

Dividing and Conquering Logic

Eyal Amir

Computer Science Department

Stanford University

Goals

- Build large knowledge bases.
- Reason with large structured logical theories.
- Real-world reactive agents that use commonsense.

Contents

1. Reasoning algorithms for logical partitions.
2. Automatic partitioning of logical theories.
3. Object-Oriented design in logic.
4. Reactive control with subsumption of partitions.

An Espresso Machine and its Parts



A breakdown of the process of making an espresso.

A partitioning and its intersection graph

A

$\neg ok_pump \vee \neg on_pump \vee water$
 $\neg man_fill \vee water$
 $\neg man_fill \vee \neg on_pump$
 $man_fill \vee on_pump$
 $\neg water \vee \neg ok_boiler \vee \neg on_boiler$
 $\vee steam$
 $water \vee \neg steam$
 $ok_boiler \vee \neg steam$
 $on_boiler \vee \neg steam$
 $\neg steam \vee \neg coffee \vee hot_drink$
 $coffee \vee teabag$
 $\neg steam \vee \neg teabag \vee hot_drink$



Axioms describing espresso making.

A partitioning and its intersection graph

A

$\neg ok_pump \vee \neg on_pump \vee water$
 $\neg man_fill \vee water$
 $\neg man_fill \vee \neg on_pump$
 $man_fill \vee on_pump$
 $\neg water \vee \neg ok_boiler \vee \neg on_boiler$
 $\vee steam$
 $water \vee \neg steam$
 $ok_boiler \vee \neg steam$
 $on_boiler \vee \neg steam$
 $\neg steam \vee \neg coffee \vee hot_drink$
 $coffee \vee teabag$
 $\neg steam \vee \neg teabag \vee hot_drink$

A_1

(1) $\neg ok_pump \vee \neg on_pump$
 $\vee water$
 (2) $\neg man_fill \vee water$
 (3) $\neg man_fill \vee \neg on_pump$
 (4) $man_fill \vee on_pump$

A_2

(5) $\neg water \vee \neg ok_boiler \vee$
 $\neg on_boiler \vee steam$
 (6) $water \vee \neg steam$
 (7) $ok_boiler \vee \neg steam$
 (8) $on_boiler \vee \neg steam$

A_3

(9) $\neg steam \vee \neg coffee \vee hot_drink$
 (10) $coffee \vee teabag$
 (11) $\neg steam \vee \neg teabag \vee hot_drink$

$water$

$steam$

A partitioning of A and its intersection graph.

A Message-Passing Proof

Proving *hot_drink* after asserting *ok_pump* (12) in \mathcal{A}_1 and *ok_boiler* (13), *on_boiler* (14) in \mathcal{A}_2 .

\mathcal{A}_1



water

\mathcal{A}_2



steam

\mathcal{A}_3



A Message-Passing Proof

Proving *hot_drink* after asserting *ok_pump* (12) in \mathcal{A}_1 and *ok_boiler* (13), *on_boiler* (14) in \mathcal{A}_2 .

\mathcal{A}_1

- (1) $\neg ok_pump \vee \neg on_pump \vee water$
- (2) $\neg man_fill \vee water$
- (3) $\neg man_fill \vee \neg on_pump$
- (4) $man_fill \vee on_pump$

Resolve

Generating

- (2),(4) $on_pump \vee water$ (m1)
- (m1),(1) $\neg ok_pump \vee water$ (m2)
- (m2),(12) $water$ (m3)

clause *water* passed from \mathcal{A}_1 to \mathcal{A}_2

\mathcal{A}_2

water



\mathcal{A}_3

steam



A Message-Passing Proof

Proving *hot_drink* after asserting *ok_pump* (12) in \mathcal{A}_1 and *ok_boiler* (13), *on_boiler* (14) in \mathcal{A}_2 .

\mathcal{A}_1



water

\mathcal{A}_2

- (5) $\neg water \vee \neg ok_boiler \vee \neg on_boiler \vee steam$
- (6) $water \vee \neg steam$
- (7) $ok_boiler \vee \neg steam$
- (8) $on_boiler \vee \neg steam$

\mathcal{A}_3

steam



Resolve

Generating

clause *water* passed from \mathcal{A}_1 to \mathcal{A}_2

(m3),(5) $ok_boiler \wedge on_boiler \Rightarrow steam$ (m4)

(m4),(13) $\neg on_boiler \vee steam$ (m5)

(m5),(14) *steam* (m6)

clause *steam* passed from \mathcal{A}_2 to \mathcal{A}_3

A Message-Passing Proof

Proving *hot_drink* after asserting *ok_pump* (12) in \mathcal{A}_1 and *ok_boiler* (13), *on_boiler* (14) in \mathcal{A}_2 .

\mathcal{A}_1



water

\mathcal{A}_2



Resolve

Generating

clause steam passed from \mathcal{A}_2 to \mathcal{A}_3

steam

\mathcal{A}_3

(9) $\neg steam \vee \neg coffee \vee hot_drink$
 (10) $coffee \vee teabag$
 (11) $\neg steam \vee \neg teabag \vee hot_drink$

(9),(10) $\neg steam \vee teabag \vee hot_drink$ (m7)
 (m7),(11) $\neg steam \vee hot_drink$ (m8)
 (m8),(m6) *hot_drink* (m9)

A Message-Passing Proof

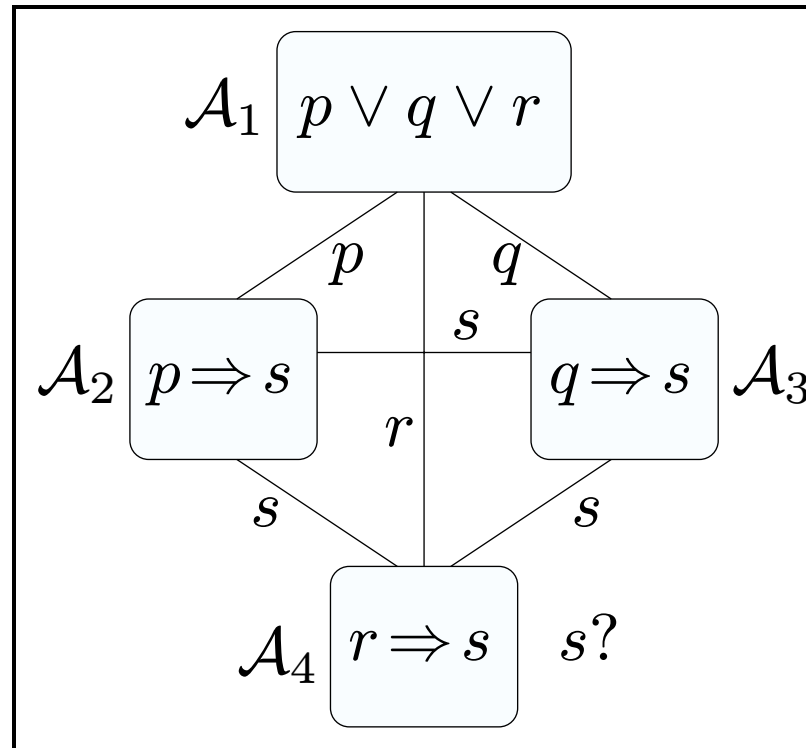
Proving *hot_drink* after asserting *ok_pump* (12) in \mathcal{A}_1 and *ok_boiler* (13), *on_boiler* (14) in \mathcal{A}_2 .

\mathcal{A}_1	Resolve	Generating	
(1) $\neg ok_pump \vee \neg on_pump$ $\vee water$	(2),(4)	$on_pump \vee water$	(m1)
(2) $\neg man_fill \vee water$	(m1),(1)	$\neg ok_pump \vee water$	(m2)
(3) $\neg man_fill \vee \neg on_pump$	(m2),(12)	$water$	(m3)
(4) $man_fill \vee on_pump$			
\mathcal{A}_2		$water$	
(5) $\neg water \vee \neg ok_boiler \vee$ $\neg on_boiler \vee steam$	(m3),(5)	$ok_boiler \wedge on_boiler$ $\Rightarrow steam$	(m4)
(6) $water \vee \neg steam$	(m4),(13)	$\neg on_boiler \vee steam$	(m5)
(7) $ok_boiler \vee \neg steam$	(m5),(14)	$steam$	(m6)
(8) $on_boiler \vee \neg steam$			
\mathcal{A}_3		$steam$	
(9) $\neg steam \vee \neg coffee \vee hot_drink$	(9),(10)	$\neg steam \vee teabag$ $\vee hot_drink$	(m7)
(10) $coffee \vee teabag$	(m7),(11)	$\neg steam \vee hot_drink$	(m8)
(11) $\neg steam \vee \neg teabag \vee hot_drink$	(m8),(m6)	hot_drink	(m9)

clause *water* passed from \mathcal{A}_1 to \mathcal{A}_2

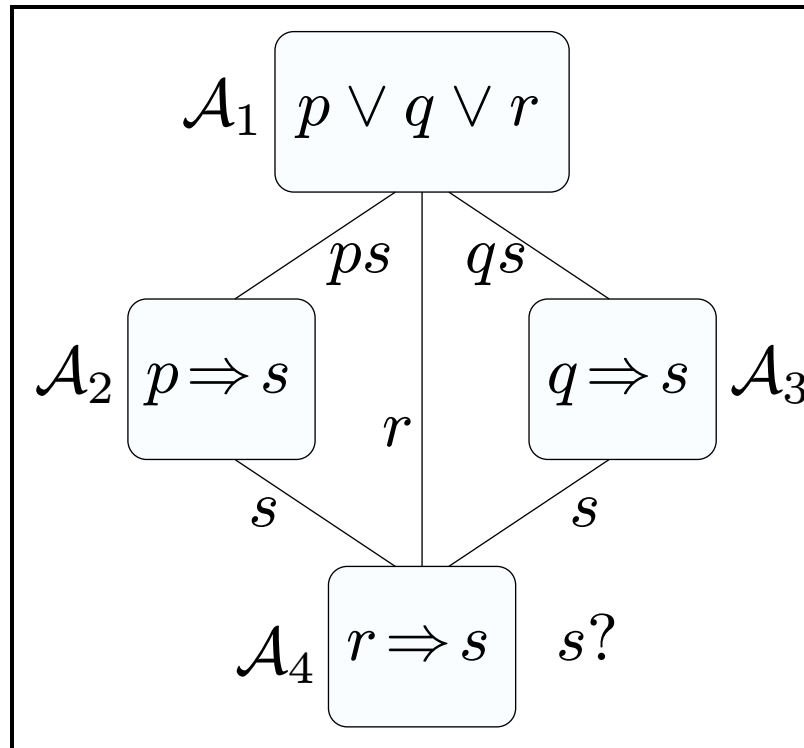
clause *steam* passed from \mathcal{A}_2 to \mathcal{A}_3

Graphs with Cycles



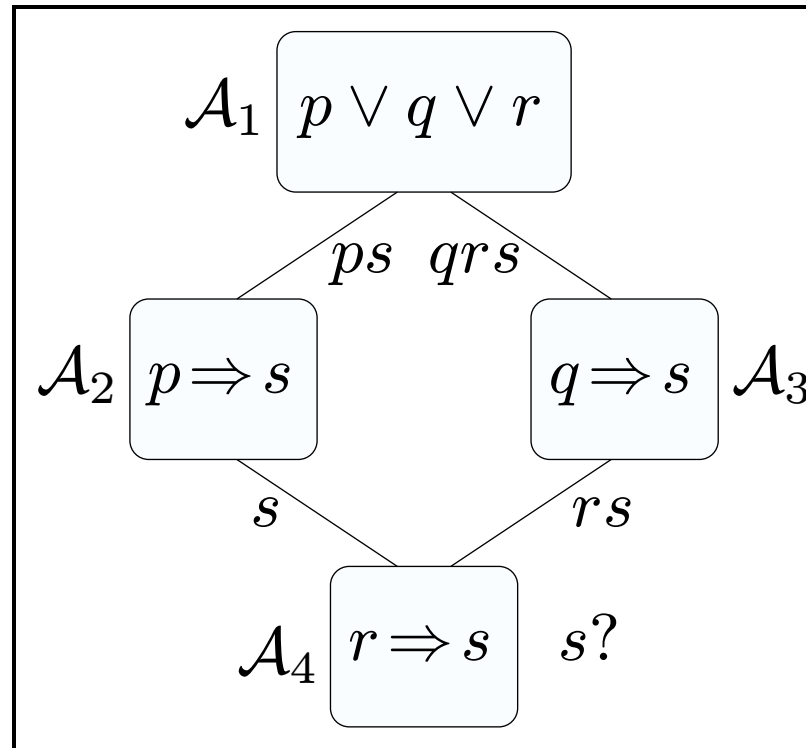
An intersection graph

Graphs with Cycles



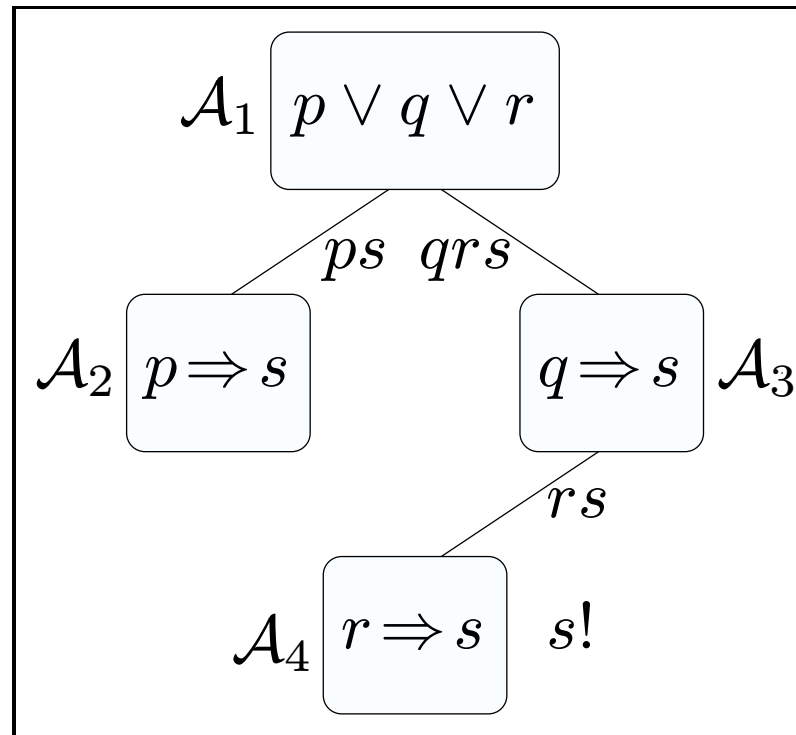
Applying BREAK-CYCLES.

Graphs with Cycles



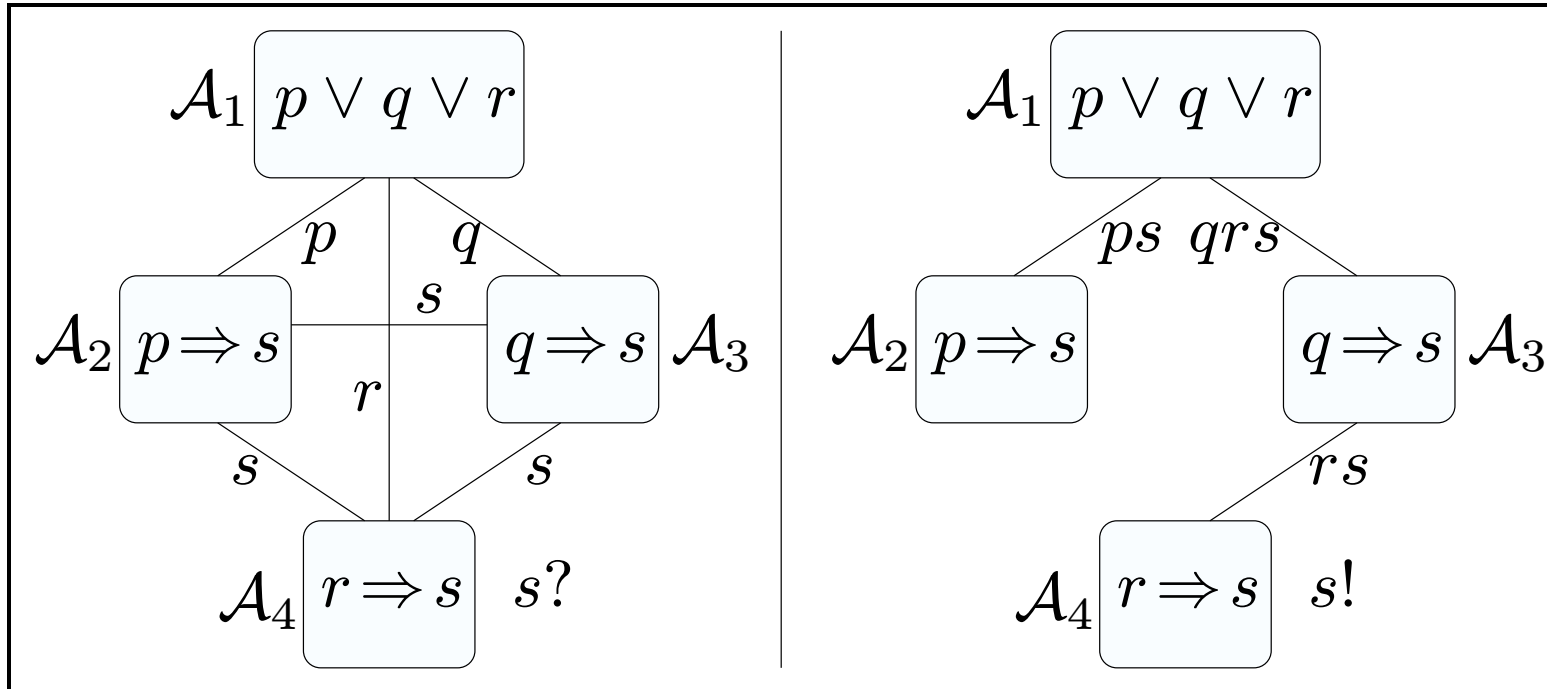
Applying BREAK-CYCLES.

Graphs with Cycles



Applying BREAK-CYCLES.

Graphs with Cycles



An intersection graph before (left) and after (right)
applying BREAK-CYCLES.

Soundness and Completeness of MP

Craig's Interpolation Theorem: *If $\alpha \vdash \beta$, then there is a formula γ involving only symbols common to both α and β , such that $\alpha \vdash \gamma$ and $\gamma \vdash \beta$.*

Theorem 1 *Message-Passing with Break-Cycles is sound and complete, if the reasoning procedure in each partition is complete for consequence finding.*

(e.g., $\alpha = \mathcal{A}_1, \beta = \mathcal{A}_2 \Rightarrow Q$)

Benefits of Message-Passing

- Search space is restricted.
- Allows parallel processing.
- Sound and complete.
- Can use different reasoners for each partition.
- Small links imply short proofs.
- Small partitions imply short proofs.

A Word of Caution

- Query contained in the language of a partition.
- Partitions graph must be converted to a tree.
- Reasoning procedures in each partition:
 - Linear resolution
 - Symbol ordering: directed resolution, A-ordering, lock resolution
 - Set-of-Support and Semantic resolution
 - \mathcal{L} -prime implicate finders, incremental \mathcal{L} -prime implicate finders.

Propositional SAT for Partitioned Theories

Procedure Linear-Partition-SAT:

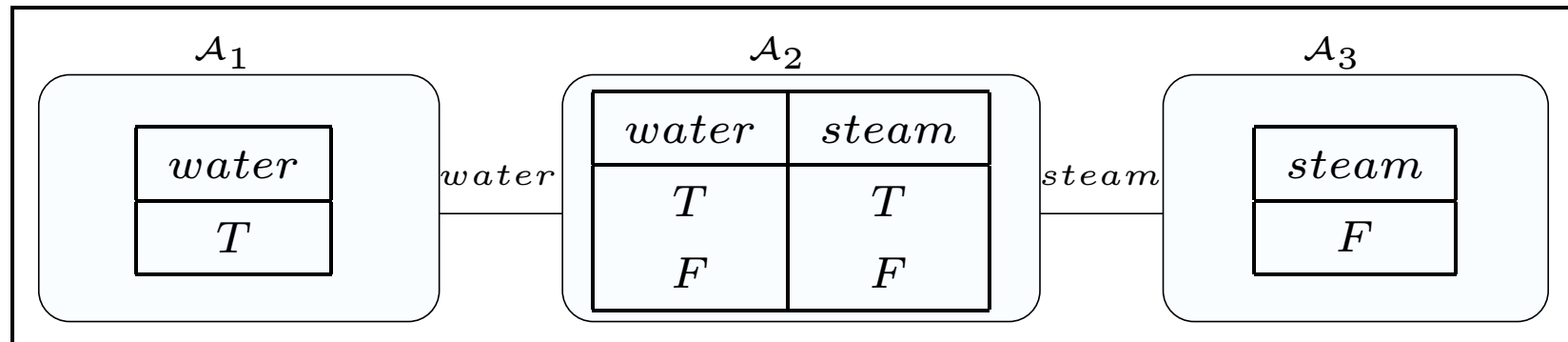
1. For every partition, for every truth assignment to the links, perform SAT. Store results in a table.
2. Join the tables in an MP fashion. If at any point the table is empty, there is no model.



Propositional SAT for Partitioned Theories

Procedure Linear-Partition-SAT:

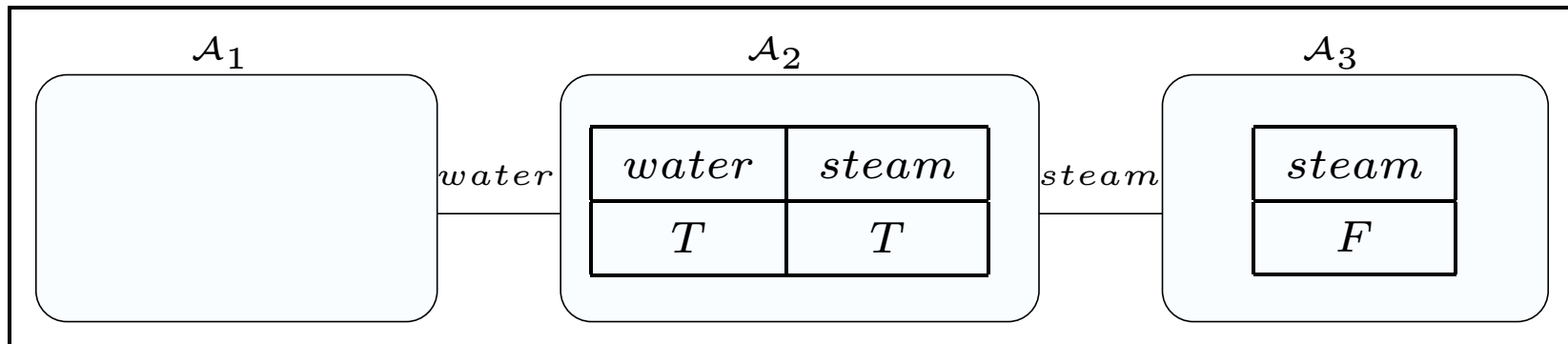
1. For every partition, for every truth assignment to the links, perform SAT. Store results in a table.
2. Join the tables in an MP fashion. If at any point the table is empty, there is no model.



Propositional SAT for Partitioned Theories

Procedure Linear-Partition-SAT:

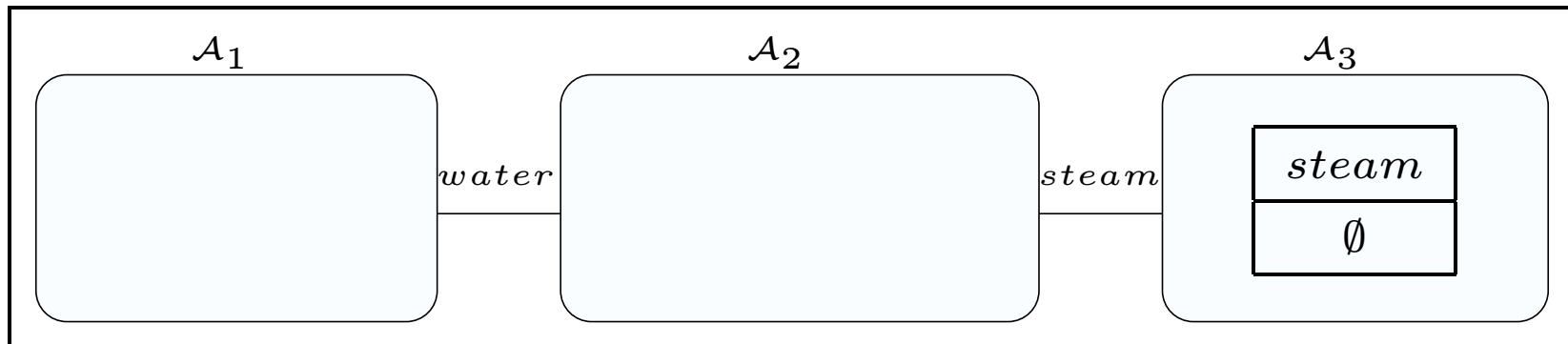
1. For every partition, for every truth assignment to the links, perform SAT. Store results in a table.
2. Join the tables in an MP fashion. If at any point the table is empty, there is no model.



Propositional SAT for Partitioned Theories

Procedure Linear-Partition-SAT:

1. For every partition, for every truth assignment to the links, perform SAT. Store results in a table.
2. Join the tables in an MP fashion. If at any point the table is empty, there is no model.



Computational Analysis

Linear-Partition-SAT takes time

- **linear** in the number of partitions.
- **exponential** in the size of the largest partition.
- $Time(k, n) = O(n * 2^k)$
 - k propositions in largest partition
 - n partitions

Summary: Reasoning with Partitions

- Multiple Knowledge Bases that share contents.
- Reasoning with FOL and Propositional Logic.
- Different reasoning procedures for different partitions.
- Restricted overall search space.
- Parallel reasoning.

Contents

1. Reasoning algorithms for logical partitions.
2. **Automatic partitioning of logical theories.**
3. Object-Oriented design in logic.
4. Reactive control with subsumption of partitions.

A Good Partitioning

A tree of partitions with minimum

- number of axioms and symbols in each partition (dominated by the largest partition).
- number of symbols contained in all links to/from each node.

Partitioning the Symbols Graph



An espresso machine

Partitioning the Symbols Graph

A

$\neg ok_pump \vee \neg on_pump \vee water$

$\neg man_fill \vee \neg on_pump$

$\neg water \vee \neg ok_boiler \vee \neg on_boiler \vee steam$

$ok_boiler \vee \neg steam$

$\neg steam \vee \neg coffee \vee hot_drink$

$\neg steam \vee \neg teabag \vee hot_drink$

$\neg man_fill \vee water$

$man_fill \vee on_pump$

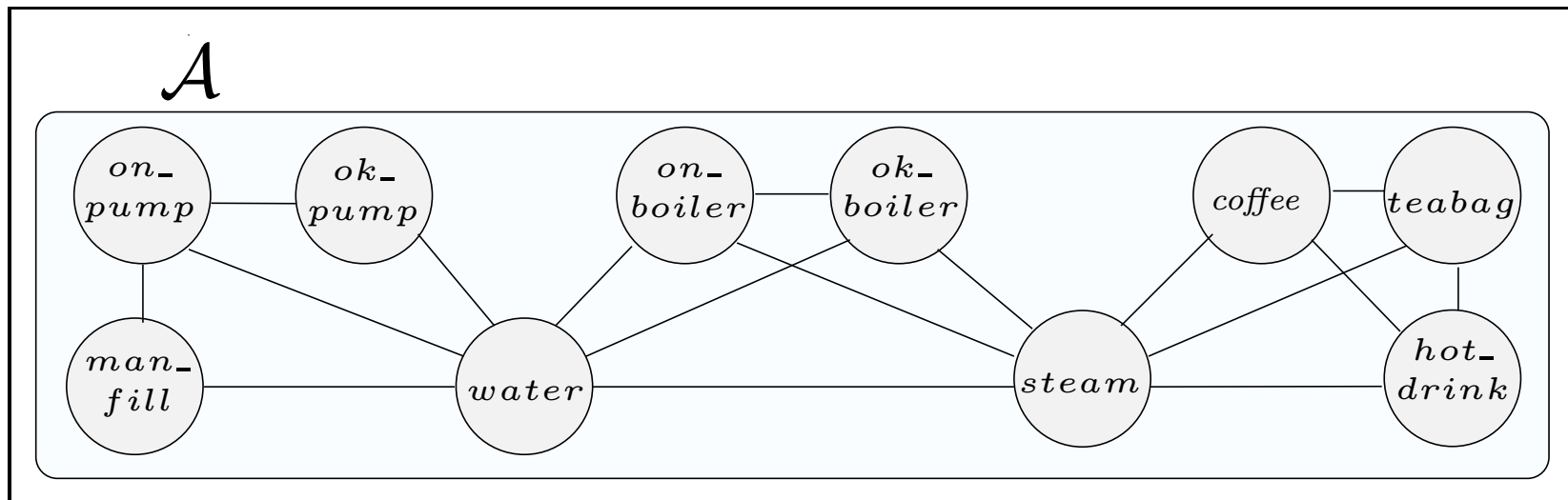
$water \vee \neg steam$

$on_boiler \vee \neg steam$

$coffee \vee teabag$

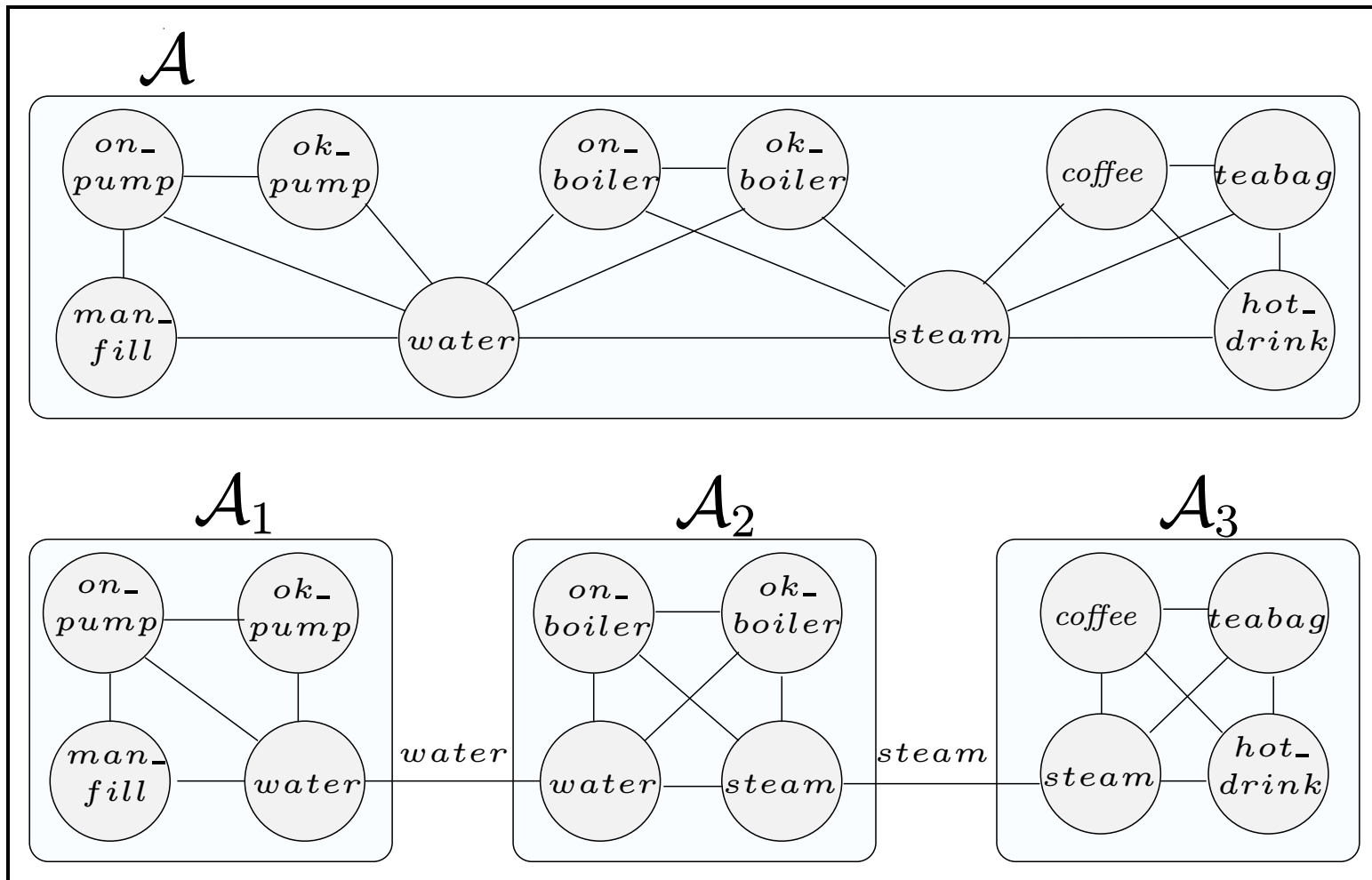
An espresso machine's theory, *A*

Partitioning the Symbols Graph



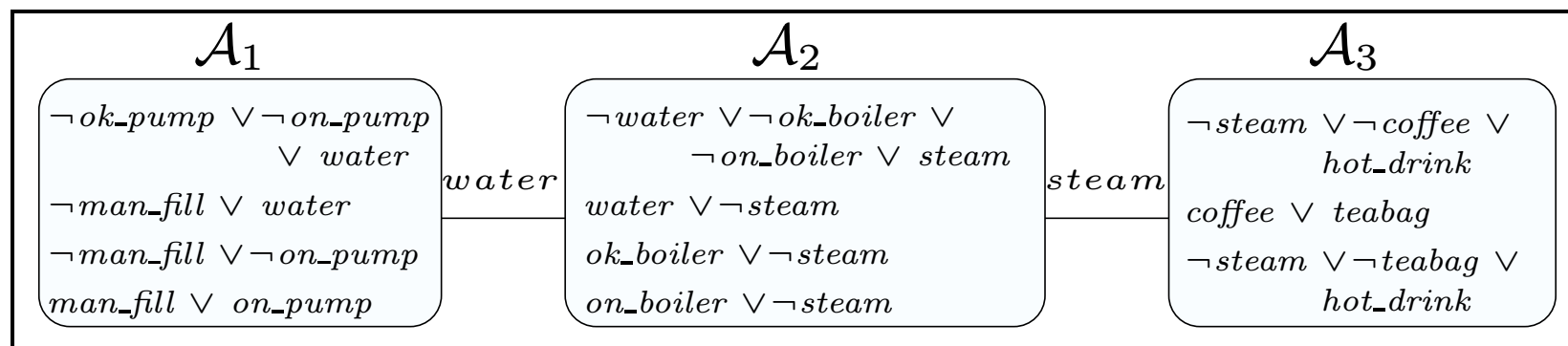
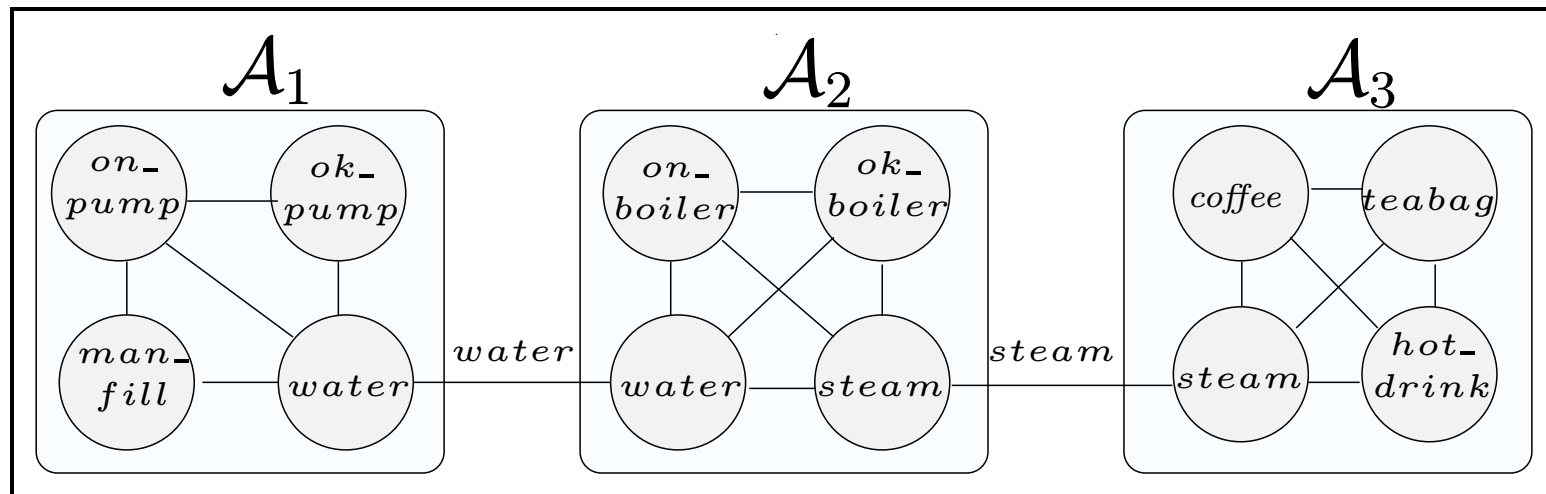
Theory *A*'s symbols graph

Partitioning the Symbols Graph



Finding a *Tree Decomposition* of *A*'s symbols graph.

