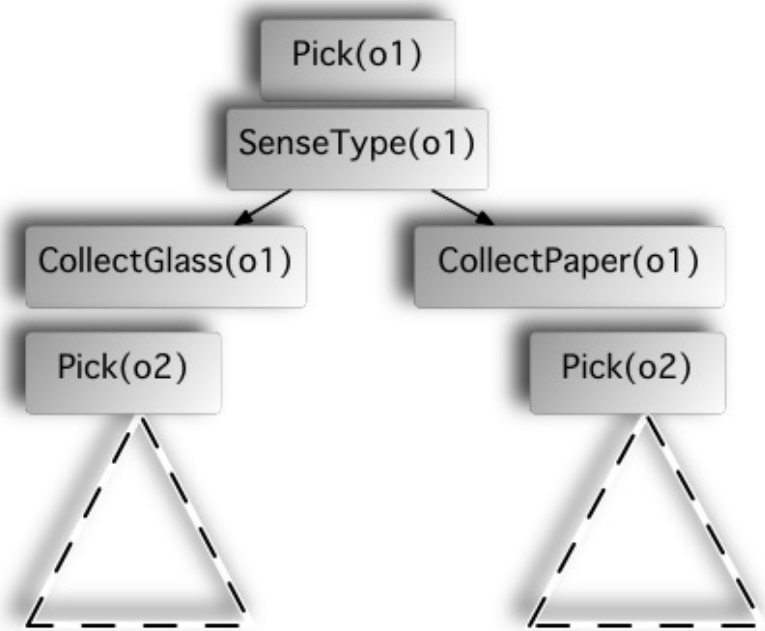
Computing Applicability Conditions for Plans with Loops

Siddharth Srivastava Neil Immerman
Shlomo Zilberstein

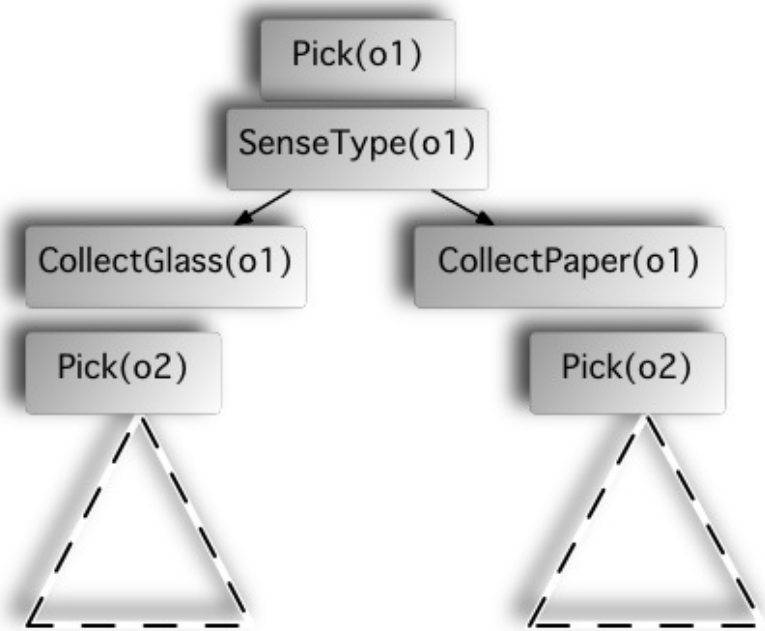
Department of Computer Science,
University of Massachusetts Amherst

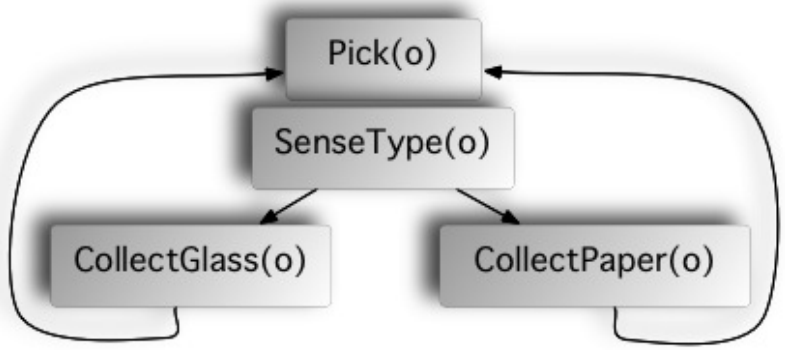
Twentieth International Conference on Automated Planning and Scheduling
15 May, 2010

Plans with Loops



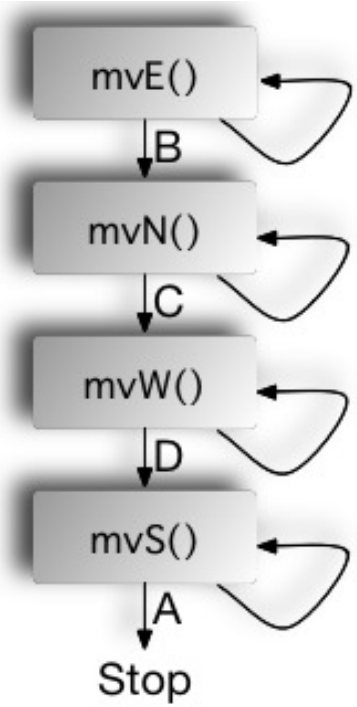
Plans with Loops



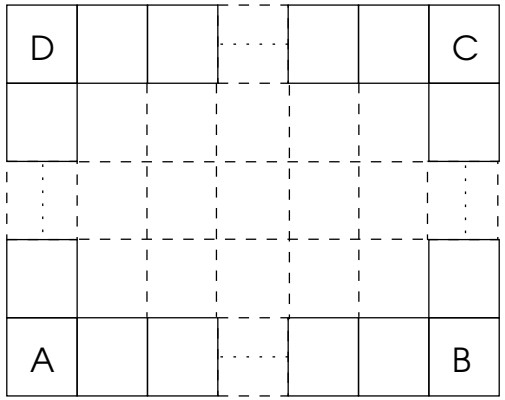


Loops $\Longrightarrow$ smaller, more general plans; allow unknown quantities

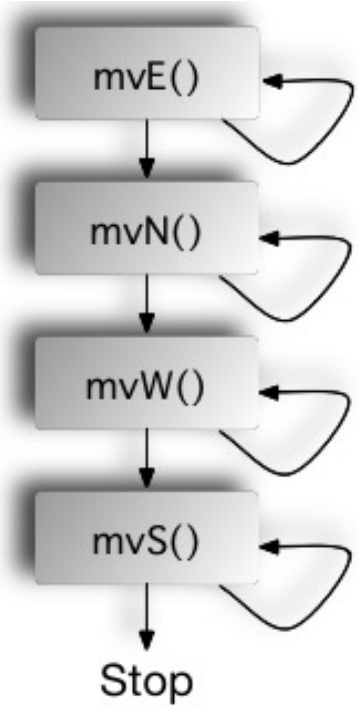
Plans with Loops: Power

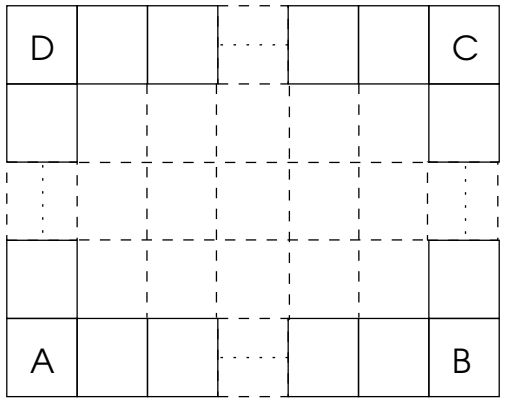


[Bonet et al., 2009]



