CS-E3220 Declarative Programming

Jussi Rintanen

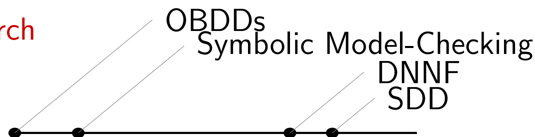
Department of Computer Science
Aalto University

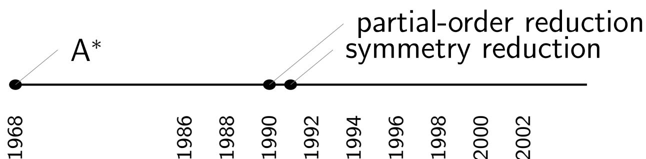
October 16, 2019

State Space Search Methods

SAT-based search

OBDD-based search





explicit state-space search

SAT-Based State Space Search

In OBDD-based symbolic state-space traversal we do

$$
\begin{aligned}
S_1 &= \text{subst}_{X/X'}(\exists X.(S_0 \wedge \rho))) \\
S_2 &= \text{subst}_{X/X'}(\exists X.(\text{subst}_{X/X'}(\exists X.(S_0 \wedge \rho))) \wedge \rho))) \\
S_3 &= \text{subst}_{X/X'}(\exists X.(\text{subst}_{X/X'}(\exists X.(\text{subst}_{X/X'}(\exists X.(S_0 \wedge \rho))) \wedge \rho))) \wedge \rho))) \\
&\vdots
\end{aligned}
$$

Another option is to use a SAT solver to test the satisfiability of

$$S_0 \wedge \text{subst}_{X@1/X'}(\rho) \wedge \text{subst}_{X@1/X, X@2/X'}(\rho) \wedge \text{subst}_{X@2/X, X@3/X'}(\rho) \wedge \cdots \wedge \text{subst}_{X@n/X}(G).$$

Here $X@i$ contains $x@i$ for every state variable $x \in X$.

SAT Reachability: Example

Successor relation of 3-bit integers

$(000, 001), (001, 010), (010, 011), (011, 100), (100, 101), (101, 110), (110, 111), (111, 000)$:

$$
\begin{aligned}
\mathit{inc} \quad = \quad & (\neg c \wedge c' \wedge (b \leftrightarrow b') \wedge (a \leftrightarrow a')) \\
& \vee(\neg b \wedge c \wedge b' \wedge \neg c' \wedge (a \leftrightarrow a')) \\
& \vee(\neg a \wedge b \wedge c \wedge a' \wedge \neg b' \wedge \neg c') \\
& \vee(a \wedge b \wedge c \wedge \neg a' \wedge \neg b' \wedge \neg c')
\end{aligned}
$$

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

$$
\mathit{ml2} = (a' \leftrightarrow b) \wedge (b' \leftrightarrow c) \wedge \neg c'
$$

SAT Reachability: Example

Can bit-vector 101 be reached from bit-vector 000 by four steps, by using actions *inc* and *ml*2?

$$
\begin{aligned}
&\neg a@0 \land \neg b@0 \land \neg c@0 \\
&\land(inc[X@0/X, X@1/X'] \lor ml2[X@0/X, X@1/X']) \\
&\land(inc[X@1/X, X@2/X'] \lor ml2[X@1/X, X@2/X']) \\
&\land(inc[X@2/X, X@3/X'] \lor ml2[X@2/X, X@3/X']) \\
&\land(inc[X@3/X, X@4/X'] \lor ml2[X@3/X, X@4/X']) \\
&\land a@4 \land \neg b@4 \land c@4
\end{aligned}
$$

Satisfying assignment

i	$a@i$	$b@i$	$c@i$	action
0	0	0	0	*inc*
1	0	0	1	*inc* or *ml*2
2	0	1	0	*ml*2
3	1	0	0	*inc*
4	1	0	1	

SAT-Based State Space Search

Procedure:

- Generate the formulas $\Phi_0, \Phi_1, \ldots$ for $0, 1, \ldots$ steps.
- Test their satisfiability one by one. Stop when satisfiable formula found.

