



$AR(GH)! \dots$

***A Discussion of
“Planning via Model Checking:
A Decision Procedure for AR ”
by Alessandro Cimatti, et al.***

David “Dave” Dagon

dagon@cc.gatech.edu

College o’ Computing
Georgia Institute of Technology

- What I Planned to Discuss
- What I Did Instead
- What The Paper *Might* Have Said

What I Planned to Discuss

- Let's assume *arguendo* that I read the paper...
- Skimming through, it looks like the authors claimed:
 - Use a high level lisp-like language to describe problems reduced to a formal notation (\mathcal{AR})
 - Use FSM to model actions
 - Use model-based reasoning to create plans

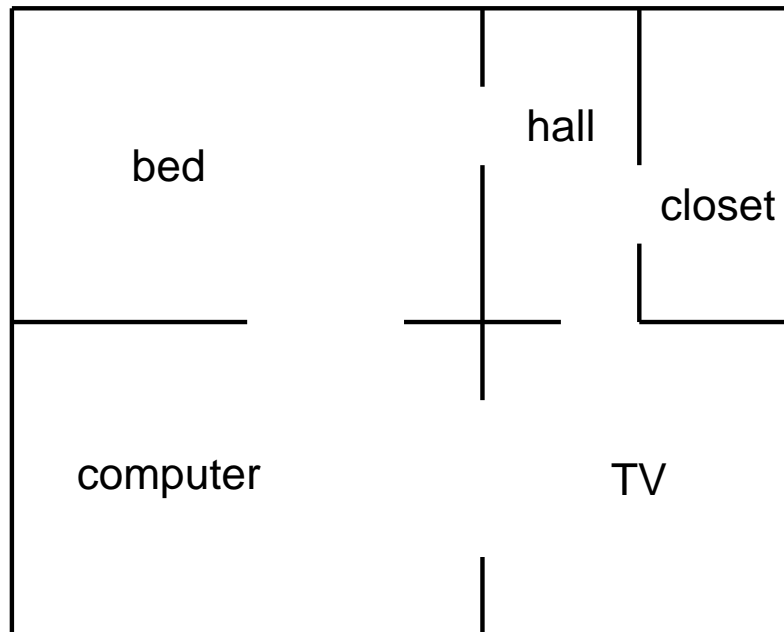
What I Did Instead

- Unfortunately, I did not want to read the paper.
- Instead, I decided to sleep all day and watch TV.
- But like any good student, I *planned* this carefully.

Planning to Avoid Planning

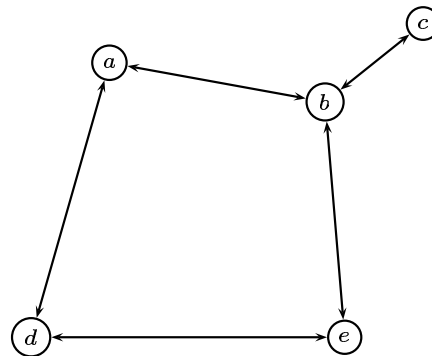
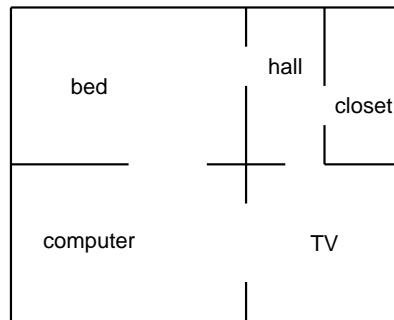
Homework

- If I wake up, I have to find a path to the Television.
- I have to avoid the computer, where work must be done.



