Abstraction in Software Model Checking

Principles and Practice

Dennis Dams
Bell Labs & TU/e

Outline

PART I
- Introduction / Methodology
- Theory

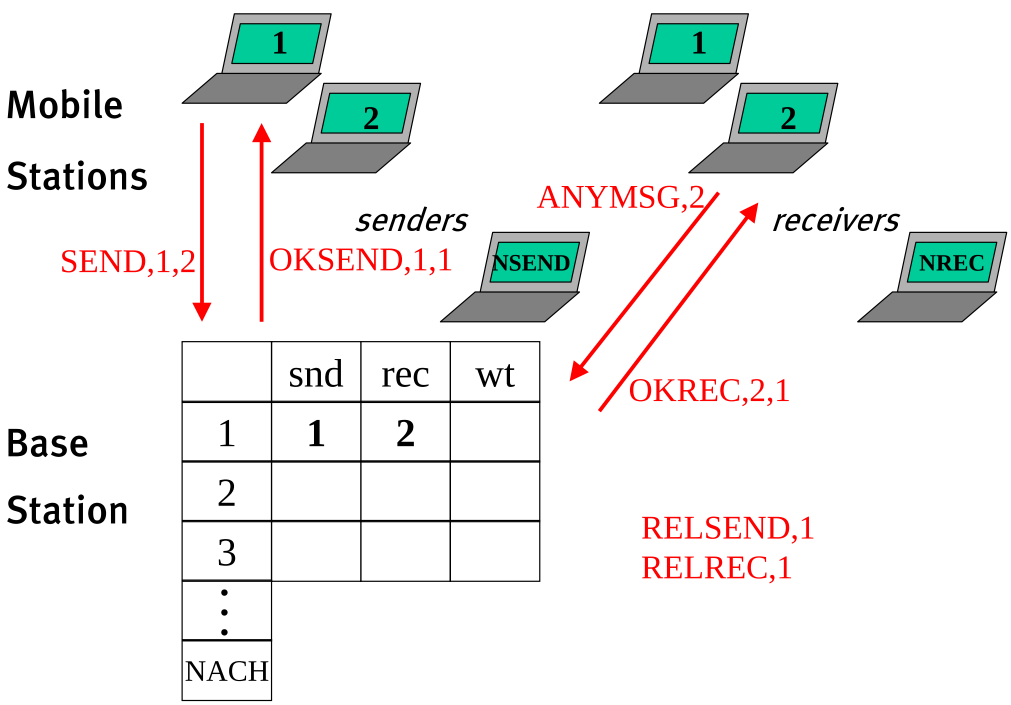
PART II
- Techniques & Algorithms
- Tools

(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,..,r,..) has been inserted in the Base Station’s table.” (ψ)

$\forall$NSEND, NREC, NACH:Nat

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

Verifying a Concrete Instance

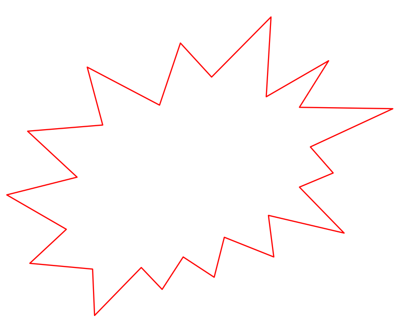
For NSEND=3, NREC=3, NACH=2,
r=2, c=1: $\quad \neg((\neg\varphi)\ \mathrm{U}\ \psi)$

But:

Would have to check several “representative” instances,

and argue why that suffices.

And:



Step 1: Data Abstraction

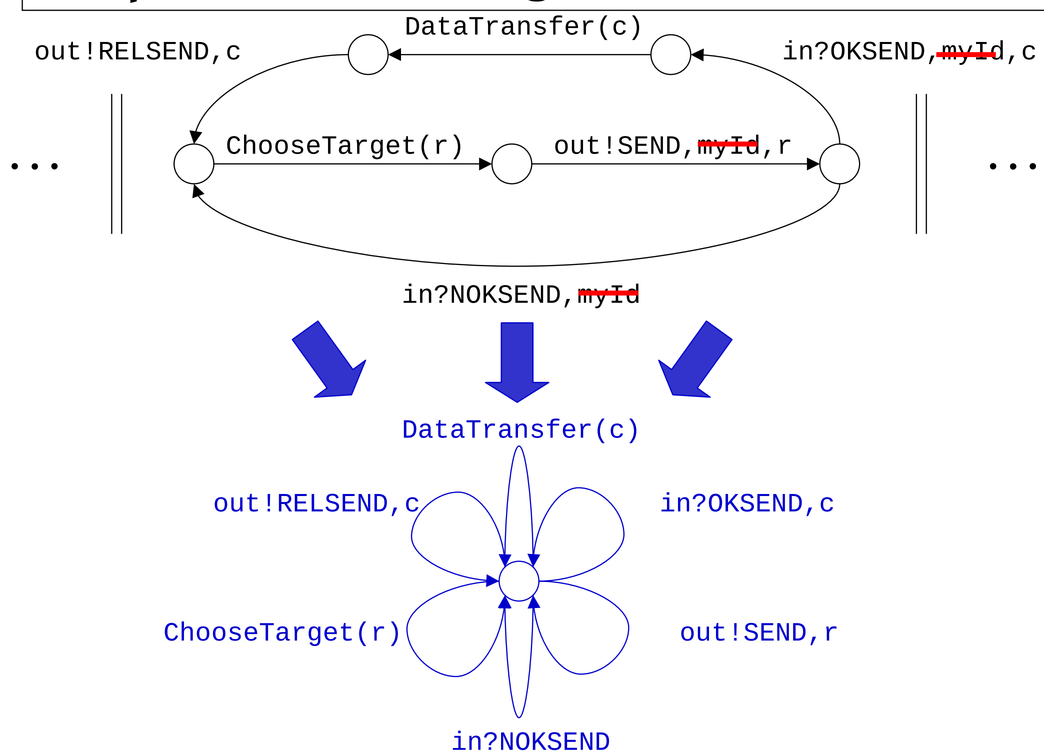
typedef MesgType = {SEND, OKSEND, ANYMSG, …};

~~SenderId = 1..NSEND;~~

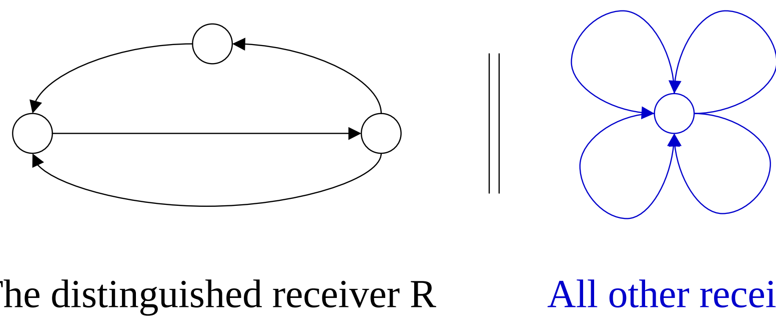
ReceivId = ~~1..NREC;~~ {R, NR};

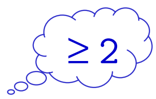
ChanIndx = ~~0..NACH;~~ {C, NC};

