DCSYNTH: Guided Reactive Synthesis with Soft Requirements

Amol Wakankar[2], Paritosh Pandya[1], Rajmohan Mattaplacket[1]

Tata Institute of Fundamental Research, Mumbai
Bhabha Atomic Research Center, Mumbai

November 24, 2017

Requirement Modelling and Analysis

Requirement notation may be Informal, Formal, Visual, Textual. Some form of temporal logic is used for Formal Specification of embedded system components.

Promise of Formal Specification

- Unambiguous
- Requirement Analysis: Consistency, Completeness, implication checking, realizability checking.
- Verification: Model checking, Runtime verification, Automatic test suite generation.
- Synthesis: Automatic construction of controller which matches the specification.

Some Consistency Checking Questions

Specification $D(I, O)$.

- Is specification CONSISTENT? $\exists I \exists O . D(I, O)$
- Is It RECEPTIVE? $\forall I \exists O . D(I, O)$
- Is it REALIZABLE? Is there a Mealy Machine $O = f(I)$ s.t. $\forall I . D(I, f(I))$
- Is the specification COMPLETE? What does it mean?

Use Case Completeness

Is the specification consistent with each of the use case?
Let $UseCase(I, O)$ be a formula describing example behaviour.
$\exists I \exists O . D(I, O) \wedge UseCase(I, O)$ must be satisfiabile.

DCTOOLS: Architecture

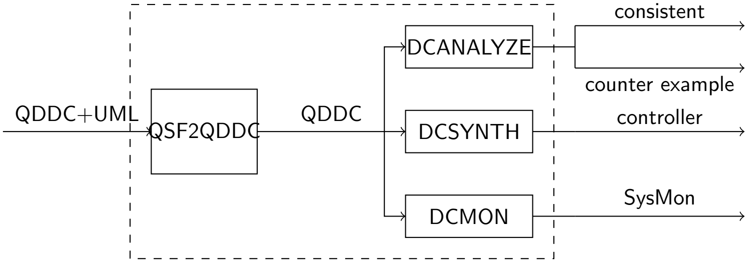


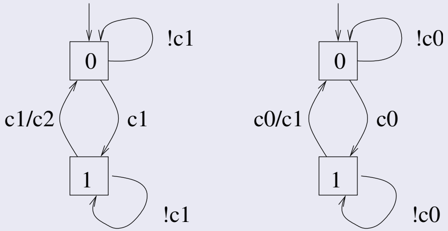
Figure : DCTOOLS.

Synchronous Programs

Embedded System Controllers, Clocked Digital Hardware Circuits

Automaton Model: Mealy Machines

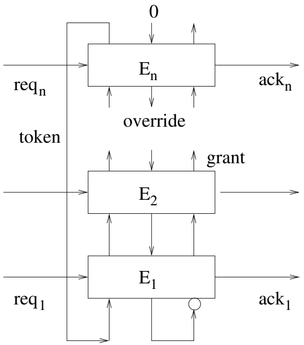
Example: Two bit counter



Synchronous Programming Languages

- Esterel, SCADE/Lustre
- Simulink
- Verilog/VHDL (with some restrictions)

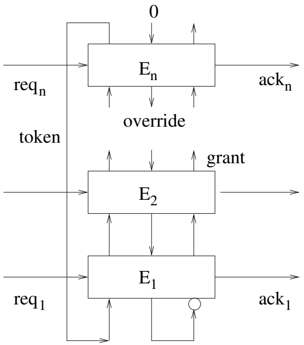
MacMillan's Arbiter

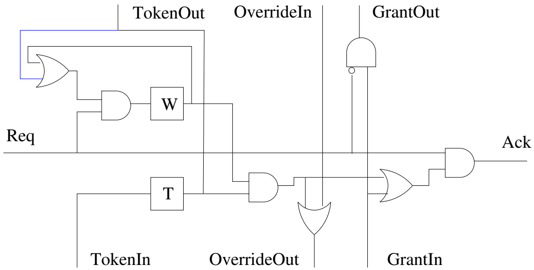


In each cycle

- a subset of requests is high.
- Mutex At most one of the acks must be high.
- Fairness Every cell must get a chance!

MacMillan's Arbiter





In each cycle

- a subset of requests is high.
- Mutex At most one of the acks must be high.
- Fairness Every cell must get a chance!

