UNIVERSITY OF TWENTE.
formal methods & tools.

# Problem Solving with Model Checking Techniques

Jaco van de Pol and Michael Weber

June 12, 2011

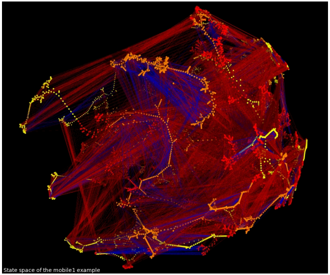
ICAPS Tutorial, Freiburg, Germany

# Table of Contents

# Table of Contents

# Model Checking in a Nutshell

- Specification of system: logical formula $\varphi$
- Implementation of system: Kripke structure
- Question: Does the system meet its specification?




$$\stackrel{?}{\models} \varphi$$

- Applications: hardware, software, wetware
- Method: (Variations of) Graph Reachability

# What is Model Checking?

## Model Checking

- Check if a given model satisfies a given property
- Promise: Automatic answer to combinatorial questions

# What is Model Checking?

## Model Checking

- Check if a given model satisfies a given property
- Promise: Automatic answer to combinatorial questions

## Models: discrete dynamics

- software / hardware / embedded systems
- communicating concurrent components
- biological systems, intra/inter cell level

# What is Model Checking?

## Model Checking

- Check if a given model satisfies a given property
- Promise: Automatic answer to combinatorial questions

## Models: discrete dynamics

- software / hardware / embedded systems
- communicating concurrent components
- biological systems, intra/inter cell level

## Properties: of transition graphs

- invariants, assertions, absence of errors
- absence / presence of event orderings
- complicated fairness restrictions possible

# Applications

**Impressive Applications . . . . . . . . . . . . . Turing Award 2007**

- Numerous case studies have been published:
  - communication and security protocols
  - embedded controllers, e.g. elevators, railways, cars
  - concurrent and distributed algorithms
  - Biology: signaling pathways, gene regulation, differentiation

# Applications

## Impressive Applications .............. Turing Award 2007

- Numerous case studies have been published:
  - communication and security protocols
  - embedded controllers, e.g. elevators, railways, cars
  - concurrent and distributed algorithms
  - Biology: signaling pathways, gene regulation, differentiation
- Leading industries rely on model checking for quality control:
  - Intel/IBM's processors go through extensive model checking (they report that it replaced a considerable amount of testing)
  - Microsoft's Static Device Verifier is part of the WDK ($3^{rd}$ party device drivers are checked for interface compliance)

# Applications

## Impressive Applications . . . . . . . . . . . . . Turing Award 2007

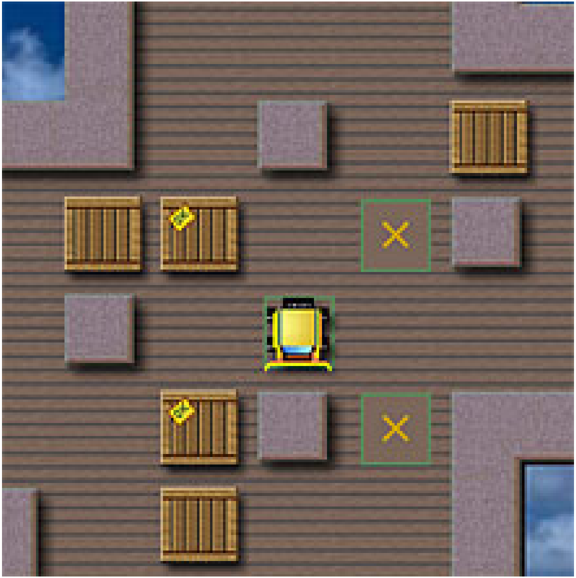
- Numerous case studies have been published:
  - communication and security protocols
  - embedded controllers, e.g. elevators, railways, cars
  - concurrent and distributed algorithms
  - Biology: signaling pathways, gene regulation, differentiation
- Leading industries rely on model checking for quality control:
  - Intel/IBM's processors go through extensive model checking (they report that it replaced a considerable amount of testing)
  - Microsoft's Static Device Verifier is part of the WDK (3rd party device drivers are checked for interface compliance)

## But still, model checking is...

- Not built into CASE tools, despite its "push-button" nature
- Not available to the average (SME-type) software engineer

# Table of Contents

# Sokoban as you know it



# Modelling Sokoban as a Transition System

## States and Transitions

- States:
  - View each location on the board as a variable $x_{i,j}$
  - Possible values: $x_{i,j} \in \{wall, man, block, empty\}$

# Modelling Sokoban as a Transition System

## States and Transitions

- States:
  - View each location on the board as a variable $x_{i,j}$
  - Possible values: $x_{i,j} \in \{wall, man, block, empty\}$
- Transitions – distinguish moves and pushes in four directions
  - Move right:
    if $x_{i,j} = man$ and $x_{i,j+1} = empty$
    then set $x_{i,j} := empty$ and $x_{i,j+1} := man$.
  - Push down:
    if $x_{i,j} = man$ and $x_{i+1,j} = block$ and $x_{i+2,j} = empty$
    then set: $x_{i,j} := empty$, $x_{i+1,j} := man$ and $x_{i+2,j} := block$.

# Modelling Sokoban as a Transition System

## States and Transitions

- States:
  - View each location on the board as a variable $x_{i,j}$
  - Possible values: $x_{i,j} \in \{wall, man, block, empty\}$
- Transitions – distinguish moves and pushes in four directions
  - Move right:
    if $x_{i,j} = man$ and $x_{i,j+1} = empty$
    then set $x_{i,j} := empty$ and $x_{i,j+1} := man$.
  - Push down:
    if $x_{i,j} = man$ and $x_{i+1,j} = block$ and $x_{i+2,j} = empty$
    then set: $x_{i,j} := empty$, $x_{i+1,j} := man$ and $x_{i+2,j} := block$.
- Initial state: an assignment to all the $x_{i,j}$
- Goal state: If for all goal positions (i,j): $x_{i,j} = block$
  then do special action finish

# Next: Solve Sokoban with Brute Force

## Reachability

- This is just the reachability problem
- Property: "Finish action is not reachable"
- Counter-example: trace to a finish-action

# Next: Solve Sokoban with Brute Force

## Reachability

- This is just the reachability problem
- Property: "Finish action is not reachable"
- Counter-example: trace to a finish-action

## Brute Force Exploration

- On-the-fly:
  - start with the initial state
  - expand newly encountered states
  - stop when the goal is reached
- Breadth-first strategy guarantees the shortest solution
- Limitations:
  - Only feasible for about $10^9$ states
  - General: no utilization of specific structure of Sokoban

# Small Problem: Exponential Growth. . .

| screen | states |
|---|---|
| 107 | 10,165 |
| 1001 | 127,509 |
| 387 | 1,235,214 |
| 372 | 10,992,856 |
| 792 | 117,434,655 |
| 747 | 1,307,942,326 |
| 38 | 12,197,960,188 |
| 754 | 308,479,382,084 |
| 2 | 4,748,854,893,784 |

# Small Problem: Exponential Growth. . .

| screen | states |
|---|---|
| 107 | 10,165 |
| 1001 | 127,509 |
| 387 | 1,235,214 |
| 372 | 10,992,856 |
| 792 | 117,434,655 |
| 747 | 1,307,942,326 |
| 38 | 12,197,960,188 |
| 754 | 308,479,382,084 |
| 2 | 4,748,854,893,784 |
| 376 | 1,066,317,080,868,510,000 |
| 45 | 1,195,159,588,147,710,000,000 |
| 71 | 1,238,482,287,687,790,000,000,000 |
| 551 | 1,654,592,279,840,460,000,000,000,000 |
| 48 | 40,215,766,407,984,300,000,000,000,000,000 |
| 22 | 3,531,895,015,833,180,000,000,000,000,000,000 |
| 10 | 12,864,741,234,813,200,000,000,000,000,000,000,000 |
| 558 | 218,612,674,527,952,000,000,000,000,000,000,000,000 |
| 778 | 9,341,745,200,574,070,000,000,000,000,000,000,000,000,000 |

