BRICS Qualifying Exam

Automation for Formal Verification: Exploiting Structure

M. Oliver Möller

Supervision: Michael I. Schwartzbach / Kim Guldstrand Larsen

Roadmap

Part I Automation in theorem proving: congruence closure

Part II Automation in model checking: hierarchical decomposition

Part III Automation in concurrency: identifying togetherness

Roadmap

Part I Automation in theorem proving: congruence closure

- ⋆ congruence closure framework
- ⋆ bit-vector theories
- ⋆ computational complexity of solving

Part II Automation in model checking: hierarchical decomposition

Part III Automation in concurrency: identifying togetherness

Automated Theorem Proving

$$\Gamma \vdash \varphi$$

Γ : assumptions

φ : conclusion

Inference system $\rightsquigarrow$ **Theorem prover**

We want *reasonable* tool support:

- high-level strategies (like *induct & simplify*)
- heuristics to search for proofs
- full automation in simple cases

Automated Theorem Proving

$$\Gamma \vdash \varphi$$

Γ : assumptions

φ : conclusion

Inference system	$\rightsquigarrow$	**Theorem prover**
sequent calculus		PVS

We want *reasonable* tool support:

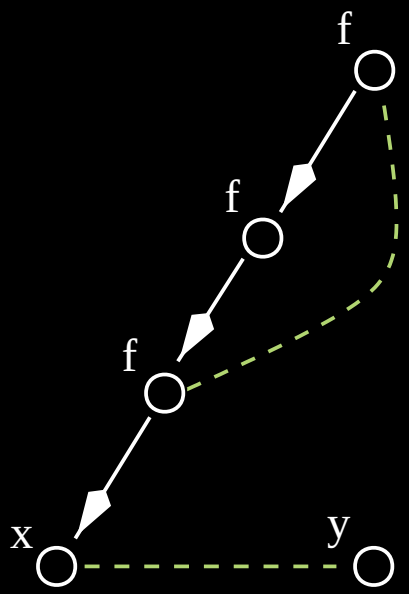
- high-level strategies (like *induct & simplify*) ✓
- heuristics to search for proofs ✓
- full automation in simple cases congruence closure

Encoding Equations in a DAG

Given: $f\,f\,f\,x = f\,x, \quad x = y$

Conclude: $f\,f\,f\,f\,f\,y = f\,x$

as a graph

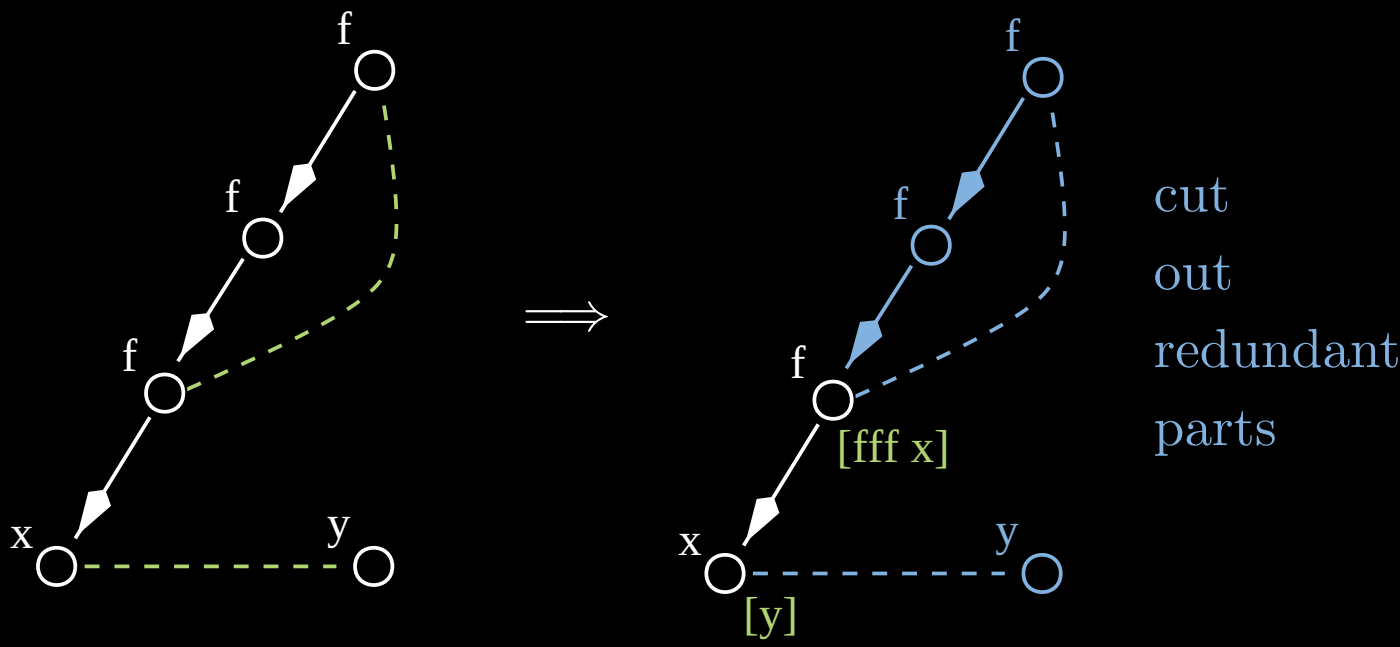


Encoding Equations in a DAG

Given: $f\,f\,f\,x = f\,x, \quad x = y$

Conclude: $f\,f\,f\,f\,f\,y = f\,x$

as a graph



cut out redundant parts

$$f\,f\,f\,f\,f\,y = f\,f\,f\,f\,f\,x$$

$$f\,f\,f\,f\,f\,x = f\,f\,f\,x$$

$$f\,f\,f\,x = f\,x$$

Shostak Style Framework

Theorem

$$T \vdash a = b \quad \text{iff} \quad node(a)\ \hat{R}_T\ node(b)$$

Shostak '84: It is possible to combine *several* theories in this framework, if they are *algebraically solvable*.
For every theory we have

canonizer: term $t \quad \longrightarrow \quad \sigma(t)$

unique representation: $\models\ t = u \ \Leftrightarrow\ \sigma(t) \doteq \sigma(u)$

solver: equation $t = u \quad \longrightarrow \quad \bigwedge_i x_i = s_i$

explicit description of all solutions

Bit-Vector Theories

Core

- $x_{[n]} \ : \ bvec_n$
  number of bits n a constant
- Constants $c_{[n]}$, $c \in \mathbb{N}$
- Concatenation:

$x_{[3]}$ $y_{[3]}$ $x_{[3]} \otimes y_{[3]}$

- Extraction:

$[0:1]$

$x_{[3]}$ $x_{[3]}[0:1]$

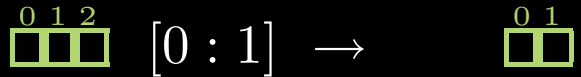
Bit-Vector Theories

Core

- $x_{[n]} \ : \ bvec_n$
  number of bits n a constant
- Constants $c_{[n]}$, $c \in \mathbb{N}$
- Concatenation:

$x_{[3]}$ $y_{[3]}$ $x_{[3]} \otimes y_{[3]}$

- Extraction:



