

CS-E4800 Artificial Intelligence

Jussi Rintanen

Department of Computer Science
Aalto University

March 1, 2017

Second Assignment (out on February 3)

- Assignments emailed as PDF files on February 3
- Solution document as a PDF or text file, containing
 - your **name** and **student number**
 - your solution specification, including comments/documentation
 - **MathSAT output with -models option** with *false* lines removed
 - All filenames start with **your student number!**
- Return to the course **assignments web page** by deadline Thursday February 16 at 23:59
- **Feel free to ask questions and help, and try to do it well before the deadline.**

Second Assignment (out on February 3)

- Model your problem in the propositional logic (expressed in a high-level typed syntax with first-order features (Forall, Forsome, Exactlyone))
- Translate into the SMT syntax (translator available from course web page):
`./grunder solution.fma`
- **There is no easy way to run the translator in Windows. Please get access to (university) Linux computers!!!**
- Test satisfiability with **MathSAT**
<http://mathsat.fbk.eu/download.html>:
`./mathsat -model < solution.smt`



Example of the modeling language for logic used in the 2nd assignment

```
type M = { Eng, Spa, Ukr, Nor, Jap };
type C = { OldGold, Kools, Chesterfields, LuckyStrike, Pa
type P = { Red, Green, Ivory, Yellow, Blue };
type P = { Dog, Snails, Fox, Horse, Zebra };
type D = { Coffee, Tea, Milk, OrangeJuice, Water };
type H = { 1,2,3,4,5 }; // House numbers
type H4 = { 1,2,3,4 };
```



Zebra Puzzle

Background assumptions about uniqueness

```
forall m : M exactlyone c : C Smokes(m,c) end end;
forall c : C exactlyone m : M Smokes(m,c) end end;

forall m : M exactlyone a : A Pet(m,a) end end;
forall a : A exactlyone m : M Pet(m,a) end end;

forall h : H exactlyone p : P Color(h,p) end end;
forall p : P exactlyone h : H Color(h,p) end end;

forall m : M exactlyone d : D Drinks(m,d) end end;
forall d : D exactlyone m : M Drinks(m,d) end end;

forall m : M exactlyone h : H LivesIn(m,h) end end;
forall h : H exactlyone m : M LivesIn(m,h) end end;
```

Zebra Puzzle

```
// The Ukrainian drinks tea.

Drinks(Ukr,Tea);

// The green house is immediately to the right of the ivory
forall i : H4 (Color(i,Ivory) & Color(i+1,Green)) end;

// The Old Gold smoker owns snails.

forsome i : M (Smokes(i,OldGold) & Pet(i,Snails)) end;
```

Zebra Puzzle

```
// The Englishman lives in the red house.

forsome i : H LivesIn(Eng,i) & Color(i,Red) end;

// The Spaniard owns the dog.

Pet(Spa,Dog);

// Coffee is drunk in the green house.

forsome i : M j : H
    (LivesIn(i,j) & Drinks(i,Coffee) & Color(j,Green))
end;
```

Zebra Puzzle

```
// Kools are smoked in the yellow house.

forsome i : M j : H ( LivesIn(i,j) & Color(j,Yellow) & Smokes(i,Kools)) end;

// Milk is drunk in the middle house.

forsome i : M ( LivesIn(i,3) & Drinks(i,Milk)) end;

// The Norwegian lives in the first house.

LivesIn(Nor,1);
```



Zebra Puzzle

```
// The man who smokes Chesterfields lives in the house next to the house where t
forsome i : M j : M k : H l : H
    (Smokes(i,Chesterfields) &
     Pet(j,Fox) &
     LivesIn(i,k) &
     LivesIn(j,l) &
     (k = l+1 | l = k+1))
end;
```

Zebra Puzzle

```
// Kools are smoked in the house next to the house where t
forsome i : M j : M k : H l : H
    (Smokes(i,Kools) &
     Pet(j,Horse) &
     LivesIn(i,k) &
     LivesIn(j,l) &
     (k = l+1 | l = k+1))
end;
```



Zebra Puzzle

```
// The Lucky Strike smoker drinks orange juice.
forsome i : M (Smokes(i,LuckyStrike)
               & Drinks(i,OrangeJuice));

// The Japanese smokes Parliaments.
Smokes(Jap,Parliaments);

// The Norwegian lives next to the blue house.
forsome i : H j : H
    (LivesIn(Nor,i) & Color(j,Blue)
     & (i = j+1 | j = i+1))
end;
```

Nesting of ExactlyOne

```
type A = {a1,a2}; type B = {b1,b2};
exactlyone a:A b:B R(a,b) end;
exactlyone a:A exactlyone b:B R(a,b) end end;
```

The two exactly-one constraints above **not equivalent**.
The first means that exactly one of the following holds.