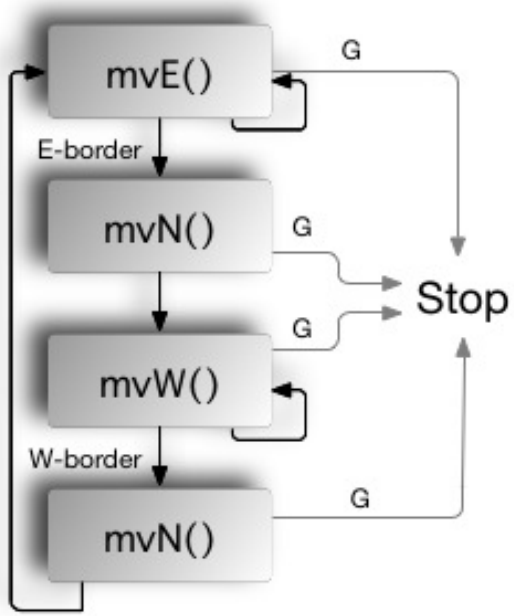
Plans with Loops: Power

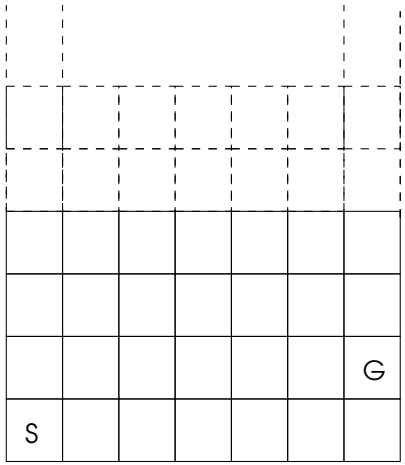




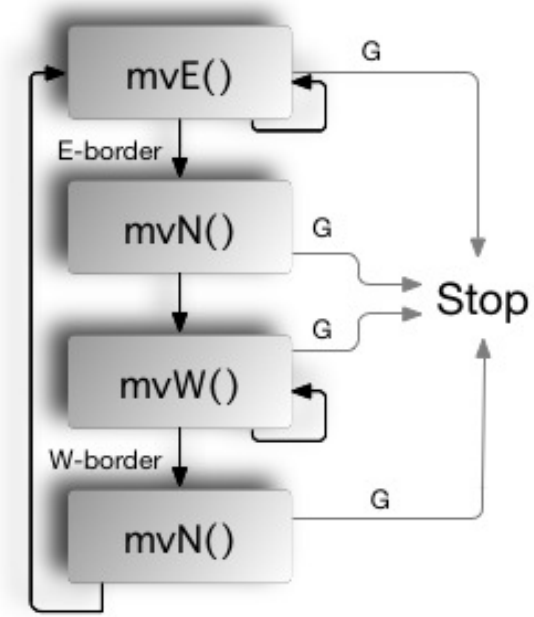
Correct numbers of iterations can reach any point in the interior

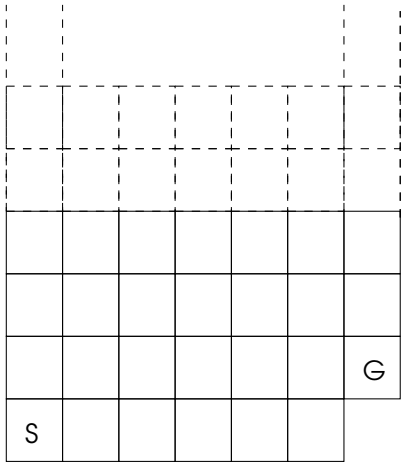
Plans with Loops: Risk





Plans with Loops: Risk





Never reaches G!

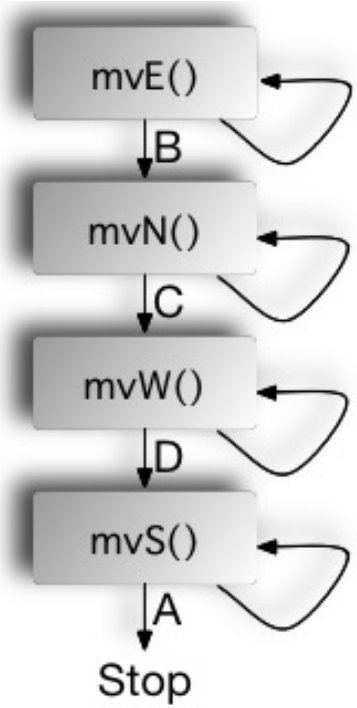
A plan with loops may traverse almost the entire state space – without solving the problem

Loop Effects and Preconditions

Why analyze loops of actions?

- During construction: utility, safety of potential loops
- After construction: will plan Π solve problem instance x?
- Longstanding issues of plan reuse

  Triangle tables, macro operators: linear sequences [Fikes et al., 1972]



Existing Approaches

[Levesque, 2005, Winner and Veloso, 2007, Bonet et al., 2009]

Plan Generation

- Observe working plans
- Extract repetitive patterns and make loops
- Loop utility justified largely by prior experience

Plan Execution

- Apply on any problem instance
- No preconditions; instantiation may traverse the entire state space! (EX: [Levesque, 2005])

Fundamental problems behind these limitations: termination, reachability in plans with loops.

Overview

- Abacus Programs
  - Linear Abacus Programs
  - Abacus Programs with Simple Loops
  - Abacus Programs with Complex Loops
  - Theoretical Results
- Translating Plans to Abacus Programs
- Empirical Results

Abacus Programs

Definition

$\mathcal{R}$ finite set of registers

$\mathcal{S}$ finite set of states

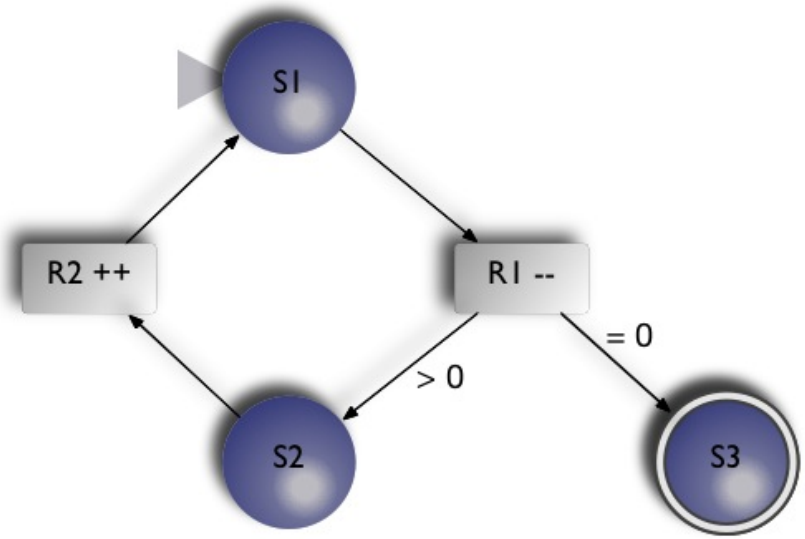
s_0 initial state

s_h halt state

ℓ $\mathcal{S} \rightarrow A$

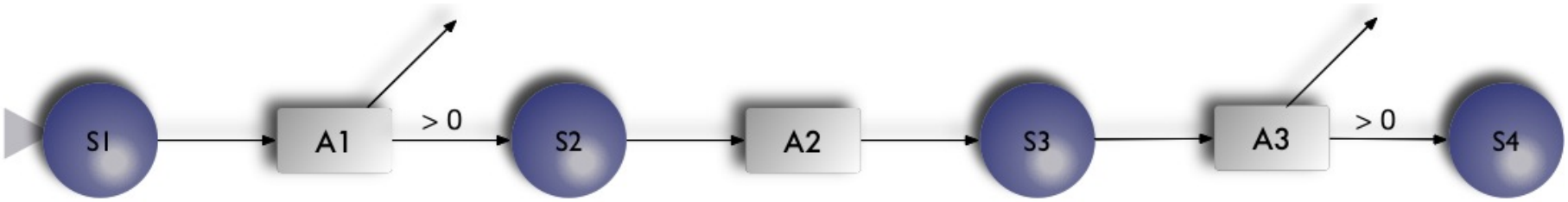
Actions (A):