Partitioning the Symbols Graph



Optimal partitioning

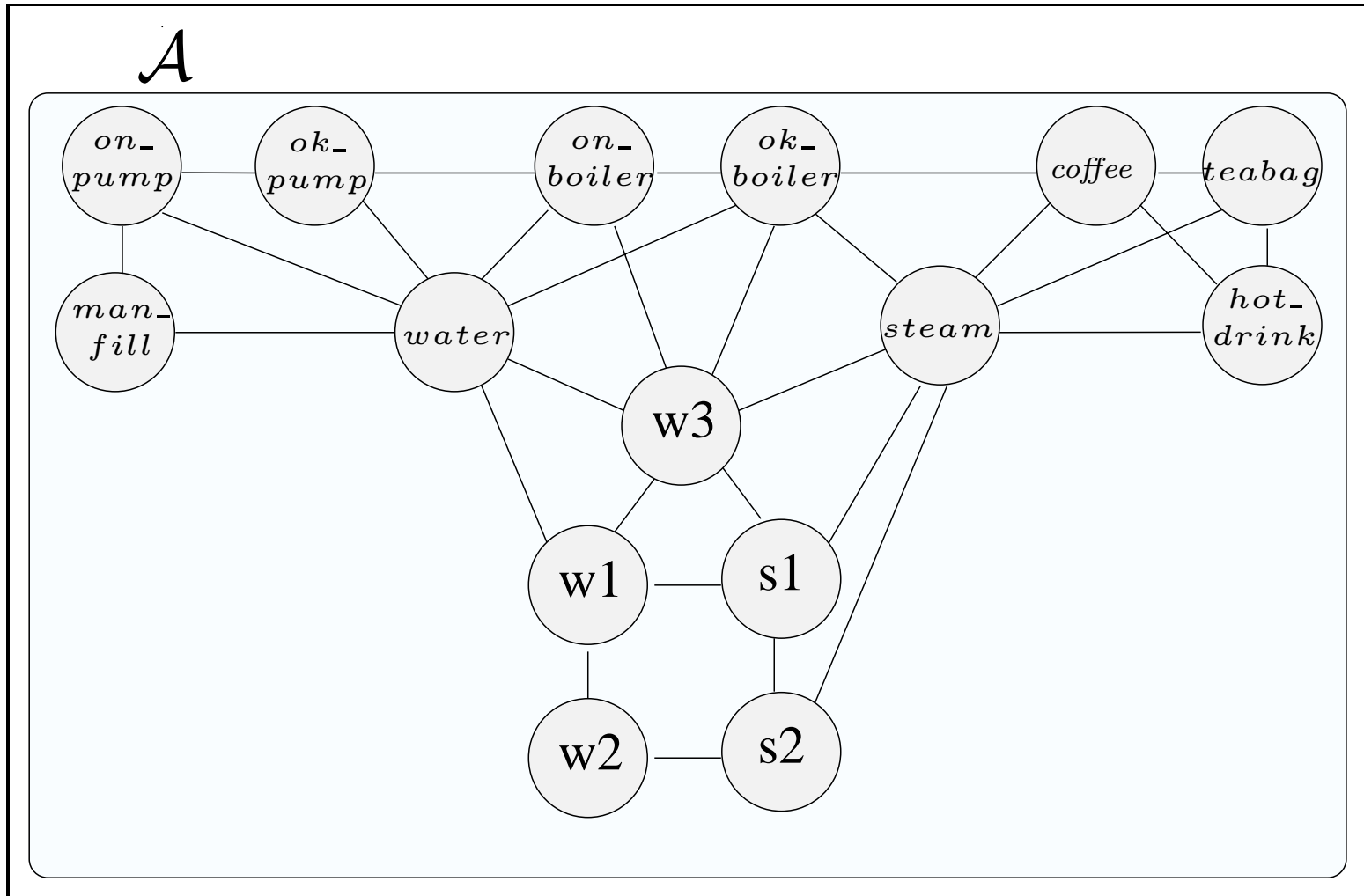
- Equivalent to Tree Decomposition, and Triangulation of minimum TREEWIDTH.
- Finding optimal partitioning is NP-hard (Arnborg et al. 1987).
- Can the optimal be approximated with constant factor in polynomial time? – Open question.
- Many partitioning algorithms so far: E.g., (Bodlaender 1996), (Becker & Geiger 1996), (Shoikhet & Geiger 1997).

Our New Graph Partitioning Algorithms

- Current algorithms cannot decompose large graphs ($|V| > 100$ or treewidth > 10).
- Our algorithms recursively break the graph into 2 or 3 parts.
 1. 2-way separators: factor-4 and factor- $4\frac{1}{2}$ approximations.
 2. 3-way separators: factor- $3\frac{2}{3}$ and factor- $O(\lg k)$ approximations.

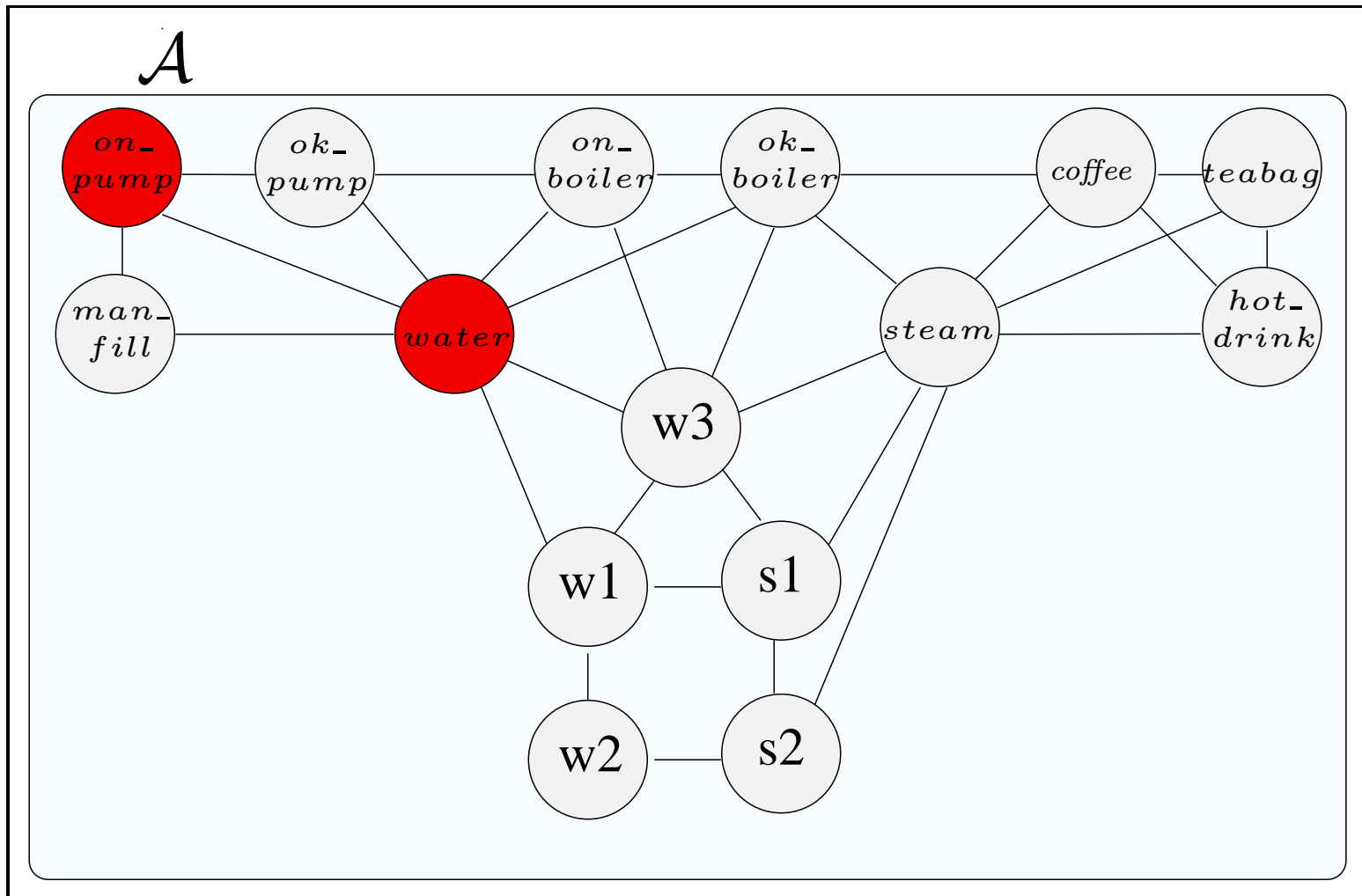
* $n = |V|$ and k is the actual treewidth.

Automated Partitioning using *triangulate*



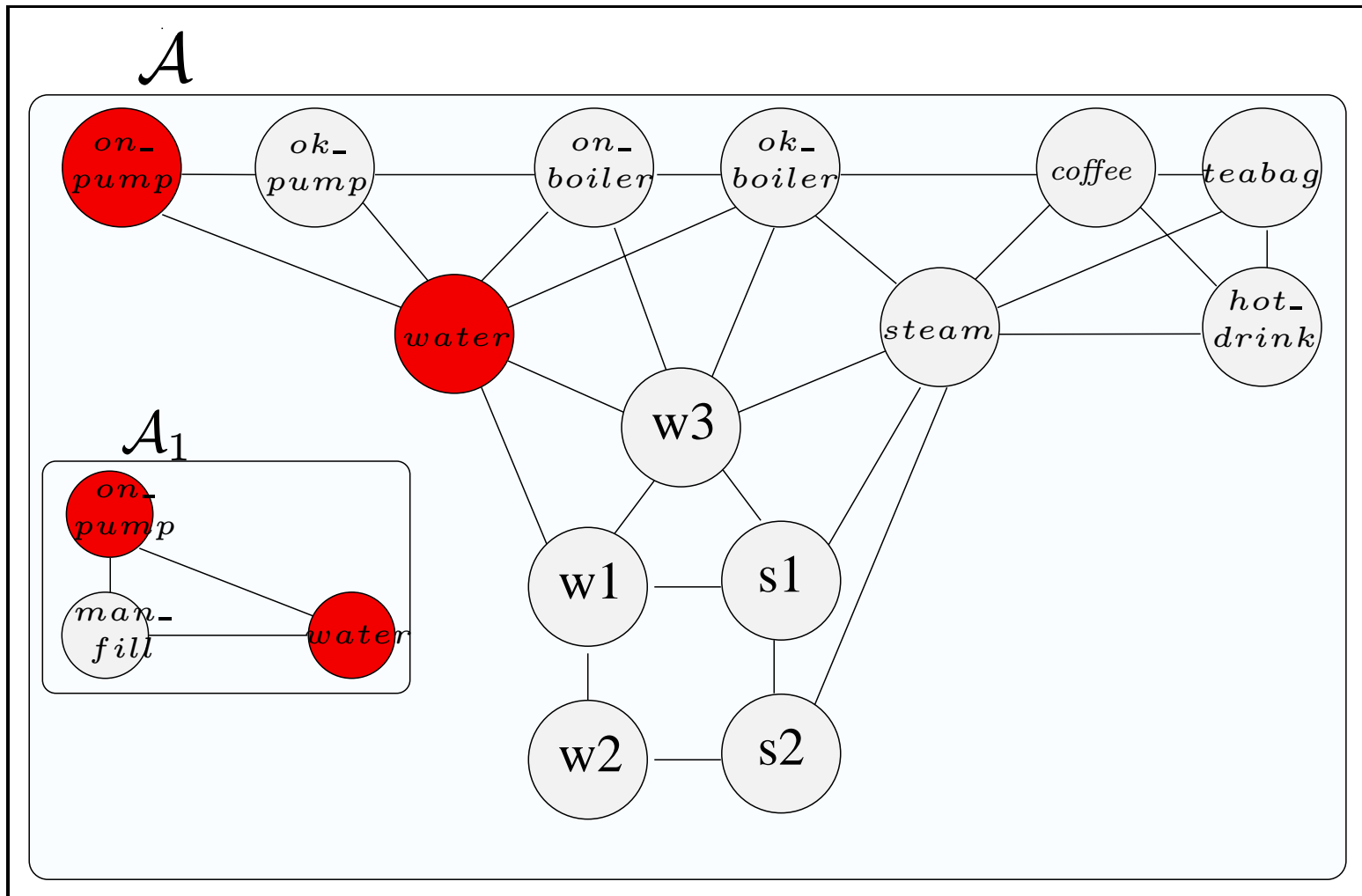
Using a *balanced separator* of previous separators

Automated Partitioning using *triangulate*



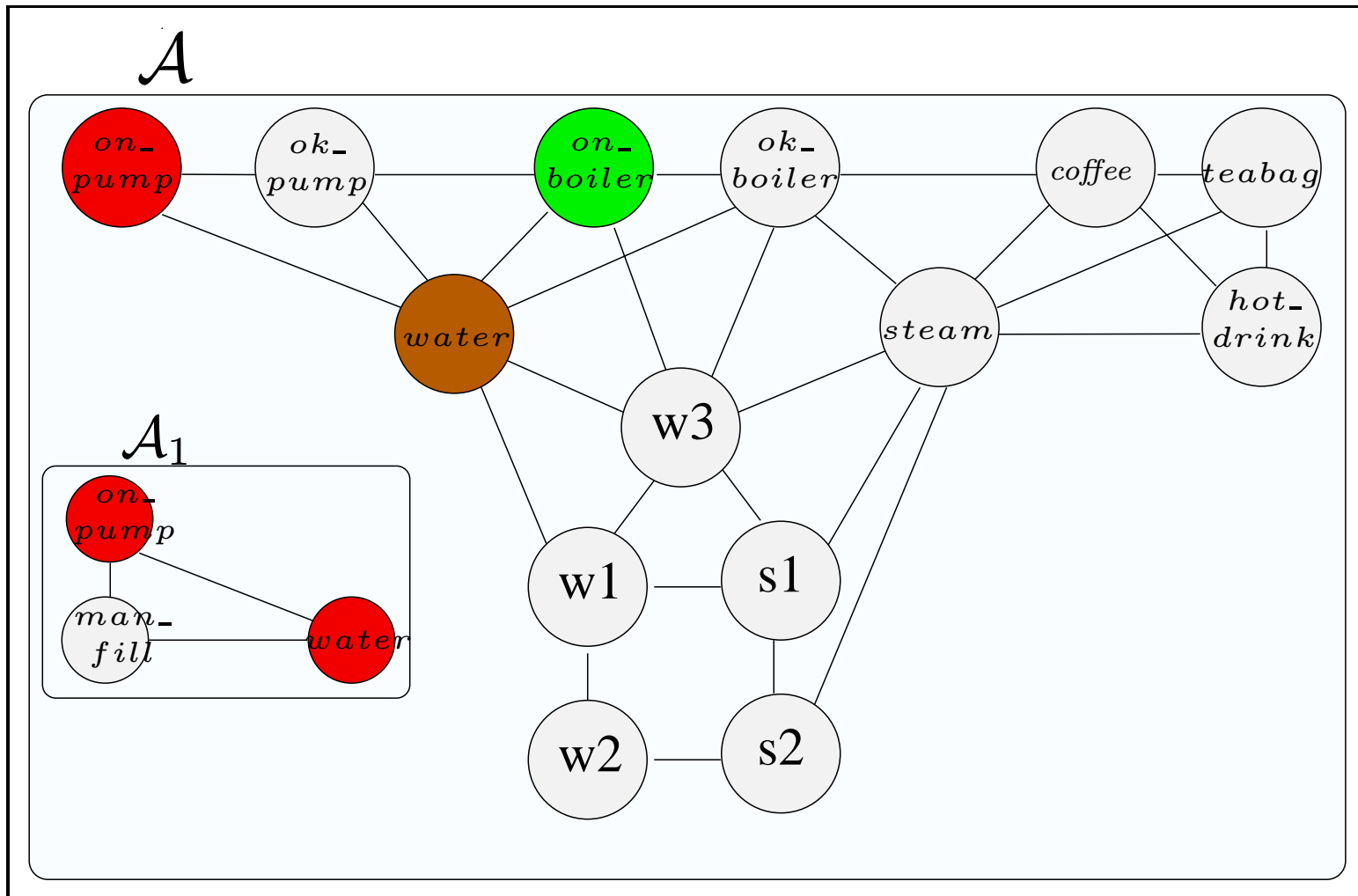
Using a *balanced separator* of previous separators

Automated Partitioning using *triangulate*



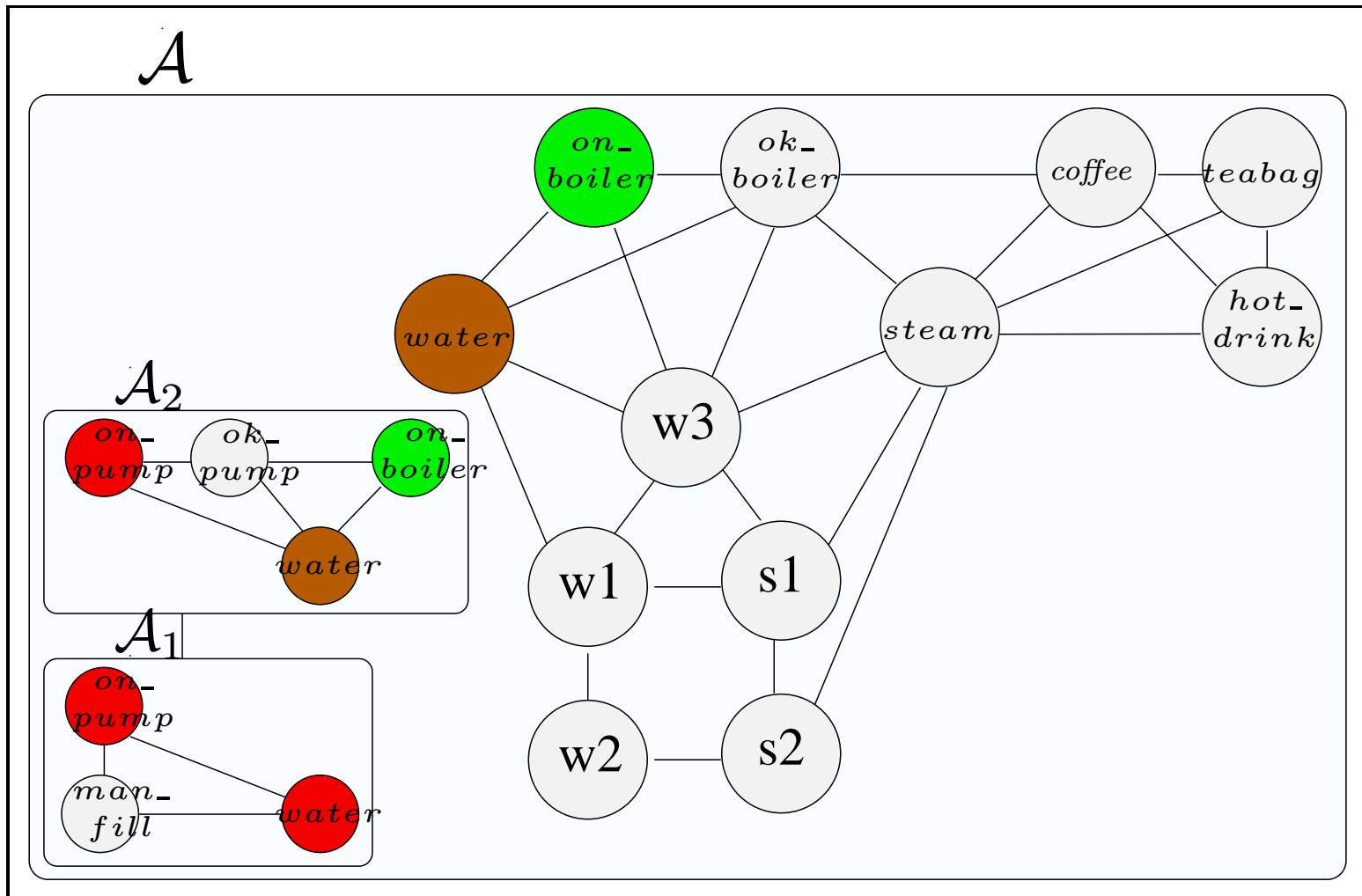
Using a *balanced separator* of previous separators

Automated Partitioning using *triangulate*



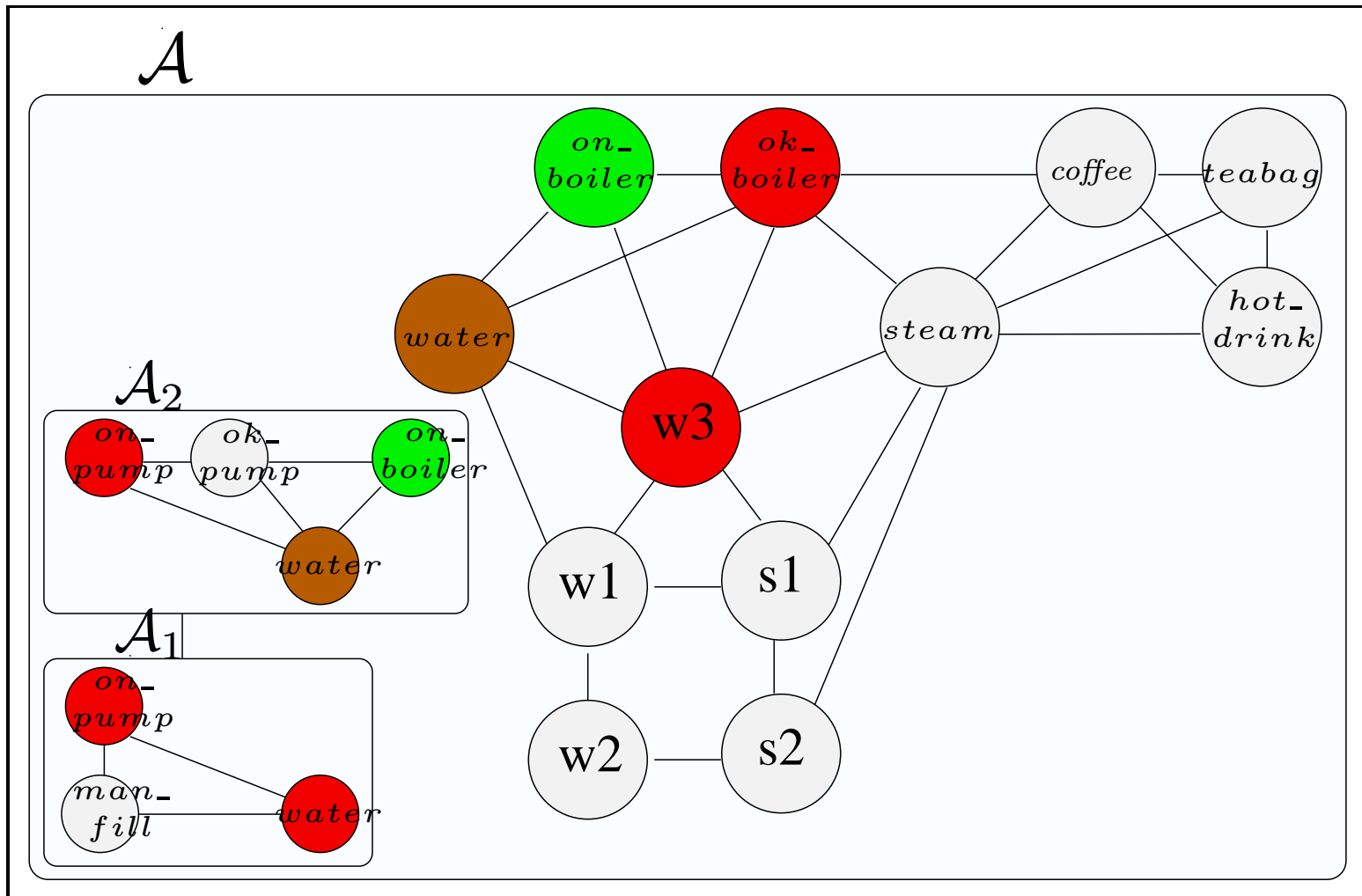
Using a *balanced separator* of previous separators

Automated Partitioning using *triangulate*



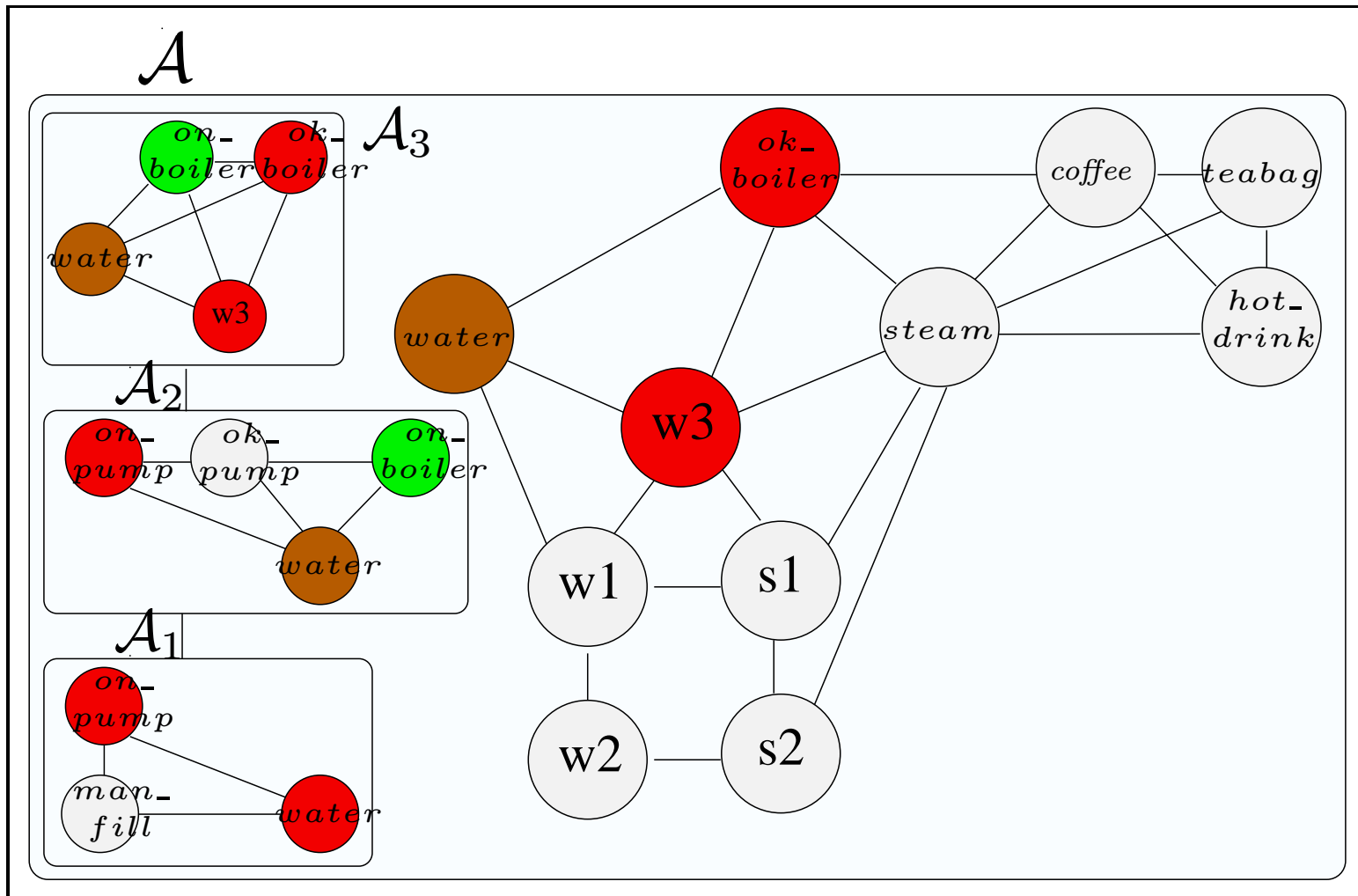
Using a *balanced separator* of previous separators