Step 2: Abstracting senders' control



Step 3: Abstracting away the NR receivers

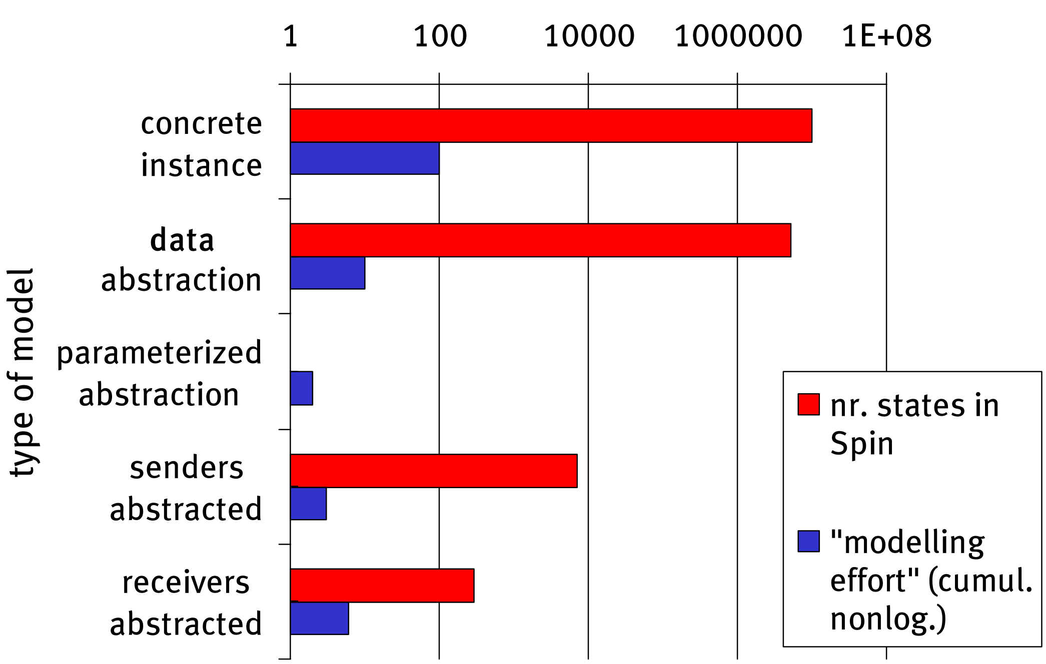




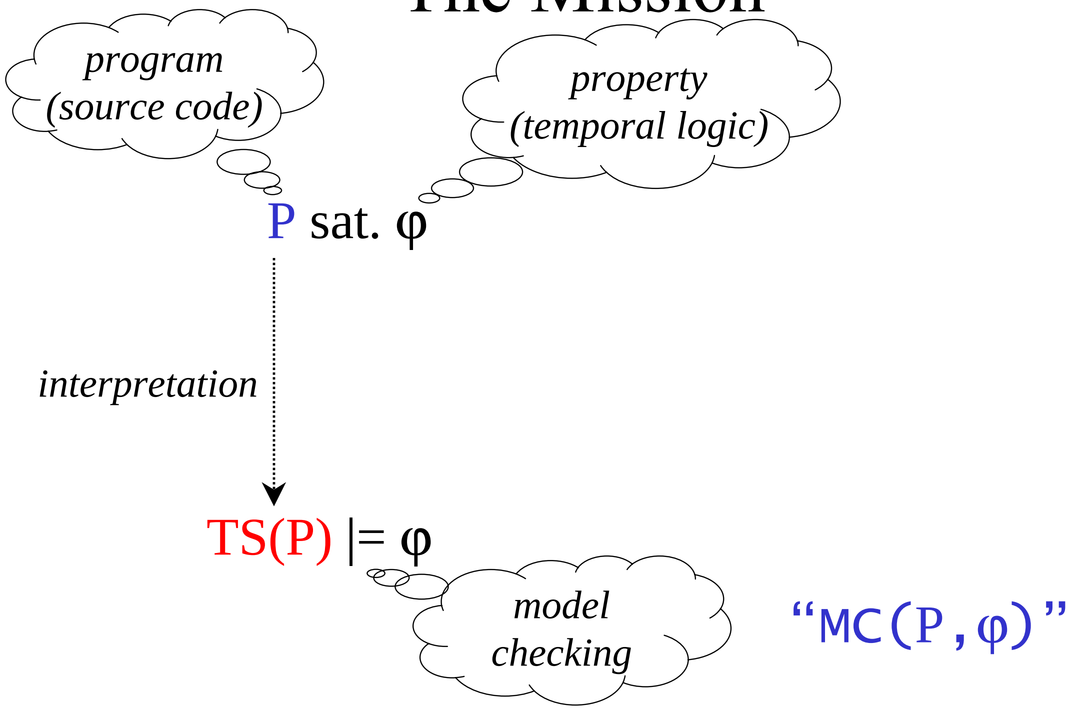
$\forall$NSEND, NREC, NACH:Nat

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

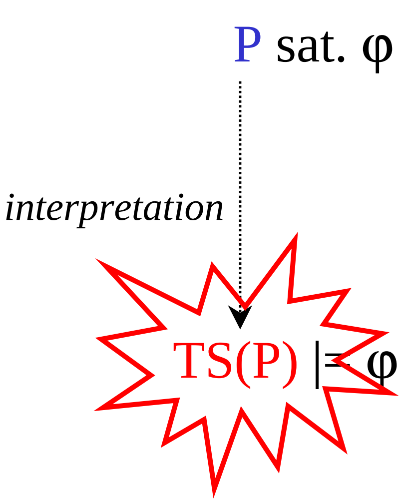
Verification results



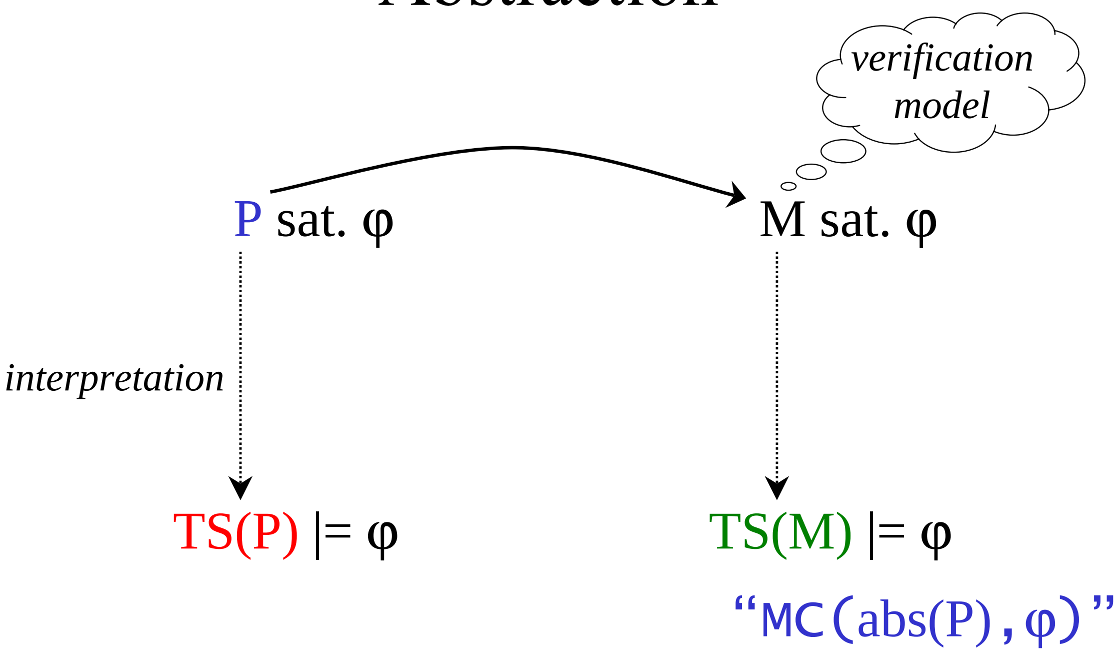
The Mission



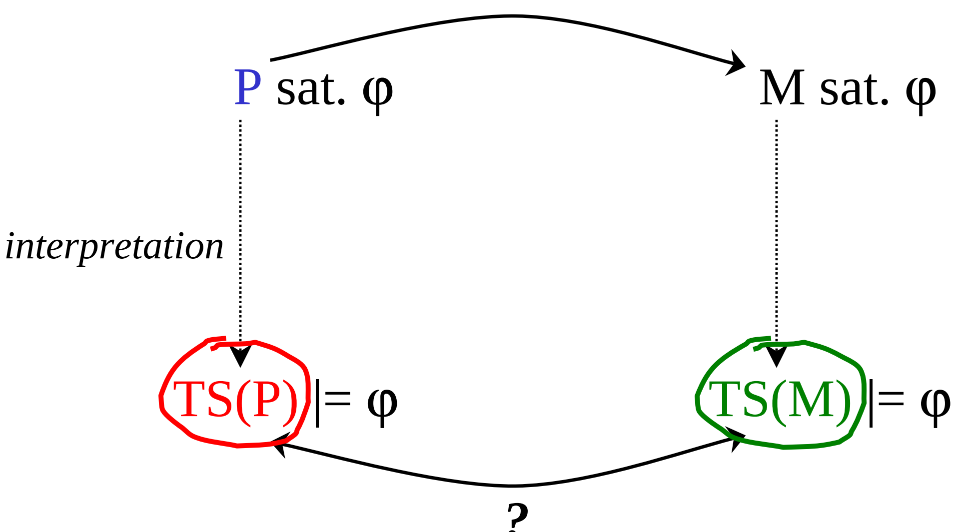
The Problem



Abstraction



Abstraction Relation



Property Preservation: A Choice

- ⇔ "strong preservation"
- ⇐ "weak preservation with false negatives"
- ⇒ "weak preservation with false positives"

$$TS(P) \models \varphi \quad \begin{matrix}\Leftrightarrow\\ \Leftarrow\\ \Rightarrow\end{matrix} \quad TS(M) \models \varphi$$

Property Preservation: A Choice

- ⇔ "strong preservation"
- ⇐ "weak preservation with false negatives"
- ⇒ "weak preservation with false positives"

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

Strong Preservation

– Puts lower bound on size of suitable abstractions (*e.g. bisim. reduction*)

– Difficult to construct abstraction in general

$$TS(P) \models \varphi \quad \Leftrightarrow \quad TS(M) \models \varphi$$

Weak Preservation

+ No lower bound and easy to construct

– "Incomplete proof method"

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

OR

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

Usually: *iterative refinement until strong preservation*

Weak Preservation

+ No lower bound and easy to construct

– "Incomplete proof method"

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

OR

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

Usually: *iterative refinement until strong preservation*

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