- $Inc(r, s)$: $r++$; goto s
- $Dec(r, s_1, s_2)$: if $r = 0$ goto s_1
  else $r--$; goto s_2



Turing-complete $\Longrightarrow$ reachability is undecidable

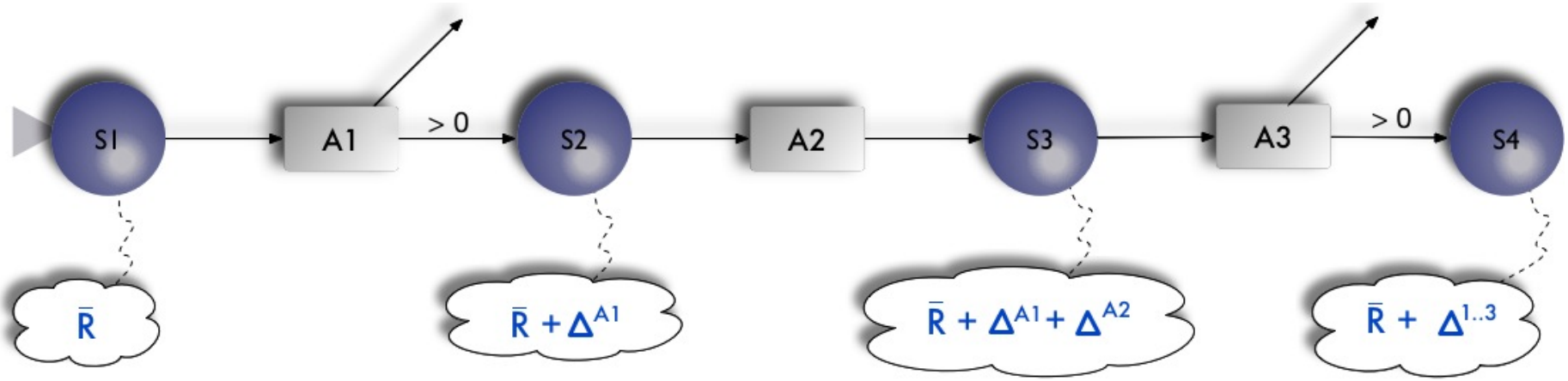
Linear Abacus Programs



Can Efficiently Find

- Cumulative action effects
- Branch Conditions

Linear Abacus Programs



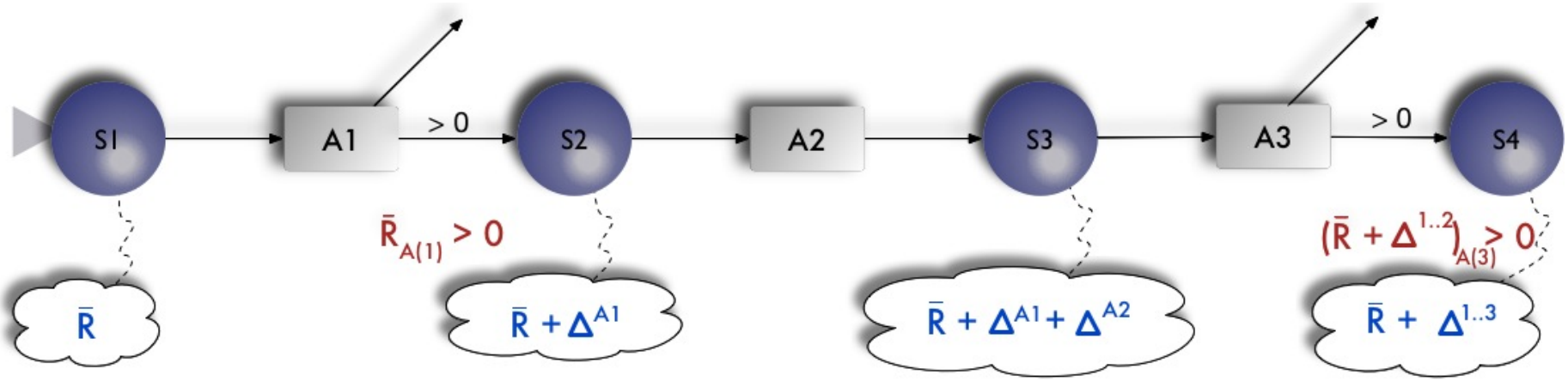
Can Efficiently Find

- Cumulative action effects
- Branch Conditions

Notation

$\bar{R}$	$\langle R_1, R_2, \ldots R_k \rangle$
Δ^{Ai}	Ai's change vector
$A(i)$	Index of Ai's register
$\bar{R}_i$	R_i
$\Delta^{1\ldots m}$	$\Delta^{A1} + \ldots + \Delta^{Am}$

Linear Abacus Programs



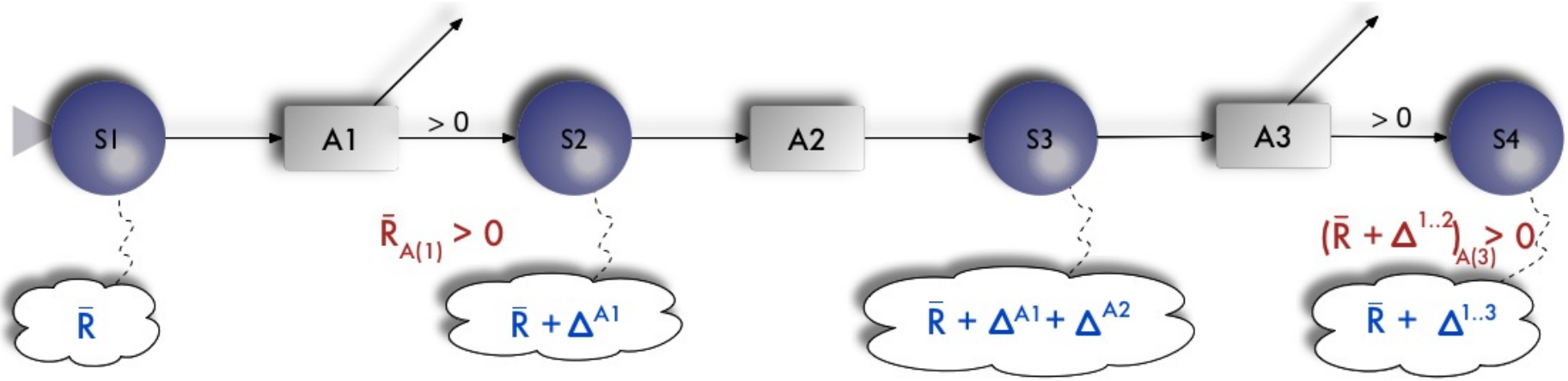
Can Efficiently Find

- Cumulative action effects
- Branch Conditions

Notation

$\bar{R}$	$\langle R_1, R_2, \ldots R_k \rangle$
Δ^{Ai}	Ai's change vector
$A(i)$	Index of Ai's register
$\bar{R}_i$	R_i
$\Delta^{1\ldots m}$	$\Delta^{A1} + \ldots + \Delta^{Am}$