$R(a1,b1) \vee R(a1,b2) \vee R(a2,b1) \vee R(a2,b2)$

The second that exactly one of following holds:

```
exactlyone b:B R(a1,b) end;
exactlyone b:B R(a2,b) end;
```

This allows $R(a1,b1)$, $R(a2,b1)$, $R(a2,b2)$ to be true.

There is exactly one married couple vs. There is exactly one woman, who is married to exactly one man (and others are either single or polygamists).



Grounding of the Specification

- Standard grounding process (similar to PDDL)
 - forall $x : \{a, b, c\} \phi(x) = \phi(a) \wedge \phi(b) \wedge \phi(c)$
 - forsome $x : \{a, b, c\} \phi(x) = \phi(a) \vee \phi(b) \vee \phi(c)$
 - atmostone $x : \{a, b, c\} \phi(x) = \neg(\phi(a) \wedge \phi(b)) \wedge \neg(\phi(a) \wedge \phi(c)) \wedge \neg(\phi(b) \wedge \phi(c))$
 - exactlyone $x : \exists \phi(x) = \text{forsome } x : \exists \phi(x) \wedge \text{atmostone } x : \exists \phi(x)$
- Integer equalities simplified to TRUE and FALSE, eliminated.

$$TRUE \wedge \phi \equiv \phi \quad (1)$$

$$FALSE \wedge \phi \equiv FALSE \quad (2)$$

$$TRUE \vee \phi \equiv TRUE \quad (3)$$

$$FALSE \vee \phi \equiv \phi \quad (4)$$



Solution with the MathSAT solver

```
mathsat-5.3.6-linux-x86_64/bin/mathsat -stats < zebra.smt
sat
;; statistics
(
:sat-checks 1
:sat-restarts 1
:sat-decisions 7
:sat-random-decisions 0
:sat-propagations 286
:sat-theory-propagations 0
:sat-watched-clauses-visited 356
:sat-binary-watched-clauses-visited 253
:sat-conflicts 0
:sat-theory-conflicts 0
...
:time-seconds 0.021
:memory-mb 9.777
)
```



Solution with the MathSAT solver

1. `./grunder zebra.fma > zebra.smt` produces a 691 line, 68 kB file
2. `./mathsat -stats < zebra.smt` finds a solution in 21 milliseconds, with a couple of search steps



Solution with the MathSAT solver

```
(Color_1_Yellow true)
(Color_2_Blue true)
(Color_3_Red true)
(Color_4_Ivory true)
(Color_5_Green true)
(Drinks_Eng_Milk true)
(Drinks_Jap_Coffee true)
(Drinks_Nor_Water true)
(Drinks_Spa_OrangeJuice true)
(Drinks_Ukr_Tea true)
(LivesIn_Eng_3 true)
(LivesIn_Jap_5 true)
(LivesIn_Nor_1 true)
(LivesIn_Spa_4 true)
(LivesIn_Ukr_2 true)
(Pet_Eng_Snails true)
(Pet_Jap_Zebra true)
(Pet_Nor_Fox true)
(Pet_Spa_Dog true)
(Pet_Ukr_Horse true)
(Smokes_Eng_OldGold true)
(Smokes_Jap_Parliaments true)
(Smokes_Nor_Kools true)
(Smokes_Spa_LuckyStrike true)
(Smokes_Ukr_Chesterfields true)
```



Zebra Puzzle in the Predicate Logic

- The puzzle can be formalized in Predicate Logic
- Requires **Domain Closure Assumption** and **Unique Names Assumption** \implies finite domain
- Typed quantification missing: clumsier formalization
- Theorem provers for the Predicate Logic usually have **no advantage** over SAT/SMT solvers with finite domain problems