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

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 (∈ LTL / ∀CTL* / µcalc): counterexamples can be dealt with one-by-one, while evidence needs to be considered as a whole


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

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

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

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

Outline

PART I

- Introduction / Methodology
- Theory

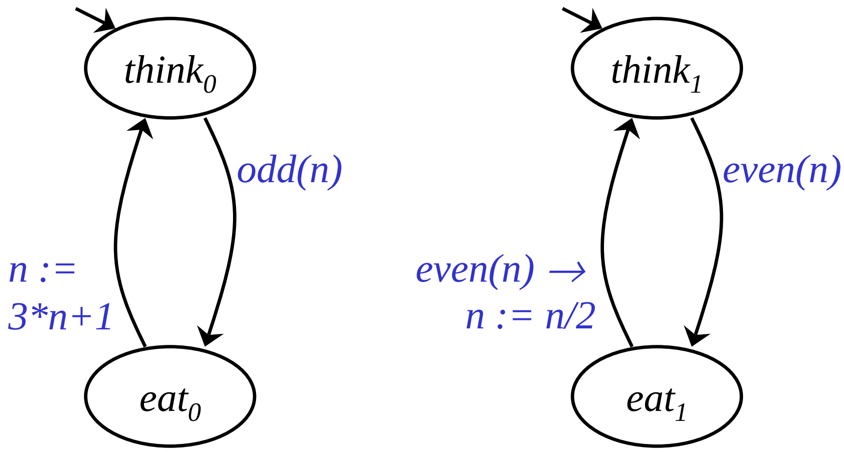
PART II

- Techniques & Algorithms
- Tools

(PART III)

- Challenges

Running Example



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

Kripke structures: “Statics”

$\longrightarrow$ Predicates p over states

e.g.

$\langle think, think, 7\rangle$	$\boldsymbol{p = 0}$
$\langle eat, think, 7\rangle$	$\boldsymbol{p = 0}$
$\langle think, think, 22\rangle$	$\boldsymbol{p = 1}$

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

Kripke structures: “Dynamics”

$\longrightarrow$ Relation R between states

$$\begin{tikzcd} \langle think, think, 7\rangle \arrow[d, "(R = 1)"] \\ \langle eat, think, 7\rangle \arrow[d, "(R = 1)"] \\ \langle think, think, 22\rangle \end{tikzcd}$$

Temporal Logic (CTL*)

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

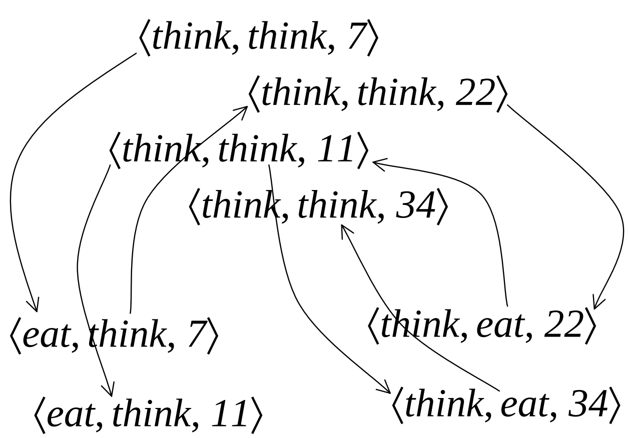
$$\forall \mathbf{G} \neg (l_0 = eat \wedge l_1 = eat)$$

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

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

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

Abstracting Kripke structures



Abstraction =

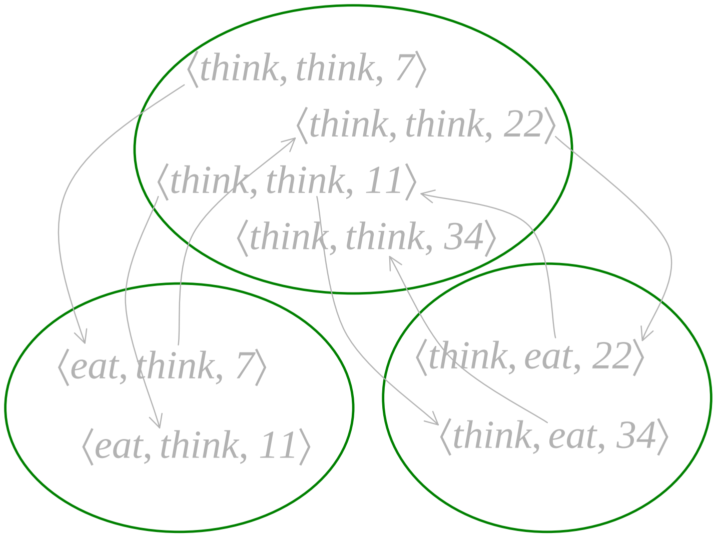
partitioning of states into *abstract states*

For now, assume

α : states → abs. states

$\alpha(c) = a$: "$c \in a$"

