



Copyright © 1999 David Farley, d-farley@metalab.unc.edu
<http://metalab.unc.edu/Dave/difun.html>

This cartoon is made available on the Internet for personal viewing only. Opinions expressed herein are solely those of the author.

Bill Gates' next book

Abstraction in Software Model Checking

Principles and Practice

Dennis Dams
Bell Labs & TU/e

Outline

PART I

- Introduction / Methodology
- Theory

PART II

- Techniques & Algorithms
- Tools

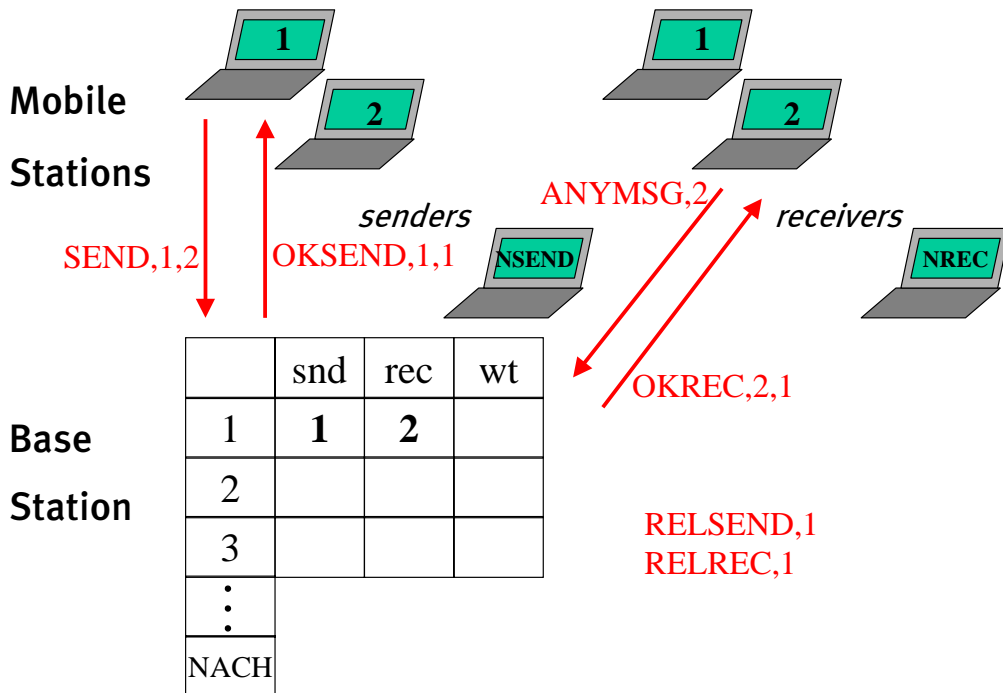
(PART III)

- Challenges

Warm-up: Verifying MASCARA

- Protocol to extend ATM into wireless
- 10,000s lines of SDL code
(= 100s of pages of SDL diagrams)

A Promela model of MASCARA



A Correctness Property

“For all values of NSEND, NREC, NACH,
for every receiver r and every channel c :

If, at some point,

channel c is allocated to receiver r (φ)

then somewhere before that point

**an entry of the form (c, \dots, r, \dots) has
been inserted in the Base Station's table.”** (ψ)

$\forall \text{NSEND, NREC, NACH: Nat}$

$\forall r: 1..NREC \ \forall c: 1..NACH \quad \neg((\neg\varphi) \cup \psi)$

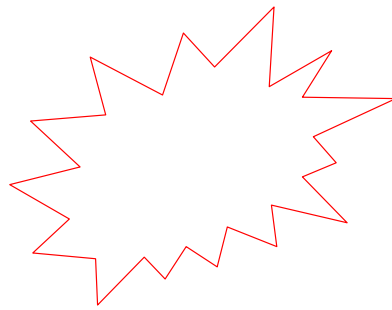
Verifying a Concrete Instance

For $NSEND=3, NREC=3, NACH=2,$
 $r=2, c=1: \quad \neg((\neg\varphi) \cup \psi)$

But:

Would have to check several “representative” instances,
and argue why that suffices.

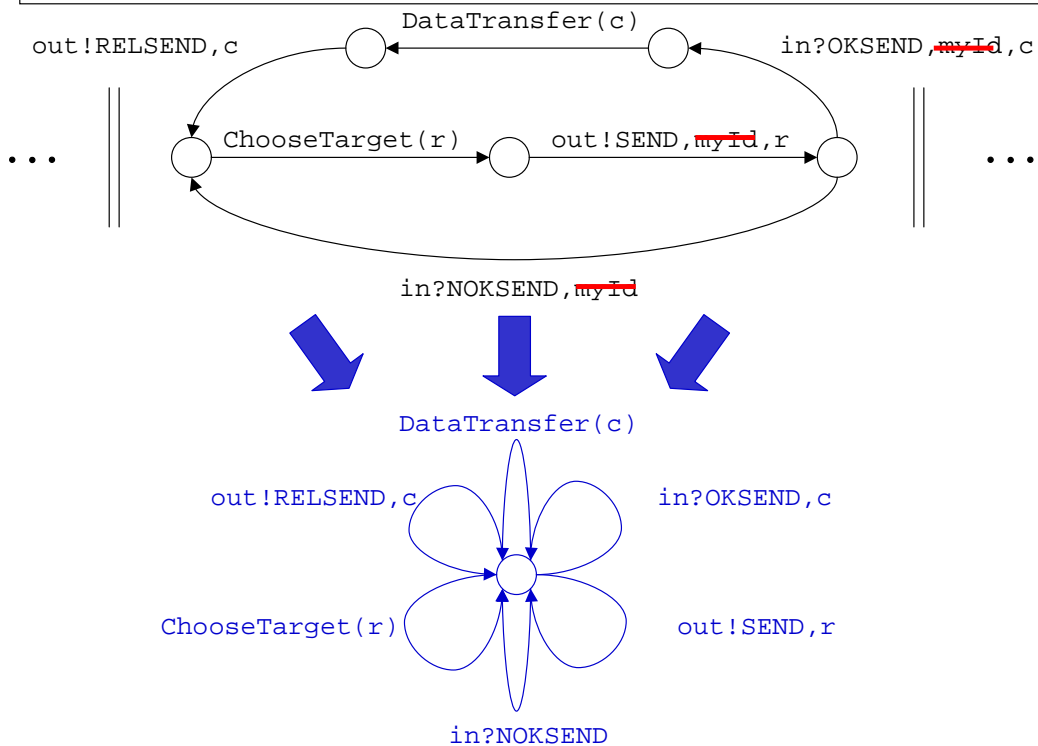
And:



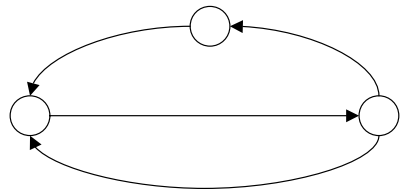
Step 1: Data Abstraction

```
typedef MesgType = {SEND, OKSEND, ANYMSG, ...};  
SenderId = 1..NSEND;  
ReceivId = 1..NREC; {R, NR};  
ChanIndx = 0..NACH; {C, NC};
```

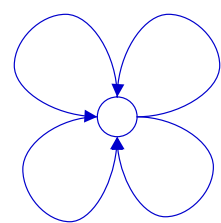
Step 2: Abstracting senders' control



Step 3: Abstracting away the NR receivers



The distinguished receiver R



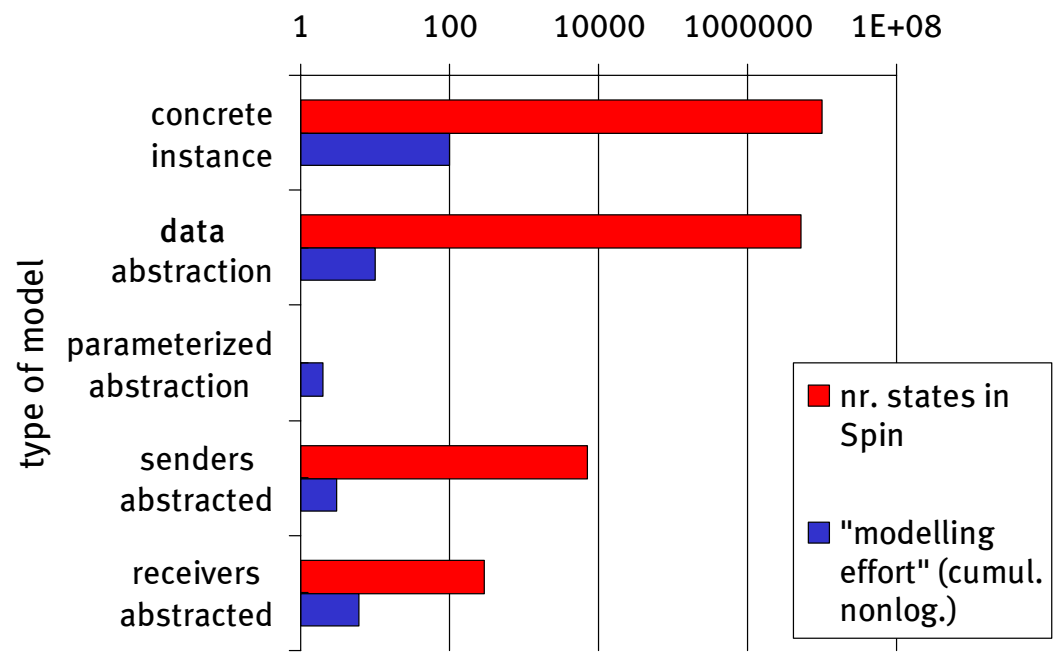
All other receivers (NR)