Zebra Puzzle in the Predicate Logic

DCA, UNA

Domain Closure Assumption

$$\forall x(x = A \vee x = B \vee x = C \vee x = D)$$

Unique Names Assumption

$$\neg(A = B) \neg(A = C) \neg(A = D) \neg(B = C) \neg(B = D) \\ \neg(C = D)$$



Zebra Puzzle in the Predicate Logic

Typed quantification

```
forsome i : M j : H
    (LivesIn(i,j) & Drinks(i,Coffee) & Color(j,Green))
end end;
```

in the Predicate Logic would be

```
 $\exists i \exists j (Man(i) \wedge House(j) \wedge LivesIn(i,j) \wedge Drinks(i, Coffee) \wedge Color(j, Green))$ 
```

State-Space Search Methods: Summary

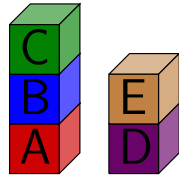
- 1 **explicit state-space search**
 - breadth-first, depth-first, A*, IDA*, greedy best-first
 - **forwards** in the state space
- 2 **backwards** in the space of **sets of states** represented as **formulas**, with actions “reversed” e.g. by weakest preconditions
 - same algorithms and heuristics applicable
- 3 logic-based **symbolic** methods (today’s lecture)
 - satisfiability: iterative deepening search with SAT tests
 - Binary Decision Diagrams: breadth-first with state sets (forwards or backwards)



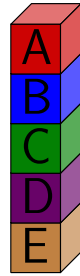
State-space search by logic

Example

initial state



goal state



Problem solved almost without search:

- Formulas for lengths 1 to 4 shown unsatisfiable without any search.
- Search tree for plan length 5 has 3 nodes
- Plans have 5 to 7 operators, optimal plan has 5.



State-space search by logic

Example

	0	1	2	3	4	5	0	1	2	3	4	5	0	1	2	3	4	5	
clear(a)	F	F					F	F	F	T	T	T	F	F	F	T	T	T	T
clear(b)	F		F				F	F	T	T	F	F	F	F	T	T	F	F	T
clear(c)	T	T		F	F		T	T	T	F	F	F	T	T	T	F	F	F	F
clear(d)	F	T	T	F	F	F	F	T	T	F	F	F	F	F	T	T	F	F	F
clear(e)	T	T	F	F	F	F	T	T	F	F	F	F	F	F	F	F	F	F	F
on(a,b)	F	F	F	F	T		F	F	F	F	F	F	F	F	F	F	F	F	F
on(a,c)	F	F	F	F	F		F	F	F	F	F	F	F	F	F	F	F	F	F
on(a,d)	F	F	F	F	F		F	F	F	F	F	F	F	F	F	F	F	F	F
on(a,e)	F	F	F	F	F		F	F	F	F	F	F	F	F	F	F	F	F	F
on(b,a)	T	T		F	F		T	T	F	F			T	T	F	F			
on(b,c)	F	F		T			F	F	F	T			F	F	F	T			
on(b,d)	F	F	F	F	F		F	F	F	F	F		F	F	F	F	F		
on(b,e)	F	F	F	F	F		F	F	F	F	F		F	F	F	F	F		
on(c,a)	F	F	F	F	F		F	F	F	F	F		F	F	F	F	F		
on(c,b)	T		F	F	F		T	T	F	F			T	T	F	F			
on(c,d)	F	F	T	T	T		F	F	T	T	T		F	F	T	T	T		
on(c,e)	F	F	F	F	F		F	F	F	F	F		F	F	F	F	F		
on(d,a)	F	F	F	F	F		F	F	F	F	F		F	F	F	F	F		
on(d,b)	F	F	F	F	F		F	F	F	F	F		F	F	F	F	F		
on(d,c)	F	F	F	F	F		F	F	F	F	F		F	F	F	F	F		
on(d,e)	F	F	T	T	T		F	F	T	T	T		F	F	T	T	T		
on(e,a)	F	F	F	F	F		F	F	F	F	F		F	F	F	F	F		
on(e,b)	F	F	F	F	F		F	F	F	F	F		F	F	F	F	F		
on(e,c)	F	F	F	F	F		F	F	F	F	F		F	F	F	F	F		
on(e,d)	T	F	F	F	F		T	F	F	F	F		T	F	F	F	F		
ontable(a)	T	T	T		F		T	T	T	T			T	T	T	T			
ontable(b)	F	F		F	F		F	F	F	F			F	F	F	F			
ontable(c)	F		F	F	F		F	F	F	F			F	F	F	F			
ontable(d)	T	T	F	F	F		T	T	F	F	F		T	T	F	F	F		
ontable(e)	F	T	T	T	T		F	T	T	T	T		F	T	T	T	T		

