstring
  operators all;
ENDNEWTYPE Graphicstring;

NEWTYPE Universalstring
  inherits Charstring
  operators all;
ENDNEWTYPE Universalstring;

NEWTYPE Enumeration
  operators
    pred  : Enumeration -> Enumeration;
    succ  : Enumeration -> Enumeration;
    first : Enumeration -> Enumeration;
    last  : Enumeration -> Enumeration;
    num   : Enumeration -> Integer;
    "<"   : Enumeration, Enumeration -> Boolean;
    "<="  : Enumeration, Enumeration -> Boolean;
    ">"   : Enumeration, Enumeration -> Boolean;
    ">="  : Enumeration, Enumeration -> Boolean;
ENDNEWTYPE Enumeration;

SYNONYM PLUS_INFINITY  Real = external;
SYNONYM MINUS_INFINITY Real = external;
```

```
/*!Z105*/ /* Don't change this line */
NEWTYPE Bit
  inherits Boolean
  literals 0 = false, 1 = true;
  operators all;
ENDNEWTYPE Bit;

Encoding ::=  ENUMERATED{BER,CER,DER,PER};

NEWTYPE Bitstring String0(Bit,"B);
  adding
    literals nameclass('0' or '1')*'B',
             nameclass(('0':'9') or ('A':'F'))*'H';
  operators
    "not": Bitstring             -> Bitstring;
    "and": Bitstring, Bitstring -> Bitstring;
    "or" : Bitstring, Bitstring -> Bitstring;
    "xor": Bitstring, Bitstring -> Bitstring;
    "=>" : Bitstring, Bitstring -> Bitstring;
ENDNEWTYPE Bitstring;

SYNTYPE Octet = Bitstring constants size (8)
ENDSYNTYPE Octet;

NEWTYPE Octetstring String(Octet,"B)
    literals nameclass(('0' or '1')8)+'B',
             nameclass((('0':'9') or ('A':'F'))2)+'H';
    operators
      bitstring   : Octetstring -> Bitstring;
      octetstring : Bitstring -> Octetstring;
      Bit_String  : Octetstring -> Bitstring; /* SDL 96 version */
      Octet_String : Bitstring -> Octetstring; /* SDL 96 version */
ENDNEWTYPE Octetstring;

syntype Octet_String = Octetstring endsyntype;
syntype Bit_String = Bitstring endsyntype;

NEWTYPE NULL
  literals null;
ENDNEWTYPE NULL;

NEWTYPE Object_element
  literals nameclass ('0':'9')+;
ENDNEWTYPE Object_element;

NEWTYPE Object_identifier String(Object_element,emptystring)
ENDNEWTYPE Object_identifier;

NEWTYPE Any_type
ENDNEWTYPE Any_type;

GeneralizedTime       ::=       Visiblestring;
ATCTime ::= Visiblestring;
UTCTime ::= Visiblestring;
EXTERNAL_Type ::= sequence
        { direct_reference      Object_identifier optional,
          indirect_reference    Integer          optional,
          data_value_descriptor ObjectDescriptor  optional,
          encoding choice { single_ASN1_type Any_type,
                            octet_aligned    Octetstring,
                            arbitrary        Bitstring
                          }
        };
ObjectDescriptor ::= Graphicstring;
```

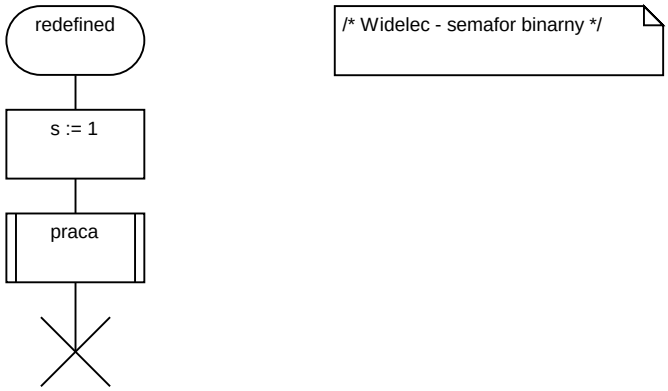
Package predefined 4(4)

```
/***** ASN.1 GENERATORS *****/
GENERATOR String0(TYPE Itemsort, LITERAL Emptystring)
    String(Itemsort,Emptystring)
ENDGENERATOR;

GENERATOR Bag(type Itemsort)
  literals Empty;
  operators
    incl   : Itemsort, Bag -> Bag;
    del    : Itemsort, Bag -> Bag;
    length : Bag           -> Integer;
    take   : Bag           -> Itemsort;
    makebag: Itemsort      -> Bag;
    "in"   : Itemsort, Bag -> Boolean;
    "<"    : Bag, Bag      -> Boolean;
    ">"    : Bag, Bag      -> Boolean;
    "<="   : Bag, Bag      -> Boolean;
    ">="   : Bag, Bag      -> Boolean;
    "and"  : Bag, Bag      -> Bag;
    "or"   : Bag, Bag      -> Bag;
ENDGENERATOR;