$[0:1] \ \rightarrow$

$x_{[3]}$ $x_{[3]}[0:1]$

Extensions

- Boolean operations

$$x_{[3]} \wedge y_{[3]}$$

- Arithmetic (modulo 2^n)

$$x_{[3]} \ +_{[3]} \ y_{[3]}$$

- Variable length

$x_{[n]}$: $bvec_n$

n : $\mathbb{N}$

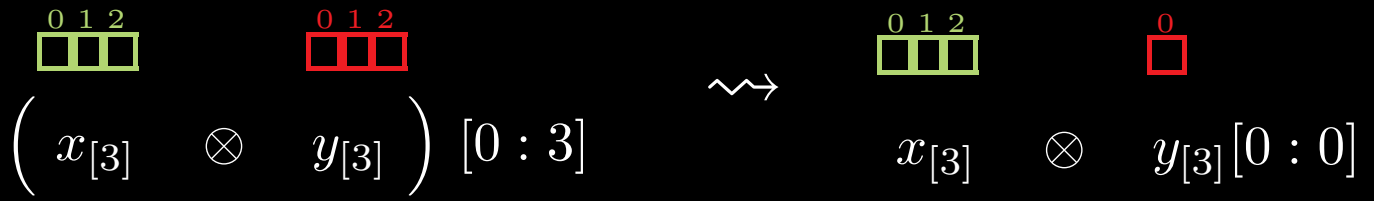
- Variable Extraction

$x_{[3]}$[i:j] : $bvec_{j-i+1}$

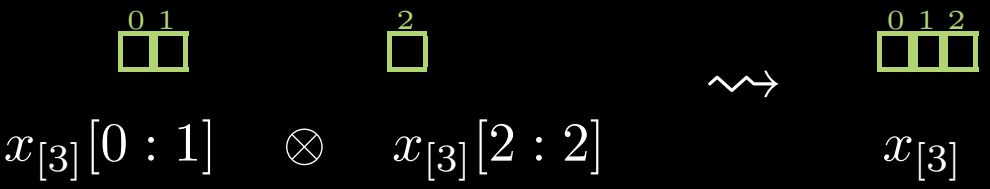
i, j : $\mathbb{N}$

Canonizing Bit-Vector Terms

Phase α: flatten extractions



Phase β: glue together



Bit-Vector Solver: Chop

$$(1) \qquad p_{[n]} \otimes t = q_{[n]} \otimes u \qquad \left\{ \begin{array}{l} p_{[n]} = q_{[n]} \\ t = u \end{array} \right\}$$

$$(1') \qquad p_{[n]} \otimes t = q_{[m]} \otimes u, \quad n < m \qquad \left\{ \begin{array}{l} p_{[n]} = \sigma(q_{[m]}[0 : n-1]) \\ \sigma(q_{[m]}[n : m-1]) \otimes u = t \\ q_{[m]} = \sigma(q_{[m]}[0 : n-1]) \otimes \sigma(q_{[m]}[n : m-1]) \end{array} \right\}$$

$$(1'') \qquad p_{[n]} \otimes t = q_{[m]} \qquad \left\{ \begin{array}{l} p_{[n]} = \sigma(q_{[m]}[0 : n-1]) \\ \sigma(q_{[m]}[n : m-1]) = t \\ q_{[m]} = \sigma(q_{[m]}[0 : n-1]) \otimes \sigma(q_{[m]}[n : m-1]) \end{array} \right\}$$

Bit-Vector Solver: Propagate

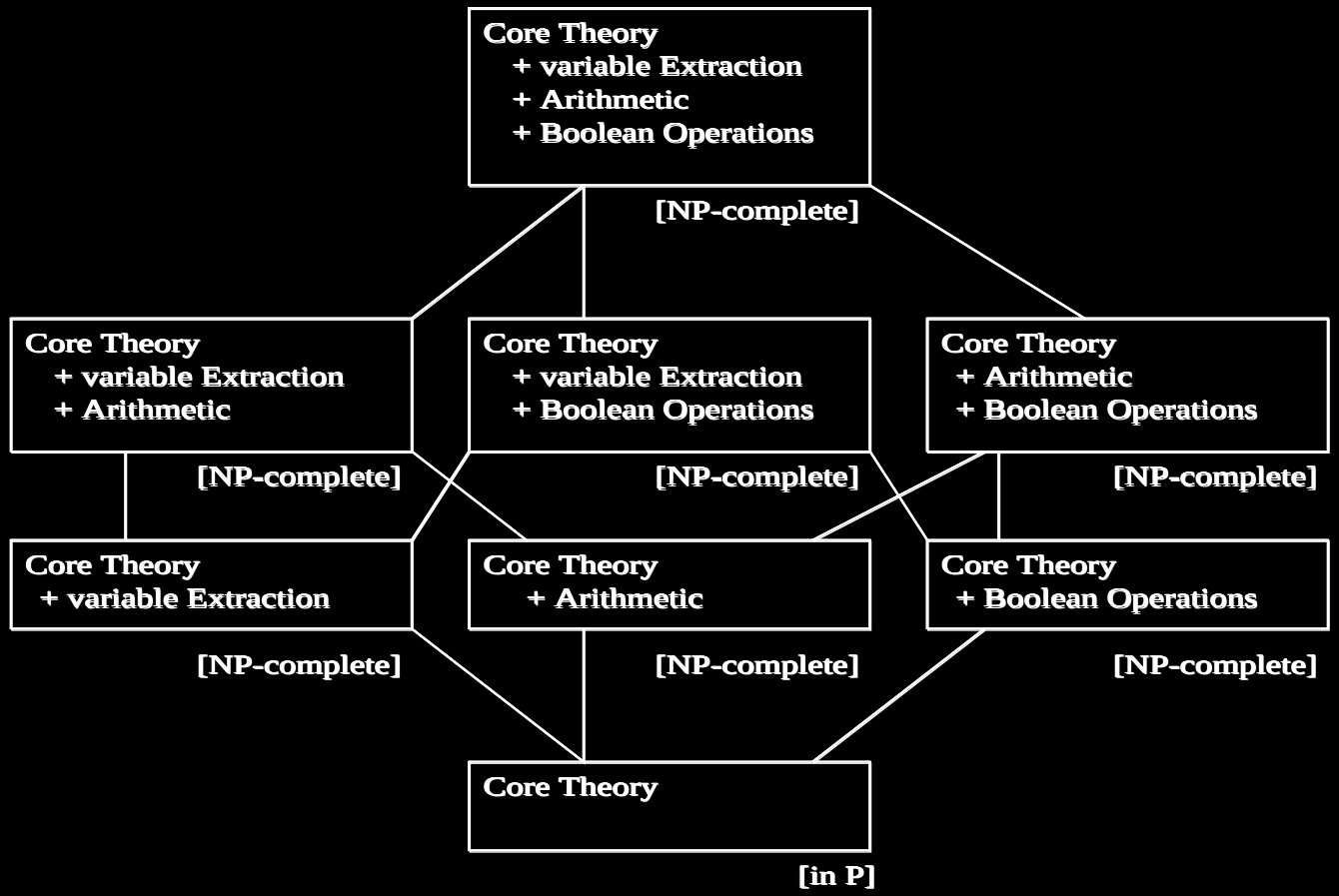
(2)	$c = d\ ,$	$c \dot{\neq} d$	**FAIL**
(3)	$t = t$		$\{\}$
(4)	$\left\{ \begin{array}{l} p = t \\ q = u \end{array} \right\}\ ,$	$q \preceq t$	$\left\{ \begin{array}{l} p = t[q/u] \\ q = u \end{array} \right\}$
(5)	$\left\{ \begin{array}{l} p = q \\ q = r \end{array} \right\}$		$\left\{ \begin{array}{l} p = r \\ q = r \end{array} \right\}$
(6)	$\left\{ \begin{array}{l} p = q \\ q = p \end{array} \right\}$		$\{\ p = q\ \}$
(7)	$\left\{ \begin{array}{l} p = t \\ p = u \end{array} \right\}$		$\left\{ \begin{array}{l} p = t \\ u = t \end{array} \right\}$
(8)	$c = t,$	$t \notin \mathcal{C}$	$\{\ t = c\ \}$
(9)	$p = q \otimes t,$	$p \dot{\neq} \sigma(q \otimes t)$	$\{\ q \otimes t = p\ \}$