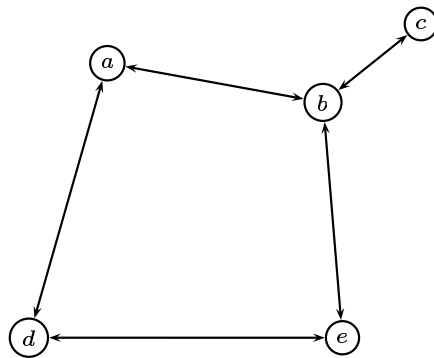
How to Avoid Work

- To properly avoid work, one must plan carefully
- How can I go from *bed* \rightarrow *TV*?
- The map can be represented as graph:



State Representation

- How can we convert the graph node to a BDD?
- With 5 locations, 3 bits are needed



Node	Bits
A	000
B	001
C	010
D	011
E	100

Transition Representation

- How does one model movements between states?
- We can think of the 3-bit state model as \mathbb{B}^3 . We need merely express adjacencies in \mathbb{B}^6

action *to/from* \rightarrow *legal?*

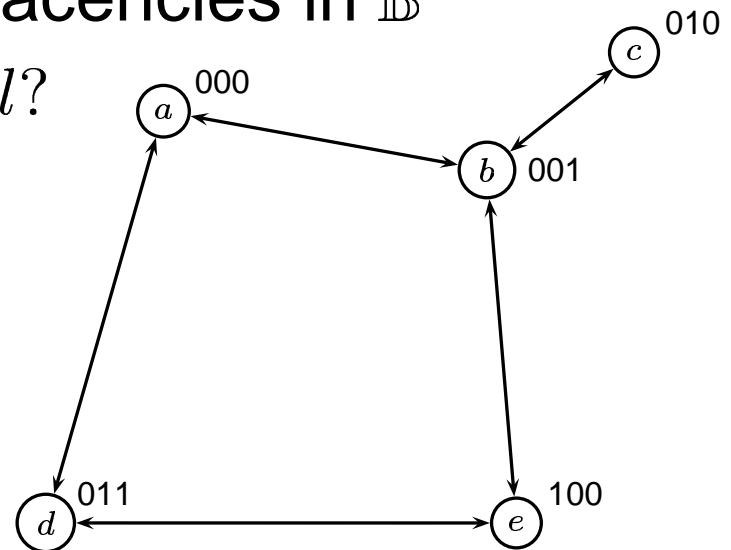
a_1 000000 \rightarrow \perp

a_2 000001 \rightarrow \top

a_3 000010 \rightarrow \top

a_4 000011 \rightarrow \perp

... ...



Efficient Transition Representation

- A complete $N \times N$ table (cartesian product of sets) would be sparse and inefficient for domains with few actions.
- A hashtable of positive values might do instead.

action *to/from* \rightarrow *legal?*

a_1 000000 $\rightarrow \perp$

a_2 000001 $\rightarrow \top$

a_3 000010 $\rightarrow \top$

a_4 000011 $\rightarrow \perp$

... ...

Kripke Structures

- A Kripke structure M over propositions Φ is the four tuple $M = (S, S_0, R, L)$
 - S is a finite set of states
 - $S_0 \subseteq S$ is the set of initial states
 - $R \subseteq S \times S$ is the transition relation, s.t.
 $\forall s \in S, \exists s' \in S$ s.t. $R(s, s')$.
 - $L : S \rightarrow 2^\Phi$ is a function that labels each state with the propositions true for that state
- A path in M is a finite set of sequences $\pi = s_0 s_1 \dots s_n$ where $s_0 = s$ and $R(s_i, s_{i+1})$ holds for $i \geq 0$.

Kripke Structures

- Guess what? Our simple \mathbb{B}^6 representation of the domain is a Kripke structure
- BDDs are useful representations of Kripke structures.
- It remains to write a program to convert the Kripke structure into a BDD representation.