Linear Abacus Programs: Preconditions



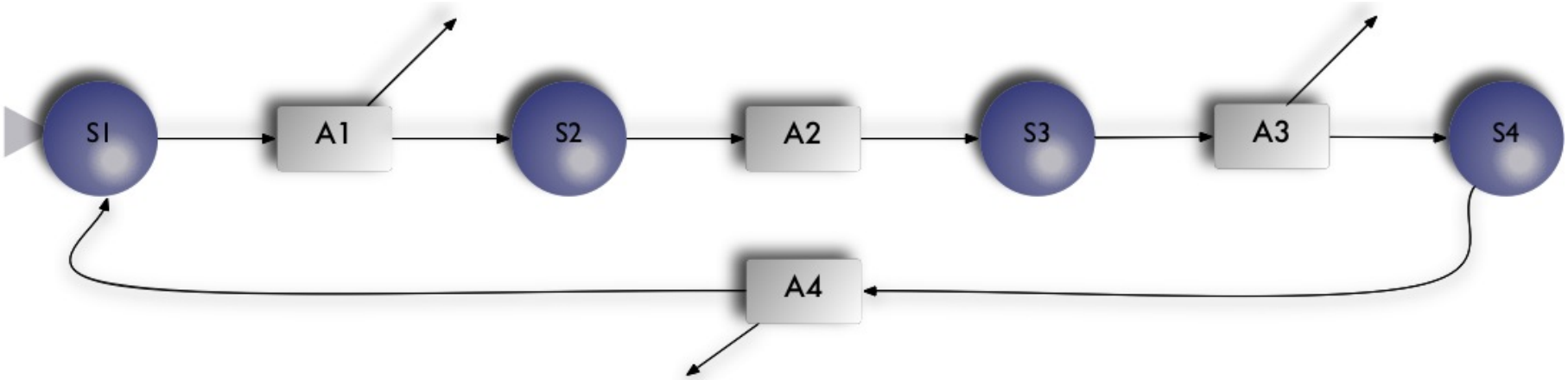
Necessary and Sufficient Conditions

Sequence of n actions; $\bar{F}$ = final register values

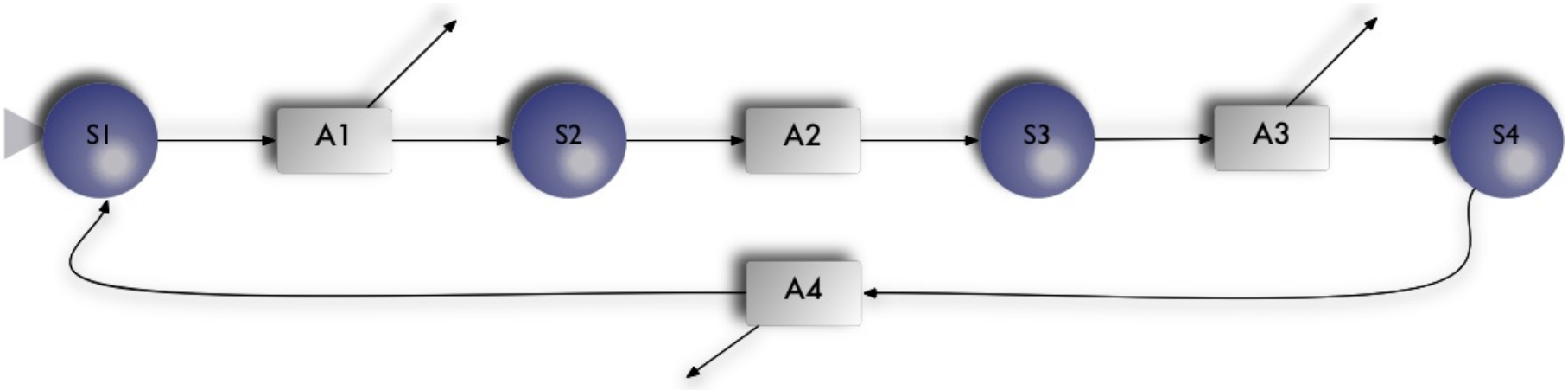
$$(\bar{R} + \Delta^{1\ldots i-1})_{A(i)} \quad \circ \quad 0, \qquad i = 1 \ldots n$$