$\forall \text{NSEND}, \text{NREC}, \text{NACH}:\text{Nat}$

≥ 2

$\forall r:1..NREC \forall c:1..NACH: \neg((\neg\varphi) U \psi)$

Verification results



The Mission

program
(source code)

property
(temporal logic)

$P \text{ sat. } \varphi$

interpretation

$TS(P) \models \varphi$

model
checking

$"MC(P, \varphi)"$

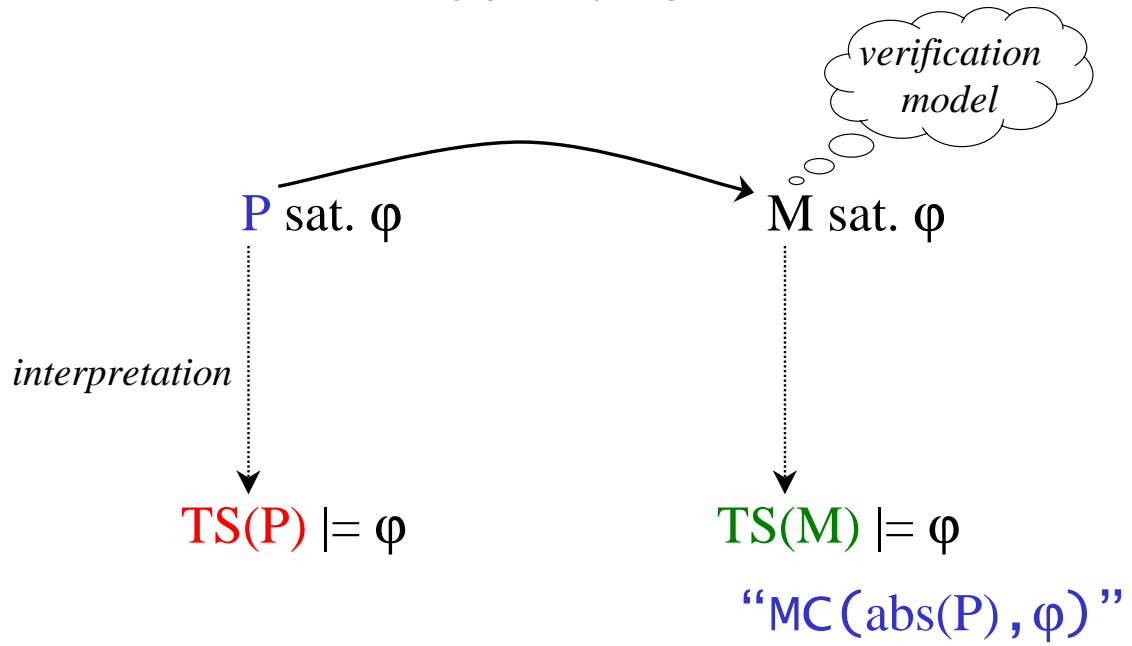
The Problem

$P \text{ sat. } \varphi$

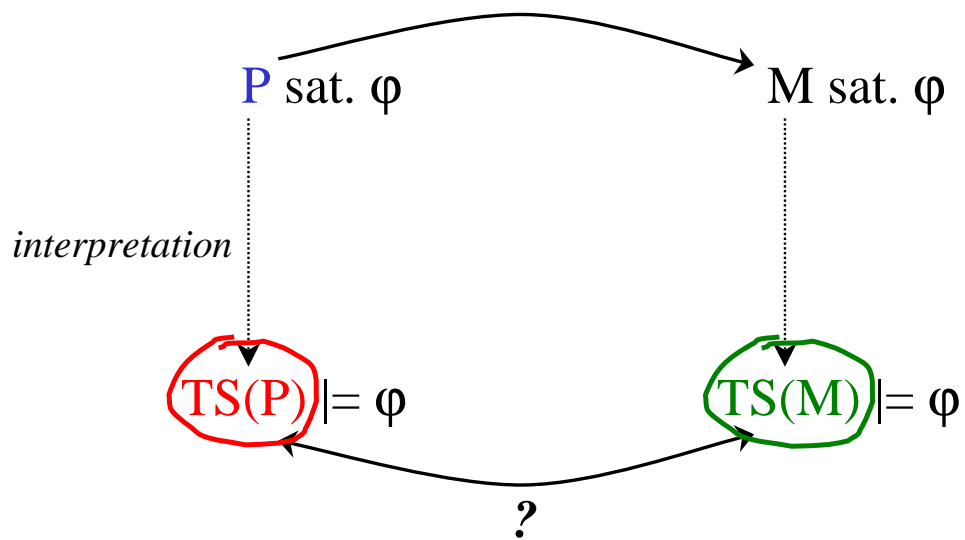
interpretation

$TS(P) \not\models \varphi$

Abstraction



Abstraction Relation

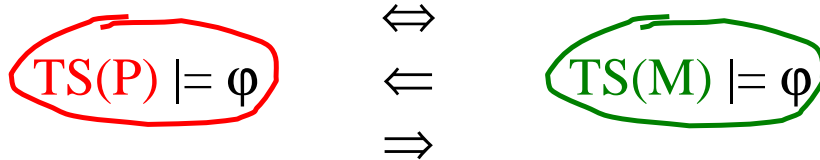


Property Preservation: A Choice

\Leftrightarrow “strong preservation”

\Leftarrow “weak preservation with false negatives”

\Rightarrow “weak preservation with false positives”

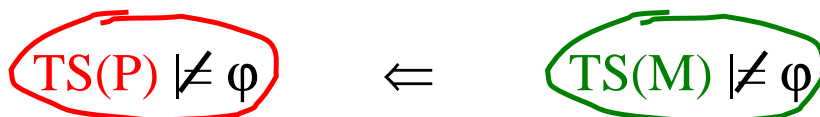


Property Preservation: A Choice

\Leftrightarrow “strong preservation”

\Leftarrow “weak preservation with false negatives”

\Rightarrow “weak preservation with false positives”



Strong Preservation

- Puts lower bound on size of suitable abstractions (*e.g. bisim. reduction*)
- Difficult to construct abstraction in general

$$\text{TS(P)} \models \varphi \quad \Leftrightarrow \quad \text{TS(M)} \models \varphi$$

Weak Preservation

- + No lower bound and easy to construct
- “Incomplete proof method”

$$\text{TS(P)} \models \varphi \quad \Leftarrow \quad \text{TS(M)} \models \varphi$$

OR

$$\text{TS(P)} \not\models \varphi \quad \Leftarrow \quad \text{TS(M)} \not\models \varphi$$

Usually: *iterative refinement until strong preservation*

Weak Preservation

- + No lower bound and easy to construct
- “Incomplete proof method”

$$\begin{array}{ccc} \text{TS(P)} \models \varphi & \Leftarrow & \text{TS(M)} \models \varphi \\ & \text{OR} & \\ \text{TS(P)} \not\models \varphi & \Leftarrow & \text{TS(M)} \not\models \varphi \end{array}$$

Usually: *iterative refinement until strong preservation*

False Negatives or Positives?

```
while MC(abs(P), φ) =  
  no + cntrexp do  
{ inspect cntrexp;  
  if true_cntrexp  
  then debug P  
  else refine abs;  
}
```

$\text{TS(P)} \models \varphi \Leftarrow \text{TS(M)} \models \varphi$

```
while MC(̃abs(P), φ) =  
  yes + evidence do  
{ inspect evidence;  
  if true_evidence  
  then halt  
  else refine abs;  
}
```

$\text{TS(P)} \models \varphi \Rightarrow \text{TS(M)} \models \varphi$

False Negatives or Positives?

```
while MC(abs(P), φ) =  
  no + cntrexmp do  
{ inspect cntrexmp;  
  if true_cntrexmp  
  then debug P  
  else refine abs;  
}
```

$TS(P) \models \varphi \Leftarrow TS(M) \models \varphi$

```
while true do  
{  
  while MC(̂abs(P), φ) =  
    yes + evidence do  
  { inspect evidence;  
    if true_evidence  
    then halt  
    else refine abs;  
  }  
  debug P;  
}
```

$TS(P) \models \varphi \Rightarrow TS(M) \models \varphi$

False Negatives or Positives?