# A Glimpse of Hope: Transition Locality

## Locality

- Transitions may depend on a part of the state vector only
- In Sokoban, every transition depends on 2 or 3 variables; independent of the size of the Sokoban screen
- In general, if you learn one transition and (statically) know the dependency matrix, then you can infer many more transitions

# A Glimpse of Hope: Transition Locality

## Locality

- Transitions may depend on a part of the state vector only
- In Sokoban, every transition depends on 2 or 3 variables; independent of the size of the Sokoban screen
- In general, if you learn one transition and (statically) know the dependency matrix, then you can infer many more transitions

## Example matrix for a Sokoban instance (fragment)

```
+++-----
++------
+----+--
-++-----
-+----+-
--+----+
---+++--
---++---
```

- The dependency matrix is sparse (good)
- The $+$'s are often close together
- Some $+$'s are far apart (2D board)
- Heuristic regrouping can help a lot (cf. BDD variable reordering)

# Exploiting Locality

By locality, successor states are "much alike":

## Locality helps Implementation

- Compression schemes for storing sets of states
- Incremental hashing / storage / compression / etc.
- Communicate diffs only, save bandwidth on clusters

# Exploiting Locality

By locality, successor states are "much alike":

## Locality helps Implementation

- Compression schemes for storing sets of states
- Incremental hashing / storage / compression / etc.
- Communicate diffs only, save bandwidth on clusters

## Locality helps Algorithms (orders of magnitude!)

- Cache intermediate results to save computations
- Store sets of states in Binary Decision Diagrams
- Apply transitions to sets of states symbolically

# Model Checking Techniques I

## Mentioned later

- Multi-core and Grid implementations
- State space reduction (e.g. partial-order reduction)
- Symbolic Model Checking using BDDs
- Adapting the search order of reachability
- Effects of changing the search order:
  - the peak memory of intermediate BDDs is reduced
  - however, a shortest solution is not guaranteed

# Model Checking Techniques II

## Alternatives

- Bounded model checking, based on SAT solving
  - Runs of increasing, fixed length, are encoded into one big satisfiability problem
  - Huge potential, but Sokoban runs are very long

# Model Checking Techniques II

## Alternatives

- Bounded model checking, based on SAT solving
  - Runs of increasing, fixed length, are encoded into one big satisfiability problem
  - Huge potential, but Sokoban runs are very long
- Directed Model Checking
  - Gives priority to transitions towards a “promising” direction
  - Here maybe: number of blocks already in correct position

# Model Checking Techniques II

## Alternatives

- Bounded model checking, based on SAT solving
  - Runs of increasing, fixed length, are encoded into one big satisfiability problem
  - Huge potential, but Sokoban runs are very long
- Directed Model Checking
  - Gives priority to transitions towards a “promising” direction
  - Here maybe: number of blocks already in correct position
- Ad Hoc techniques for (classes of) puzzles
  - Avoid “deadlock” situations
  - Use high-level planning, e.g. recognize rooms

# Model Checking Techniques II

## Alternatives

- Bounded model checking, based on SAT solving
  - Runs of increasing, fixed length, are encoded into one big satisfiability problem
  - Huge potential, but Sokoban runs are very long
- Directed Model Checking
  - Gives priority to transitions towards a "promising" direction
  - Here maybe: number of blocks already in correct position
- Ad Hoc techniques for (classes of) puzzles
  - Avoid "deadlock" situations
  - Use high-level planning, e.g. recognize rooms
- Here is where model checking might learn from planning!

# Beyond Reachability

## More complicated planning problems

- Avoid unwanted situations (still a reachability problem)
  - in LTL logic: **F** *Good* becomes $(\neg \mathit{Bad})$ **Until** *Good*
  - alternatively: restrict the transition relation
- Associate costs with transitions (e.g.: minimal pushes)
  - this requires an adapted search strategy

# Beyond Reachability

## More complicated planning problems

- Avoid unwanted situations (still a reachability problem)
  - in LTL logic: **F** *Good* becomes $(\neg \textit{Bad})$ **Until** *Good*
  - alternatively: restrict the transition relation
- Associate costs with transitions (e.g.: minimal pushes)
  - this requires an adapted search strategy
- Take into account real time
  - UPPAAL: real-time model checker; used for HRT scheduling
- Compute (optimal) cyclic schedules (mean pay-off games)
  - Liveness in logic: **GF** *Good*, reachability infinitely often

# Beyond Reachability

## More complicated planning problems

- Avoid unwanted situations (still a reachability problem)
  - in LTL logic: **F** *Good* becomes $(\neg$*Bad*$)$ **Until** *Good*
  - alternatively: restrict the transition relation
- Associate costs with transitions (e.g.: minimal pushes)
  - this requires an adapted search strategy
- Take into account real time
  - UPPAAL: real-time model checker; used for HRT scheduling
- Compute (optimal) cyclic schedules (mean pay-off games)
  - Liveness in logic: **GF** *Good*, reachability infinitely often
- Plan in the presence of uncertainty (two-player games)
  - and/or graphs with loop restrictions: parity games, $\mu$-calculus
  - can even take into account stochastic environment (MDP)

# Table of Contents

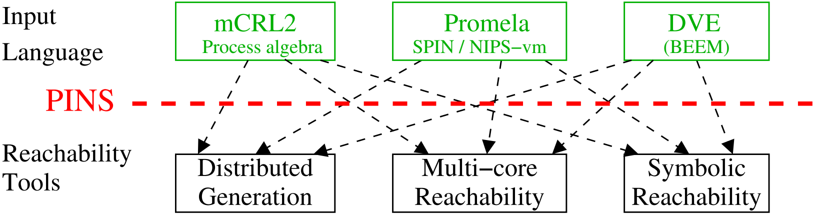
# LTSmin Tool Architecture

# Great Abstractions are Cheap!

- Automatic model translations are not good enough
- Separate languages and tools via a clean interface (API)
- API should be simple: allow many different languages
- API should be rich: expose structure, enable algorithms

# Great Abstractions are Cheap!

- Automatic model translations are not good enough
- Separate languages and tools via a clean interface (API)
- API should be simple: allow many different languages
- API should be rich: expose structure, enable algorithms



# PINS in a Nutshell

**Interface based on a Partitioned Next-State function**

- State: Fixed-size vector of integers $\langle 3,5,5,4,1,3\rangle_{\mathrm{IV}}$
- Partitioned transition relation $\rightarrow = \bigcup_{i}^{K} \rightarrow_i$

# PINS in a Nutshell

**Interface based on a Partitioned Next-State function**

- State: Fixed-size vector of integers $\langle 3, 5, 5, 4, 1, 3 \rangle_{\mathrm{IV}}$
- Partitioned transition relation $\rightarrow = \bigcup_{i}^{K} \rightarrow_i$
- Dependency Matrix $[D]_{N \times K}$:
  - Language module guarantees: if $D_{i,j} = -$, then transition group $\rightarrow_i$ is independent of state slot $j$.
  - Matrix: statically known (currently)
  - Language module may over-approximate dependencies

$$
\begin{array}{c|c}
 & \begin{array}{ccccc} x_1 & \cdots & x_j & \cdots & x_N \end{array} \\
\begin{array}{c} \rightarrow_1 \\ \vdots \\ \rightarrow_K \end{array} &
\begin{bmatrix} + & \cdots & - \\ - & \cdots & + \\ + & \cdots & + \end{bmatrix}
\end{array}
$$

# Example Dependency Matrix

```
int x=7;
process p1() {
do
::{x>0 => x--;y++}
::{x>0 => x--;z++}
od }
```

# Example Dependency Matrix

```
int x=7;
process p1() {
do
::{x>0 => x--;y++}
::{x>0 => x--;z++}
od }
```

```
int y=3;
process p2() {
do
::{y>0 => y--;x++}
::{y>0 => y--;z++}
od }
```

```
int z=9;
process p3() {
do
::{z>0 => z--;x++}
::{z>0 => z--;y++}
od }
```