Complexity of the Satisfiability Problem

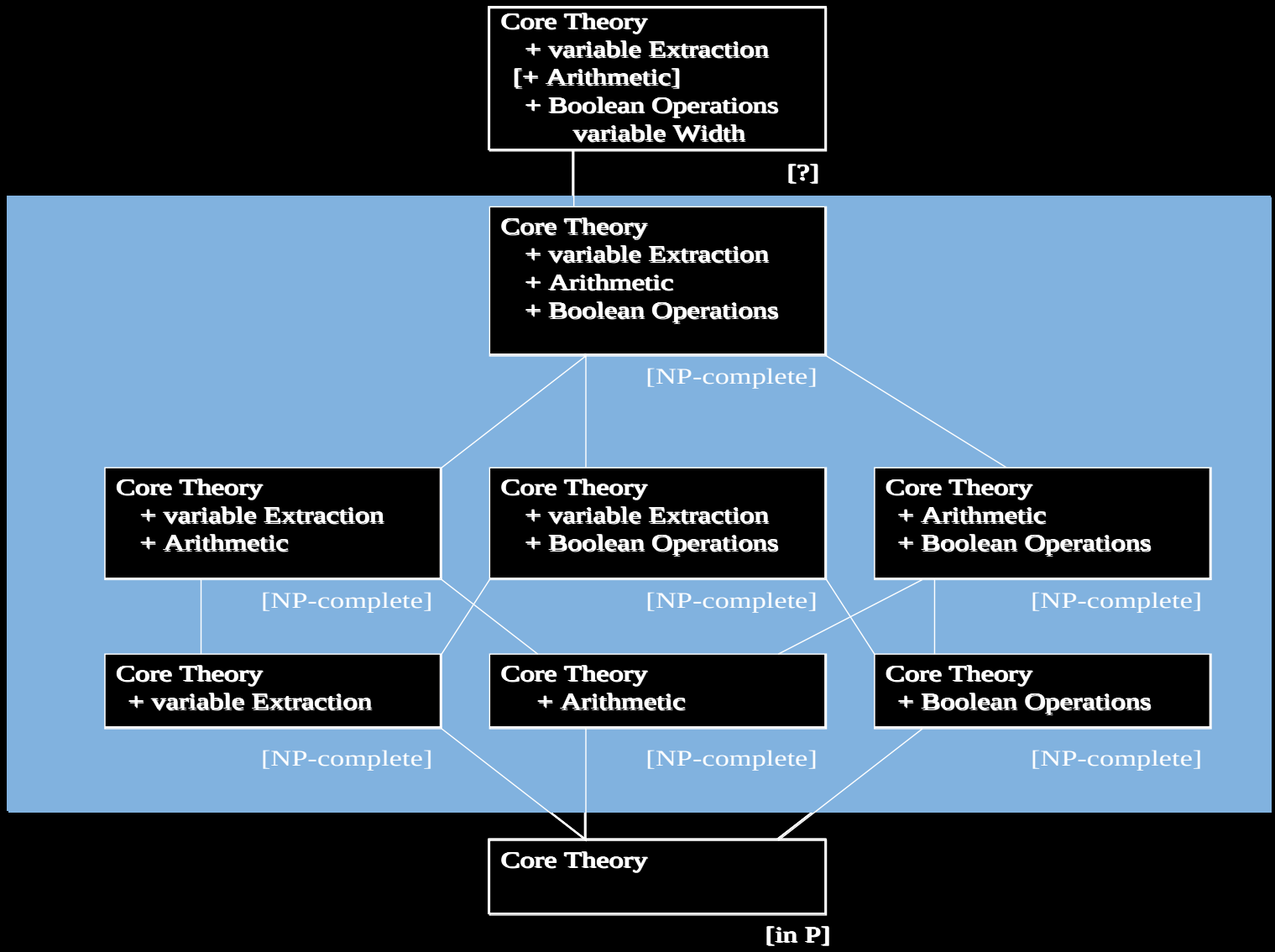
Core Theory

[in P]

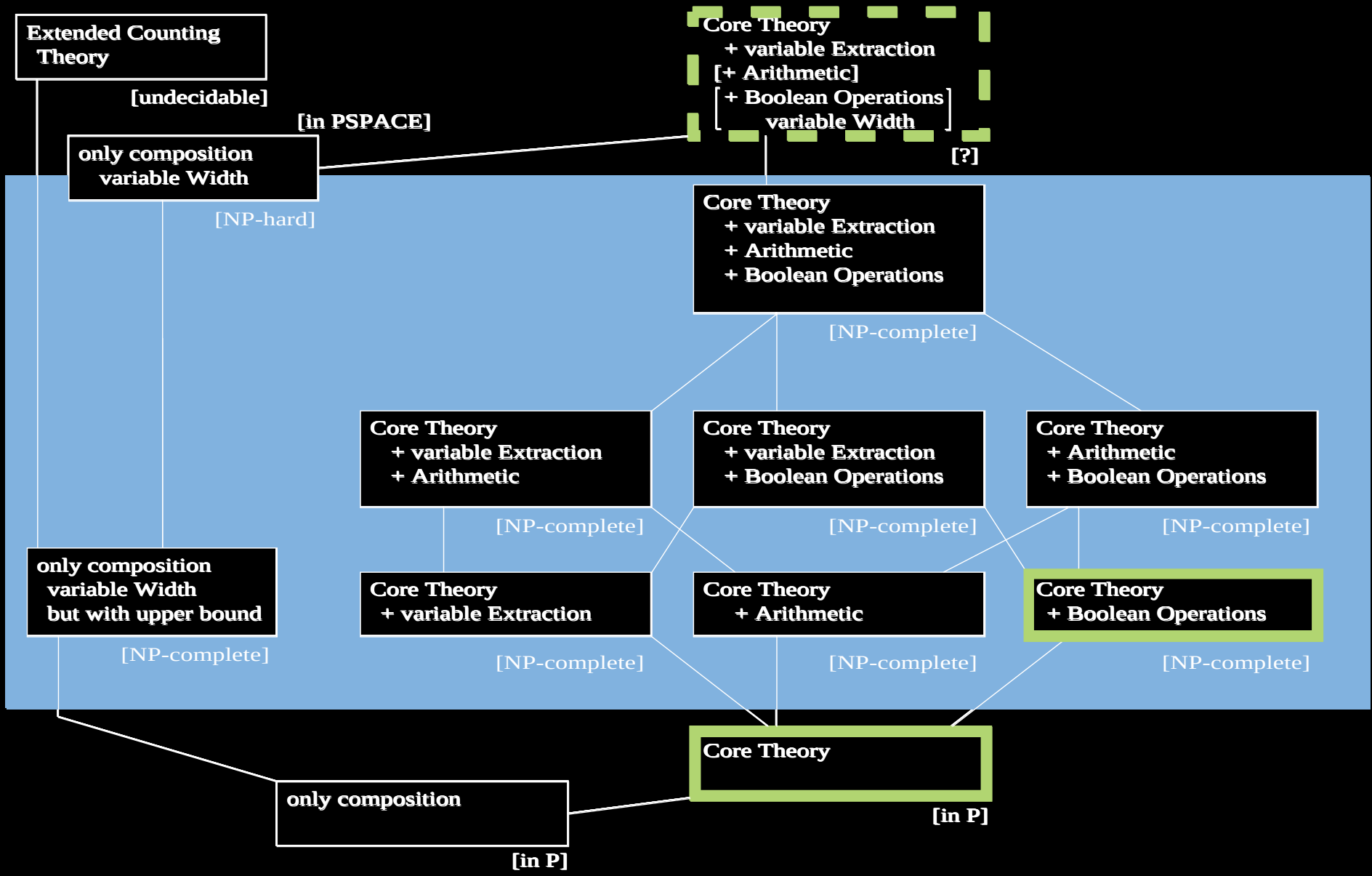
Complexity of the Satisfiability Problem



Complexity of the Satisfiability Problem



Complexity of the Satisfiability Problem



Expressiveness of Solving

$$\forall x.\exists y.\exists z.(x \vee y) \leftrightarrow \neg z$$

$$solve\big((x \vee y) \leftrightarrow \neg z\big) \quad = \quad \left\{ \begin{array}{lcc} x & = & a \\ y & = & b \\ z & = & \neg a \wedge \neg b \end{array} \right\}$$

$$\forall x.\exists y.\exists z.(x \vee y) \leftrightarrow \neg z \quad : \quad \mathit{true}$$

