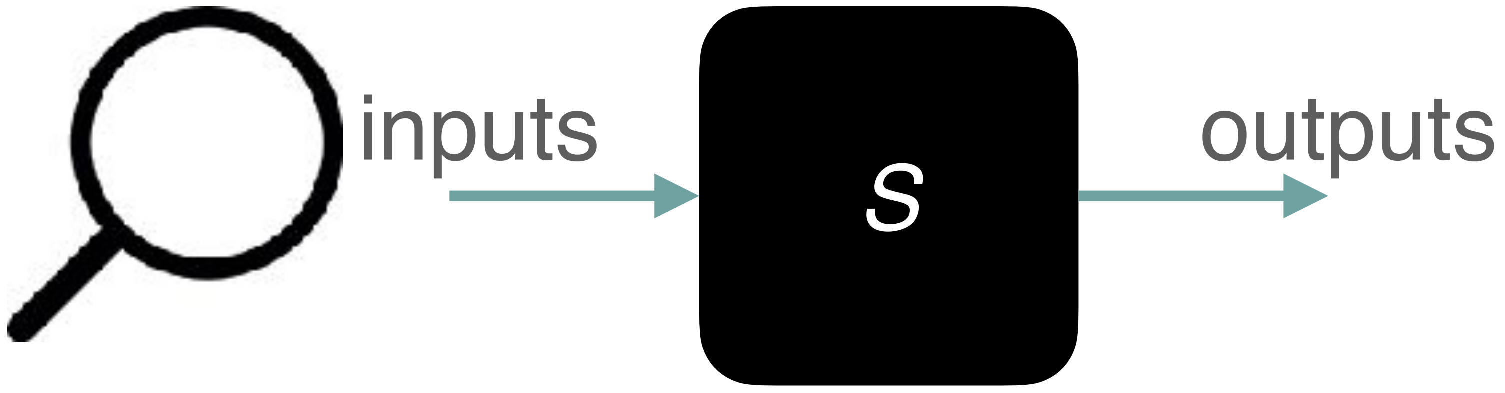
Lecture 17

Pete Manolios
Northeastern

Reasoning About Systems

- System *S*
- Specification *f*



- *S* is *correct* if it satisfies *f*
  - *f* is a claim about the *observable* behaviors of *S*
- Once we agree on *f*, the internals of *S* can be ignored
  - Typically, *f* is much simpler than *S*
  - Specs are reusable, allow for separation of concerns, …
- Specifications should be as simple as possible to minimize the cognitive load required for humans to accept them

Sorting Specification

Ask Northeastern Logic and Computation freshmen to define a spec for sorting

Sorting Specification

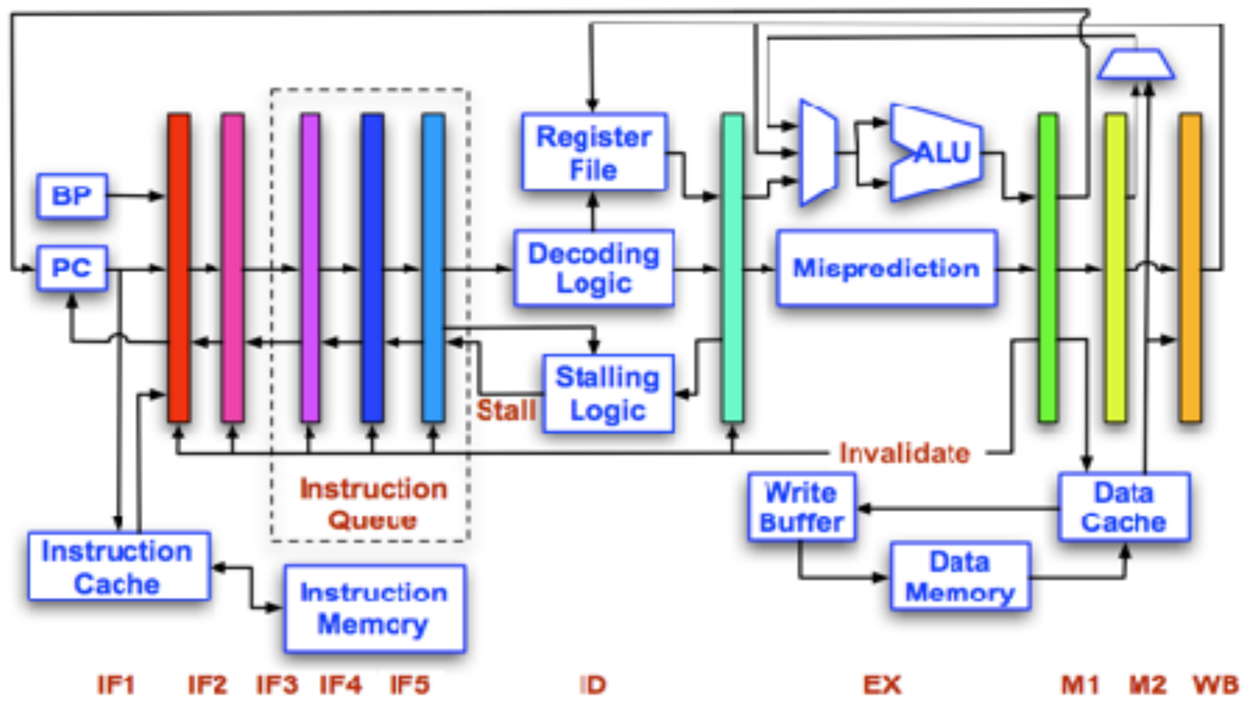
- The output is ordered
  - For all l, i, j s.t. l is a list of numbers, i, j are natural numbers and $i < j <$ (len (sort l)): (sort l)[i] $\leq$ (sort l)[j]
- What if (sort l) is empty, but l is not?
  - For all l s.t. l is a list of numbers: (len (sort l)) = (len l)
- What if $n \in l$, but $n \notin$ (sort l)?
  - For all l, n s.t. …: $n \in l \Rightarrow n \in$ (sort l)
- What if n appears i times in l, but $j \neq i$ times in (sort l) ?
  - (sort l) is a permutation of l
- Define permutation
- Are we done (are the properties complete)?
- Are the properties independent?

Sorting Spec via Refinement

- What about
  - For all *l* s.t. *l* is a list of numbers: (sort *l*) = (isort *l*)
  - Where isort is insertion sort?
- Advantages
  - No need to define permutation, …
  - Defining isort is simple
  - Completeness trivial
  - Independence trivial
  - Easier spec to understand and accept

Reactive Systems

- Refinement
  - *I* refines *S* if every behavior of *I* is allowed by *S*
- Examples
  - Pipelined Microprocessors
  - Sequential Consistency
  - Communications Protocols
  - Event Processing Systems
  - Distributed Databases



Hardware Verification: Motivation

International Technology Roadmap for Semiconductors, 2004 Edition.

Verification has become the dominant cost in the design process. On current projects, verification engineers outnumber designers, with this ratio reaching two or three to one for the most complex designs.

...

Without major breakthroughs, verification will be a non-scalable, show-stopping barrier to further progress in the semiconductor industry.