Invariant Properties

- Mutual exclusion: In any excution, At any time: at most one ack must be true.

  $G \ \bigwedge_{i \neq j} \ \neg(ack_i \wedge ack_j)$
- No Spurious acknowledgements: ack only if req

  $G \ \bigwedge_{i} \ (ack_i \Rightarrow req_i)$
- No Lost Cycles: A cycle is lost if there are requests but there is no ack.

  $G \neg Lostcycle$ where

  $Lostcycle \overset{\text{def}}{=} (\vee req_i) \ \wedge \ \neg(\vee ack_i)$

How do we verify these properties? How do we state these properties precisely?

Complex Properties

- Fairness If request is kept continuously high, eventually there will be acknowledgement.

  $GF\ (\neg req_i \ \vee \ ack_i)$

- Response time: The number of cycles for which req_i must be continuously high to guarantee an ack_i signals is at most k.

  ```
  G[]( ([[req]] && slen=k-1) => <><ack>) )
  ```

- 3-cycle response time: The number of cycles for which req_i must be continuously high to guarantee 3 ack_i signals is at most k.

  ```
  G[]( ([[req]] && slen=k-1) => scount ack >= 3) )
  ```

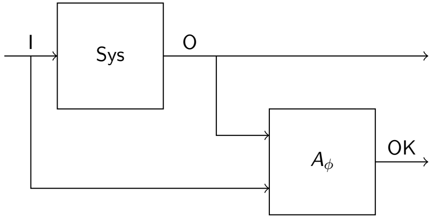
Some Background

Given requirement formula $\phi(I, O)$

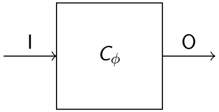
- Monitor Synthesis Problem [Buchi] construct monitor automaton A_ϕ such that $L(\phi) = L(A_\phi)$. Adapted to Temporal logic LTL [Pnueli, Vardi-Wolper].
- Controller Synthesis problem [Church] construct mealy machine C_ϕ giving $O = C(I)$ such that $\forall I.\ \ \phi(\ I,\ \ C(I)\ )$.
- Solved for logic MSO using Buchi-Landweber games [Thomas].
- LTL controller synthesis [Pnueli]
- Finite state automaton based synthesis [Wonham-Ramage]
- Strategy Synthesis for Gail-Shapely graph games with Borel objectives [Martin].
- GR1 synthesis in polynomial time [Peterman et al]

Requirement Monitor vs Controller

Given complex specification $\phi(I, O)$



Requirement Aware Design.



Controller (correct by construction).

$$\forall I.\ \ \phi(\ I,\ C(I)\ )$$

Approach: Safety Synthesis

Given specification $\phi(I, O)$

- Monitor Automaton Construction Construct language equivalent deterministic finite automaton using Monitor Synthesis.
- Maximally Permissive Non-deterministic Controller (MPNC) Construction Remove edges going to bad states. Remove unrealizable states. Repeat till stable. (Attractor strategy for bad states).
- Deterministic Controller (DetC) Construction Resolve nondeterministic choice between outputs.

Logic QDDC: Specifying Regular Patterns

A powerful logic to specify regular languages (finite automata). Based on Duration Calculus of [Zhou, Hoare and Ravn, 1991].

- Like Regular Expressions with intersection and negation
- Counting and Quantification of Temporal Variables.
- uses CHOP instead of CATENATION
  (For theory see chapter in Modern Applications of Automata)
- Specifies patterns within behaviour (which match subintervals
  .

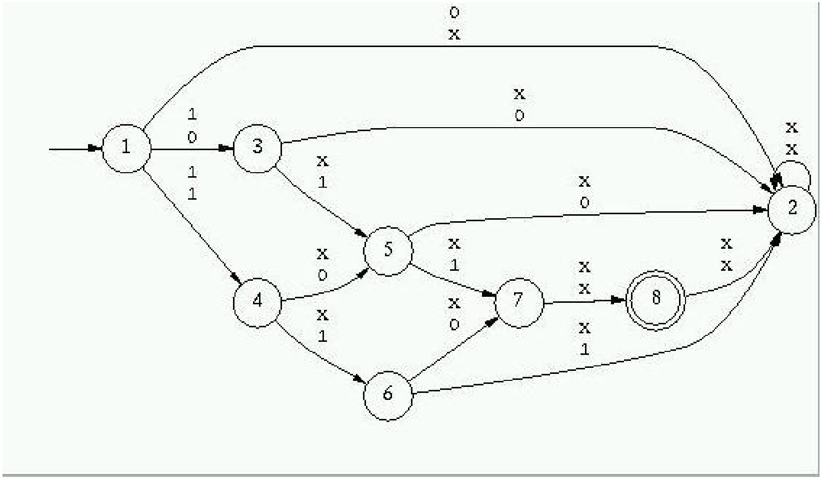
DCVALID

Model checker for QDDC formulae.