In practice the duals are not equally good:

- Goal is to have correct programs: fits better with true positives
- φ is typically universal (\in LTL / \forall CTL* / μ calc): counterexamples can be dealt with one-by-one, while evidence needs to be considered as a whole

$TS(P) \models \varphi \Leftarrow TS(M) \models \varphi$

$TS(P) \models \varphi \Rightarrow TS(M) \models \varphi$

Outline

PART I

- Introduction / Methodology
- ▶ • Theory

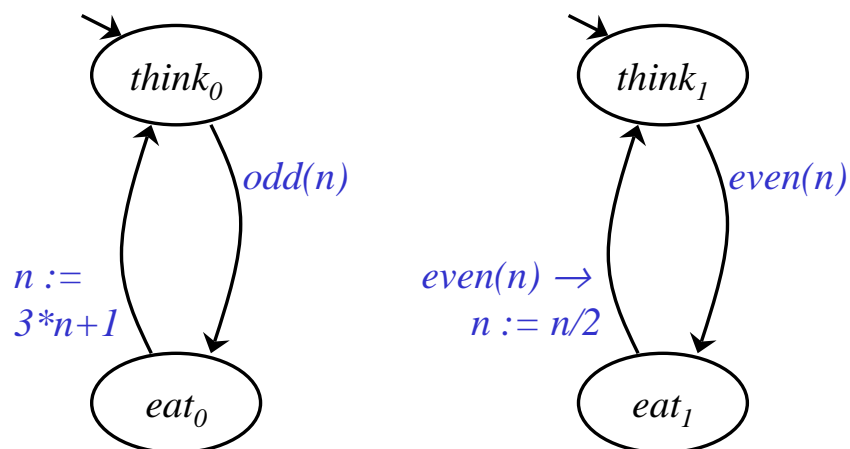
PART II

- Techniques & Algorithms
- Tools

(PART III)

- Challenges

Running Example



(global) state: $\langle l_0, l_1, n \rangle$

Temporal Logic (CTL*)

→ Can express static and dynamic aspects:
propositional logic + temporal operators

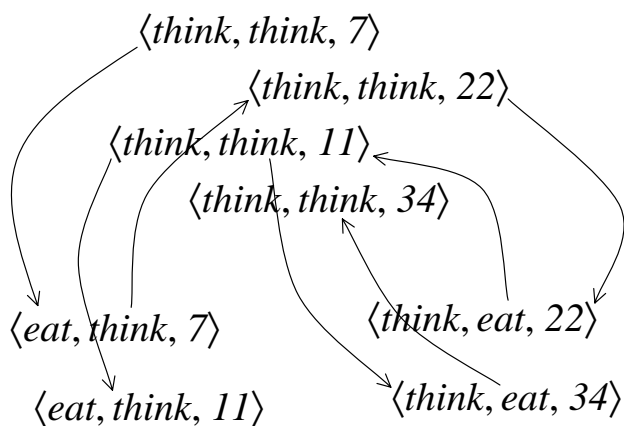
$$\forall \mathbf{G} \neg (l_0 = \text{eat} \wedge l_1 = \text{eat})$$

$$\forall \mathbf{G} (l_0 = \text{eat} \rightarrow \forall \mathbf{F} l_1 = \text{eat})$$

$$\forall \mathbf{G} (l_1 = \text{eat} \rightarrow \forall \mathbf{F} l_0 = \text{eat})$$

$$\exists \mathbf{F} (l_0 = \text{eat} \vee l_1 = \text{eat})$$

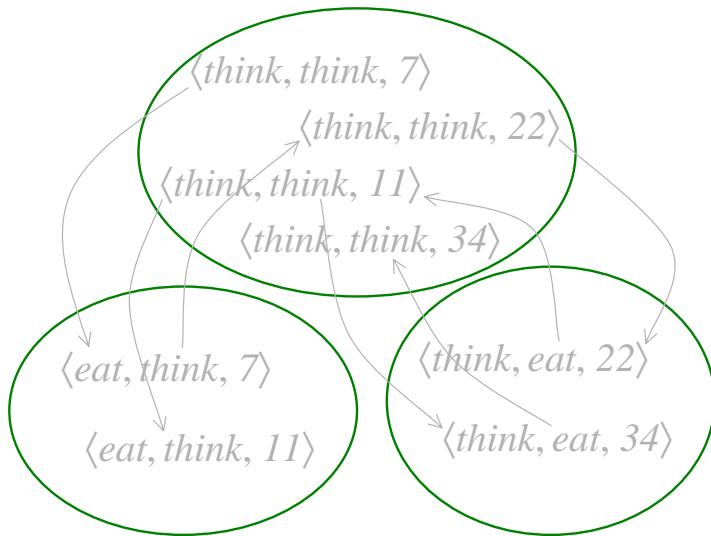
Abstracting Kripke structures



Abstraction =
partitioning of states
into *abstract states*

For now, assume
 $\alpha : \text{states} \rightarrow \text{abs. states}$
 $\alpha(c) = a : "c \in a"$

Abstracting Kripke structures



Abstraction =
partitioning of states
into *abstract states*

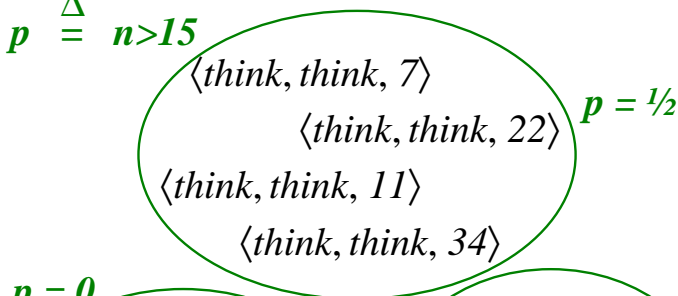
For now, assume
 $\alpha : \text{states} \rightarrow \text{abs. states}$
 $\alpha(c) = a : "c \in a"$

$$C \models \varphi \iff A \models \varphi$$

Abs. Kripke structs: Statics

For $c \in a$: $c \models p \iff a \models p$ (for all $p \in \text{Prop}$)

$p \stackrel{\Delta}{=} n > 15$

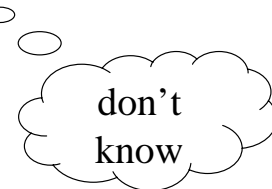


$p(a) =$

1 if $\forall c \in a . p(c) = 1$

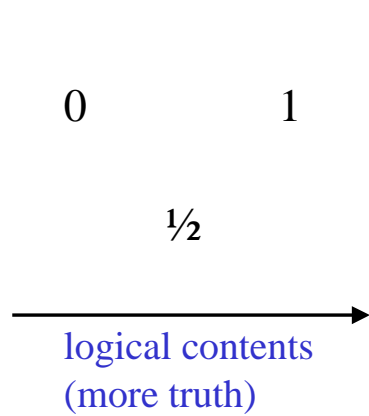
0 if $\forall c \in a . p(c) = 0$

$\frac{1}{2}$ otherwise



Intermezzo: 3-valued logic

information contents
(more knowledge)



Kleene interpretation:

$$\neg \frac{1}{2} = \frac{1}{2}$$

$$0 \wedge \frac{1}{2} = \frac{1}{2} \wedge 0 = 0$$

$$1 \wedge \frac{1}{2} = \frac{1}{2} \wedge 1 = \frac{1}{2}$$

$$\frac{1}{2} \wedge \frac{1}{2} = \frac{1}{2}$$

$$0 \vee \frac{1}{2} = \frac{1}{2} \vee 0 = \frac{1}{2}$$

$$1 \vee \frac{1}{2} = \frac{1}{2} \vee 1 = 1$$

$$\frac{1}{2} \vee \frac{1}{2} = \frac{1}{2}$$

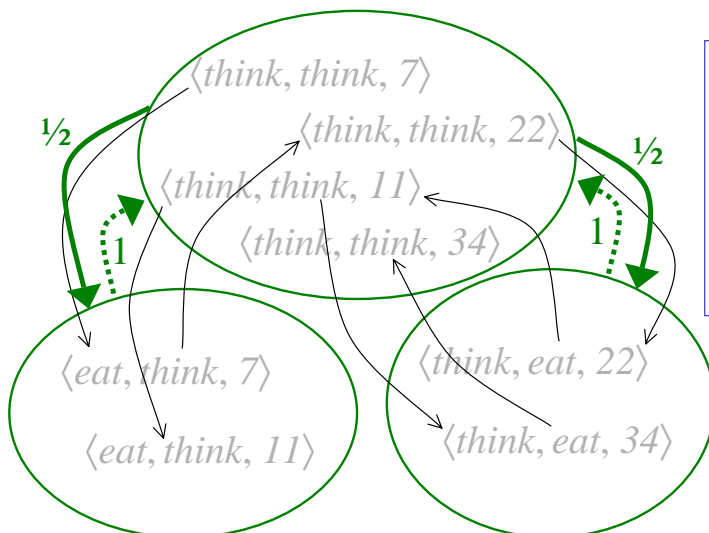
$$0 \rightarrow \frac{1}{2} = \frac{1}{2} \rightarrow 1 = 1$$

$$1 \rightarrow \frac{1}{2} = \frac{1}{2} \rightarrow 0 = \frac{1}{2}$$

$$\frac{1}{2} \rightarrow \frac{1}{2} = \frac{1}{2}$$

Abs. Kripke structs: Dynamics

For $c \in a$: $c \models \varphi \Leftarrow a \models \varphi$ (for $\varphi = \exists \psi, \neg \exists \psi$)