…

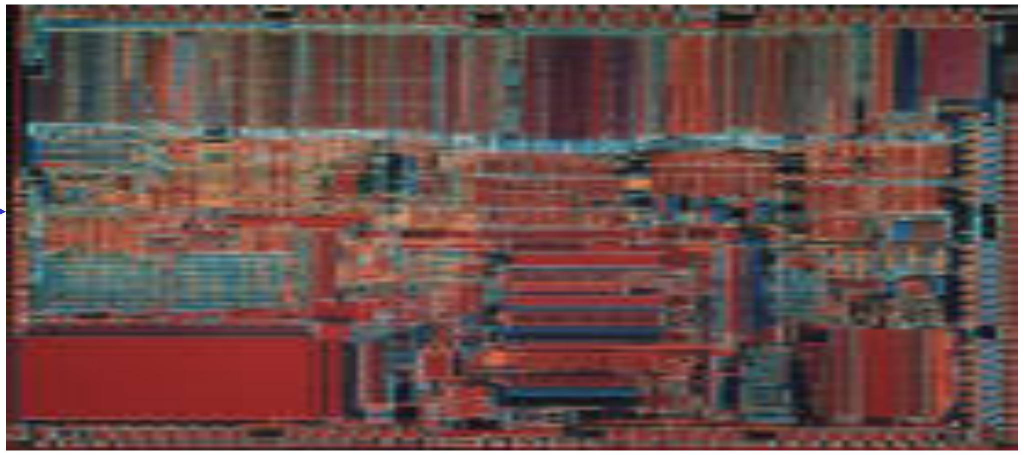
The overall trend from which these breakthroughs will emerge is the shift from ad hoc verification methods to more structured, formal processes.

Hardware Verification Challenge

- Verification costs range from 30%-70% of the entire design cost.
- R&D for typical CPU: 500+ team, costing $0.5-1B.
- Pentium 4 (Bob Bently CAV 2005).
  - Full-chip simulation ~20Hz on Pentium 4.
  - Used ~6K CPUs 24/7: ~3 years later <1 minute of simulation cycles.
  - Exhaustive testing is impossible.
  - First large-scale formal verification at Intel: 60 person years.
  - Checked over 14K properties: Decode, Floating point, pipeline.
  - Found bugs, but no silicon bugs found to date in these areas.

Pentium FDIV
(Floating point DIVision) bug in
Pentium 386 led to a
$475 million write-off by Intel

Bob Bently CAV **$12B** 2005 terms



Outline

- Pleasantness Problem
- Refinement
- Local Reasoning
- Pipelined Machine Verification
- Automating Refinement
- Refinement Map Factor
- Compositional Reasoning
- Deductive Methods & Decision procedures
- Conclusions

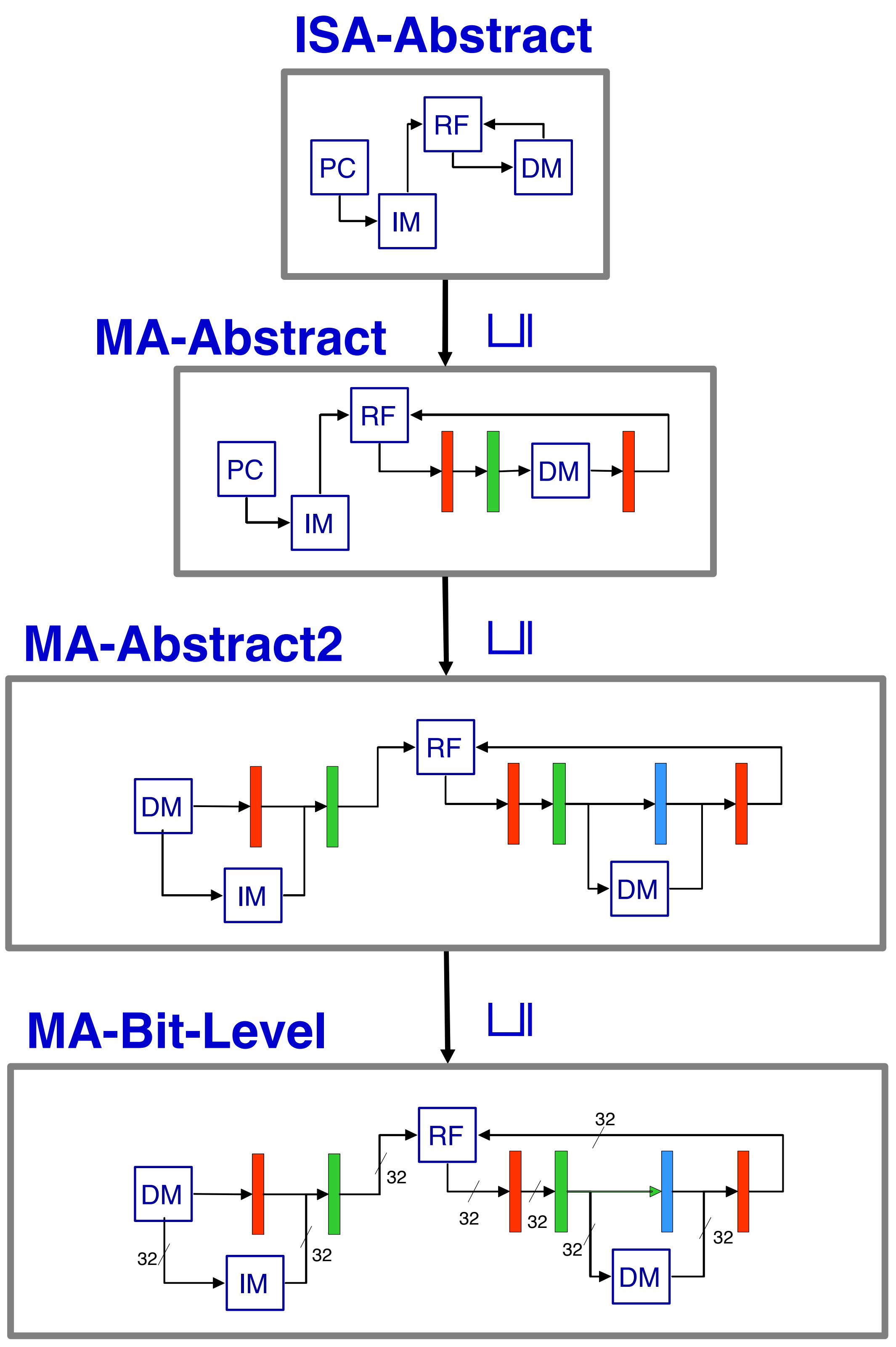
Approaches to Verification

- Property Based.
  - Relational, e.g., sorting.
  - Temporal logic, e.g., reactive systems.
- Refinement.
  - *I* refines *S* if every behavior of *I* is allowed by *S*.
  - The dual of abstraction.
  - Sorting.
  - Pipelined machines.
  - Communications Protocols.
  - Distributed Databases.