# Example Dependency Matrix

```
int x=7;
process p1() {
do
::{x>0 => x--;y++}
::{x>0 => x--;z++}
od }
```

```
int y=3;
process p2() {
do
::{y>0 => y--;x++}
::{y>0 => y--;z++}
od }
```

```
int z=9;
process p3() {
do
::{z>0 => z--;x++}
::{z>0 => z--;y++}
od }
```

**Default Matrix**

$$\begin{array}{cc} & \begin{array}{ccc} x & y & z \end{array} \\ \begin{array}{c} p1 \\ p2 \\ p3 \end{array} & \begin{bmatrix} + & + & + \\ + & + & + \\ + & + & + \end{bmatrix} \end{array}$$

# Example Dependency Matrix

```
int x=7;
process p1() {
do
::{x>0 => x--;y++}
::{x>0 => x--;z++}
od }
```

```
int y=3;
process p2() {
do
::{y>0 => y--;x++}
::{y>0 => y--;z++}
od }
```

```
int z=9;
process p3() {
do
::{z>0 => z--;x++}
::{z>0 => z--;y++}
od }
```

**Default Matrix**

$$\begin{array}{c c} & \begin{array}{ccc} x & y & z \end{array} \\ \begin{array}{c} p1 \\ p2 \\ p3 \end{array} & \begin{bmatrix} + & + & + \\ + & + & + \\ + & + & + \end{bmatrix} \end{array}$$

**Better Matrix**

$$\begin{array}{c c} & \begin{array}{ccc} x & y & z \end{array} \\ \begin{array}{c} p1.1 \\ p1.2 \\ p2.1 \\ p2.2 \\ p3.1 \\ p3.2 \end{array} & \begin{bmatrix} + & + & - \\ + & - & + \\ + & + & - \\ - & + & + \\ + & - & + \\ - & + & + \end{bmatrix} \end{array}$$

# Example Dependency Matrix

```
int x=7;
process p1() {
do
::{x>0 => x--;y++}
::{x>0 => x--;z++}
od }
```

```
int y=3;
process p2() {
do
::{y>0 => y--;x++}
::{y>0 => y--;z++}
od }
```

```
int z=9;
process p3() {
do
::{z>0 => z--;x++}
::{z>0 => z--;y++}
od }
```

**Better Matrix**

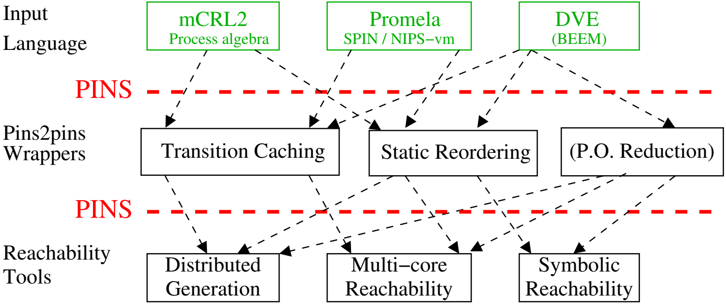
$$\begin{array}{c c} & \begin{array}{ccc} x & y & z \end{array} \\ \begin{array}{c} p1.1 \\ p1.2 \\ p2.1 \\ p2.2 \\ p3.1 \\ p3.2 \end{array} & \begin{bmatrix} + & + & - \\ + & - & + \\ + & + & - \\ - & + & + \\ + & - & + \\ - & + & + \end{bmatrix} \end{array}$$

state $= \langle 7,3,9 \rangle_{\mathrm{IV}}$

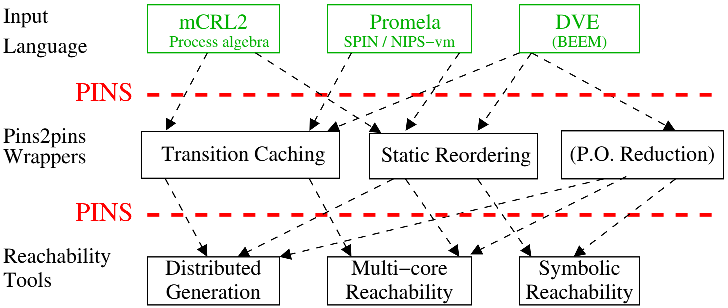
$$\langle 7,3,9 \rangle_{\mathrm{IV}} \xrightarrow{p1.1} \langle 6,4,9 \rangle_{\mathrm{IV}}$$
$$\langle 7,3,* \rangle_{\mathrm{IV}} \xrightarrow{p1.1} \langle 6,4,* \rangle_{\mathrm{IV}}$$

$$\langle 7,3,9 \rangle_{\mathrm{IV}} \xrightarrow{p3.2} \langle 7,4,8 \rangle_{\mathrm{IV}}$$
$$\langle *,3,9 \rangle_{\mathrm{IV}} \xrightarrow{p3.2} \langle *,4,8 \rangle_{\mathrm{IV}}$$

# PINS / LTSmin


Input
Language
mCRL2
Process algebra
Promela
SPIN / NIPS–vm
DVE
(BEEM)
PINS
Pins2pins
Wrappers
Transition Caching
Static Reordering
(P.O. Reduction)
PINS
Reachability
Tools
Distributed
Generation
Multi–core
Reachability
Symbolic
Reachability

# PINS / LTSmin




## Basic functions (algorithms call language modules)

- GETMATRIX : returns the dependency matrix $[D]_{N \times K}$
- INITSTATE(): returns the initial state vector
- NEXTSTATE(I,S): successors of state $s$ in transition group $i$

# Table of Contents

# Local Transition Caching

# Caching Local Transitions (1)

- Recall local transition in specification:

```
p3.2: atomic { z>0 -> z--; y++ }
```

- Dependency matrix row: $\begin{array}{cc} & \begin{array}{ccc} x & y & z \end{array} \\ p3.2 & \begin{bmatrix} 0 & 1 & 1 \end{bmatrix} \end{array}$
- Define projection: $\pi_{p3.2}\langle x, y, z\rangle = \langle y, z\rangle$

# Caching Local Transitions (1)

- Recall local transition in specification:

  `p3.2: atomic { z>0 -> z--; y++ }`

- Dependency matrix row: $\begin{array}{cc} & \mathtt{x} \quad \mathtt{y} \quad \mathtt{z} \\ p3.2 & \begin{bmatrix} 0 & 1 & 1 \end{bmatrix} \end{array}$
- Define projection: $\pi_{p3.2}\langle x, y, z\rangle = \langle y, z\rangle$

- Next, consider two consecutive calls to `p3.2`:

  first call: $\langle x, y, z\rangle$
  successor: $\langle x, y', z'\rangle$
  project and $\langle y, z\rangle \to$
  store in cache: $\langle y', z'\rangle$

# Caching Local Transitions (1)

- Recall local transition in specification:

```
p3.2: atomic { z>0 -> z--; y++ }
```

- Dependency matrix row: $\begin{array}{cc} & \begin{array}{ccc} \mathtt{x} & \mathtt{y} & \mathtt{z} \end{array} \\ p3.2 & \begin{bmatrix} 0 & 1 & 1 \end{bmatrix} \end{array}$
- Define projection: $\pi_{p3.2}\langle x, y, z\rangle = \langle y, z\rangle$
- Next, consider two consecutive calls to `p3.2`:

| | | | |
|---|---|---|---|
| first call: | $\langle x, y, z\rangle$ | second call: | $\langle x'', y, z\rangle$ |
| successor: | $\langle x, y', z'\rangle$ | project: | $\langle y, z\rangle$ |
| project and | $\langle y, z\rangle \rightarrow$ | cache lookup: | $\rightarrow \langle y', z'\rangle$ |
| store in cache: | $\langle y', z'\rangle$ | expand: | $\langle x'', y', z'\rangle$ |

# Caching Local Transitions (1)

- Recall local transition in specification:

```
p3.2: atomic { z>0 -> z--; y++ }
```

- Dependency matrix row: $\begin{array}{cc} & \mathtt{x} \quad \mathtt{y} \quad \mathtt{z} \\ p3.2 & \begin{bmatrix} 0 & 1 & 1 \end{bmatrix} \end{array}$
- Define projection: $\pi_{p3.2}\langle x, y, z\rangle = \langle y, z\rangle$

- Next, consider two consecutive calls to `p3.2`:

| | | | |
|---|---|---|---|
| first call: | $\langle x, y, z\rangle$ | second call: | $\langle x'', y, z\rangle$ |
| successor: | $\langle x, y', z'\rangle$ | project: | $\langle y, z\rangle$ |
| project and | $\langle y, z\rangle \rightarrow$ | cache lookup: | $\rightarrow \langle y', z'\rangle$ |
| store in cache: | $\langle y', z'\rangle$ | expand: | $\langle x'', y', z'\rangle$ |

- Maintain a memoization table $cache[i]$ for each transition group $i$

## Caching Local Transitions (2)

- Caching can save calls to the language module
- Still some work for every concrete state (cache lookup)
- Caching is useful especially:
  - For expressive/inefficient languages
  - When dependency matrices are sparse
- Always uses a bit more memory (tables)

# Static Matrix Reordering

## PINS Optimization

- Regrouping similar transition groups reduces overhead
- Reordering state variables reduces BDD sizes (a.o.)