$R(a,b) =$

1 if $\forall c \in a \exists d \in b. R(c,d)=1$

0 if $\forall c \in a \forall d \in b. R(c,d)=0$

$\frac{1}{2}$ otherwise

..... \rightarrow “constrained”

———— \rightarrow “free”

Preservation Theorem

For all $\varphi \in \text{CTL}^*$:

- if $a \models_3 \varphi = 1$, then $c \models \varphi$
- if $a \models_3 \varphi = 0$, then $c \not\models \varphi$
- (if $a \models_3 \varphi = \frac{1}{2}$, then $c \models \varphi$ or $c \not\models \varphi$)




*abstract,
3-valued
world*




*concrete,
2-valued
world*


Another way to look at $\models_3 \varphi$

Define 2-valued \models on abstract side for pnf as follows:

$a \models p$ iff 
(p holds in every $c \in a$)

$a \models \neg p$ iff 
($\neg p$ holds in every $c \in a$)

$a \models \exists \dots$ iff 
(there exists a path, from every $c \in a$)

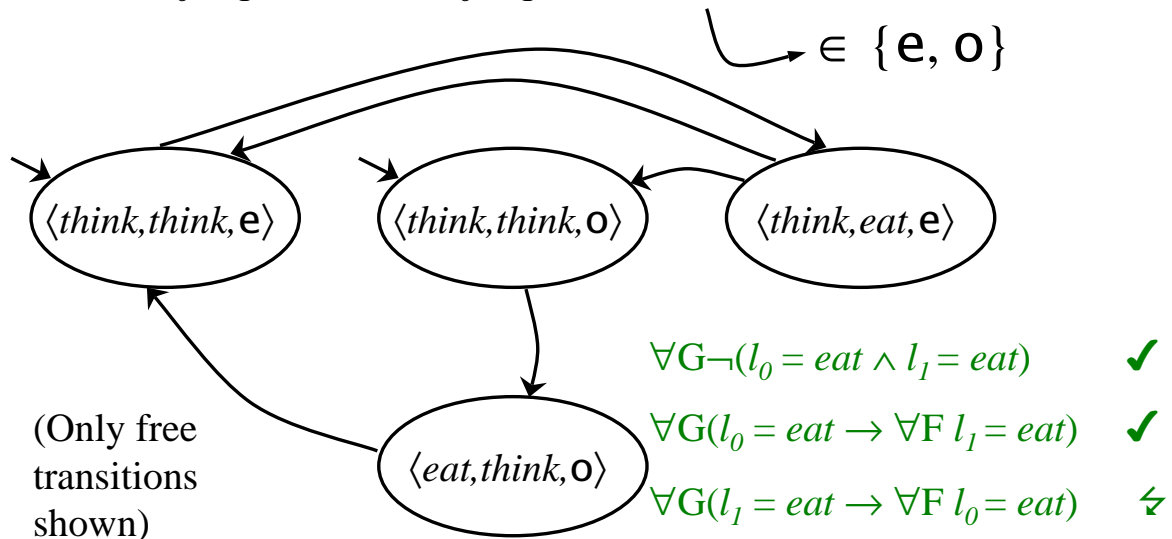
$a \models \forall \dots$ iff 
(there may exist a path, from some $c \in a$)

Model Checking on Abstract Kripke Structures

1. Bring φ and $\neg\varphi$ in pnf (push negations inside): φ' , φ''
2. Model check both, using the interpretation on prev. slide
3. Return “yes” if φ' succeeds, “no” if φ'' succeeds, “don’t know” otherwise

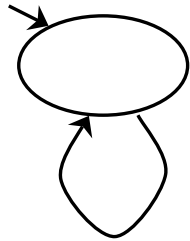
Dinner Time

Let $\alpha(\langle l_0, l_1, n \rangle) = \langle l_0, l_1, \text{parity}(n) \rangle$



An Existential Property

Add a “restart” process:

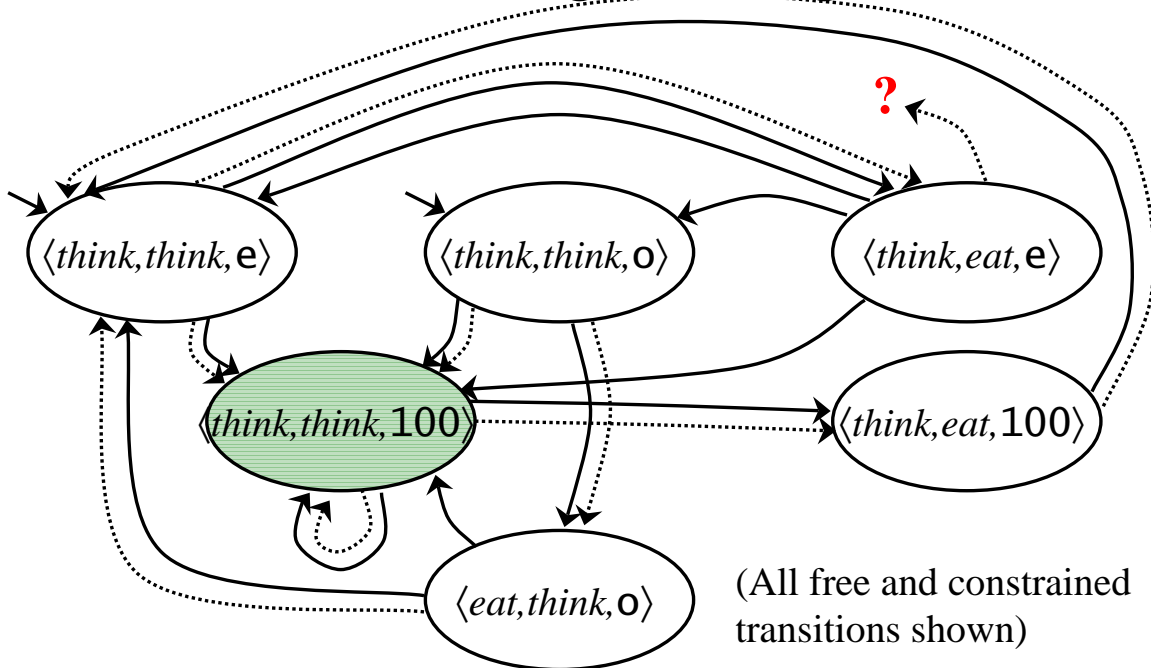


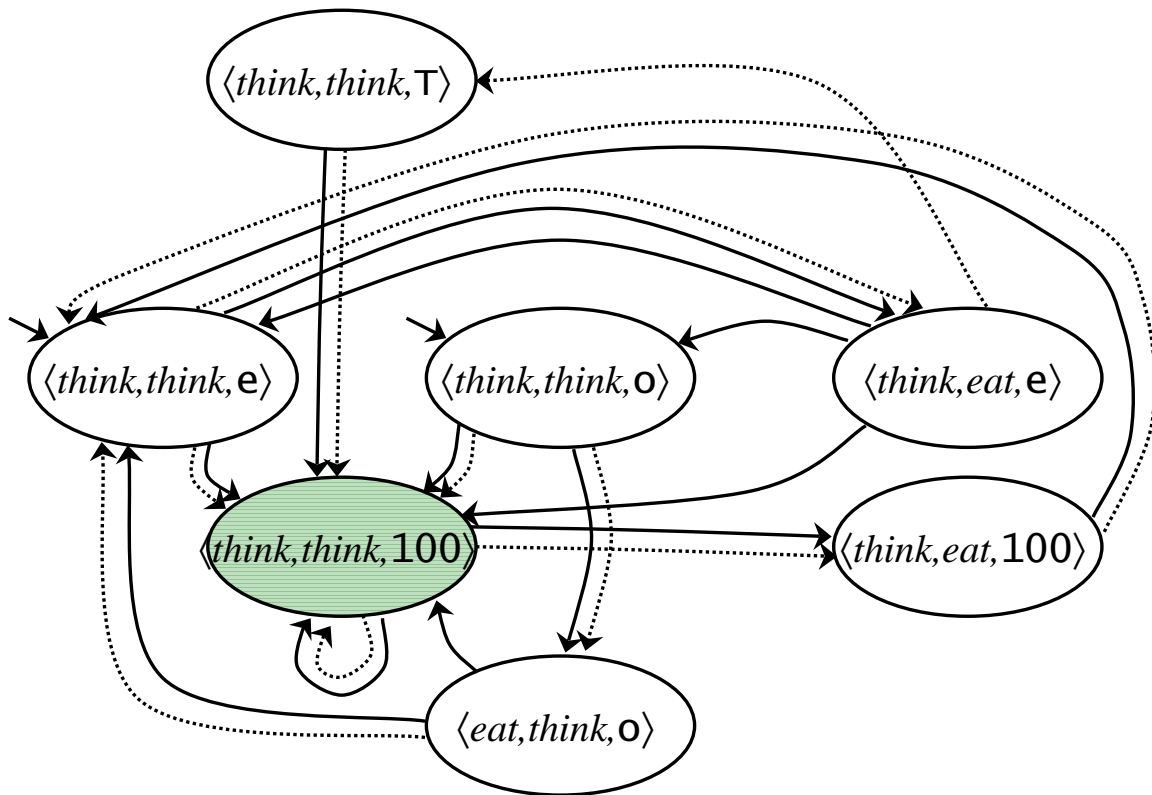
$\forall G \exists F$ “restart” ?