Advantages:

- Unlike with OBDDs, memory consumption not as big an issue
- Scalability often far better than with OBDDs
- Extensions of the propositional logic applicable to more general reachability problems

Parallel Actions

- OBDD-based search limited by number of state variables
- SAT-based search is limited also by the length of transition sequences
- How to reduce that length?
  - Pack more than one action/event in one step
  - Possible when no or weak dependencies between actions/events
  - Considerable scalability improvement! (orders of magnitude: $10^2, 10^3, \ldots$)

What Does "Multiple Actions in Parallel" Mean?

Requirements:

- No conflicting effects e.g. $x := 0$ and $x := 1$
- All actions already enabled (actions' preconditions hold)

Interleaving semantics

Events $e_1, \ldots, e_n$ occurring "in parallel" means that they occur in some total ordering $e_{i_1}, e_{i_2}, \ldots, e_{i_n}$ with $\{i_1, \ldots, i_n\} = \{1, \ldots, n\}$.

Choice: We require that

1. actions executable in any order, or
2. actions executable in at least one order?

Parallel Actions

We start from actions (c, e) where e are "programs" as before:

- $x_i := F(x_1, \ldots, x_n)$ for $x_i \in X = \{x_1, \ldots, x_n\}$
- $\text{eITE}(\phi, e_1, e_2)$ (for IF-THEN-ELSE)
- $e_1; e_2$ (sequential composition)
- $e_1 : e_2$ (parallel composition)

What we need from each action a with effect e:

- Condition c that enables the action
- For each state variable x:
  - $\text{cc}_a(x) = \text{cc}_e(x)$: Under what condition is x changed to *true*?
  - $\text{cc}_a(\neg x) = \text{cc}_e(\neg x)$: Under what condition is x changed to *false*?

Condition for Change

$$
\begin{aligned}
\mathsf{cc}_{\epsilon}(x) &= \bot & (1)\\
\mathsf{cc}_{x:=\phi}(x) &= \phi & (2)\\
\mathsf{cc}_{y:=\phi}(x) &= \bot \quad \text{when } x \neq y & (3)\\
\mathsf{cc}_{x:=\phi}(\neg x) &= \neg\phi & (4)\\
\mathsf{cc}_{y:=\phi}(\neg x) &= \bot \quad \text{when } x \neq y & (5)\\
\mathsf{cc}_{\mathsf{eITE}(\phi,e_1,e_2)}(l) &= (\phi \wedge \mathsf{cc}_{e_1}(l)) \vee (\neg\phi \wedge \mathsf{cc}_{e_2}(l)) & (6)\\
\mathsf{cc}_{e_1;e_2}(l) &= (\mathsf{cc}_{e_1}(l) \wedge wp_{e_1}(\neg \mathsf{cc}_{e_2}(\bar{l}))) \vee wp_{e_1}(\mathsf{cc}_{e_2}(l)) & (7)\\
\mathsf{cc}_{e_1:e_2}(l) &= \mathsf{cc}_{e_1}(l) \vee \mathsf{cc}_{e_2}(l) & (8)
\end{aligned}
$$

Parallel Actions: The Transition Relation

Propositional variables used:

x	value of state variable x
x'	value of state variable x at the next time point
a	action a is executed?

If an action is taken, its precondition holds: $a \rightarrow p$.

New value of $x \in X$ determined by:

$$x' \leftrightarrow \left(\bigvee_{a \in A} (a \wedge \mathrm{cc}_a(x)) \right) \vee \left(x \wedge \bigwedge_{a \in A} \neg(a \wedge \mathrm{cc}_a(\neg x)) \right)$$

Change by equivalences $x' \leftrightarrow \cdots$ for all $x \in X$, is deterministic, *after* the choice of actions $a \in A$ has been fixed.

Parallel Actions

Example

Consider actions

- $a_1 = (x, y := 0)$
- $a_2 = (y, x := 0)$

As a formula this is

$$\begin{array}{ll} x' \leftrightarrow (x \wedge \neg a_2) & a_1 \rightarrow x \\ y' \leftrightarrow (y \wedge \neg a_1) & a_2 \rightarrow y \end{array}$$

If both a_1 and a_2 are true, both x' and y' will be false. This does not correspond to any sequential execution of the actions, as the actions disable each other.