- QDDC to Symbolic FSA conversion
- Based on MONA: Efficient BDD based representation of transition relation.
- Available on Web since 1997.

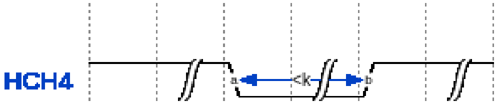
DCVALID Example

```
(<P>^true) && (slen =(4-1)) && (sdur Q = 2)
```



QDDC+

- Powerful and Visual Notation.
- Example Between two successive episods of *HCH*4 there are at most than *k* cycles.
  `![] ([HCH4]^([!HCH4] && slen >= k) ^ <HCH4>)`
- Generalises QDDC to specify Mealy Machines.



Formula Automaton Construction

Theorem (Automata Theoretic Decidability of *QDDC*)

- *For each $D \in QDDC$ we can effectively construct finite state automaton A_D such that $L(D) = L(A_D)$.*
- *For each FSM A we can effectively construct $D_A \in QDDC$ such that $L(A) = L(D_A)$.*

Tool DCVALID – next slide.

Problem

Size of minimum automaton can be non-elementary in size of formula in the worst case. Thus formula of size n automaton size $O(2^{2^{2^{\cdots}}})$, tower of height n.

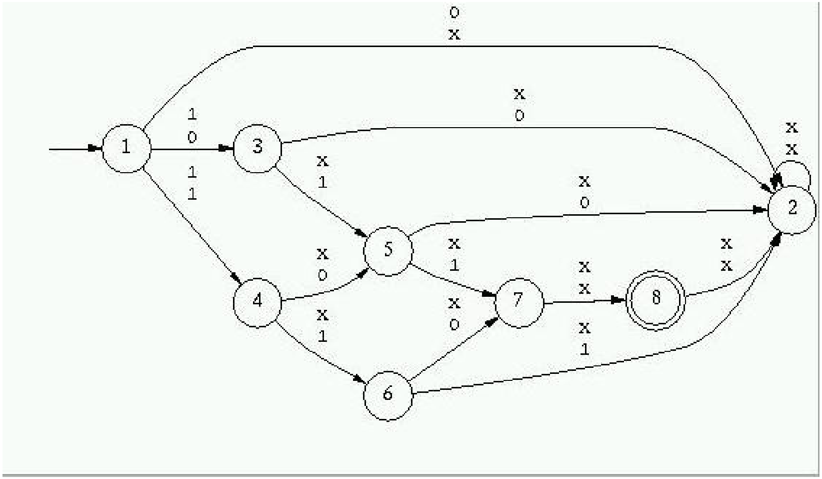
DCVALID: Validity/Model Checker for QDDC formulas

- Constructs finite state deterministic automaton $\mathcal{A}(\mathcal{D})$ for QDDC formula D.
- The automaton is used as a synchronous observer to model check QDDC properties of Esterel, SMV, Verilog, SCADE/Lustre and SAL models.
- Uses efficient BDD-based representation of automata using MONA.
- Constructs automaton for formula bottom up keeping each automaton in minimal deteterminstic form.

[RTTOOLS2001, TACAS2001, SLAP2002, AVOCS2004, FSTTCS2005, TACAS2006]