/*!SDL2000*/ /* Don't change this line */
exception
            OutOfRange,           /* A range check has failed. */
            InvalidReference,     /* Null was used incorrectly. Wrong Pid for this signal. */
            NoMatchingAnswer,  /* No answer matched in a decision without else part. */
            UndefinedVariable,        /* A variable was used that is "undefined". */
            UndefinedField,       /* An undefined field of a choice or struct was accessed. */
            InvalidIndex,         /* A String or Array was accessed with an incorrect index. */
            DivisionByZero,      /* An Integer or Real division by zero was attempted. */
            Empty;                /* No element could be returned. */
```


Block filozofowie
1(1)
signal chce(PId), masz, wolny;
/* blok w ktorym wszystko sie rozgrywa - 5 filozofow, 5 widelcow i 1 pokoj. Na raz do pokoju moze wejsc 4 filozofow (zgodnie z M.Ben-Ari - 'Podstawy programowania wspolbieznego) */
filozof
semafor
widelec
pokoj
p:pokoj
f1:filozof
f2:filozof
f3:filozof
f4:filozof
f5:filozof
w1:widelec
w2:widelec
w3:widelec
w4:widelec
w5:widelec
chce, wolny
masz
wl
wp
f
p

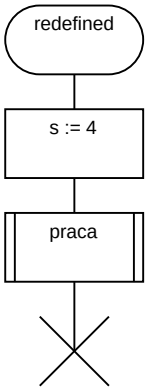
Process Type widelec 1(1)

inherits semafor

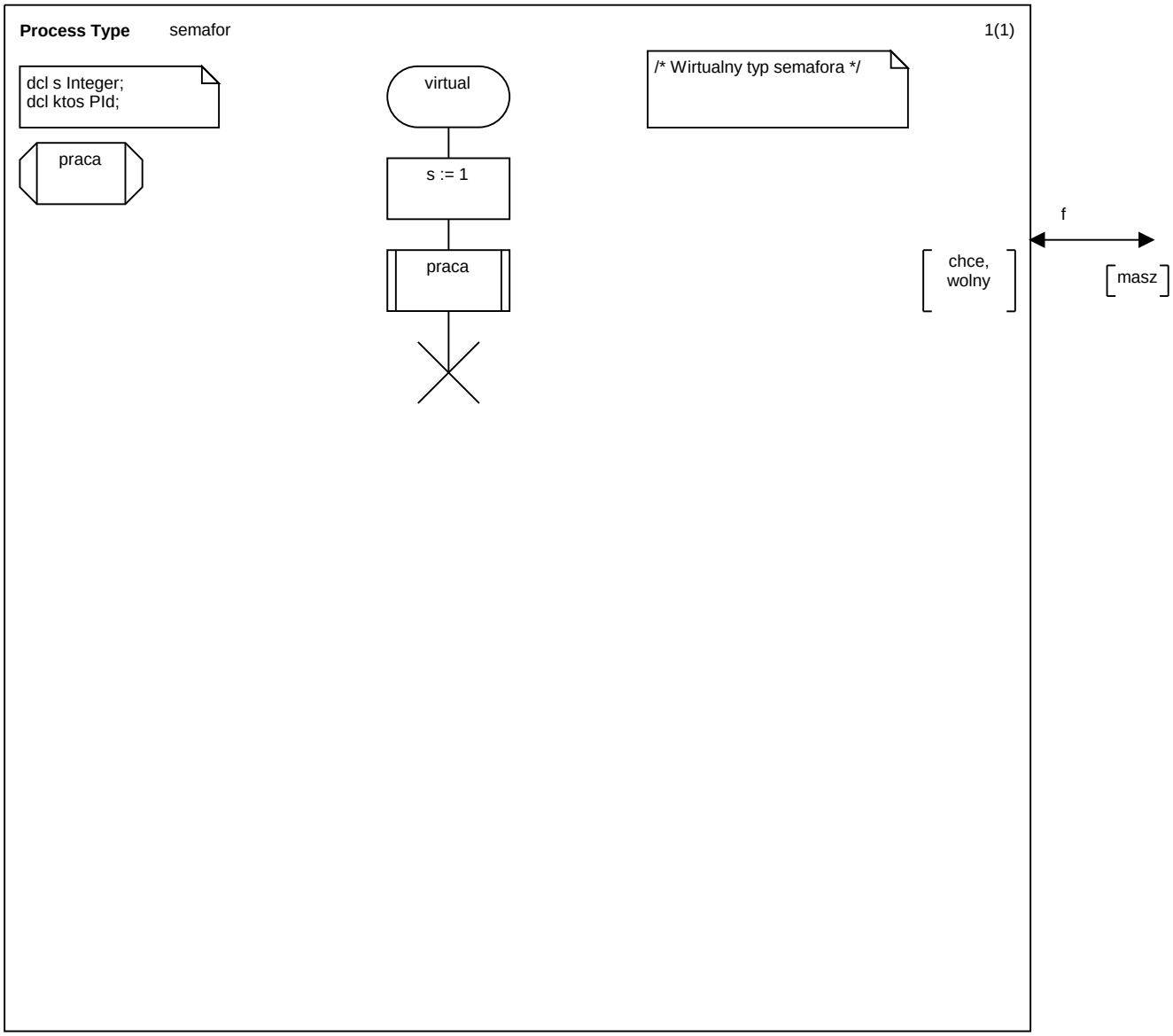


Process Type pokoj 1(1)

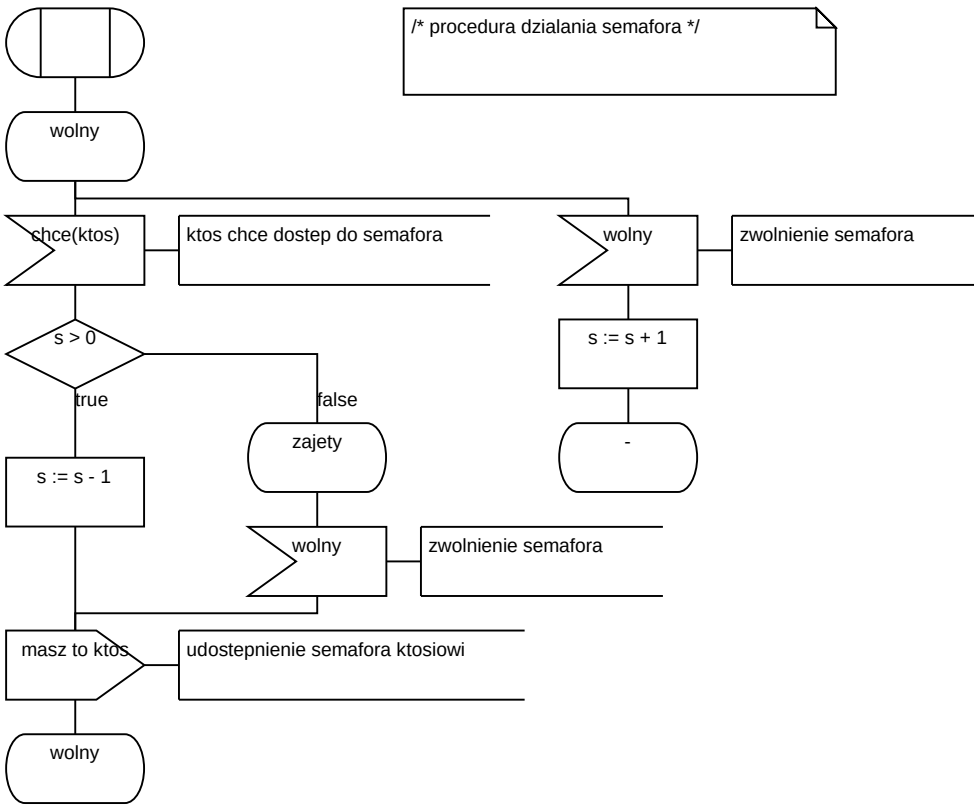
inherits semafor

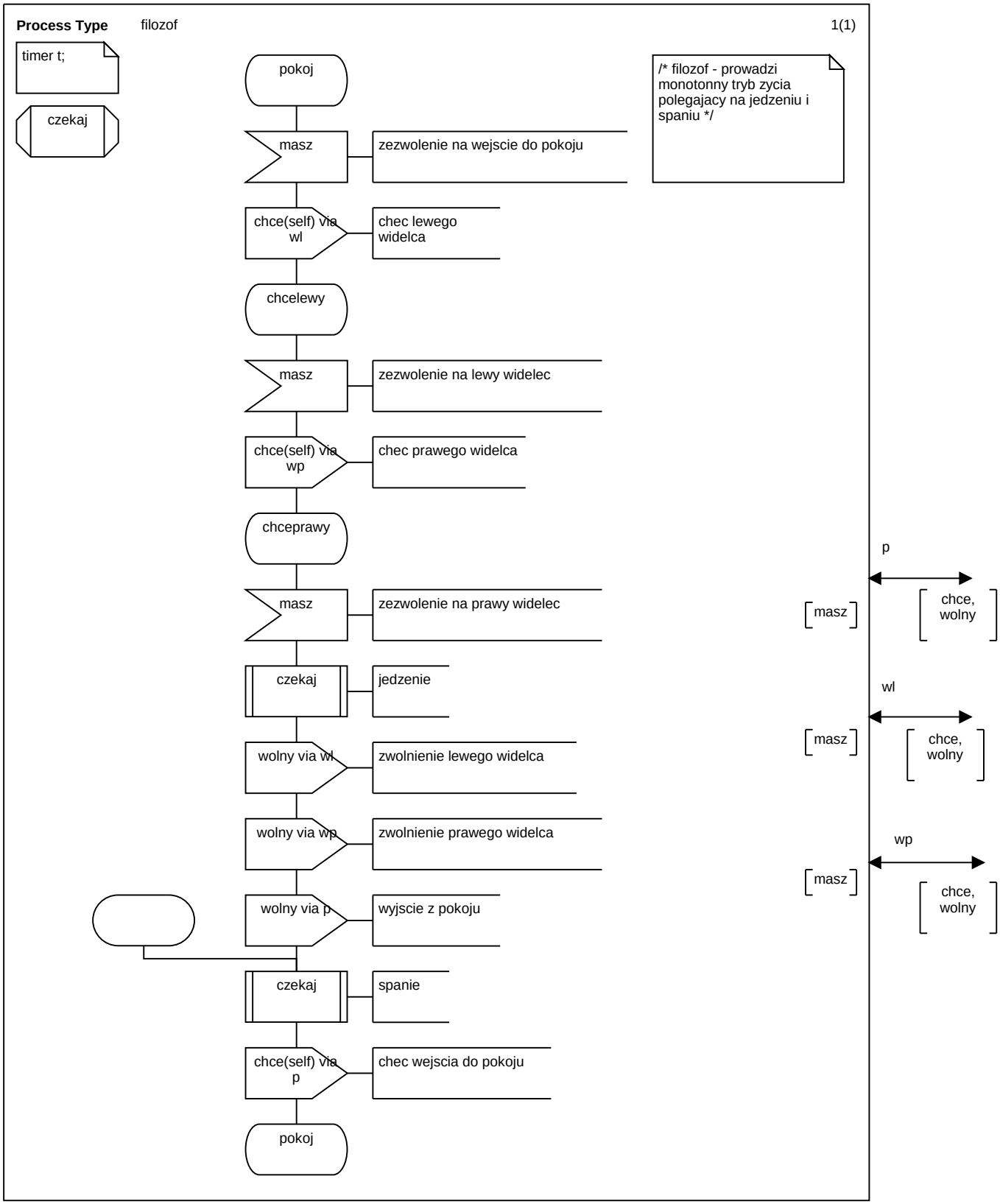