Dependencies between Actions

Definition

Action (p_1, e_1) (possibly) disables (p_2, e_2) if some $x \in X$ occurs in e_1 in an assignment $x := \cdots$ and in p_2.

Definition

Action (p_1, e_1) affects (p_2, e_2) if some $x \in X$ occurs in e_1 in an assignment $x := \cdots$ and in ϕ for some eITE$(\phi, \cdots, \cdots)$ in p_2 or in an assignment $x := \cdots$ in p_2.

Note that these definitions are very approximate. E.g. $(\top, (a := 1))$ "disables" $(a, (x := 1))$, although it really doesn't. These definitions could be substantially strenghtened e.g. by considering whether the state variables occur positively or negatively.

Dependencies between Actions

Example

$(\top, (x := 0))$ disables $(x \vee y, (a := 1))$

Example

$(\top, (x := 0))$ affects $(\top, (x := 1))$

Example

$(\top, (x := 0))$ affects $(\top, \mathsf{eITE}(x, (a := 1), (b := 1)))$

Parallel Actions: All Execution Orders

Definition (Interference)

a_1 and a_2 interfere if

1. a_1 can disable or affect a_2, or
2. a_2 can disable or affect a_1

The following guarantees that execution in any order is possible and leads to the same state.

Interference Constraints

If a_1 and a_2 interfere, then include the following formula.

$$\neg a_1 \vee \neg a_2$$

Parallel Actions: At Least One Execution Order

Definition (Directed Interference)

a_1 interferes with a_2 if a_1 can disable or affect a_2.

We will (arbitrarily) fix a total ordering on actions: $a_1, \ldots, a_n$.

Directed Interference Constraints

If a_i interferes with a_j and $j > i$ then include the following formula.

$$\neg a_1 \vee \neg a_2$$

Parallel Actions: Example

Nesting of Russian dolls

a_i does not disable or affect any a_j with $j > 0$:

$$
\begin{aligned}
a_1 &= (\mathsf{out1} \wedge \mathsf{out2} \wedge \mathsf{empty2}, (\mathsf{1in2} := 1; \mathsf{out1} := 0; \mathsf{empty2} := 0)) \\
a_2 &= (\mathsf{out2} \wedge \mathsf{out3} \wedge \mathsf{empty3}, (\mathsf{2in3} := 1; \mathsf{out2} := 0; \mathsf{empty3} := 0)) \\
a_3 &= (\mathsf{out3} \wedge \mathsf{out4} \wedge \mathsf{empty4}, (\mathsf{3in4} := 1; \mathsf{out3} := 0; \mathsf{empty4} := 0))
\end{aligned}
$$

3 steps $\{a_1\}, \{a_2\}, \{a_3\}$ with the first definition (all orderings)

1 step $\{a_1, a_2, a_3\}$ with the second definition (at least one ordering)

Search through Horizon Lengths

The planning problem is reduced to the satisfiability tests for

$$\Phi_0 = I \wedge G$$
$$\Phi_1 = I \wedge T[X@1/X', A@0/A] \wedge G[X@1/X]$$
$$\Phi_2 = I \wedge T[X@1/X', A@0/A] \wedge T[X@1/X, X@2/X', A@1/A] \wedge G[X@2/X]$$
$$\Phi_3 = I \wedge T[X@1/X', A@0/A] \wedge T[X@1/X, X@2/X', A@1/A] \wedge T[X@2/X, X@3/X', A@2/A] \wedge G[X@3/X$$
$$\vdots$$
$$\Phi_n = I \wedge T[X@1/X', A@0/A] \wedge \cdots \wedge T[X@(n-1)/X, X@n/X', A@(n-1)/A] \wedge G[X@n/X]$$

where n is the maximum length.

Q: How to schedule these satisfiability tests?