Key Concepts in Refinement

- Refinement: *I* refines *S* if every behavior of *I* is allowed by *S*
- Identify what is *observable*
  - Transformational systems: input & output
  - Reactive systems: refinement maps, e.g., *I* may contain more components and may use different data representations than *S*
- Theoretical, *semantic-based* approach
  - Use Kripke structures $M = \langle S, \dashrightarrow, L \rangle$
  - Language agnostic
- Direct support for *Stuttering*
  - *I* may require several steps to match *S*

Refinement, the Picture



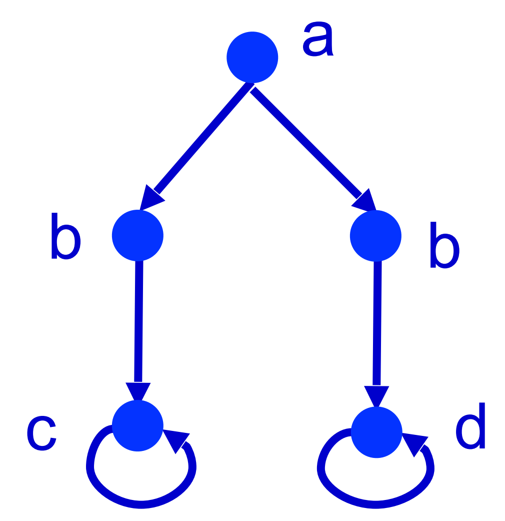
- Formal connection between different abstraction levels.
- Compositional.
- Preservation of safety and liveness.
- Avoid “leaky abstractions.”

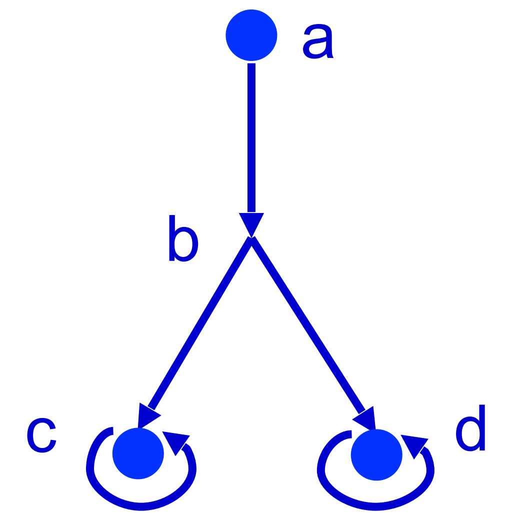
Behaviors

I refines *S* if every behavior of *I* is allowed by *S*

What are the behaviors of a system?

- Linear time
  - Programs and properties are sets of infinite sequences
  - Trace containment, equivalence (PSPACE-complete)
  - LTL
- Branching time
  - Programs and properties are sets of infinite trees
  - Simulation, bisimulation (in P)
  - ACTL*, CTL*, μ-calculus





Existence of Refinement Maps

Abadi and Lamport. *The Existence of Refinement Mappings,* Theoretical Computer Science, 1991.

- Spec: state machine & supplementary property (fairness)
- Proving that *I* refines *S* requires reasoning about infinite sequences: if *I* allows $\langle\langle e_0,z_0\rangle, \langle e_1,z_1\rangle, \langle e_2,z_2\rangle, \ldots\rangle$, then *S* allows $\langle\langle e_0,y_0\rangle, \langle e_1,y_1\rangle, \langle e_2,y_2\rangle, \ldots\rangle$
- Reason ***locally*** (structurally)!
- Definition: If $f(e_n,z_n) = \langle e_n,y_n\rangle$, *f* can be used to prove, locally, that *I* preserves safety properties of *S*. If *f* preserves liveness, then it is a *refinement mapping*.
- Theorem: If the machine-closed specification *I* implements *S*, a specification that has finite invisible nondeterminism and is internally continuous, then there is a specification I^h obtained from *I* by adding a history variable and a specification I^{hp} obtained from I^h by adding a prophecy variable such that there exists a refinement mapping from I^{hp} to *S*.

My Refinement Results

- A *compositional* theory of refinement that deals with liveness
- Branching time
  - Theorem: If *I* implements *S*, there exists a refinement mapping from *I* to *S*.
- Linear time
  - Theorem: If *I* implements *S*, then there is a specification I^o, obtained from *I* by adding an oracle variable, such that there exists a refinement mapping from I^o to *S*.
- Proofs show how to construct refinement maps & provide key insights for mechanization

Outline

Refinement

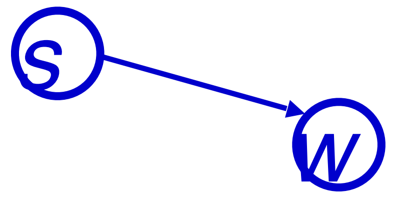
- Transition System (TS) $\mathcal{M} = \langle S, \dashrightarrow, L \rangle$
- Let
  - $\mathcal{M} = \langle S, \dashrightarrow, L \rangle$ (the implementation)
  - $\mathcal{M}' = \langle S', \dashrightarrow', L' \rangle$ (the specification)
  - $r : S \rightarrow S'$
- We say that $\mathcal{M}$ is a *simulation refinement* of $\mathcal{M}'$ with respect to refinement map r, written $\mathcal{M} \sqsubseteq_r \mathcal{M}'$, if there exists a relation, B, such that:
  - $\langle \forall s \in S :: sB(r.s) \rangle$
  - B is an STS on the TS $\langle S \uplus S', \dashrightarrow \uplus \dashrightarrow', \mathcal{L} \rangle$, where $\mathcal{L}.s = L'(s)$ for s an S' state, else $\mathcal{L}.s = L'(r.s)$.
- Compositional: $\mathcal{M} \sqsubseteq_r \mathcal{M}' \wedge \mathcal{M}' \sqsubseteq_q \mathcal{M}'' \Rightarrow \mathcal{M} \sqsubseteq_{r;q} \mathcal{M}''$

Stuttering Simulation

Transition System (TS) $\mathcal{M} = \langle S, \dashrightarrow, L \rangle$

B is a stuttering simulation (STS) on $\mathcal{M}$ iff for all *s*, *w* such that *sBw:*

1. $L.s = L.w$



$\longrightarrow$: *B*

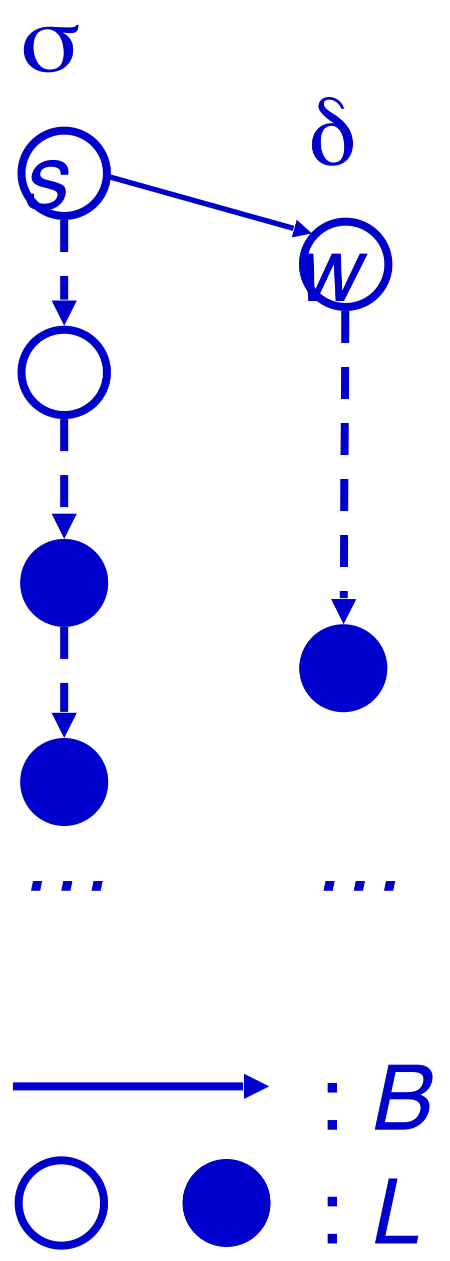
○ ● : *L*

Stuttering Simulation

Transition System (TS) $\mathcal{M} = \langle S, \dashrightarrow, L \rangle$

B is a stuttering simulation (STS) on $\mathcal{M}$ iff for all *s*, *w* such that *sBw:*

