DCSynthG

USER MANUAL

Introduction

DCSynthG is a tool to automatically synthesize the controller for specification given in QDDC with partitioning of variabled in inputs and outputs, provided that the specificaiton is realization. A specification is said to be realizable iff there exist atleast one transducer which satisfies the requirements specified. The generated controller is basically a transducer automaton (automaton with outputs for a given input), which can then be translated into any target programming/specification language. DCSynthG currently supports two target namely SCADE/Lustre and NuSMV.

DCSynthG is based on the tool DCVALID is generates the monitor automaton for a given QDDC specification. This monitor automaton is then used to sythesize the controller for a given input-output partitioning of varibles used in the specification.

DCVALID makes use of the the validity checker for WS1S formulae, MONA1.4, from Aarhus University, Denmark. We are grateful the designers of MONA for kindly permitting us to use it. Information on MONA may be obtained on the web at the following URL:http://www.brics.dk/~mona/

We first describe the logic QDDC in the following section, which will be taken as an input by the tool DCSynthG.

QDDC Specification:

The requirement specification is given as a dcsynth file, the DC-Synth specification has the form:

```
BEGIN QDDCSYNTH <SpecificationName>
INTERFACESPEC
$var1$: INPUT
$var2$: INPUT
$var3$: OUTPUT MONITOR <Proposition1>
.................
$varm$: OUTPUT MONITOR x
;

SOFTREQS
<PropositionalFormula1> >> ..... >> <PropositionalFormulan>
;

AUXVARS
$varm+1$, ..., $vark$ ;

CONSTANTS

const1 = <integer>, ....., constn=<integer>
;

DEFINE

define <name1>[Params] as DCFormula1 ;
define <name2>[Params] as DCFormula2 ;
...................
define <name_n>[Params] as DCFormulan;
```

```
infer <SpecificationName> as
INDICATORS
 var_i : <DCFormula_i>
 ..............
 var_j : <DCFormula_j>
;

ASSUME
<name_i>[Params]
.......
<name_j>[Params]
;

REQUIRES
<name_k>[Params]
.......
<name_l>[Params]
;

SYTHESIZE
SynthG <SpecificationName>

END QDDCSYNTH
```

Now we give the informal meaning of each of the sections given in the DCSynthe specification file.

1. INTERFACESPEC:
   Interface specification list down all the input and output variables along with their type as INPUT or OUTPUT. Each variable has to be defined on newline and should be enclosed inside the $ sign. When the variables are actually used inside the definition then the $ sign is not required to be used. User also needs to define the MONITOR formula, which can be used to filter-out the counter example when specification is unrealizable. If user don't want to use MONITOR formula, then it should be mentioned as X (don't care).

2. SOFTREQS:
   SOFTREQS represent the Soft Requirements section. This section contains a prioritized list of propositional formulae seperated be >> operator, based on variables declared in INTERFACESPEC section. These formulae are used to produce a deterministic controller, when there are multiple implementations possible for the given specification. When multiple implementations are possible for the specification then the synthesis algorithm will choose the outputs based on SOFTREQS, which satisfies the maximal prefix of the list of soft requirements. This is called as locally optimal deterministic controller with respect to these soft requirements.

3. AUXVARS:
   Auxiliary variables are those variables which are niether input nor output. These are kind of internal variables which are required to modularly write the specifications. These

variables are also required to be enclosed in $ while defining.

4. CONSTANTS:
   Constants are the names with fixed value and can be used inside the specification to make it more readable.

5. DEFINES:
   This section consist of several definitions, which can then be used inside ASSUME and REQUIRES section infer section to give the requirements formula. Defines help in writing modular specification. DEFINES are macro substituted by there definition where they are used. Each definition inside DEFINES section defines a parameterized operator and its definition is given in-terms of QDDC formula. We will give the syntax and semantics of QDDC in later section of this user manual.

6. INDICATORS:
   Indicators section consist of an output variable and its corresponding formula. These variables are used to keep track of value of a corresponding formula at a given instant.

7. ASSUME:
   This section is sub-part of infer section and it lists all the environmental assumptions for the requirements to hold. This is written using the definitions given in DEFINES section. The user can write multiple assumes (each on the newline) and these finally would be converted to the conjunction of assumptions.

