# Assessed Coursework: Systems Verification

Ioannis Kassinopoulos

February 27, 2013

## Question 1

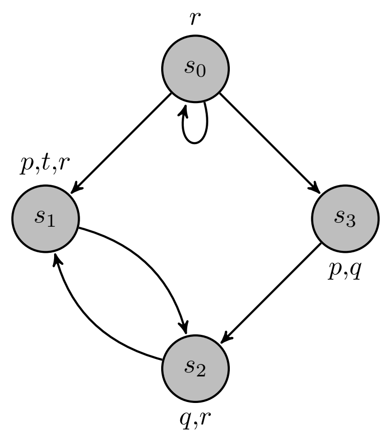

Figure 1: The transition system $\mathcal{M}_1$.

### Algebraic Form

A transition system $\mathcal{M} = (S, \rightarrow, \pi)$ is a set of states $S$ endowed with a transition relation $\rightarrow$ (a binary relation on $S$), such that every $s \in S$ has some $s' \in S$ with $s \rightarrow s'$, and an inverse labeling function $\pi : \mathcal{P} \rightarrow S$.

Our system $\mathcal{M}_1$ (figure: 1) can be described as following:
$\mathcal{P} = \{p, q, r, t\}$

$\mathcal{M}_1 = \{\{s_0, s_1, s_2, s_3\}, \{(s_0, s_0), (s_0, s_1), (s_0, s_3), (s_1, s_2), (s_2, s_1), (s_3, s_2)\}, \pi\}$

$\pi(p) = \{s_1, s_3\}$

$\pi(q) = \{s_2, s_3\}$

$\pi(r) = \{s_0, s_1, s_2\}$

$\pi(t) = \{s_1\}$

## Infinite Tree

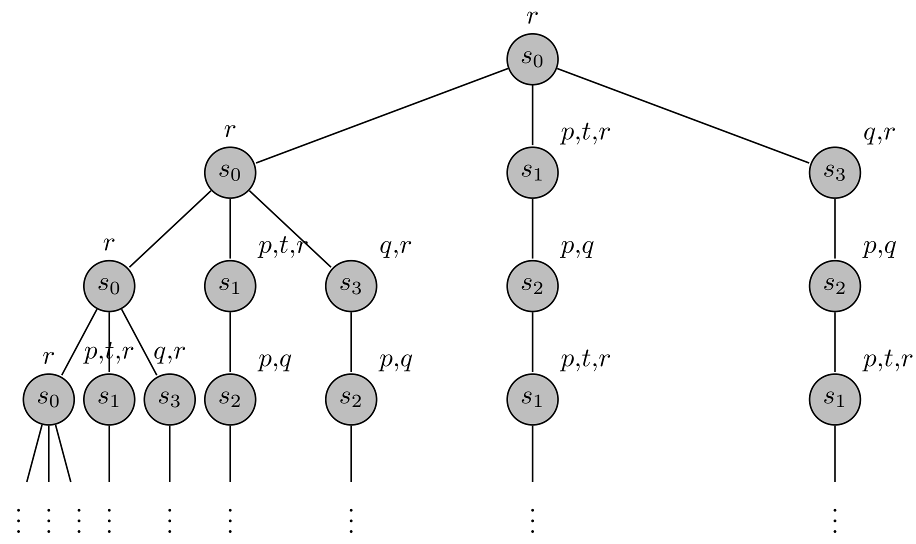

Figure 2: Unwinding the system described by $\mathcal{M}_1$ as an infinite tree of all computation paths beginning in $s_0$ (first layer).

## Satisfiability

**(a) LTL:** $\phi = Ft$

$(\mathcal{M}_1, s_0) \not\models Ft$

since for path $\rho = x_0, x_1, ... = s_0^+$ no state $x_i \notin \pi(t)$ since $s_0 \notin \pi(t)$

$(\mathcal{M}_1, s_2) \models Ft$

since the only path that exist is $\rho = x_0, x_1, ... = (s_2, s_1)^+$ and for $x_i = s_1 \Rightarrow s_1 \in \pi(t)$