Abstracting Kripke structures



Abstraction =

partitioning of states into *abstract states*

For now, assume

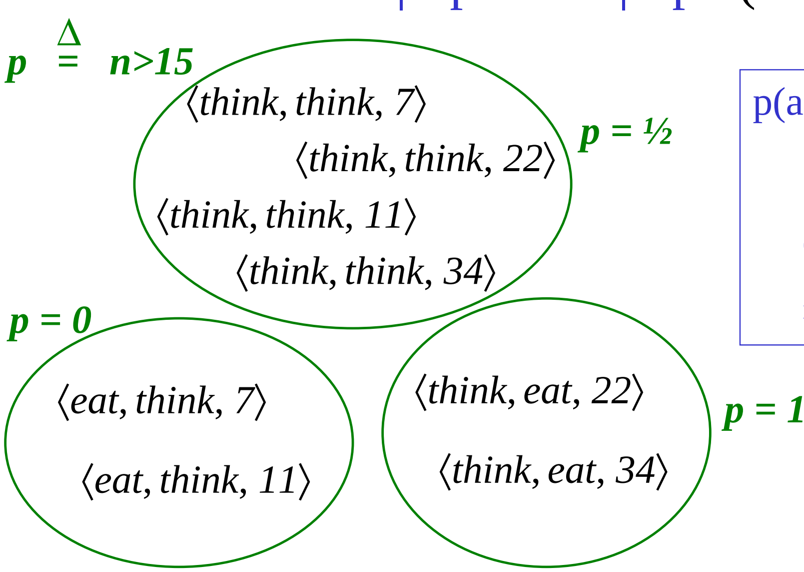
α : states $\rightarrow$ abs. states

$\alpha(c) = a$: "$c \in a$"

$$C \models \varphi \quad \Leftarrow \quad A \models \varphi$$

Abs. Kripke structs: Statics

For $c \in a$: $c \models p \ \Leftarrow\ a \models p$ (for all $p \in$ Prop)

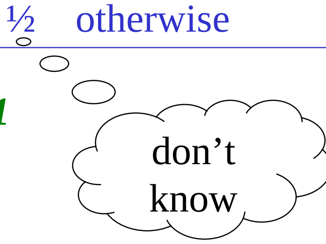


p(a) =

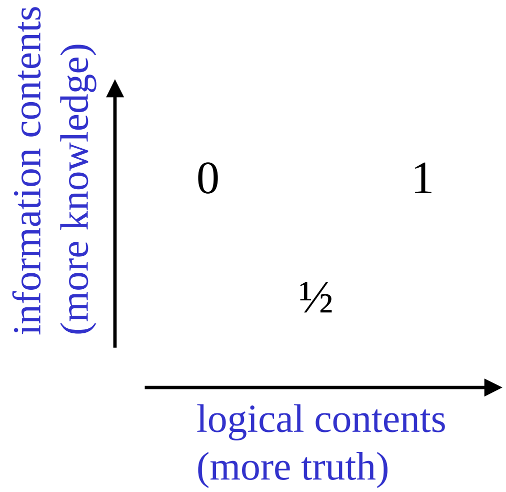
1 if $\forall c \in a$. p(c) = 1

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

½ otherwise



Intermezzo: 3-valued logic



Kleene interpretation:

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

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

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

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

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

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

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

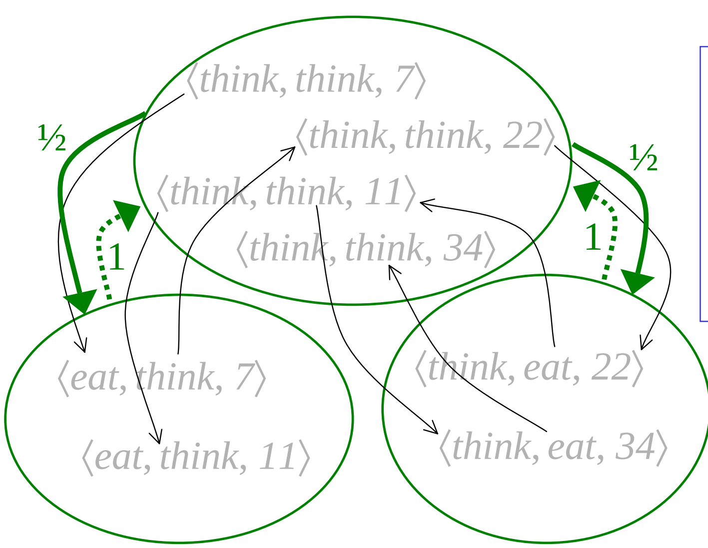
$0 \rightarrow \frac{1}{2} = \frac{1}{2} \rightarrow 1 = 1$

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

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

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

½ otherwise

Preservation Theorem

For all $\varphi \in$ CTL* :

- if $a \models_3 \varphi = 1$, then $c \models \varphi$
- if $a \models_3 \varphi = 0$, then $c \not\models \varphi$
- (if $a \models_3 \varphi = ½$, then $c \models \varphi$ or $c \not\models \varphi$)

abstract,
3-valued
world

concrete,
2-valued
world

Another way to look at $\models_3 \varphi$

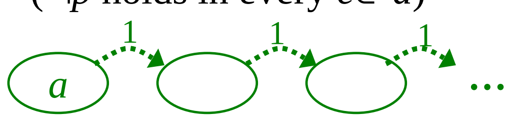
Define 2-valued |= on abstract side for pnf as follows:

$a \models p$ iff 

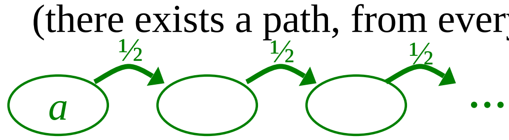
(p holds in every $c \in a$)

$a \models \neg p$ iff 

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

$a \models \exists$ ... iff 

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

$a \models \forall$ ... iff 

(there may exist a path, from some $c \in a$)

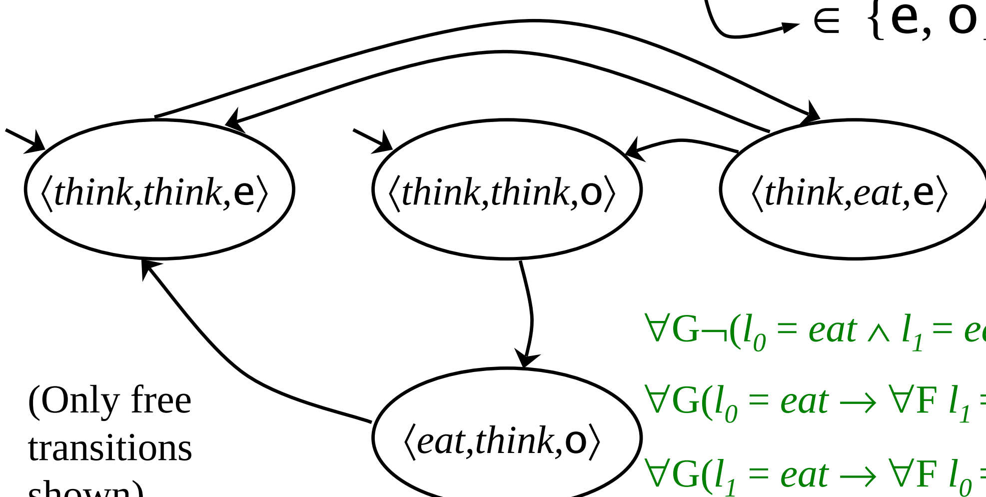
Model Checking on Abstract Kripke Structures