Expressiveness of Solving

$$\forall x.\exists y.\exists z.(x \vee y) \leftrightarrow \neg z$$

$$solve\big((x \vee y) \leftrightarrow \neg z\big) \quad = \quad \left\{ \begin{array}{lcc} x & = & a \\ y & = & b \\ z & = & \neg a \wedge \neg b \end{array} \right\}$$

$\forall x.\forall y.\forall z.(x \vee y) \leftrightarrow \neg z$	:	*false*	$\exists x.\forall y.\forall z.(x \vee y) \leftrightarrow \neg z$	: *false*
$\forall x.\forall y.\exists z.(x \vee y) \leftrightarrow \neg z$	:	*true*	$\exists x.\forall y.\exists z.(x \vee y) \leftrightarrow \neg z$	: *true*
$\forall x.\exists y.\forall z.(x \vee y) \leftrightarrow \neg z$	:	*false*	$\exists x.\exists y.\forall z.(x \vee y) \leftrightarrow \neg z$	: *false*
$\forall x.\exists y.\exists z.(x \vee y) \leftrightarrow \neg z$	:	*true*	$\exists x.\exists y.\exists z.(x \vee y) \leftrightarrow \neg z$	: *true*

Quantification Lemma

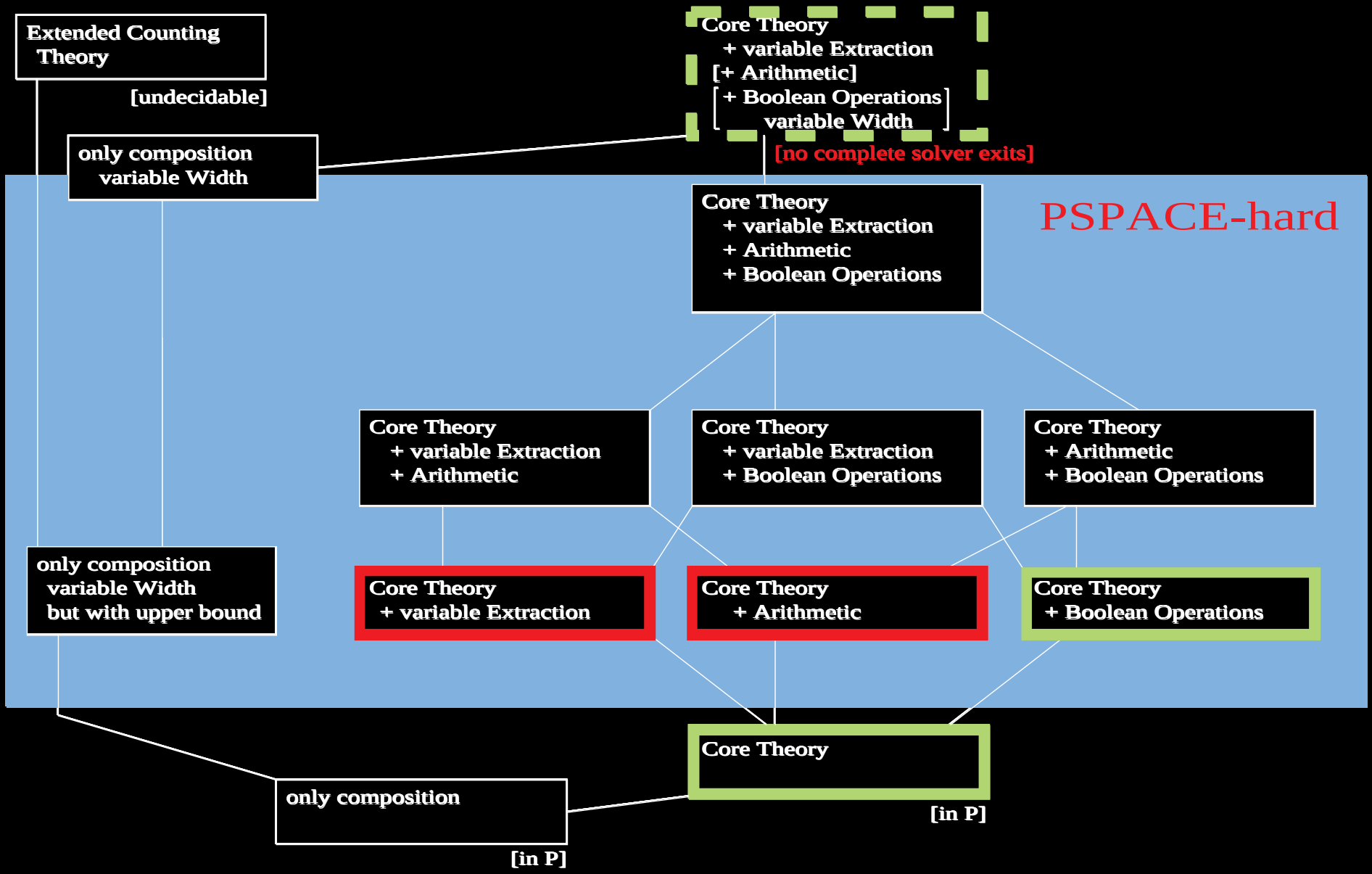
General Method

- start with a quantified equation $Q_1 x_1 ... Q_n x_n.\ t = u$
- let $S = solve(t = u)$;
  derive $\bigwedge_i x_i = s_i$ where
  - all x_i appear in the right order
  - terms s_i do not refer to x_j with $j > i$
- check for $\forall$-quantified x_i, whether s_i is *unrestricted*

} small overhead in the size of the solution $\left(\mathcal{O}(|S| \cdot \log |S|)\right)$

$\Rightarrow$ Solving essentially decides all quantified versions

Complexity Revisited



Summary on Part I

We established

- canonizer and solver for the core theory
- two extensions:
  boolean operations (canonizer + solver)
  variable width (solver)
- quantification lemma

Further development?

- high computational complexity
- interesting fragments?
  $\Rightarrow$ algebraically solvability a serious restriction
- extension of the framework?
  $\Rightarrow$ seems to boil down to a case-split

Roadmap

Part I Automation in theorem proving: congruence closure

- ⋆ congruence closure framework
- ⋆ bit-vector theories
- ⋆ computational complexity of solving

Part II Automation in model checking: hierarchical decomposition

- ⋆ model checking and temporal logic
- ⋆ a technique for temporal scaling
- ⋆ hierarchical structuring