8. REQUIRES:
   This section is sub-part of infer section and it lists all the requirement which should hold if the environment follows the assumptions in ASSUME section. This is written using the definitions given in DEFINES section. The user can write multiple requirements (each on the newline) and finally the requirement would be converted to the conjunction these requirements. The final formula for which the sythesis algorithm tries to synthesize the controller is ASSUME imples REQUIRES i.e. the controller would try to gurantee the requirements if the assumptions hold.

9. SYTHESIZE:
   This section specifies the liveness goal for synthesis. The synthesis engine sythesizes the controller for one of the liveness goals given by SynthG, SynthF, SynthFG and SynthGF. SynthG goal asks the synthesis engine to produce a controller that meets the requirements invariantly if assumptions are met. Currently only SynthG goal is supported by the tool DCSynthG.

SynthF asks the synthesis engine to generate a controller which will eventually meet the requirement if assumptions are met. Similarly SynthFG and SynthGF ask controller to generate the controller which eventuall always satisfies the requirements and infinitely often meets the requirements respectively.

Now we give the syntax and semantics of DCFormulae which are used inside the DEFINES and INDICATORS section.

DCFormula (QDDC)

DCFormulae can have propositional state variables as parameters. Actual parameters can be arbitrary propositions. A behaviour is a finite sequence of states which (in the interval spanning the whole sequence) satisfies or violates a formula.

Notations:

We assume that P, Q are propositional state variables occurring FREE in formulas. All such variables must be declared before there use in formula. Variable names "x1", "x2", ..., "x27" etc starting with "x" followed by a number, are reserved.
However, "xx1" etc are available. The following variable names are also reserved. "define", "macro", "as", "slen", "dur", "ext", "pt", "true", "false", "ex", "all", "mu", "nu", "infer", "subword", "entire", "var", "const", INTERFACESPEC, DEFINES, AUXVARS, CONSTANTS, SOFTREQS, ASSUME, REQUIRES, INDICATORS.

We also assume that in the further desciription of syntax of Dcformula M,N are constant names and ce1, ce2 are corresponding constant expressions. DEFINES part is optional. If there are no definitions directly type "infer". Definitions cannot be recursive. A definition can have zero or more formal parameters.

Convention:

P, Q	propositional variables
A, B	propositions
D, D1, D2	QDDC formulae
c, c1	Natural number constants
ce	Constant expression.

Syntax of Propositions (states):

tt	state "true"
ff	state "false"
P	state variable "P"
st	state true only at position 0
!A	negation
A && B	conjunction
A || B	disjunction
A => B	implication
A <=> B	equivalence
(A)	brackets can be used

-A value of A in previous state.
+A value of A in next state.

precedence

!, *, + st > && > || > =>,<=>

Syntax of Constant (integer) expressions

c integer constant
x constant name (must be defined earlier)
ce1 + ce2
ce1 - ce2
(ce)

Syntax and Semantics of DCFormula

The formula occurs in scope of an alphabet ALPHA which is a finite set of propositional variables declared by the var declaration. A state assigns a truth value to each proposition in ALPHA.

STATE = ALPHA -> {0,1}

A behavior is a finite nonempty sequence of states

BEH = STATE+

Given a behaviour I, let #I denote its length. Then,

dom(I) = {0,..,#I-1}.

Let I,i |=A denote that proposition A is true at position i within behaviour I with obvious meaning. The meanings of *A and +A are defined below (these should be used with care):

I,i |= *A iff i>0 and I,i-1 |= A
I,i |= +A iff i< #I-1 and I,i+1 |= A

A formula D is evaluated to true or false for a subword (interval) of the behaviour. The set of all intervals within I is given by

INTV(I) = { [b,e] in dom(I) X dom(I) | b <= e }

I,[b,e] |= D denotes that D evaluates to true for interval [b,e], within the behaviour I.

< A > now A

I,[b,e] |= <A> iff b=e and I,b |= A

[[A]] invariant A

I,[b,e] |= [[A]] iff forall m: b<=m<=e. I,m|=A

[A] Everywhere inside A

I,[b,e] |= [A] iff b < e and
forall m: b< =m< e. I[A](m)=true

{{ A }} onestep A
I,[b,e] |= {{A}} iff e=b+1 and I,b|= A

Terms in Duration calculus have the form

slen length of interval
scount A count of how many times A is true in interval [b,e]
sdur A count of how many times A is true in [b,e] excluding e