# Static Matrix Reordering

## PINS Optimization

- Regrouping similar transition groups reduces overhead
- Reordering state variables reduces BDD sizes (a.o.)

```
+--+---------+---
--------------+++
---------------++
-----------+----+
-------------+++-
---------------++
--------------++-
----------+----+-
...(58 rows)....
---+---+---------
--+++------------
--++-------------
--+---+----------
-+---+---+-------
++---------------
-+---+-----------
+---+---+--------
++---------------
+---+------------
```

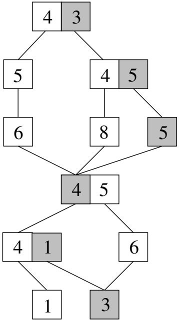
Sokoban before and after regrouping

```
++---------------
+--+---+---------
-++-+------------
--++-------------
--+-++-----------
---+---+--+------
----+++----------
----+--++--------
-----+----++-----
-----+------++---
------+--+--+----
-------++-----+--
-------+-++------
--------+--++----
---------++----+-
----------++-+---
-----------++---+
-------------++--
---------------++
```

# Multi-Valued Decision Diagrams

# Multi-Valued Decision Diagrams

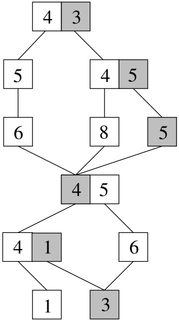
| | | | | | |
|---|---|---|---|---|---|
| 4 | 5 | 6 | 4 | 4 | 1 |
| 3 | 4 | 8 | 4 | 4 | 1 |
| 3 | 5 | 5 | 4 | 4 | 1 |
| 4 | 5 | 6 | 4 | 1 | 3 |
| 3 | 4 | 8 | 4 | 1 | 3 |
| 3 | 5 | 5 | 4 | 1 | 3 |
| 4 | 5 | 6 | 5 | 6 | 3 |
| 3 | 4 | 8 | 5 | 6 | 3 |
| 3 | 5 | 5 | 5 | 6 | 3 |




- Every path in the MDD represents a concrete state vector

# Multi-Valued Decision Diagrams

| | | | | | |
|---|---|---|---|---|---|
| 4 | 5 | 6 | 4 | 4 | 1 |
| 3 | 4 | 8 | 4 | 4 | 1 |
| 3 | 5 | 5 | 4 | 4 | 1 |
| 4 | 5 | 6 | 4 | 1 | 3 |
| 3 | 4 | 8 | 4 | 1 | 3 |
| 3 | 5 | 5 | 4 | 1 | 3 |
| 4 | 5 | 6 | 5 | 6 | 3 |
| 3 | 4 | 8 | 5 | 6 | 3 |
| 3 | 5 | 5 | 5 | 6 | 3 |




- Every path in the MDD represents a concrete state vector
- Potential gain in memory saving: exponential (here: $54 \to 15$)
- Symbolic Reachability: explore sets of states stored as MDDs

## Symbolic Reachability Algorithm
[ICTAC 2008]

- $L$, $V$: MDDs for sets of long state vectors (level, visited)
- $R_i$: MDDs to store transition relation $i$ on short vectors
- $L_i$, $V_i$: MDDs for sets of short state vectors (level,visited for $i$)

# Symbolic Reachability Algorithm
[ICTAC 2008]

- $L$, $V$: MDDs for sets of long state vectors (level, visited)
- $R_i$: MDDs to store transition relation $i$ on short vectors
- $L_i$, $V_i$: MDDs for sets of short state vectors (level,visited for $i$)

**symbolic-reachability()**

(1) $L := \{\textsc{InitState}()\}$; $V := L$; all $R_i := \varnothing$; all $V_i := \varnothing$
(2) while $L \neq \varnothing$ do
(3) for $i \in groups$ do /* enumerate short vectors */
(4) $L_i := \pi_i([D]_{N \times K}, L) \setminus V_i$; $V_i := V_i \cup L_i$
(5) $R_i := R_i \cup \{(s, s') \mid s \in L_i \wedge s' \in \textsc{NextState}(i, s)\}$
(6) $L := \bigcup_i (R_i(L) \setminus V)$; $V := V \cup L$
(7) return $V$

# Symbolic Reachability

## Symbolic Reachability with PINS

- Global set of reachable states is computed as fix point
- Stored as a multi-valued decision diagram (MDD)
- Learn symbolic sub-groups $R_i$ on-the-fly (via NextState)

# Symbolic Reachability

## Symbolic Reachability with PINS

- Global set of reachable states is computed as fix point
- Stored as a multi-valued decision diagram (MDD)
- Learn symbolic sub-groups $R_i$ on-the-fly (via NEXTSTATE)

## Extensions

- Multiple exploration strategies:
  - Breadth-first: $(T_1 + T_2 + \cdots + T_n)^*$
  - Chaining: $(T_1 \circ T_2 \circ \cdots \circ T_n)^*$
  - Saturation-like: $(T_1^* \circ T_2^* \circ \cdots \circ T_n^*)^*$
  - Full Saturation: $((((T_1^* T_2)^* T_3)^* \cdots)^* T_n)^*$
- Static variable reordering boosts performance

# Table of Contents

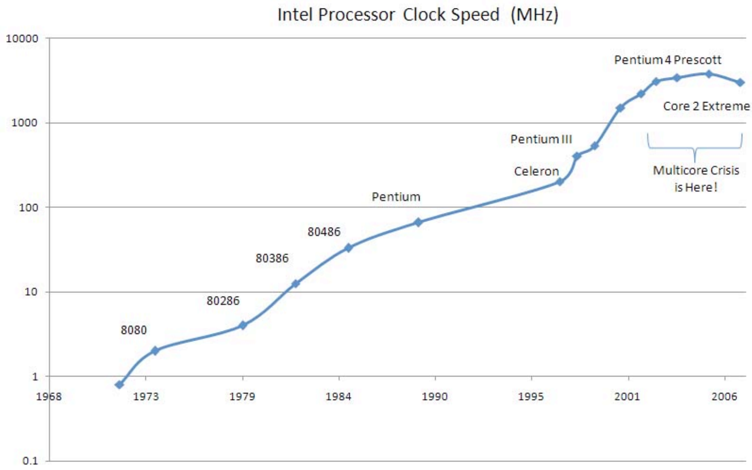
# Multi-Core Algorithms

# Multi-Core Reality

## Recent standard hardware (x86)

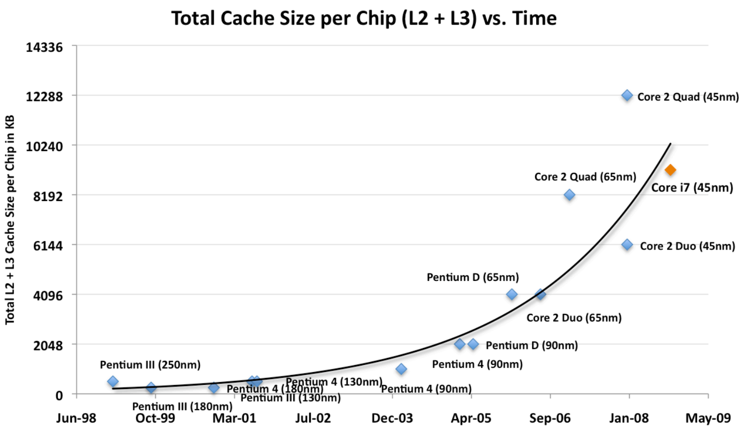
- Multiple cores per processor, multiple processors
- Typical big machine: 48–64 cores, 256 GB shared memory
- Communication via L1, L2 caches: cache-coherence protocols