Part III Automation in concurrency: identifying sequential parts

Model Checking

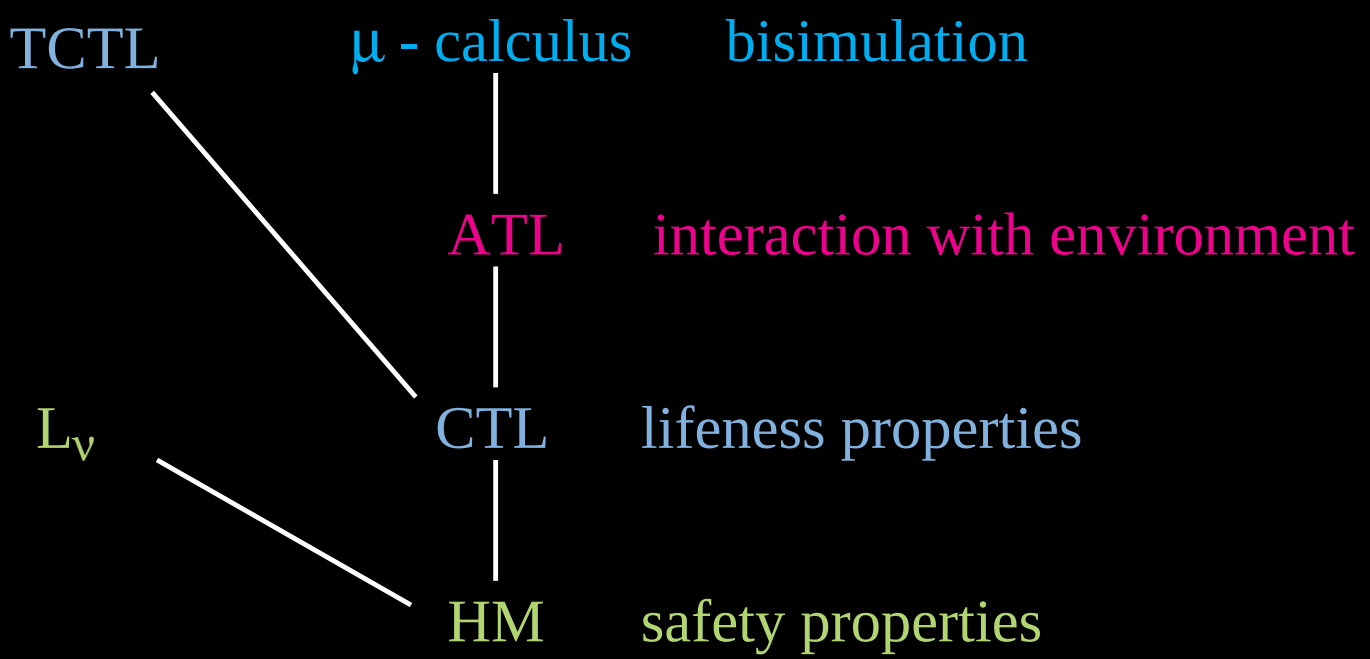
$$M \models \varphi$$

M : description of the system

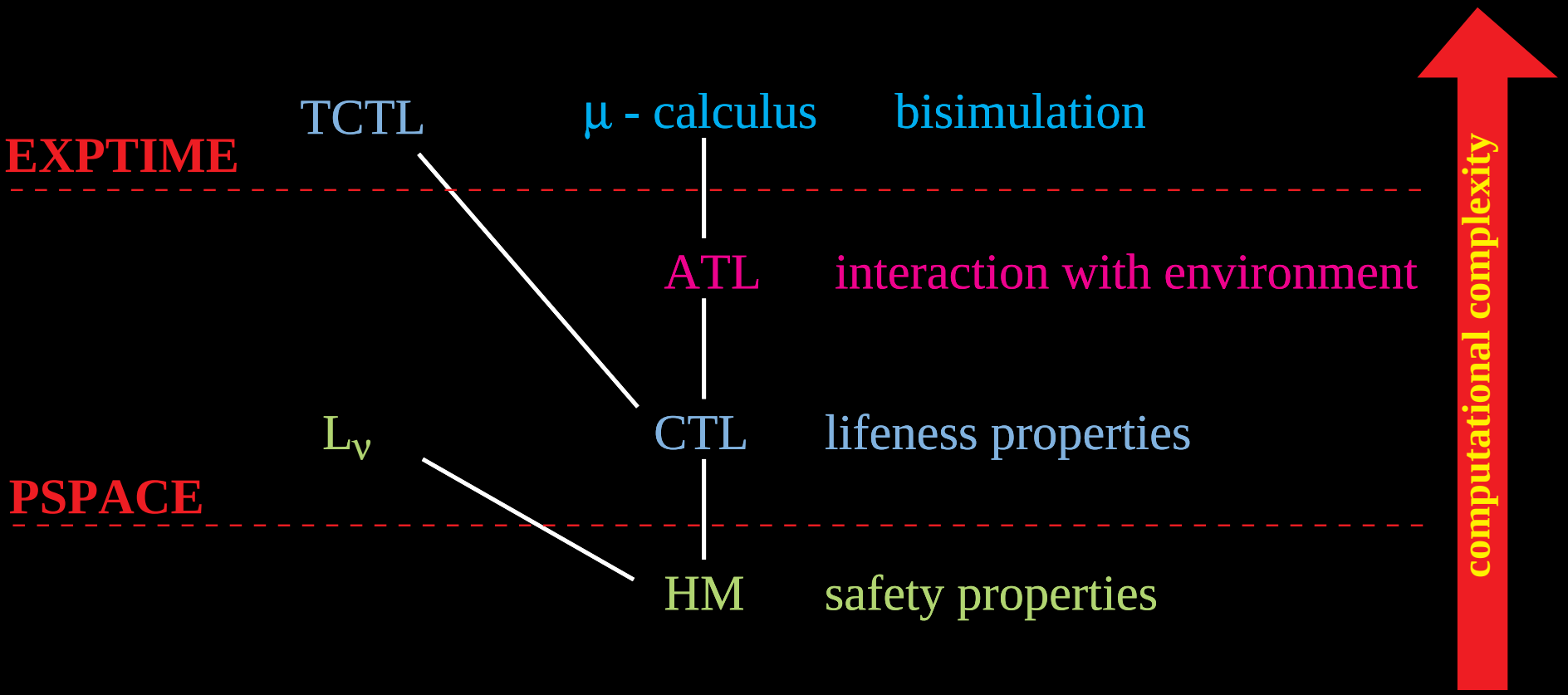
φ : desired property

- easier than proving a general theorem
- completely automatic ('yes' or counterexample)
- efficient algorithms tailored for classes of problems

Temporal Logics



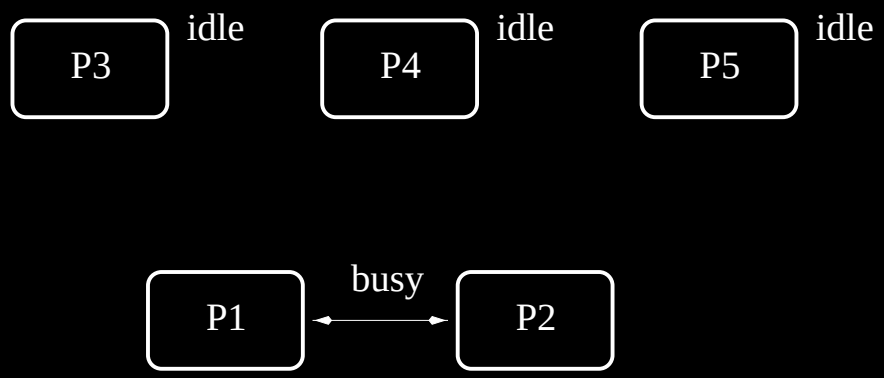
Temporal Logics



Problems with Model Checking

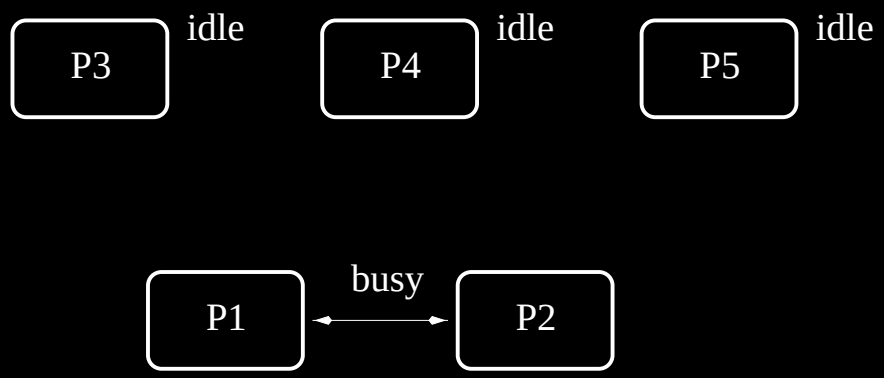
- exponential growth of the state space
  $\Rightarrow$ *state explosion problem*
- restricted expressiveness of the model
  $\Rightarrow$ experience required to make apt abstractions
- restricted expressiveness of the logics
  $\Rightarrow$ properties often have to be *encoded*
- excessive time/space consumption
  $\Rightarrow$ heuristics to make *state space exploration* more efficient