Search through Horizon Lengths

Sequential Strategy

Test the satisfiability of $\Phi_0, \Phi_1, \Phi_2, \ldots$ one by one.

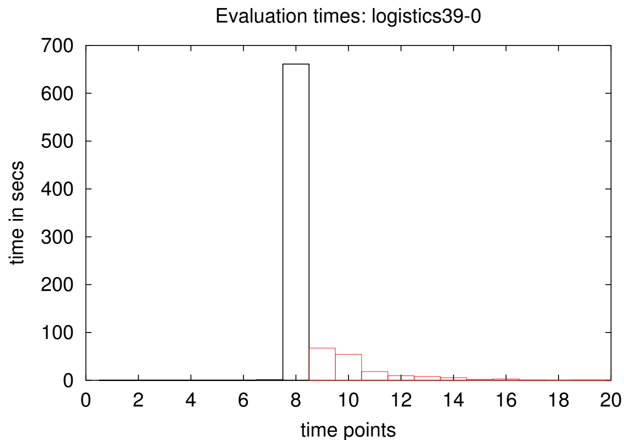
Parallel Strategy A: Fixed Number of Processes

1. Start n processes/threads for SAT solvers with $\Phi_0, \ldots, \Phi_{n-1}$.
2. When a process terminates, start a new one with next $\Phi_n, \Phi_{n+1}, \ldots$.

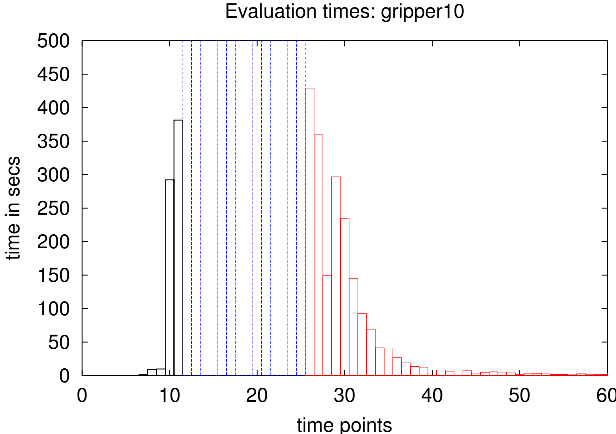
Parallel Strategy B: Geometric

1. Start SAT solving for "all" of $\Phi_0, \Phi_1, \Phi_2, \ldots$.
2. For solving Φ_i, allocate CPU time proportional to γ^i.