Automated Partitioning using *triangulate*



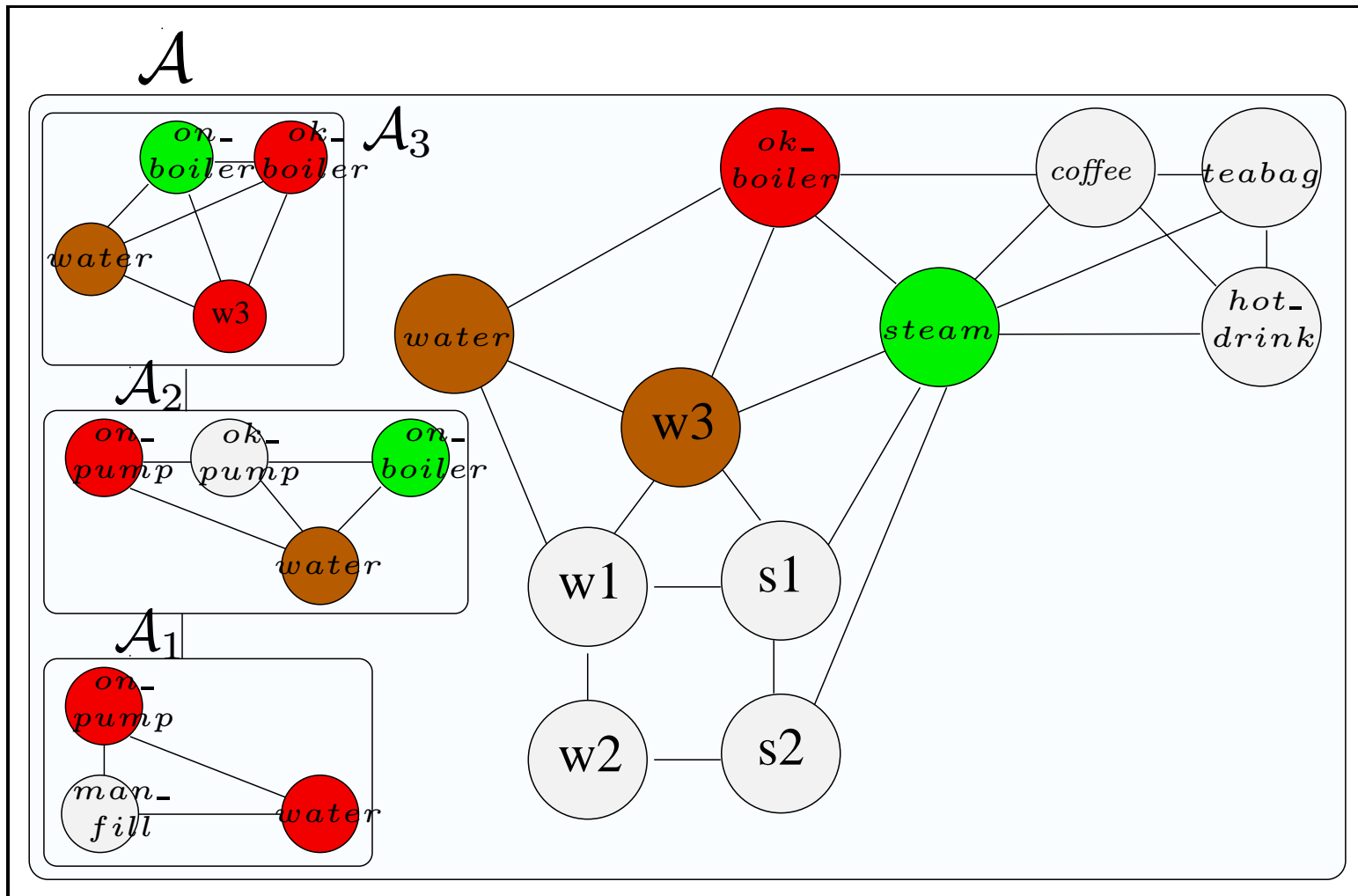
Using a *balanced separator* of previous separators

Automated Partitioning using *triangulate*



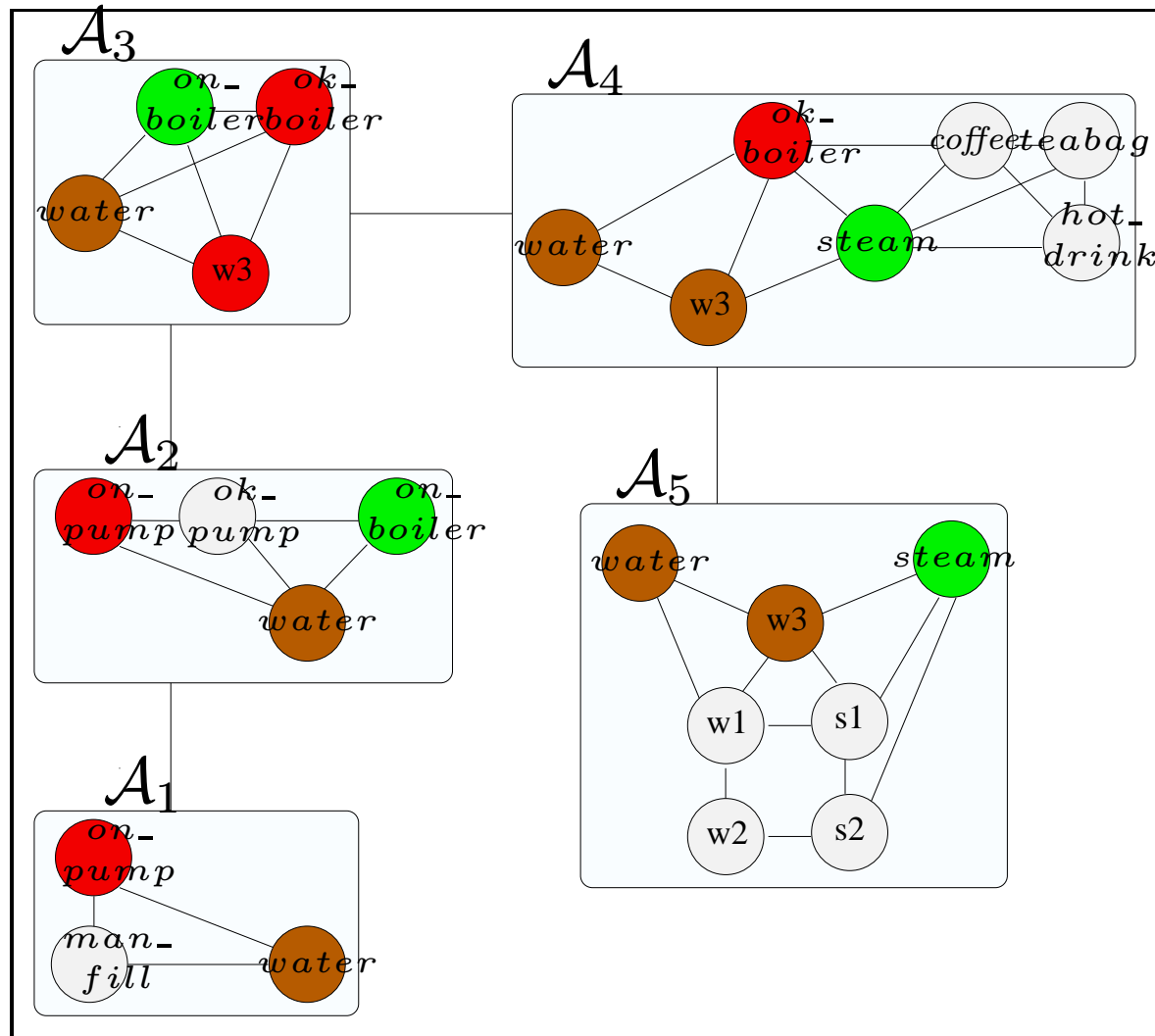
Using a *balanced separator* of previous separators

Automated Partitioning using *triangulate*



Using a *balanced separator* of previous separators

Automated Partitioning using *triangulate*



Overall decomposition graph.

Algorithms' Properties

algorithm	time bound	apx factor
<i>3way-triang</i>	$O(2^{3.7k} n^3 k^3 \lg^4 n)$	$3\frac{2}{3}$
<i>2way-2/3-triang</i>	$O(2^{4.38k} n^2 k^2)$	4
<i>2way-half-triang</i>	$O(2^{3k} n^2 k^{\frac{5}{2}})$	$4\frac{1}{2}$
<i>lgk-triang</i>	$O(n^3 \lg^4 n k^5 \lg k)$	$O(\lg k)$
(Shoikhet & Geiger 1997)	?	1
(Becker & Geiger 1996)	$O(2^{4.66k} n \text{ poly}(n))$	$3\frac{2}{3}$
(Robertson & Seymour 1995)	$O(2^{4.75k} n^2 k^2)$	4
(Reed 1992)	$O(2^{6.34k} n k^2 \lg n)$	5

Decomposition algorithms and their theoretical bounds.

Experimental Results

Knowledge Base	symbols	axioms	edges
CYC1: spatial axioms KB	142	223	469
CPCS1: Bayes Net for medical diagnosis	360 (RV)		1036
CPCS2: Bayes Net for medical diagnosis	421 (RV)		1704
HPKB1: political influence KB	446	1199	2637
HPKB2: political influence KB	570	688	3840

Data Sets

Graph	Nodes	Edges	Time		Width+1	
			$4\frac{1}{2}$ -apx	4-apx	$4\frac{1}{2}$ -apx	4-apx
CYC1	142	469	1min 2sec	6min 34sec	21	21
CPCS1	360	1036	8min 50sec	1hr 11min	28	26
CPCS2	421	1704	15min 40sec	3hr 55min	33	33
HPKB1	446	2637	2hr 7min	14hr 13min	58	45
HPKB2	570	3840	7hr 52min	5dy 23.5hr	70	60

Graphs and their decomposition.

Summary: Partitioning Logical Theories

- Reduce *partitioning axioms* to Tree Decomposition.
- Faster algorithms for tree decomposition.
- Decompose large KBs with high treewidth.
- Applicable to logic, Bayes networks and CSPs.
- Variants: weights on symbols and axioms, partitioning axioms.

Contents

1. Reasoning algorithms for logical partitions.
2. Automatic partitioning of logical theories.
3. **Object-Oriented design in logic.**
4. Reactive control with subsumption of partitions.

Object-Oriented Design Principles

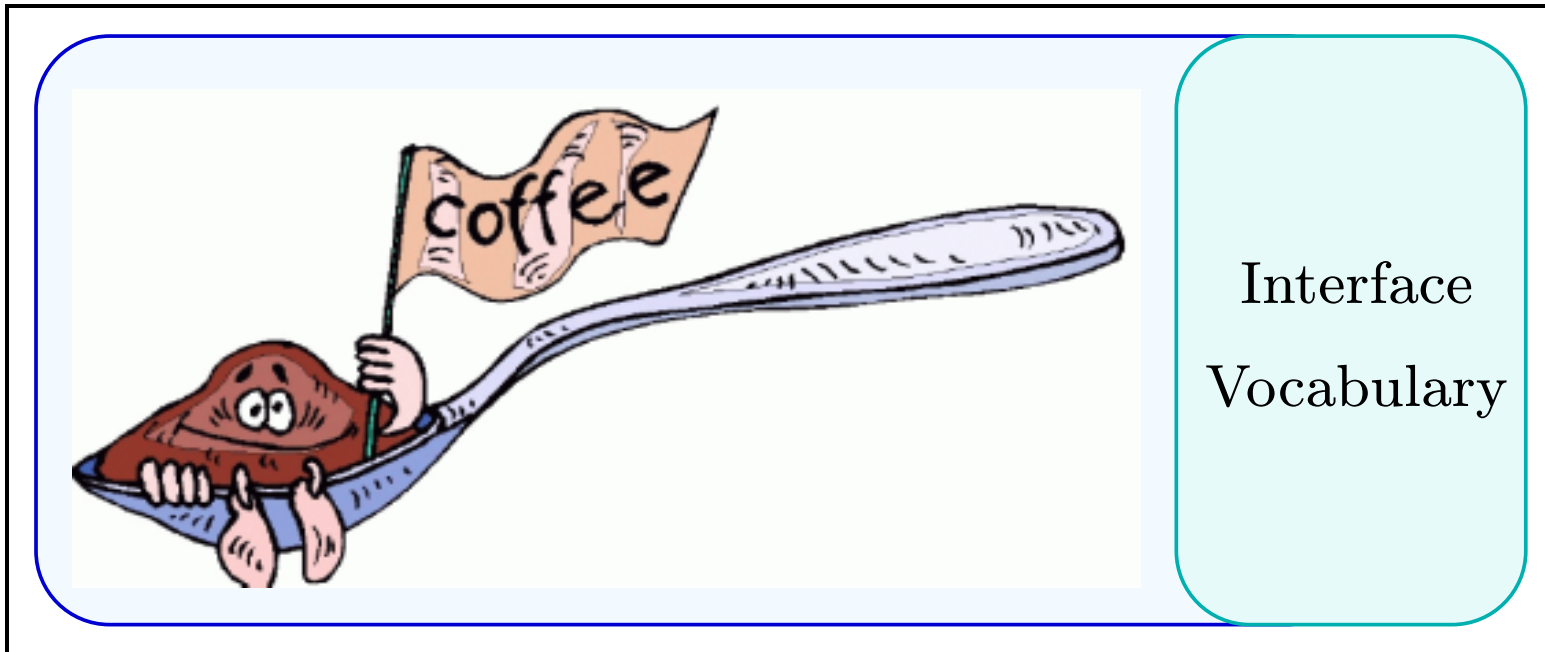
- Knowledge bases are connected fragments (*objects*).
- Each *object*
 - influences other objects via its interface.
 - contains a copy of its *class*.
- Each *class*
 - can be constructed independently of the rest.
 - contains knowledge about a topic.
- *Inheritance* allows reuse and expansion of objects.

An Espresso Machine and its Parts



A breakdown of the process of making an espresso.

A Hot-Drink Class



- Language/Vocabulary.
- Interface Vocabulary.
- Axioms.

A Hot-Drink Class

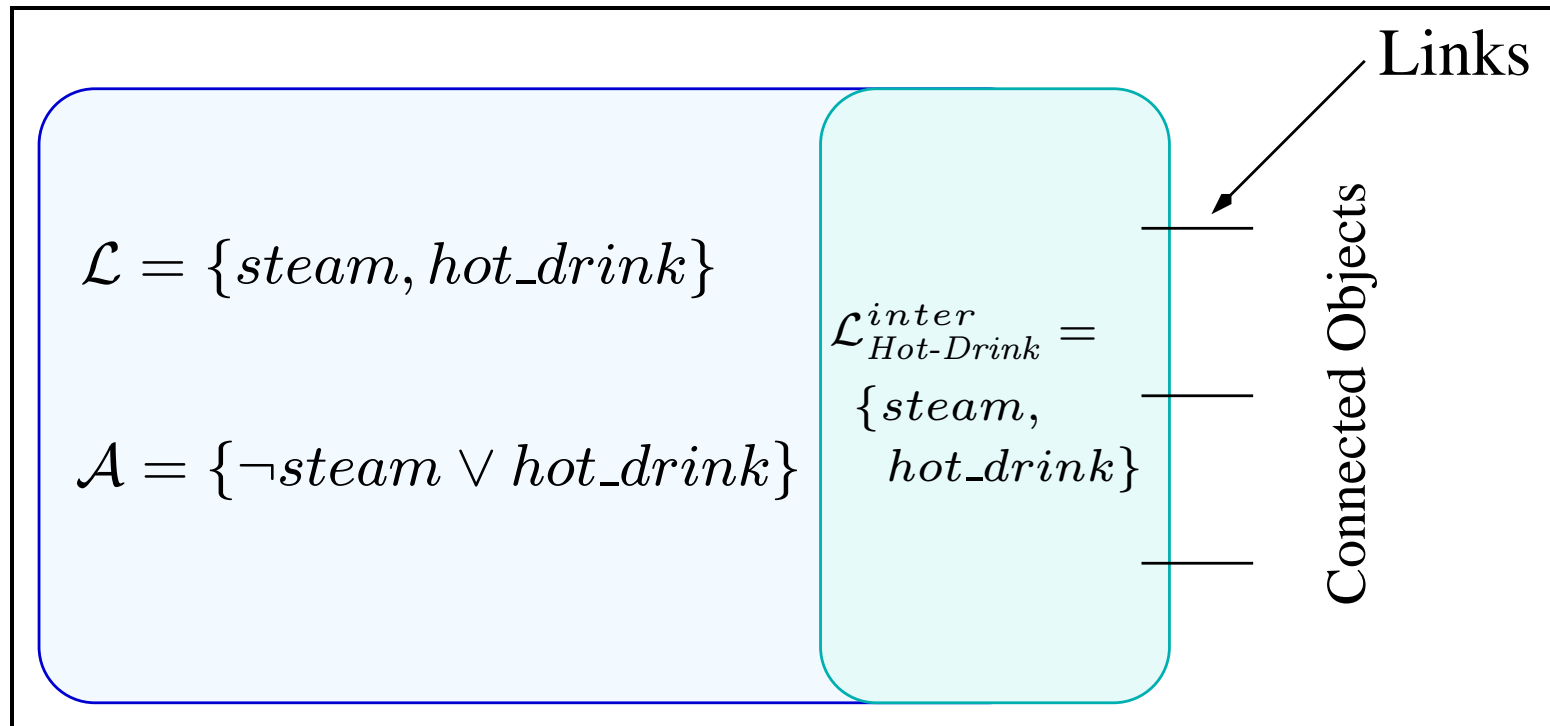
$$\mathcal{L} = \{steam, hot_drink\}$$

$$\mathcal{A} = \{\neg steam \vee hot_drink\}$$

$$\mathcal{L}_{Hot-Drink}^{inter} = \{steam, hot_drink\}$$

- Language/Vocabulary.
- Interface Vocabulary.
- Axioms.

A Hot-Drink Object



- Language/Vocabulary.
- Axioms.
- Interface Vocabulary.
- Links to other objects.

Steam-Pressure is a *Subclass* of Hot-Drink

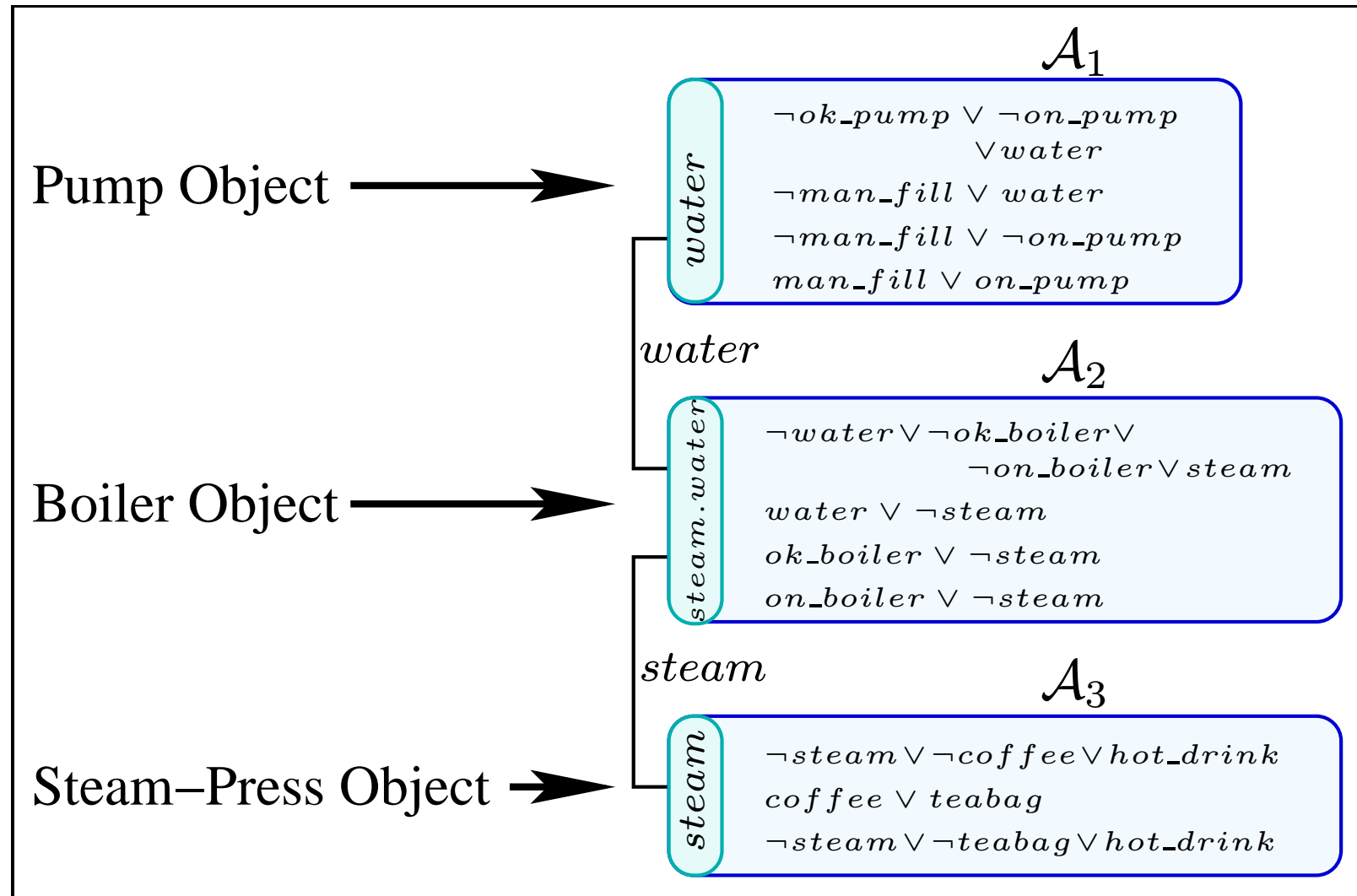
$$\mathcal{L}_{\text{Steam-Pressure}} = \underline{\mathcal{L}_{\text{Hot-Drink}}} \cup \{coffee, teabag, hot_drink'\}$$

$$\begin{aligned} \mathcal{A}_{\text{Steam-Pressure}} = & \\ & \underline{(\mathcal{A}_{\text{Hot-Drink}})_{[hot_drink/hot_drink']}} \cup \\ & \{\neg coffee \vee \neg hot_drink' \vee hot_drink \\ & \quad coffee \vee teabag \\ & \quad \neg teabag \vee \neg hot_drink' \vee hot_drink\} \end{aligned}$$

$$\begin{aligned} \mathcal{L}_{\text{Steam-Pressure}}^{inter} \\ = \\ \underline{\mathcal{L}_{\text{Hot-Drink}}^{inter}} \end{aligned}$$

- A *Class* is a template for objects. It has no links.
- A subclass inherits all the vocabulary and axioms.

An Object-Oriented Espresso Machine



An object-oriented view of an espresso machine in logic.

The Espresso_Machine Class

$\mathcal{L}_{Espresso_machine}^{inter} = \{hot_drink, location, in\}$

Pump Object

$\neg ok_pump \vee \neg on_pump$
 $\vee water$
 $\neg man_fill \vee water$
 $\neg man_fill \vee \neg on_pump$
 $man_fill \vee on_pump$

$\mathcal{L} = \dots$

water

Boiler Object

$\neg water \vee \neg ok_boiler \vee$
 $\neg on_boiler \vee steam$
 $water \vee \neg steam$
 $ok_boiler \vee \neg steam$
 $on_boiler \vee \neg steam$

$\mathcal{A} = \dots$

water
steam

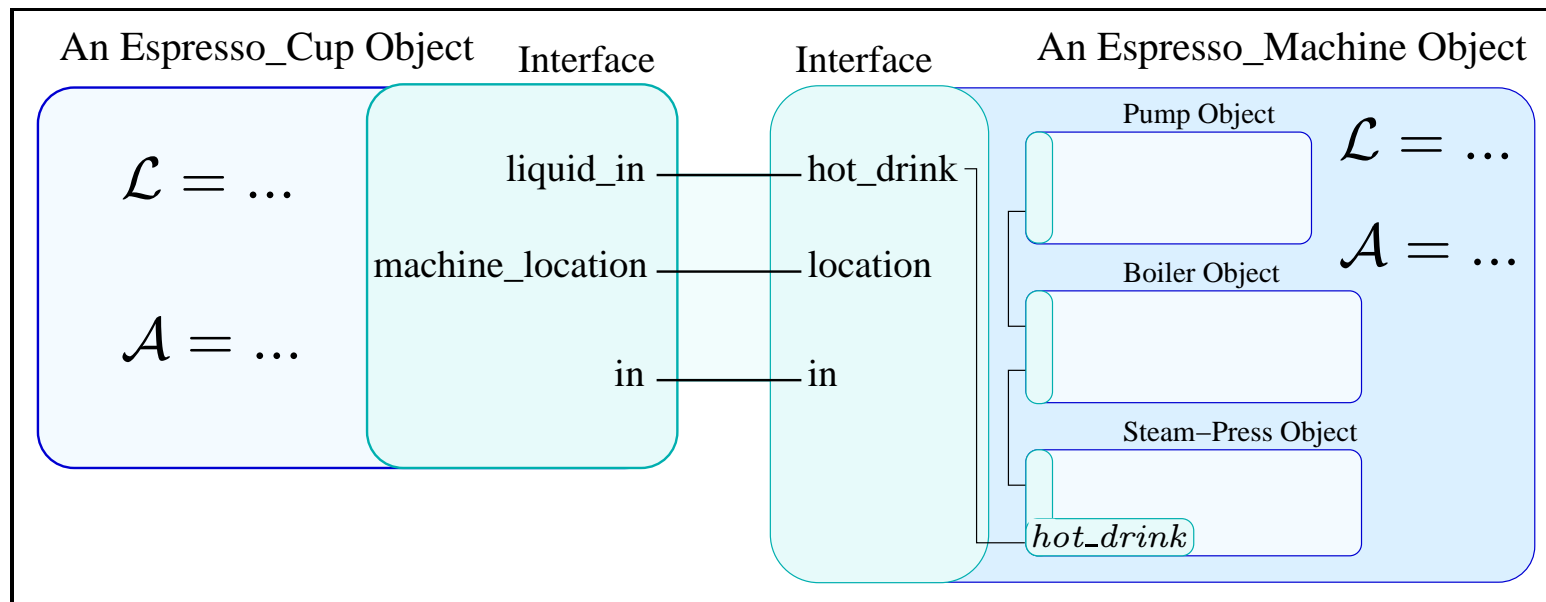
steam

Steam-Press Object

$\neg steam \vee \neg coffee \vee hot_drink$
 $coffee \vee teabag$
 $\neg steam \vee \neg teabag \vee hot_drink$
 hot_drink

steam
hot_drink

Espresso_Cup and Espresso_Machine



- We assume symbols in different objects are different.
- Links correspond to equality/equivalence relations between the respective symbols.

Formal Semantics

- A *Class* C is a tuple $\langle \mathcal{L}_C, \mathcal{A}_C, \mathcal{I}_C \rangle$. ($\mathcal{I}_C \subset \mathcal{L}_C$).
⟨FOL vocabulary, Axioms, interface vocabulary⟩.
- An *OOFOL Theory* T is a set of statements

Object(C, N, L).

C - Class, N - Name, L - Linking Relation.

FOL Semantics for OOFOL

- Translate T to FOL (\tilde{T}) by:
 - Every symbol P in object N translates to $N.P$.
 - \tilde{T} also includes *linking* axioms

$$\forall \vec{x} (N.P(\vec{x}) \equiv N'.P'(\vec{x}))$$

- M is a model of T iff $M \models \tilde{T}$.

Object-Oriented Knowledge Bases

- We have applied it to:
 - Situation Calculus and other temporal logics
 - The frame problem in temporal reasoning
 - Joining multiple domain theories
 - Represent loosely interacting agents

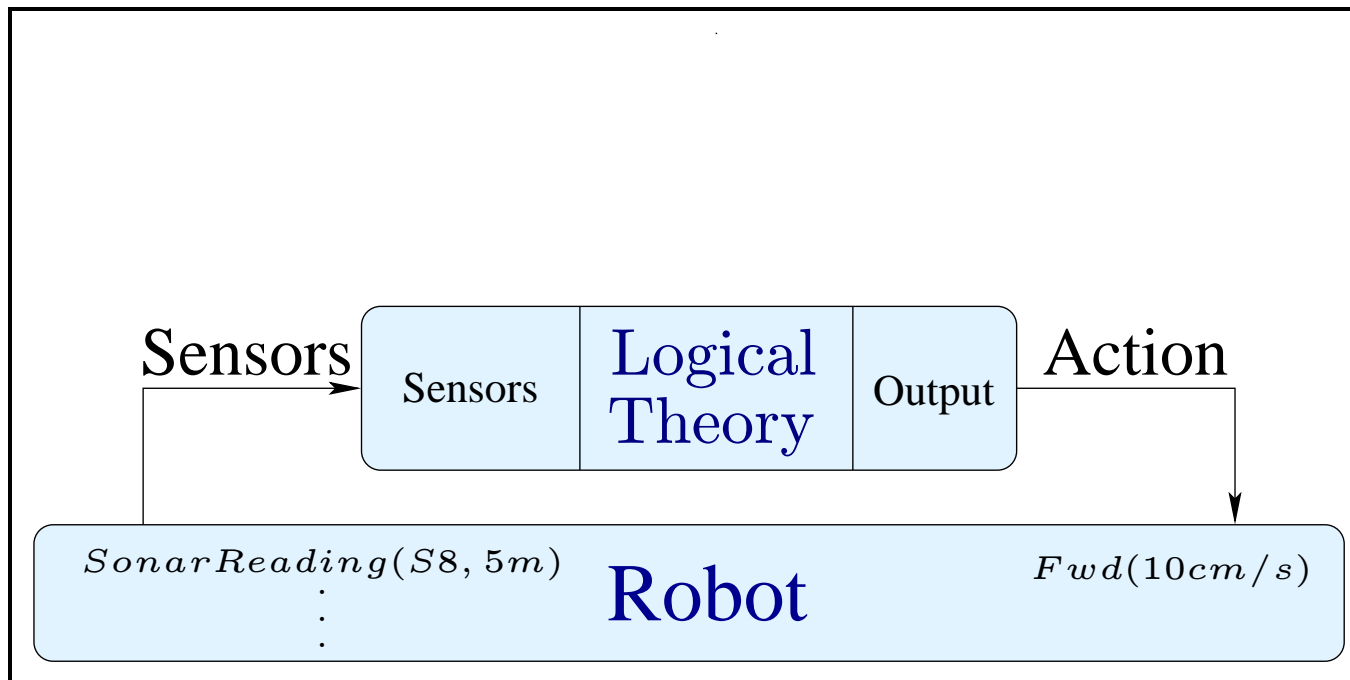
Summary: Object-Oriented FOL

- Object-Oriented tools and design for logic.
- Same simple semantics as FOL.
- Easier to build and maintain knowledge bases.
- Can extend expressiveness of Frame systems.