Temporal Scaling

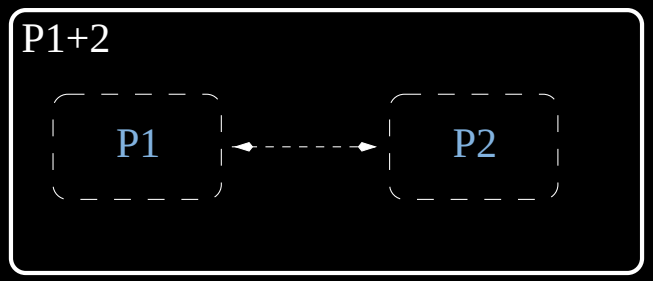


$$P1 \,||\, P2 \,||\, P3 \,||\, P4 \,||\, P5$$

Temporal Scaling



$\Rightarrow$



Instead of:

$$P1 \,||\, P2 \,||\, P3 \,||\, P4 \,||\, P5$$

Use:

$$\textit{hide busy in } (P1 \,||\, P2) \,||\, P3 \,||\, P4 \,||\, P5$$

Model Checking with MOCHA

- parallel execution of components
- communication via shared variables (1 write/multi read)
- *ATL* model checker
- invariant check:
  allows temporal scaling via "next" Θ for P
  (*Rajeev Alur and Bow-Yaw Wang, CONCUR'99*)
- heuristic to preprocess a system for "next"

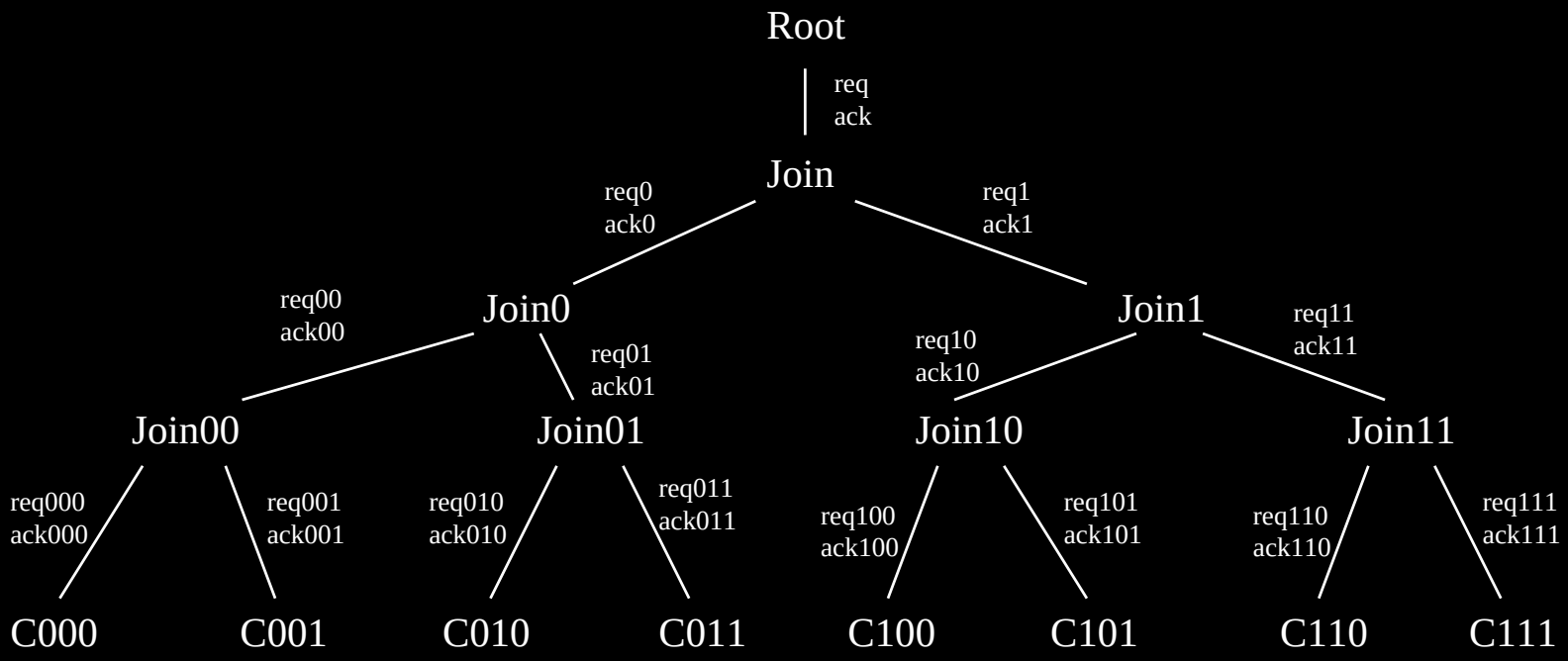
Incremental Clustering

input: hypergraph $\mathcal{H} = (\mathcal{C}, \mathcal{E})$

output: *forest* with leaves $\mathcal{C}$

PriorityQueue Q
forest $:= \{\ \}$
FORALL considered candidates $\mathcal{A} \subseteq \mathcal{C}$
    insert$(\mathcal{A}, Q)$
WHILE *notempty*(Q)
    $\mathcal{A} :=$ *top*(Q)
    fresh node $\textcircled{\mathcal{A}}$
    forest $:=$ *forest* $+\ (\textcircled{\mathcal{A}} \mapsto \mathcal{A})$
    redirect hyperedges: $A_i \in \mathcal{A}$ becomes $\textcircled{\mathcal{A}}$
    FORALL $\mathcal{B} \in Q$ with $\mathcal{B} \cap \mathcal{A} \neq \emptyset$
        remove$(\mathcal{B},\ Q)$
    FORALL new candidates $\mathcal{D}$ containing $\textcircled{\mathcal{A}}$
        insert$(\mathcal{D},\ Q)$