$l_0 = think \wedge l_1 = think$
 $\rightarrow n := 100$

$\alpha(\langle l_0, l_1, n \rangle) = \langle l_0, l_1, parity'(n) \rangle$
 $\hookleftarrow \in \{e, o, 100\}$

Something missing





Galois Connection Framework

$\gamma : \text{abs. states} \rightarrow 2^{\text{states}}$

$\gamma(a) = \text{the set of all states described by } a$

$\gamma(a') \subset \gamma(a) : a' \text{ is more precise than } a$

$\alpha : 2^{\text{states}} \rightarrow \text{abs. states}$

$\alpha(S) = \text{the most precise description of } S$

There is an old story of a boilermaker who was hired to fix a huge steamship boiler system that was not working well. After listening to the engineer's description of the problems and asking a few questions, he went to the boiler room. He looked at the maze of twisting pipes, listened to the thump of the boiler and the hiss of escaping steam for a few minutes, and felt some pipes with his hands. Then he hummed softly to himself, reached into his overalls and took out a small hammer, and tapped a bright red valve, once. Immediately the entire system began working perfectly, and the boilermaker went home. When the steamship owner received a bill for \$1,000 he complained that the boilermaker had only been in the engine room for fifteen minutes, and requested an itemized bill. This is what the boilermaker sent him:

For tapping with hammer:	.50
For knowing where to tap:	\$999.50
Total:	<u>\$1,000.00</u>

RICHARD BANDLER AND JOHN GRINDER
Frogs into Princes: Neuro Linguistic Programming, 1979

Outline

PART I

- Introduction / Methodology
- Theory

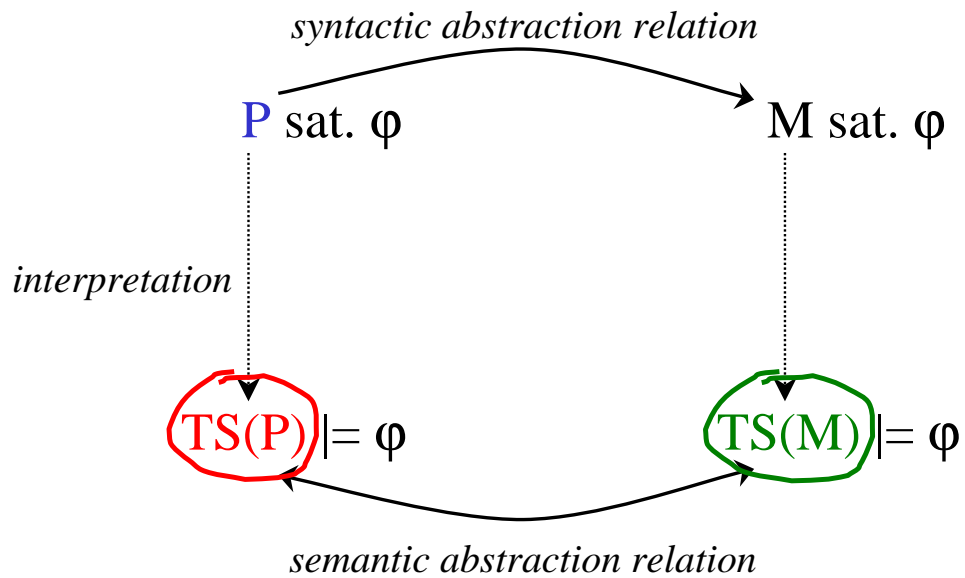
▶ PART II

- Techniques & Algorithms
- Tools

(PART III)

- Challenges

Formal Abstraction



Issues

- Which techniques/algorithms can be used to construct $M = \text{abs}(P)$?
- How to find a suitable abstraction abs ?
(given a program and correctness property)

Abstract Interpretation: Example

```

type E1 = {e1, e2, e3,
           e4, e5, ...}
type Li = listof E1

```

```

fun head : Li->E1 =
  ...
fun tail : Li->Li =
  ...
fun one_e1:Li->Bool

```

```

type E1 = {e, e', ne}
type Li = {eps, e, e',
           ee', e'e}

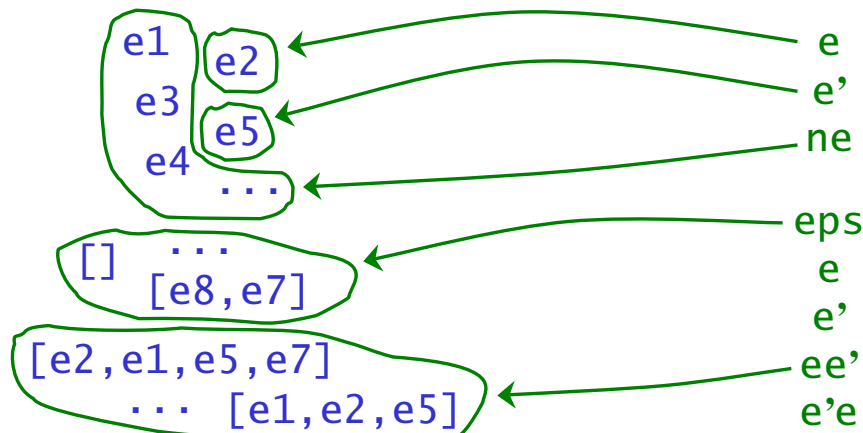
```

```

fun head : Li->E1 =
  ...
fun tail : Li->Li =
  ...
fun one_e1:Li->Bool

```

Abstract Interpretation: Example



head([e2, e1, e5, e7])=e2

head(ee')=NONDET(e, ne)

one_e1([])=false

one_e1(eps)=
NONDET(true, false)

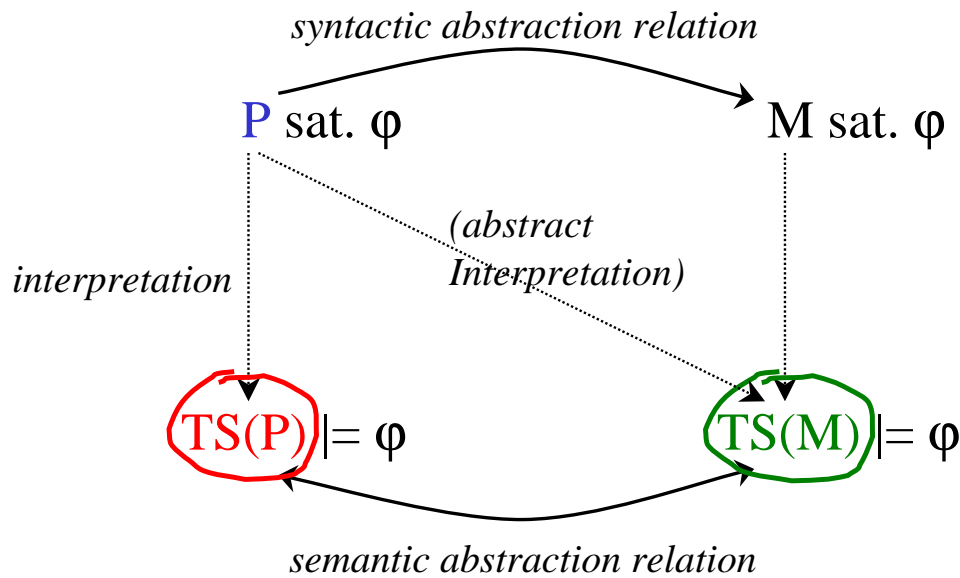
Abs. Int. + M.C.: Tools

- α SPIN (U. Malaga). XML for transformations
- Bandera (Kansas State U.) Library of abstractions.
- FeaVer (Bell Labs). Per-statement lookup tables.
- 3VMC (Tel Aviv U.)

Abstract Interpretation: Facts

- essence (narrow sense): replacing ADTs by smaller ADTs
- weak preservation
- amounts to manually specifying $P \rightarrow M$; effort is in finding abs. and justifying their correctness (safety proving)

Formal Abstraction



Program Slicing: Example

```
...  
{ float x,y,z; int n;  
  n = *p;  
  z = x;  
  if (n>0) x = pow(x,y);  
  printf("x = %f\n", x);  
...  
...
```

slicing
criterion

Program Slicing: Example

```
...  
{ float x,y,z; int n;  
  n = *p;  
  z = x;  
  if (n>0) x = pow(x,y);  
  printf("x = %f\n",x);  
...  
...
```

Program Slicing: Tools

- Bandera slicer. For Java
- CodeSurfer (U. Wisconsin-Madison → GrammaTech). ANSI-C; <100 KSLOC
- Unravel (Nat. Inst. Stand. & Tech.) ANSI-C

Issues: language, static / dynamic, intra /inter-procedural, integr. pointer analysis

Program Slicing: Facts

- “all-or-nothing” per-variable abstraction
- strong preservation (often too much detail)
- $P \rightarrow M$ automatic (given criterion)
- tools aimed at analysis / code understanding

Too Much Detail

...

```
if (BatteryPwr() > BATT_LOW)
signal_strength *= 2 /*double*/
else
  signal_strength += 1; /*step*/
```

...