- 1 State variable values inferred from **initial values** and **goals**.
- 2 Branch: $\neg\text{clear}(b)$ @1.
- 3 Branch: **clear(a)**@3.
- 4 Plan found:

```

0 1 2 3 4
fromtable(a,b) F F F F T
fromtable(b,c) F F F T F
fromtable(c,d) F F T F F
fromtable(d,e) F T F F F
totable(b,a) F F T F F
totable(c,b) F T F F F
totable(e,d) T F F F F
    
```



Encoding of state-space search as formulas

Atomic propositions for state variables

clear(x)@i $x \in \{a, b, c, d, e\}, i \in \{0, \dots, 5\}$
 on(x,y)@i $x \in \{a, b, c, d, e\}, y \in \{a, b, c, d, e\},$
 $x \neq y, i \in \{0, \dots, 5\}$
 ontable(x)@i $x \in \{a, b, c, d, e\}, i \in \{0, \dots, 5\}$

Atomic propositions for actions

fromtable(x,y)@i $x \in \{a, b, c, d, e\}, y \in \{a, b, c, d, e\},$
 $x \neq y, i \in \{0, \dots, 4\}$
 totable(x,y)@i $x \in \{a, b, c, d, e\}, y \in \{a, b, c, d, e\},$
 $x \neq y, i \in \{0, \dots, 4\}$
 move(x,y,z)@i $x \in \{a, b, c, d, e\}, y \in \{a, b, c, d, e\},$
 $z \in \{a, b, c, d, e\}, x \neq y, x \neq z, y \neq z,$
 $i \in \{0, \dots, 4\}$



Encoding of state-space search as formulas

Effects of actions

move(A,B,C)

- move block *a* from block *b* to block *c*.
- precondition $\text{on}(a, b) \wedge \text{clear}(c)$
- effects $\{\neg\text{on}(a, b), \text{on}(a, c), \text{clear}(b), \neg\text{clear}(c)\}$

Formulas for precondition

for all $i \in \{0, \dots, 4\}$

$$\text{move}(a, b, c)@i \rightarrow \text{on}(a, b)@i \wedge \text{clear}(c)@i$$

Formulas for effects

for all $i \in \{0, \dots, 4\}$

$$\text{move}(a, b, c)@i \rightarrow \neg\text{on}(a, b)@(i + 1)$$



Encoding of state-space search as formulas

Frame axioms

- When does a state variable **not change**?
Often written in different ways:
 - $x@i \wedge \neg x@(i+1) \rightarrow \text{causesForChangeToFalse}@i$
 - $\neg \text{causesForChangeToFalse}@i \wedge x@i \rightarrow x@(i+1)$
 - $\neg x@i \vee x@(i+1) \vee \text{causesForChangeToFalse}@i$
- Called **frame axiom** for historical reasons

Frame axioms

for all $i \in \{0, \dots, 4\}$

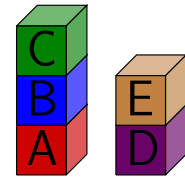
$$\text{on}(a, b)@i \wedge \neg \text{on}(a, b)@(i+1) \rightarrow \text{totable}(a, b)@i \vee \text{move}(a, b, c)@i \vee \text{move}(a, b, d)@i \vee \text{move}(a, b, e)@i$$