1. Bring φ and ¬φ 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 $\alpha(\langle l_0, l_1, n\rangle) = \langle l_0, l_1, parity(n)\rangle$

$\in \{\mathsf{e}, \mathsf{o}\}$



(Only free transitions shown)

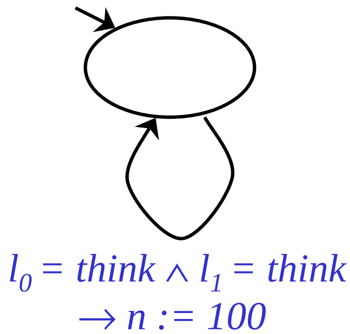
$\forall G \neg(l_0 = eat \wedge l_1 = eat)$ ✓

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

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

An Existential Property

Add a “restart” process:

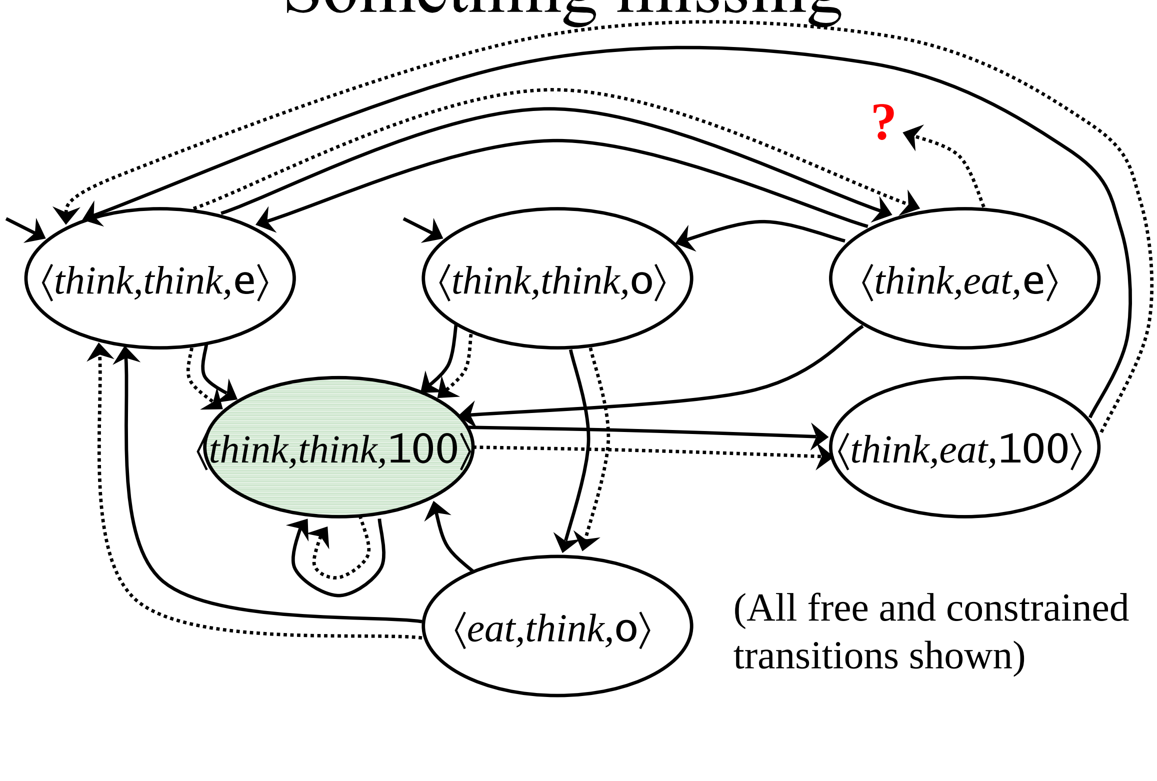


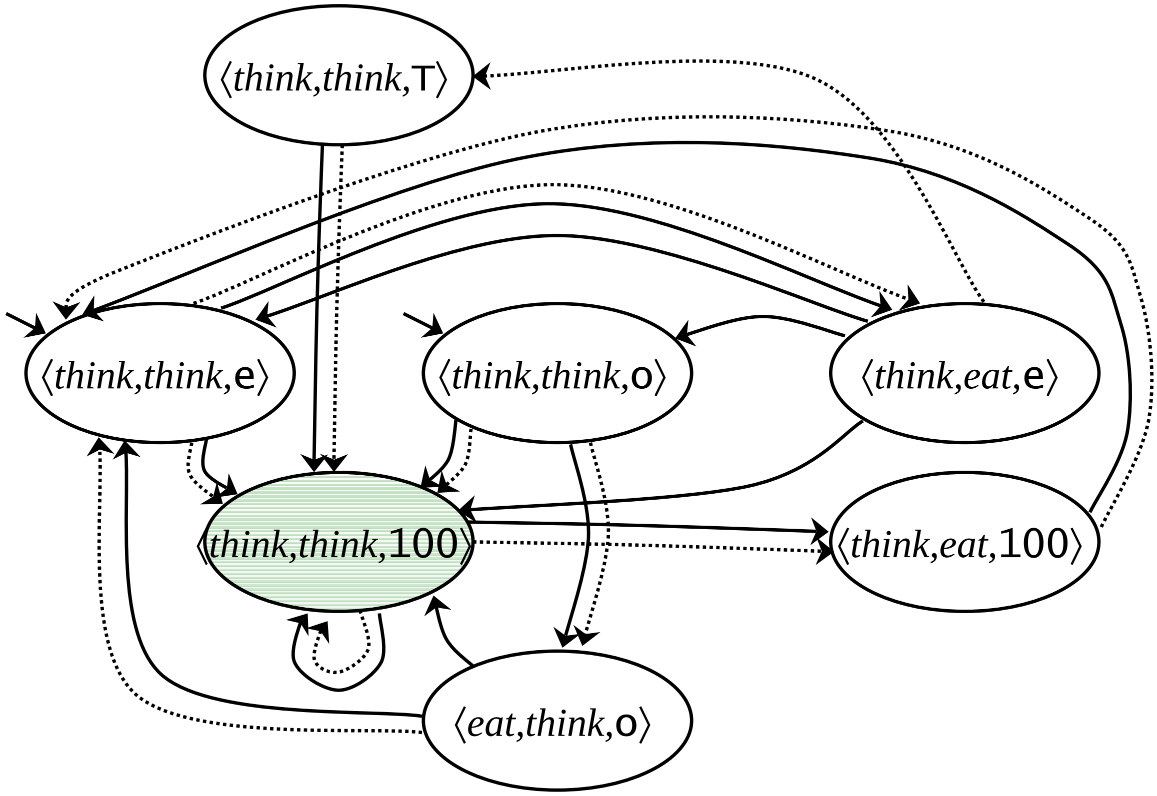
∀G∃F “restart” ?

$\alpha(\langle l_0, l_1, n\rangle) = \langle l_0, l_1, parity'(n)\rangle$

$\in \{\mathtt{e}, \mathtt{o}, \mathtt{100}\}$

Something missing





Galois Connection Framework

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

$\gamma(a)$ = *the set of all states described by a*

$\gamma(a') \subset \gamma(a)$: *a' is more precise than a*

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

$\alpha(S)$ = *the most precise description of S*

Outline

PART I

- Introduction / Methodology
- Theory

▸ PART II

- Techniques & Algorithms
- Tools

(PART III)

- Challenges

Formal Abstraction

$$
\begin{tikzcd}
P \text{ sat. } \varphi \arrow[r, bend left, "\textit{syntactic abstraction relation}"] \arrow[d, dotted, "\textit{interpretation}"'] & M \text{ sat. } \varphi \arrow[d, dotted] \\
TS(P) \models \varphi \arrow[r, leftrightarrow, bend right, "\textit{semantic abstraction relation}"'] & TS(M) \models \varphi
\end{tikzcd}
$$

Issues

- Which techniques/algorithms can be used to construct M = abs(P) ?
- How to find a suitable abstraction abs ?