# Multi-Core "Crisis"




(Source: Smoothspan)

# Moore's Law: More Cache!




(Source: Anandtech)

# Scalable Multi-Core Reachability
(cf. Holzmann, FMCAD 2006)

- Exploitable parallelism must double every 2 years (Corollary of Moore's Law)
- (Graph) Reachability is basis of many verification problems
- Multi-Core Model Checking: state-of-the-art not very impressive

# Scalable Multi-Core Reachability
(cf. Holzmann, FMCAD 2006)

- Exploitable parallelism must double every 2 years (Corollary of Moore's Law)
- (Graph) Reachability is basis of many verification problems
- Multi-Core Model Checking: state-of-the-art not very impressive


---

Having trouble with scaling simple (enumerative) reachability?

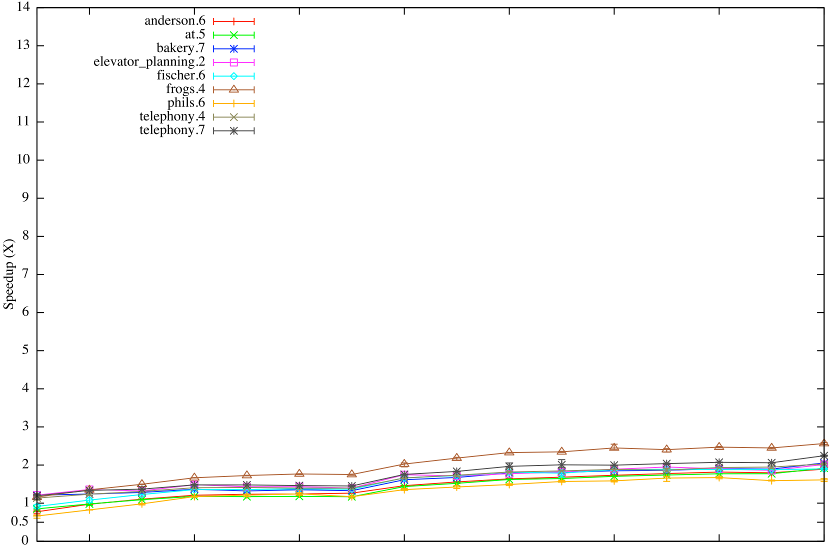
- Then what are the chances to parallelize: Liveness, partial-order reduction, symbolic reachability, . . . ?

# Multi-core reachability

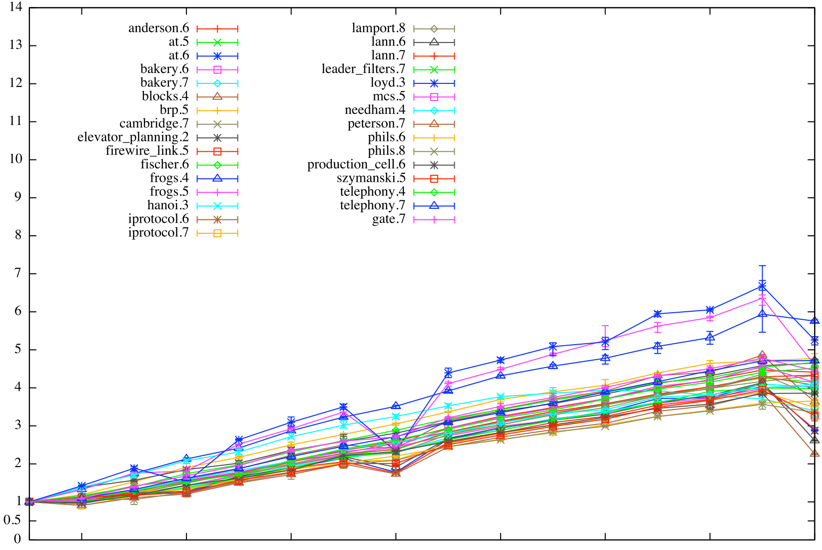
## Problems for Model Checking: Visited Set

- Parallel access to hash table: correctness and efficiency
- Parallel access requires synchronization . . . . . . lock contention
- Graph traversal: Random memory access . . . . . . . cache misses
- Main problem with cache lines: . . . . . . . . . . . . . . . . . false sharing

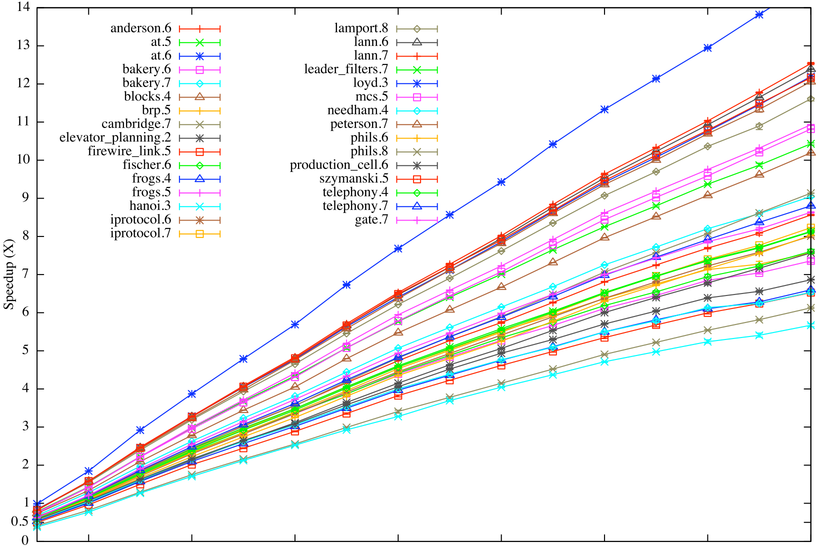
# Experiments: SPIN 5.2.4 (NASA/JPL)



# Experiments: DiVinE 2.2 (Brno,CZ)



# Experiments: LTSmin



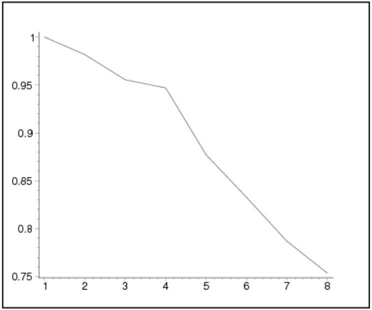
# Multi-Core Reachability

Study

- Multi-core reachability: (pseudo) breadth-first, depth-first, ...
- Load balancing

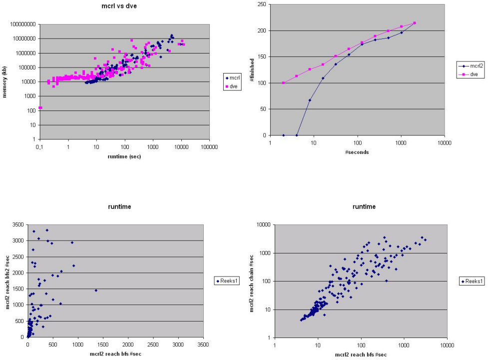
Where is efficiency lost?

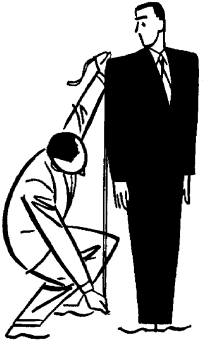
- Lock Contention
- Lock Convoying
- Cache-Line Sharing
- Two-Step Dance
- Stampeding
- ...