Encoding of state-space search as formulas

Initial state

$\text{on}(c, b)@0$	$\text{on}(b, a)@0$	$\text{on}(e, d)@0$
$\text{clear}(c)@0$	$\text{clear}(e)@0$	$\text{ontable}(a)@0$
$\text{ontable}(d)@0$	$\neg \text{on}(a, b)@0$	$\neg \text{on}(a, c)@0$
$\neg \text{on}(a, d)@0$	$\neg \text{on}(a, e)@0$	$\neg \text{on}(b, c)@0$
$\neg \text{on}(b, d)@0$	$\neg \text{on}(b, e)@0$	$\neg \text{on}(c, a)@0$
$\neg \text{on}(c, d)@0$	$\neg \text{on}(c, e)@0$	$\neg \text{on}(d, a)@0$
$\neg \text{on}(d, b)@0$	$\neg \text{on}(d, c)@0$	$\neg \text{on}(d, d)@0$
$\neg \text{on}(e, a)@0$	$\neg \text{on}(e, b)@0$	$\neg \text{on}(e, c)@0$
$\neg \text{ontable}(c)@0$	$\neg \text{ontable}(b)@0$	$\neg \text{ontable}(e)@0$
$\neg \text{clear}(a)@0$	$\neg \text{clear}(b)@0$	$\neg \text{clear}(d)@0$



Goal states

$\text{on}(a, b)@5 \wedge \text{on}(b, c)@5 \wedge \text{on}(c, d)@5 \wedge \text{on}(d, e)@5$



Relations in the Propositional Logic

A binary relation $R \subseteq X \times X$ is a set of pairs $(a, b) \in X \times X$.

Problem: How to represent the pair (0001, 1100)?

Solution: To index the bits use variables

$X_{01} = \{A@0, B@0, C@0, D@0, A@1, B@1, C@1, D@1\}$
instead of $X = \{A, B, C, D\}$.

Pair (0001, 1100) is hence represented as 00011100.

Pair $\begin{matrix} A@0 & B@0 & C@0 & D@0 & A@1 & B@1 & C@1 & D@1 \\ (0 & 0 & 0 & 1, & 1 & 1 & 0 & 0) \end{matrix}$ therefore corresponds to the valuation that assigns 1 to $D@0$, $A@1$ and $B@1$ and 0 to all other variables.



Relations as Formulas

Example

$(A@0 \leftrightarrow A@1) \wedge (B@0 \leftrightarrow B@1) \wedge (C@0 \leftrightarrow C@1) \wedge (D@0 \leftrightarrow D@1)$
represents the identity relation of 4-bit bit-vectors.

Example

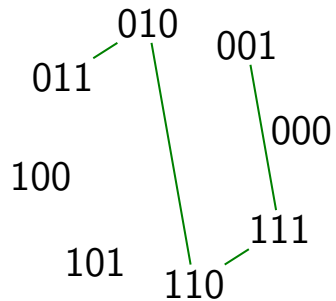
$$\text{inc}_{01} = (\neg C@0 \wedge C@1 \wedge (B@0 \leftrightarrow B@1) \wedge (A@0 \leftrightarrow A@1)) \vee (\neg B@0 \wedge C@0 \wedge B@1 \wedge \neg C@1 \wedge (A@0 \leftrightarrow A@1)) \vee (\neg A@0 \wedge B@0 \wedge C@0 \wedge A@1 \wedge \neg B@1 \wedge \neg C@1) \vee (A@0 \wedge B@0 \wedge C@0 \wedge \neg A@1 \wedge \neg B@1 \wedge \neg C@1)$$

represents the successor relation of 3-bit integers
 $\{(000, 001), (001, 010), (010, 011), (011, 100), (100, 101), (101, 110), (110, 111), (111, 000)\}$



Relations as Formulas

{(001, 111),
(010, 110),
(011, 010),
(111, 110)}



a@0	b@0	c@0	a@1	b@1	c@1	φ
⋮						
0	0	1	1	1	1	1
⋮						
0	1	0	1	1	0	1
⋮						
0	1	1	0	1	0	1
⋮						
1	1	1	1	1	0	1
⋮						

$(\neg a@0 \wedge \neg b@0 \wedge c@0 \wedge a@1 \wedge b@1 \wedge c@1) \vee$
 $(\neg a@0 \wedge b@0 \wedge \neg c@0 \wedge a@1 \wedge b@1 \wedge \neg c@1) \vee$
 $(\neg a@0 \wedge b@0 \wedge c@0 \wedge \neg a@1 \wedge b@1 \wedge \neg c@1) \vee$
 $(a@0 \wedge b@0 \wedge c@0 \wedge a@1 \wedge b@1 \wedge \neg c@1)$



Applications

Example

Can bit-vector 100 be reached from bit-vector 000 by four steps?