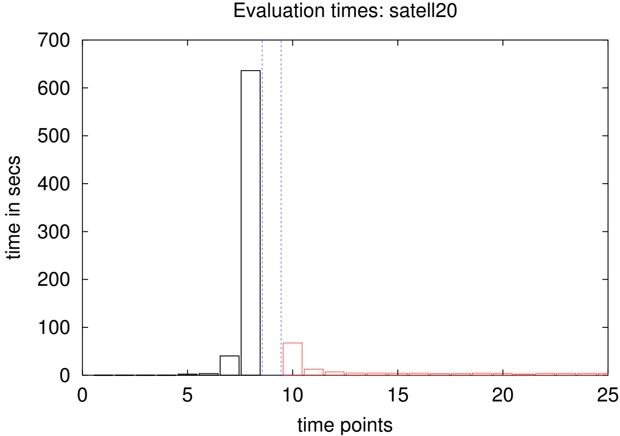
Some Runtime Profiles



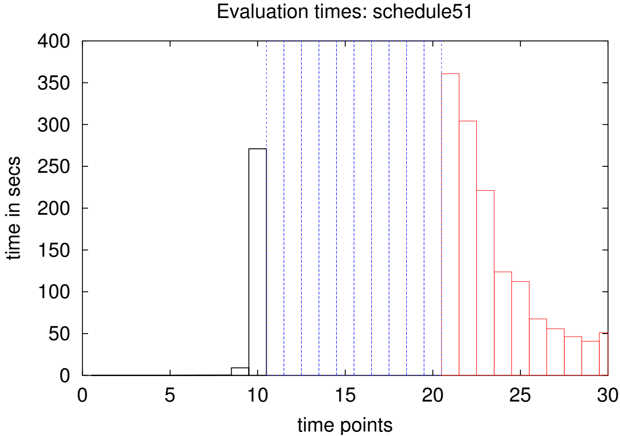
Some Runtime Profiles



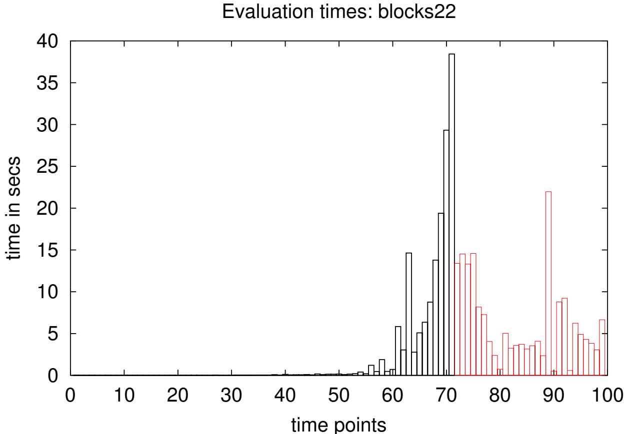
Some Runtime Profiles



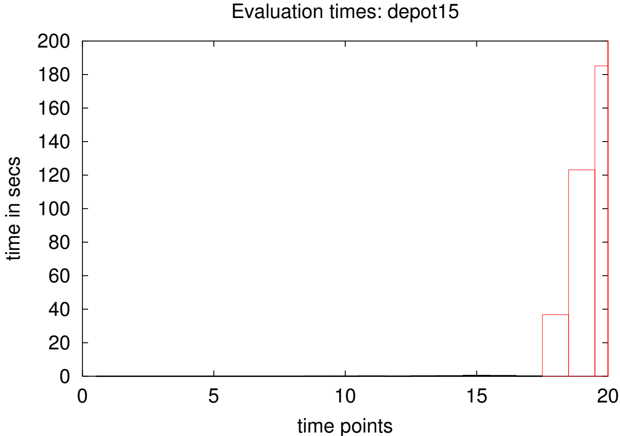
Some Runtime Profiles



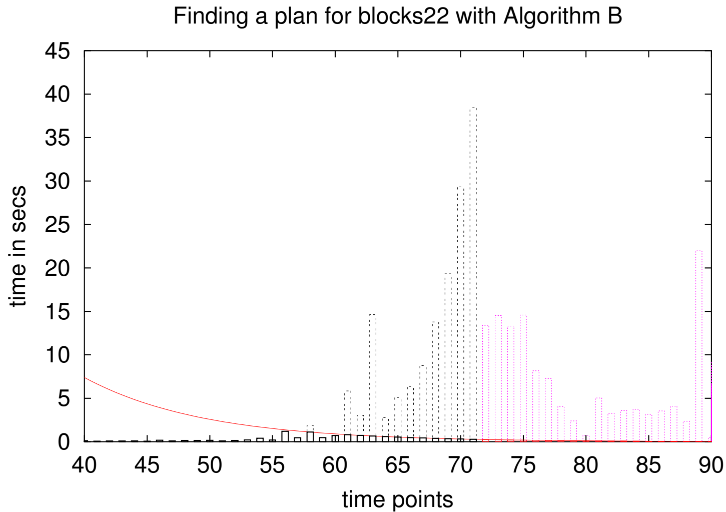
Some Runtime Profiles



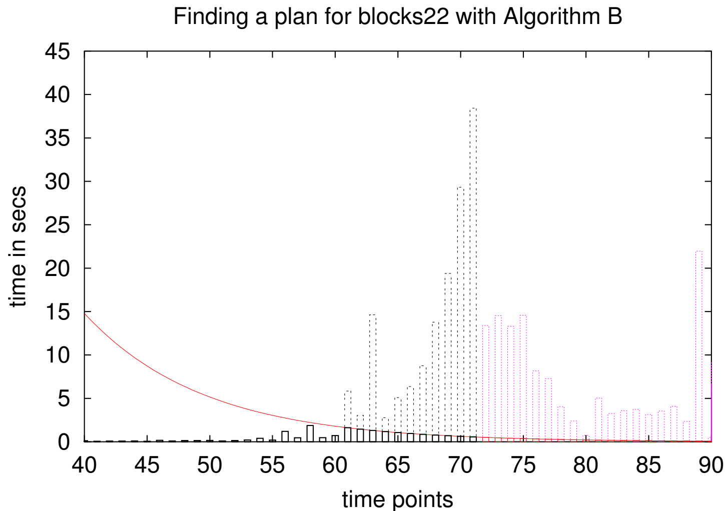
Some Runtime Profiles



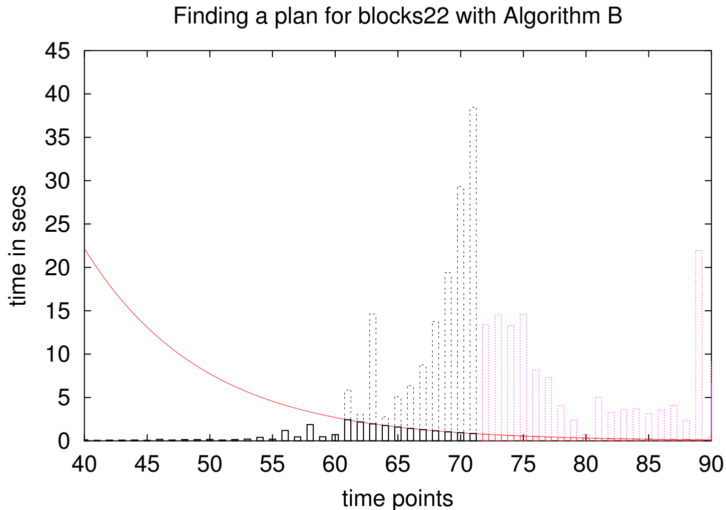
Geometric Allocation of CPU



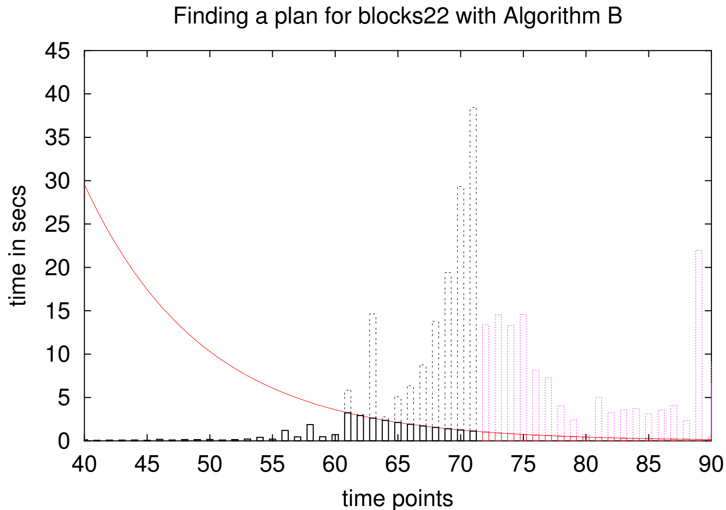
Geometric Allocation of CPU



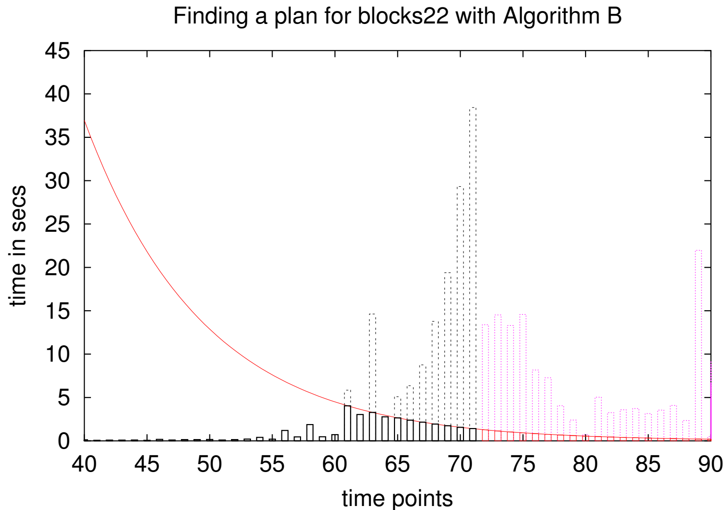
Geometric Allocation of CPU



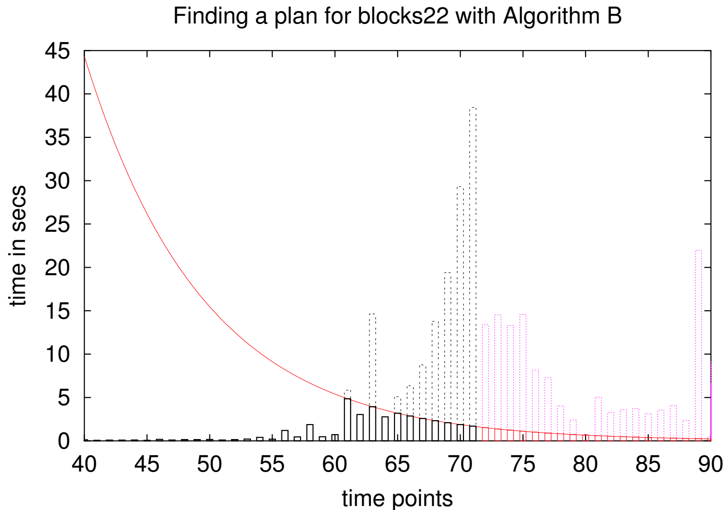
Geometric Allocation of CPU



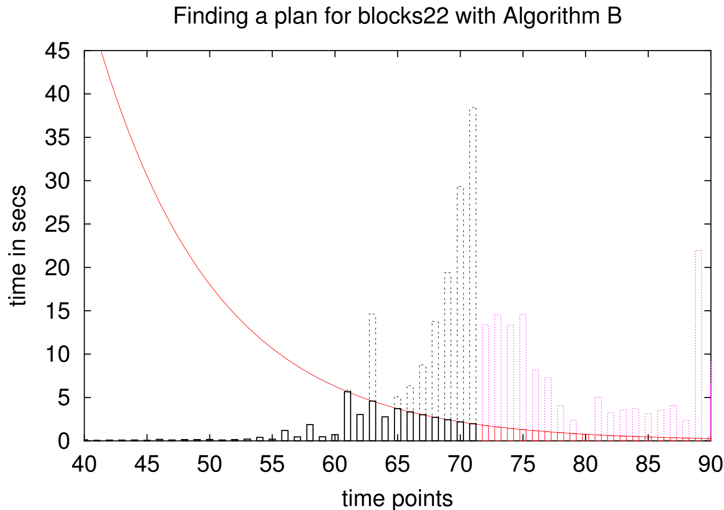
Geometric Allocation of CPU



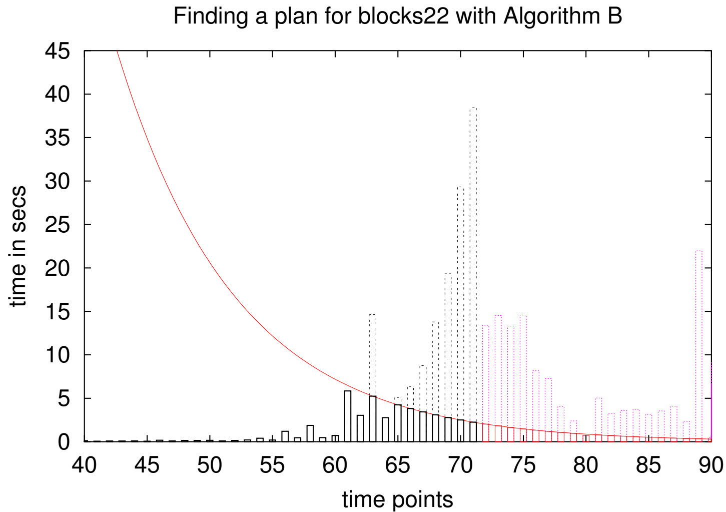
Geometric Allocation of CPU



Geometric Allocation of CPU



Geometric Allocation of CPU



Other Methods for Improving SAT-Based Reachability

- Incremental SAT solving (Eén & Srensson, 2003)
  - Clauses learned for Φ_i re-used when solving Φ_{i+1}
  - Dramatic speed-ups in some cases
- Symmetry Reduction (Clarke, Enders, Filkorn & Jha, 1996)