$$\bar{F} \quad = \quad \bar{R} + \Delta^{1..n}$$

Abacus Programs with Simple Loops

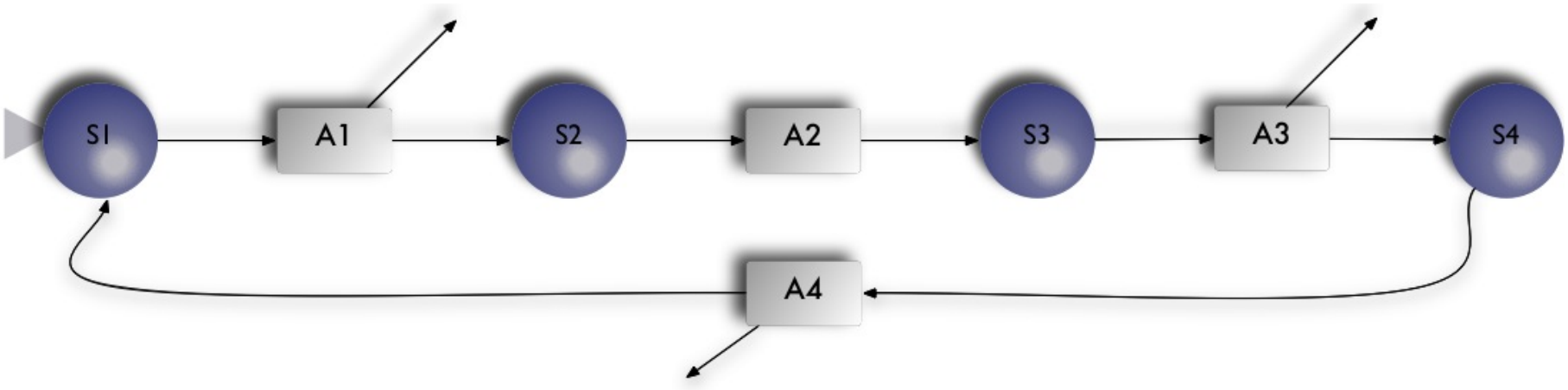


Abacus Programs with Simple Loops



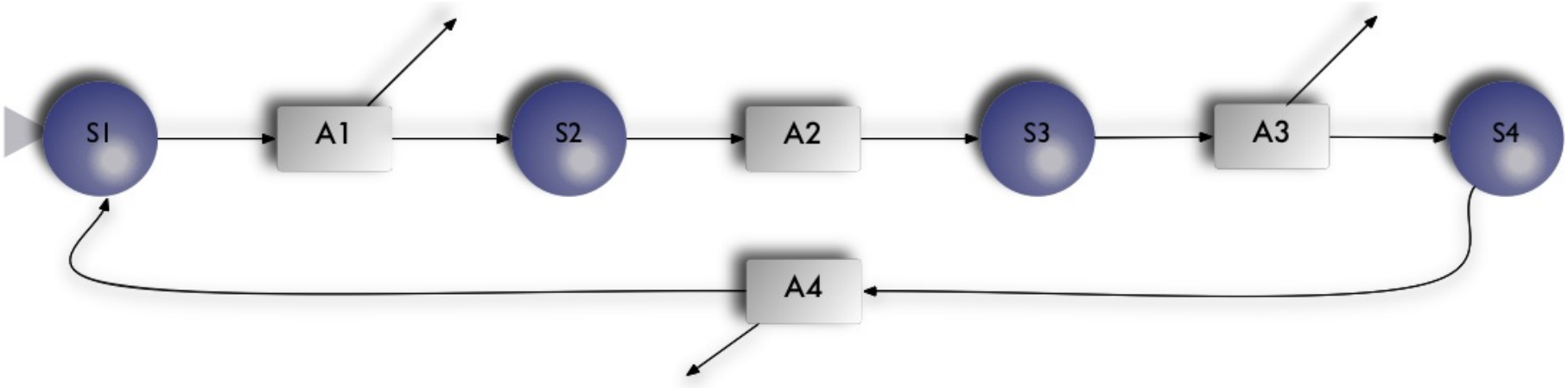
- $LoopIneq(\bar{R}^0) \equiv$
$$(\bar{R}^0 + \Delta^{1\ldots i-1})_{A(i)} \circ 0 \qquad i = 1 \ldots n$$

Abacus Programs with Simple Loops



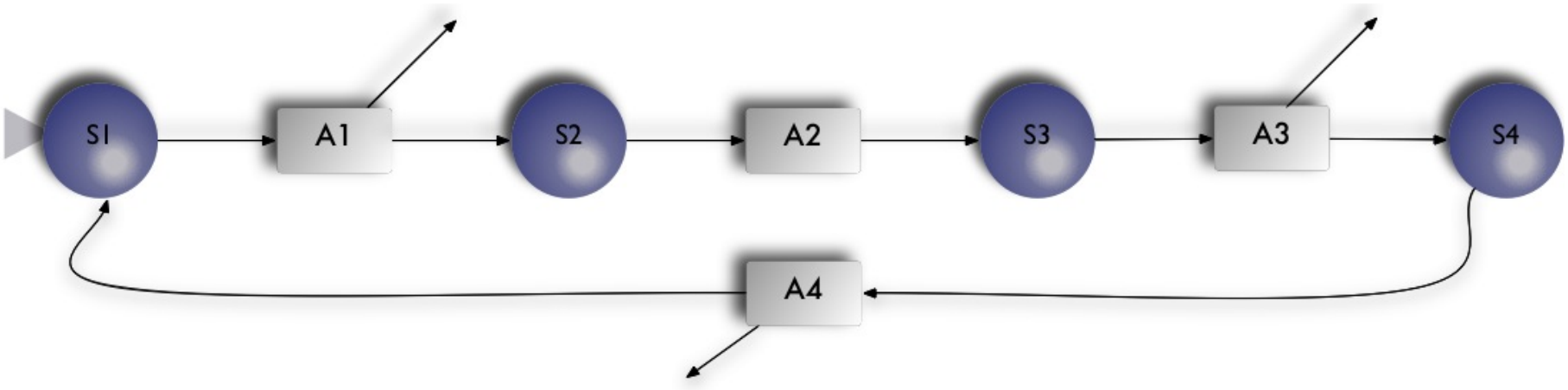
- $LoopIneq(\bar{R}^0) \equiv$
$$(\bar{R}^0 + \Delta^{1\ldots i-1})_{A(i)} \circ 0 \qquad i = 1 \ldots n$$
- $\bar{R}^x = \bar{R}^0 + x \cdot \Delta^{1..n}$

Abacus Programs with Simple Loops



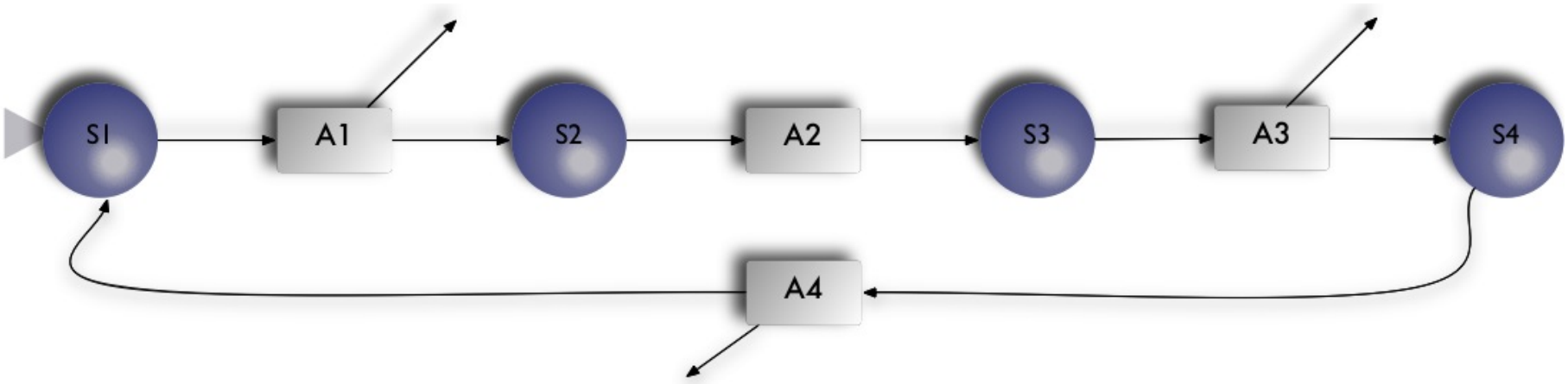
- $LoopIneq(\bar{R}^0) \equiv$
$$(\bar{R}^0 + \Delta^{1\ldots i-1})_{A(i)} \circ 0 \qquad i = 1 \ldots n$$
- $\bar{R}^x = \bar{R}^0 + x \cdot \Delta^{1..n}$
- For l complete iterations:
$$LoopIneq(\bar{R}^0) \wedge LoopIneq(\bar{R}^1) \wedge \cdots \wedge LoopIneq(\bar{R}^{l-1})$$
$$\equiv LoopIneq(\bar{R}^0) \wedge LoopIneq(\bar{R}^{l-1})$$

Abacus Programs with Simple Loops



- $LoopIneq(\bar{R}^0) \equiv$
$$(\bar{R}^0 + \Delta^{1\ldots i-1})_{A(i)} \circ 0 \qquad i = 1 \ldots n$$
- $\bar{R}^x = \bar{R}^0 + x \cdot \Delta^{1..n}$
- For l complete iterations:
$$LoopIneq(\bar{R}^0) \wedge LoopIneq(\bar{R}^1) \wedge \cdots \wedge LoopIneq(\bar{R}^{l-1})$$
$$\equiv LoopIneq(\bar{R}^0) \wedge LoopIneq(\bar{R}^{l-1})$$
- $\bar{F} = \bar{R}^l = \bar{R}^0 + l \cdot \Delta^{1\ldots n}$

Abacus Programs with Simple Loops: Preconditions



Necessary and Sufficient Conditions

For l complete iterations of a loop with n actions

$$LoopIneq(\bar{R}^0) \wedge LoopIneq(\bar{R}^{l-1})$$
$$\bar{F} = \bar{R}^l = \bar{R}^0 + l \cdot \Delta^{1\ldots n}$$

Recall: Approach

- Translate plans with loops into abacus programs (preserve structure)
- Design algorithms for finding preconditions of classes of abacus programs

Need to represent “sensing” actions.