This is equivalent to the satisfiability of the following formula.

$$\neg A@0 \wedge \neg B@0 \wedge \neg C@0 \wedge$$

$$inc_{01} \wedge inc_{12} \wedge inc_{23} \wedge inc_{34} \wedge$$

$$A@4 \wedge \neg B@4 \wedge \neg C@4$$



Applications

Propositional formulas can represent the transition relation (graph) of **any** finite discrete system.
Multi-step reachability by chaining the transition relation,.

Example

Make several copies of inc_{01} , and change the subscripts of the variables correspondingly:

$$inc_{01} \wedge inc_{12} \wedge inc_{23} \wedge inc_{34}.$$



Applications

Interesting with **choice** of actions at each step!

Multiply by 2 (left shift, losing the most significant bit):

$$ml2_{01} = (A@1 \leftrightarrow B@0) \wedge (B@1 \leftrightarrow C@0) \wedge \neg C@1$$

Example

Can bit-vector 111 be reached from bit-vector 000 by four steps, by using operations inc and $ml2$?

$$\neg A@0 \wedge \neg B@0 \wedge \neg C@0 \wedge$$

$$(inc_{01} \vee ml2_{01}) \wedge (inc_{12} \vee ml2_{12}) \wedge (inc_{23} \vee ml2_{23}) \wedge (inc_{34} \vee ml2_{34}) \wedge$$

$$A@4 \wedge B@4 \wedge C@4$$



Translation of Actions to Formulas

We consider two systematic translations

- ① Transition relations with one action at each step
- ② Transition relations with multiple actions at each step

Remark: Translations use **equivalences** $x@i \leftrightarrow F(\dots)$ which express new value of state variable in terms of previous state. (Rather than separating this to effect formulas and frame axioms.)

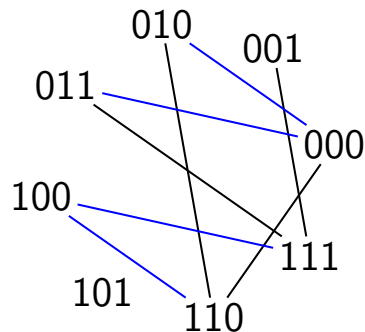
Translation of Actions to Formulas

PRECONDITION: $a = 0$ EFFECTS: $a := 1, b := 1$

PRECONDITION: $b = 1$ EFFECTS: $b := 0, c := 0$

$$(\neg a@0 \wedge a@1 \wedge b@1 \wedge (c@0 \leftrightarrow c@1)) \vee (b@0 \wedge (a@0 \leftrightarrow a@1) \wedge \neg b@1 \wedge \neg c@1)$$

{(000, 110),
(001, 111),
(010, 000),
(010, 110),
(011, 000),
(011, 111),
(110, 100),
(111, 100)}



Translation of Actions to Formulas

Some notation:

- set X of Boolean state variables, subscripted as X_i with time $@i$
- Θ_{ij} with $j = i + 1$ is **transition relation formula** for change between times i and j , with atomic propositions in $X_i \cup X_{i+1}$



Translation of Actions to Formulas

effect	τ_x
$x := 1;$	$x@1$
$x := 0;$	$\neg x@1$
IF ϕ THEN $x := 0;$	$(x@0 \wedge \neg \phi@0) \leftrightarrow x@1$
IF ϕ' THEN $x := 1;$	$(x@0 \vee \phi'@0) \leftrightarrow x@1$
IF ϕ THEN $x := 0;$ IF ϕ' THEN $x := 1;$	$((x@0 \wedge \neg \phi@0) \vee \phi'@0) \leftrightarrow x@1$ (See note below!)
-	$x_0 \leftrightarrow x_1$

$\phi@t = \phi$ with atomic propositions subscripted with t
Note: x will be *true* iff it was *true* and doesn't become *false*, or, it becomes *true*.



Translation of Actions to Formulas

Action as a Formula

An action is represented as a formula

$$\varphi @ 0 \wedge \tau_{x^1} \wedge \tau_{x^2} \wedge \dots \wedge \tau_{x^m}$$

where φ is the precondition of the action and τ_x are as on the previous slide for all state variables x^1, \dots, x^m .