```
/* pokoj - semafor ogolny o wartosci poczatkowej
4. Zgodnie z ksiazka M. Ben-Ari 'Podstawy
programowania wspolbieznego' - do pokoju moze
probowac wejsc maksymalnie 4 filozofow */
```

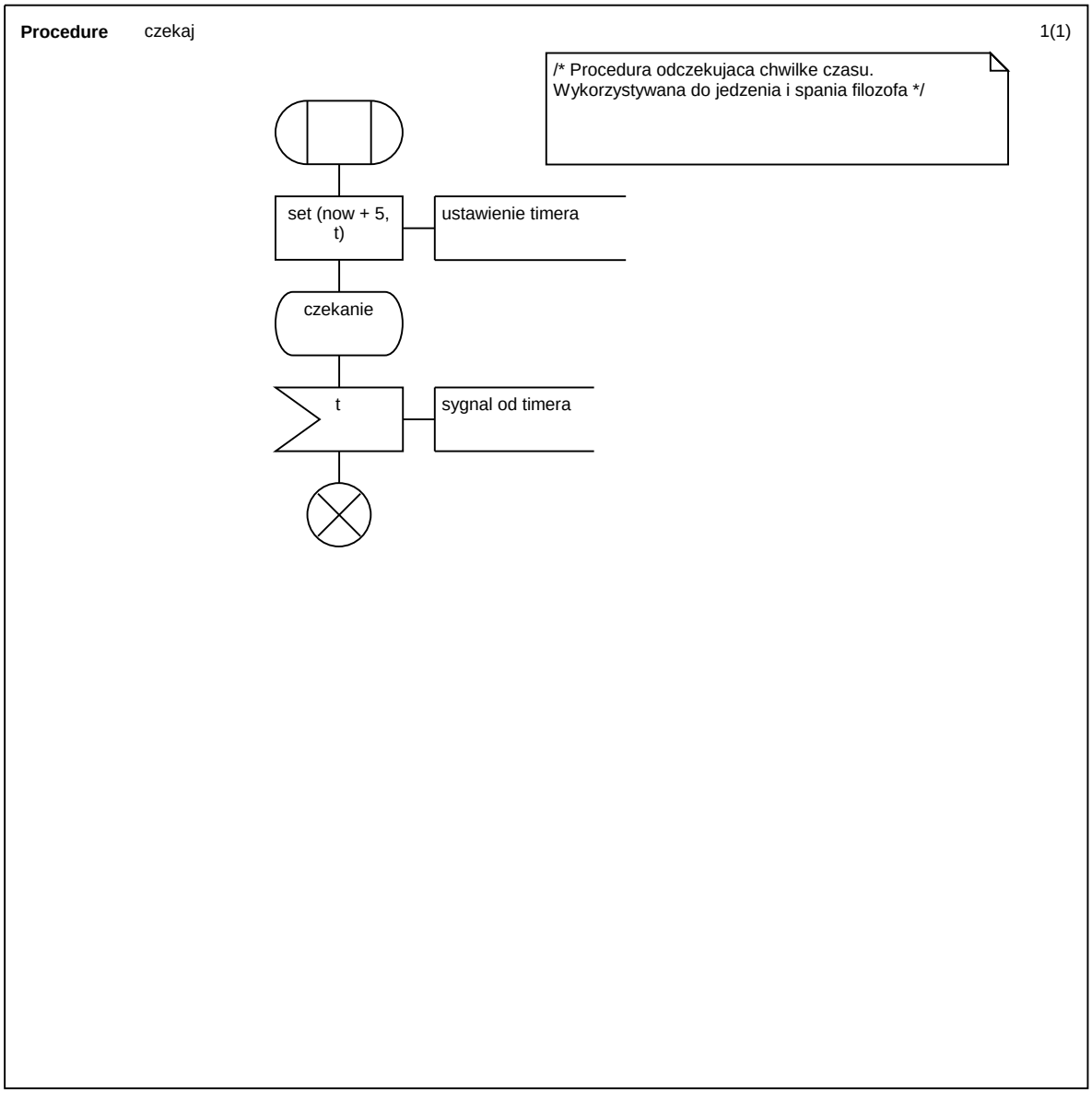

Process Type
semafor
1(1)
dcl s Integer;
dcl ktos PId;
praca
virtual
s := 1
praca
/* Wirtualny typ semafora */
f
chce,
wolny
masz

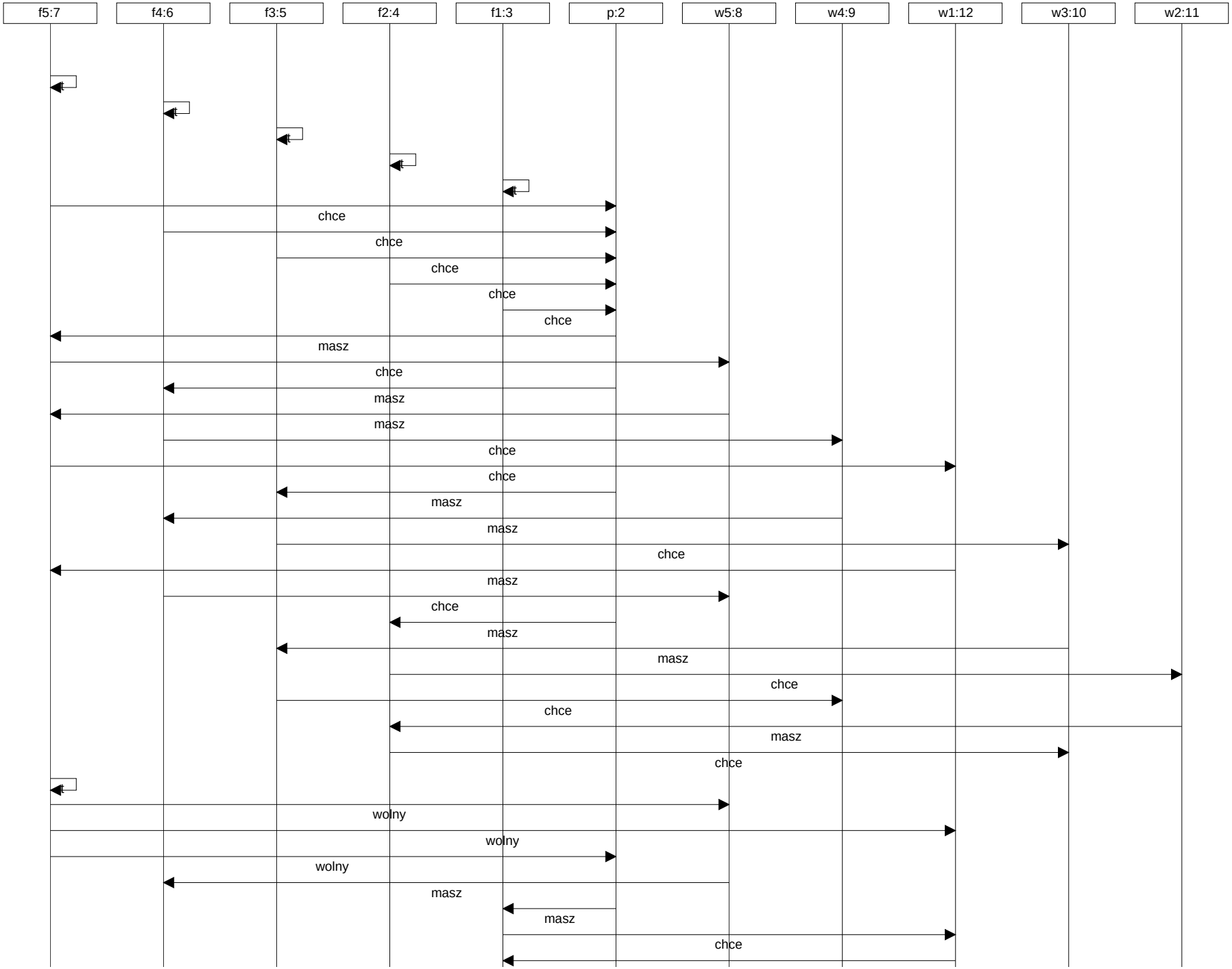