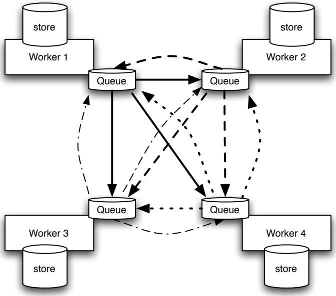
# Measure, Measure, Measure

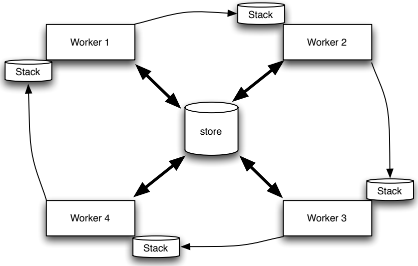
- Analyze & distill thousands of measurements
- Guides decisions what to tackle next



# Reachability Architectures




- DiVinE 2.2:
  Static Partitioning
- BFS, high comm. cost, static load balancing

- SPIN 5.2.4:
  Shared Storage & Stack Slicing
- DFS, multiple sync. points, integrated load balancing

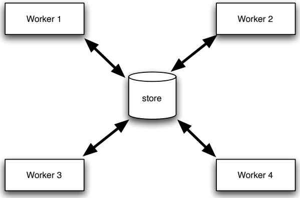
# Reachability Architectures II

Shared state storage as main sync point

- Flexible reachability algorithms
- Flexible load balancing

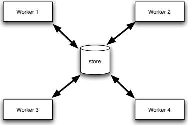
*Shared Hashtables in Parallel Model Checking* (Barnat, Ročkai 2007)

- "our shared hash tables do not scale beyond 8 cores"
- "could not investigate lockless hash table solution"
- "haven't found the cause of the scalability issues with pre-sized tables"



# Lockless Hash Table: Design
[FMCAD 2010]

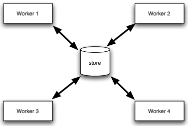
1. Investigate Requirements on Shared Storage
2. Investigate Hardware Support
   fine-grained synchronization, caches, ...
3. Exploit LTSmin Infrastructure
   incremental (Zobrist) hashing



# Lockless Hash Table: Design

[FMCAD 2010]

1. Investigate Requirements on Shared Storage
2. Investigate Hardware Support
   fine-grained synchronization, caches, ...
3. Exploit LTSmin Infrastructure
   incremental (Zobrist) hashing



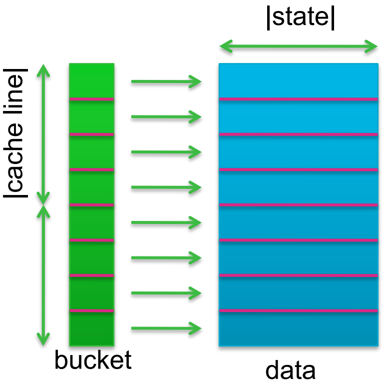

Hash Table Designed for Model Checking:

- FINDORPUT operation only
- Don't store pointers, no allocation
- No resizing!
- Fixed-size keys (state vectors)
- Low memory working set improves scalability

# Lockless Hash Table: Layout

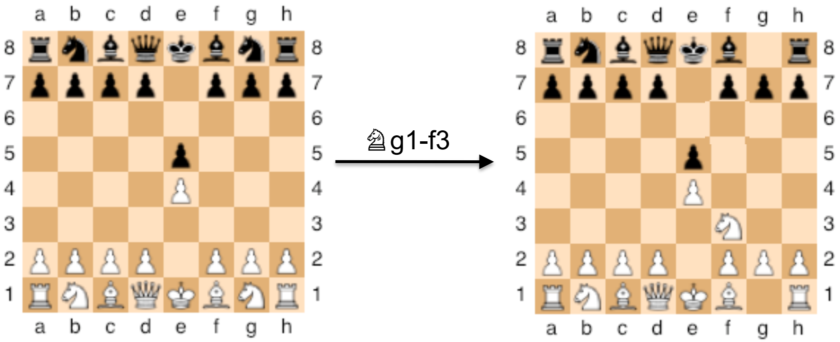
[FMCAD 2010]

- Open Addressing
- Hash Memoization
- Separate Data
- *Walking the Line*
- Lockless (CAS + write bit)




See also: Cliff Click (JavaOne 2007 presentation)

# Incremental Hashing




$H_x$ $\quad (H_x \oplus Z_{♘,g,1}) \oplus Z_{♘,f,3} =$ $\quad H_y$

Albert L. Zobrist, A New Hashing Method with Application for Game Playing, Tech. Rep. 88, Computer Sciences Department, University of Wisconsin, 1969.

# Incremental Hashing For State Vectors

**Input** : transition $s \rightarrow_i s'$
**Input** : hash value $h$ of $s$
**Output**: hash value $h'$ of $s'$

$$s = \langle s_1, \ldots, s_N \rangle$$
$$s' = \langle s'_1, \ldots, s'_N \rangle$$
$$h' \leftarrow h$$

**for** $j \in \{j \mid D^W_{i,j} = 1\}$ **do**
- $h' \leftarrow h' \oplus Z[j][s_j \bmod L]$
- $h' \leftarrow h' \oplus Z[j][s'_j \bmod L]$

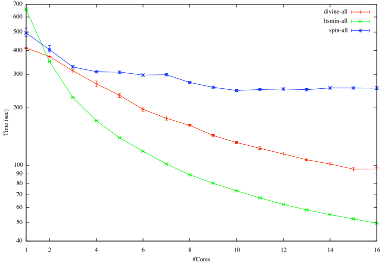
- Zobrist matrix $Z$: filled with random integers
- Dependency Matrix: find modified vector elements
- Difference to Chess: possible values not statically known
- Limit size of $Z$ (e.g., $L = 2^6$), domain of each vector slot is usually small

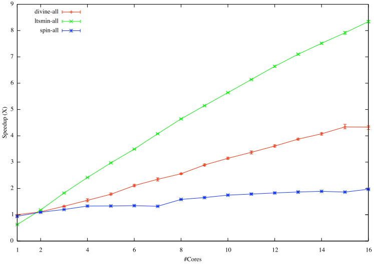
# Experiments

# Experimental Setup

- 16-core AMD Opteron 8356, 64 GB RAM
- Linux 2.6.32+patch
- BEEM model database (250+ models)
  - extensive collection of models for enumerative model checkers
  - case studies, protocols, games, planning, synthetic models, ...
  - `http://anna.fi.muni.cz/models/`
- Statically sized hash tables (no resizing)
- Speedups: relative to best sequential(!) tool

# Experiments: Summary



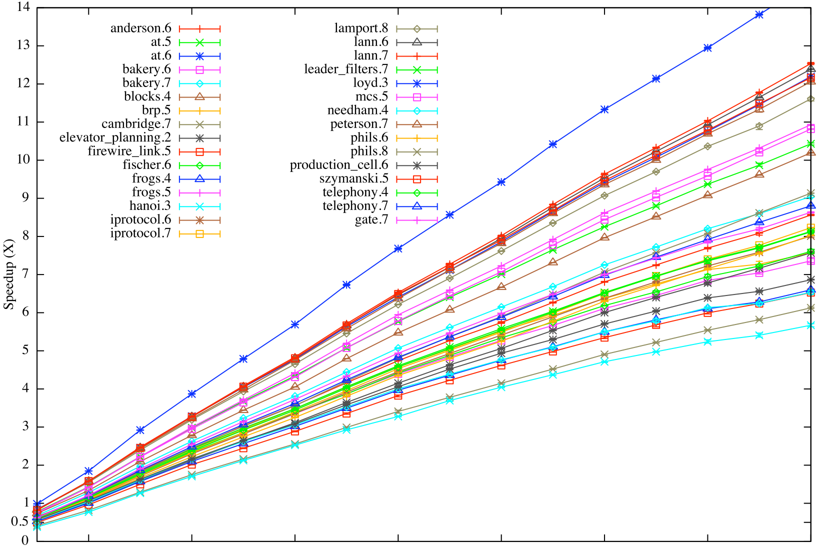




Green LTSmin (UTwente)