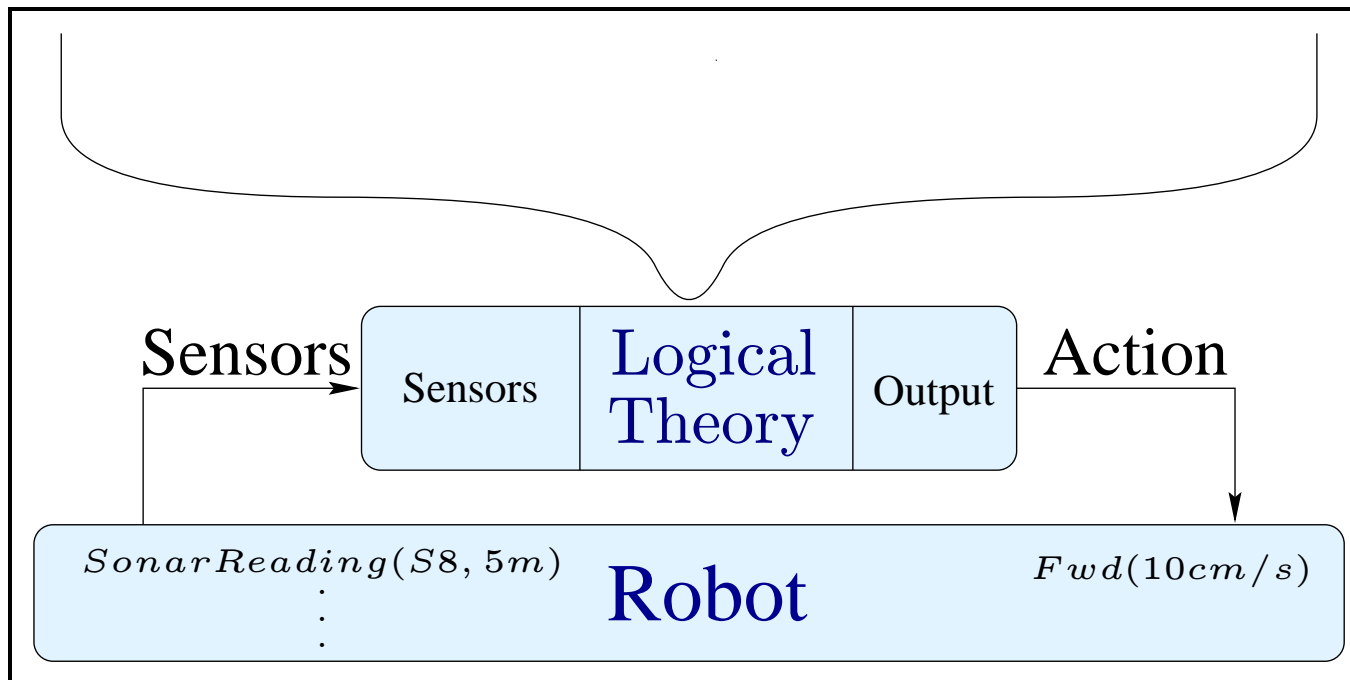
Contents

1. Reasoning algorithms for logical partitions.
2. Automatic partitioning of logical theories (glance).
3. Object-Oriented design in logic (glance).
4. **Reactive control with subsumption of partitions.**

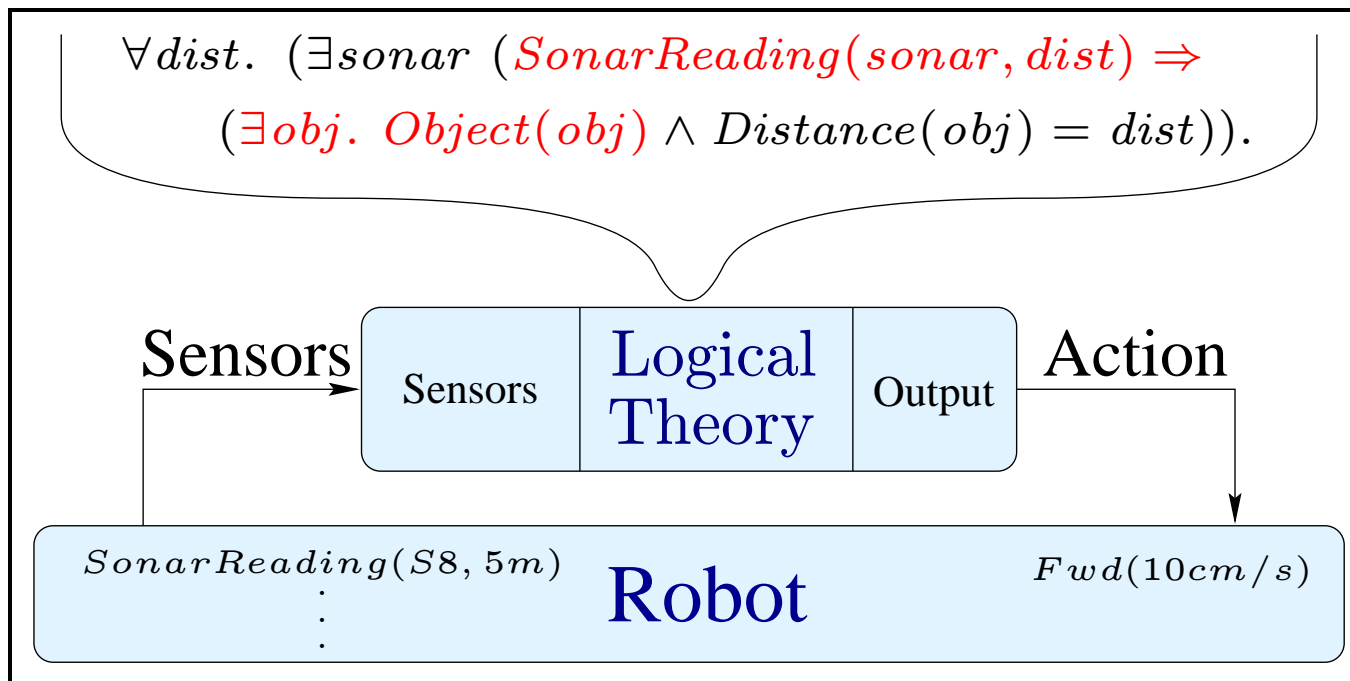
Naive Robot-Control using a Logical Theorem Prover



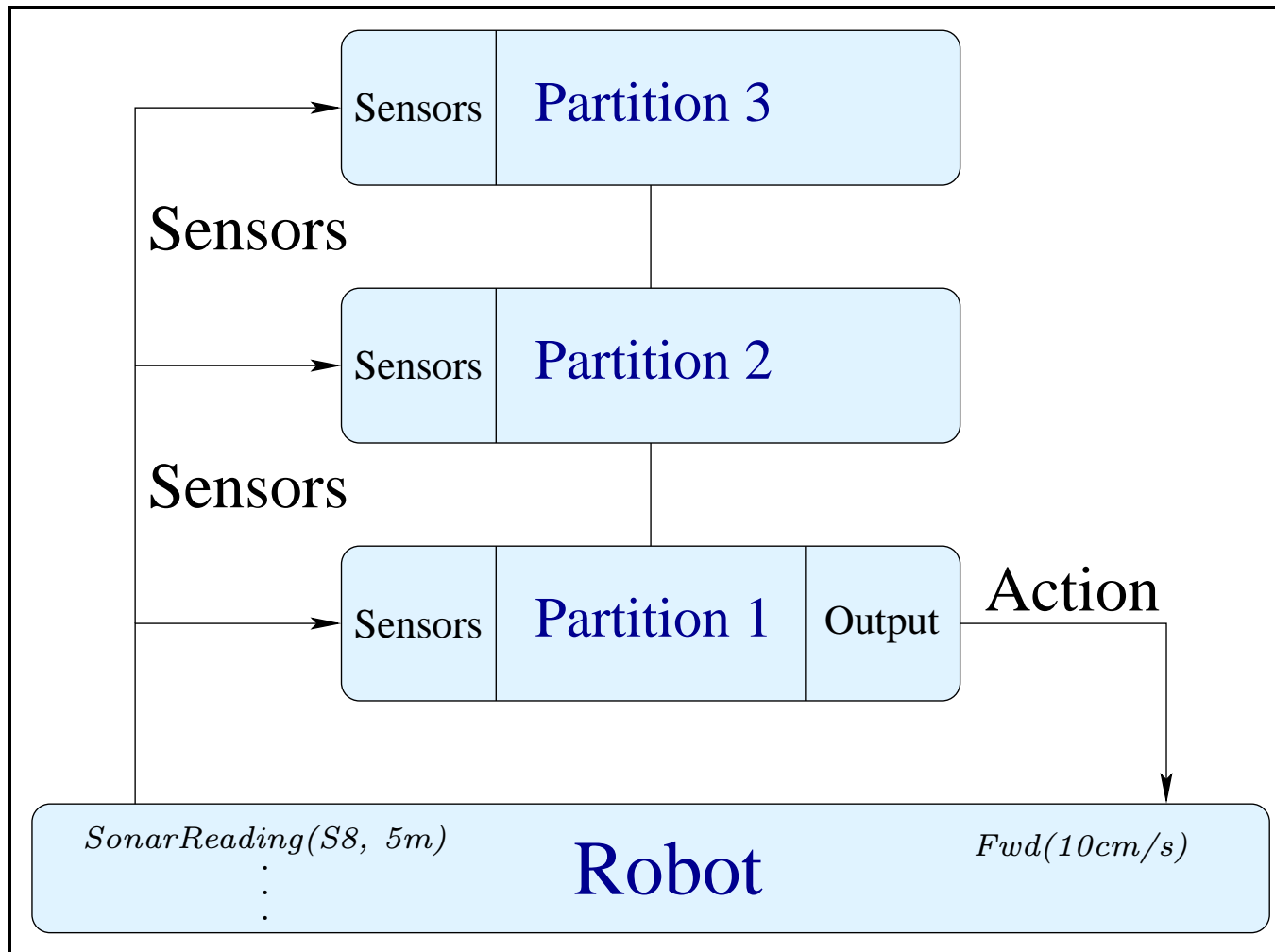
Naive Robot-Control using a Logical Theorem Prover



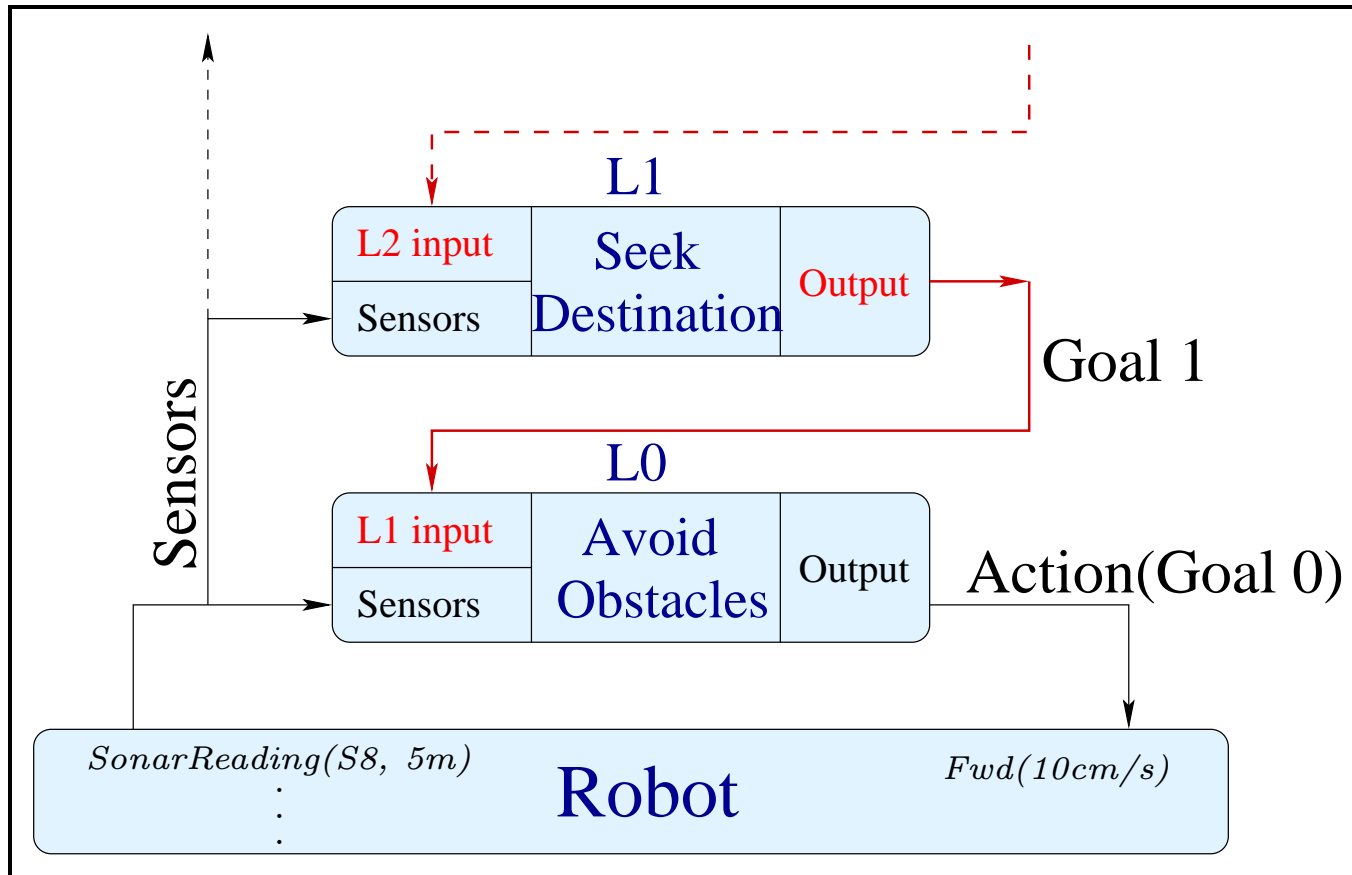
Naive Robot-Control using a Logical Theorem Prover



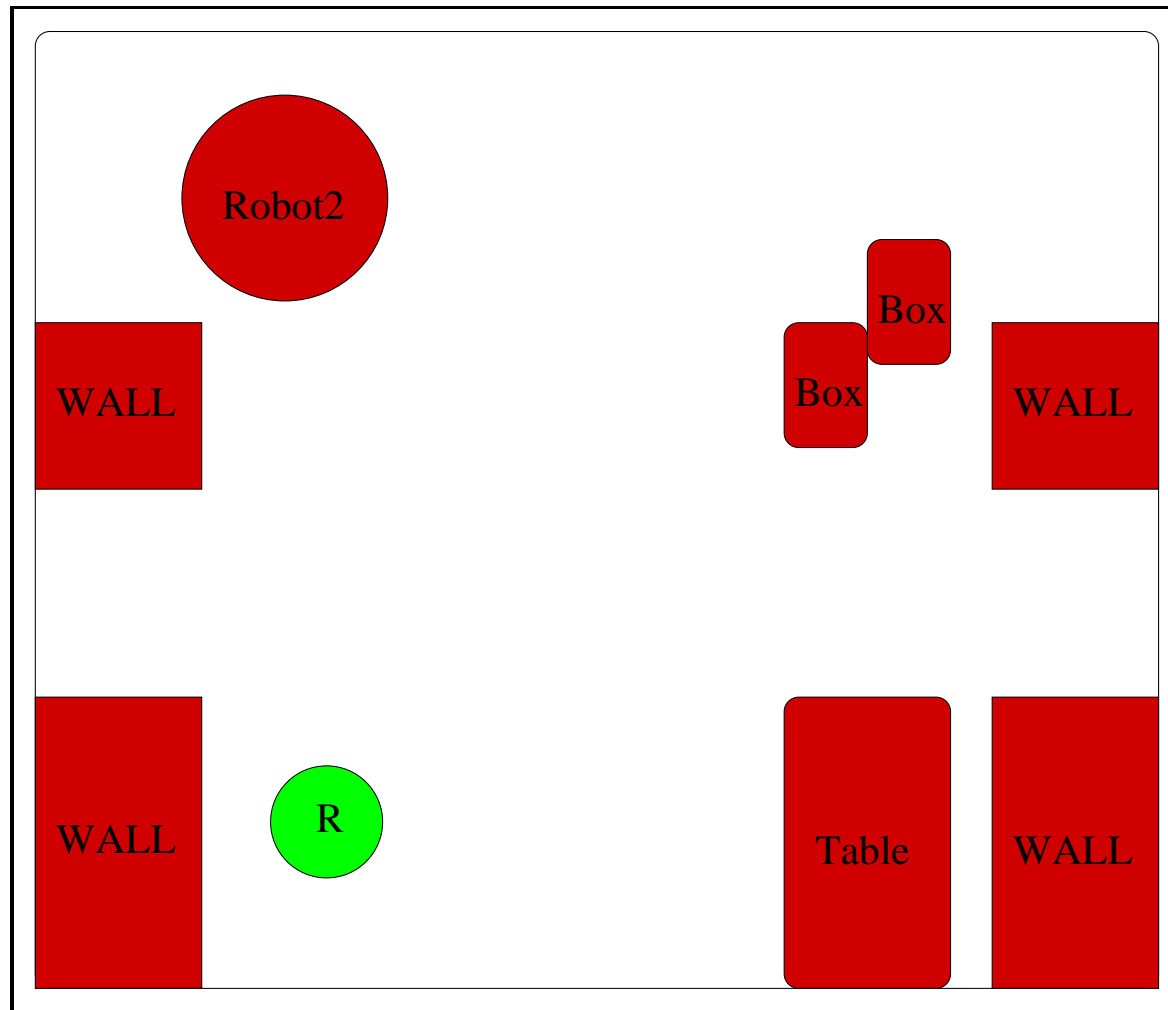
Naive Control using Logical Partitions



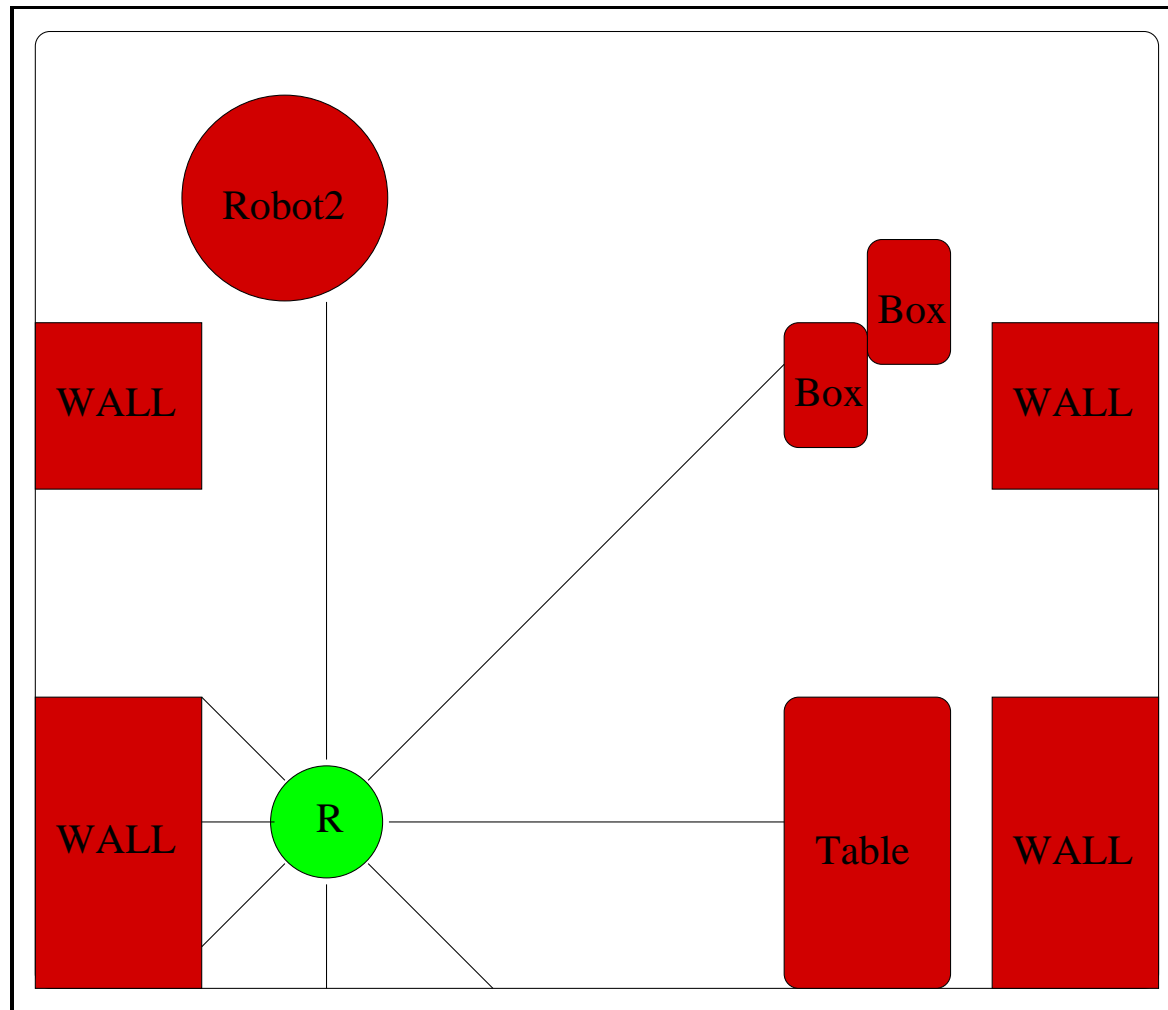
Logic-Based Subsumption (LSA)