  (given a program and correctness property)

Abstract Interpretation: Example

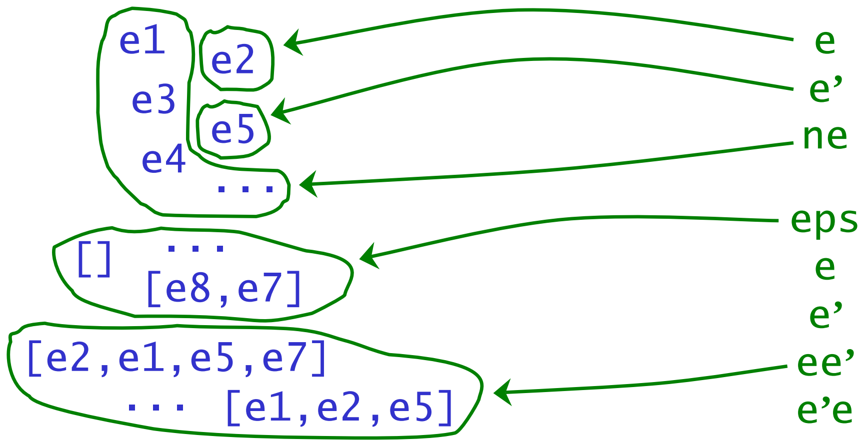
```
type El = {e1,e2,e3,
  e4,e5,...}
type Li = listof El

fun head : Li->El =
  ...
fun tail : Li->Li =
  ...
fun one_el:Li->Bool
```

```
type El = {e,e',ne}
type Li = {eps,e,e',
  ee',e'e}

fun head : Li->El =
  ...
fun tail : Li->Li =
  ...
fun one_el:Li->Bool
```

Abstract Interpretation: Example



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

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

one_el([])=false

one_el(eps)=
NONDET(true,false)

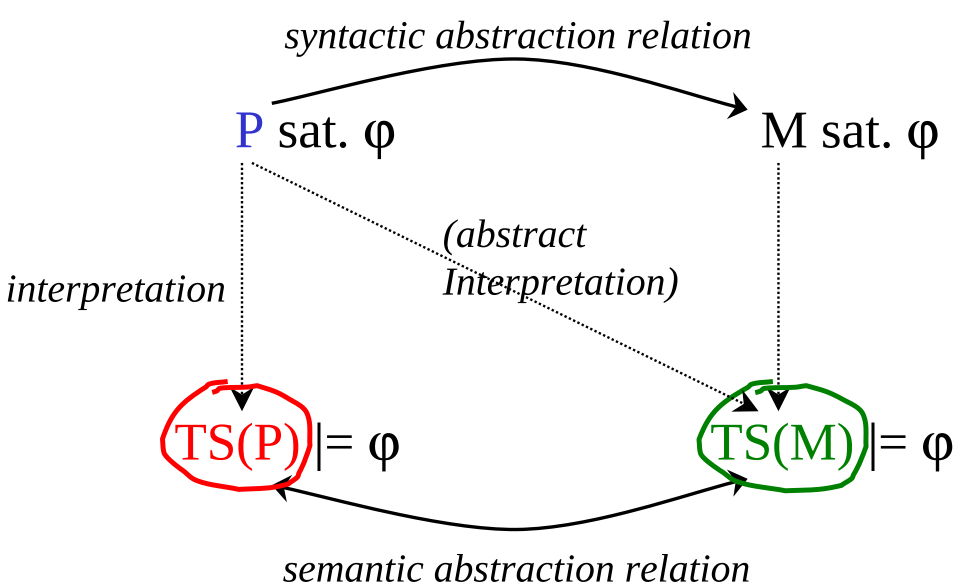
Abs. Int. + M.C.: Tools

- αSPIN (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 $\longrightarrow$ 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 $\rightarrow$ 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 $\longrightarrow$ M automatic (given hiding crit.)
- smaller state vectors, hopefully smaller state space