Variable Hiding: Example

```
int timer;  
...  
timer--;  
saved_tm = timer;  
if (timer>0)  
    retry_shutoff(MainEngine,&stat)  
else  
    raise_alarm();  
if (stat==ok) ...
```

Variable Hiding: Example

```
int timer;  
...  
timer--;  
saved_tm = timer;  
if (NONDET)  
    retry_shutoff(MainEngine,&stat)  
else  
    raise_alarm();  
if (stat==ok) ...
```

Variable Hiding: Tools

- abC (Bell Labs). ANSI-C(+), includes pointer alias analysis
- predefined abstraction in Bandera. Java, being extended with alias analysis
- Pet (Bell Labs). Pascal

Variable Hiding: Facts

- “dual” to slicing + cut-off boundary
- weak preservation (tunable)
- $P \rightarrow M$ automatic (given hiding crit.)
- smaller state vectors, hopefully smaller state space

Transforming C

```

if (k < m) {
  int n;
  a[i++] = b[j++] = (n = k++, k);
}

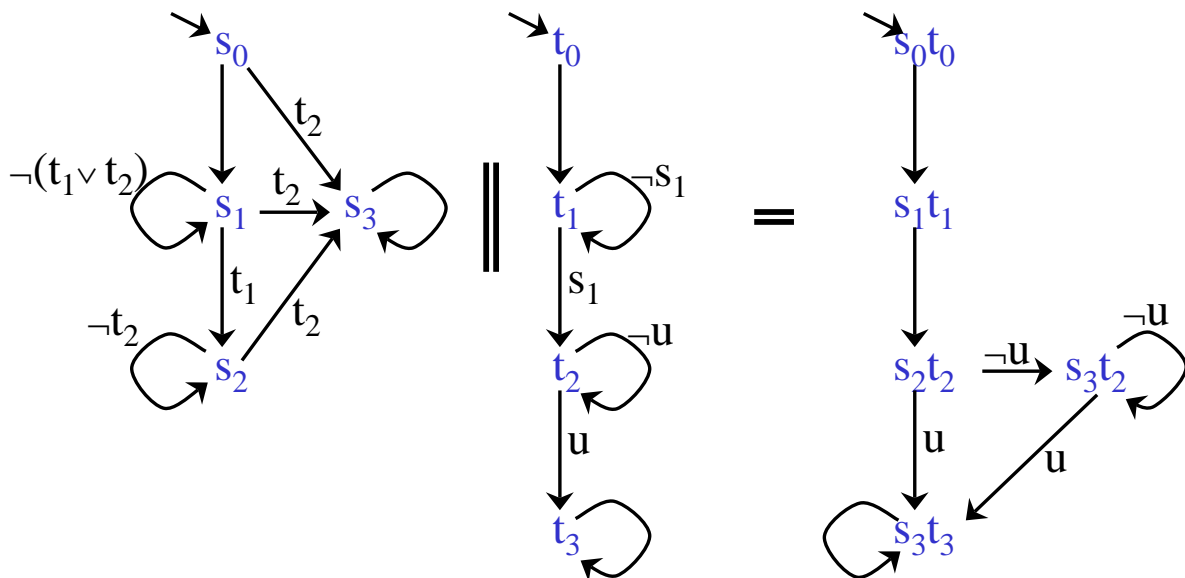
```

```

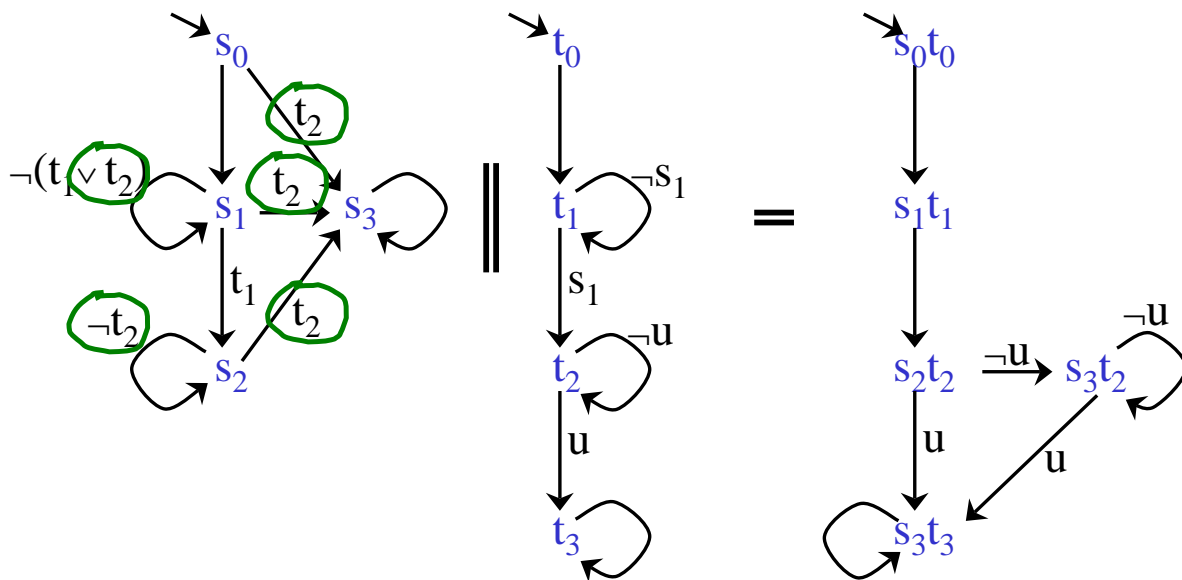
if (NONDET) {
  (i++, b[j++] = (k++, k));
}

```

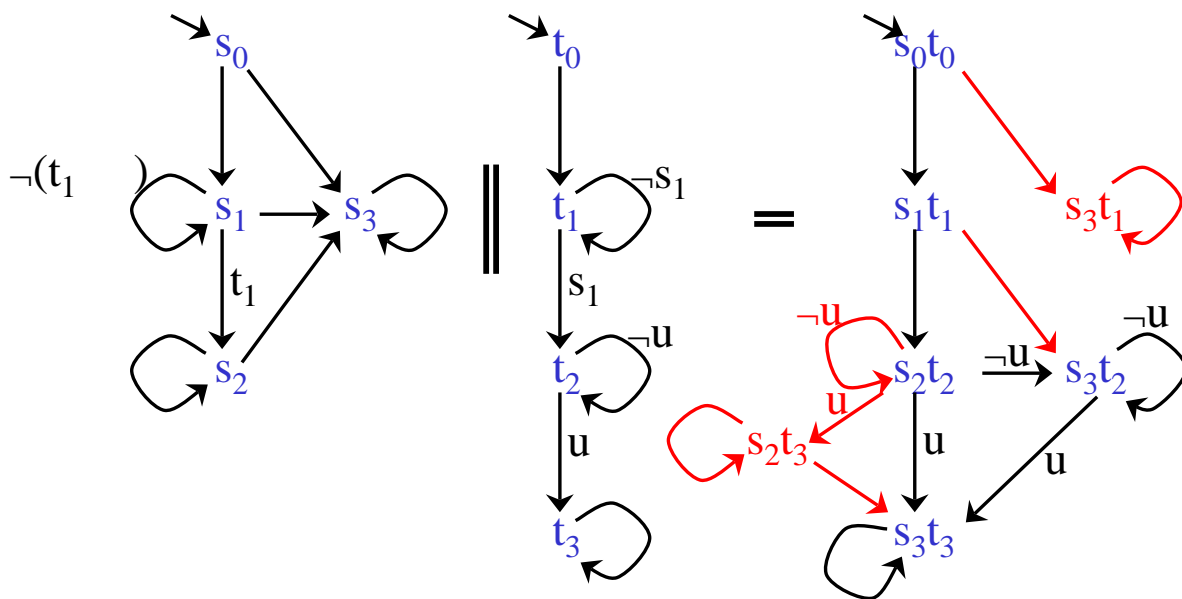
Interaction Loosening: Example



Interaction Loosening: Example



Interaction Loosening: Example



Interaction Loosening: Facts

- = manually specifying $P \rightarrow M$ by giving synchronizations to loosen; effort is in finding those
- weak preservation (tunable)
- successful in BDD-based MC

Interaction Loosening: Experim.