Set of Actions as a Formula

Let $\phi^1, \phi^2, \dots, \phi^n$ be formulas for actions a^1, a^2, \dots, a^n .

Choice between actions is represented by

$$\Theta_{01} = \phi^1 \vee \phi^2 \vee \dots \vee \phi^n.$$



Plans with Multiple Simultaneous Actions

More actions per step \Rightarrow Fewer steps \Rightarrow Smaller formulas \Rightarrow Far lower runtimes

- Dynamics of every state variable separately:
 $x @ (i + 1) \leftrightarrow ((x @ i \wedge \text{no action produces } \neg x) \vee \text{some action produces } x)$
- $a @ i \rightarrow$ the precondition of a
- $\neg(a_1 @ i \wedge a_2 @ i)$ whenever a_1 and a_2 **interfere**, i.e. either
 - a_1 falsifies the precondition of a_2 (or vice versa), or
 - a_1 and a_2 can have conflicting effects.

This guarantees that a set of simultaneous actions can always be executed in any order.



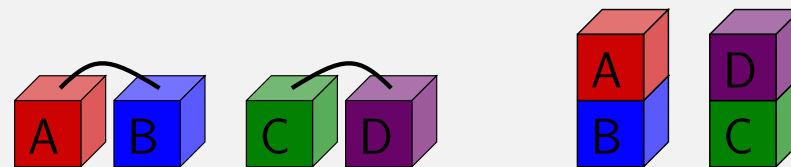
Reachability as Satisfiability

Theorem

A state s' such that $s'(G) = 1$ is reachable from a state s such that $s(I) = 0$ by a sequence of T actions if and only if $I @ 0 \wedge \Theta_{01} \wedge \dots \wedge \Theta_{(T-1)T} \wedge G @ T$ is satisfiable.

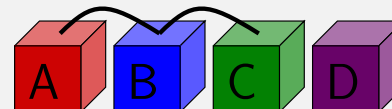
Plans with Multiple Simultaneous Actions

Actions do not interfere



Actions can be taken simultaneously.

Actions interfere



If A moved on B first, B won't be clear and can't be moved.



Example: Robots moving objects

```
type package = { p1,p2,p3,p4,p5 };
type location = { room1, room2};
type robot = { robot1, robot2 };

forall r : robot l : location i : [1..11]
  (at(r,l,i) <-> ((at(r,l,i-1)
    & forall l2 : location not move(r,l,l2,i-1) end)
    | forsome l2 : location move(r,l2,l,i-1) end))
end;
```

Example: Robots moving objects

```
forall p : package l : location i : [1..11]
  (at(p,l,i) <-> ((at(p,l,i-1)
    & forall r : robot not load(r,p,l,i-1) end)
    | forsome r : robot unload(r,p,l,i-1) end))
end;

forall p : package r : robot i : [1..11]
  (holds(r,p,i) <-> ((holds(r,p,i-1)
    & forall l : location not unload(r,p,l,i-1) &
    | forsome l : location load(r,p,l,i-1) end))
end;
```



Example: Robots moving objects

```
forall r : robot i : [1..11]
  (free(r,i) <->
    ((free(r,i-1)
    & forall p : package l : location not load(r,p,l,i-1) end)
    | forsome p : package l : location unload(r,p,l,i-1) end))
end;
```

Example: Robots moving objects

```
// Preconditions of actions

forall r : robot i : [0..10] p : package l : location
  (load(r,p,l,i) -> (at(r,l,i) & at(p,l,i) & free(r,i))) &
  (unload(r,p,l,i) -> (at(r,l,i) & holds(r,p,i)))
end;

forall r : robot i : [0..10] l1 : location l2 : location
  (move(r,l1,l2,i) -> ((l1 != l2) & at(r,l1,i)))
end;
```



Example: Robots moving objects

```
// Exclusive actions:
// At most one move from a location at a time
forall r : robot i : [0..10] l1 : location
  atmostone l2 : location move(r,l1,l2,i)
end end;

// At most one robot can load for each object at a time
forall p : package i : [0..10] l : location
  atmostone r : robot load(r,p,l,i)
end end;
```