Procedure praca 1(1)

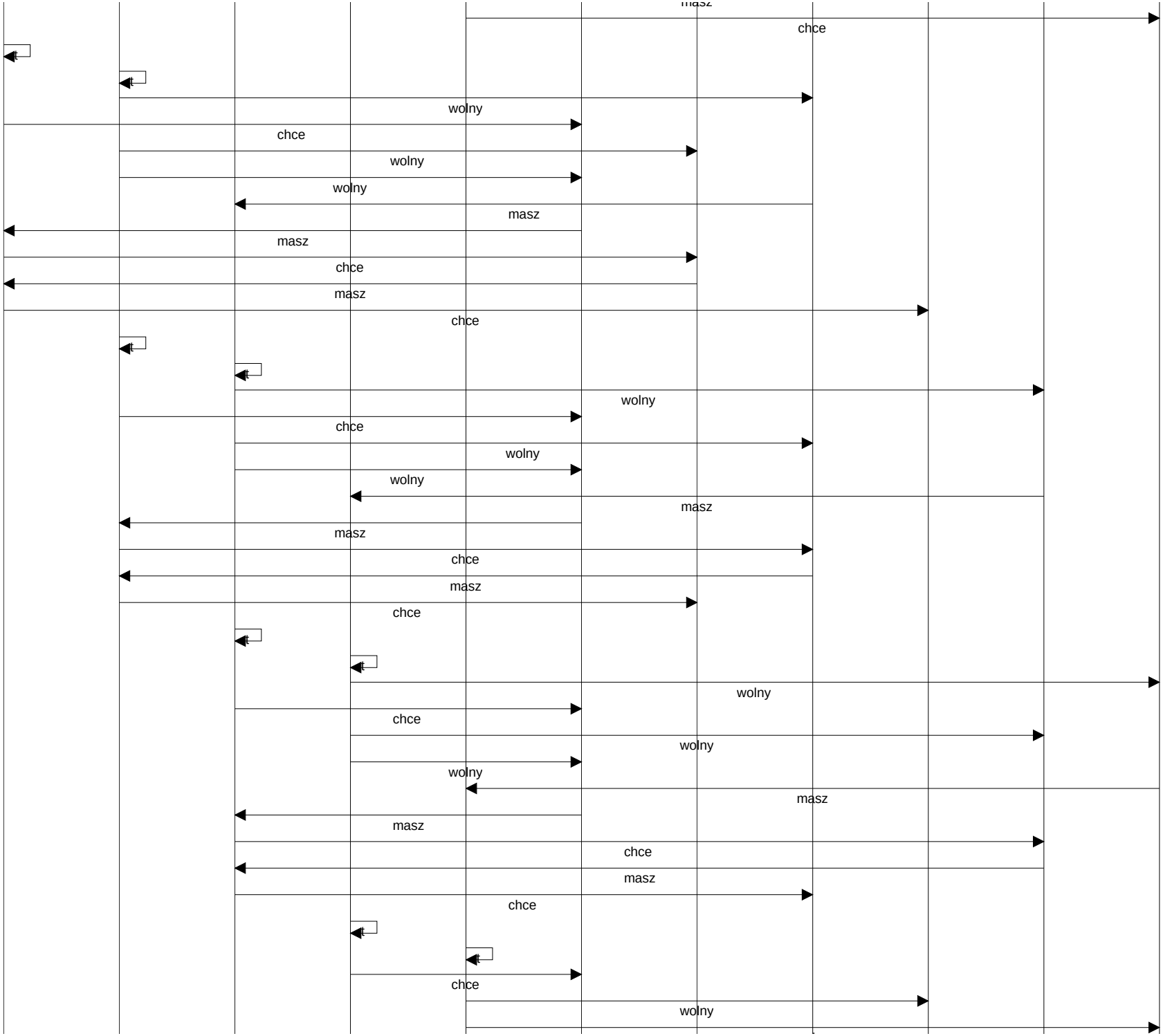


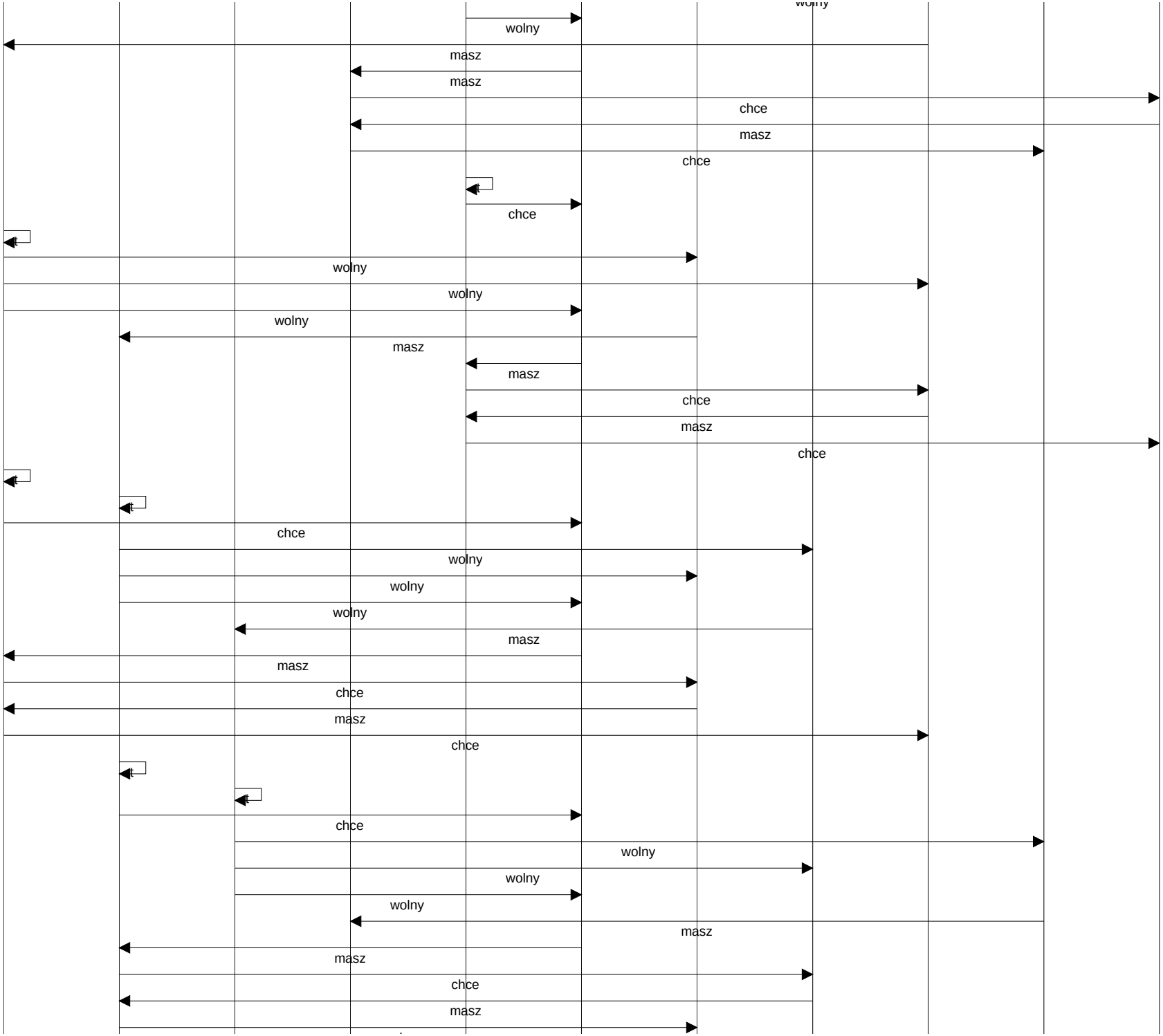

Process Type
filozof
1(1)
timer t;
czekaj
pokoj
masz
zezwolenie na wejscie do pokoju
/* filozof - prowadzi monotonny tryb zycia polegajacy na jedzeniu i spaniu */
chce(self) via wl
chec lewego widelca
chcelewy
masz
zezwolenie na lewy widelec
chce(self) via wp