Asynchronous Parity Computer

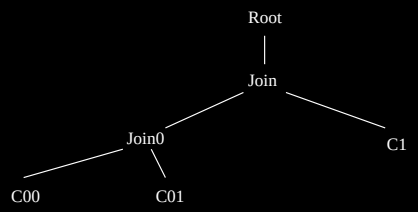


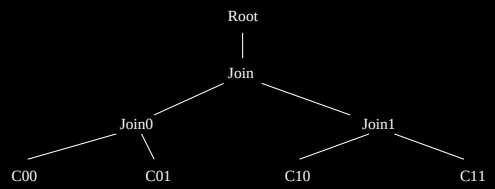
Clients : issue value *true* or *false*

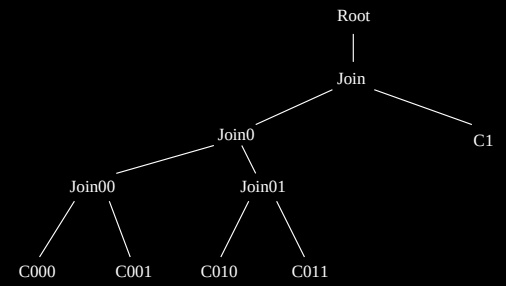
Joins : compute *xor*

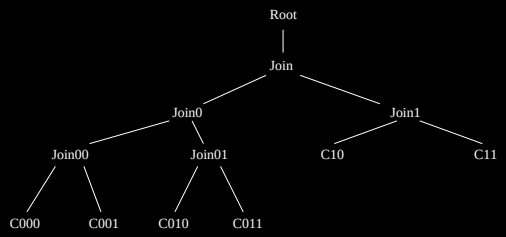
Root : acknowledges

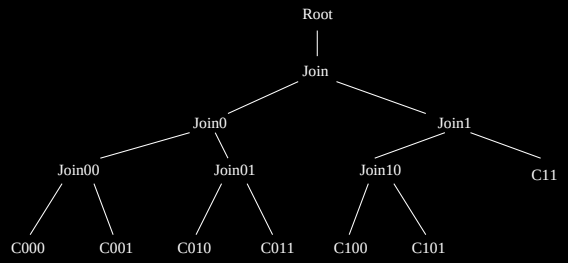
Good Heuristic Function

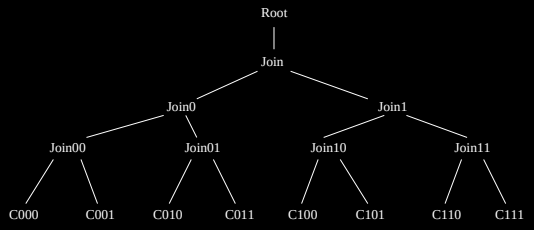






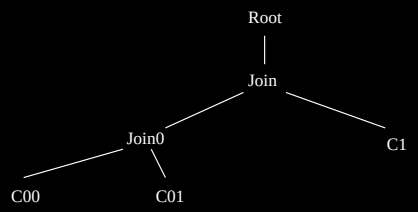


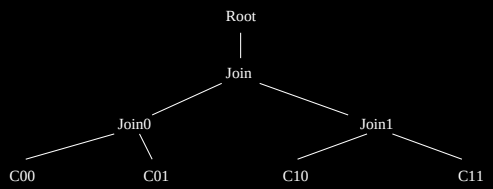


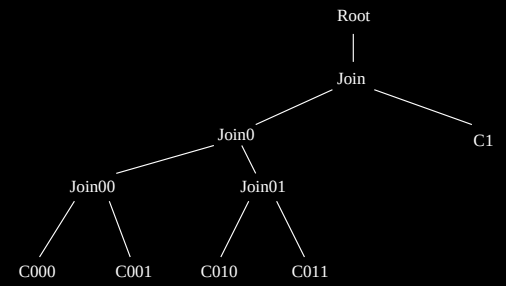


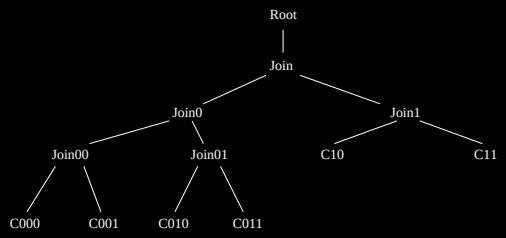
Cover-Number Size Egdes Depth

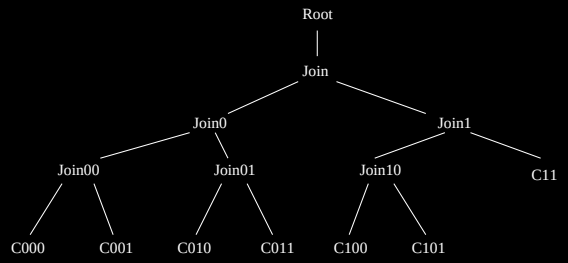
Good Heuristic Function

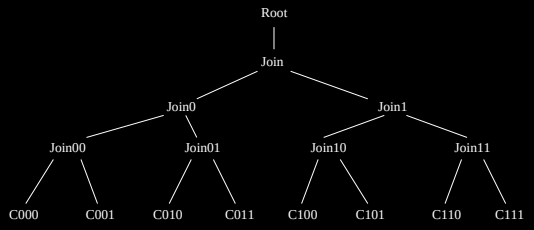








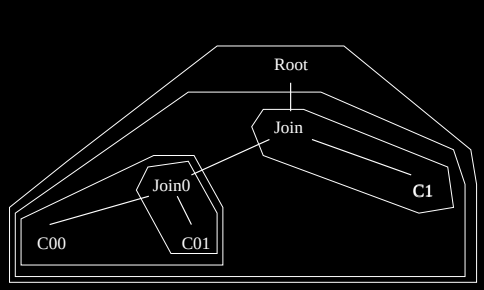


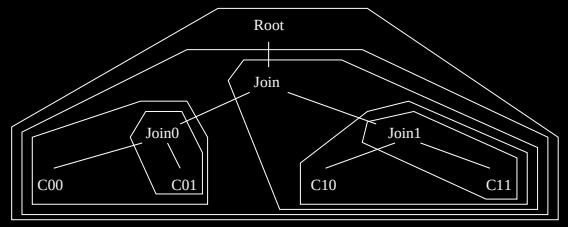


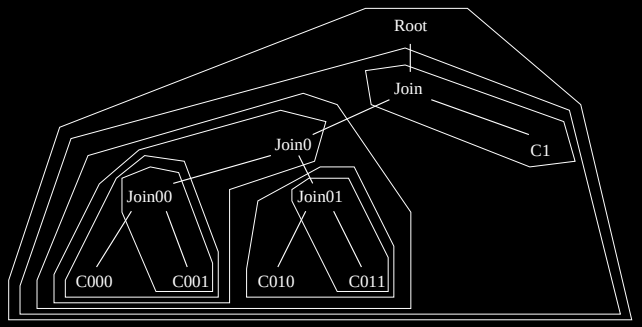
$$\mathbf{r}^{+}_{pref}(\mathcal{A}) := \frac{|\{e \in \mathcal{E} \mid e \subseteq \mathcal{A}\}|}{|\mathcal{A}|^2} + \frac{\varepsilon_1}{|\{e \in \mathcal{E} \mid e \cap \mathcal{A} \neq \emptyset\}|} + \frac{\varepsilon_2}{depth(\mathcal{A})}$$

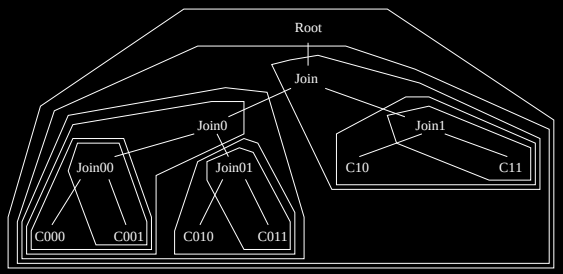
Cover-Number Size Egdes Depth

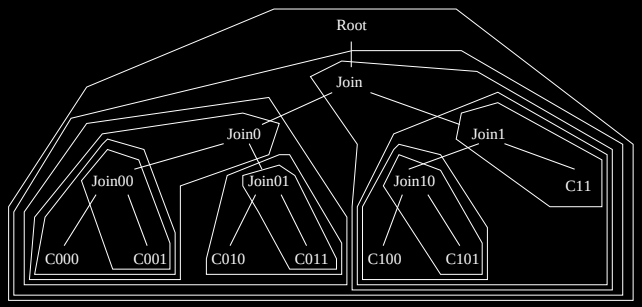
Good Heuristic Function

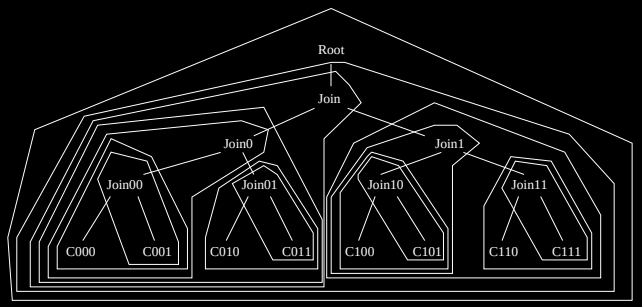












$$\mathbf{r}^{+}_{pref}(\mathcal{A}) := \frac{|\{e \in \mathcal{E} \,|\, e \subseteq \mathcal{A}\}|}{|\mathcal{A}|^2} + \frac{\varepsilon_1}{|\{e \in \mathcal{E} \,|\, e \cap \mathcal{A} \neq \emptyset\}|} + \frac{\varepsilon_2}{depth(\mathcal{A})}$$

Cover-Number Size Egdes Depth

Summary on Part II

We established

- incremental method for hierarchical decomposition
  (has direct application with temporal scaling)
- heuristic function based on 4 criteria:
  Cover-Number, Size, Edges, Depth
- sample problem, where this heuristic is well-behaved

Left to do

- validate method/function with other sample topologies
- investigate applicability beyond invariant properties
- (possibly) integration in the next MOCHA release