Example: Robots moving objects

```
// Initial state

forall r : robot free(r,0) end;
forall r : robot at(r,room1,0) end;
forall r : robot not at(r,room2,0) end;
forall p : package at(p,room1,0) end;
forall p : package not at(p,room2,0) end;
forall p : package not holds(robot1,p,0) end;

// Goal state

forall p : package at(p,room2,11) end;
```

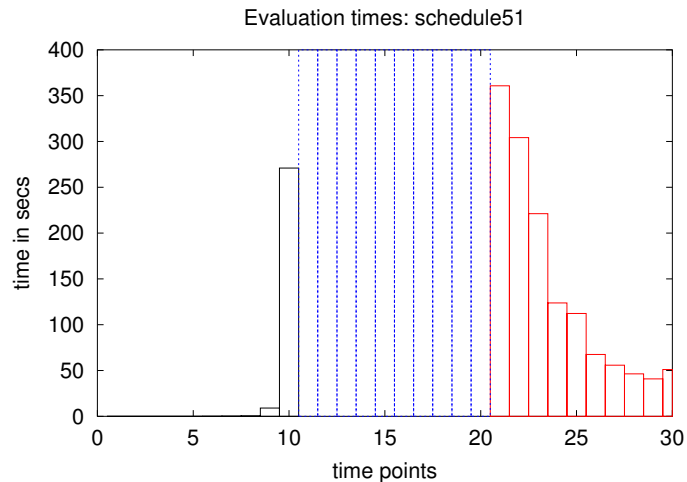
Example: Robots moving objects

```
// Robot cannot move and load or unload at the same time
forall r : robot i : [0..10] l1 : location l2 : location
  move(r,l1,l2,i) -> forall p : package (not load(r,p,l1,i)
                                          & not unload(r,p,l1,i))
                                          end
end;
end;
```

Why Finding Shortest Solutions is Hard?

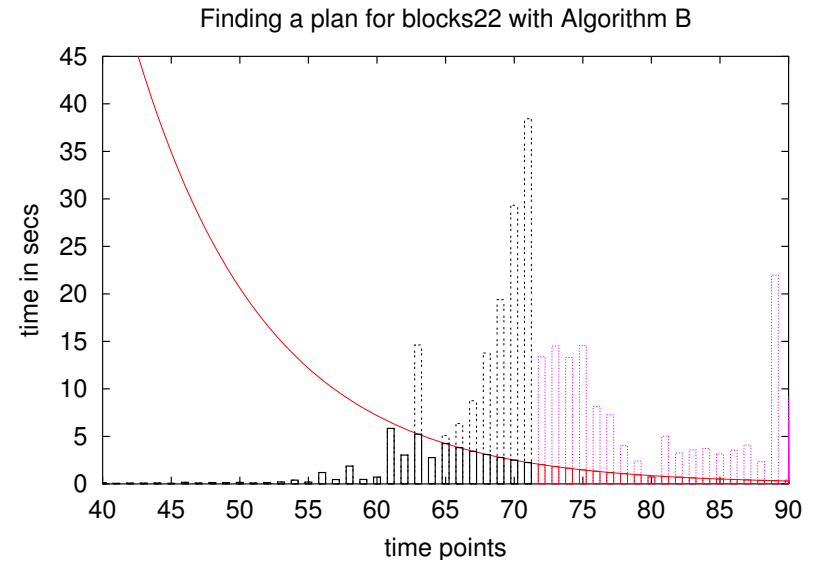
- Finding one (arbitrary) solution can be easy
- Proving that solution is **best** involves testing that no better solutions exist
- Need to **exhaustively** cover a potentially **very large** search space
- Hence, finding optimal solutions is generally much more expensive than finding any solution
- This is analogous to A^* vs. suboptimal algorithms (Best First, WA^*)

SAT Solver Runtime Profiles



(black = unsatisfiable, blue = unknown, red = satisfiable)

Running SAT Solvers in Parallel



Reachability by Satisfiability

This was a **break-through** application for satisfiability testing in AI and other areas of computer science:

- planning, control in AI (Kautz & Selman 1996)
- sequential circuit verification (Bounded Model Checking) (Biere et al. 1999)
- software verification (Clarke et al. 2005)
- many other applications

An important part of this development has been the fast progress in **algorithms** for satisfiability testing.
(ICS-E5050 Advanced Course in Boolean Satisfiability)