**(b) CTL:** $\phi = \neg EGr$

$EGr = \neg AF\neg r \Rightarrow \neg EGr = \neg\neg AF\neg r = AF\neg r$

$(\mathcal{M}_1, s_0) \not\models AF\neg r$ since for the path $\rho = x_0, x_1, ... = s_0^+$ every state $x_i = s_0 \in \pi(r)$

$\Rightarrow (\mathcal{M}_1, s_0) \not\models \neg EGr$

$(\mathcal{M}_1, s_2) \not\models A\neg r$ since for the only path $\rho = x_0, x_1, ... = (s_2, s_1)^+$ every state $x_i = s_1 \in \pi(r)$ or $x_i = s_2 \in \pi(r)$

$\Rightarrow (\mathcal{M}_1, s_2) \not\models \neg EGr$

**(c) CTL:** $\phi = E(tUq)$

$(\mathcal{M}_1, s_0) \not\models E(tUq)$

since for every (and therefore for at least one) path $\rho = x_0, x_1, ... = s_0, ...$ at $s_0$, $s_0 \notin \pi(t) \cup \pi(q)$.

therefore, since at the initial state we have neither t nor q we cannot say that a path exists starting from $s_0$ such that t until q holds.

$(\mathcal{M}_1, s_2) \models E(tUq)$

since for the only path $\rho = x_0, x_1, ... = (s_2, s_1)^+$ we have at $x_0 = s_2$, and $(\mathcal{M}_1, s_2) \models tUq$ since $s_2 \in \pi(q)$. This means that t is always true up to the point that q gets true.

**(d) CTL*:** $\phi = E(FGp)$

For this formula we first need to consider all the possible tuples of transition relation given by the $\rightarrow$ set:

$\{(s_0, s_0), (s_0, s_1), (s_0, s_3), (s_1, s_2), (s_2, s_1), (s_3, s_2)\}$

As we can see from the $\pi(p)$ set, no two concequtive states $x_i, x_{i+1} \in \pi(p)$ which means that for every state $s_i$: $(\mathcal{M}_1, s_i) \not\models Gp$, $(\mathcal{M}_1, s_i) \not\models FGp$ and $(\mathcal{M}_1, s_i) \not\models E(FGp)$

Therefore we can say that:

$(\mathcal{M}_1, s_0) \not\models E(FGp)$

$(\mathcal{M}_1, s_2) \not\models E(FGp)$

since no path exists which eventually gets forever p starting from either $s_0$ or $s_2$

**(e) CTL:** $\phi = EGr$

$EGr = \neg\neg EGr$ therefore from the $\phi = \neg EGr$ solution we can deduce that:

$(\mathcal{M}_1, s_0) \models EGr$ since $(\mathcal{M}_1, s_0) \not\models \neg EGr$ since $true = \neg false$

and

$(\mathcal{M}_1, s_2) \models EGr$ since $(\mathcal{M}_1, s_2) \not\models \neg EGr$ since $true = \neg false$

**(f) LTL:** $\phi = G(r \vee q)$

From the model we can see that the following are true:

$s_0 \in \pi(r) \cup \pi(q)$ since $s_0 \in \pi(r)$ so $(\mathcal{M}_1, s_0) \models r \vee q$

$s_1 \in \pi(r) \cup \pi(q)$ since $s_1 \in \pi(r)$ so $(\mathcal{M}_1, s_1) \models r \vee q$

$s_2 \in \pi(r) \cup \pi(q)$ since $s_2 \in \pi(q)$ so $(\mathcal{M}_1, s_2) \models r \vee q$

$s_3 \in \pi(r) \cup \pi(q)$ since $s_3 \in \pi(q)$ so $(\mathcal{M}_1, s_3) \models r \vee q$

We can therefore say that for every state $s_i$, $(\mathcal{M}_1, s_i) \models G(r \vee q)$ since every path, from every state, will satisfy $r \vee q$ forever (globally).

$(\mathcal{M}_1, s_0) \models G(r \vee q)$

$(\mathcal{M}_1, s_2) \models G(r \vee q)$

**(g) LTL:** $\phi = falseUp$

$(\mathcal{M}_1, s_0) \not\models falseUp$ since $s_0 \notin \{\} \cup \pi(p)$ therefore since the first state of the path does not fulfill p

$\Rightarrow$ false until p cannot be true.