Red DiVinE 2.2

Blue SPIN 5.2.4

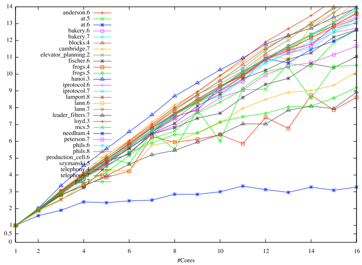
# Experiments: LTSmin (Detailed)



# Effects of Load Balancing

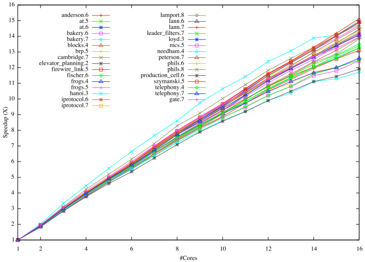
## Static Load Balancing (SLB)

- Feasible for many models
- Threads can run out of work




## Synchronous Random Polling (Sanders '97)

- Work stealing/hand-off
- Almost no overhead vs. SLB
- Additional Improvements
  - Shared-Memory Multi-Core
  - Informed Polling
  - Scare off stampeding threads



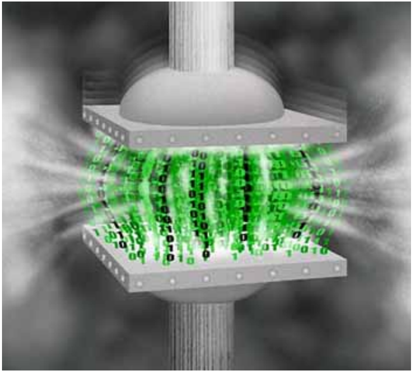
# Observations

Model Checking Limitations:

- Old Days: Memory
- Availability of large RAM:
  Time-Outs ("Patience-Out")
- Multi-Core Reachability
  - Analyzing 10 Million states/sec
    on 16-core AMD
  - Allocation rate: 1 GB/sec

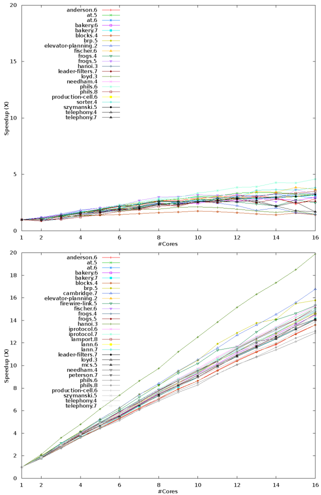
Memory is again Bottleneck

# State Vector Compression



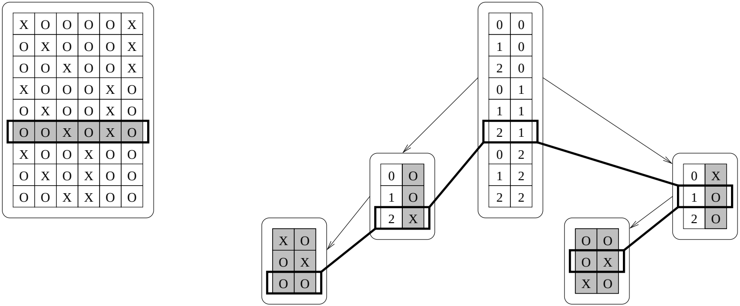
# Tree Compression
[PDMC 2007, JLC 2009]

- State vectors highly similar
- Compression via tree of tables, recursive version of SPIN's COLLAPSE
- Multi-Core version: Lockless hash table as building block!
- Increases *arithmetic intensity*: Super-linear speedups!
- "For Free"! (pays its own way)



# Tree Compression

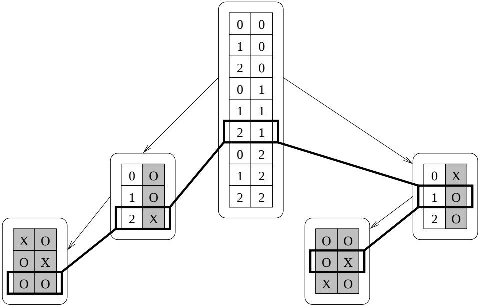
[PDMC 2007, JLC 2009]




- Folded vector $\langle 2, 1\rangle_{\mathrm{FV}}$ represents $\langle 0, 0, 1, 0, 1, 0\rangle_{\mathrm{IV}}$ resp. $\langle O, O, X, O, X, O\rangle_{\mathrm{SV}}$ (with $O \leftrightarrow 0$, $X \leftrightarrow 1$)
- Selected tree fringe of grey boxes corresponds to state vector

# Tree Compression
[PDMC 2007, JLC 2009]

| | | | | | |
|---|---|---|---|---|---|
| X | O | O | O | O | X |
| O | X | O | O | O | X |
| O | O | X | O | O | X |
| X | O | O | O | X | O |
| O | X | O | O | X | O |
| O | O | X | O | X | O |
| X | O | O | X | O | O |
| O | X | O | X | O | O |
| O | O | X | X | O | O |



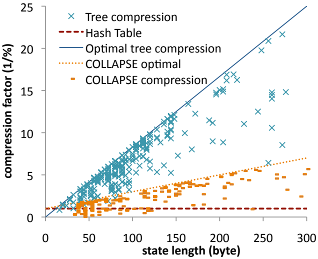

- Folded vector $\langle 2,1\rangle_{\mathrm{FV}}$ represents $\langle 0,0,1,0,1,0\rangle_{\mathrm{IV}}$ resp. $\langle O,O,X,O,X,O\rangle_{\mathrm{SV}}$ (with $O \leftrightarrow 0$, $X \leftrightarrow 1$)
- Selected tree fringe of grey boxes corresponds to state vector
- Potential gain (here $54 \rightarrow 42$ entries):
  - main table of size $N$ is only two integers wide
  - small tables of size $\mathcal{O}(\sqrt{N})$ only

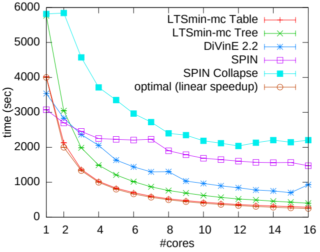
# Multi-Core Tree Compression

[NASA FM 2011, SPIN 2011]

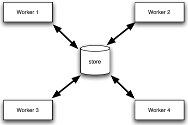
- State vectors highly similar:
  Exploit combinatorial structure
- Extreme example:
  Memory/Time usage for
  `firewire_tree.5`:

  Uncompressed: 14 GB
  Tree Compression: 96 MB
- Near 100 % efficiency!
- Dependency Matrix:
  Incremental Tree Compression



# Observations




- Centralized state storage scales at least as well as static state space partitioning
- Shared state storage orthogonal to: search strategy, load balancing, ...
- Simpler architecture
- Arithmetic Intensity (hide memory latency)
- Data layout important (caches), sadly no language support
- Designed from the ground up for Scalability
- Establishing correctness of implementation is a pain ...

# Algorithm Engineering

- $> 10\times$ improvement over sequential algorithms
- Beats state-of-the-art tools, reopens earlier conclusions
- Research questions guided by experiments
- Engineering approach, repeatable benchmark results


- Hash table as building block for multi-core xDDs, liveness checking, ...