The value of term is denoted by I,[b,e](term). Then,
I,[b,e](slen) = e - b
I,[b,e](scount A) = sum (if I,i|= A then 1 else 0) for b <= i <= e
I,[b,e](sdur A) = sum (if I,i|= A then 1 else 0) for b <= i < e

term = ce I,[b,e](term) = ce
term < ce
term <= ce
term > ce
term >= ce
(note that "ce = term" is illegal)

(Note: The support for lengths, counts and durations is quite inefficient at present. Please use these constructs with caution with large constants ce.)

ext means slen > 0
pt means slen=0
true
false

D1^D2 chop This is the only modality of Duration Calculus
I,[b,e] |= D1^D2 iff there exists m: b<=m<=e such that
I,[b,m] |= D1 and I,[m,e] |= D2

!D	not
D1 && D2	and
D1 || D2	or
D1 => D2	implies
D1 <=> D2	equivalent
(D)	Brackets for grouping

<>D somewhere D, which baically means true^D^true
[]D Everywhere D, means ~<>~D

ex P. D second order existential quantification over state P.
all P. D second order universal quantification over state P.
*D Kleene Closure using chop in place of catenation i.e. pt || D || D^D || D^D^D || .....

<< D -> A >> means ! (<> (D^<!A>)) i.e. for every subinterval where D holds, A holds at its endpoint.

{A} +> {B} means once A becomes true, B will eventually become true within the interval, and A persists till then.

{A} -> {B} UNLESS means << [A && !B] -> A || B >>
i.e. once A becomes true, it will persist till B becomes or till the end of the interval.

{A} =ce=> {B} FOLLOWS means
<< ([A] || <A>) && slen>=ce -> B >>
i.e. If A becomes true and keeps true for ce time then B will become true after the ce time. Moreover, B will persist till A remains true.

{A} <=ce= {B} TRACKS means
<< <!-A>^([[A]] && slen < ce) -> B >>
i.e. B will remain true for the FIRST ce time units of A becoming (and remaining) true. FIRST here refers to a rising edge for A (or initial time point 0).

{A} <-ce- STABLE
<< [!A]^([A] && slen < ce) -> A >>
i.e. A will persist for ce time once it becomes true.

Precedence of Operators:

Highest	!D, <>D, []D, *D
	D1^D2
	D1 && D2
	D1 || D2
	D1 => D2, D1 <=> D2
Lowest	ex P. D, all P. D

association

Operators =>, <=> associate to the right, i.e.
A => B => C means A => (B => C).

Operators &&, || associate to the left.

The validity of D over a behaviour I

I |= D iff I,[0,#I-1] |= D

Tool Usage:

Installation:

Obtain the distribution of DCSynthG from https://bitbucket.org/amolwak/dcsynthg. Go to dcsynthg/Source/Synthesis directory and execute ./compile.sh command. This will build the source

for the host. DCSynthG has been developed over the MONA source code and it also internally uses DCVALID tools and user need to install that tool for DCSynthG to work.

You also need to install the Lustre V4 tool to simulate the synthesize scade controller. Now include the path of dcsynthg/Source/Synthesis directory in your search PATH. Toos is now ready for controller synthesis.

Sythesizing the SCADE/NuSMV controller from the tool:

Create a file say dcsynthfile, containing a dcsynth spec using any text editor and execute following command in following format.

DCSynth <DC-Synth file> <target scade | smv>

e.g. to generate the SCADE controller use

DCSynth dcsynthfile scade

e.g. to generate the NuSMV controller use

DCSynth dcsynthfile smv

The tool will report whether the specification is realizable. In case it is realizable the file with extension .lus or .smv (based on the target specified) will be generated in the present directory, which will contain the controller that realizes the specification.

If the target selected for controller is scade, then it can be simulated by using luciole tool provided by lustre tool set as follows

luciole file.lus <nodeName>

If the specification is not realizable then the .lus or .smv file will say specification is not realizable and will produce the counter examples in graphviz format. You can convert this into ps file showing the counter example tree by using the following command.

aut2ps file.lus

Sythesizing the SCADE/NuSMV Monitor from the tool:

Create a file say dcsynthfile, containing a dcsynth spec using any text editor and execute following command in following format.

DCObs <DC-Synth file> <target scade | smv> <System File> <System Module Name>

e.g. to generate the SCADE Monitor use

DCObsAuxVarAsInput Minepump_Indicator_Vars scade minepump_indicatorVar .lus minepump_indicatorVar

Now a suitable model checker can be used to check the property OK is true in the main node of generated monitor.