$(\mathcal{M}_1, s_2) \not\models falseUp$ since $s_2 \notin \{\} \cup \pi(p)$ therefore since the first state of the path does not fulfill p

$\Rightarrow$ false until p cannot be true.

**(h) CTL:** $\phi = A(pU(EFq))$

First, let's find which states satisfy $EFq$

$\{s_0, s_1, s_2, s_3\}$ satisfy $EFq$ since all of them are either labelled $q$

or have one of their next states satisfying $q$.

This intuitively says for all states $s_i$, $(\mathcal{M}_1, s_i) \models EFq$.

This means that for all paths starting from every state of our model $pU(EFq)$ is true since we don't care about p being true since $EFq$ is always true at the first (initial) state.

We can therefore deduce from the above that for all states $s_i$, $(\mathcal{M}_1, s_i) \models A(pUEFq)$ and:

$(\mathcal{M}_1, s_0) \models A(pUEFq)$

$(\mathcal{M}_1, s_2) \models A(pUEFq)$

## Question 2

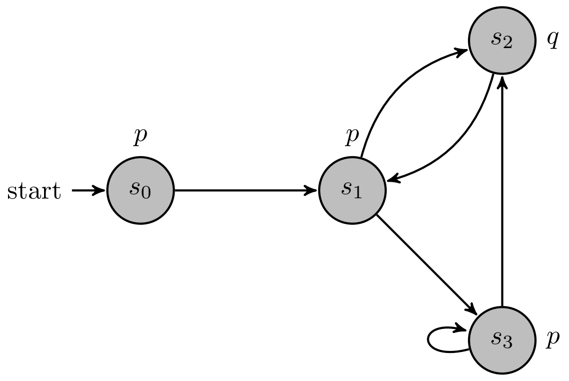

Figure 3: The transition system $\mathcal{M}_2$.

### Calculating

$\phi = p$

$SAT(p) = \{s \in S \mid p \in L(s)\} = \{s_0, s_1, s_3\}$

$\Rightarrow [\![\phi]\!] = \{s_0, s_1, s_3\}$

$\phi = AGEFp$

$SAT(EFp) = SAT(E[trueUp]) = SAT_{eu}(true, p)$

$SAT_{eu}(true, p)$

$W = \{s_0, s_1, s_2, s_3\}$

$Y_0 = \{s_0, s_1, s_3\}$

$Y_1 = \{s_0, s_1, s_2, s_3\}$

$Y_2 = Y_1$

$SAT(AGEFp) = SAT(\neg EF\neg EFp)$

$SAT(EF\neg EFp) = SAT(E[trueU\neg EFp]) = SAT_{eu}(true, \neg EFp)$

$SAT_{eu}(true, \neg EFp)$

$W = \{s_0, s_1, s_2, s_3\}$

$Y_0 = S \backslash \{s_0, s_1, s_2, s_3\} = \{\}$

$Y_1 = Y_0$

$SAT(AGEFp) = SAT(\neg EF\neg EFp) = S \backslash SAT(EF\neg EFp) = \{s_0, s_1, s_2, s_3\}$