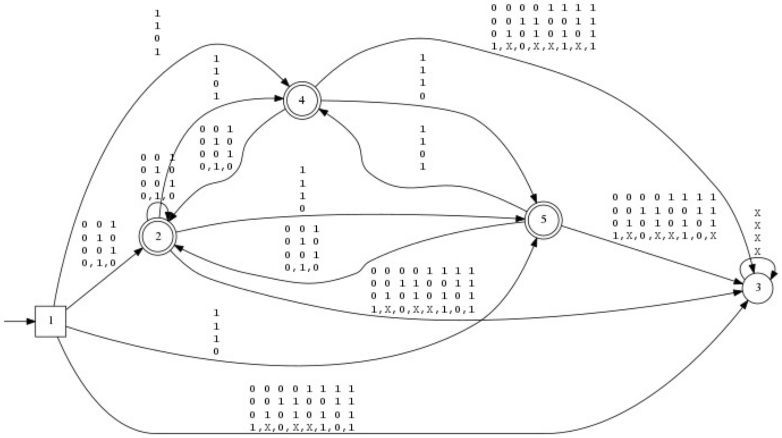
DCVALID Example

$$(\langle P \rangle \frown true) \wedge (slen = 4) \wedge (sdurQ = 2)$$



Monitor Automaton

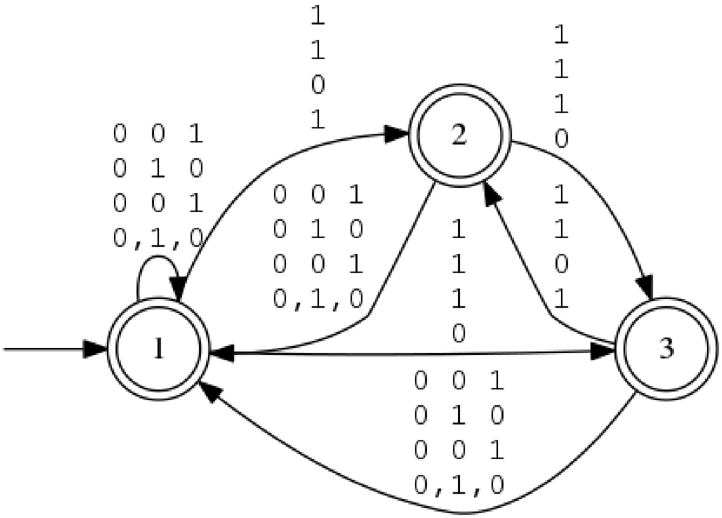
Property monitor automaton specifying Mealy Machine



1
2
3
4
5
1 1 0 1
1 1 0 1
0 0 1 / 0 1 0 / 0 0 1 / 0,1,0
0 0 1 / 0 1 0 / 0 0 1 / 0,1,0
0 0 1 / 0 1 0 / 0 0 1 / 0,1,0
0 0 1 / 0 1 0 / 0 0 1 / 0,1,0
1 1 1 0
1 1 1 0
1 1 0 1
1 1 1 0
0 0 0 0 1 1 1 1 / 0 0 1 1 0 0 1 1 / 0 1 0 1 0 1 0 1 / 1,X,0,X,X,1,X,1
0 0 0 0 1 1 1 1 / 0 0 1 1 0 0 1 1 / 0 1 0 1 0 1 0 1 / 1,X,0,X,X,1,0,X
0 0 0 0 1 1 1 1 / 0 0 1 1 0 0 1 1 / 0 1 0 1 0 1 0 1 / 1,X,0,X,X,1,0,1
0 0 0 0 1 1 1 1 / 0 0 1 1 0 0 1 1 / 0 1 0 1 0 1 0 1 / 1,X,0,X,X,1,0,1
X X X X

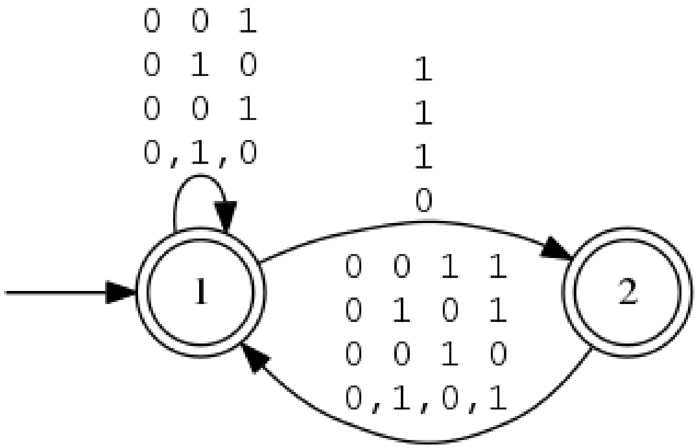
MPNC Controller

Maximally Permissive Nondeterminstic Controller



LODC

Locally Optimal Deterministic Controller



Guided Controller Synthesis

- Controllers are underspecified. How to choose between controllers? Optimization!
- Resolving Conflicting requirements.