Analysis of Requirements

Create a file say dcsynthfile, containing a dcsynth spec using any text editor and execute following command in following format.

DCAnalyze <DC-Synth file>

This will output the satisfying and non-satisfying example for the specification.

Example (Minepump)

Imagine a minepump which keeps the water level in a mine under control for the safety of miners. The pump is driven by a controller which can switch it on and off. Mines are prone to methane leakage trapped underground which is highly flammable. So as a safety measure if a methane leakage is detected the controller is not allowed to switch on the pump under no circumstances.

The controller has two input sensors - HH2O which becomes 1 when water level is high, and HCH4 which is 1 when there is a methane leakage; and can generate two output signals - ALARM which is set to 1 to sound/persist the alarm, and PUMPON which is set to 1 to switch on the pump. The objective of the controller is to safely operate the pump and the alarm in such a way that the water level is never dangerous, indicated by the indicator variable DH2O, whenever certain assumptions hold. We have the following assumptions on the mine and the pump.

- Sensor reliability assumption: If HH2O is false then so is DH2O.
- Water seepage assumptions: The minimum no. of cycles for water level to become dangerous once it becomes high is k1.
- Pump capacity assumption: If pump is switched on for at least k2 + 1 cycles then water level will not be high after k2 cycles.
- Methane release assumptions: The minimum separation between the two leaks of methane is k3 cycles and the methane leak cannot persist for more than k4 cycles.
- Initial condition assumption: Initially neither the water level is high nor there is a methane leakage.

The requirements are:

- Alarm control: If the water level is dangerous then alarm will be high after k5 cycles and if there is a methane leakage then alarm will be high after k6 cycles. If neither the water level is dangerous nor there is a methane leakage then alarm should be off after k7 cycle.
- Safety condition: The water level should never become dangerous and whenever there is a methane leakage, pump should be off.

These requirements are formally given in dcspec file as follows:
File Name: Minepump

```
BEGIN QDDCSYNTH minepump_indicator
INTERFACESPEC
$HH2Op$: INPUT
$HCH4p$: INPUT
$ALARMp$: OUTPUT MONITOR x
$PUMPONp$: OUTPUT MONITOR x
$YHCH4p$: OUTPUT MONITOR x
;

SOFTREQS
((!$YHCH4p$)|(!$PUMPONp$))
;

AUXVARS
$DH2O$ ;

CONSTANTS
-- delta   response time of PUMP and ALARMS after trigger
      -- epsilon time taken by pump to bring water level to below HH2O
      -- w       time taken for water level to be dangerous after high
      -- zeta        minimum separation between two methane leaks.
      -- kappa   maximum duration of methane leak.
```

```
delta = 1, w = 8, epsilon=2 , zeta=10, kappa=1
--delta = 1, w = 6, epsilon=3, zeta=6, kappa=2
;

DEFINE
-- Alarm control

define alarm1[HH2O, DH2O, HCH4, ALARM, PUMPON] as
  {DH2O} =delta=> {ALARM} ;

define alarm2[HH2O, DH2O, HCH4, ALARM, PUMPON] as
  {HCH4} =delta=> {ALARM} ;

define alarm3[HH2O, DH2O, HCH4, ALARM, PUMPON] as
  { !HCH4 && !DH2O } =delta=> {!ALARM}   ;

-- Water seepage Assumptions
define water1[HH2O, DH2O, HCH4, ALARM, PUMPON] as
   [] ( [[ DH2O => HH2O ]] ) ;

define water2[HH2O, DH2O, HCH4, ALARM, PUMPON] as
   {HH2O} <=w= {! DH2O}  ;

-- Pump capacity assumption

define pumpcap1[HH2O, DH2O, HCH4, ALARM, PUMPON] as

  {PUMPON} =epsilon=> {!HH2O} ;

--  Methane Release assumptions

define methane1[HH2O, DH2O, HCH4, ALARM, PUMPON] as
   [] ( [HCH4]^[!HCH4]^<HCH4> => slen > zeta ) ;

define methane2[HH2O, DH2O, HCH4, ALARM, PUMPON] as
   [] ( [[HCH4]] => slen < kappa ) ;

-- Initial condition assumption
define initdry[HH2O, DH2O, HCH4, ALARM, PUMPON] as
  <!HH2O>^true ;

-- Pump control
define pump1[HH2O, DH2O, HCH4, ALARM, PUMPON] as
  { HH2O && !HCH4 } =delta=> {PUMPON}  ;

define pump2[HH2O, DH2O, HCH4, ALARM, PUMPON] as
  { !HH2O || HCH4 } =delta=> {!PUMPON} ;

-- safety condition
define safe[HH2O, DH2O, HCH4, ALARM, PUMPON] as
[[!DH2O && ( (HCH4 || !HH2O) => !PUMPON)]];


define plant[HH2O, DH2O, HCH4, ALARM, PUMPON] as
initdry[HH2O, DH2O, HCH4, ALARM, PUMPON] &&
water1[HH2O, DH2O, HCH4, ALARM, PUMPON] &&
water2[HH2O, DH2O, HCH4, ALARM, PUMPON] &&
pumpcap1[HH2O, DH2O, HCH4, ALARM, PUMPON] &&
methane1[HH2O, DH2O, HCH4, ALARM, PUMPON] &&
methane2[HH2O, DH2O, HCH4, ALARM, PUMPON];

define req[HH2O, DH2O, HCH4, ALARM, PUMPON] as
safe[HH2O, DH2O, HCH4, ALARM, PUMPON];
```

```
infer minepump_indicatorVar as
INDICATORS
 YHCH4p : (slen=2 && <><HCH4p>)
;
ASSUME
plant[HH2Op, DH2O, HCH4p, ALARMp, PUMPONp]
true
;
REQUIRES
true
req[HH2Op, DH2O, HCH4p, ALARMp, PUMPONp]
;

SYTHESIZE
SynthG minepump_indicatorVar

END QDDCSYNTH
```