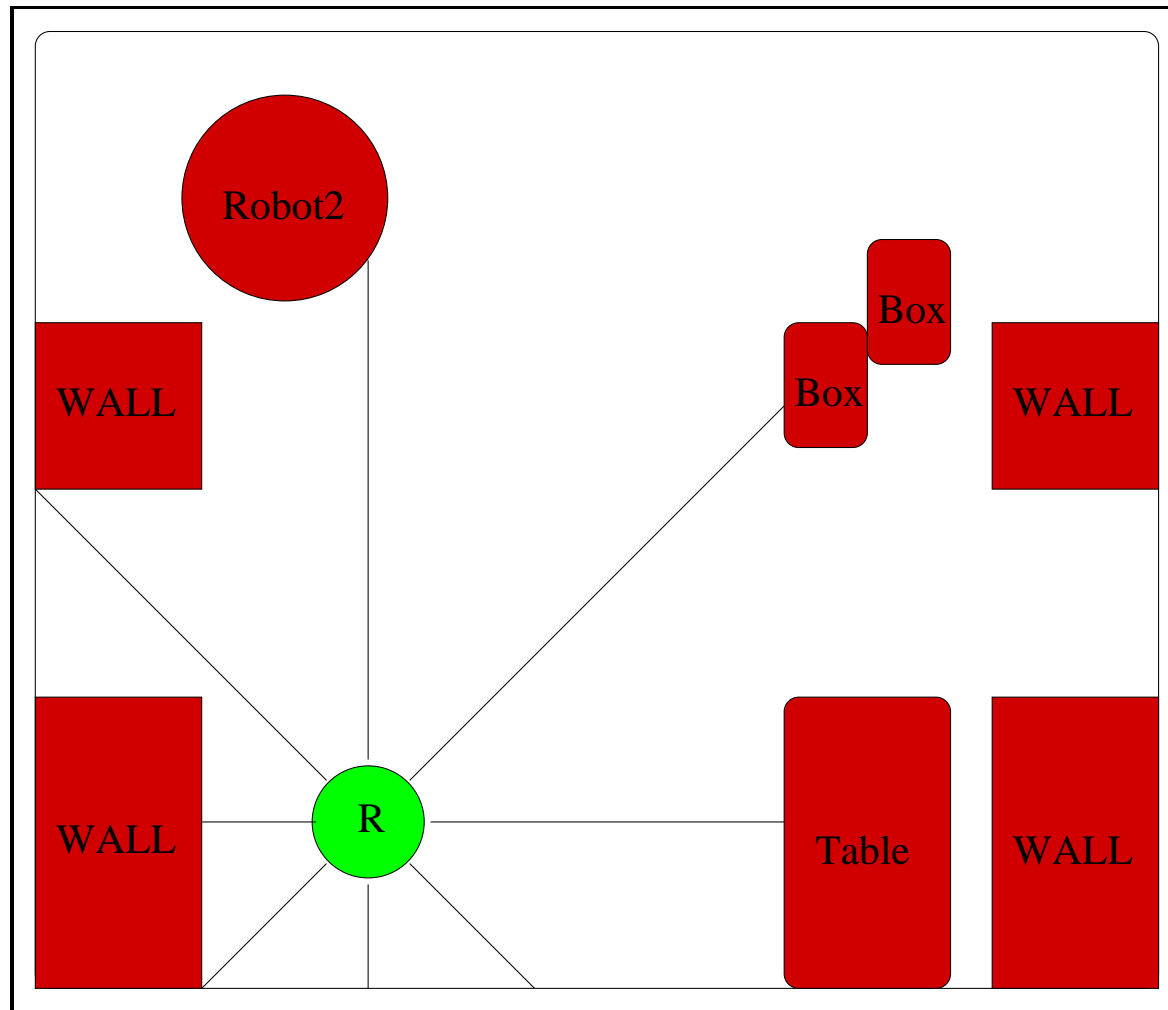
Robot Motion using Sonars



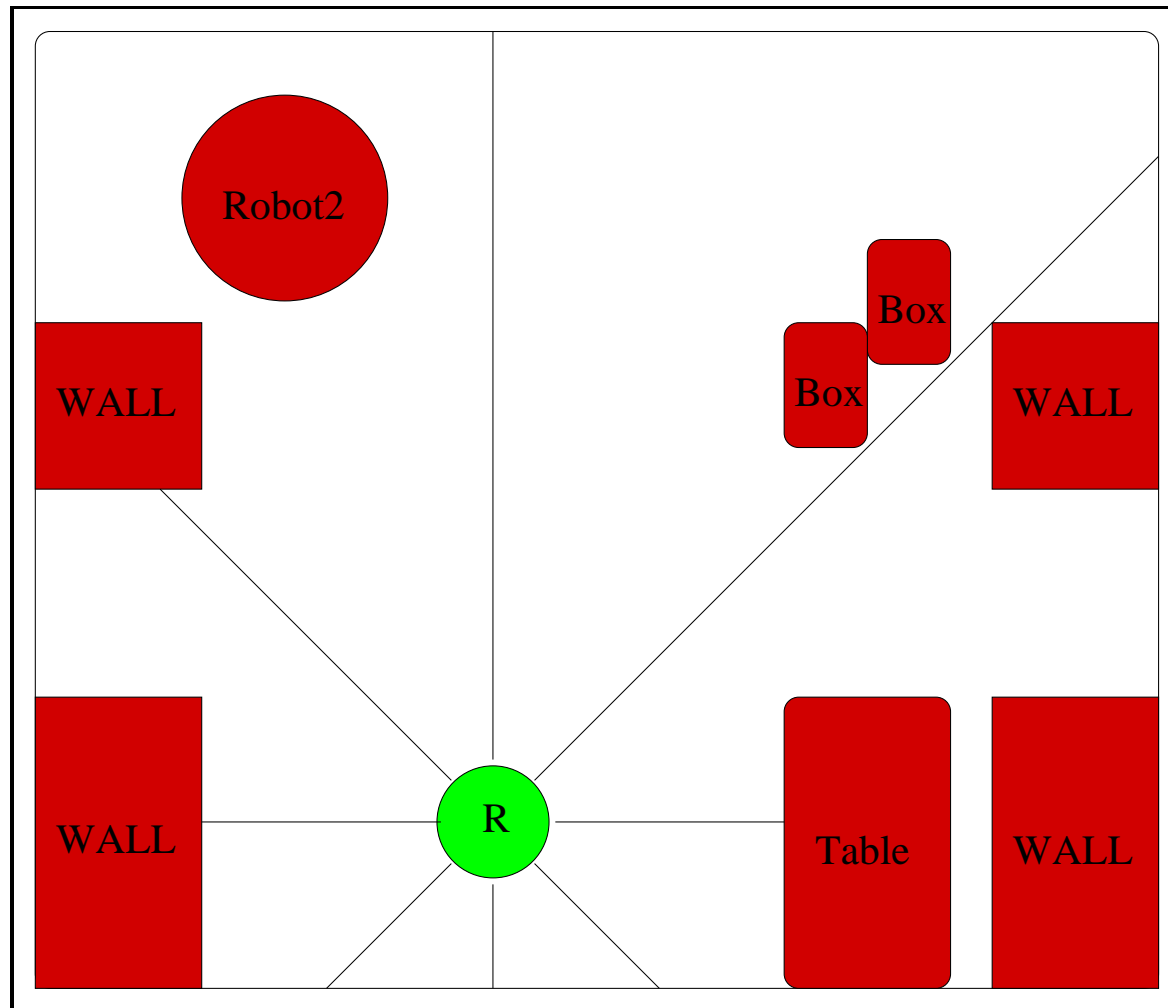
Robot Motion using Sonars



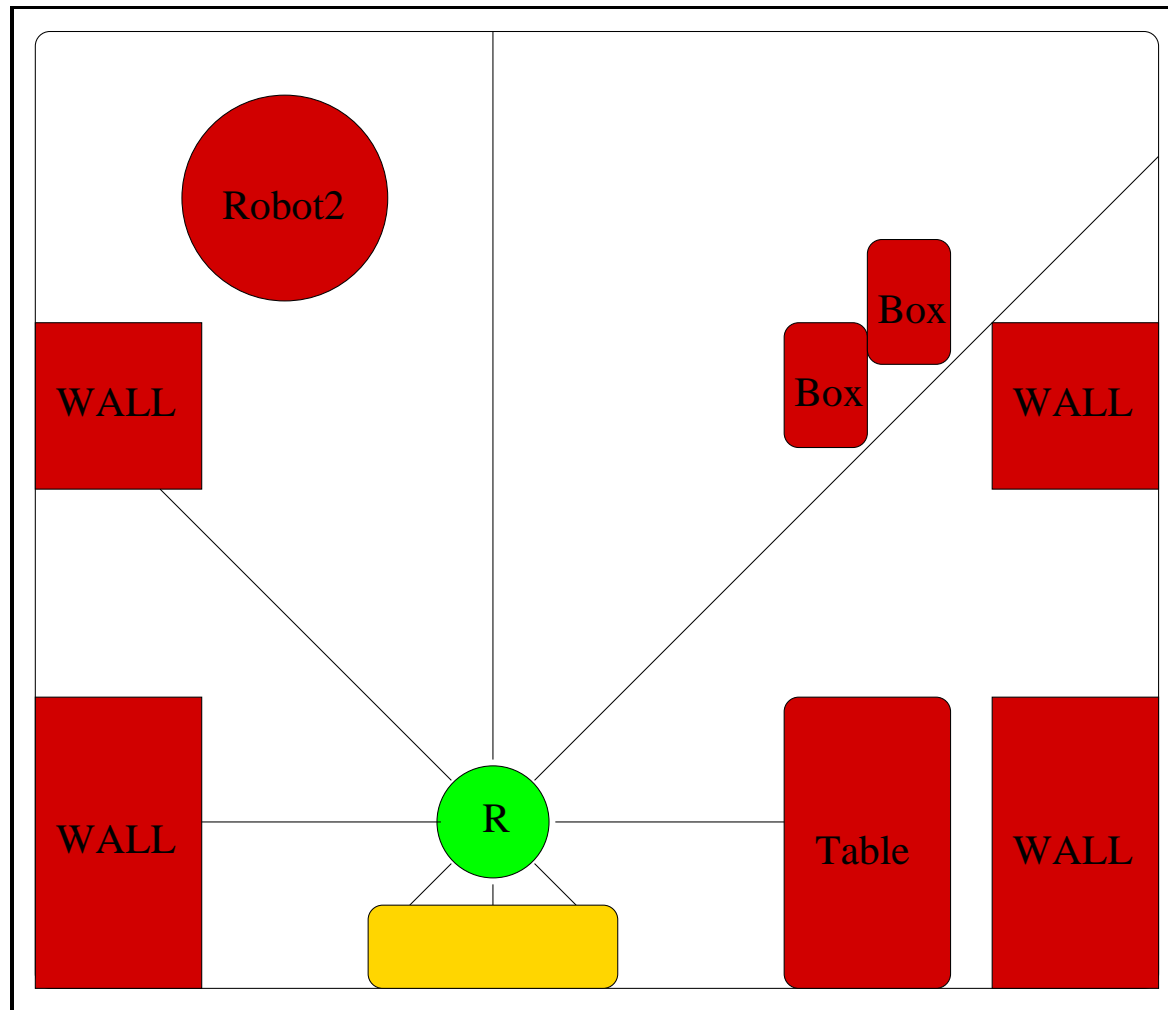
Robot Motion using Sonars



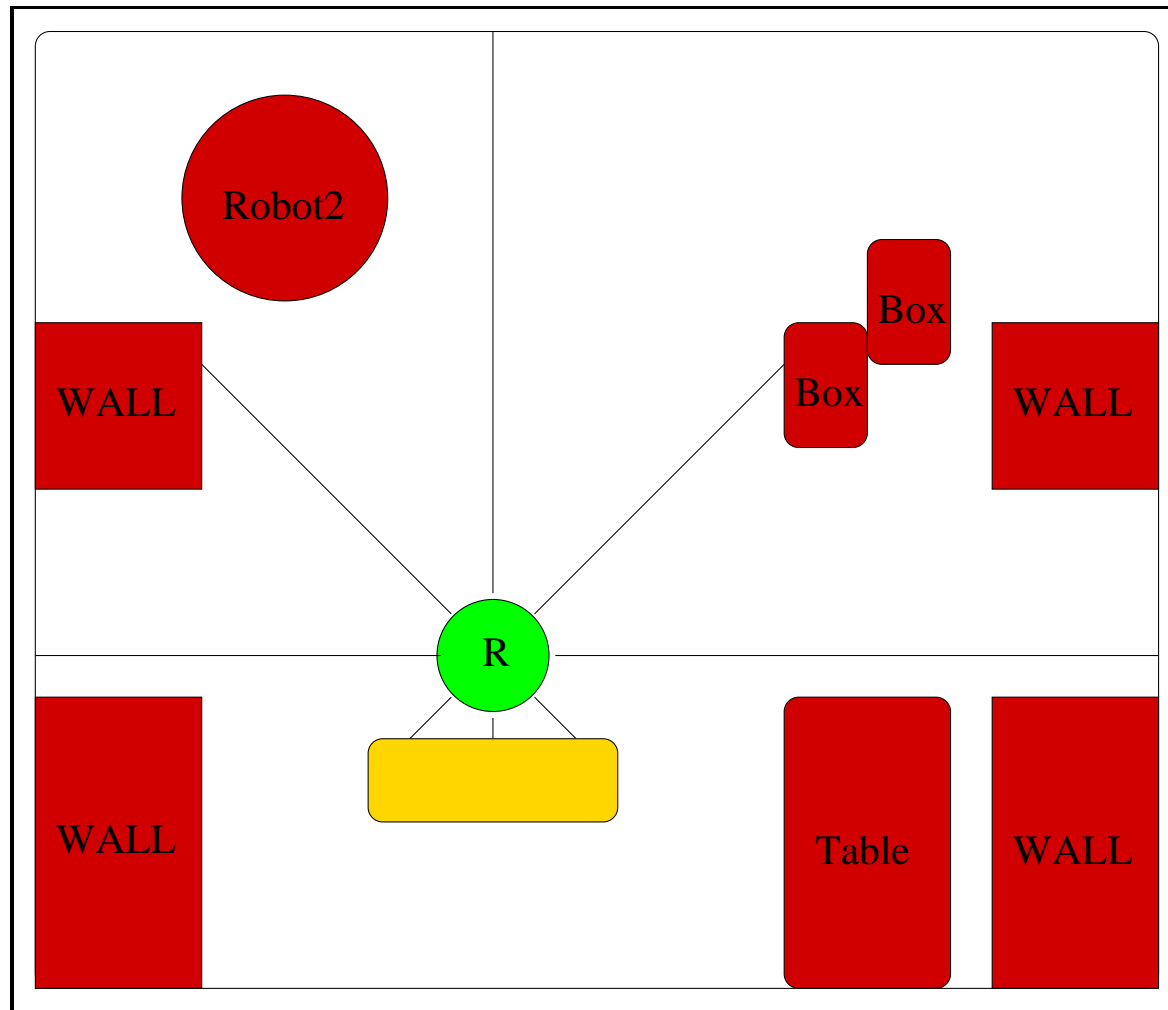
Robot Motion using Sonars



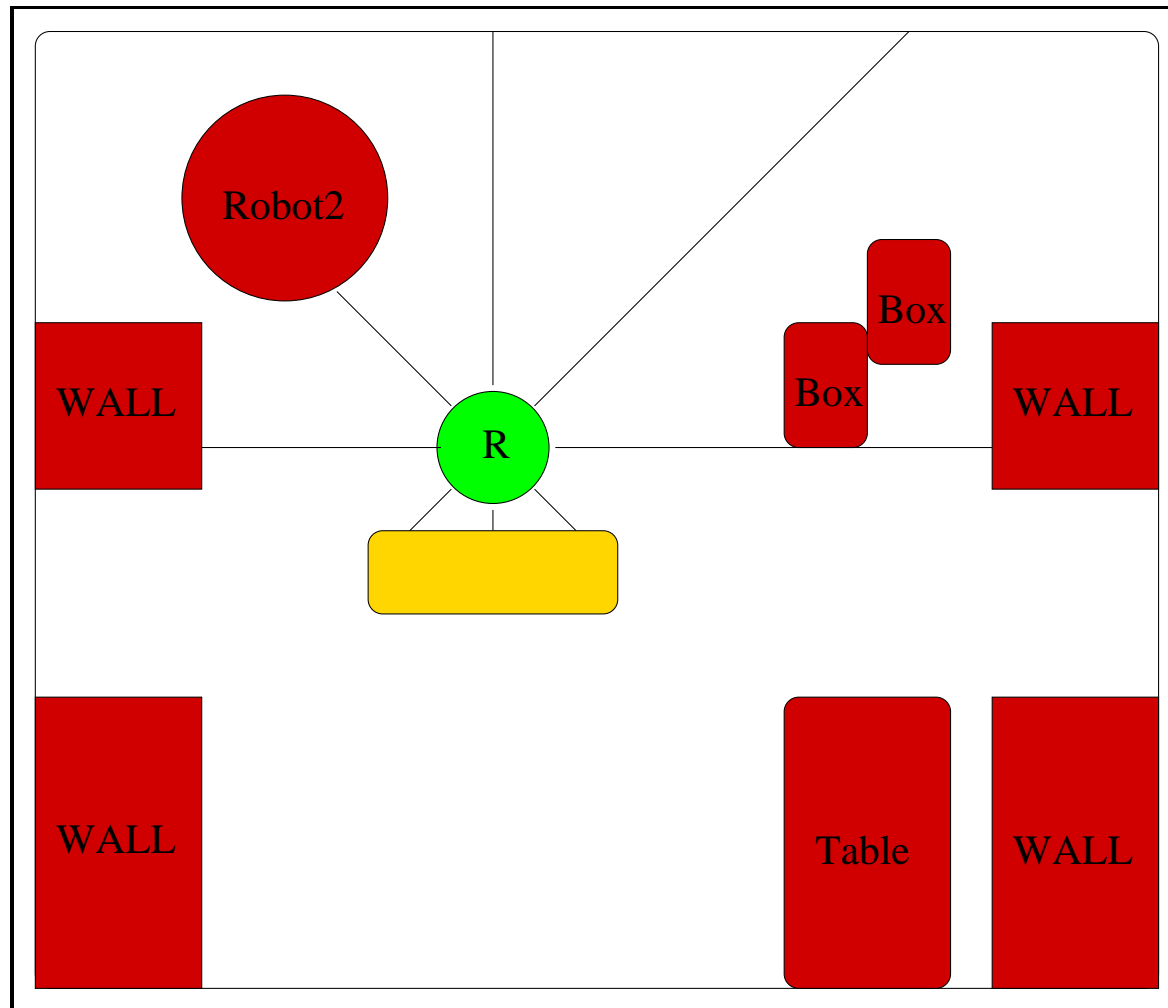
Robot Motion using Sonars



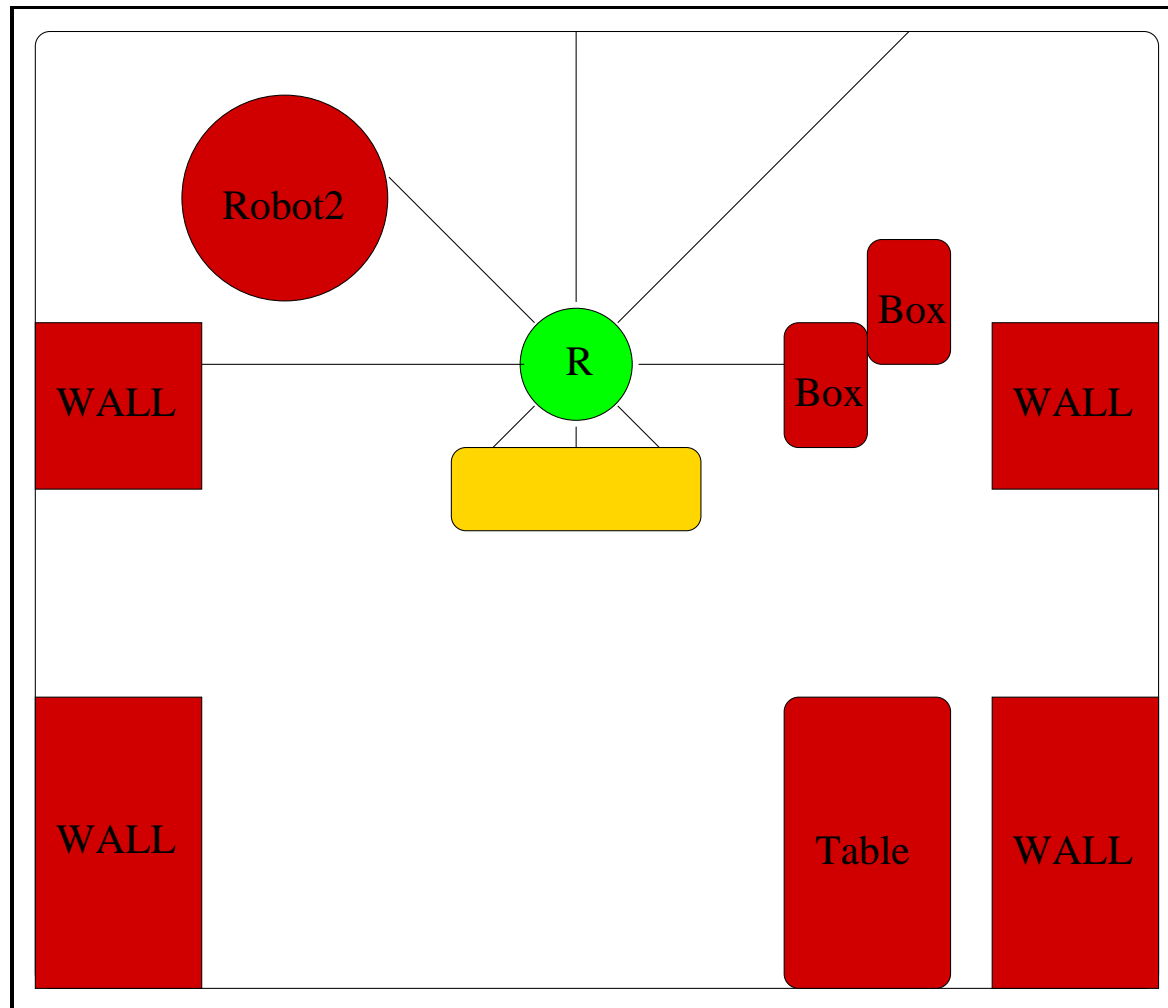
Robot Motion using Sonars



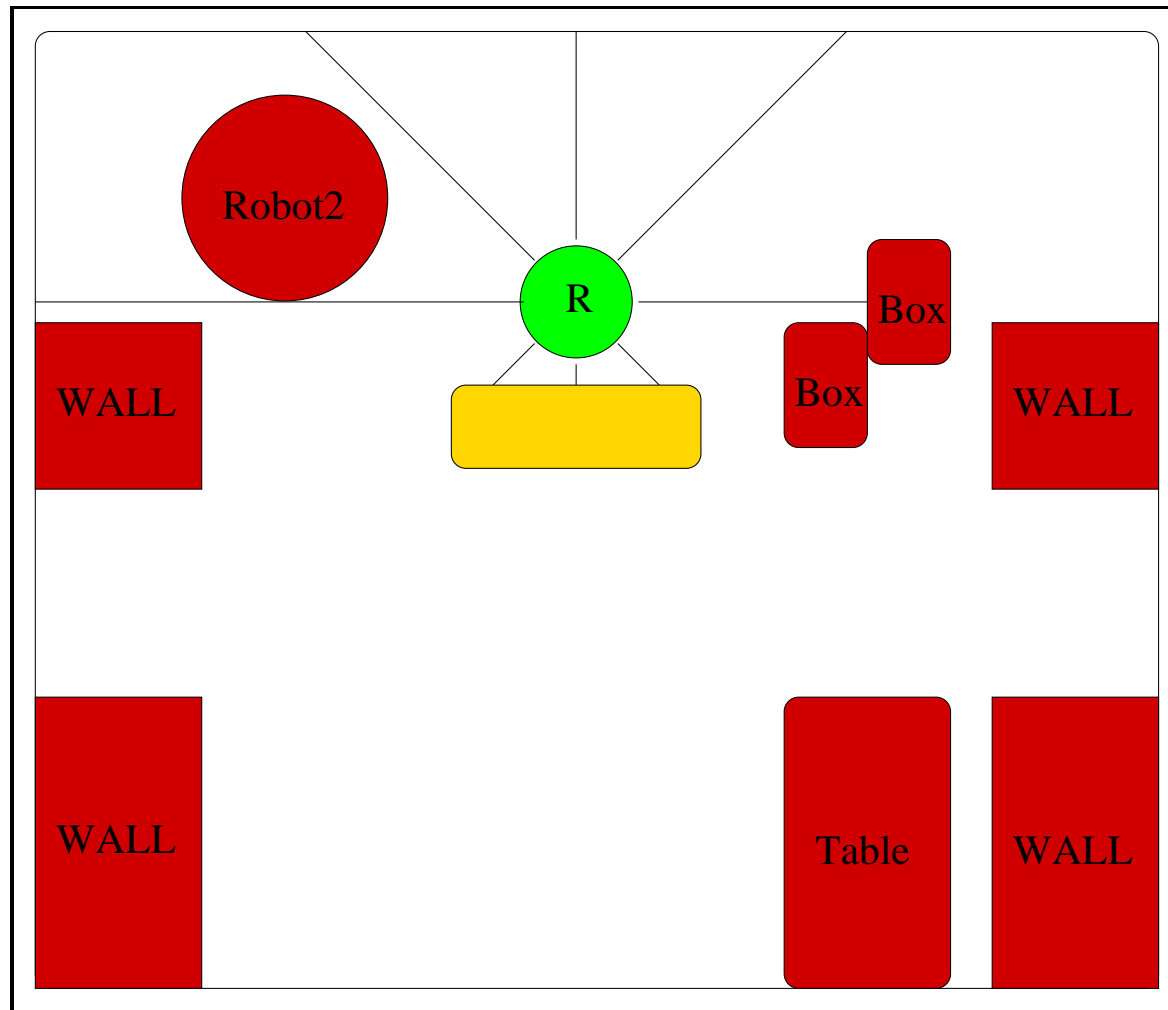
Robot Motion using Sonars



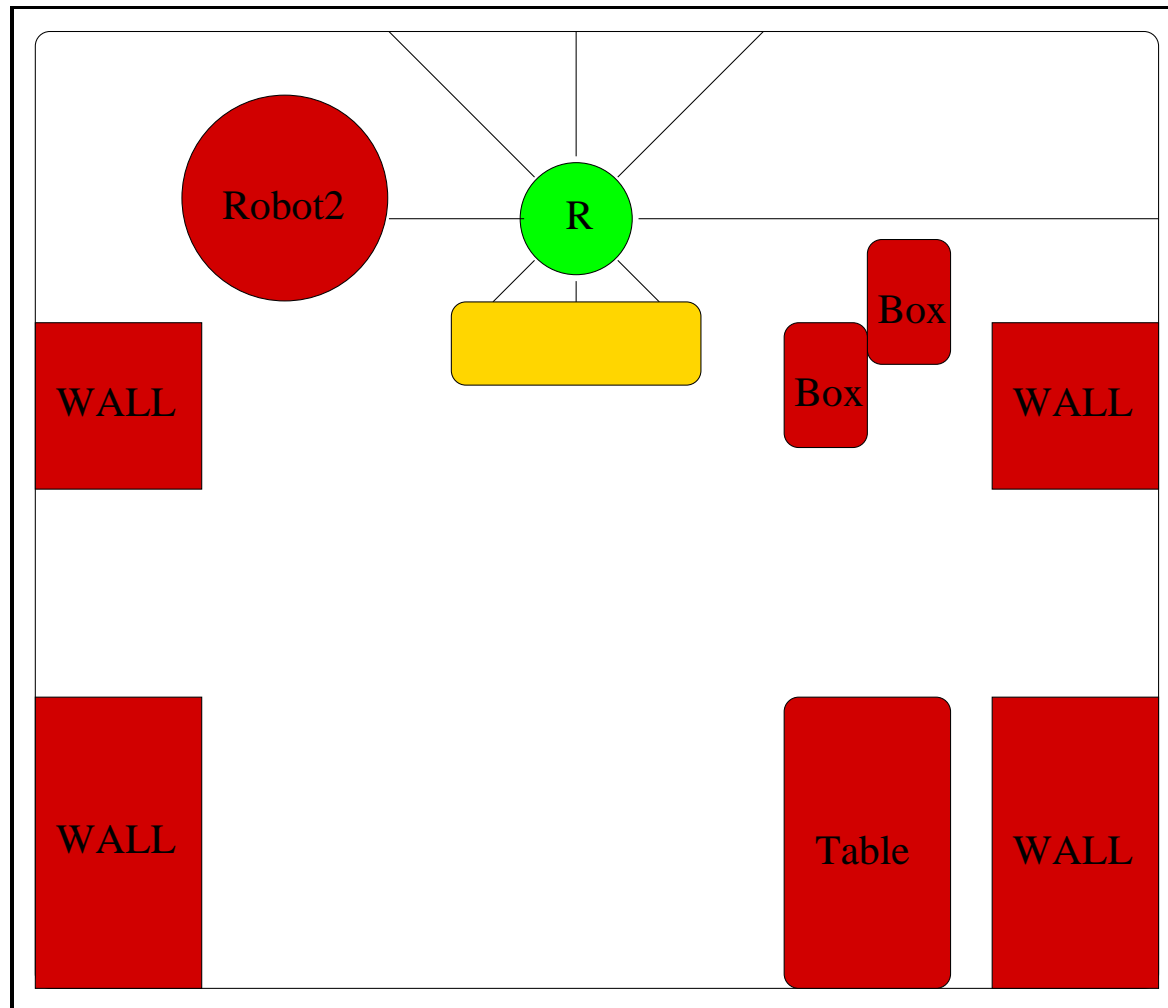
Robot Motion using Sonars



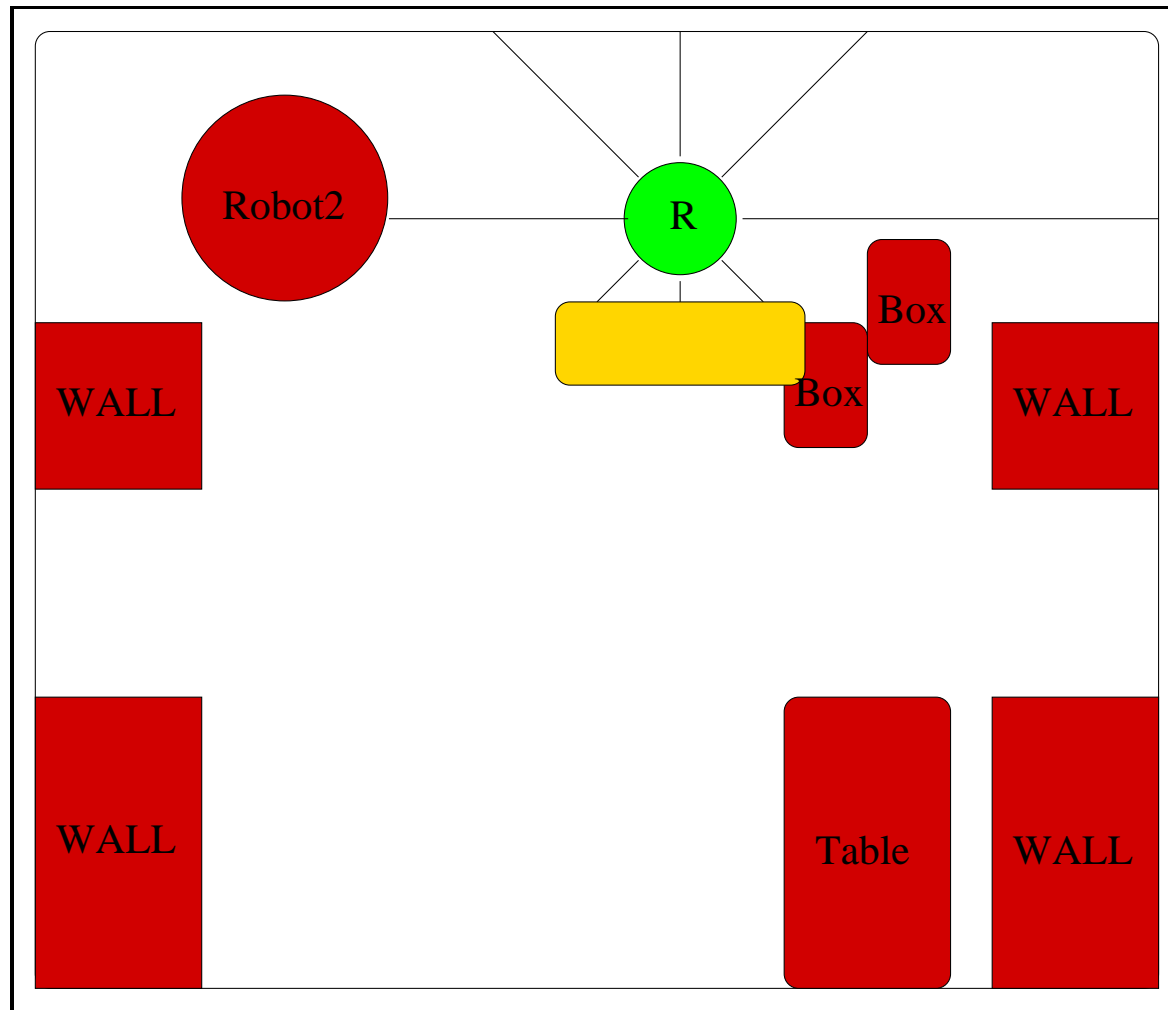
Robot Motion using Sonars



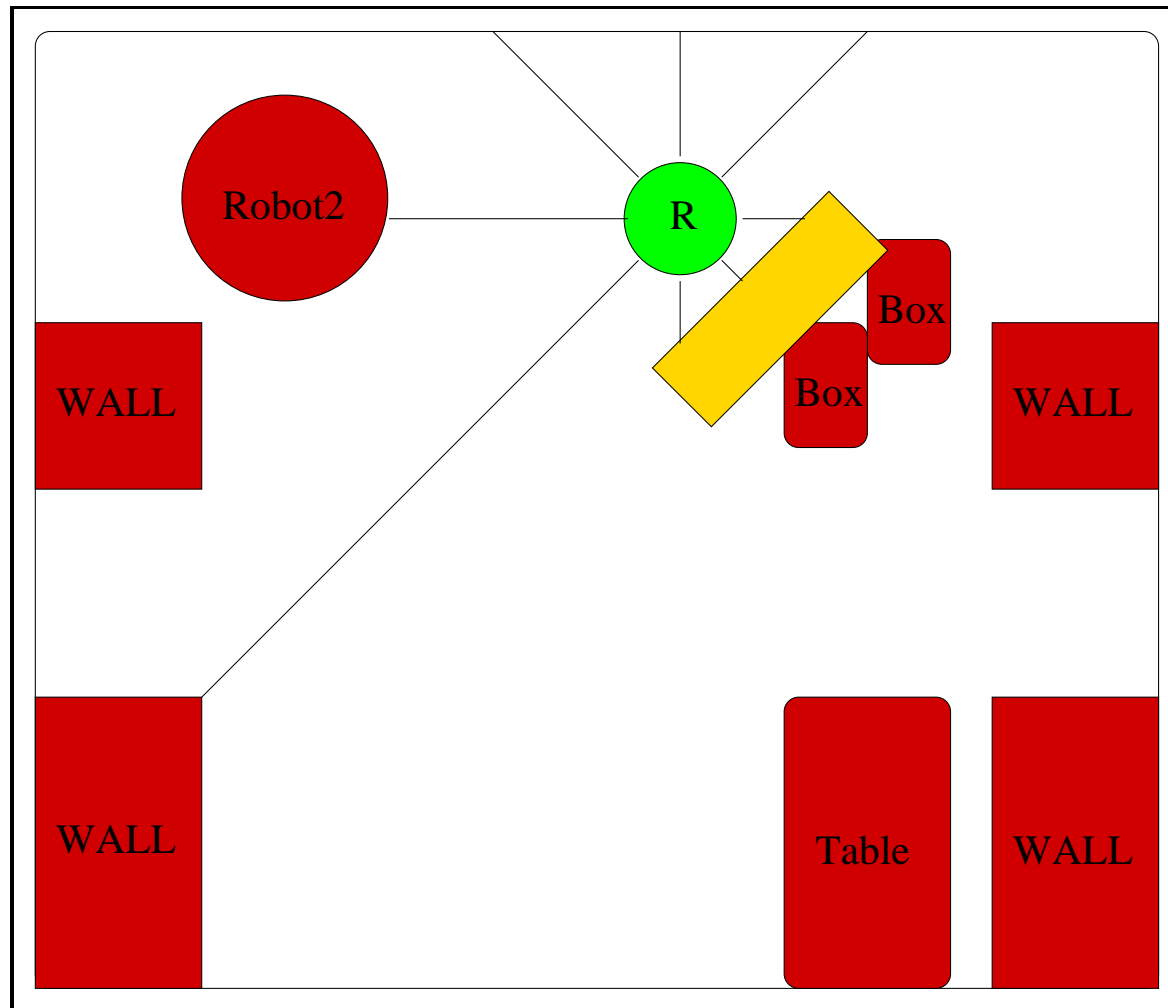
Robot Motion using Sonars



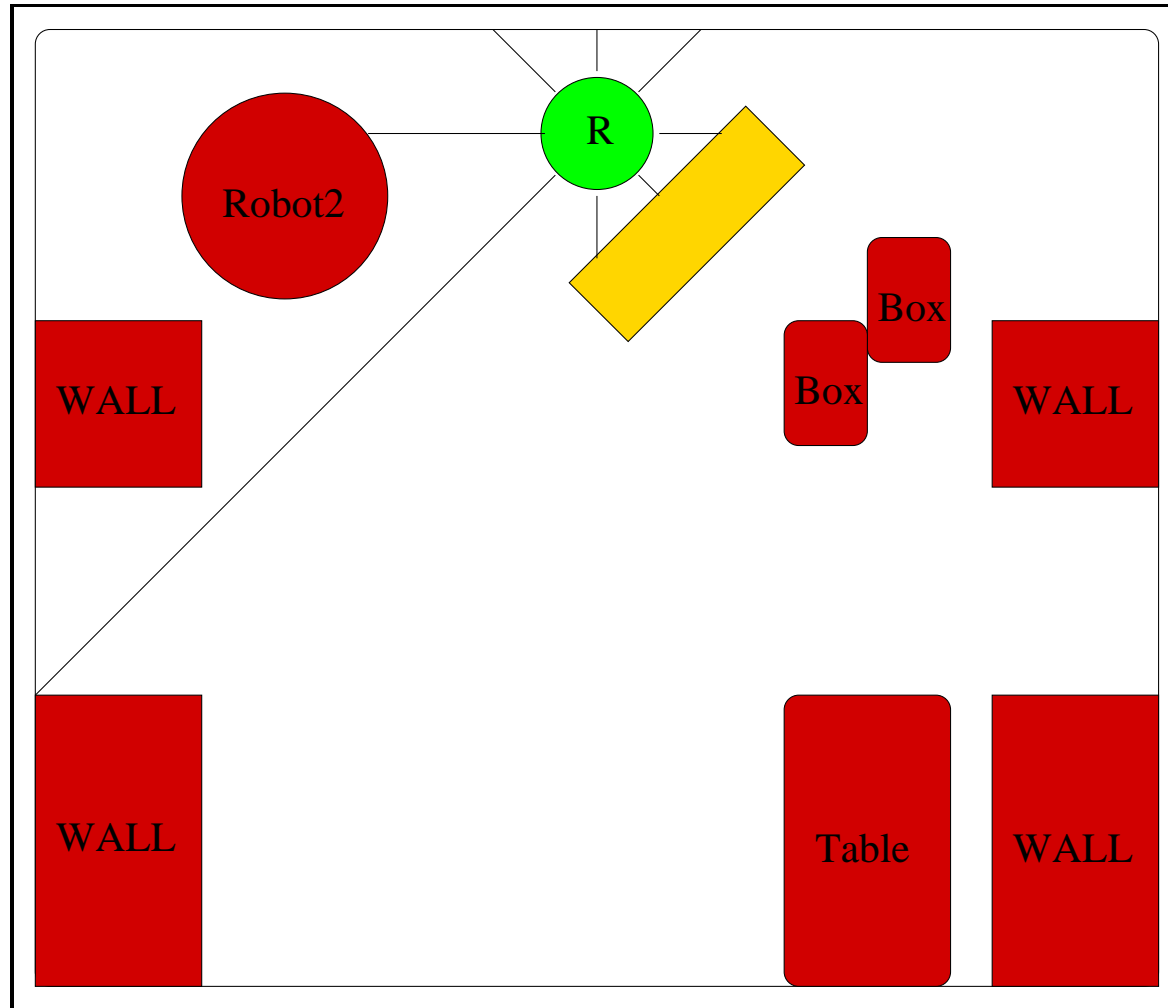
Robot Motion using Sonars



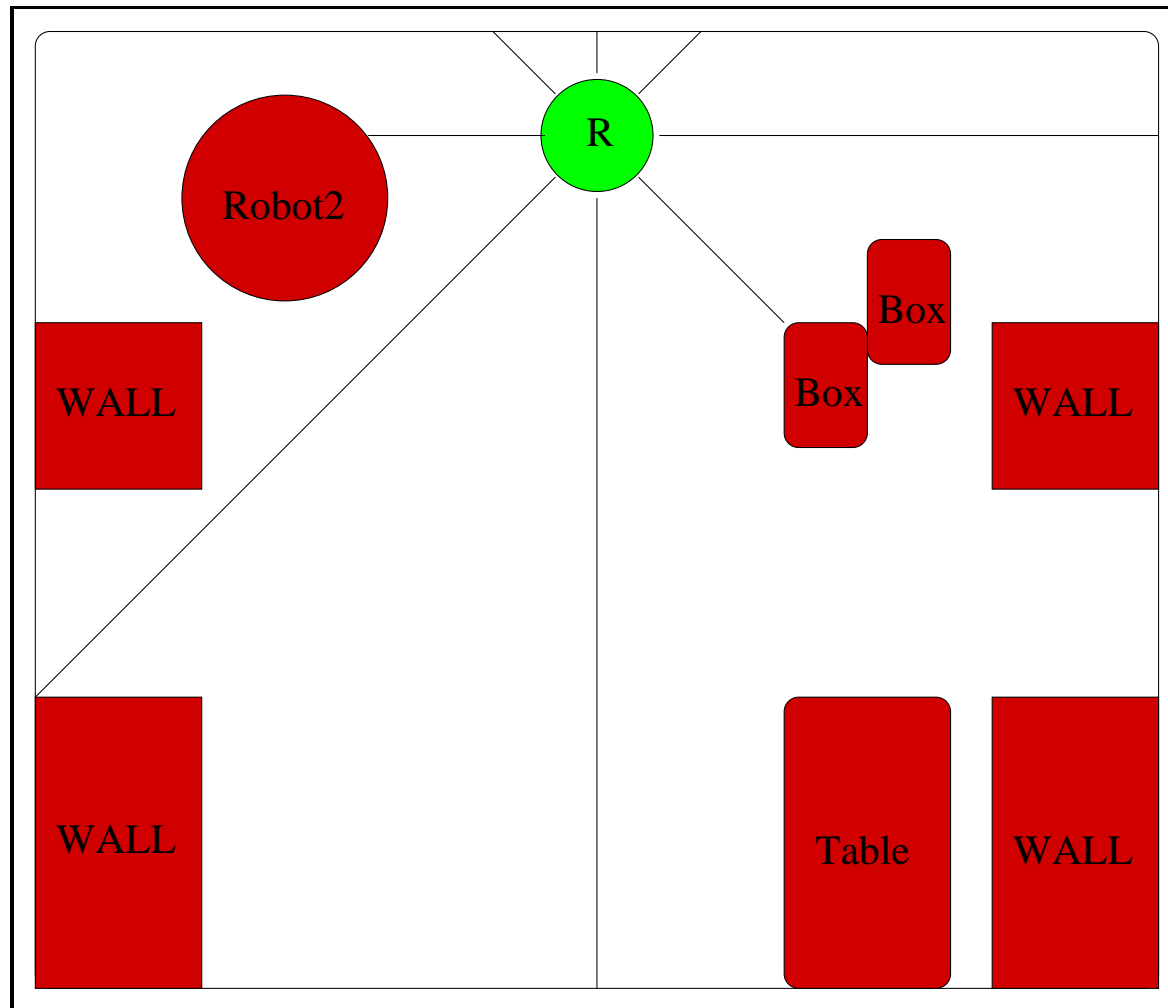
Robot Motion using Sonars



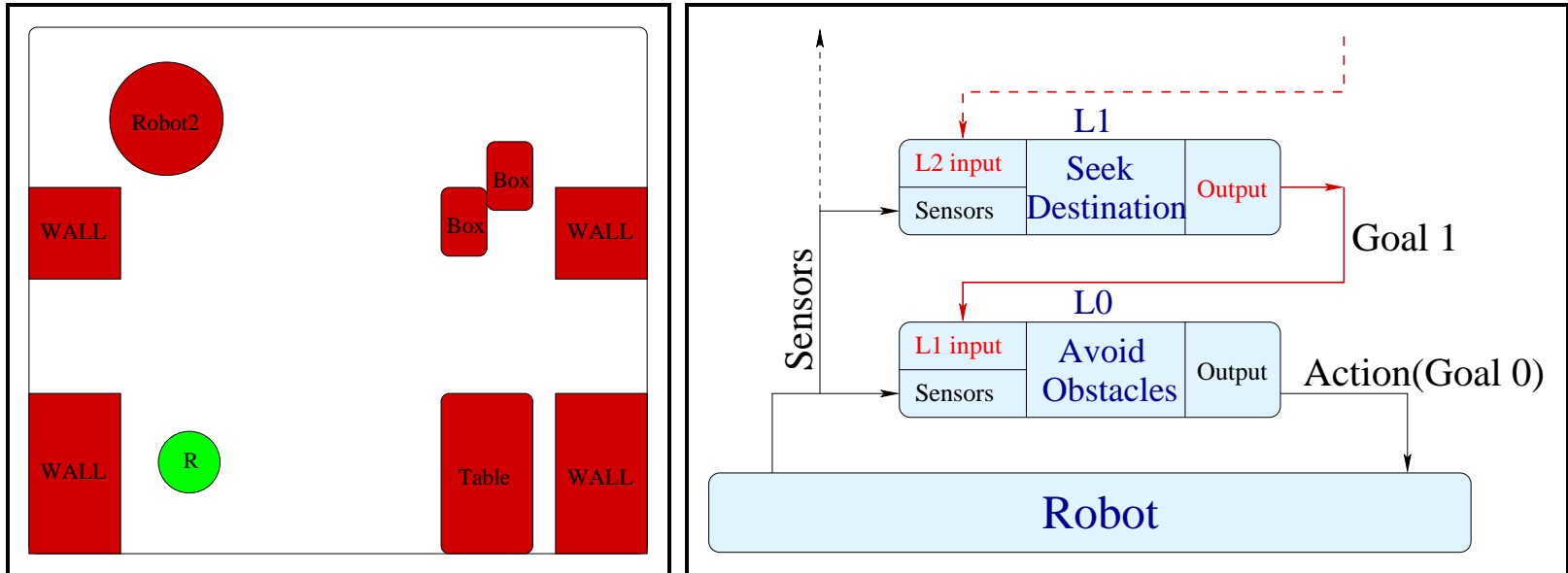
Robot Motion using Sonars



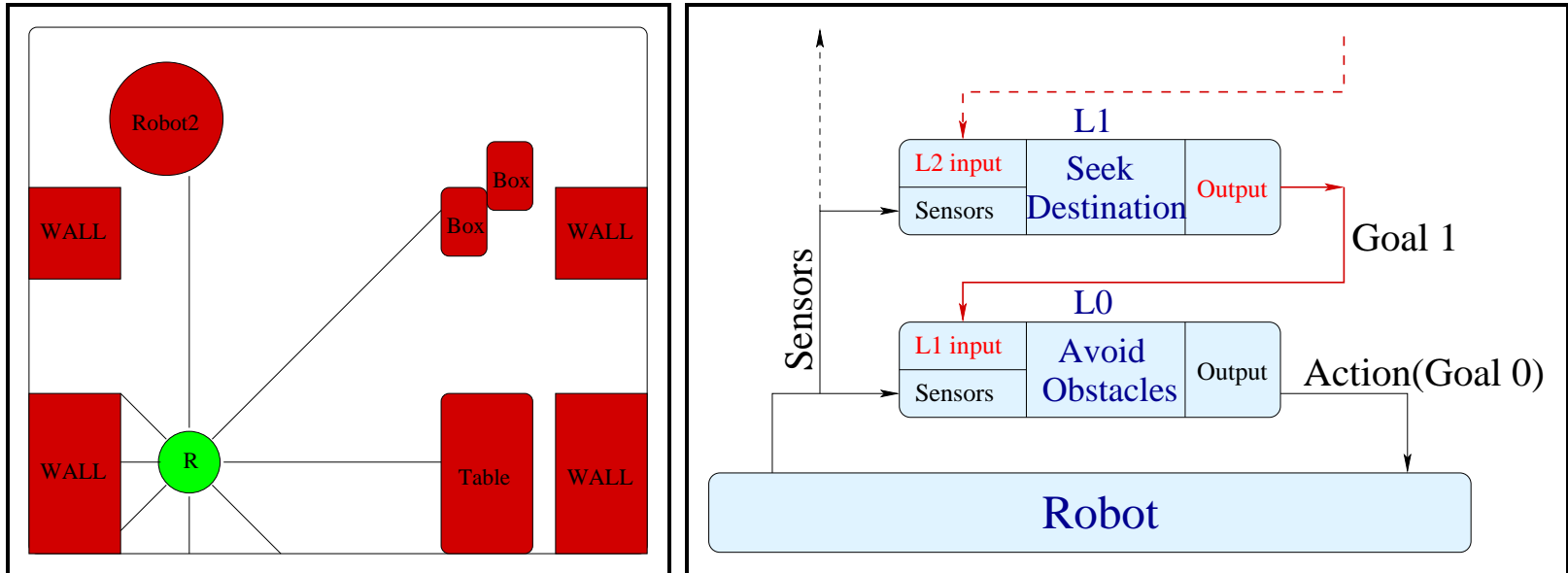
Robot Motion using Sonars



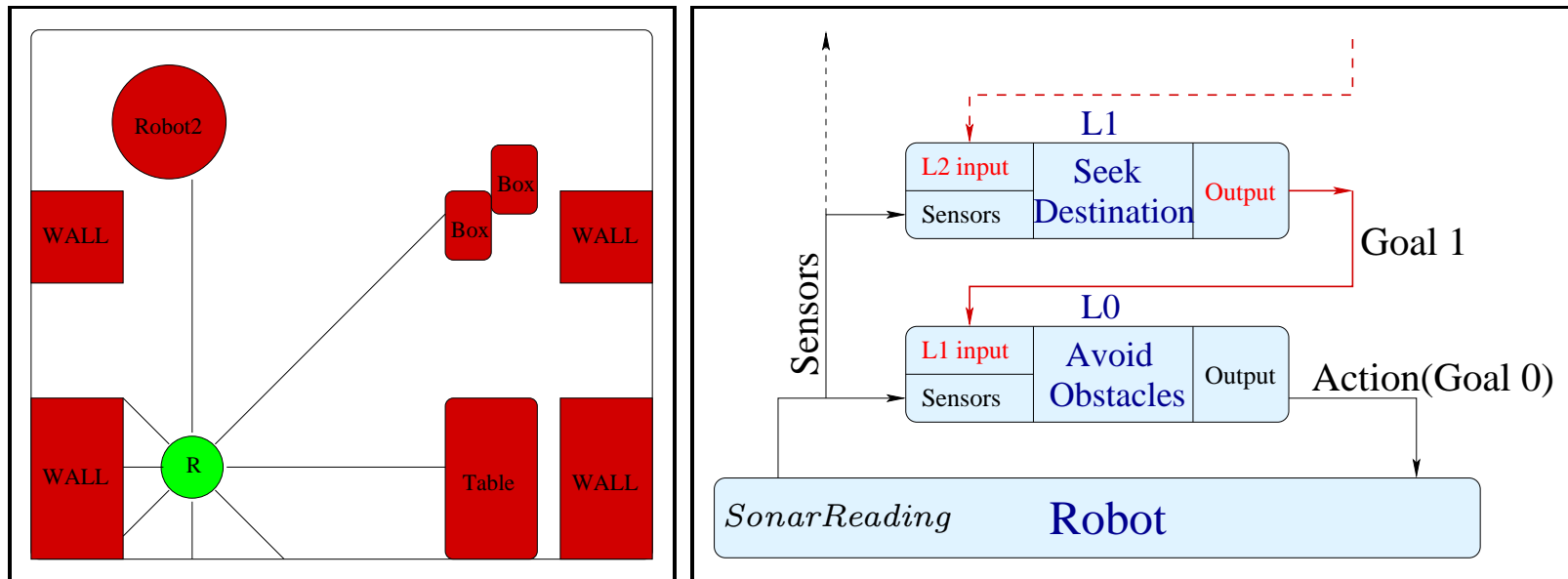
Logic-Based Subsumption (LSA)



Logic-Based Subsumption (LSA)

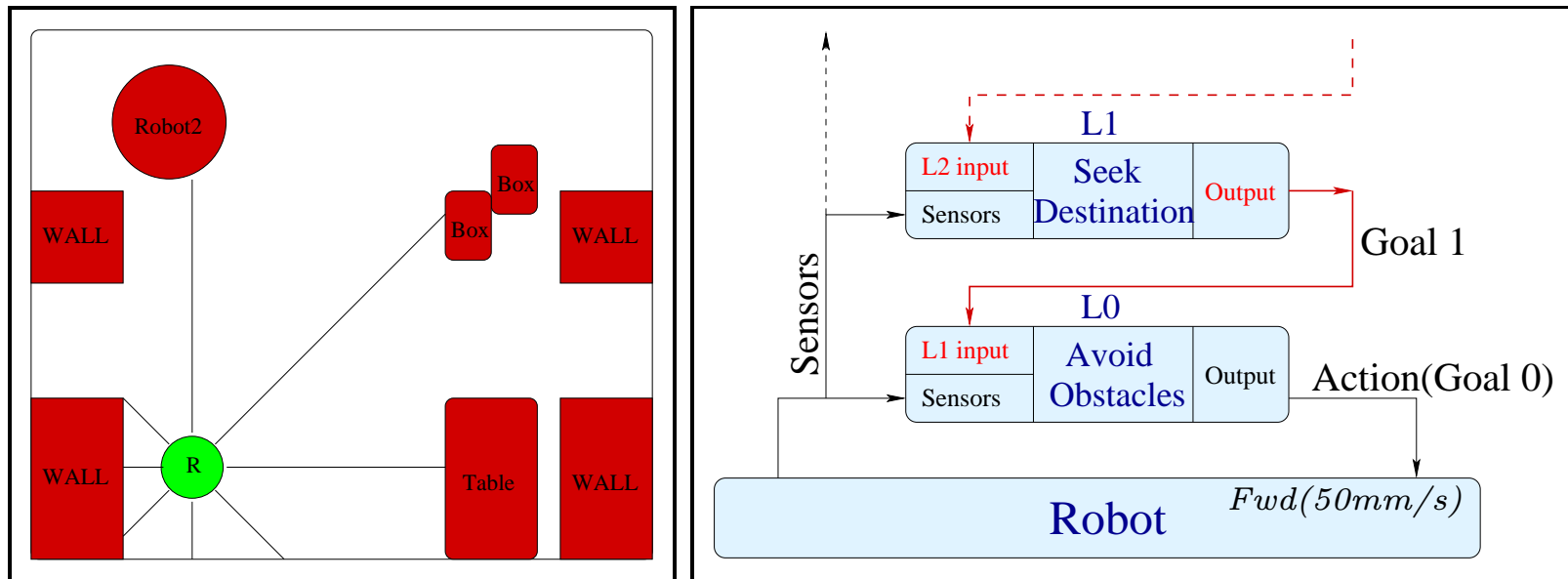


Logic-Based Subsumption (LSA)



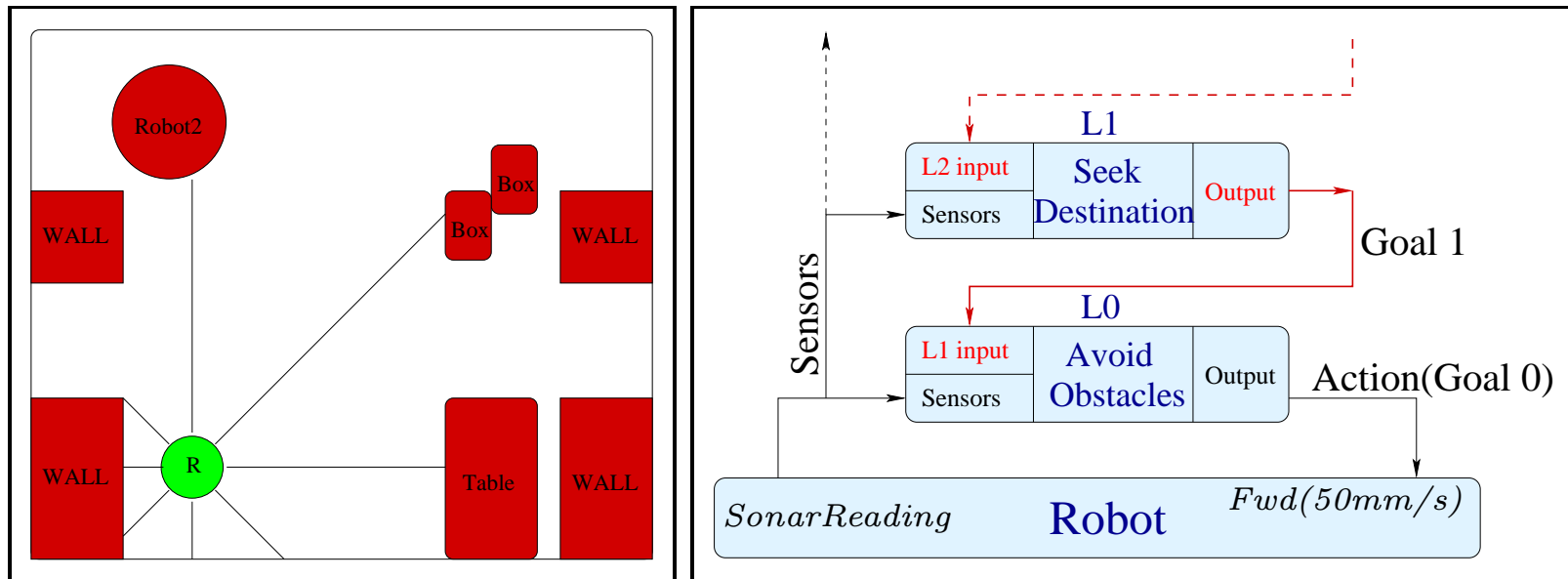
Layer 0 requests sensory information from the robot.
It receives: $SonarReading(s8, 5m), \dots$