1. $L.s = L.w$
2. $\langle \forall \sigma : fp.\sigma.s : \langle \exists \delta : fp.\delta.w : match(B,\sigma,\delta) \rangle\rangle$



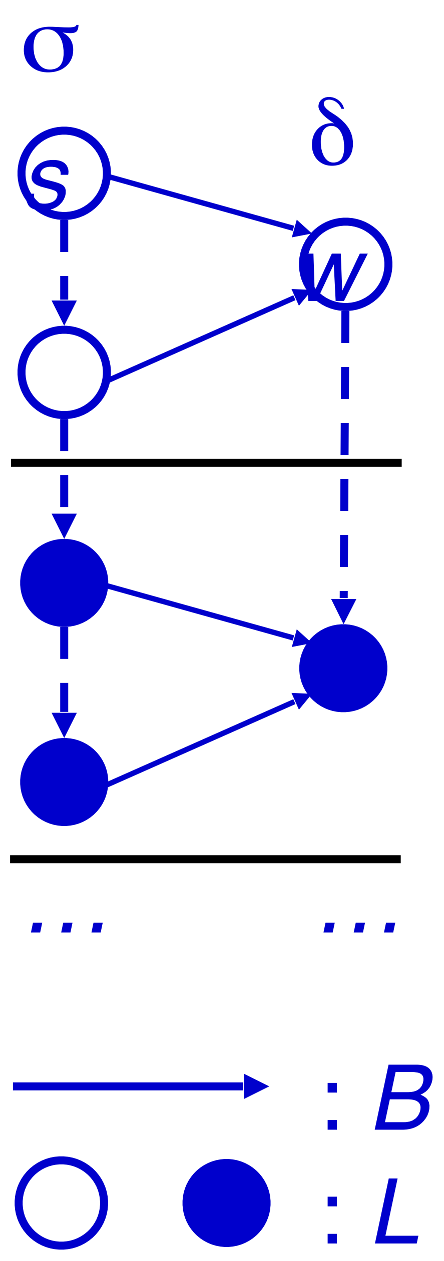
Stuttering Simulation

Transition System (TS) $\mathcal{M} = \langle S, \dashrightarrow\ , L\rangle$

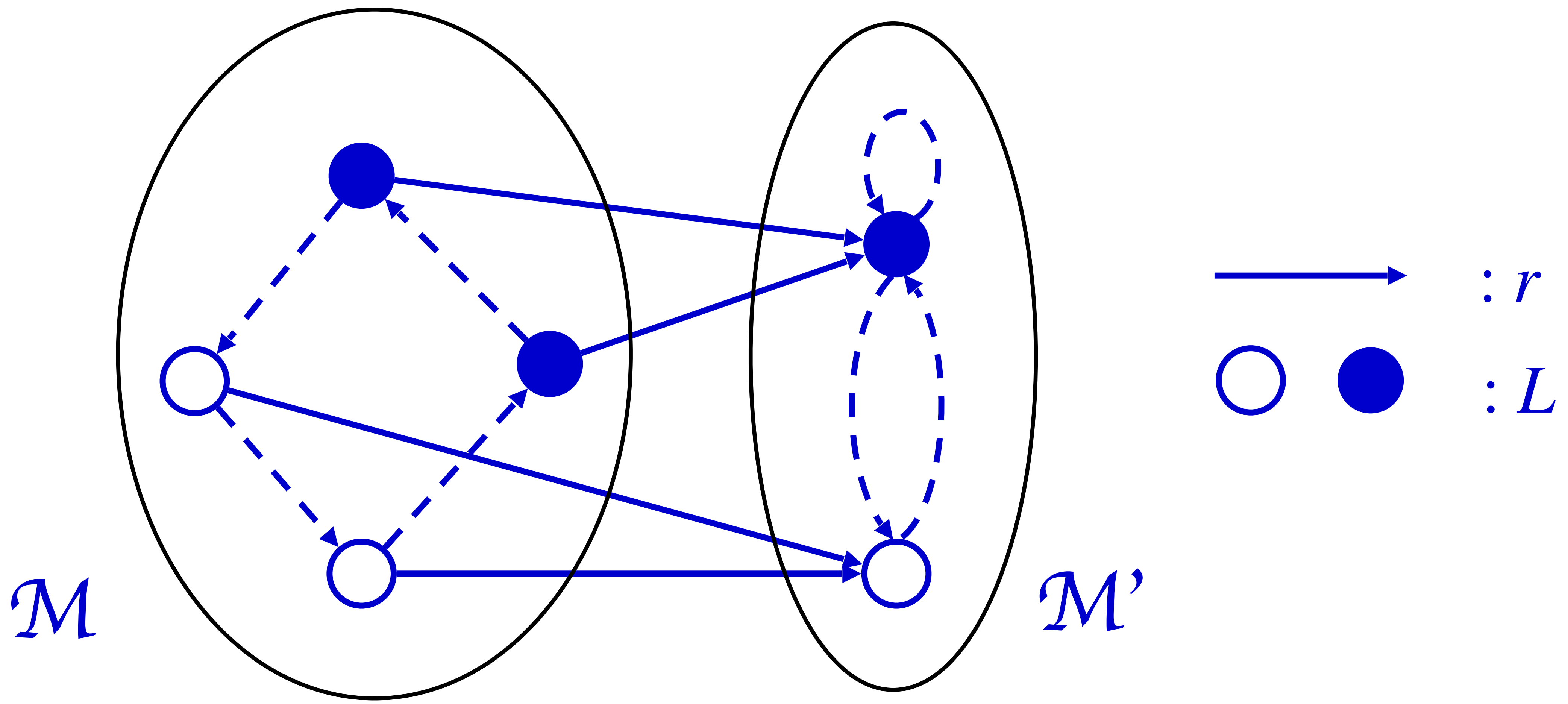
B is a stuttering simulation (STS) on $\mathcal{M}$ iff for all *s*, *w* such that *sBw:*

1. $L.s = L.w$
2. $\langle\forall\sigma : fp.\sigma.s : \langle\exists\delta : fp.\delta.w : match(B,\sigma,\delta)\rangle\rangle$

match(B,σ,δ): σ,δ can be partitioned into non-empty, finite segments such that states in related segments are related by *B*.