  - Interchangeability of state variables $\longrightarrow$ symmetry in the state space
  - Symmetry-breaking constraints eliminate symmetry, reduce size of state space
- Counter-example guided abstraction refinement (CEGAR, Clarke et al. 2000)
  - Generate an abstraction by eliminating state variables ($\exists x$)
  - Abstraction has a larger state space
  - If no faulty behavior in abstract space, then no such behavior in original space
  - If abstract faulty behavior found, generate stricter abstraction and test again

Other Methods for Improving SAT-Based Reachability

- Incremental SAT solving (Eén & Srensson, 2003)
  - Clauses learned for Φ_i re-used when solving Φ_{i+1}
  - Dramatic speed-ups in some cases
- Symmetry Reduction (Clarke, Enders, Filkorn & Jha, 1996)
  - Interchangeability of state variables $\longrightarrow$ symmetry in the state space
  - Symmetry-breaking constraints eliminate symmetry, reduce size of state space
- Counter-example guided abstraction refinement (CEGAR, Clarke et al. 2000)
  - Generate an abstraction by eliminating state variables ($\exists x$)
  - Abstraction has a larger state space
  - If no faulty behavior in abstract space, then no such behavior in original space
  - If abstract faulty behavior found, generate stricter abstraction and test again

Other Methods for Improving SAT-Based Reachability

- Incremental SAT solving (Eén & Srensson, 2003)
  - Clauses learned for Φ_i re-used when solving Φ_{i+1}
  - Dramatic speed-ups in some cases
- Symmetry Reduction (Clarke, Enders, Filkorn & Jha, 1996)
  - Interchangeability of state variables $\longrightarrow$ symmetry in the state space
  - Symmetry-breaking constraints eliminate symmetry, reduce size of state space
- Counter-example guided abstraction refinement (CEGAR, Clarke et al. 2000)
  - Generate an abstraction by eliminating state variables ($\exists x$)
  - Abstraction has a larger state space
  - If no faulty behavior in abstract space, then no such behavior in original space
  - If abstract faulty behavior found, generate stricter abstraction and test again

Detecting Unreachability: Reachability Diameter

- Unsatisfiability of $\Phi_i \longrightarrow$ No goal-reaching sequences of length i
- Determine that no goal-reaching sequences of any length exist?
- Find bound b so that Φ_b is unsatisfiable iff no goal-reaching sequences exists
  - Trivial bound $b = 2^n - 1$ when there are n Boolean state variables
  - Hardware systems often have tight bounds that are easy to compute (Baumgartner, Kuehlmann & Abraham 2002)

OBDDs or SAT?

OBDD

- Good for long transition sequences and up to 100 Boolean state variables
- Not very good when the number of state variables is > 100
- Preferred option for proving that no desired transition sequence exists

SAT

- Good for short transition sequences $< 20, 50$, even with lots of state variables
- Not so good with long sequences with > 100 transitions
- Not so good when no desired transition sequence exists

More General Reachability Problems

SAT has turned out to be more flexible more generally:

- Extension: SAT modulo Theories
  - numeric variables and arithmetic constraints
  - for handling numeric state variables and integer/rational time
- Extension: Quantification
  - Quantified Boolean formulas (QBF) with $\exists x$ and $\forall y$
  - Stochastic Satisfiability (SSAT) with stochasti quantification
  - problems outside NP, especially ones PSPACE-complete, Σ_2^p-complete, ...
  - Reachability under a branching program and nondeterministic actions