$\Rightarrow [\![\phi]\!] = \{s_0, s_1, s_2, s_3\}$

$\phi = AFq$

$SAT(AFq) = SAT_{af}(q)$

$SAT_{af}(q)$

$Y_0 = \{s_2\}$

$Y_1 = Y_0$

$\Rightarrow [\![\phi]\!] = \{s_2\}$

## $\phi = AGp \vee Afq$

$SAT(AGp) = SAT(\neg EF \neg p)$

$SAT(EF\neg p) = SAT(E[trueU\neg p]) = SAT_{eu}(true, \neg p)$

$SAT_{eu}(true, \neg p)$

$W = \{s_0, s_1, s_2, s_3\}$

$Y_0 = S \backslash SAT(p) = \{s_2\}$

$Y_1 = \{s_1, s_2, s_3\}$

$Y_2 = \{s_0, s_1, s_2, s_3\}$

$Y_3 = Y_2$

$SAT(\neg EF \neg p) = S \backslash SAT(EF\neg p) = \{\}$

$SAT(AGp \vee Afq) = SAT(AGp) \cup SAT(AFq) = \{\} \cup \{s_2\} = \{s_2\}$

$\Rightarrow [\![\phi]\!] = \{s_2\}$

## $\phi = E(pU(AFq))$

$SAT(E[pU(AFq)]) = SAT_{eu}(p, AFq)$

$SAT_{eu}(p, AFq)$

$W = SAT(p) = \{s_0, s_1, s_3\}$

$Y_0 = SAT(AFq) = \{s_2\}$

$Y_1 = \{s_1, s_2, s_3\}$

$Y_2 = \{s_0, s_1, s_2, s_3\}$

$Y_3 = Y_2$

$\Rightarrow [\![\phi]\!] = \{s_0, s_1, s_2, s_3\}$

## Question 3

### As one module

```
MODULE main
VAR
t1: {no_coal,has_coal,waiting,tunnel};
t2: {no_coal,has_coal,waiting,tunnel};
t3: {no_coal,has_coal,waiting,tunnel};
cn: {0,1,2,3};
ASSIGN
init(t1) := no_coal;
next(t1) :=
case
t1 = no_coal: has_coal;
t1 = has_coal: waiting;
t1 = waiting & (cn=1): tunnel;
t1 = waiting & !(cn=1): waiting;
t1 = tunnel: no_coal;
esac;

init(t2) := no_coal;
next(t2) :=
case
t2 = no_coal: has_coal;
t2 = has_coal: waiting;
t2 = waiting & (cn=2): tunnel;
t2 = waiting & !(cn=2): waiting;
t2 = tunnel: no_coal;
esac;

init(t3) := no_coal;
next(t3) :=
case
t3 = no_coal: has_coal;
t3 = has_coal: waiting;
t3 = waiting & (cn=3): tunnel;
t3 = waiting & !(cn=3): waiting;
t3 = tunnel: no_coal;
esac;
init(cn) := 0;

next(cn) :=
case
t1=waiting : 1;
t2=waiting : 2;
t3=waiting : 3;
1: 0;
esac;

SPEC
AG(!(t1=tunnel & t2=tunnel & t3=tunnel))

SPEC
AG(!(t1=tunnel) | AF(!(t1=tunnel)))

SPEC
AG(!(t1=has_coal) | EF(!(t1=has_coal)))

SPEC
```

```
AF(t1=tunnel)
```

reachable states: 11 ($2^{3.45943}$) out of 256 ($2^8$)

## CUDD statistics

```
**** CUDD modifiable parameters ****
Hard limit for cache size: 2730666
Cache hit threshold for resizing: 30%
Garbage collection enabled: yes
Limit for fast unique table growth: 1638400
Maximum number of variables sifted per reordering: 1000
Maximum number of variable swaps per reordering: 2000000
Maximum growth while sifting a variable: 1.2
Dynamic reordering of BDDs enabled: no
Default BDD reordering method: 4
Dynamic reordering of ZDDs enabled: no
Default ZDD reordering method: 4
Realignment of ZDDs to BDDs enabled: no
Realignment of BDDs to ZDDs enabled: no
Dead nodes counted in triggering reordering: no
Group checking criterion: 7
Recombination threshold: 0
Symmetry violation threshold: 0
Arc violation threshold: 0
GA population size: 0
Number of crossovers for GA: 0
Next reordering threshold: 4004
**** CUDD non-modifiable parameters ****
Memory in use: 10544032
Peak number of nodes: 1022
Peak number of live nodes: 498
Number of BDD variables: 17
Number of ZDD variables: 0
Number of cache entries: 262144
Number of cache look-ups: 1047
Number of cache hits: 207
Number of cache insertions: 874
Number of cache collisions: 5
Number of cache deletions: 0
Cache used slots = 0.33% (expected 0.33%)
Soft limit for cache size: 18432
Number of buckets in unique table: 4608
Used buckets in unique table: 14.15% (expected 14.15%)
Number of BDD and ADD nodes: 722
Number of ZDD nodes: 0
Number of dead BDD and ADD nodes: 238
Number of dead ZDD nodes: 0
Number of LIVE BDD and ADD nodes: 484
Number of LIVE ZDD nodes: 0
Total number of nodes allocated: 722
Total number of nodes reclaimed: 190
Garbage collections so far: 0
Time for garbage collection: 0.00 sec
Reorderings so far: 0
Time for reordering: 0.00 sec
```