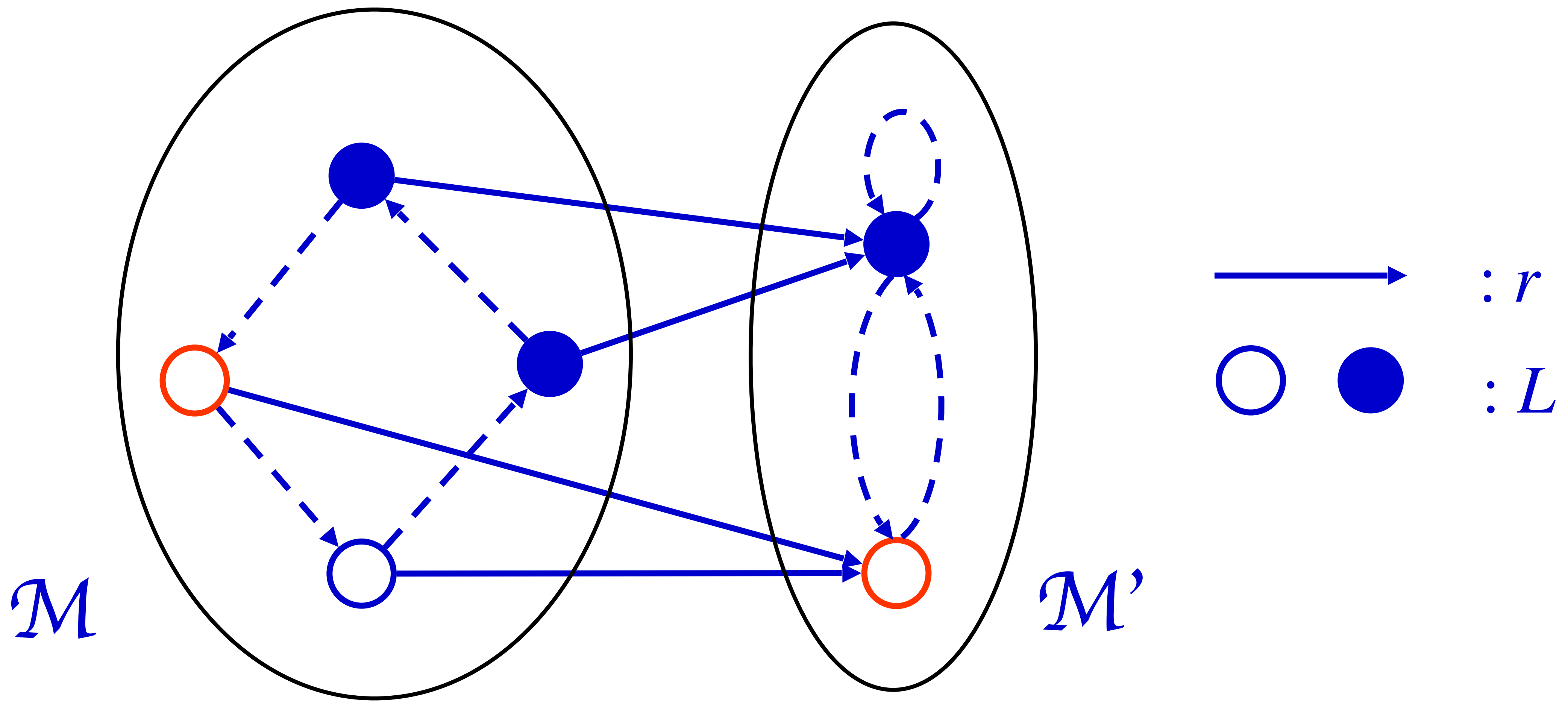


An Example



$\mathcal{M} \sqsubseteq_r \mathcal{M}'$ with witness B, the relation induced by r.

An Example



$\mathcal{M} \sqsubseteq_r \mathcal{M}'$ with witness *B*, the relation induced by *r*.

Stuttering Simulation

- For every TS $\mathcal{M}$, there is a greatest STS on $\mathcal{M}$:
  - Let C be a set of STS's, then $\cup_{B \in C}$ is an STS.
- If B is an STS, so is B*.
  - $B^* = \cup_{i \in \mathbb{N}} B^i$.
  - The identity relation is an STS.
  - If *R* and *S* are STSs, so is *R*;*S*, their composition.
- The greatest STS, G, on $\mathcal{M}$ is a preorder.
  - G* is an STS.
  - $G \subseteq G^*$, but also $G^* \subseteq G$.

Stuttering Simulation

Theorem:

Let B be a STS on $\mathcal{M}$ and let sBw.

For every ACTL*\X formula f,

if $\mathcal{M}, w \models f$, then $\mathcal{M}, s \models f$.

Outline

- Pleasantness Problem
- Refinement
- Local Reasoning
- Pipelined Machine Verification
- Automating Refinement
- Refinement Map Factor
- Compositional Reasoning
- Deductive Methods & Decision procedures
- Conclusions

Proof Considerations

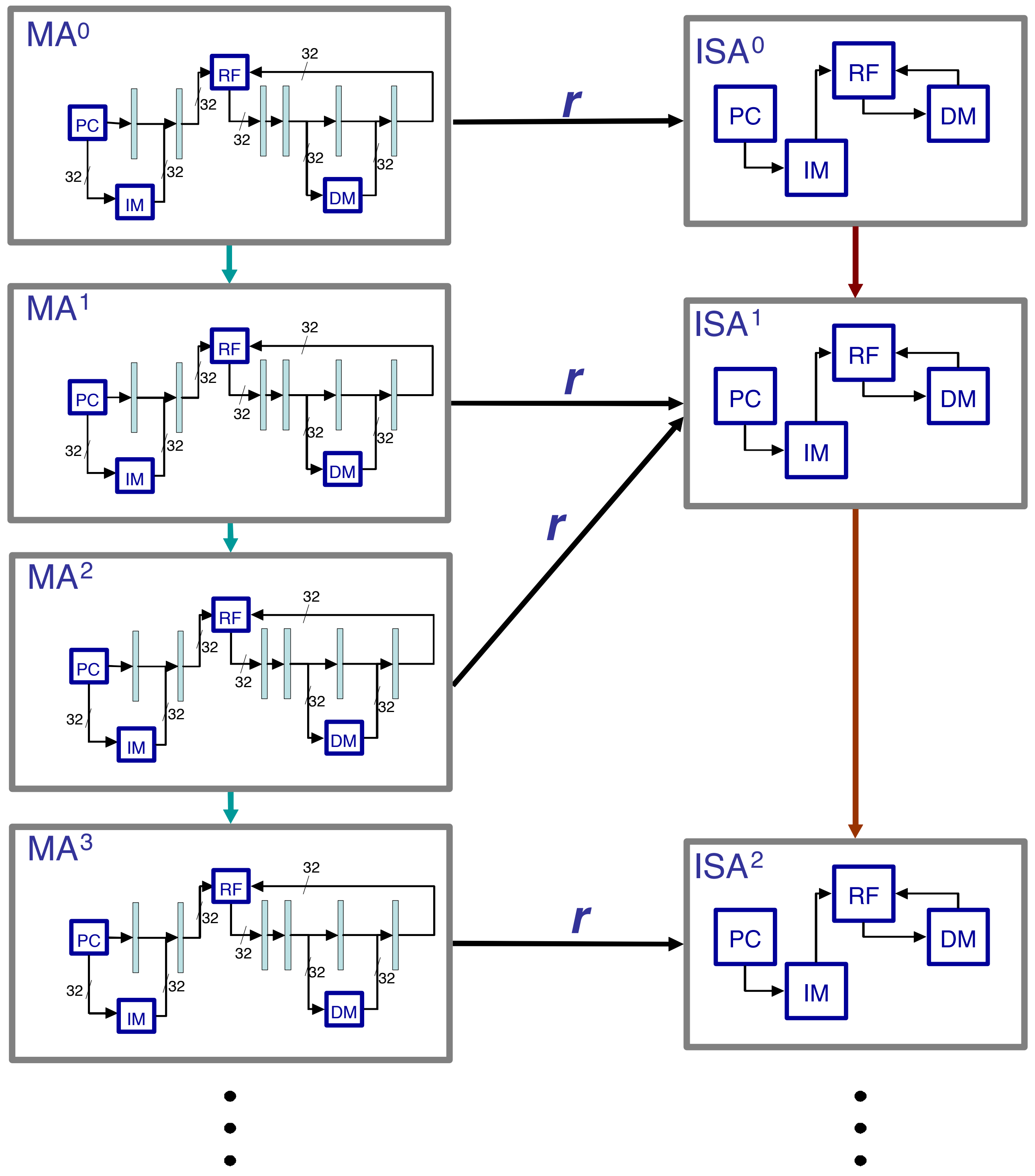
- Proving $\langle \forall \sigma{:}\mathit{fp}.\sigma.s : \langle \exists \delta{:}\mathit{fp}.\delta.w : \mathit{match}(B,\sigma,\delta)\rangle\rangle$ requires *global* reasoning; to do it mechanically is arduous.
- We define WFS and show it is equivalent to STS.
  - Soundness (easy).
  - Completeness (harder).
- Reasoning about WFS is *local.*
- The branching time result does not require machine closure, finite invisible nondeterminism, internal continuity, history variables, prophecy variables, etc.
- The linear time result only needs oracle variables.