Now the tool can be used to generate a the controller for this specification. The SCADE/Lustre controller for this specification is as follows:

```
-- Var_type: INPUT    to be Monitored is : NIL
-- Var_type: INPUT    to be Monitored is : NIL
-- Var_type: OUTPUT   to be Monitored is : NIL
-- Var_type: OUTPUT   to be Monitored is : NIL
-- Var_type: OUTPUT   to be Monitored is : NIL
--Soft reqirements are =  ((!$4$)|(!$3$)) --starting MPNC construction
--no.of states in original aut / no. of states in MPNC aut = 238 /153
--starting LODC construction
--no.of states in MPNC aut / no. of states in LODC aut = 153 /45
const dontCare = false;
node minepump_indicatorVar ( HH2Op, HCH4p:bool) returns ( ALARMp, PUMPONp,
YHCH4p:bool)
 var cstate: int;
let
 ALARMp, PUMPONp, YHCH4p, cstate =
 ( if true  and not HH2Op and not HCH4p then ( true,  false,  false,   2)
  else if true  and not HH2Op and HCH4p then ( true,  false,  false,   3)
  else if true  and HH2Op then ( true,  true,  false,   4)
 else ( dontCare,  dontCare,  dontCare,  1)) ->
 if pre cstate = 1 and not HH2Op and not HCH4p then ( true,  false,  false, 2)
 else if pre cstate = 1 and not HH2Op and HCH4p then ( true,  false,  false, 3)
 else if pre cstate = 1 and HH2Op then ( true,  true,  false, 4)
 else if pre cstate = 2 and not HH2Op and not HCH4p then ( true,  false,  false,
5)
 else if pre cstate = 2 and not HH2Op and HCH4p then ( true,  false,  false, 6)
 else if pre cstate = 2 and HH2Op and not HCH4p then ( true,  true,  false, 7)
 else if pre cstate = 2 and HH2Op and HCH4p then ( true,  false,  false, 6)
 else if pre cstate = 3 and not HH2Op and not HCH4p then ( true,  false,  false,
8)
 else if pre cstate = 3 and not HH2Op and HCH4p then ( true,  true,  false, 4)
 else if pre cstate = 3 and HH2Op and not HCH4p then ( true,  true,  false, 8)
 else if pre cstate = 3 and HH2Op and HCH4p then ( true,  true,  false, 4)
 else if pre cstate = 4 then ( true,  true,  false, 4)
 else if pre cstate = 5 and not HH2Op and not HCH4p then ( true,  false,  false,
5)
 else if pre cstate = 5 and not HH2Op and HCH4p then ( true,  false,  true, 6)
 else if pre cstate = 5 and HH2Op and not HCH4p then ( true,  true,  false, 7)
 else if pre cstate = 5 and HH2Op and HCH4p then ( true,  false,  true, 6)
```

```
 else if pre cstate = 6 and not HCH4p then ( true,  false,  true, 8)
 else if pre cstate = 6 and HCH4p then ( true,  true,  false, 4)
 else if pre cstate = 7 and not HH2Op and not HCH4p then ( true,  false,  false,
5)
 else if pre cstate = 7 and not HH2Op and HCH4p then ( true,  false,  true, 6)
 else if pre cstate = 7 and HH2Op and not HCH4p then ( true,  true,  false, 9)
 else if pre cstate = 7 and HH2Op and HCH4p then ( true,  false,  true, 6)
 else if pre cstate = 8 and not HCH4p then ( true,  false,  true, 10)
 else if pre cstate = 8 and HCH4p then ( true,  true,  false, 4)
 else if pre cstate = 9 and not HH2Op and not HCH4p then ( true,  false,  false,
5)
 else if pre cstate = 9 and not HH2Op and HCH4p then ( true,  false,  true, 6)
 else if pre cstate = 9 and HH2Op then ( true,  true,  false, 4)