## Modular

```
MODULE train(signal,label)
VAR
state : {no_coal, has_coal, waiting, tunnel};
ASSIGN
  init(state) := no_coal;
  next(state) :=
  case
  (state = no_coal): has_coal;
  (state = has_coal) : waiting;
  (state = waiting) & (signal = label) : tunnel;
  (state = tunnel) : no_coal;
  1: state;
  esac;

MODULE controller(t1, t2, t3)
VAR
  signal: {0,1,2,3};

ASSIGN
 init(signal):= 0;
 next(signal):=
case
t1.state = waiting : 1;
t2.state = waiting : 2;
t3.state = waiting : 3;
1: 0;
esac;

MODULE main
 VAR
   cn: controller(t1,t2,t3);
t1: train(cn.signal,1);
  t2: train(cn.signal,2);
  t3: train(cn.signal,3);


 SPEC
  AG(!(t1.state=tunnel & t2.state=tunnel & t3.state=tunnel));
 SPEC
  AG(t1.state=tunnel -> AF!(t1.state=tunnel));
 SPEC
  AG(t1.state=has_coal -> EF !(t1.state=has_coal));
 SPEC
  AF(t1.state=tunnel);
 SPEC
  AG(EF(t1.state=tunnel));
```

reachable states: 11 ($2^{3.45943}$) out of 256 ($2^8$)

### CUDD statistics

```
 **** CUDD modifiable parameters ****
Hard limit for cache size: 2730666
Cache hit threshold for resizing: 30%
Garbage collection enabled: yes
Limit for fast unique table growth: 1638400
Maximum number of variables sifted per reordering: 1000
Maximum number of variable swaps per reordering: 2000000
Maximum growth while sifting a variable: 1.2
Dynamic reordering of BDDs enabled: no
Default BDD reordering method: 4
Dynamic reordering of ZDDs enabled: no
Default ZDD reordering method: 4
Realignment of ZDDs to BDDs enabled: no
Realignment of BDDs to ZDDs enabled: no
Dead nodes counted in triggering reordering: no
Group checking criterion: 7
Recombination threshold: 0
Symmetry violation threshold: 0
Arc violation threshold: 0
GA population size: 0
Number of crossovers for GA: 0
Next reordering threshold: 4004
**** CUDD non-modifiable parameters ****
Memory in use: 10544032
Peak number of nodes: 1022
Peak number of live nodes: 491
Number of BDD variables: 17
Number of ZDD variables: 0
Number of cache entries: 262144
Number of cache look-ups: 888
Number of cache hits: 243
Number of cache insertions: 680
Number of cache collisions: 3
Number of cache deletions: 0
Cache used slots = 0.26% (expected 0.26%)
Soft limit for cache size: 18432
Number of buckets in unique table: 4608
Used buckets in unique table: 12.15% (expected 12.59%)
Number of BDD and ADD nodes: 635
Number of ZDD nodes: 0
Number of dead BDD and ADD nodes: 168
Number of dead ZDD nodes: 0
Number of LIVE BDD and ADD nodes: 467
Number of LIVE ZDD nodes: 0
Total number of nodes allocated: 635
Total number of nodes reclaimed: 137
Garbage collections so far: 0
Time for garbage collection: 0.00 sec
Reorderings so far: 0
Time for reordering: 0.00 sec
```

## Question 4

Let $\phi = (x_1 \wedge x_2) \vee (y_1 \wedge y_2)$, the following truth table is derived to help us with our calculations

| $x_1$ | $x_2$ | $y_1$ | $y_2$ | $x_1 \wedge x_2$ | $y_1 \wedge y_2$ | $(x_1 \wedge x_2) \vee (y_1 \wedge y_2)$ |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 | 1 | 1 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 |