Non-deterministic Actions

NSet Action

$NSet(r, s_1, s_2)$		set r to 0; goto $s_1 \in S$
	or	set r to 1; goto $s_2 \in S$

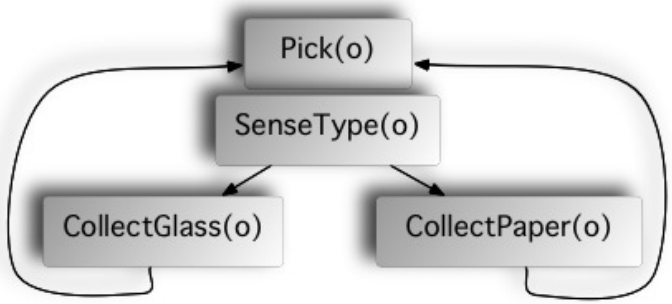
Non-deterministic Actions

NSet Action

$NSet(r, s_1, s_2)$ set r to 0; goto $s_1 \in S$

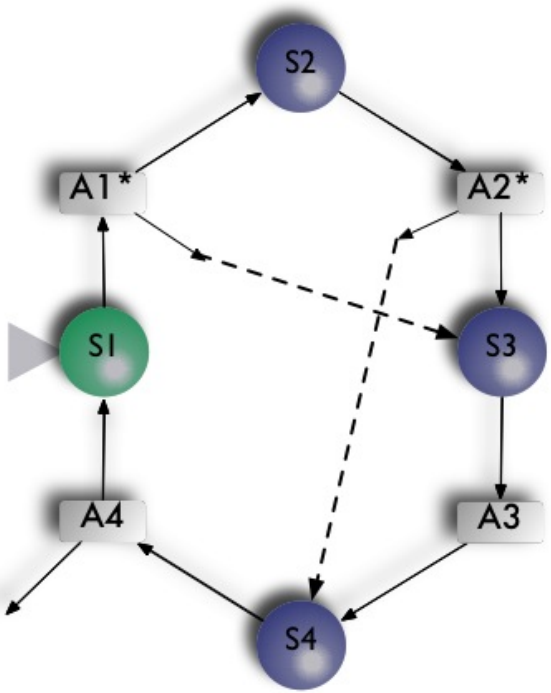
or set r to 1; goto $s_2 \in S$

- Abacus programs don't need *NSet* for computational power (Turing-complete).
- *NSet* allows in-place translation of plans with sensing actions.