Our approach: Soft Requirements. Requirements are past time temporal formulas.

- Requirements partitioned into hard and soft requirements (with priorities).
- Hard requirements must be invariantly satisfied.
- Soft requirements are satisfied “as much as possible” in best effort manner.

DCSYNTH Specification

Specification $S = (I, O, D^h, \wedge_{i=1}^{i=k}(w_i \iff D_i^s), \langle P_1, \cdots, P_l \rangle)$

- Input and output variables I, O respectively.
- D^h conjunction of hard requirements.
- D_i^s soft requirement witnessed by proposition w_i
- $\langle P_1, \cdots, P_l \rangle$ lexicographically prioritized list of literals from $O \cup W$.

Locally Optimal Determinstic Controller (LODC)

At each step, from available outputs in MPNC, choose one that maximizes the soft requirements.

Claim: This simplistic strategy seems to work well!

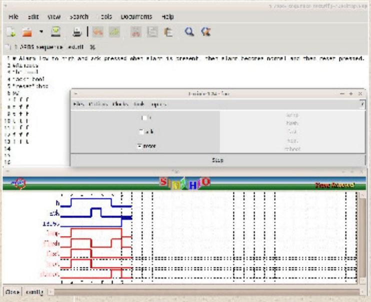
Case Studies

- Bus Arbiter automatic synthesis of arbiter code.
- Mine Pump automantic synthesis of pump controller.
- Traffic Light automatic synthesis of controller
- Alarm anunciation automatic synthesis of alarm controller (BARC Example)
- Discordance logic automatic synthesis of controller (BARC Example)
- Amba Bus Controller automatic controller synthesis (ongoing)
- Autonomous Parking Robot Controller (ongoing)

AAS

```
-- st1, st2;
-- lamp flash fast group has states {off, fastflash, steady, slowflash}
-- hoot,rbhoot has two states {off, hoot, rbhoot}
-- st1,st2  := 00  normal 10 abnormal 01 acknowledged 11 unreset
var h,ack,reset,lamp,flash,fast,hoot,rbhoot;

define unless[P11,Q11] as
  !(([[!Q11]])^<!P11>^true);

define persist[P111,Q111] as
[](<P111>^ext => slen=1^unless[P111,Q111]);

define transit[P,Q,R] as
[](<P >^slen=1^<Q> => true^<R>);

infer
ex st1. ex st2.
<!st1 && !st2>^true &&  -- initial state
persist[!st1 && !st2, h] &&
transit[!st1 && !st2, h, (st1 && !st2) ]  &&

persist[st1 && !st2,(ack) ] &&
transit[st1 && !st2, ack && h,!st1 && st2]  &&

transit[st1 && !st2, ack && !h,st1 && st2]  &&

persist[!st1 && st2, !h] &&
transit[!st1 && st2, !h, st1 && st2] &&

persist[st1 && st2, !h && reset || h] &&
transit[st1 && st2, !h&&reset, !st1 && !st2] &&

transit[st1 && st2, h, st1 && !st2] &&
[[ !st1 && !st2 => !lamp && !hoot && !rbhoot ]] &&
[[ st1 && !st2 => lamp && flash && fast && hoot && !rbhoot]] &&
[[ !st1 && st2 => lamp && !flash && !hoot && !rbhoot]] &&
[[ st1 && st2 => lamp && flash && !fast && !hoot && rbhoot]] && true.
```

Change of state	**OUTPUT**	
At INPUT	**Lamp**	**Audio**
Normal to Alarm	Fast Flash	Normal Alarm Hooter On
Acknowledged	**Steady**	**Normal Alarm Hooter Off**
Alarm to Normal	Slow Flash	Ring Back Hooter On
Reset	**Off**	**Ring Back Hooter Off**



Robust Controller Synthesis

Specification is divided into set of assumptions A and commitments C. These hold intermittently during execution.

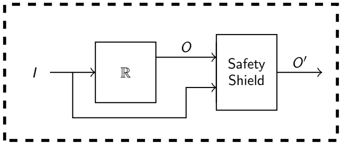
- Be-Correct `G(Pref(A) => C)`.
- Degraded-Performance `G((A => C) && (Ad => Cd)`.
- Greedy with soft requirement `C`.
- Never-Give-Up `G(A => C)` with soft requirement `C`.
- k, b-variant
  `G([]((slen = b && scount !A <= k) => C))`.