### Binary Decision Tree

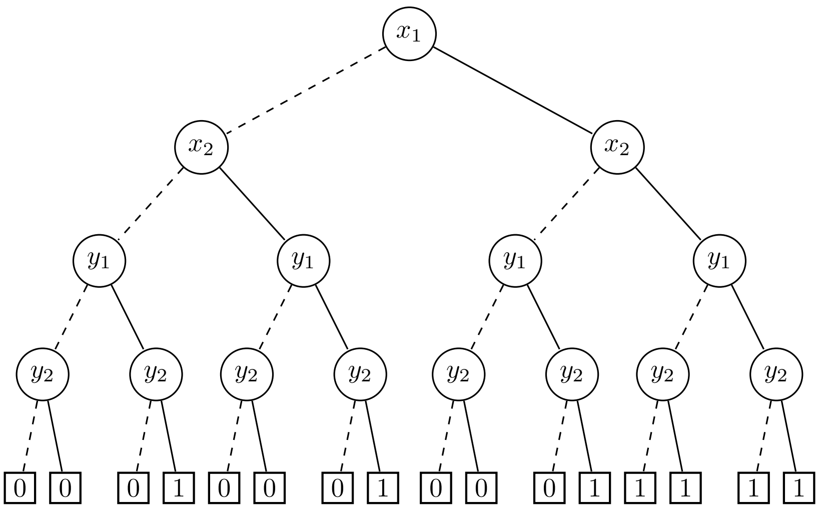

Figure 4: A BDT is easily derived from the truth table. Every non-terminal node is labelled with a variable and every terminal node is labelled with either 0 or 1.

### Reduced Ordered Binary Decision Diagrams

In order to reduce the size of the BDT we can produce a Binary Decision Diagram which is a reduced form of the BDT. Making this diagram ordered over a list of variables, results in getting an Ordered Binary Decision Diagram (OBDD) which is then unique when it is reduced until no more reduction can occur. This reduced form is called canonical form and it can be used to extract equivalences since two different but equivalent Boolean functions always have identically structured Reduced Ordered Binary Decision Diagrams if they have compatible variable orderings.

#### Reduction Algorithm

In order to reduce BDTs we use iteratively the rules C1-C3 until no more reductions can occur.

- **C1:** Removal of duplicate terminals.
- **C2:** Removal of redundant tests.
- **C3:** Removal of duplicate non-terminals.

**ROBDD under the $[x_1, x_2, y_1, y_2]$ ordering.**

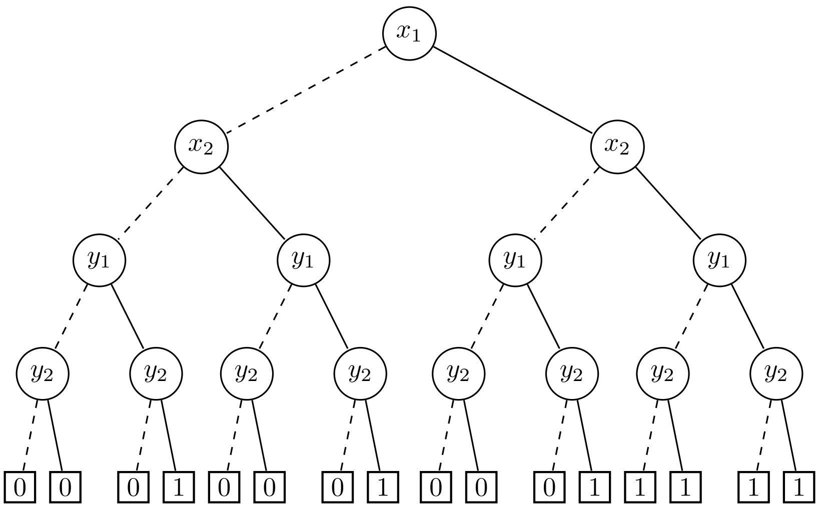

Figure 5: We start with the BDT over our ordering.