- StateCharts model of production cell (robot arm, press, feed belts, crane)
- 18 properties (univ., exist., mixed; safety, progress)
- 94% reduction on average (max. # BDD nodes)

Predicate Abstraction: Example

$n : \{e, o\}$ $b_{\text{even}(n)} : \text{Bool}$
 $b_{\text{odd}(n)} : \text{Bool}$

$n : \{e, o, 100\}$ $b_{\text{even}(n)} : \text{Bool}$
 $b_{\text{odd}(n)} : \text{Bool}$
 $b_{n=100} : \text{Bool}$

Predicate Abstraction: Definition

Given: predicates $\varphi_1, \dots, \varphi_k$ on concrete states.

Let b_1, \dots, b_k be variables of type $\{\text{tt}, \text{ff}, \top\}$.

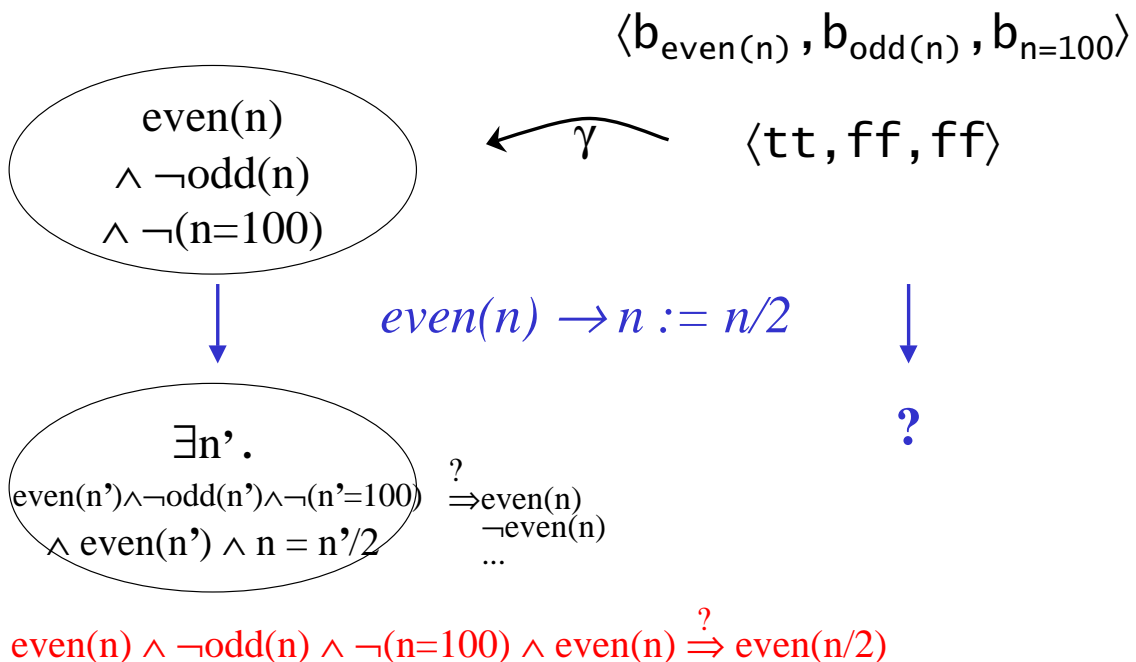
An abstract state is a valuation of b_1, \dots, b_k
or false.

I.e. an abstract state is (false or) a *monomial*
on b_1, \dots, b_k : a conjunction of b_j and $\neg b_j$
containing every b_j at most once.

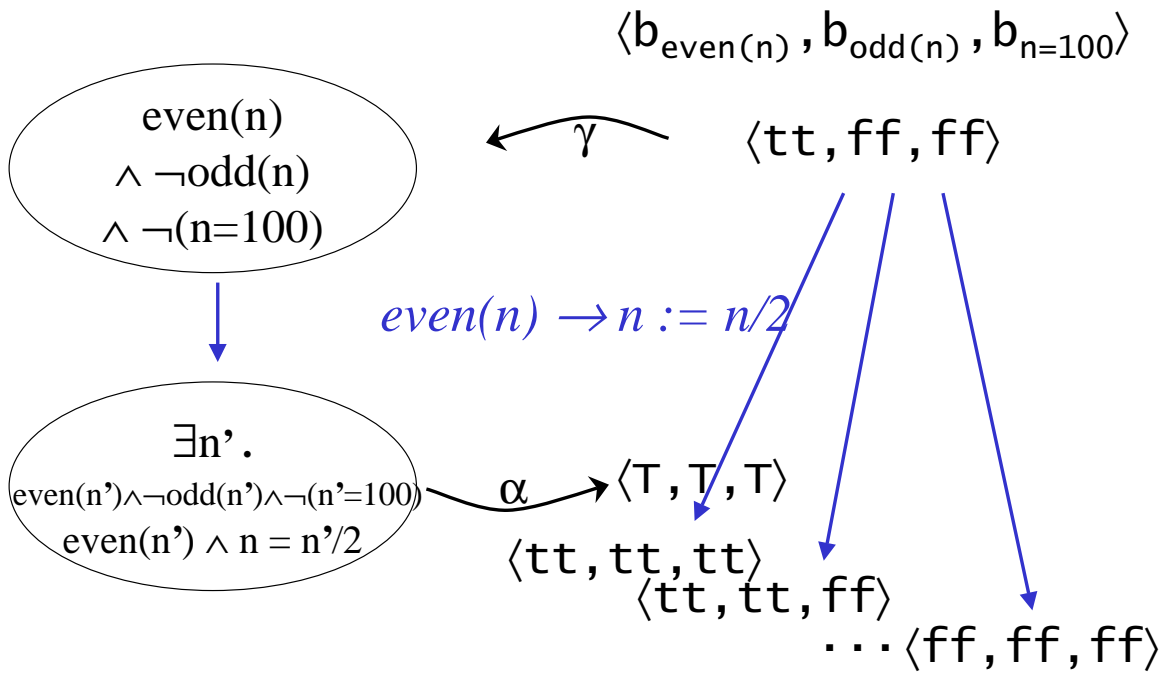
PredAbs: Further Characteristics

- use of decision procedure / thm. prover / simplifier + approximation to keep fully automatic
- abs. state always split into canonical monomials (every b_j occurs exactly once)
- method for refinement of predicates if too coarse

Using a Theorem Prover

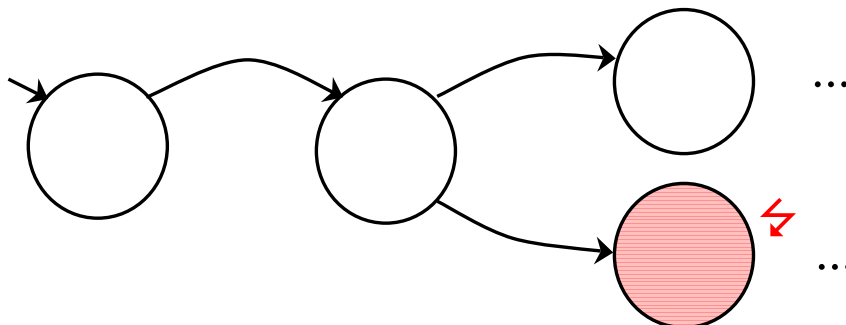


Splitting into Canon. Monomials



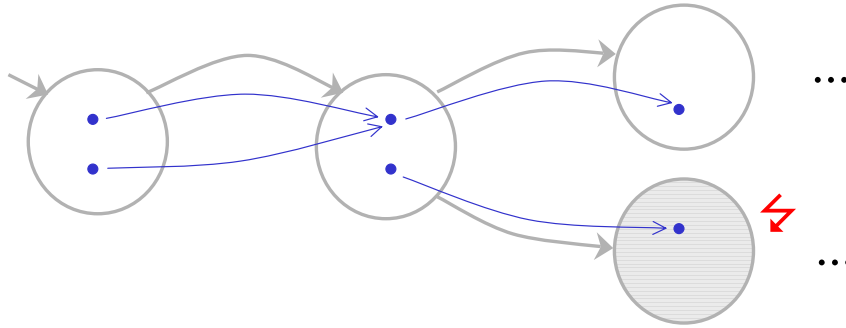
Finding Predicates

- Initial predicates:
 - state predicates from correctness property +
 - conditions from program
- Counter-example driven refinement:



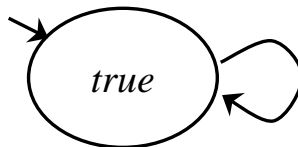
Finding Predicates

- Initial predicates:
 - state predicates from correctness property +
 - conditions from program
- Counter-example driven refinement:



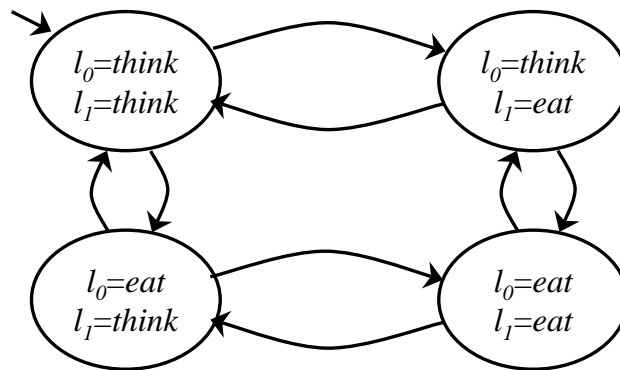
Formula-Driven Partition Refin.