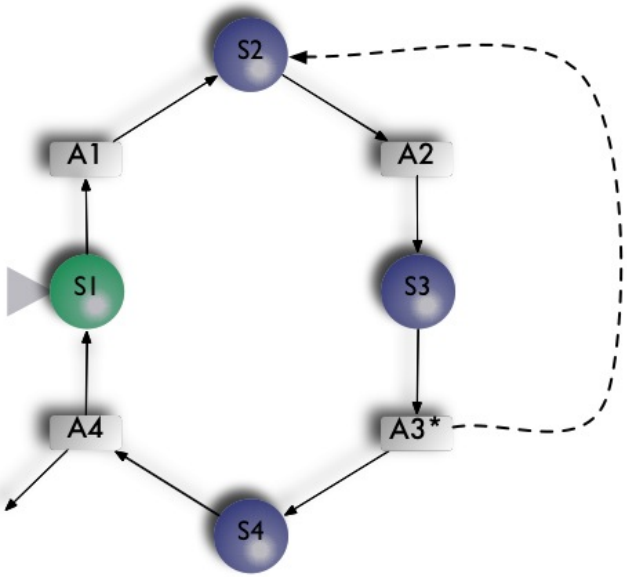


Find preconditions in terms of effect counts

Simple Loops with Shortcuts

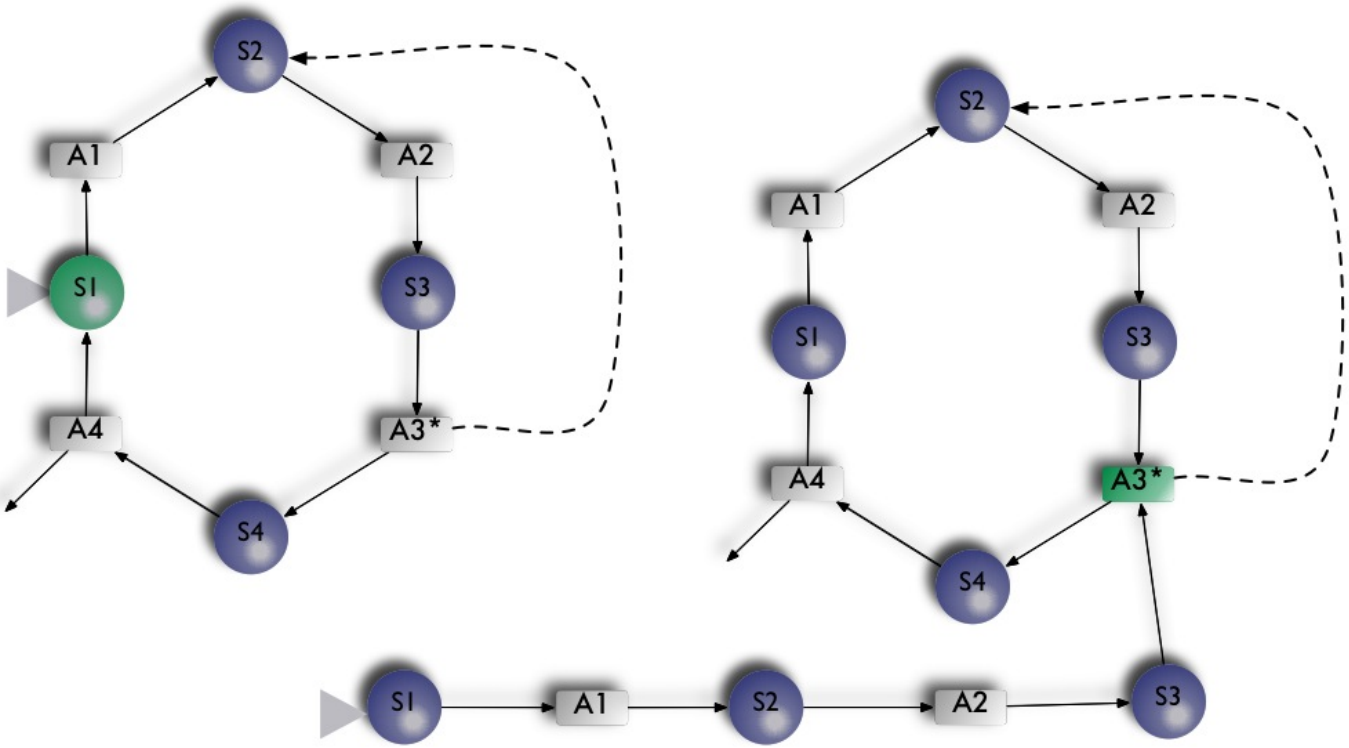


Simple Loops with Shortcuts



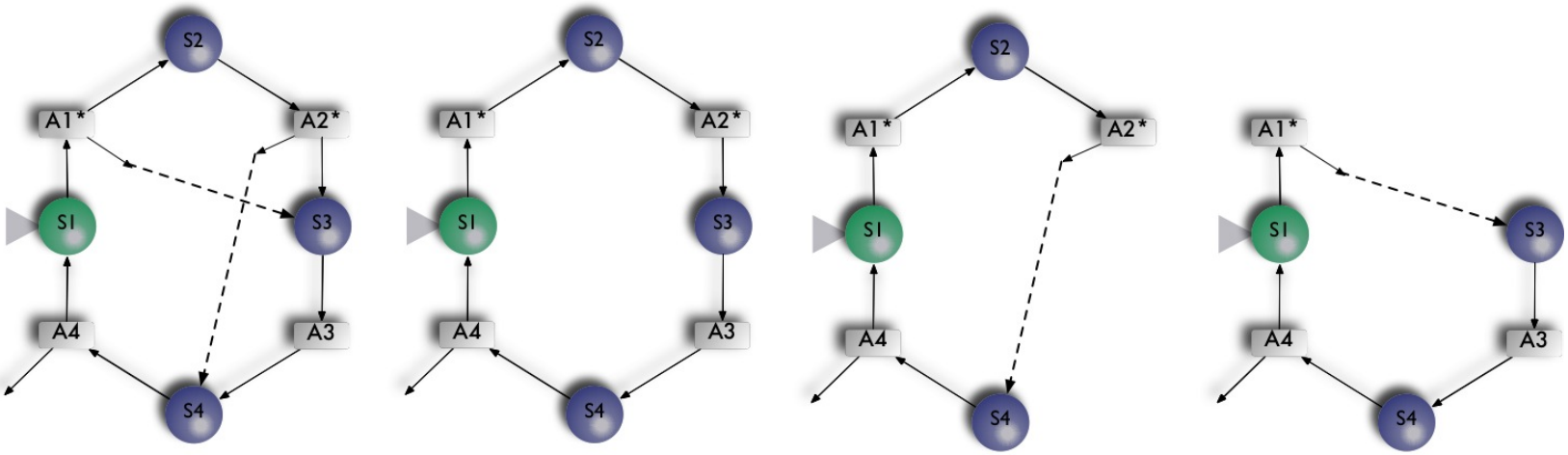
More commonly considered “nested” loops

Simple Loops with Shortcuts

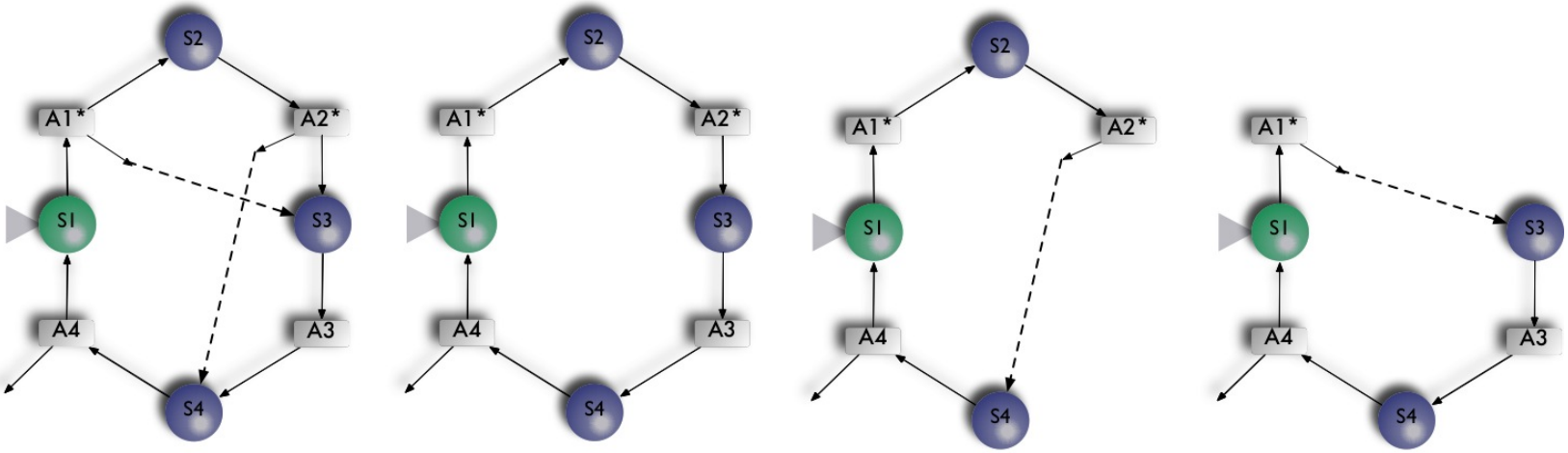


More commonly considered “nested” loops
....can be translated by changing the start node.

Simple Loops with Shortcuts



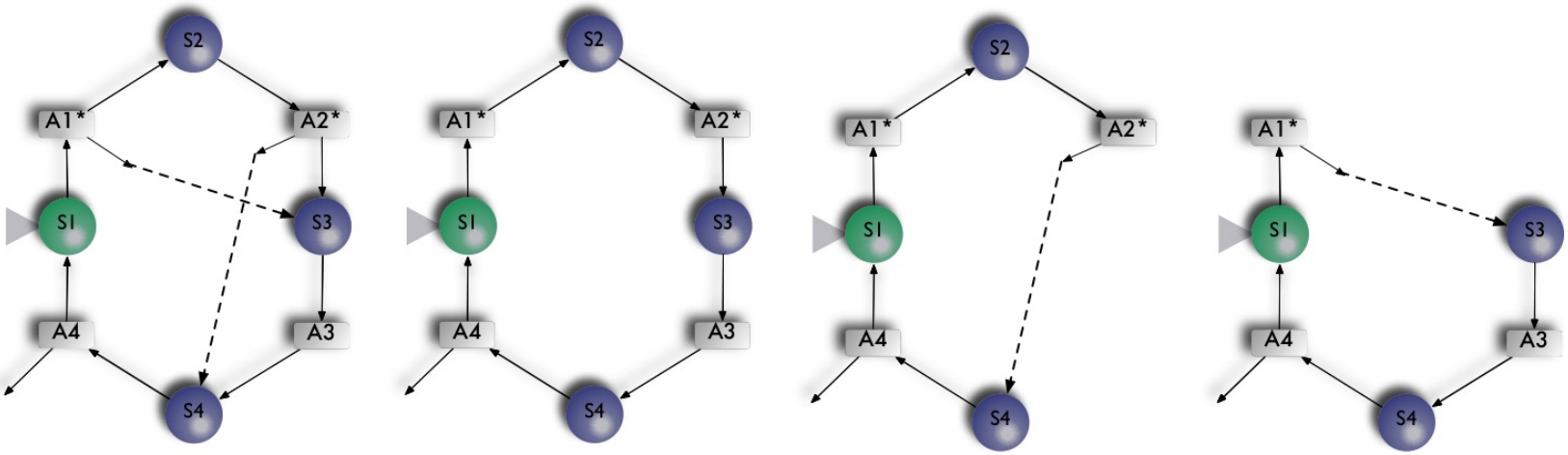
Monotonicity



Monotone Shortcuts

The *net* change on a register due to every simple loop created by shortcuts must be in the same direction (positive/negative).

Simple Loops with Shortcuts: Computing Preconditions

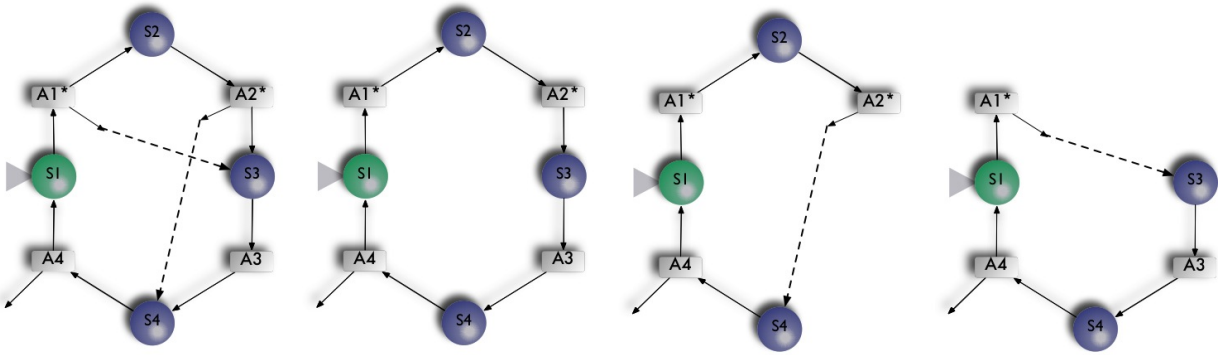


Suppose: m simple loops; $k_1, \ldots, k_m$ iterations

- $\bar{F} = \bar{R}^0 + \sum_{i=0}^{m} k_i \Delta^{loop_i}$
- k_i iterations of loop i: no register can fall below zero.