chec prawego widelca
chceprawy
masz
zezwolenie na prawy widelec
czekaj
jedzenie
wolny via wl
zwolnienie lewego widelca
wolny via wp
zwolnienie prawego widelca
wolny via p
wyjscie z pokoju
czekaj
spanie
chce(self) via p
chec wejscia do pokoju
pokoj
p
masz
chce, wolny
wl
masz
chce, wolny
wp
masz
chce, wolny

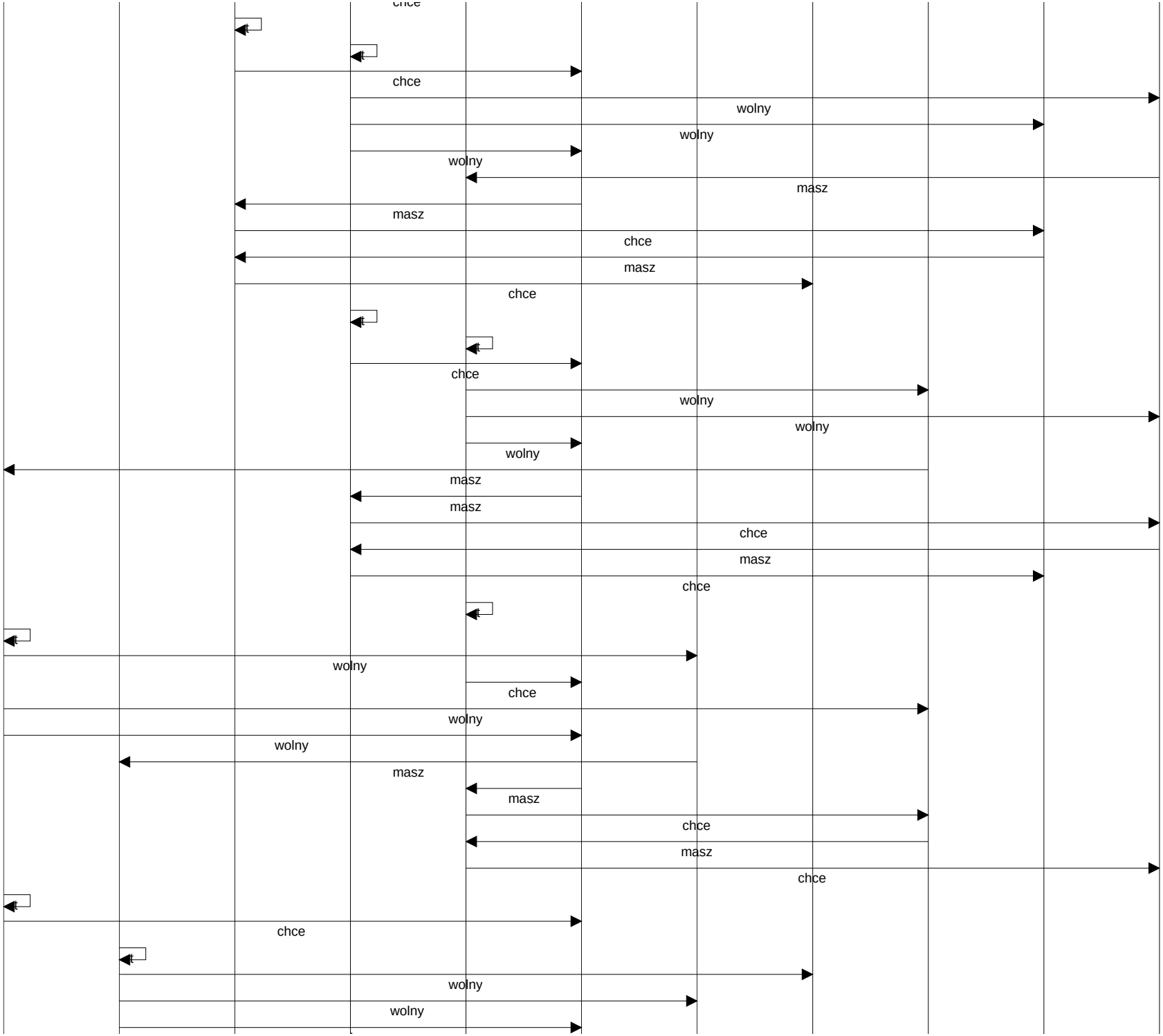
Procedure czekaj 1(1)

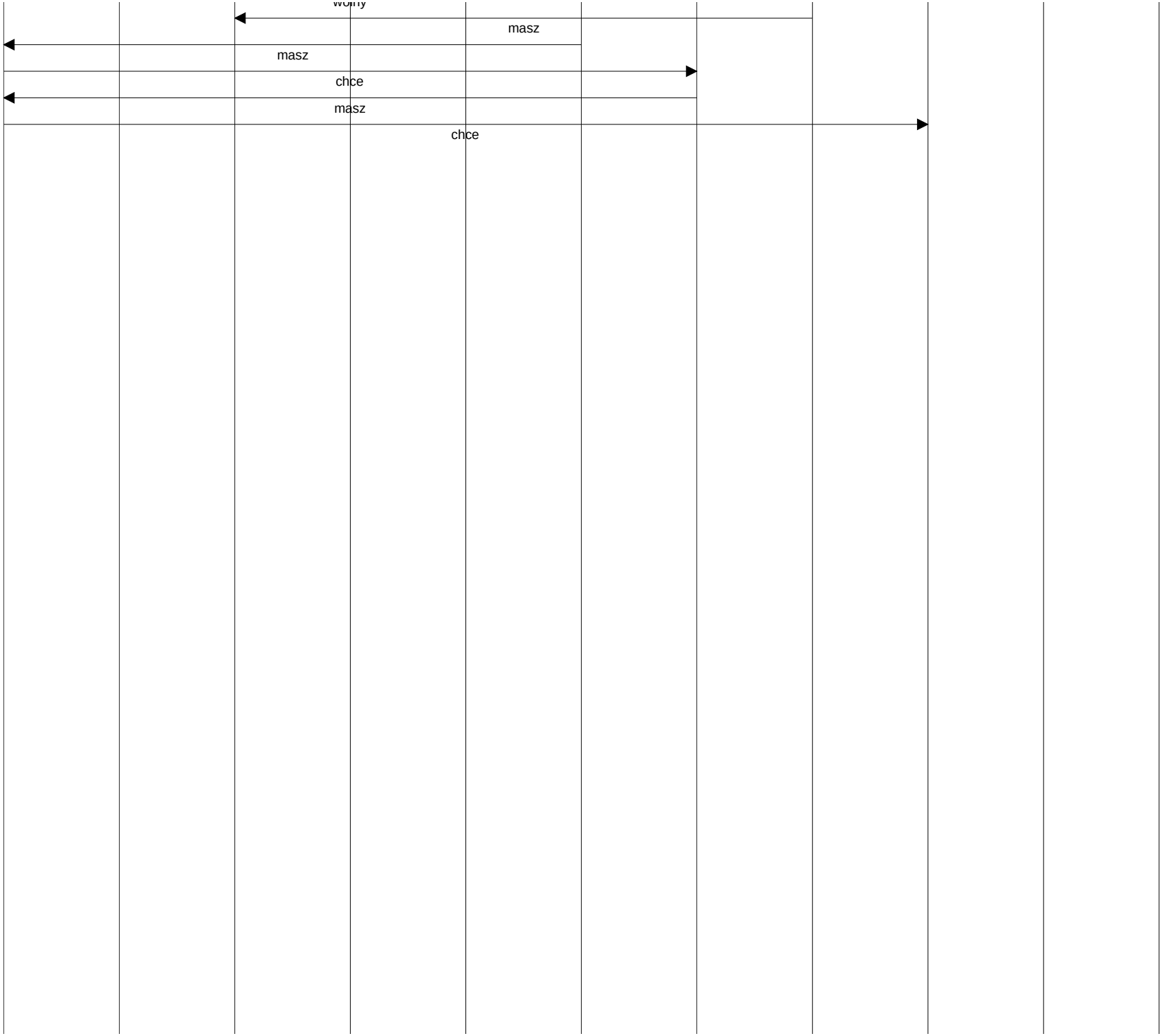



f5:7
f4:6
f3:5
f2:4
f1:3
p:2
w5:8
w4:9
w1:12
w3:10
w2:11
chce
chce
chce
chce
chce
masz
chce
masz
masz
chce
chce
masz
masz
chce
masz
chce
masz
masz
chce
chce
masz
masz
chce
wolny
wolny
wolny
masz
masz
chce


masz
chce
wolny
chce
wolny
wolny
masz
masz
chce
masz
chce
wolny
chce
wolny
wolny
masz
masz
chce
masz
chce
wolny
chce
wolny
wolny
masz
masz
chce
masz
chce
chce
wolny


wolny
wolny
masz
masz
chce
masz
chce
chce
wolny
wolny
wolny
masz
masz
chce
masz
chce
chce
wolny
wolny
wolny
masz
masz
chce
masz
chce
chce
chce
wolny
wolny
wolny
masz
masz
chce
masz


chce
chce
wolny
wolny
wolny
masz
masz
chce
masz
chce
chce
wolny
wolny
masz
masz
chce
masz
chce
wolny
chce
wolny
wolny
masz
masz
chce
masz
chce
chce
wolny
wolny
masz


wolny
masz
masz
chce
masz
chce