Logic-Based Subsumption (LSA)



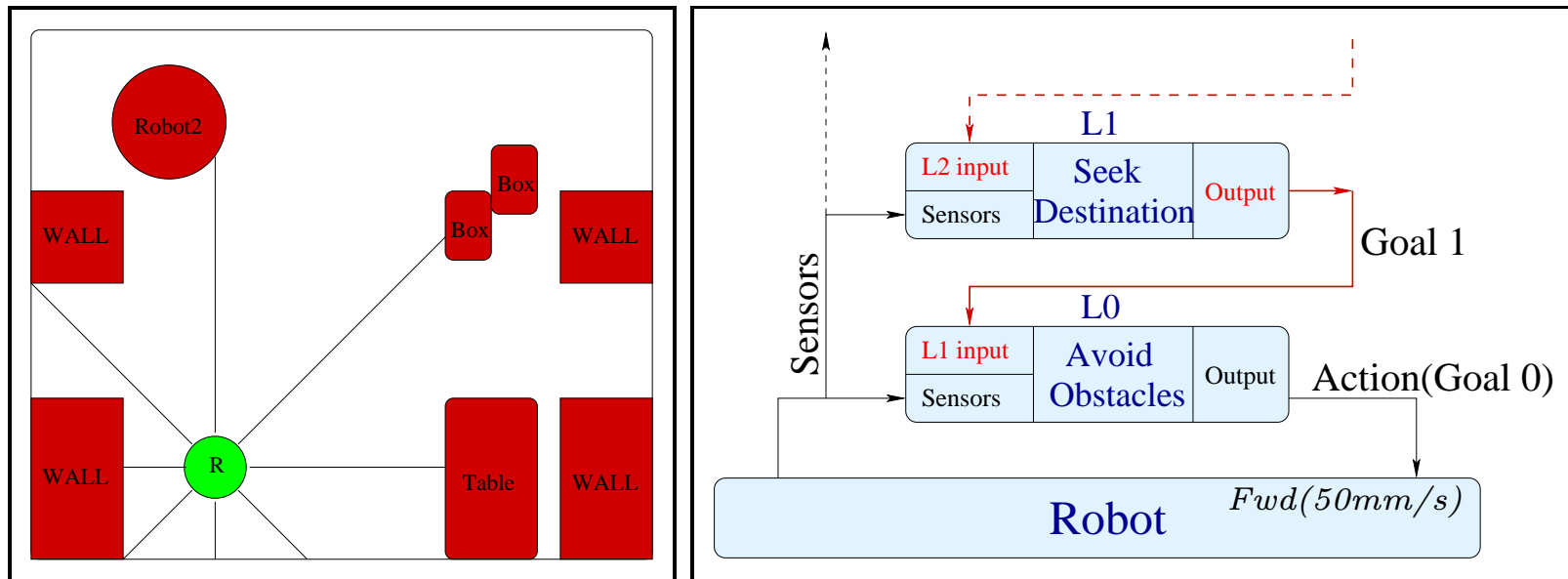
Layer 0 tries to prove $fwd(Speed)$ or $turn(Ang)$.
It proves: $fwd(50mm/s)$.

Logic-Based Subsumption (LSA)



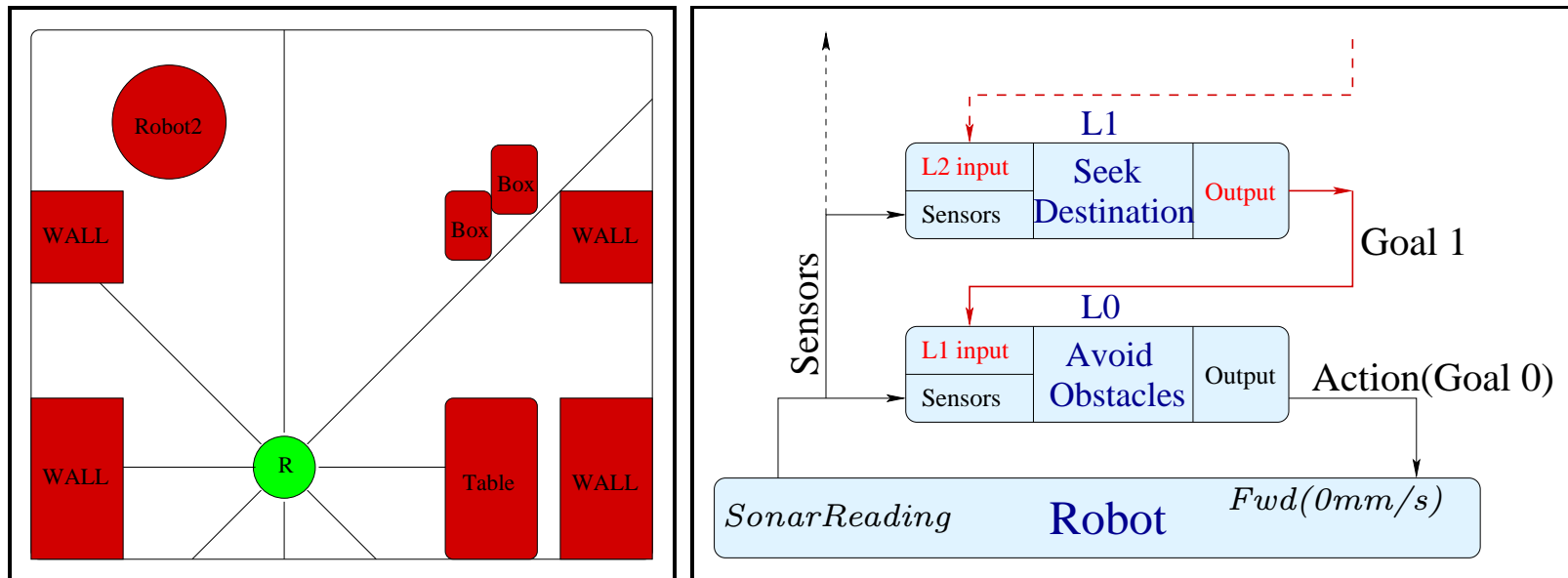
Layer 0 requests sensory information from the robot.
 It receives: $SonarReading(s8, 5m), \dots$

Logic-Based Subsumption (LSA)



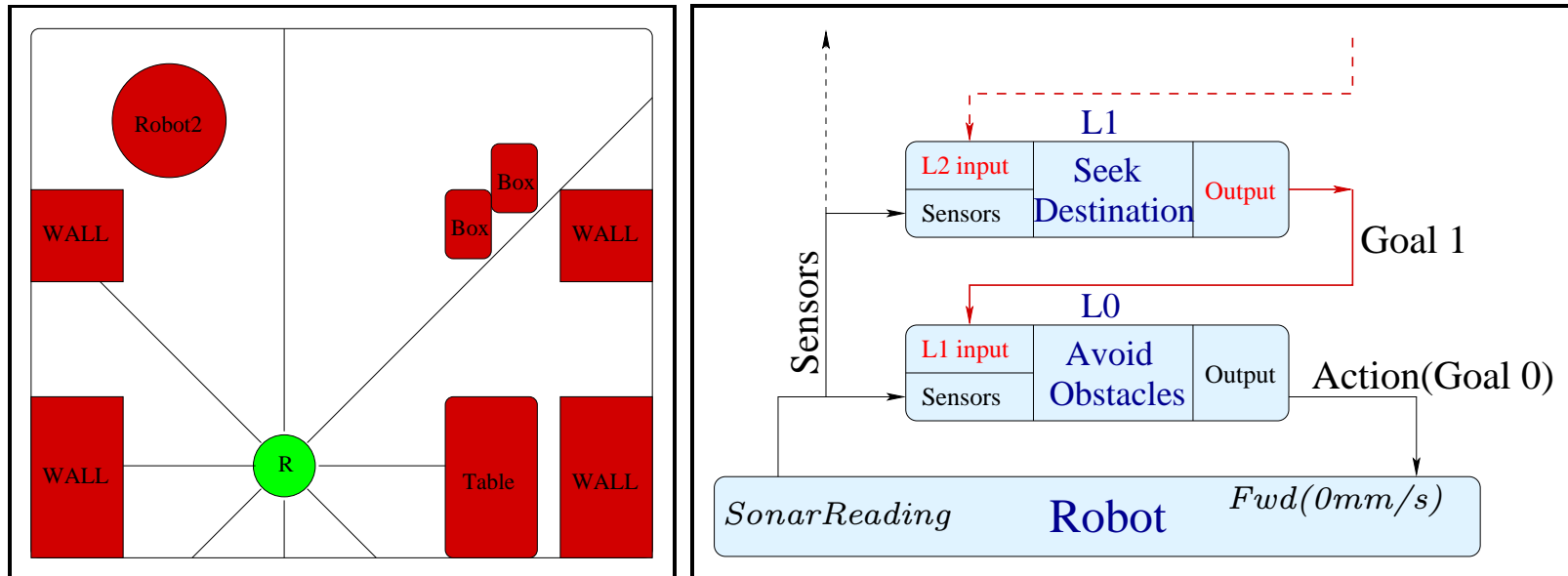
Layer 0 tries to prove $fwd(Speed)$ or $turn(Ang)$.
It proves: $fwd(48mm/s)$.

Logic-Based Subsumption (LSA)



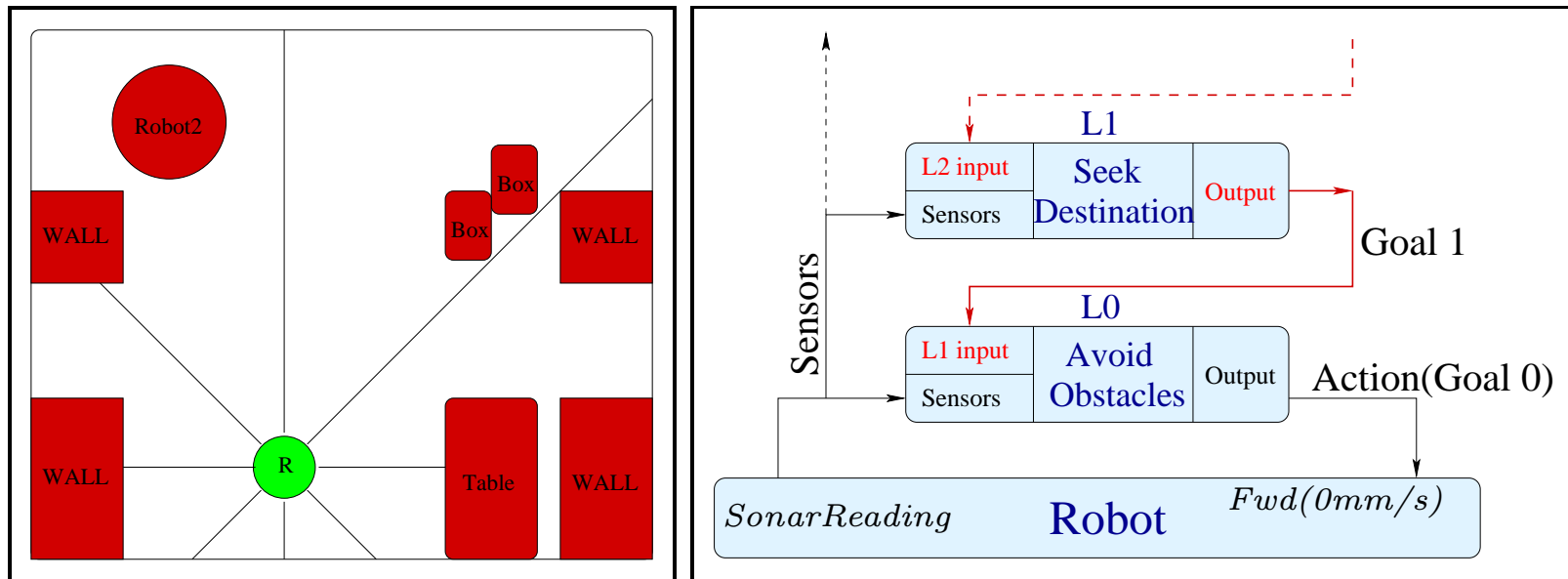
Layer 0 tries to prove $fwd(Speed)$ or $turn(Ang)$.
It proves: $fwd(0mm/s)$.

Logic-Based Subsumption (LSA)



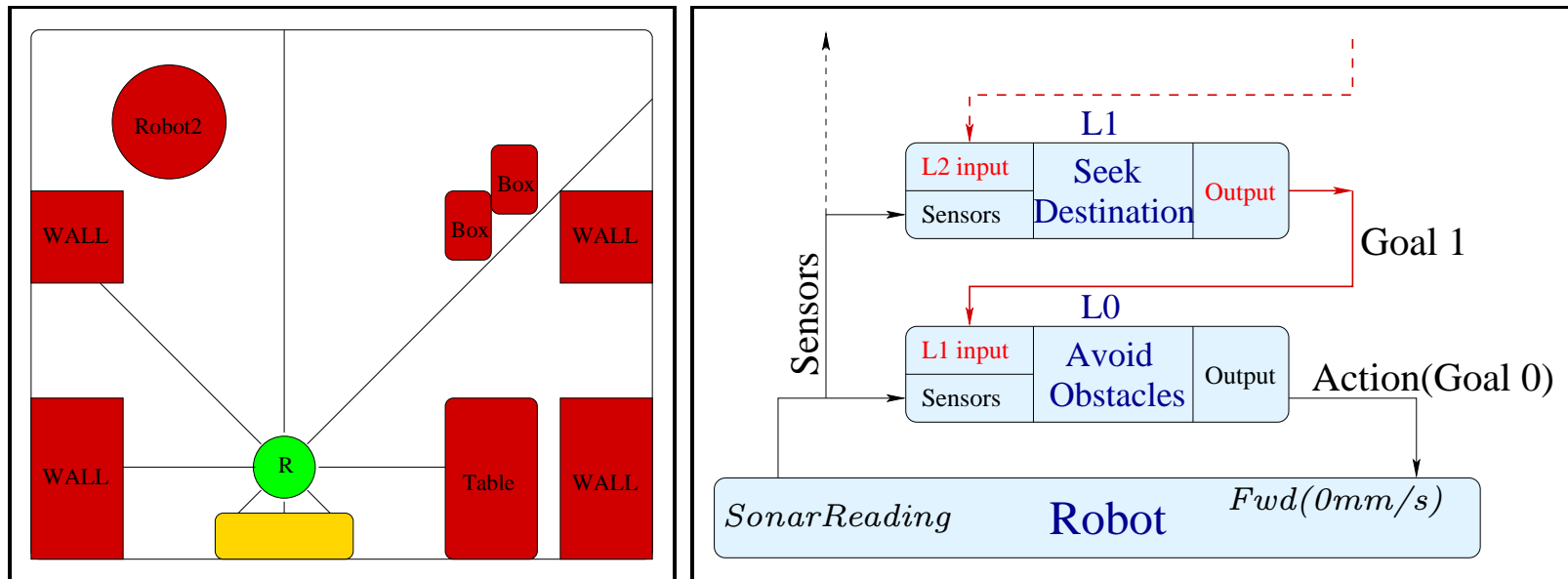
Layer 1 requests sensory information from the robot.
It receives: $SonarReading(s8, 5m), \dots,$
 $CurrentLoc(100, 800), CurrentDir(0).$

Logic-Based Subsumption (LSA)



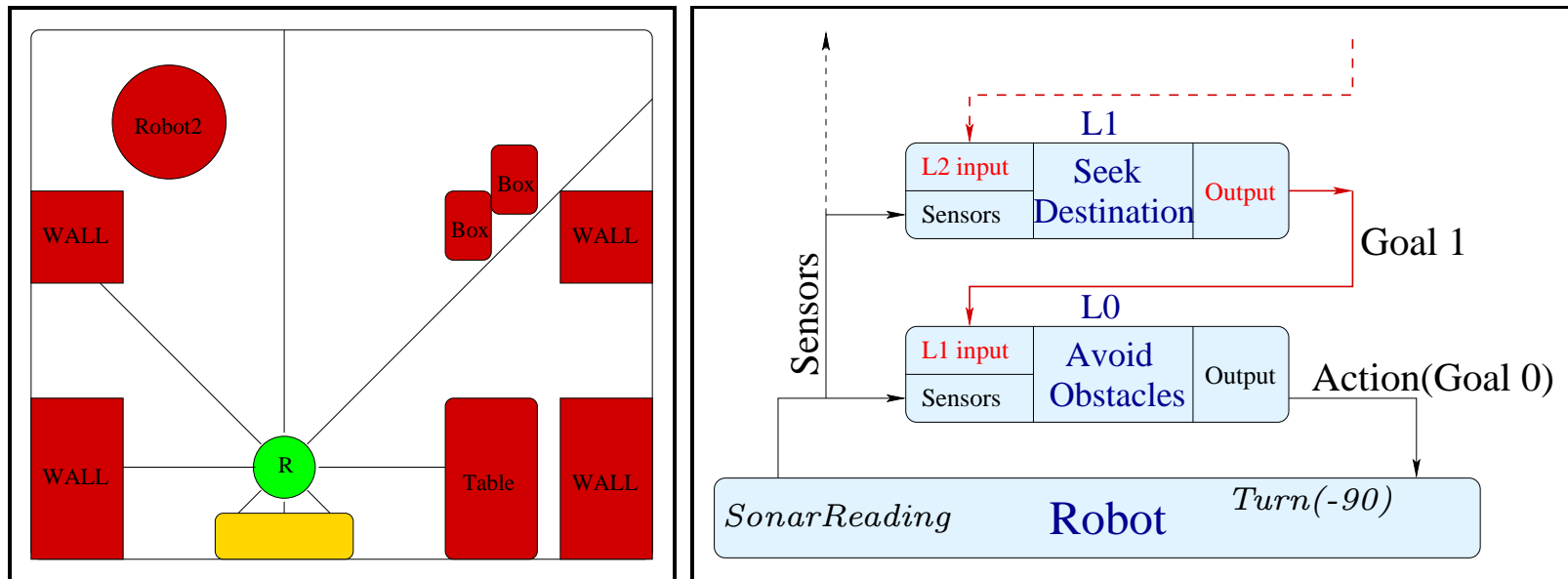
Layer 1 tries to prove $pushing_object(Obj)$.
 It proves: $pushing_object(z)$.

Logic-Based Subsumption (LSA)



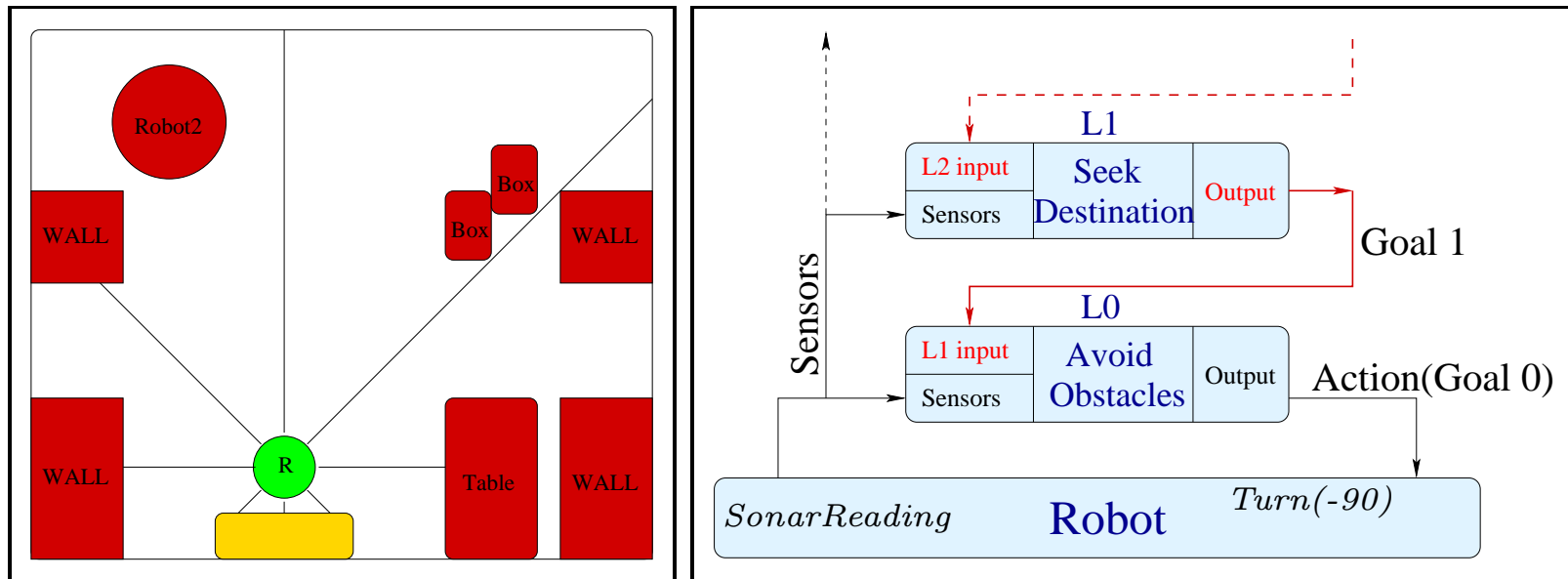
Layer 0 requests sensory information from the robot.
 It receives: *SonarReading(s8, 5m),...*

Logic-Based Subsumption (LSA)



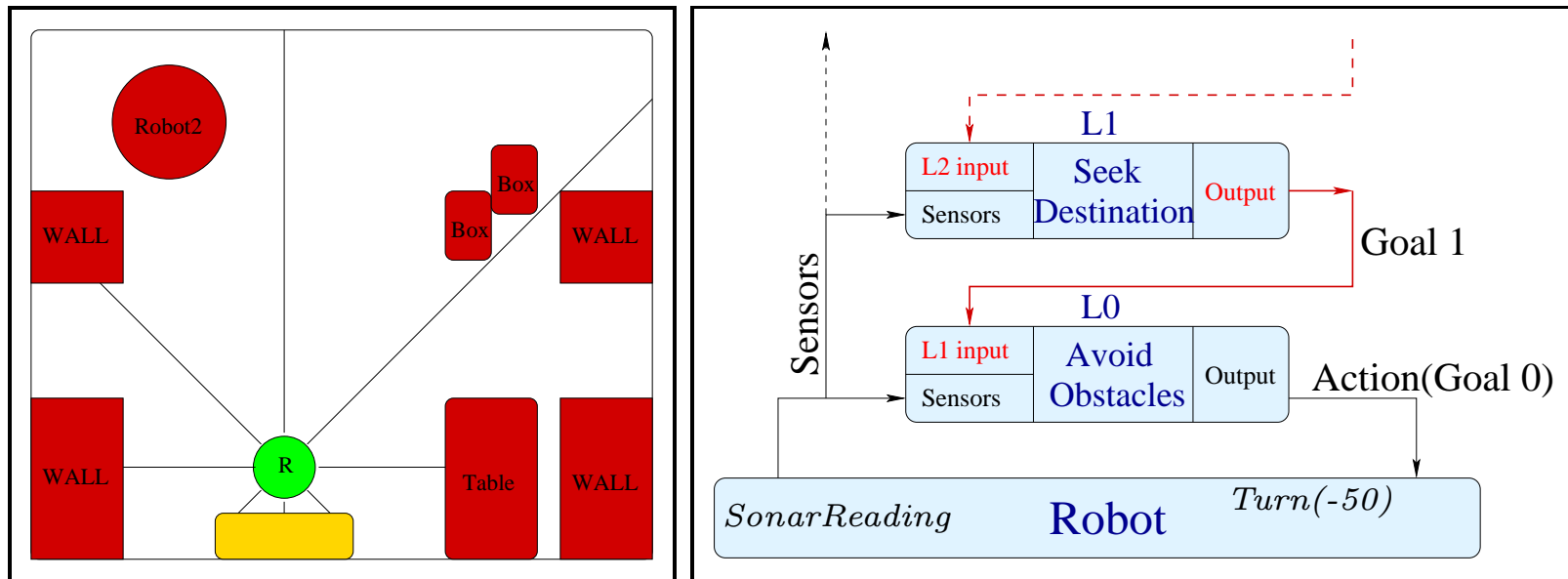
Layer 0 tries to prove $fwd(Speed)$ or $turn(Ang)$.
 It proves: $turn(-90)$.