Robust Synthesis Results

Robustness Criteria	Specification		Monitor		MPNC		LODC	
	Hard	Soft	States	Time	States	Time	States	Time
BeCorrect	$Arb^{hard}_{assume}(4,2,2)$	-	14		7	0.001	6	0.003
Never Giveup	$Arb^{hard}_{assume}(4,2,2)$	$(ARBINV >> Resp(req_1, ack_1, 2) >> \ldots Resp(req_4, ack_4, 2)$	23		22	0.05	17	0.07
K-Bounded(K=2)	-	-	55	0.06	54	0.007	29	0.006
K-B-Resilient	Unrealizable	-						
K-B-variant (K=2, B=3)	$Viol(Assum^{spec}, 2, 3) => (ARBINV \wedge Resp^{spec}(4,3))$	-	158	0.14	48	0.04	27	0.01
K-B-variant-2	Unrealizable	-						
Greedy	-	$(ARBINV >> Resp(req_1, ack_1, 2) >> \ldots Resp(req_4, ack_4, 2)$	18		17	0.04	15	0.07

Shield Synthesis

A shield corrects the output of a (possibly incorrect) controller O=R(I) to ensure requirement $REQ(I, O')$.



Criteria

- Correctness $REQ(I, O')$ always holds.
- Minimum Deviation $O \neq O'$ as much as possible.

Several notions:

- K-Shield (Bloem et al)
- Burst Error Shield (Wu et al)

Specifying Shield Synthesis

Conservative burst error shield

- Input: $I \cup O$. Output: O'
- Hard requirement: `REQ[O/O']`
- Soft requirement: `pref(true^<O=O'>)`

Shield Synthesis Results

Guarantees	States	K-Shield		Burst error shield		Conservative burst error shield	
		states	time	states	time	states	time
Toyota Powertrain	23	38	0.2	38	0.3	9	0.7
Traffic light	4	7	0.1	7	0.2	4	0.007
$F_{64}p$	67	67	0.7	67	0.5	67	0.002
$F_{256}p$	259	259	46.9	259	10.5	259	0.01
$F_{512}p$	515	515	509.1	515	54.4	515	0.07
$G(\neg q) \vee F_{64}(q \wedge F_{64}p)$	67	67	0.8	67	0.6	67	0.007
$G(\neg q) \vee F_{256}(q \wedge F_{256}p)$	259	259	46.2	259	10.7	259	0.04
$G(\neg q) \vee F_{512}(q \wedge F_{512}p)$	515	515	571.7	515	54.5	515	0.1
$G(q \wedge \neg r \rightarrow (\neg r \cup_4 (p \wedge \neg r)))$	6	15	0.1	145	0.1	6	0.004
$G(q \wedge \neg r \rightarrow (\neg r \cup_8 (p \wedge \neg r)))$	10	109	0.2	5519	4.5	10	0.005
$G(q \wedge \neg r \rightarrow (\neg r \cup_{12} (p \wedge \neg r)))$	14	753	6.3	27338	1414.5	14	0.006
AMBA G1+2+3	12	22	0.1	22	0.1	7	0.008
AMBA G1+2+4	8	61	6.3	78	2.2	8	0.6
AMBA G1+3+4	15	231	55.6	640	97.6	14	0.4
AMBA G1+2+3+5	18	370	191.8	1405	61.8	17	0.05
AMBA G1+2+4+5	12	101	3992.9	253	472.9	12	3.2
AMBA G4+5+6	26	252	117.9	205	26.4	18	0.6
AMBA G5+6+10	31	329	9.8	396	31.4	27	2.6
AMBA G5+6+9e4+10	50	455	17.6	804	42.1	46	5.2
AMBA G5+6+9e8+10	68	739	34.9	1349	86.8	64	7.6
AMBA G5+6+9e16+10	104	1293	74.7	2420	189.7	100	12.5
AMBA G5+6+9e64+10	320	4648	1080.8	9174	2182.5	316	40.9
AMBA G8+9e4+10	48	204	7.0	254	6.1	48	0.3
AMBA G8+9e8+10	84	422	22.5	685	33.7	84	0.5
AMBA G8+9e16+10	156	830	83.7	1736	103.1	156	0.9
AMBA G8+9e64+10	588	3278	2274.2	7859	2271.5	588	3.3

Conclusions

- Soft requirements provide invaluable technique to guide controller synthesis for higher quality.
- Robustness of controller can be specified using QDDC based soft and hard requirements.
- Different shield synthesis criteria can be specified using QDDC based soft and hard requirements.
- LTL[DC] is a simple way of enhancing expressiveness of LTL. All the exising LTL synthesis algorithms extend naturally to LTL[DC]
- Open: Globally optimal synthesis!