- Instead of converting a domain by hand, it's far easier (and more common) to use a higher language

```
1 (define (domain dave_navigation)
2   (:types room)
3   (:constants bed closet hallway computer
4               TV_room - room)
5   (:functions (dave_position) - room)
```

Higher Languages

- Instead of converting a domain by hand, it's far easier (and more common) to use a higher language

```
1      (:action move_dave_up
2          :precondition (or (= (dave_position) computer)
3                          (= (dave_position) TV_room)
4                          (= (dave_position) hallway))
5          :effect (and
6              (when (= (dave_position) computer)
7                  (assign (dave_position) bed))
8              (when (= (dave_position) TV_room)
9                  (assign (dave_position) hallway))
10             (when (= (dave_position) hallway )
11                 (assign (dave_position) closet))))
```

Higher Languages

- Fortunately, the authors of the paper provided me with a parser and library to convert high level domain descriptions into a BDD!
- (Further code review; demonstration)

- It turns out the paper was very worth while, and helped me plan goofing off!
- What other wonders did the paper hold?

- MBP uses boolean formulas to represent symbolic states
- From the paper:
We will consider the assignments of values to variables satisfying:
 - the formula $State_F$, defined by

$$\bigwedge_{(always\ C) \in A} C;$$

- (cont'd ...)

- From the paper (cont'd):
 - and the formula Res_F , defined by:

$$Res_F^0 \wedge$$

$$\neg \exists v_1 \dots v_n (Res_F^0 [F'_1 / v_1, \dots, v_n])$$

$$\wedge_{1 \leq i \leq m} (v_i = F_i \vee v_i = F'_i)$$

$$\vee_{1 \leq i \leq m} (v \neq F'_i)$$

$$\wedge (A \text{ possibly changes } F_i \text{ if } P) \in \mathcal{A} ((Act = A \wedge P) \supset v_i = F'_i))$$

- (Oh, it's not done....)

- From the paper (yep, still cont'd):
Where Res_F^0 is an abbreviation for

$$State_F[F_1/F'_1, \dots, F_n/F'_n] \wedge \\ \wedge (A \text{ causes } C \text{ if } P) \in \mathcal{A}((Act = A \wedge P) \supset \\ C[F_1/F'_1, \dots, F_n/F'_n]),$$

and $\alpha[a/b]$.

- Sadly, this is 90% of the author's explanation.
- cf. Winston Churchill: "This report, by its very length, defends itself against the risk of being read."

Decision Procedure

- MBP works on sets of states
- States resulting from possible actions are stored in a sequence
- The sequence is then check to see if it satisfies the goal.

- function planner(G):

```
1   Acc[0] := New[0] := Init;
2   i := 0;
3   while (New[i] != 0)
4       Reached_goals := (New[i] intersect G);
5       if (Reached_Goals != 0)
6           then return choose-plan(i);
7       i := i + 1;
8       Acc[i] := Acc[i-1] union
9           { s' : Res(s, A, s'),
10              s in New[i-1], A in Actions };
11       New[i] := Acc[i] \ Acc[i-1]
12   return false
```

- function choose-plan(N):

```
1   V[N] := choose-element(Reached_Goals);
2   for (j := N - 1, ..., 0) do
3       Back[j] := { (s,A) : Res( s, A, V[j+1]),
4                   s in New[j+1] };
5       (V[j],A[j+1]) := choose-element(Back[j]);
6
7   return A[1,...,N];
```

What the heck?!?!

- In short, this means that:
 - planner terminates
 - planner returns a solution with a minimum length

- Built on top of SMV (a well-known model checking tool)
- At a glance:
 - MBP is a total order planner; GRAPHPLAN and UCPOP are partial order
 - MBP always returns optimal plan, if it exists.
 - MBP will suffer from large n

Comparisons

- Comparisons

test	UCPOP	GRAPHPLAN	MBP
tsp.a	52	0.67	0.76
tsp.b	–	2023	62
tsp.c	–	–	49
fixit	–	0.26	3.6
logistic.a	–	2.5	80
logistic.b	–	12.4	1200

Comparison Analysis

- GRAPHPLAN is designed to solve parallel problems.
(MBP needs to work on this area.)
- MBP performs well when the actions are many; GRAPHPLAN starts to do worse in this situation.
(Recall that for BDDs, the mapping of operations is polynomial!)
- MBP will potentially suffer from extremely large state problems.
(Unless heuristics are provided for proper ordering of variables.)