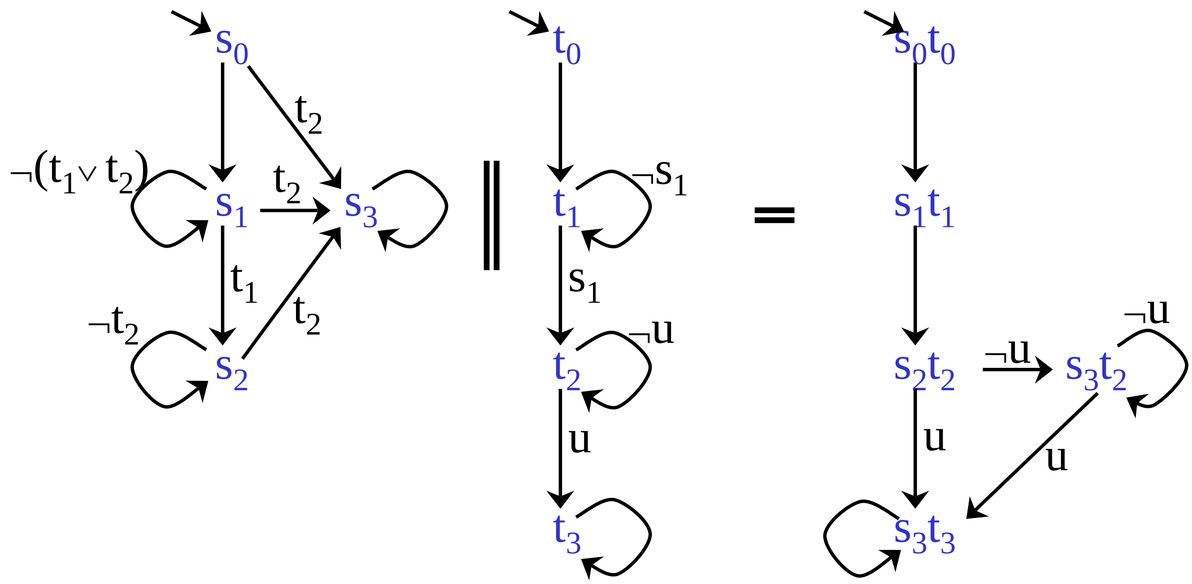
Transforming C

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

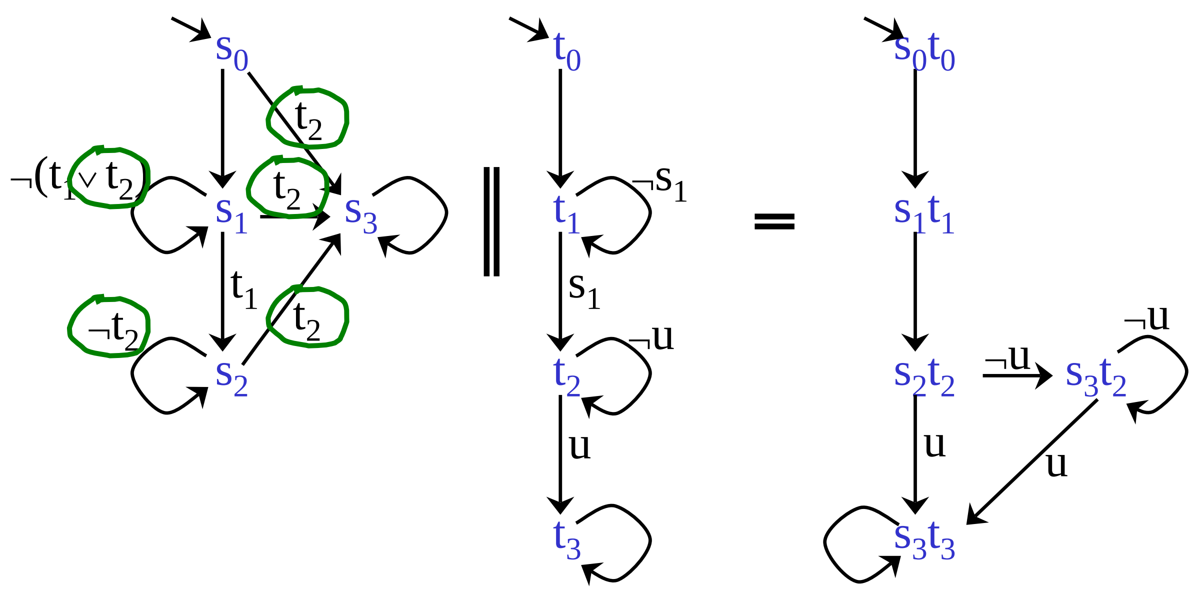
```
if (NONDET) {

  (i++,     b[j++] = (    k++, k));
}
```

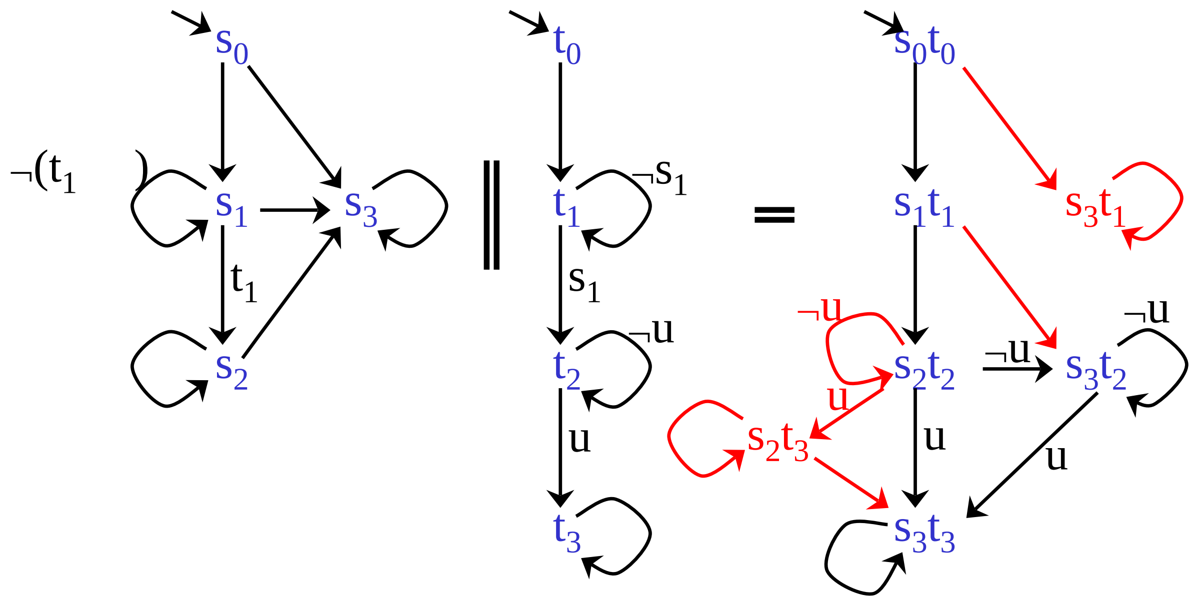
Interaction Loosening: Example



Interaction Loosening: Example



Interaction Loosening: Example



Interaction Loosening: Facts

- = manually specifying P $\longrightarrow$ 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}`	$b_{even(n)}$ `: Bool` $b_{odd(n)}$ `: Bool`
`n : {e,o,100}`	$b_{even(n)}$ `: Bool` $b_{odd(n)}$ `: Bool` $b_{n=100}$ `: Bool`

Predicate Abstraction: Definition

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

Let $\mathtt{b}_1,\ldots,\mathtt{b}_k$ be variables of type $\{\mathtt{tt},\mathtt{ff},\top\}$.

An abstract state is a valuation of $\mathtt{b}_1,\ldots,\mathtt{b}_k$ or `false`.

I.e. an abstract state is (`false` or) a *monomial* on $\mathtt{b}_1,\ldots,\mathtt{b}_k$: a conjunction of $\mathtt{b}_j$ and $\neg\mathtt{b}_j$ containing every $\mathtt{b}_j$ 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 $\mathtt{b}_j$ occurs exactly once)
- method for refinement of predicates if too coarse

Using a Theorem Prover

$\langle b_{even(n)}, b_{odd(n)}, b_{n=100} \rangle$

even(n) $\wedge \neg$odd(n) $\wedge \neg$(n=100) $\xleftarrow{\gamma}$ $\langle tt, ff, ff \rangle$

even(n) $\rightarrow$ n := n/2

?

$\exists$n'. even(n')$\wedge\neg$odd(n')$\wedge\neg$(n'=100) $\wedge$ even(n') $\wedge$ n = n'/2 $\overset{?}{\Rightarrow}$ even(n), $\neg$even(n), ...

even(n) $\wedge \neg$odd(n) $\wedge \neg$(n=100) $\wedge$ even(n) $\overset{?}{\Rightarrow}$ even(n/2)

Splitting into Canon. Monomials

$\langle b_{even(n)}, b_{odd(n)}, b_{n=100} \rangle$

even(n) $\wedge \neg$odd(n) $\wedge \neg$(n=100) $\xleftarrow{\gamma}$ $\langle tt, ff, ff \rangle$

even(n) $\rightarrow$ n := n/2

$\exists$n'. even(n')$\wedge\neg$odd(n')$\wedge\neg$(n'=100) even(n') $\wedge$ n = n'/2 $\xrightarrow{\alpha}$ $\langle T, T, T \rangle$