Logic-Based Subsumption (LSA)



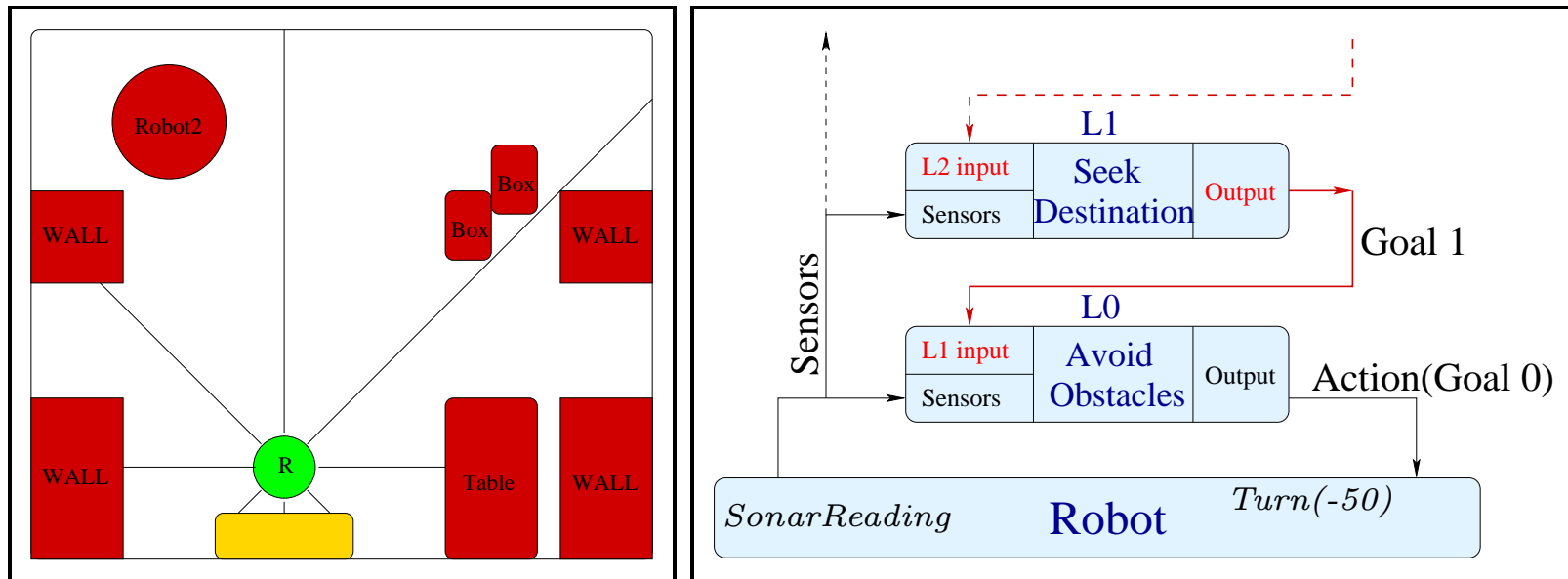
Layer 0 requests sensory information from the robot.
 It receives: $SonarReading(s8, 7m), \dots$

Logic-Based Subsumption (LSA)



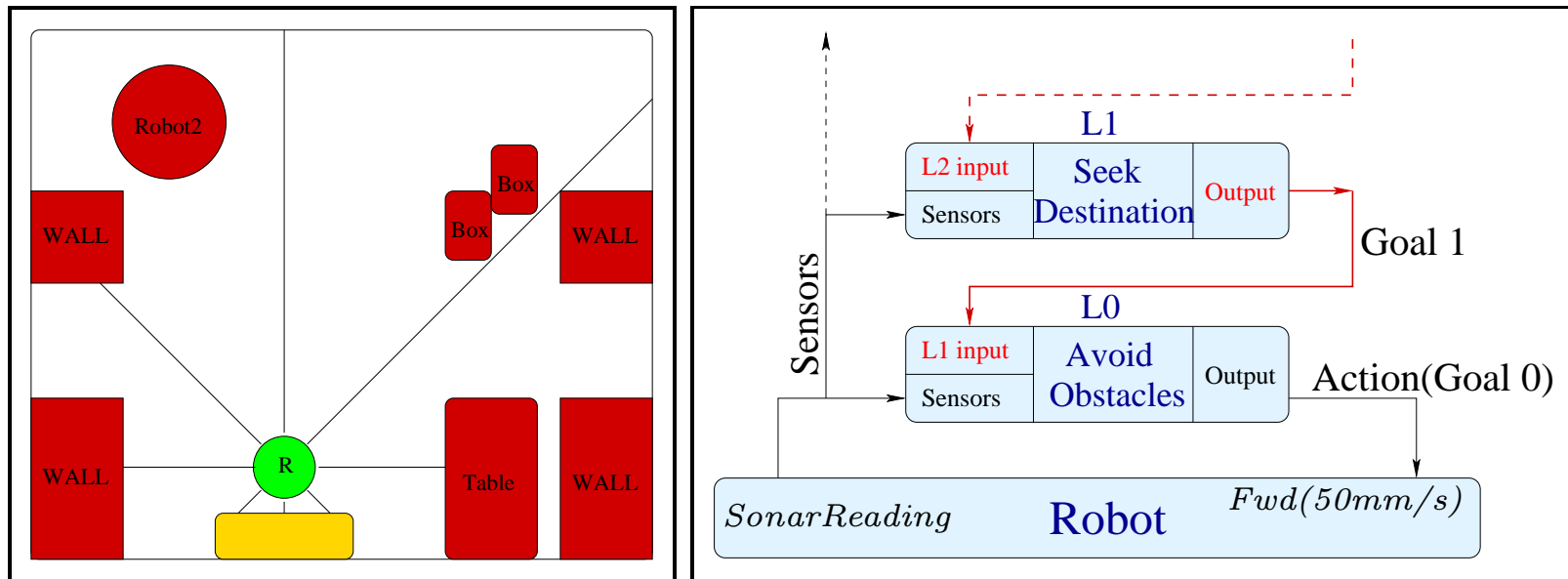
Layer 0 tries to prove $fwd(Speed)$ or $turn(Ang)$.
 It proves: $turn(-50)$.

Logic-Based Subsumption (LSA)



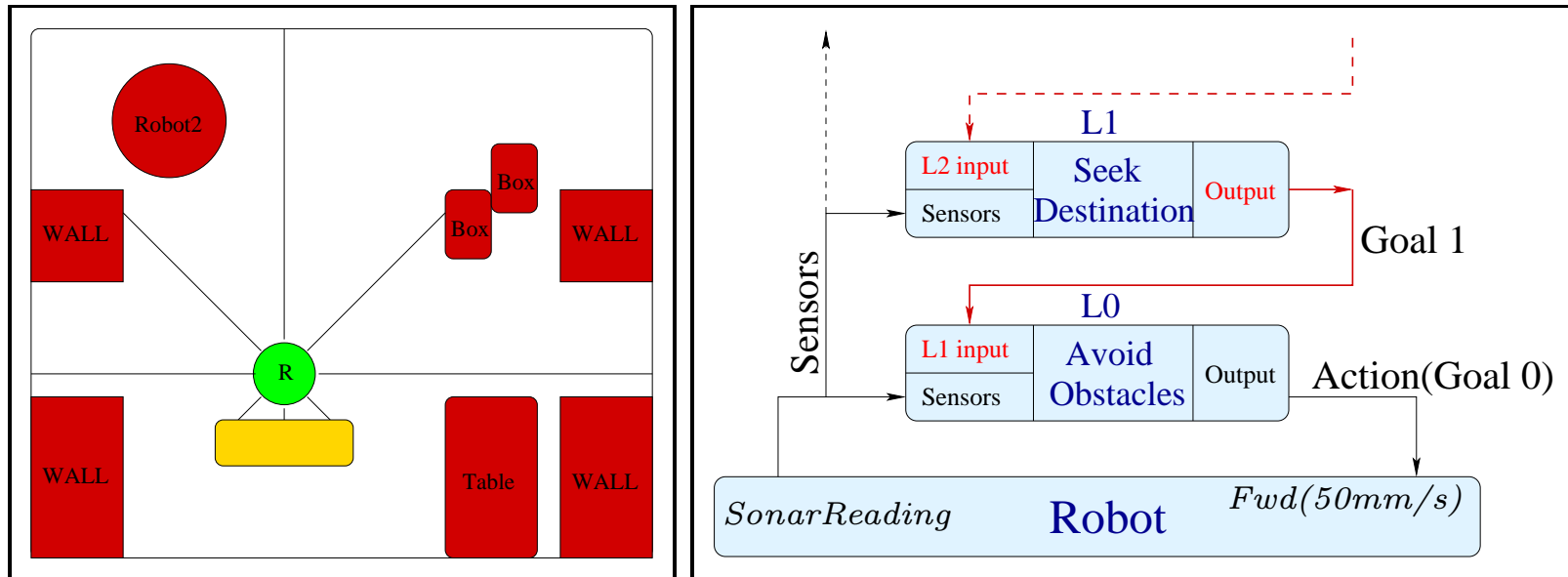
Layer 0 requests sensory information from the robot.
 It receives: $SonarReading(s8, 10m), \dots$

Logic-Based Subsumption (LSA)



Layer 0 tries to prove $fwd(Speed)$ or $turn(Ang)$.
 It proves: $Fwd(50mm/s)$.

Logic-Based Subsumption (LSA)

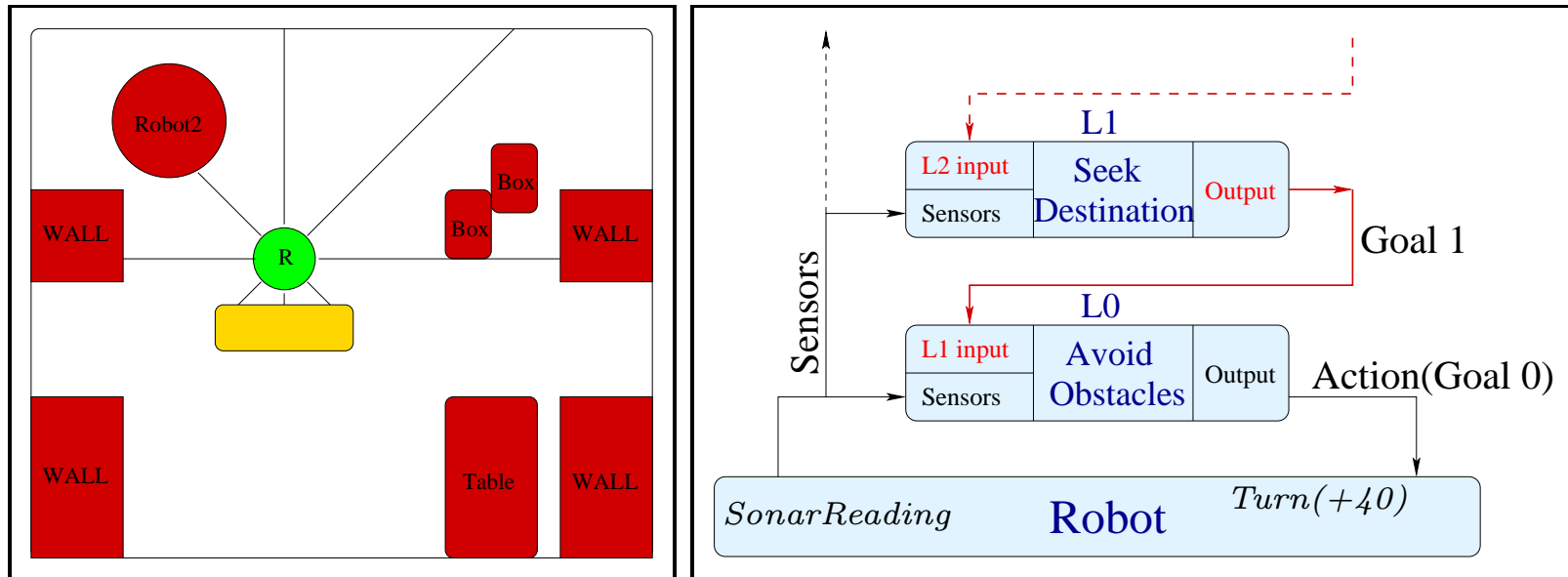


Layer 0 tries to prove $fwd(Speed)$ or $turn(Ang)$.

It proves: $Fwd(50mm/s)$.

Layer 1 proves: $pushing_object(z), \dots$

Logic-Based Subsumption (LSA)

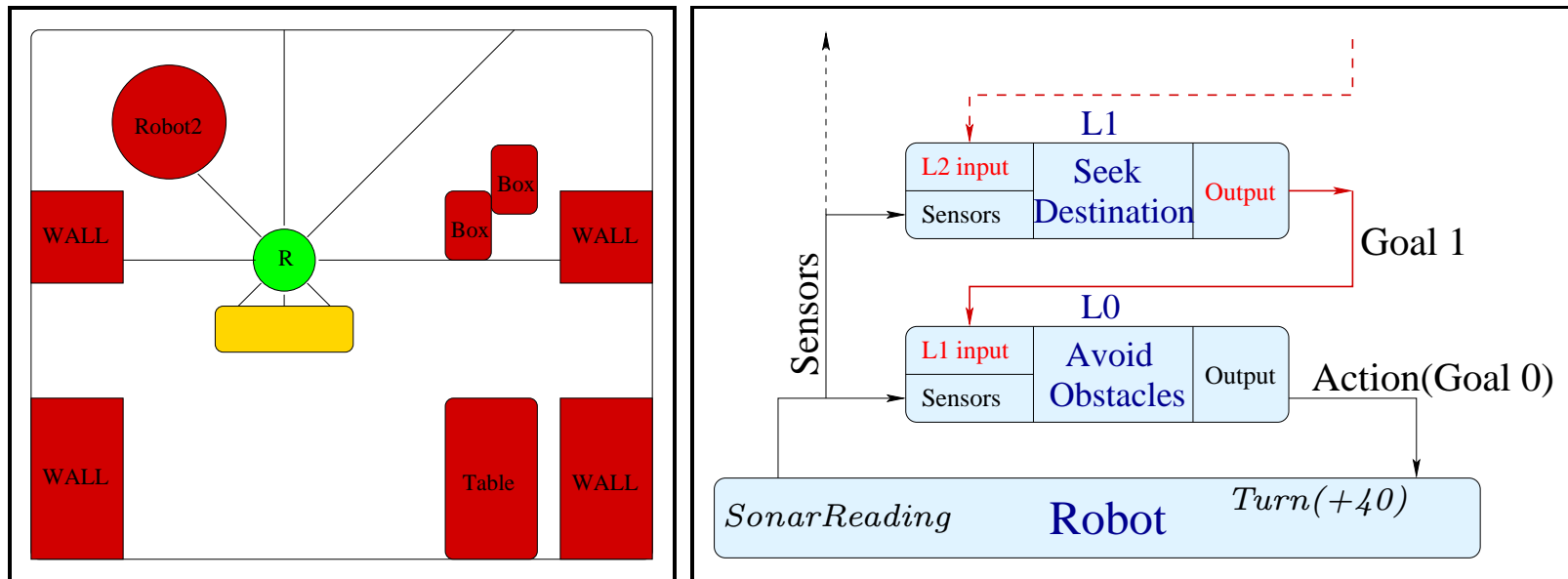


Layer 0 tries to prove $fwd(Speed)$ or $turn(Ang)$.

It proves: $Turn(+40)$.

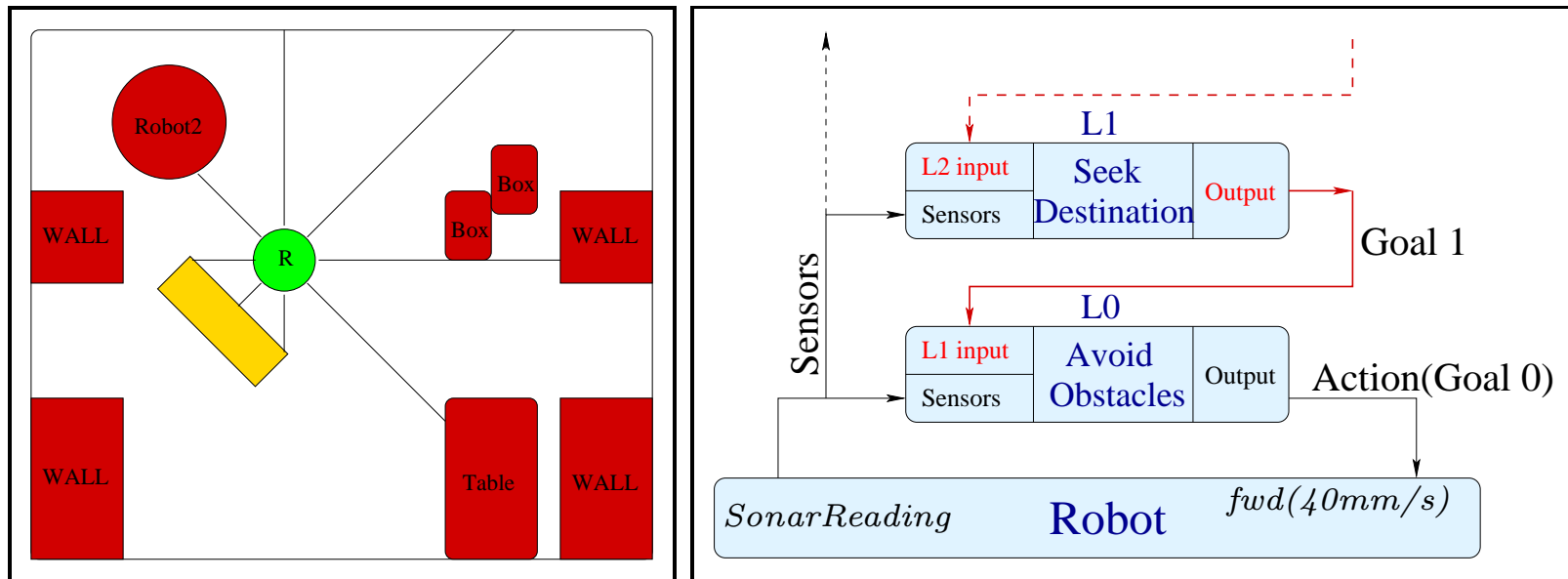
Layer 1 proves: $pushing_object(z), \dots$

Logic-Based Subsumption (LSA)



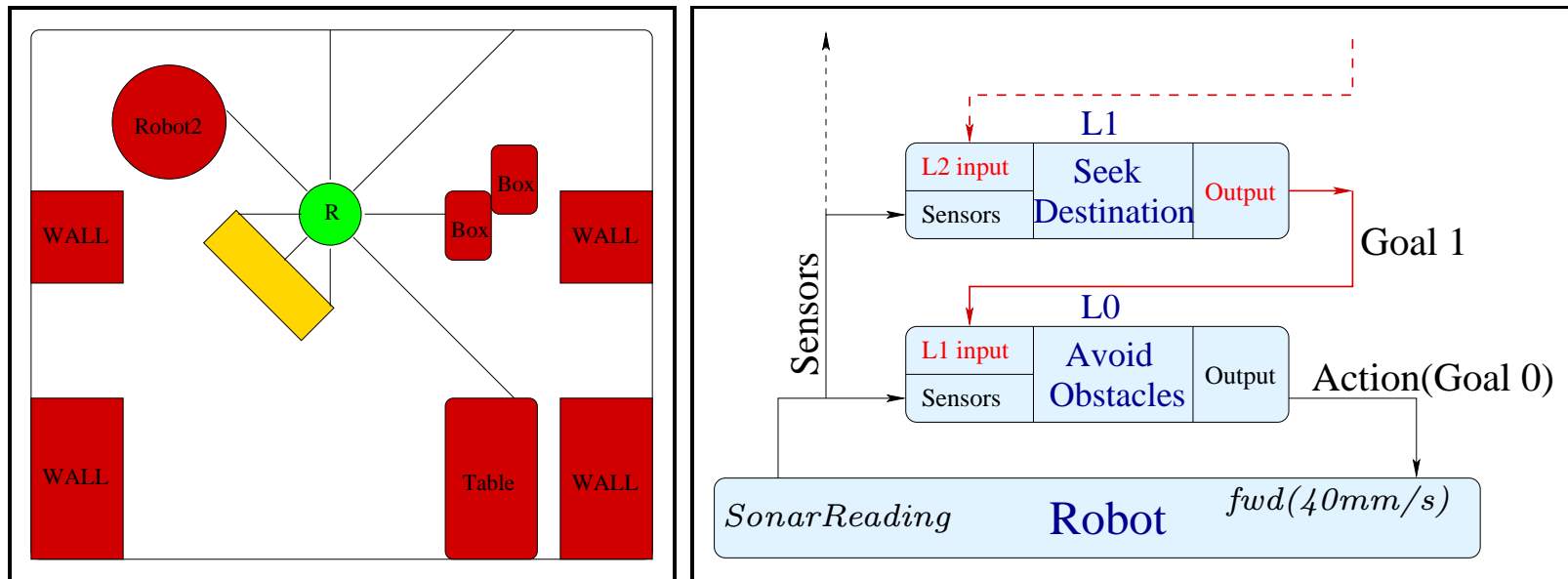
Layer 0 tries to prove $fwd(Speed)$ or $turn(Ang)$.
 It proves: $Turn(+20)$.

Logic-Based Subsumption (LSA)



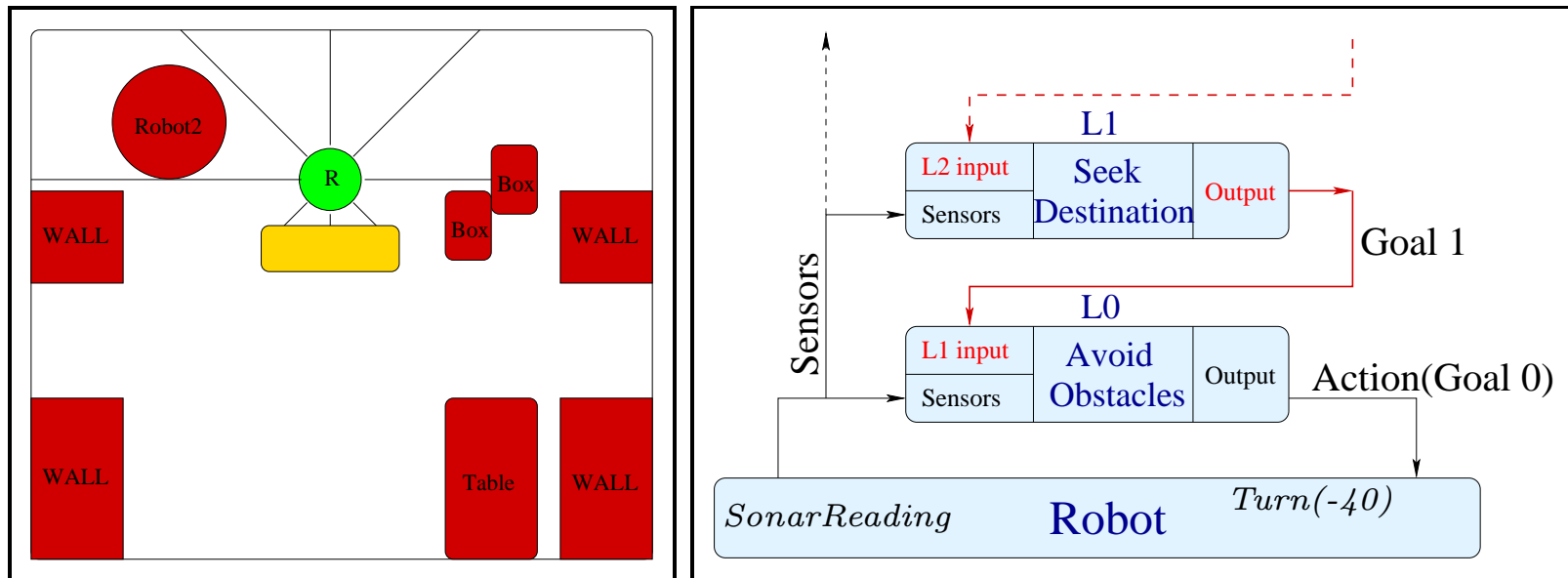
Layer 0 tries to prove $fwd(Speed)$ or $turn(Ang)$.
 It proves: $fwd(40mm/s)$.

Logic-Based Subsumption (LSA)



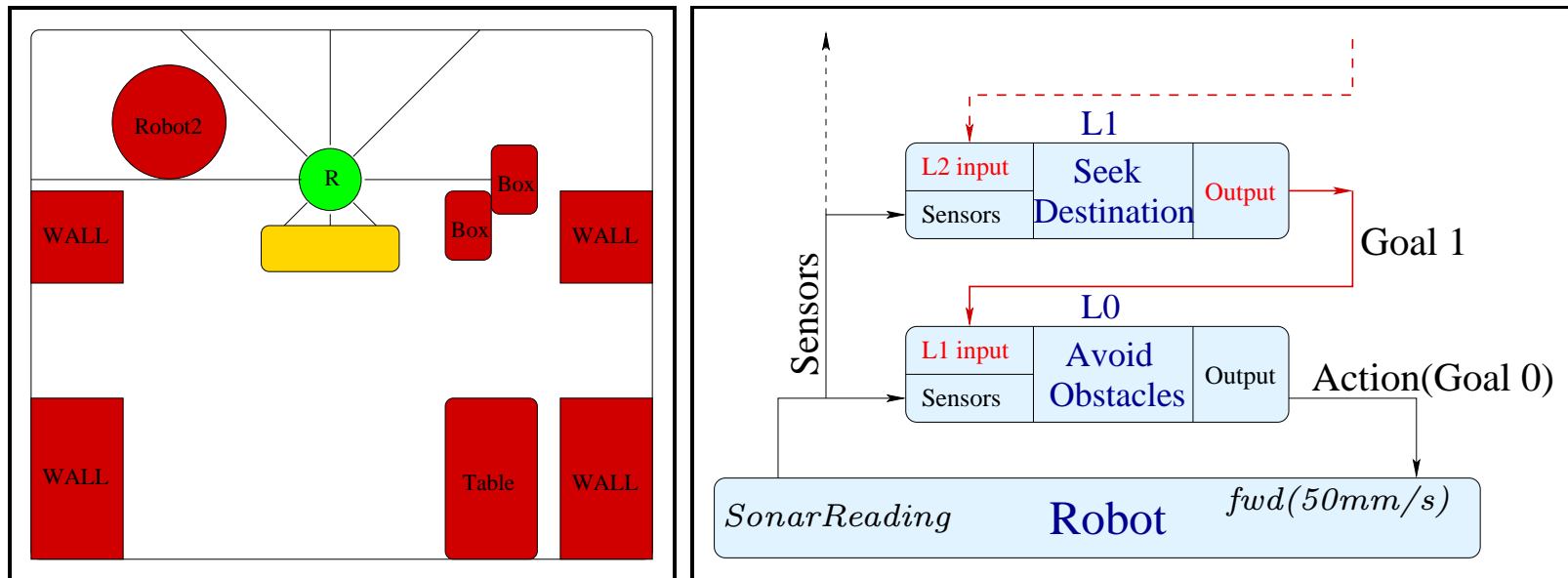
Layer 1 proves *pushing_object(z)*.

Logic-Based Subsumption (LSA)



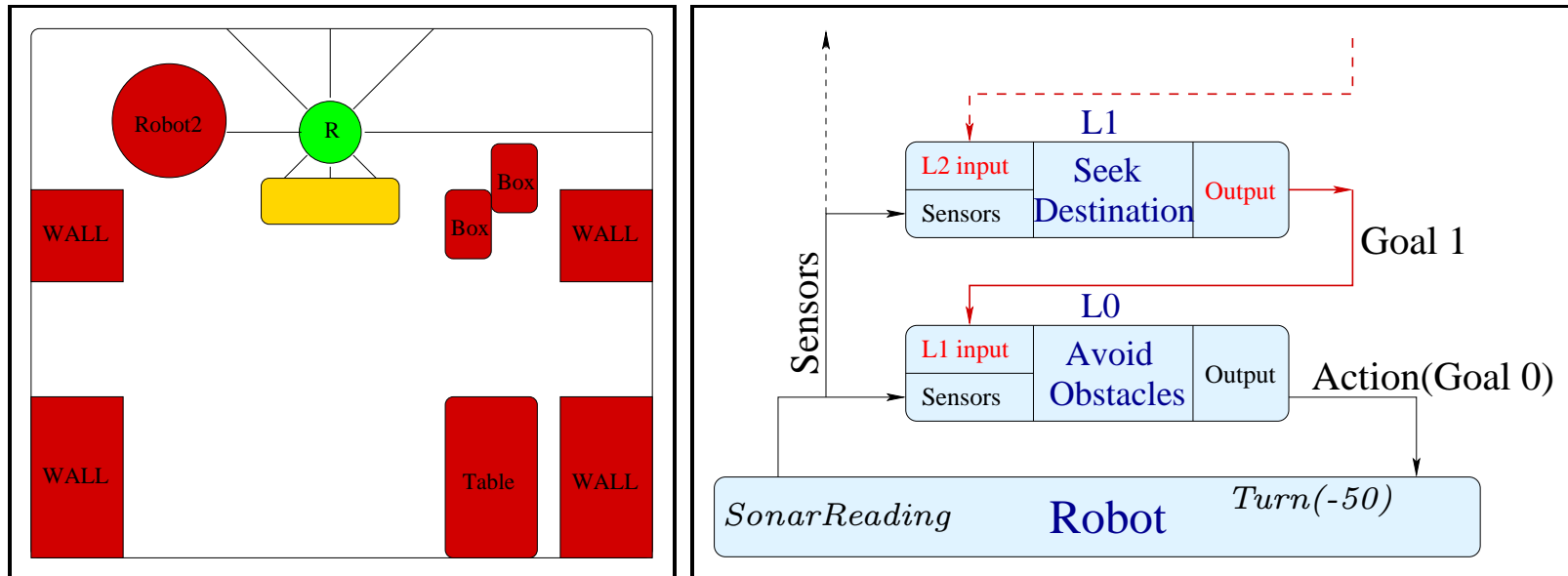
Layer 0 tries to prove $fwd(Speed)$ or $turn(Ang)$.
 It proves: $turn(-40)$.