# Conclusion

# Model Checking for Planning?

## Adoption obstacles

- Modeling effort
  - many languages
  - avoid modeling?
- Scalability
  - parallel components
  - data, buffers, . . .
- Complex tools
  - algorithms, heuristics
  - low-level details

# Model Checking for Planning?

## Adoption obstacles

- Modeling effort
  - many languages
  - avoid modeling?
- Scalability
  - parallel components
  - data, buffers, ...
- Complex tools
  - algorithms, heuristics
  - low-level details

## Algorithmic solutions (combinatorics, locality)

- on-the-fly model checking
- symbolic model checking
- bounded model checking
- parallel model checking
- partial-order reduction
- symmetry reduction

# Model Checking for Planning?

## Adoption obstacles

- Modeling effort
  - many languages
  - avoid modeling?
- Scalability
  - parallel components
  - data, buffers, ...
- Complex tools
  - algorithms, heuristics
  - low-level details

## Algorithmic solutions (combinatorics, locality)

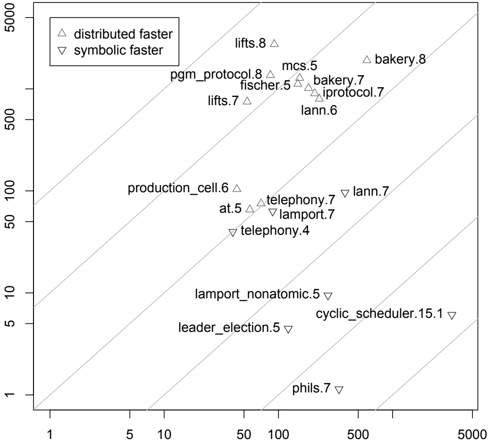
- on-the-fly model checking
- symbolic model checking
- bounded model checking
- parallel model checking
- partial-order reduction
- symmetry reduction

## Problem: Algorithms tied to specification languages

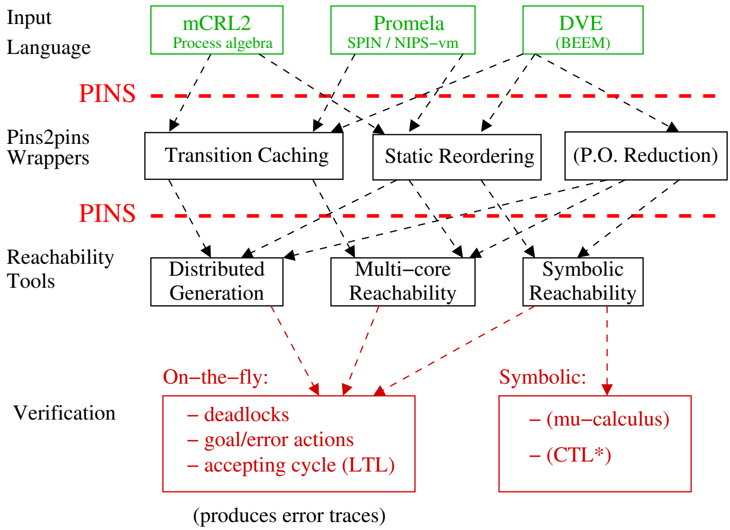
- No particular technique suits all applications / models
- Users need to rewrite model in different languages

# Silver Bullets?

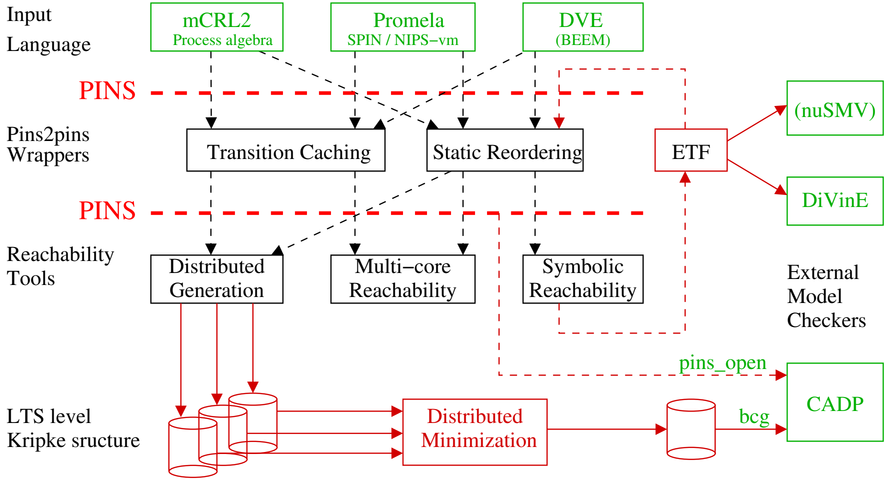
- No clear winner
- BUT: also depends on modelling
- PINS matrix can be used as predictor



# LTSmin Verification Capabilities



# Links to External Tools



# Portfolio of Language Modules

| | | | |
|---|---|---|---|
| $\mu$CRL | process algebra | (CWI) | |
| mCRL2 | process algebra, INESS | (TU/e) | |
| PROMELA | SPIN | (NASA/JPL) | Prototype |
| PROMELA | SpinJa | (UTwente) | |
| DVE | DiVinE-cluster | (MU Brno) | |
| DVE2 | DiVinE model checking toolset | (MU Brno) | |
| ETF | Enumerated Table Format | (LTSmin) | |
| GNA | Genetic Network Analyzer | (INRIA) | Prototype |
| ODE | Maple | (EC-MOAN) | Prototype |

# Portfolio of Tools

| | |
|---|---|
| ⟨spec⟩2lts-gsea | Depth/Breadth-First Enumerative Reachability |
| ⟨spec⟩-reach | MDD-based symbolic reachability |
| ⟨spec⟩2lts-mpi | Distributed state-space generation |
| ⟨spec⟩2lts-mc | Multi-Core Enumerative Reachability |
| ⟨spec⟩2torx | TorX tester RPC interface |
| pins_open | Connection to CADP toolset (VASY/INRIA) |
| ltsmin-mpi | Signature-based distributed minimization |
| ce-mpi | Orzan's distributed cycle elimination |
| ltsmin-tracepp | (Error) trace pretty printer |

Next: Partial-Order Reduction (POR),
Linear Temporal Logic (LTL), $\mu$-calculus
Saturation,

# Who benefits from LTSmin?

**Useful for end users: larger case studies**

- Model in a suitable (your favourite) specification language
- Decide later what model checking algorithm to use

# Who benefits from LTSmin?

## Useful for end users: larger case studies

- Model in a suitable (your favourite) specification language
- Decide later what model checking algorithm to use

## Useful for tool providers: algorithms for free

- Your Domain Specific Specification Language can get HPMC
- Ideally, LTSmin can be viewed as a library

# Who benefits from LTSmin?

### Useful for end users: larger case studies

- Model in a suitable (your favourite) specification language
- Decide later what model checking algorithm to use

### Useful for tool providers: algorithms for free

- Your Domain Specific Specification Language can get HPMC
- Ideally, LTSmin can be viewed as a library

### Useful for researchers: rigorous benchmarking

- LTSmin allows benchmarking algorithms on one model
- LTSmin allows comparing languages on one algorithm

# Literature

Check out: http://fmt.cs.utwente.nl/tools/ltsmin/


- Alfons Laarman, Jaco van de Pol, Michael Weber,
  Parallel Recursive State Compression for Free . . . . . . . . . . . . . . (SPIN 2011)
- Alfons Laarman, Jaco van de Pol, Michael Weber,
  Multi-Core LTSmin: Marrying Modularity and Scalability (NFM2011,tool)
- Stefan Blom, Jaco van de Pol, Michael Weber,
  LTSmin: Distributed and Symbolic Reachability . . . . . . . . (CAV 2010, tool)
- Alfons Laarman, Jaco van de Pol and Michael Weber, . . . (FMCAD 2010)
  Boosting Multi-Core Reachability Performance with Shared Hash Tables
- Stefan Blom and Jaco van de Pol, Symbolic Reachability for
  Process Algebras with Recursive Data Types . . . . . . . . . . . . . . . . (ICTAC'08)
- Stefan Blom, Bert Lisser, Jaco van de Pol and Michael Weber,
  A Database Approach to Distributed State-Space Generation (JLC 2009)
- Stefan Blom and Jaco van de Pol, Distributed Branching Bisimulation
  Minimization by Inductive Signatures . . . . . . . . . . . (PDMC'09; EPTCS 14)

# Model Checking (and related) Venues

| | |
|---|---|
| CAV | Computer Aided Verification |
| TACAS | Tools and Algorithms for the Construction and Analysis of Systems |
| VMCAI | Verification, Model Checking, and Abstract Interpretation |
| ATVA | Automated Technology for Verification and Analysis |
| FMCAD | Formal Methods in Computer Aided Design |
| SPIN | SPIN Workshop on Model Checking Software |
| PDMC | Parallel and Distributed Methods in VerifiCation |
| MoChArt | Model Checking and Artificial Intelligence |
| QEST | Quantitative Evaluation of SysTems |
| FORMATS | Formal Modelling and Analysis of Timed Systems |