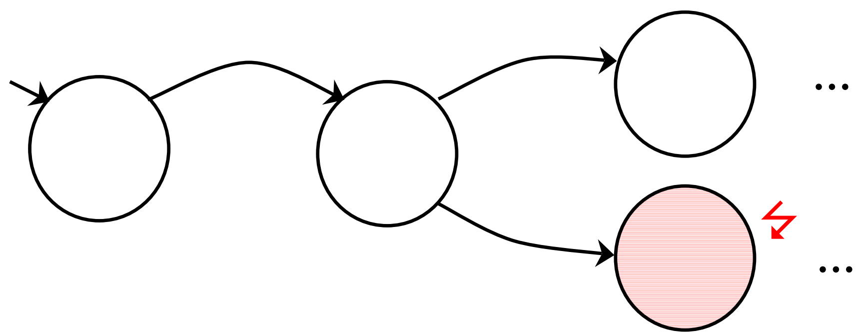
$\langle tt, tt, tt \rangle$

$\langle tt, tt, ff \rangle$

$\cdots \langle ff, ff, ff \rangle$

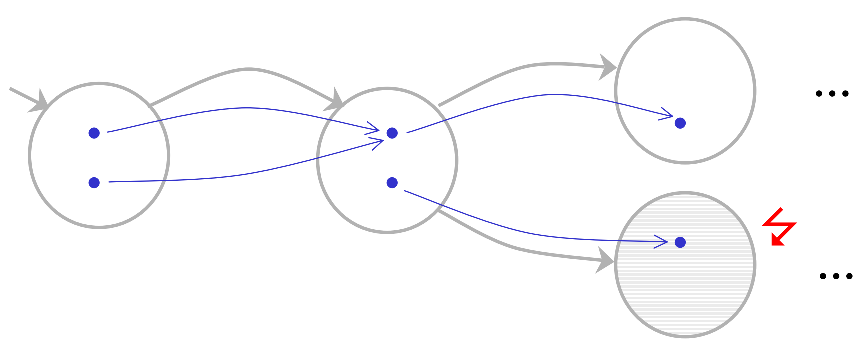
Finding Predicates

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



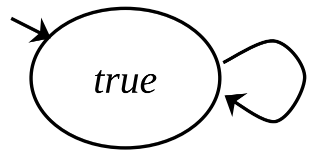
Finding Predicates

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



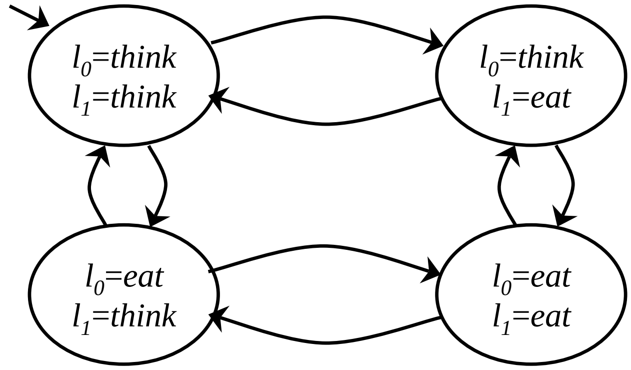
Formula-Driven Partition Refin.

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



Formula-Driven Partition Refin.

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

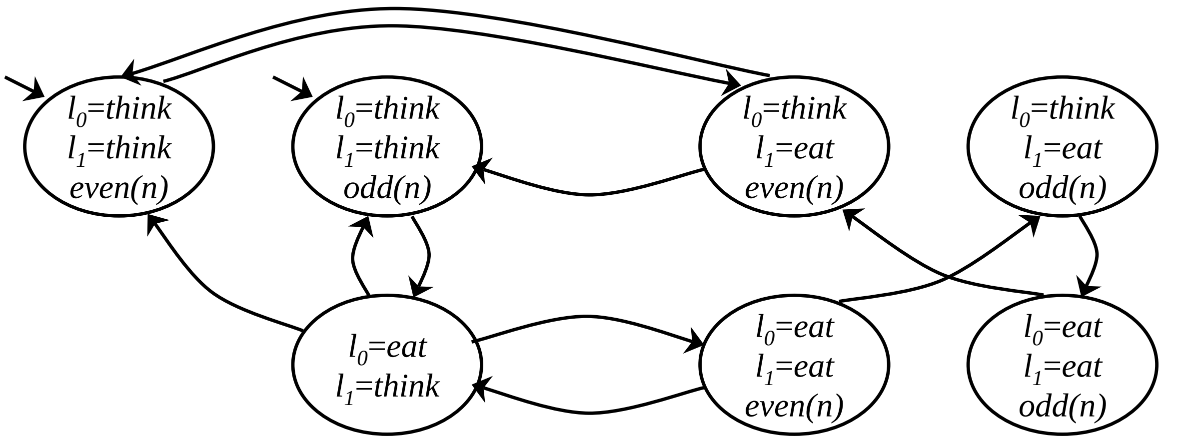


Formula-Driven Partition Refin.

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

$l_1 = eat \ \vee\ \forall X\ l_1 = eat \ \vee\ \underline{\forall X \forall X\ l_1 = eat} \ \vee\ \ldots$

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

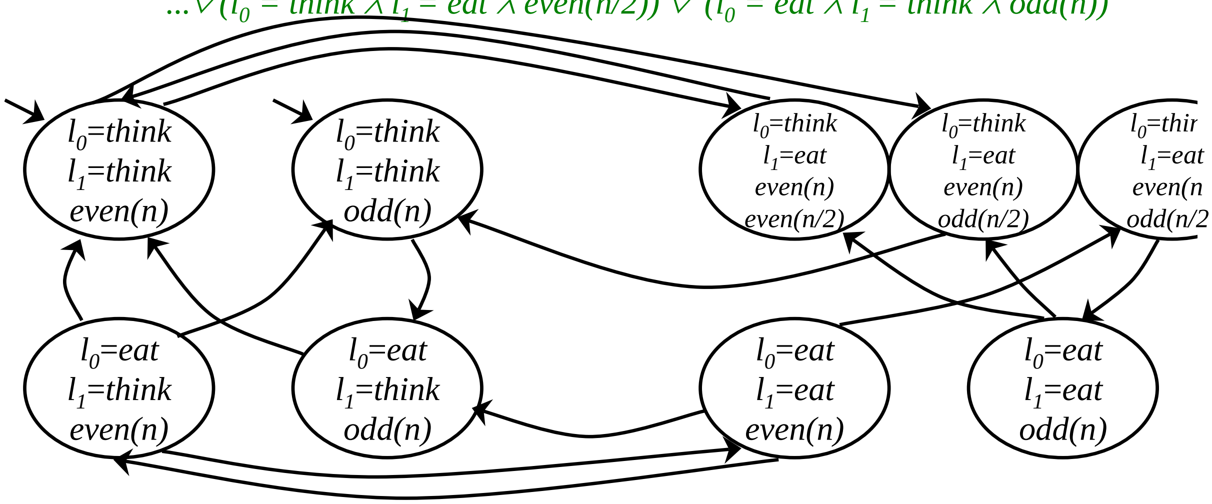


Formula-Driven Partition Refin.

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

$\ldots \vee\ \underline{\forall X \forall X \forall X\ l_1 = eat} \ \vee\ \ldots$

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

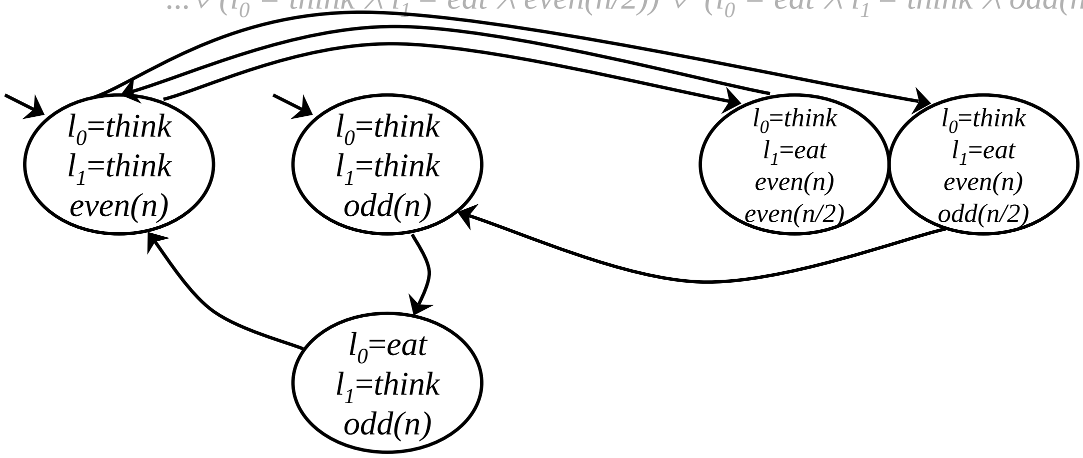


Formula-Driven Partition Refin.

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

$\ldots \vee \underline{\forall X \forall X \forall X\ l_1 = eat} \vee \ldots$

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



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

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)

Tool Building

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

Principles

- "Refinement" in presence of free and constrained relations
- Completeness result for branching time