Guiding Idea

System:

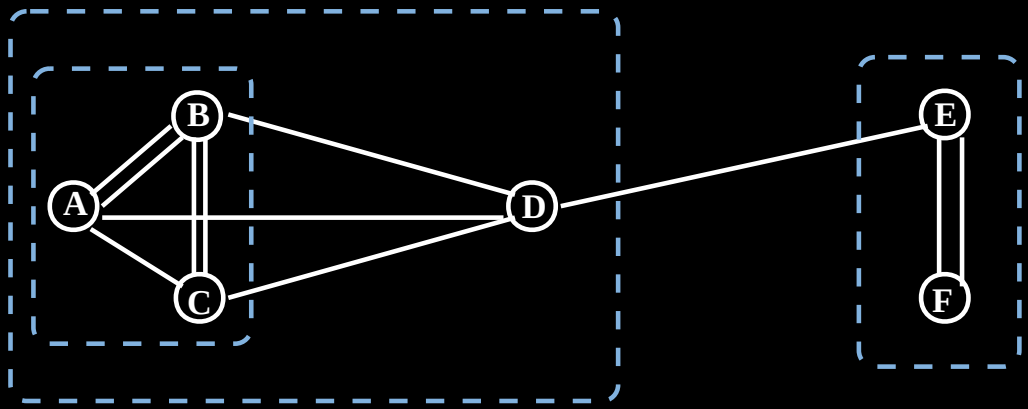
parallel components

Connections:

influence behavior

Structure:

often has *asymmetries*



{A,B,C} belong *strongly* together

{D, E} are not as "together" as {E, F}

General Observation:

If we can adapt our analysis to asymmetries, it becomes either

- more expressive

or • more efficient

We aim to describe, detect and exploit "Togetherness"

Describe Togetherness

Desired Characterization:

$$\{\text{A, B}\} \text{ are together [to extend } \alpha\text{]} \quad \Leftrightarrow \quad \textbf{?}$$

Likely: the links influence this

Unlikely: link information suffices

Methodologies:

- Identification by categorical models
- *CCS*- or *CSP*-like frameworks
- Properties derived from automata interaction

$\Rightarrow$ If goal too ambitious, restrict to special cases

Detect Togetherness

To be expected: Good characterizations are computationally expensive

$\Rightarrow$ identify typical patterns

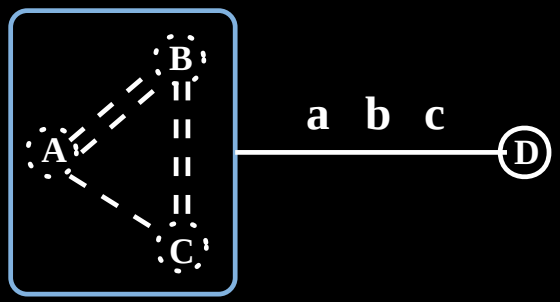
If infeasible, investigate *why*

$\Rightarrow$ lower bounds / hardness results

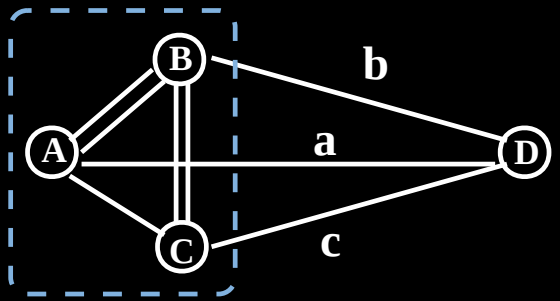
Reasonable hope for *partial* solutions

- develop heuristics (guided search)
- describe special cases

Exploit Togetherness

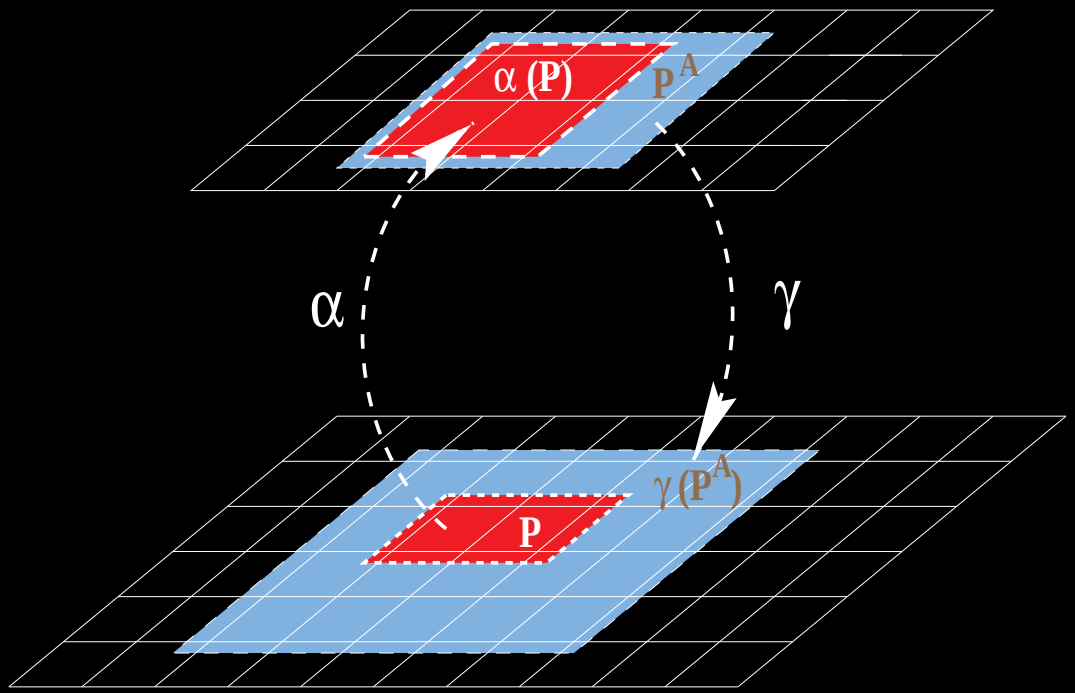


Abstraction



Given System

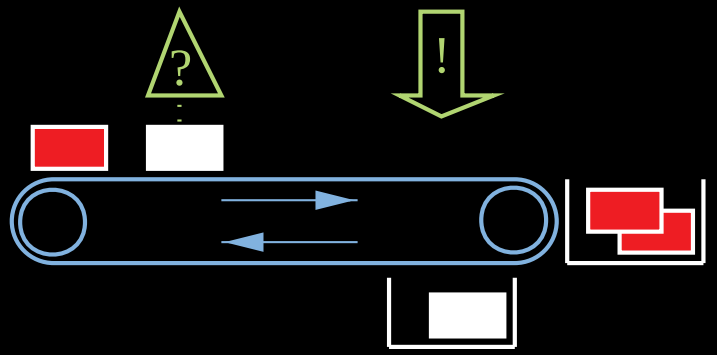
Galois Connection



$$\forall P, P^A : \ \alpha(P) \subseteq P^A \Leftrightarrow P \subseteq \gamma(P^A)$$

Property holds in abstraction $\Longrightarrow$ Property holds in actual system

Starting Point: Brick Sorter



Verify controller, running on LEGO® RCX

- Specified as timed automaton (UPPAAL)
- Proven Property: *always will kick off black*

Problem

model checking slow: many delay steps without real progress

Possible solutions

- temporal scaling: $\approx$ '*next* Θ *for* P'
- abbreviations: $1000\,a \rightarrow 512\,a + 256\,a + 128\,a + 64\,a + 32\,a + 8\,a$
- building abstraction and prove it correct

Summary

Previous work

- Bit-Vectors: 2 canonizers, 3 solvers, 2 new complexity results
- Model-Checking: heuristic to partition a system hierarchically

} backed with (experimental) implementations

Future plans

- Develop a variation of 'next', tailored for timed automata
- Explore the phenomenon of *Togetherness* in concurrent systems in greater detail