Refinement via Local Reasoning

- Refinement requires matching infinite traces of the implementation and specification.
- Refinement maps allow a *local* check.

Technical overview:

- Define well-founded simulation & bisimulation.
- Local notions.
- Show soundness & completeness.



Well-founded Simulation (WFS)

1. $\langle \forall s,w \in S : sBw : L.s = L.w \rangle$
2. There are functions *rankt*: $S^2 \rightarrow W$, *rankl*: $S^3 \rightarrow \mathbb{N}$ such that $\langle W, \prec \rangle$ is well-founded and

$$\begin{tikzcd}
s \arrow[r] \arrow[d, dashed] & w \arrow[d, dashed] \\
u \arrow[r] & v
\end{tikzcd}$$

$$\langle \forall s,u,w \in S : (sBw \wedge s \dashrightarrow \qquad u) :$$
$$\langle \exists v : w \dashrightarrow \qquad v : \; uBv \rangle$$

Well-founded Simulation (WFS)

1. $\langle \forall s,w \in S : sBw : L.s = L.w \rangle$
2. There are functions *rankt*: $S^2 \rightarrow W$, *rankl*: $S^3 \rightarrow \mathbb{N}$ such that $\langle W, \prec \rangle$ is well-founded and

$$\begin{tikzcd}
s \arrow[r] \arrow[d, dashed] & w \arrow[d, dashed] & & s \arrow[r] \arrow[d, dashed] & w \\
u \arrow[r] & v & \text{or} & u \arrow[ur] &
\end{tikzcd}$$

$rankt(u,w) \prec rankt(s,w)$

$\langle \forall s,u,w \in S : (sBw \wedge s \dashrightarrow \quad u) :$

$\quad \langle \exists v : w \dashrightarrow \quad v : uBv \rangle$

$\quad \vee \ (uBw \ \wedge \ rankt(u,w) \prec rankt(s,w))$

Well-founded Simulation (WFS)

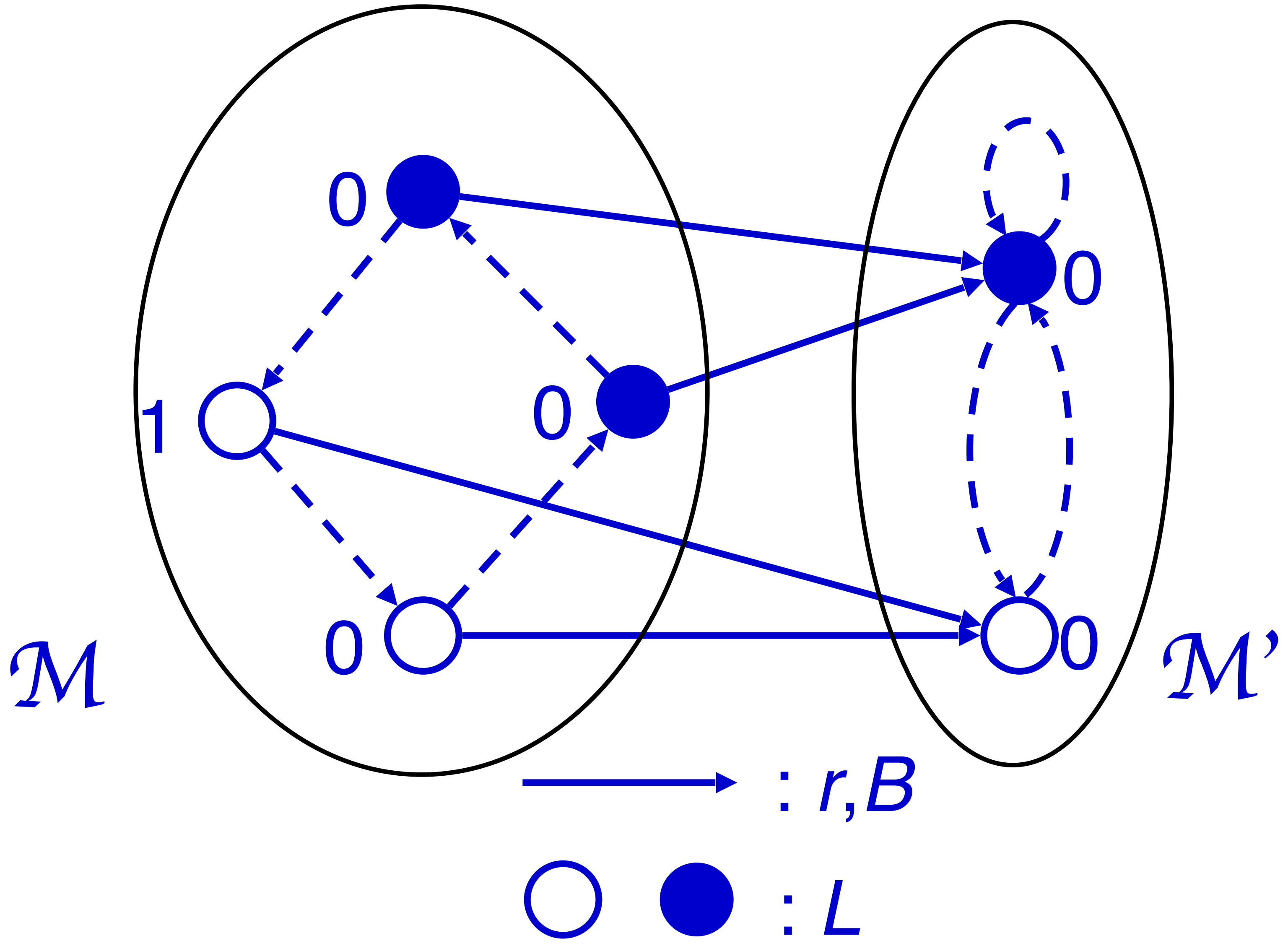
1. $\langle \forall s,w \in S : sBw : L.s = L.w \rangle$
2. There are functions *rankt*: $S^2 \rightarrow W$, *rankl*: $S^3 \rightarrow \mathbb{N}$ such that $\langle W, \prec \rangle$ is well-founded and

$$\begin{tikzcd} s \arrow[r] \arrow[d, dashed] & w \arrow[d, dashed] \\ u \arrow[r] & v \end{tikzcd} \quad \text{or} \quad \begin{tikzcd} s \arrow[r] \arrow[d, dashed] & w \\ u \arrow[ur] & \end{tikzcd} \quad \text{or} \quad \begin{tikzcd} s \arrow[r] \arrow[d, dashed] \arrow[dr] & w \arrow[d, dashed] \\ u & v \end{tikzcd}$$