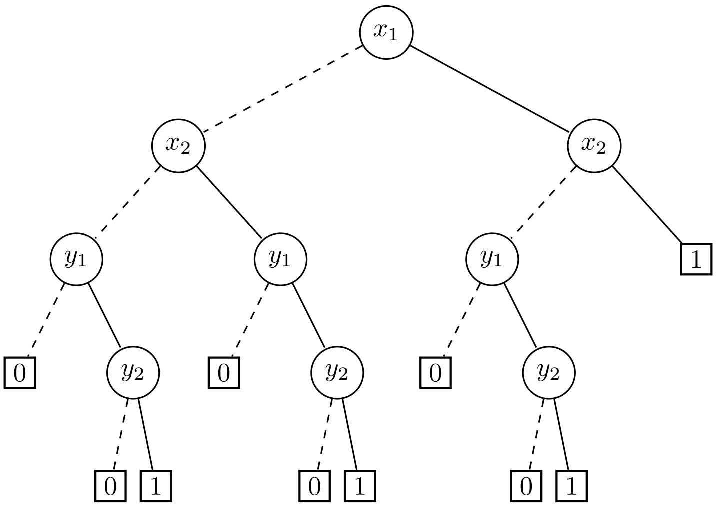

Figure 6: Using C2 we remove the redundant tests and eliminate the nodes leading to them. We derive the above reduced diagram.

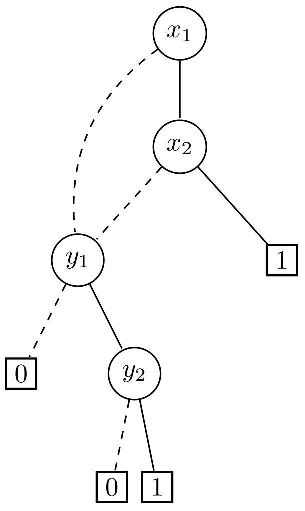

Figure 7: Using C3 we remove the duplicate non-terminals, redirect the incoming edges and derive the above reduced diagram.

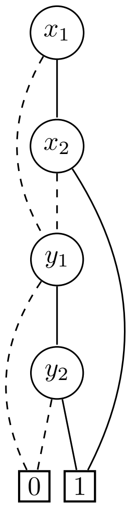

Figure 8: Using C1 we remove all the duplicate terminals. This OBDD cannot be reduced any further so we can now call it the canonical of the previous diagrams.

**ROBDD under the $[x_1, y_1, y_2, x_2]$ ordering.**

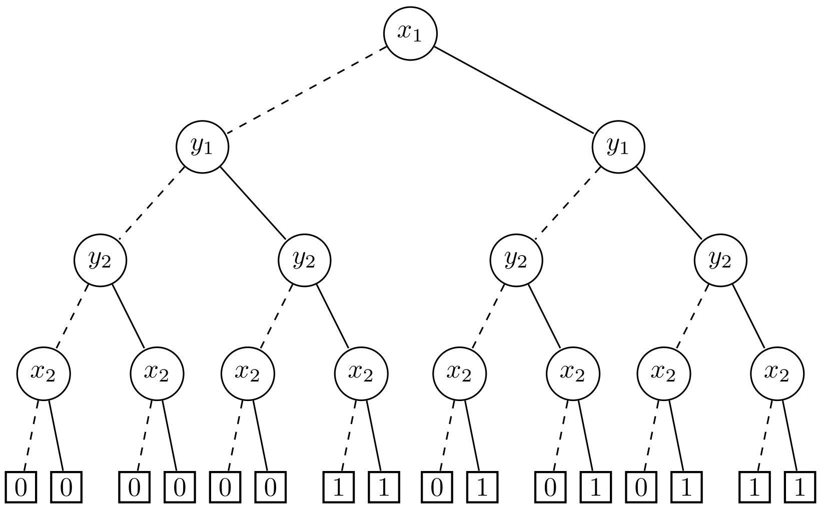

Figure 9: We start with the BDT over our ordering.

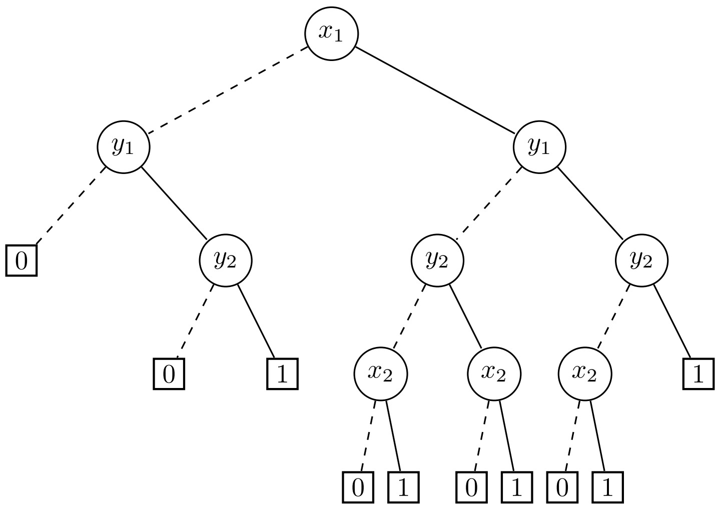

Figure 10: Using C2 we remove the redundant tests and eliminate the nodes leading to them. We derive the above reduced diagram.