Logic-Based Subsumption (LSA)

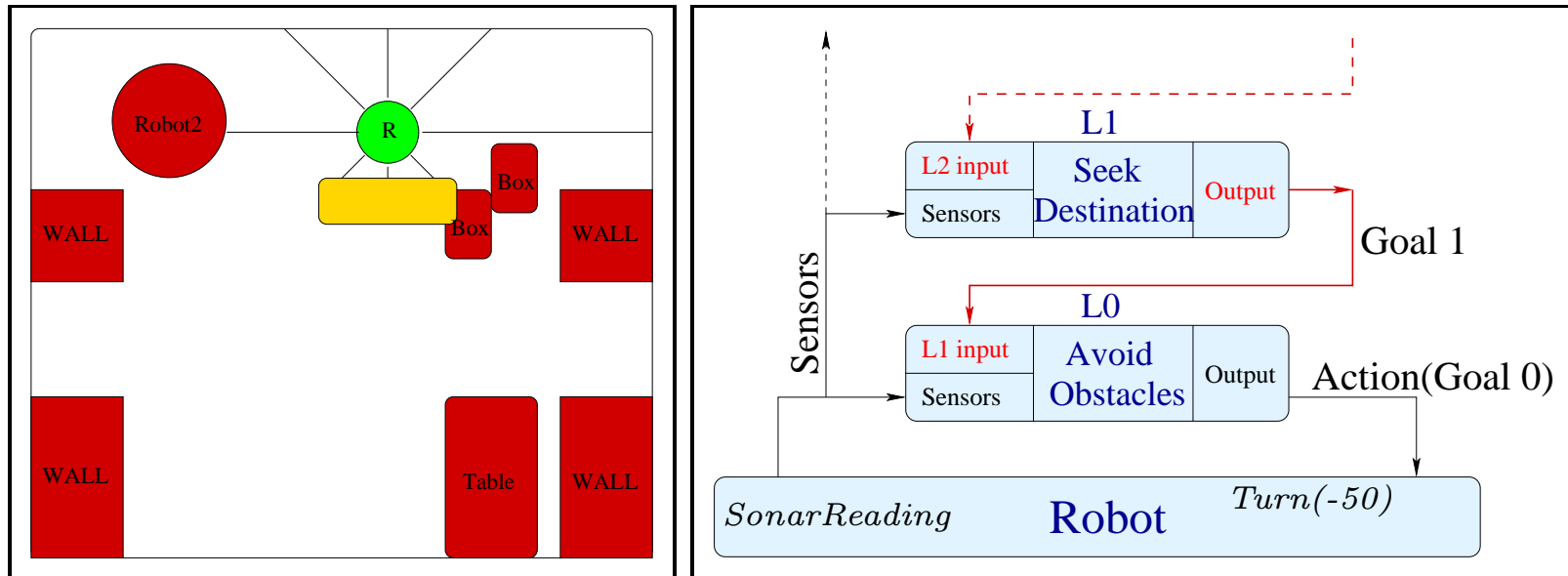


Layer 0 tries to prove $fwd(Speed)$ or $turn(Ang)$.
 It proves: $fwd(50mm/s)$.

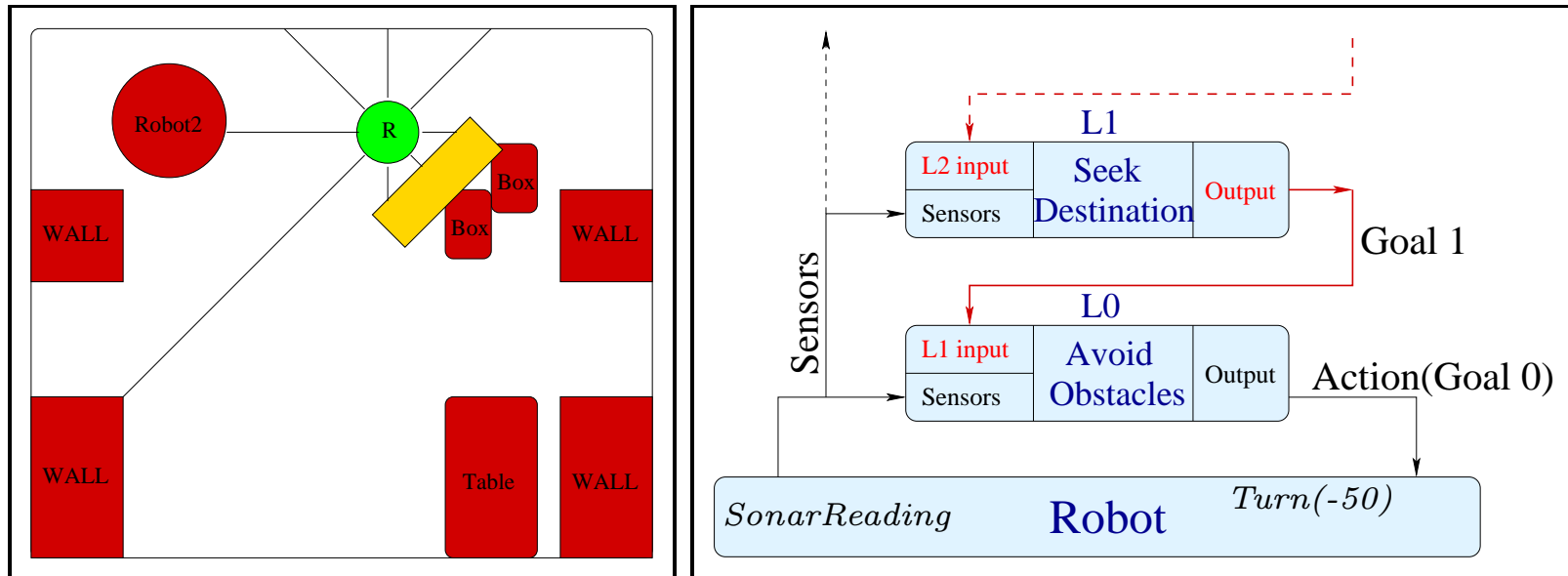
Logic-Based Subsumption (LSA)



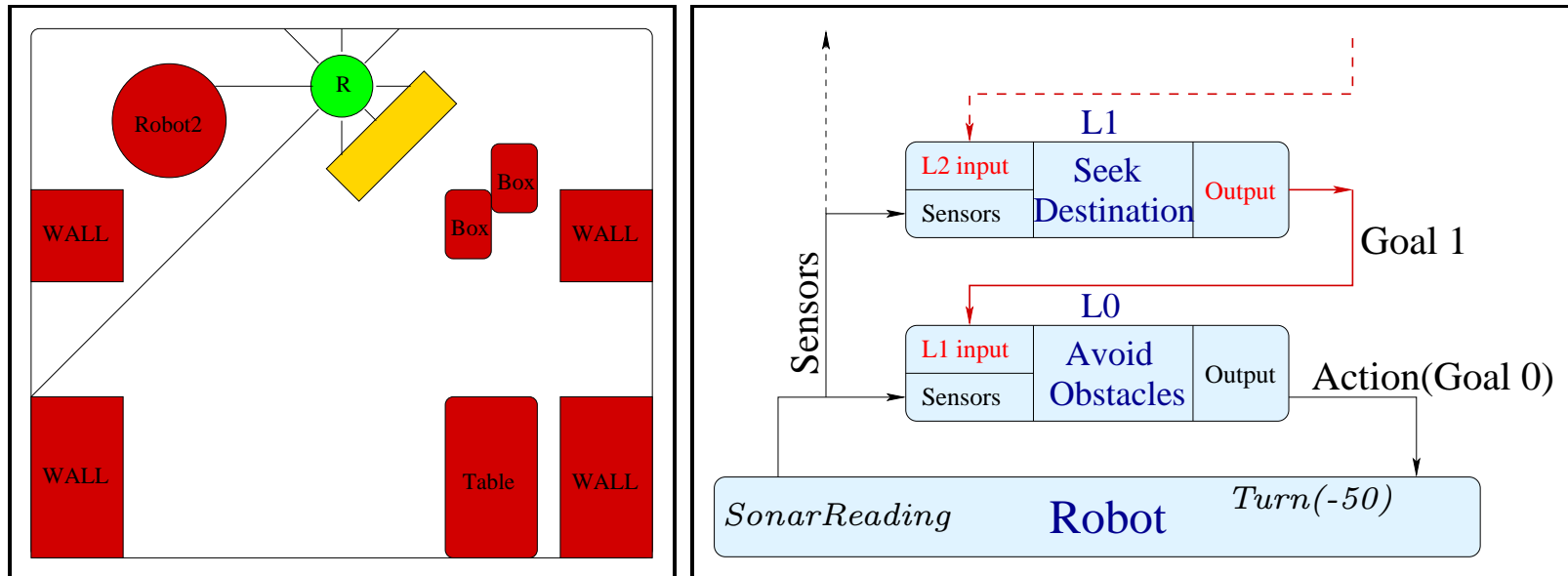
Logic-Based Subsumption (LSA)



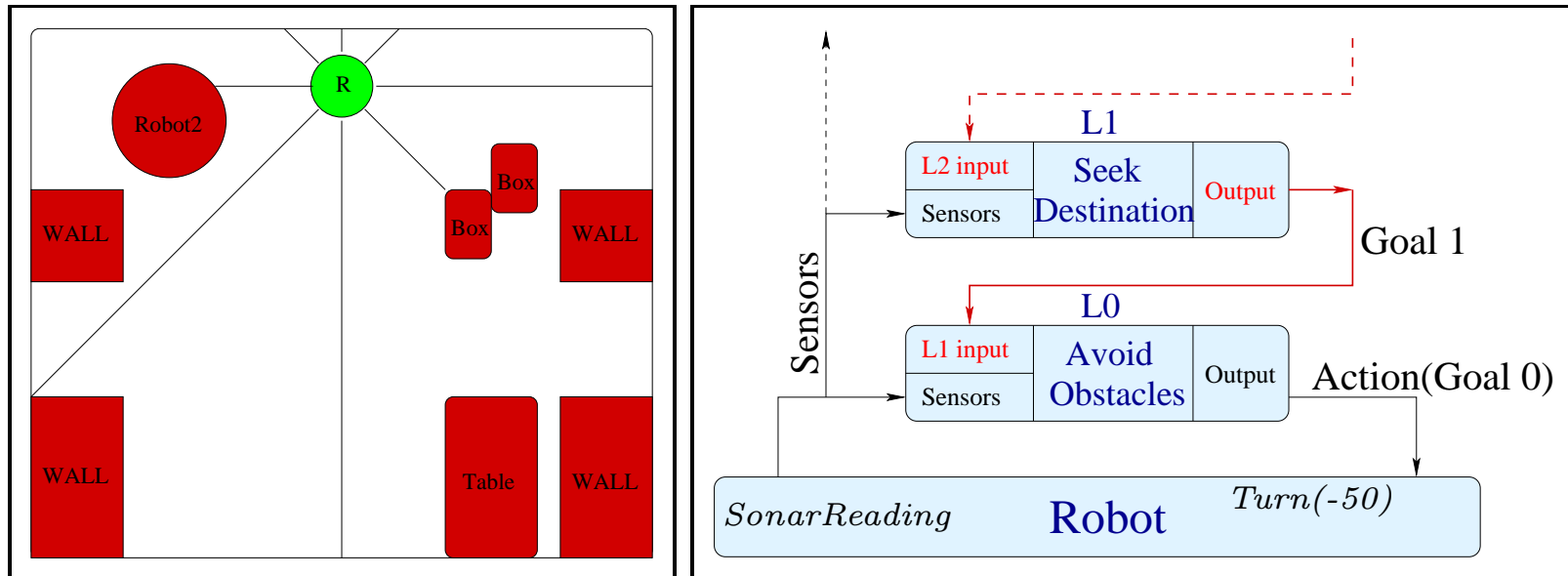
Logic-Based Subsumption (LSA)



Logic-Based Subsumption (LSA)

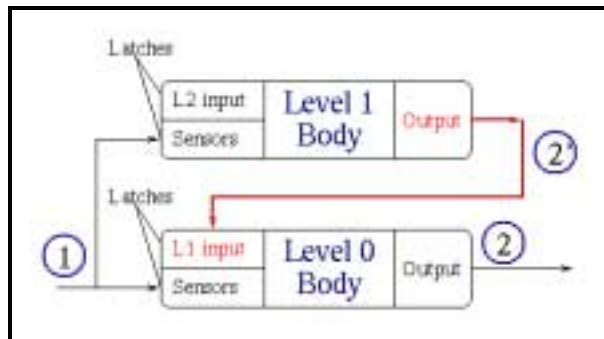


Logic-Based Subsumption (LSA)



Subsumption via Nonmonotonic Reasoning

- Each layer makes default behavioral assumptions.
- Higher layer theories can insert assertions into lower layer theories overriding these assumptions.



The LSA Algorithm

During each cycle, each layer:

1. Asserts sensory data into its Sensory Latch.
2. Combines axioms in its Body theory with those in its Sensory and Input Latches.
3. Has its theorem prover attempt to prove its goal from the combined theory given its default assumptions.
4. Asserts its proven goal in the Input Latches of lower layers.

Semantics For Layered Theories

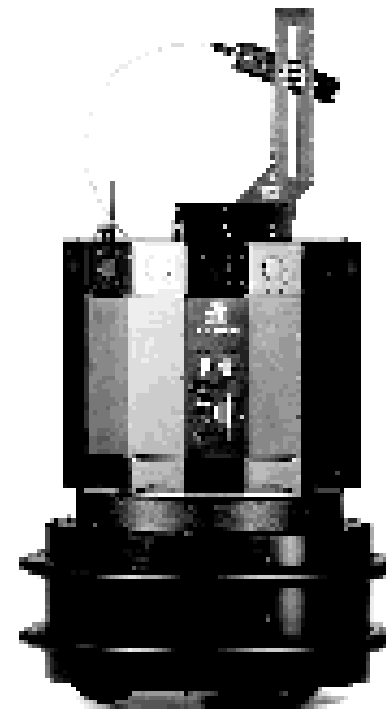
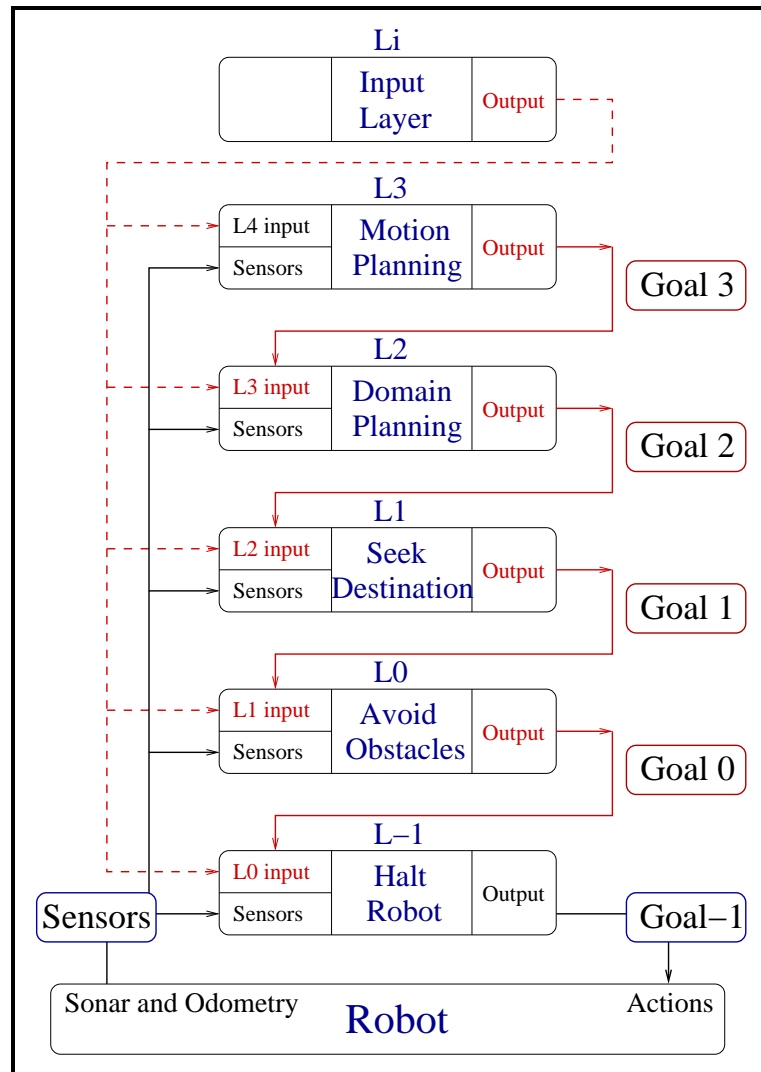
- LSA has a nonmonotonic semantics for which it is sound and complete.

Completeness: If

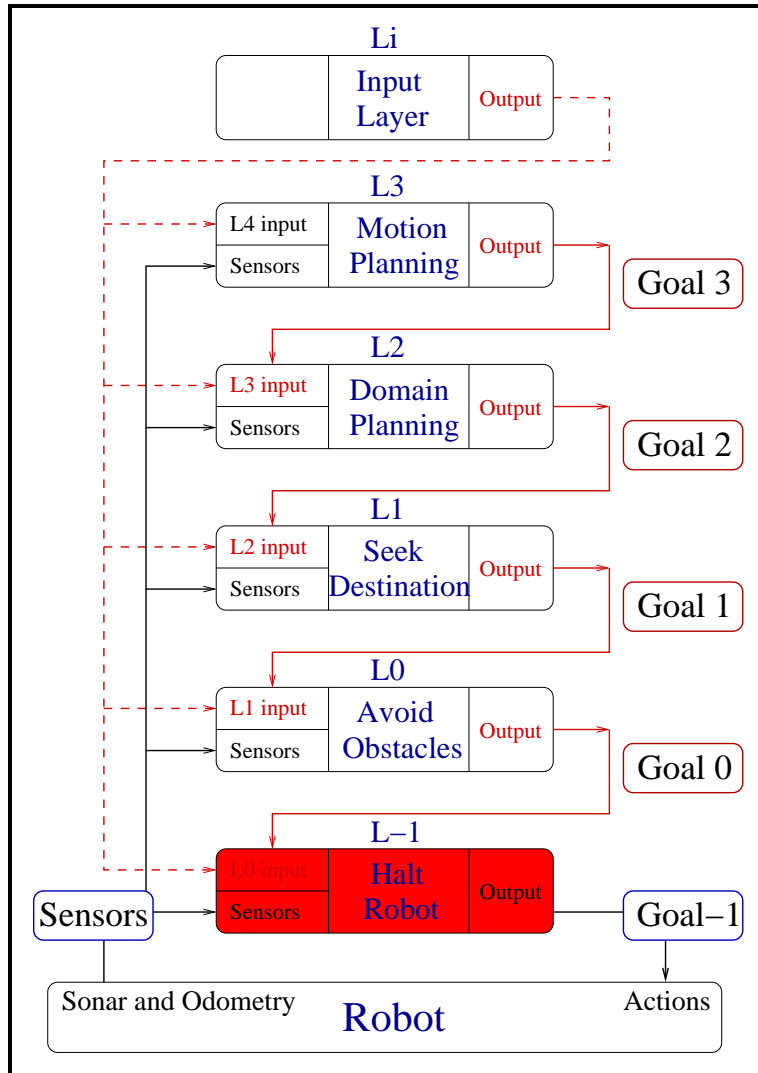
$Circ[Lay_0 \cup Circ[Lay_1 \cup \dots; P_1; Z_1]; P_0; Z_0] \models Goal_0(\vec{a})$
then LSA outputs $Goal_0(\vec{a})$ (each Lay_i is the theory for layer i).

Soundness: Only if LSA has output after all layers have concluded their proofs in order.

LSA for a Nomad200 Mobile Robot



LSA for a Nomad200 Mobile Robot



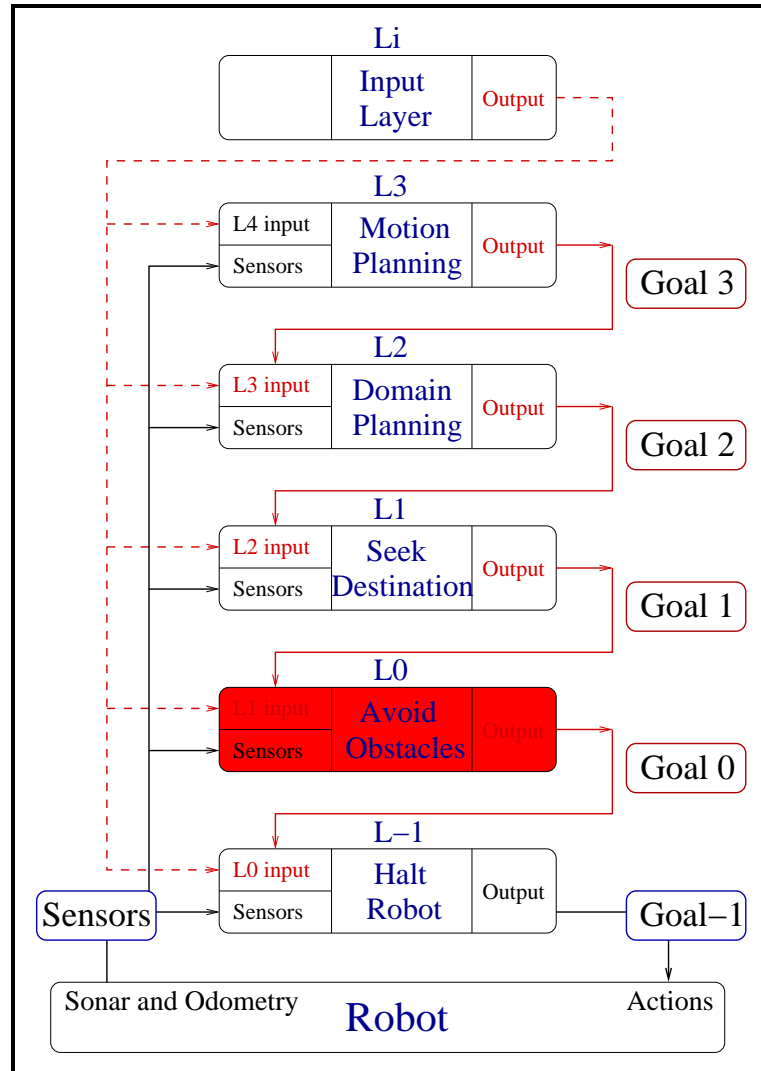
Layer -1: Halt or Go

Goals: $fwd(X); turn(Y)$

Default assumptions:

$Circ[Layer_{-1}; HaltRobot; \vec{Z}_{-1}]$.

LSA for a Nomad200 Mobile Robot



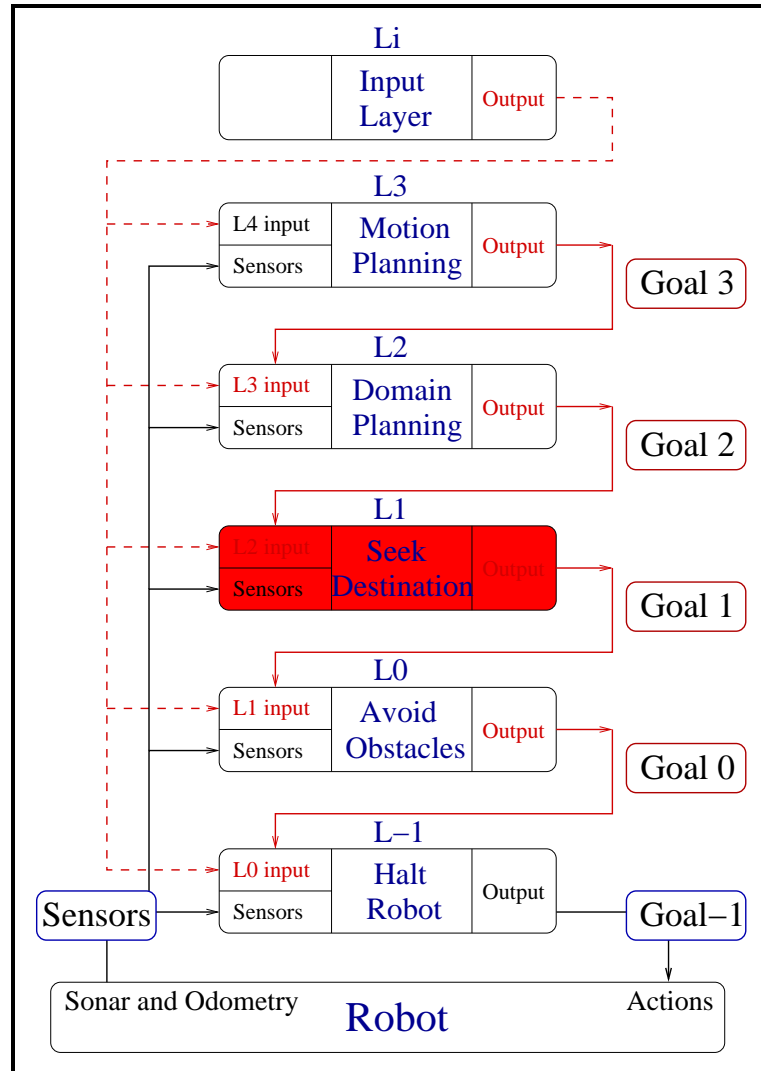
Layer 0: Avoid Obstacles

Goals: $fwd(X); turn(Y)$

Default assumptions:

$Circ[Layer_0; Object; \vec{Z}_0]$.

LSA for a Nomad200 Mobile Robot



Layer 1: Go To Destination

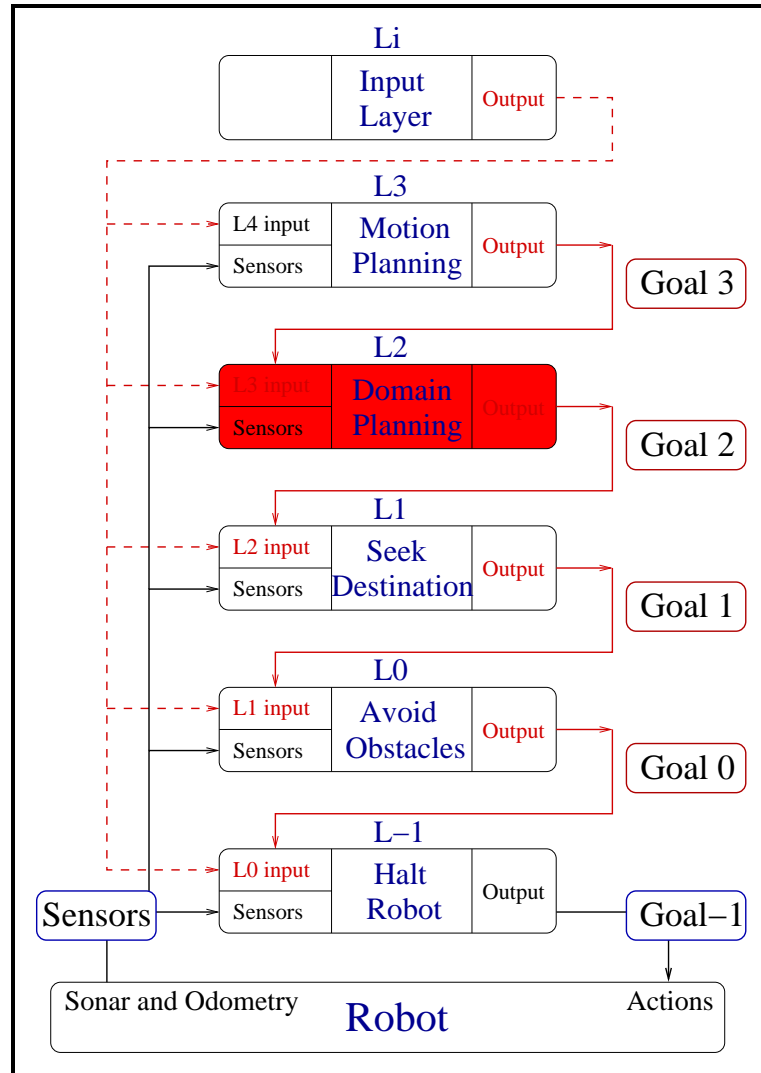
Goals:

*pushing_object(Obj),
distance(Obj, Dist),
direction(Obj, Dir)*

Default assumptions:

Circ[Layer₁; MoveCmd; \vec{Z}_1].

LSA for a Nomad200 Mobile Robot



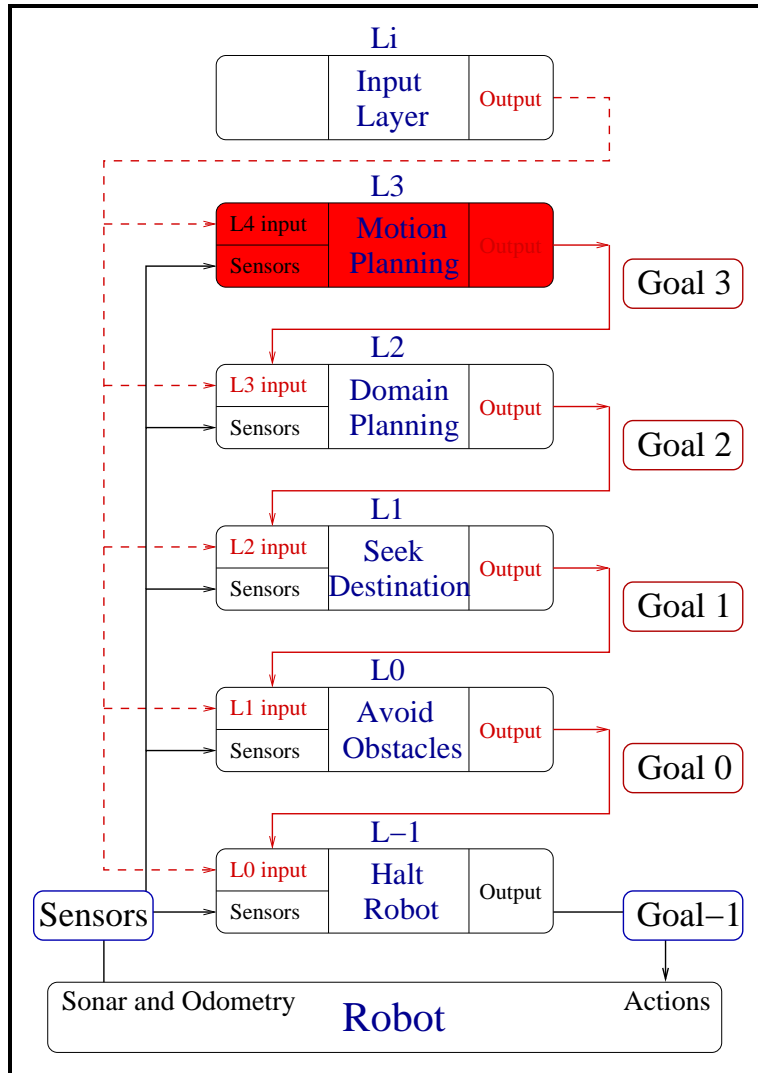
Layer 2: Move Locally

Goals: $moveCmd(X, Y);$
 $orderElev(floor(F));$
 $callElev$

Default assumptions:

$Circ[Layer_2; ShouldMove; \vec{Z}_2].$

LSA for a Nomad200 Mobile Robot



Layer 3: Move Between Landmarks

Goals: $moveTo(L)$

Defaults assumptions:

$$Circ[Layer_3; \neg RobotLost; L(Layer_3)].$$

Experimental Results

- Scenarios:
 - Simulation: Single-floor, elevator handling and multi-floor planning
 - Nomad200: Motion planning and obstacle avoidance in the robotics lab and first-floor corridors at Stanford.
- Results:
 - The robot runs smoothly and without hesitation (The lowest layer works at 10Hz).
 - Takes about 30 seconds between landmarks across the lab (10–20ft apart) with close objects.

Benefits of Logic-Based Subsumption

- + Reactive overall behavior.
 - + Simple modules, ~~no model of their environment.~~
 - + Simultaneously services multiple, conflicting goals.
 - + Supports behavioral decomposition.
-] Traditional
subsumption
systems
- + Declarative:
 - + Easy(er) to add information (offline and online).
 - + Each layer is potentially reusable for other tasks.
 - + High-level reasoning (knowledge, intentions, etc).
 - + Layers work in synergy.

Sound Reasoning versus Reactive Control

- LSA uses a limited form of MP.
- LSA includes default assumptions in layers.
- LSA includes a goal for each layer.
- LSA acts before all partitions have sent their results: requires stability analysis.

Related Work

- **Context systems:** Cyc (Lenat & Guha 1990), (Giunchiglia & Ghidini 1998).
- **Clique trees:** (Becker & Geiger 1996), (Shoikhet & Geiger 1997).
- **SAT:** (Dechter & Rish 2001), (Park & VanGelder 1996).
- **Parallel thm proving:** (Nelson & Oppen 1979), (Bonacina 1994), (Denzinger & Fuchs 1999).
- **Logic-utilizing robot systems:** (Kaelbling 1986), (Kaelbling & Rosenshein 1991), (Shanahan 1996), (Levesque et al., 1997).

What have we shown?

- Object-oriented design of logical theories.
- Message-Passing scales up reasoning for large KBs.
- Knowledge Bases can be partitioned automatically.
- Theoretical advance of graph algorithms.
- A logic-based architecture can control real-time tasks.

Future Directions

- Applications:
 - commonsense KBs, planning, formal verification.
- Extensions:
 - other logics
 - combined reasoning with other KR systems
- Reactive commonsense reasoning

<http://www-formal.stanford.edu/eyal>

Acknowledgements

Advisor: John McCarthy

Co-Authors: Pedrito Maynard-Reid II and Sheila
McIlraith

Reading Committee: Nils Nilsson, Mike Genesereth,
Leora Morgenster, Sheila McIlraith

Quail: Urszula, Lise, Pedrito, Patrick, Karl, Avi, Sunil

Many, many faculty and students ...