$rankt(u,w) \prec rankt(s,w)$ $\quad$ $rankl(v,s,u) < rankl(w,s,u)$

$$\begin{aligned} &\langle \forall s,u,w \in S : (sBw \wedge s \dashrightarrow u) : \\ &\qquad \langle \exists v : w \dashrightarrow v : uBv \rangle \\ &\qquad \vee\ (uBw \wedge rankt(u,w) \prec rankt(s,w)) \\ &\qquad \vee\ \langle \exists v : w \dashrightarrow v : sBv \wedge rankl(v,s,u) < rankl(w,s,u) \rangle \rangle \end{aligned}$$

An Example



$rankt(u,w)$ = tag of u

$rankl(v,s,u)$ = tag of v

STS is WFS

Proof Outline.

1. Define $rankt(s,w)$:

$tree(s,w)$ is defined to be the largest subtree of the computation tree rooted at s s.t. at every non-root node $\langle s, \ldots, x\rangle$, we have xBw and $\langle \forall v : w \dashrightarrow v : \neg(xBv)\rangle$

$$\begin{tikzcd} s \arrow[r] \arrow[d, dashed] & w \arrow[d, dashed] \\ u \arrow[r] & v \end{tikzcd}$$

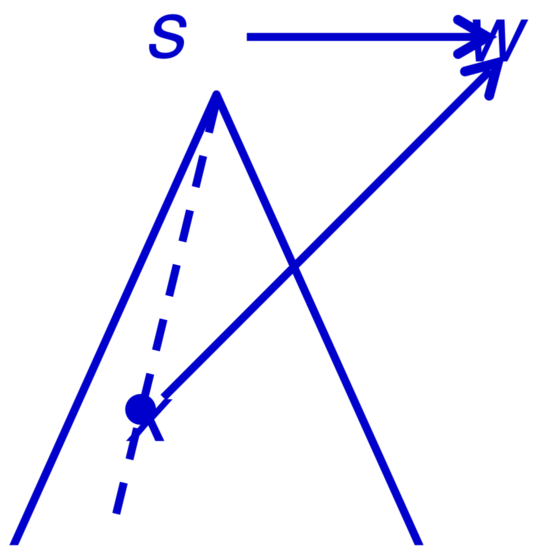
$$\begin{tikzcd} s \arrow[r] \arrow[d, dashed] & w \\ u \arrow[ur] & \end{tikzcd}$$

$rankt(u,w) < rankt(s,w)$

$$\begin{tikzcd} s \arrow[r] \arrow[d, dashed] \arrow[dr] & w \arrow[d, dashed] \\ u & v \end{tikzcd}$$

$rankl(v,s,u) < rankl(w,s,u)$

$$\begin{aligned} &\langle \forall s,u,w \in S : (sBw \wedge s \dashrightarrow u) : \\ &\quad \langle \exists v : w \dashrightarrow v : uBv\rangle \\ &\quad \vee\ (uBw \wedge rankt(u,w) < rankt(s,w)) \\ &\quad \vee\ \langle \exists v : w \dashrightarrow v : sBv \wedge rankl(v,s,u) < rankl(w,s,u)\rangle\rangle \end{aligned}$$

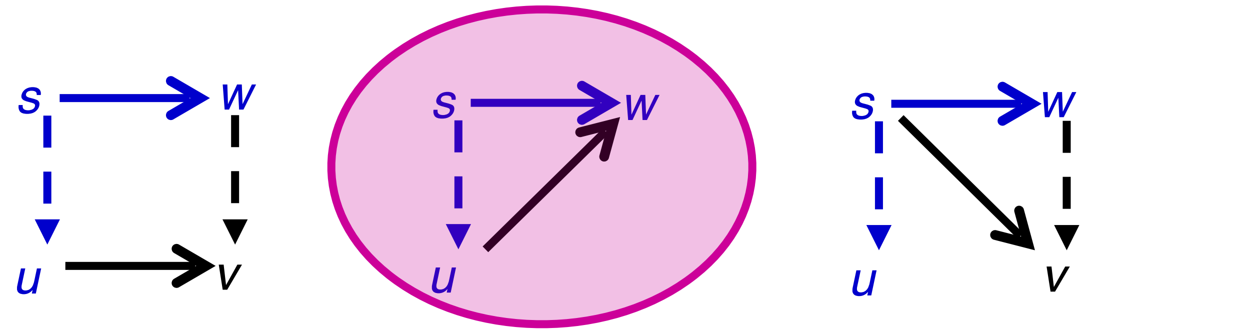


STS is WFS

Proof Outline.

1. Define *rankt*(*s*,*w*):

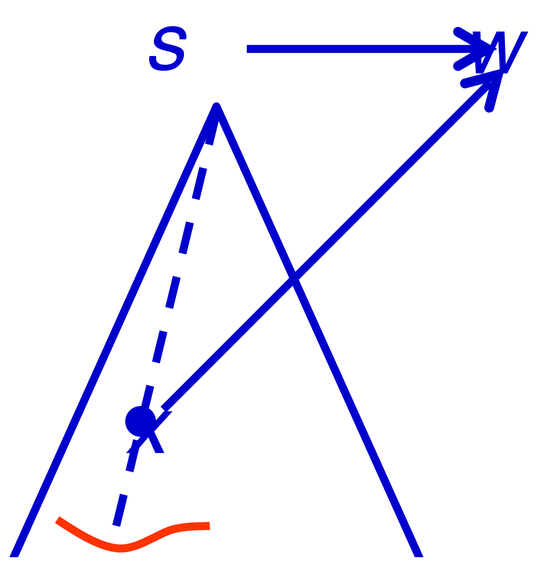
tree(*s*,*w*) is defined to be the largest subtree of the computation tree rooted at *s* s.t. at every non-root node $\langle s, \ldots, x\rangle$, we have xBw and $\langle \forall v : w \dashrightarrow v : \neg(xBv)\rangle$



$rankt(u,w) < rankt(s,w)$ $\quad$ $rankl(v,s,u) < rankl(w,s,u)$

$$\langle \forall s,u,w \in S : (sBw \wedge s \dashrightarrow u) :$$
$$\langle \exists v : w \dashrightarrow v : uBv\rangle$$
$$\vee\ (uBw \wedge rankt(u,w) < rankt(s,w)$$
$$\vee\ \langle \exists v : w \dashrightarrow v : sBv \wedge rankl(v,s,u) < rankl(w,s,u)\rangle\rangle$$

Lemma: Every path of *tree* is finite.