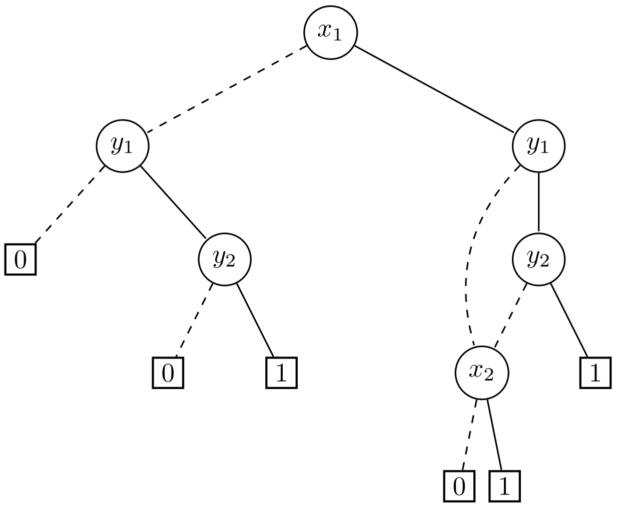

Figure 11: Using C3 we remove the duplicate non-terminals, redirect the incoming edges and derive the above reduced diagram.

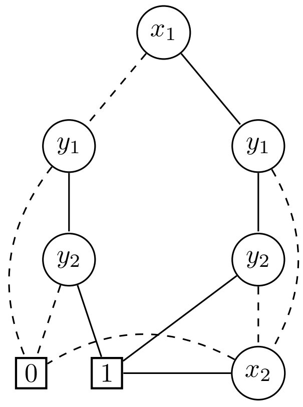

Figure 12: Using C1 we remove all the duplicate terminals. This OBDD cannot be reduced any further so we can now call it the canonical of the previous diagrams.

## How ordering impacts the ROBDD size.

The chosen variable ordering can have a very significant impact on the size of the ROBDD as we can see from the above. The OBDDs' size tend to be very sensitive to the chosen ordering. This however is not always the case and if we consider the OBDDs representing functions such as the parity functions and in principle the functions that are themselves independent of the order of the variables (symmetric boolean functions), it is clear that any change in the ordering leaves the size of the ROBDD unaffected.

In general however, the chosen ordering will have a significant impact to the size. If we consider the boolean function: $(x_1 \vee x_2) \wedge (x_3 \vee x_4) \wedge ... \wedge (x_{2n-1} \vee x_{2n})$, choosing the ordering $(x_1, x_2, ...x_{2n-1}, x_{2n})$ will have produce a ROBDD with 2n+2 nodes while choosing an ordering us $(x_1, x_3, ..., x_{2n-1}, x2, x_4, ...x_{2n})$ will produce an ROBDD of $2^{n+1}$. The level at which the result of the formula depends on a variable along with repetition, precedence and many other factors will affect the importance of the ordering.

## An algorithm for choosing ordering

The problem of choosing the variable ordering to produce an ROBDD of minimum size is similar to the Travelling Salesman Problem of finding the minimum path but propably much harder to be pictured and reason about it. Finding the minimum size is only guaranteed when comparing the resulting ROBDD of every ordering from the $n!$ possible permutations and it is NP-complete. The same goes for finding good or "better" ordering instead of the best. THis hoever can be done using heuristics in a less expensive manner.

One way is by simplifying our formula using equivalences and factorisation to eliminate repetition. We can then examine and find dependancy chains. The more dependent the result of a formula, the higher ordering the variable should have. Another way to be considered is the use of a hill climbing procedure which will redorder the given ordering such as the size decreases. A similar approach can be achieved by shifting the variables one by one until a minimum is reached.

None of these methods can however guarantee a "minimum" size but only a better one.