Discrete-time Duration Calculus

QDDC logic of finite (non-empty) state seqeunces.

```
req   1  0  1  1  0
ack   0  0  0  0  1
```

We define $\sigma, [b, e] \models D$.

Example `<req> ^ [!ack] ^ <ack>`

- Interval temporal logic
- Quantitative Measurements of Time

Example In any interval of 20 or more cycles where request is continuously high there must be at least 3 *ack* signals.

```
[]( [[req]] && slen >= 20  => scount ack >= 3)
```

QDDC Syntax

Let `P` $\in$ *Prop*(Σ), $c \in \mathbb{N}$, D1,D2 $\in$ *QDDC*. Let $\in$ `{ <=, <, =, >, >=}` Then syntax of QDDC:

```
<P> | [[P]] | slen ~ c | scount P ~ c |
D1^D2 | D* | D1 && D2 | !D |
(exists P. D)
```

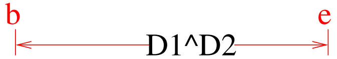
QDDC: Syntax and Semantics

$\sigma, [b, e] \models$ `<P>` **iff** $b = e$ and $\sigma, b \models P$

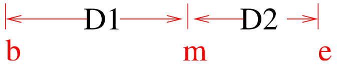
$\sigma, [b, e] \models$ `[[P]]` **iff** for all $t : \; b \leq t \leq e. \;\; \sigma, t \models P$

$\sigma, [b, e] \models$ `[P]` **iff** $b < e$ and for all $t : \; b \leq t < e. \;\; \sigma, t \models P$

$\sigma, [b, e] \models$ `D1^D2` **iff** for some $m : b \leq m \leq e.$
$\sigma, [b, m] \models$ `D1` and $\sigma, [m, e] \models$ `D2`



If for some m



Example: `[P]^[!P]^[[P]]`

A valid formula: `<P>   <=>   <P>^<P>^<P>`

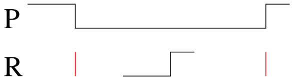
Syntax and Semantics (2)

Derived Operators

- For some subinterval `D`: `<>D` $\overset{\text{def}}{=}$ `true^D^true`
- For all subintervals `D`: `[]D` $\overset{\text{def}}{=}$ `! <> !D`

Validity in execution $\boxed{\sigma \models D \quad \textbf{iff} \quad \sigma, [0, \#\sigma - 1] \models D}$

Example: `[](<down(P)>^[!P]^<up(P)>  =>  !<>([!R]^[R])`



Measurement Formulae

Measurement Terms $\quad$ `slen` | `scount P`

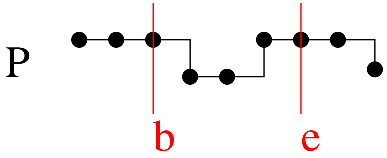
$$eval(\texttt{slen})(\sigma,[b,e]) \stackrel{\text{def}}{=} e-b$$

$$eval(\texttt{scount}\quad\texttt{P})(\sigma,[b,e]) \stackrel{\text{def}}{=} \sum_{i=b}^{e}\left\{\begin{array}{ll} 1 & if\quad \sigma,i\models P \\ 0 & otherwise \end{array}\right\}$$

Measurement Formula *mt op c*

where $op \;\in\; < \;|\; > \;|\; = \;|\; \leq \;|\; \geq.$

Measuring Counts and Durations



$eval(\texttt{slen}) = 4$

$eval(\texttt{scount P}) = 3$

Examples

```
[]( [[req]] && slen >= 20  => scount ack >= 3)
```

Between any two *P* phases there are at least 300 cycles.

```
 [] ( < down(P)>^[!P]^<up(P)>  =>  (slen >= 300) )
```

- Minimum Separation
- Upper bound
- Persistence
- Arrow operators [Ravn94]

Quantification `exists p: D`

$\sigma, [b,e] \models (\texttt{exists p: D})$ **iff** $\sigma', [b,e] \models D$ for some p-variant σ'