STS is WFS

Proof Outline.

1. Define *rankt*(s,w):

tree(s,w) is defined to be the largest subtree of the computation tree rooted at s s.t. at every non-root node $\langle s, \ldots, x\rangle$, we have xBw and $\langle \forall v : w \dashrightarrow v : \neg(xBv)\rangle$

$$\begin{tikzcd} s \arrow[r] \arrow[d, dashed] & w \arrow[d, dashed] \\ u \arrow[r] & v \end{tikzcd} \qquad \begin{tikzcd} s \arrow[r] \arrow[d, dashed] & w \\ u \arrow[ur] & \end{tikzcd} \qquad \begin{tikzcd} s \arrow[r] \arrow[d, dashed] \arrow[dr] & w \arrow[d, dashed] \\ u & v \end{tikzcd}$$

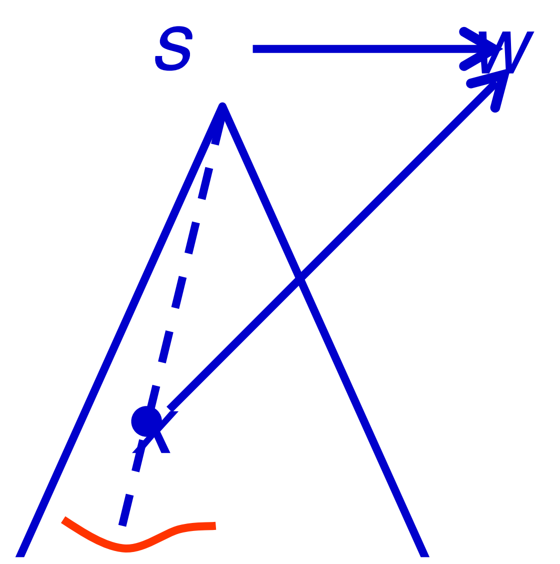
$rankt(u,w) < rankt(s,w)$ $\qquad$ $rankl(v,s,u) < rankl(w,s,u)$

$$\begin{aligned} &\langle \forall s,u,w \in S : (sBw \wedge s \dashrightarrow u) : \\ &\quad \langle \exists v : w \dashrightarrow v : uBv\rangle \\ &\quad \vee (uBw \wedge rankt(u,w) < rankt(s,w)) \\ &\quad \vee \langle \exists v : w \dashrightarrow v : sBv \wedge rankl(v,s,u) < rankl(w,s,u)\rangle\rangle \end{aligned}$$

Lemma: Every path of *tree* is finite.

Label nodes in *tree* using standard set-theory "rank" function.

Lemma: If $|S| \leq \kappa$ (where $\omega \leq \kappa$), then for all s,w∈S, tree(s,w) is labeled with an ordinal of cardinality $\leq \kappa$.

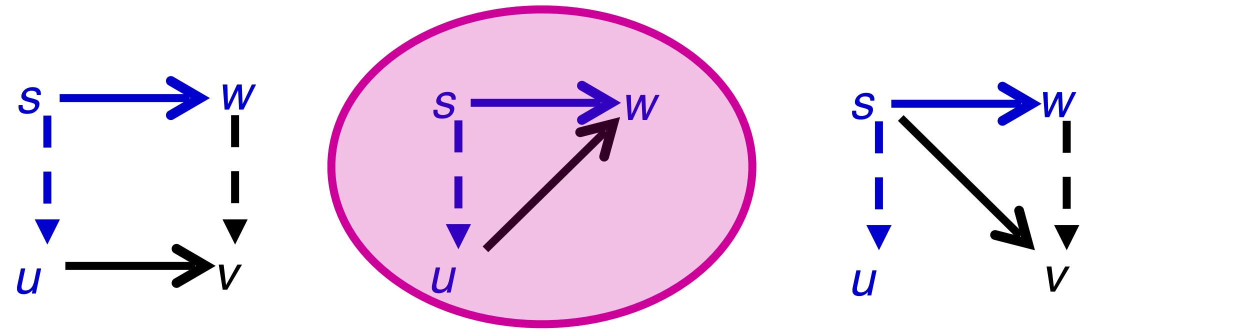


STS is WFS

Proof Outline.

1. Define *rankt*(*s,w*):

tree(*s,w*) is defined to be the largest subtree of the computation tree rooted at *s* s.t. at every non-root node ⟨*s*, …, *x*⟩, we have *xBw* and ⟨∀*v* : *w* ⇢ *v* : ¬(*xBv*)⟩



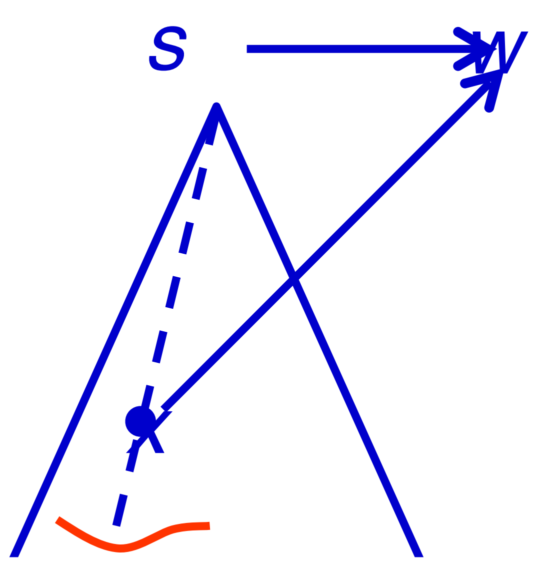
rankt(*u,w*) < *rankt*(*s,w*) *rankl*(*v,s,u*) < *rankl*(*w,s,u*)

⟨∀*s,u,w*∈ *S* : (*sBw* ∧ *s* ⇢ *u*) :
  ⟨∃*v* : *w* ⇢ *v* : *uBv*⟩
 ∨ (*uBw* ∧ *rankt*(*u,w*) < *rankt*(*s,w*)
 ∨ ⟨∃*v* : *w* ⇢ *v* : *sBv* ∧ *rankl*(*v,s,u*) < *rankl*(*w,s,u*)⟩⟩

Lemma: Every path of *tree* is finite.

Label nodes in *tree* using standard set-theory "rank" function.

Lemma: If |*S*| ≤ κ (where ω ≤ κ), then for all s,w∈S, tree(s,w) is labeled with an ordinal of cardinality ≤ κ.



Def: rankt = l.tree, where ⟨*W*, <⟩ is ⟨(|*S*|+ ω)$^+$, <⟩.

Existence of Refinement Maps

- Branching time:

  Theorem: If $\mathcal{I}$ implements $\mathcal{S}$, then there is a refinement map r such that $\mathcal{I} \sqsubseteq_r \mathcal{S}$.

- Linear time:

  Theorem: If $\mathcal{I}$ implements $\mathcal{S}$ (the set of traces of $\mathcal{I}$ is a subset of the traces of $\mathcal{S}$), then there exists $\mathcal{I}'$, where $\mathcal{I}'$ is obtained from $\mathcal{I}$ by adding an oracle variable, and a refinement map r such that $\mathcal{I}' \sqsubseteq_r \mathcal{S}$.