$$\forall G(l_0 = eat \rightarrow \forall F l_1 = eat)$$



Formula-Driven Partition Refin.

$$\forall G(l_0 = \text{eat} \rightarrow \forall F l_1 = \text{eat})$$

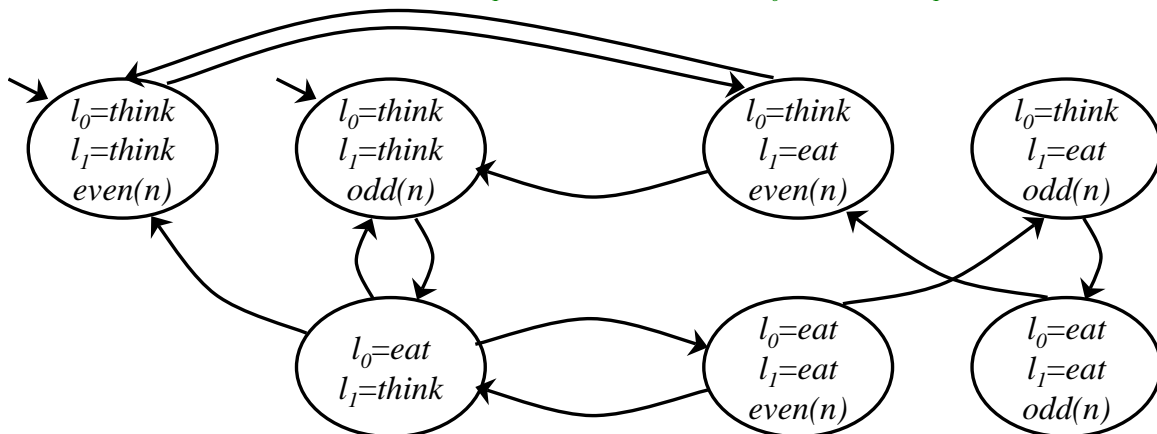


Formula-Driven Partition Refin.

$$\forall G(l_0 = \text{eat} \rightarrow \forall F l_1 = \text{eat})$$

$$\hookrightarrow l_1 = \text{eat} \vee \forall X l_1 = \text{eat} \vee \forall X \forall X l_1 = \text{eat} \vee \dots$$

$$(l_1 = \text{eat} \wedge \text{odd}(n)) \vee (l_0 = \text{think} \wedge l_1 = \text{think} \wedge \text{even}(n))$$



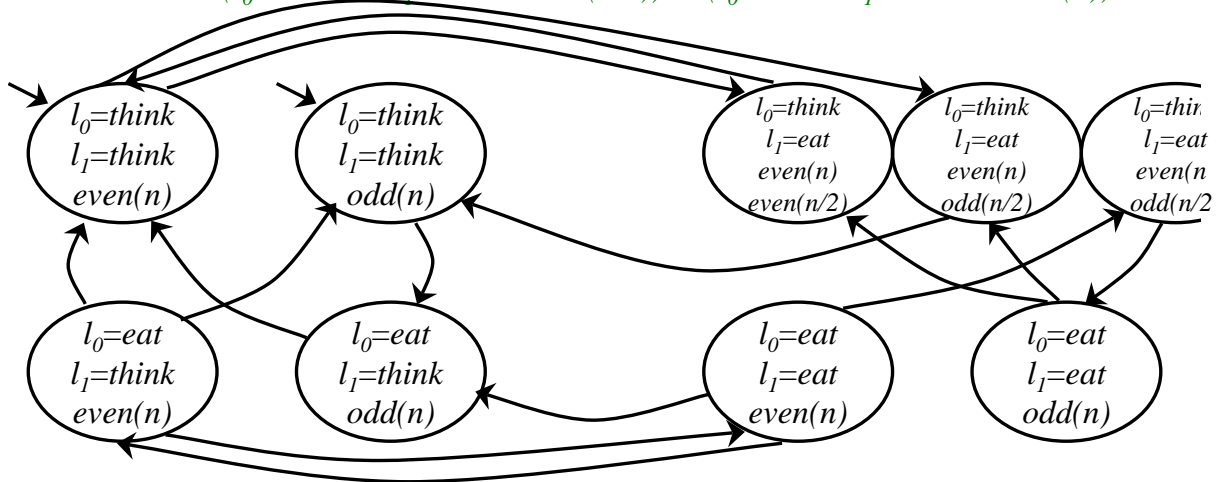
Formula-Driven Partition Refin.

$$\forall G(l_0 = eat \rightarrow \forall F l_1 = eat)$$



$$\dots \vee \forall X \forall X \forall X l_1 = eat \vee \dots$$

$$\dots \vee (l_0 = think \wedge l_1 = eat \wedge even(n/2)) \vee (l_0 = eat \wedge l_1 = think \wedge odd(n))$$



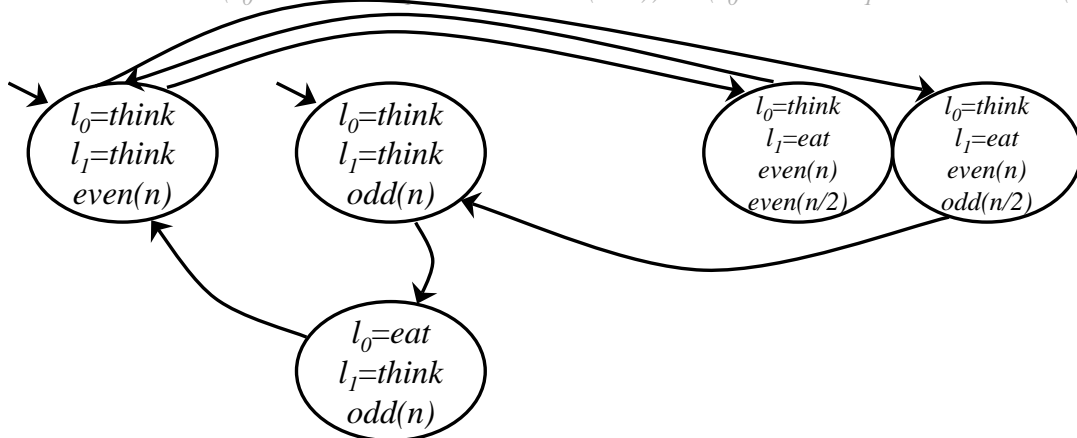
Formula-Driven Partition Refin.

$$\forall G(l_0 = eat \rightarrow \forall F l_1 = eat) \quad \checkmark$$



$$\dots \vee \forall X \forall X \forall X l_1 = eat \vee \dots$$

$$\dots \vee (l_0 = think \wedge l_1 = eat \wedge even(n/2)) \vee (l_0 = eat \wedge l_1 = think \wedge odd(n))$$



Predicate Abstraction: Tools

- AUTOABS (Bell Labs / Cadence)
- Bandera?
- BLAST (Berkeley)
- InVeSt (Verimag)
- JPF2 (NASA)
- SLAM (Microsoft)

Summarizing

- abstract interpretation
- slicing
- variable hiding
- interaction loosening
- predicate abstraction
 - automatic refinement

Organizing

- abstraction of
 - data (“*values and the operations on them*”)
 - control (“*how the operations are put together in a process*”)
 - configuration (“*how the processes are put together in a program*”)
 - communication
- weak vs. strong preservation
- degree of sophistication
- degree of automation

Outline

PART I

- Introduction / Methodology
- Theory

PART II

- Techniques & Algorithms
- Tools

▶ (PART III)

- Challenges

There is an old story of a boilermaker who was hired to fix a huge steamship boiler system that was not working well. After listening to the engineer's description of the problems and asking a few questions, he went to the boiler room. He looked at the maze of twisting pipes, listened to the thump of the boiler and the hiss of escaping steam for a few minutes, and felt some pipes with his hands. Then he hummed softly to himself, reached into his overalls and took out a small hammer, and tapped a bright red valve, once. Immediately the entire system began working perfectly, and the boilermaker went home. When the steamship owner received a bill for \$1,000 he complained that the boilermaker had only been in the engine room for fifteen minutes, and he would not pay the bill. As the boiler maker said:

Knowing where to tap

For tapping with hammer:	.50
For knowing where to tap:	\$999.50
Total:	<u>\$1,000.00</u>

RICHARD BANDLER AND JOHN GRINDER
Frogs into Princes: Neuro Linguistic Programming, 1979

Practice

- Compare approaches (tool efforts);
reproducible experiments; open tools and
case studies
- Can MC be integrated in SW development?
(SW developers will have to produce more
formal correctness requirements)



Copyright © 1995 David Farley, d-farley@teecat.com
Distributed by United Feature Syndicate
<http://www.unitedmedia.com>
This cartoon is made available on the Internet for personal viewing only.

No software is complete without the giant 800-page book that always costs \$49.95.

Tool Building

- Pareto Principle (80-20 rule)
- Seek Simplicity
- Technology push alone will not do