 else if pre cstate = 10 and not HH2Op and not HCH4p then ( true,  false,
false, 14)
 else if pre cstate = 10 and not HH2Op and HCH4p then ( true,  true,  false, 4)
 else if pre cstate = 10 and HH2Op and not HCH4p then ( true,  true,  false, 15)
 else if pre cstate = 10 and HH2Op and HCH4p then ( true,  true,  false, 4)
 else if pre cstate = 14 and not HH2Op and not HCH4p then ( true,  false,
false, 19)
 else if pre cstate = 14 and not HH2Op and HCH4p then ( true,  true,  false, 4)
 else if pre cstate = 14 and HH2Op and not HCH4p then ( true,  true,  false, 20)
 else if pre cstate = 14 and HH2Op and HCH4p then ( true,  true,  false, 4)
 else if pre cstate = 15 and not HH2Op and not HCH4p then ( true,  false,
false, 19)
 else if pre cstate = 15 and not HH2Op and HCH4p then ( true,  true,  false, 4)
 else if pre cstate = 15 and HH2Op and not HCH4p then ( true,  true,  false, 21)
 else if pre cstate = 15 and HH2Op and HCH4p then ( true,  true,  false, 4)
 else if pre cstate = 19 and not HH2Op and not HCH4p then ( true,  false,
false, 24)
 else if pre cstate = 19 and not HH2Op and HCH4p then ( true,  true,  false, 4)
 else if pre cstate = 19 and HH2Op and not HCH4p then ( true,  true,  false, 25)
 else if pre cstate = 19 and HH2Op and HCH4p then ( true,  true,  false, 4)
 else if pre cstate = 20 and not HH2Op and not HCH4p then ( true,  false,
false, 24)
 else if pre cstate = 20 and not HH2Op and HCH4p then ( true,  true,  false, 4)
 else if pre cstate = 20 and HH2Op and not HCH4p then ( true,  true,  false, 26)
 else if pre cstate = 20 and HH2Op and HCH4p then ( true,  true,  false, 4)
 else if pre cstate = 21 and not HH2Op and not HCH4p then ( true,  false,
false, 24)
 else if pre cstate = 21 and not HH2Op and HCH4p then ( true,  true,  false, 4)
 else if pre cstate = 21 and HH2Op then ( true,  true,  false, 4)
 else if pre cstate = 24 and not HH2Op and not HCH4p then ( true,  false,
false, 30)
 else if pre cstate = 24 and not HH2Op and HCH4p then ( true,  true,  false, 4)
 else if pre cstate = 24 and HH2Op and not HCH4p then ( true,  true,  false, 31)
 else if pre cstate = 24 and HH2Op and HCH4p then ( true,  true,  false, 4)
 else if pre cstate = 25 and not HH2Op and not HCH4p then ( true,  false,
false, 30)
 else if pre cstate = 25 and not HH2Op and HCH4p then ( true,  true,  false, 4)
 else if pre cstate = 25 and HH2Op and not HCH4p then ( true,  true,  false, 32)
 else if pre cstate = 25 and HH2Op and HCH4p then ( true,  true,  false, 4)
 else if pre cstate = 26 and not HH2Op and not HCH4p then ( true,  false,
false, 30)
 else if pre cstate = 26 and not HH2Op and HCH4p then ( true,  true,  false, 4)
 else if pre cstate = 26 and HH2Op then ( true,  true,  false, 4)
 else if pre cstate = 30 and not HH2Op and not HCH4p then ( true,  false,
false, 33)
 else if pre cstate = 30 and not HH2Op and HCH4p then ( true,  true,  false, 4)
 else if pre cstate = 30 and HH2Op and not HCH4p then ( true,  true,  false, 34)
 else if pre cstate = 30 and HH2Op and HCH4p then ( true,  true,  false, 4)
 else if pre cstate = 31 and not HH2Op and not HCH4p then ( true,  false,
false, 33)
 else if pre cstate = 31 and not HH2Op and HCH4p then ( true,  true,  false, 4)
```

```
 else if pre cstate = 31 and HH2Op and not HCH4p then ( true,  true,  false, 35)
 else if pre cstate = 31 and HH2Op and HCH4p then ( true,  true,  false, 4)
 else if pre cstate = 32 and not HH2Op and not HCH4p then ( true,  false,
false, 33)
 else if pre cstate = 32 and not HH2Op and HCH4p then ( true,  true,  false, 4)
 else if pre cstate = 32 and HH2Op then ( true,  true,  false, 4)
 else if pre cstate = 33 and not HH2Op and not HCH4p then ( true,  false,
false, 36)
 else if pre cstate = 33 and not HH2Op and HCH4p then ( true,  true,  false, 4)
 else if pre cstate = 33 and HH2Op and not HCH4p then ( true,  true,  false, 37)
 else if pre cstate = 33 and HH2Op and HCH4p then ( true,  true,  false, 4)
 else if pre cstate = 34 and not HH2Op and not HCH4p then ( true,  false,
false, 36)
 else if pre cstate = 34 and not HH2Op and HCH4p then ( true,  true,  false, 4)
 else if pre cstate = 34 and HH2Op and not HCH4p then ( true,  true,  false, 38)
 else if pre cstate = 34 and HH2Op and HCH4p then ( true,  true,  false, 4)
 else if pre cstate = 35 and not HH2Op and not HCH4p then ( true,  false,
false, 36)
 else if pre cstate = 35 and not HH2Op and HCH4p then ( true,  true,  false, 4)
 else if pre cstate = 35 and HH2Op then ( true,  true,  false, 4)
 else if pre cstate = 36 and not HH2Op and not HCH4p then ( true,  false,
false, 39)
 else if pre cstate = 36 and not HH2Op and HCH4p then ( true,  true,  false, 4)
 else if pre cstate = 36 and HH2Op and not HCH4p then ( true,  true,  false, 40)
 else if pre cstate = 36 and HH2Op and HCH4p then ( true,  true,  false, 4)
 else if pre cstate = 37 and not HH2Op and not HCH4p then ( true,  false,
false, 39)
 else if pre cstate = 37 and not HH2Op and HCH4p then ( true,  true,  false, 4)
 else if pre cstate = 37 and HH2Op and not HCH4p then ( true,  true,  false, 41)
 else if pre cstate = 37 and HH2Op and HCH4p then ( true,  true,  false, 4)
 else if pre cstate = 38 and not HH2Op and not HCH4p then ( true,  false,
false, 39)
 else if pre cstate = 38 and not HH2Op and HCH4p then ( true,  true,  false, 4)
 else if pre cstate = 38 and HH2Op then ( true,  true,  false, 4)
 else if pre cstate = 39 and not HH2Op and not HCH4p then ( true,  false,
false, 5)
 else if pre cstate = 39 and not HH2Op and HCH4p then ( true,  true,  false, 4)
 else if pre cstate = 39 and HH2Op and not HCH4p then ( true,  true,  false, 7)
 else if pre cstate = 39 and HH2Op and HCH4p then ( true,  true,  false, 4)
 else if pre cstate = 40 and not HH2Op and not HCH4p then ( true,  false,
false, 5)
 else if pre cstate = 40 and not HH2Op and HCH4p then ( true,  true,  false, 4)
 else if pre cstate = 40 and HH2Op and not HCH4p then ( true,  true,  false, 9)
 else if pre cstate = 40 and HH2Op and HCH4p then ( true,  true,  false, 4)
 else if pre cstate = 41 and not HH2Op and not HCH4p then ( true,  false,
false, 5)
 else if pre cstate = 41 and not HH2Op and HCH4p then ( true,  true,  false, 4)
 else if pre cstate = 41 and HH2Op then ( true,  true,  false, 4)
 else ( dontCare, dontCare, dontCare, pre cstate) ;

tel
```