Need: least intermediate value of every register ≥ 0.

Constraining Minimum Register Values



Constraints on Min Values

m simple loops; $k_1, \ldots, k_m$ iterations

$$R_j \in \mathcal{R}^+ : \quad R_j^0 + \delta_{\widehat{\jmath}} \geq 0$$

$$R_j \in \mathcal{R}^- : \quad R_j^0 + \sum_{i=0}^{m} k_i \Delta^{\text{loop}_i} + \delta_{\widehat{\jmath}} - \Delta^{\text{loop}_{\widehat{\jmath}}} \geq 0$$

Notation

$\mathcal{R}^+$	registers with net positive change
$\mathcal{R}^-$	registers with net negative change
$\widehat{\jmath}$	index of loop with greatest *partial* negative change on R_j.

Order Dependence

Constraints on Minimum Register Values

$$R_j \in \mathcal{R}^+ : \quad R_j^0 + \delta_{\widehat{\jmath}} \geq 0$$

$$R_j \in \mathcal{R}^- : \quad R_j^0 + \sum_{i=0}^{m} k_i \Delta^{\text{loop}_i} + \delta_{\widehat{\jmath}} - \Delta^{\text{loop}_{\widehat{\jmath}}} \geq 0$$

Example

loop_1 increase R_1 by 3.

loop_2 decrease R_1 by 2, then increase it by 5.

Efficiently expressing general order dependent constraints: open problem!

Order Dependence

Constraints on Minimum Register Values

$$R_j \in \mathcal{R}^+ : \quad R_j^0 + \delta_{\widehat{\jmath}} \geq 0$$

$$R_j \in \mathcal{R}^- : \quad R_j^0 + \sum_{i=0}^{m} k_i \Delta^{\text{loop}_i} + \delta_{\widehat{\jmath}} - \Delta^{\text{loop}_{\widehat{\jmath}}} \geq 0$$

Example

loop_1 increase R_1 by 3.

loop_2 decrease R_1 by 2, then increase it by 5.

Constraint $R_1^0 - 2 \geq 0$ (sufficient condition)

Efficiently expressing general order dependent constraints: open problem!

Order Dependence

Constraints on Minimum Register Values

$$R_j \in \mathcal{R}^+ : \quad R_j^0 + \delta_{\widehat{\jmath}} \geq 0$$

$$R_j \in \mathcal{R}^- : \quad R_j^0 + \sum_{i=0}^{m} k_i \Delta^{\text{loop}_i} + \delta_{\widehat{\jmath}} - \Delta^{\text{loop}_{\widehat{\jmath}}} \geq 0$$

Example

loop_1 increase R_1 by 3.

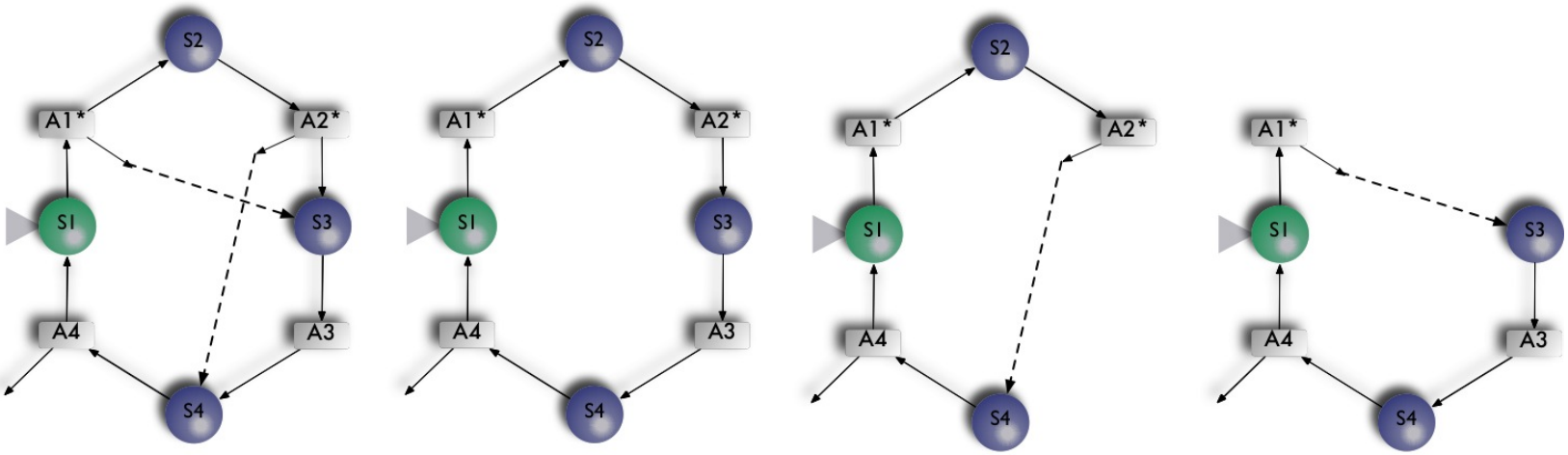
loop_2 decrease R_1 by 2, then increase it by 5.

Constraint $R_1^0 - 2 \geq 0$ (sufficient condition)

Accurate Conditions Order dependent!

Efficiently expressing general order dependent constraints: open problem!

Equality Constraints, Order Dependence



- Non-spurious equality constraint: inherently order dependent

Equality constraints, $\delta_{\hat{\jmath}} \neq \Delta^{loop_{\hat{\jmath}}}$ induce order dependence (details in paper).

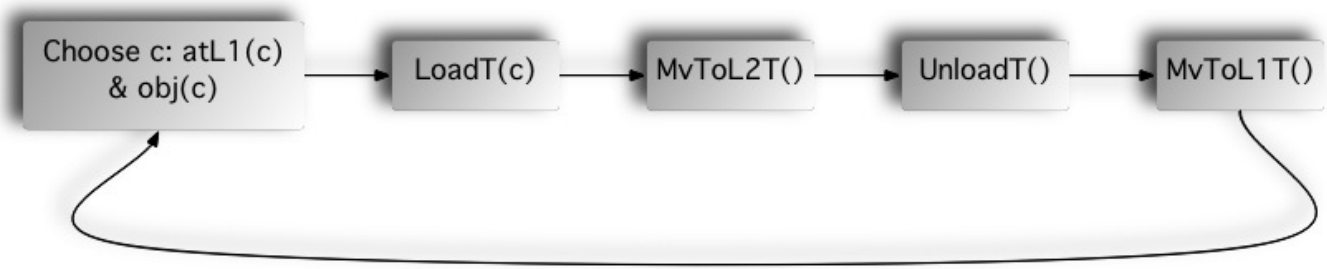
Main Results

Let S be a node in an abacus program Π, all of whose strongly connected components are simple loops or simple loops with monotone shortcuts; $\bar{F}$: a vector of register values.

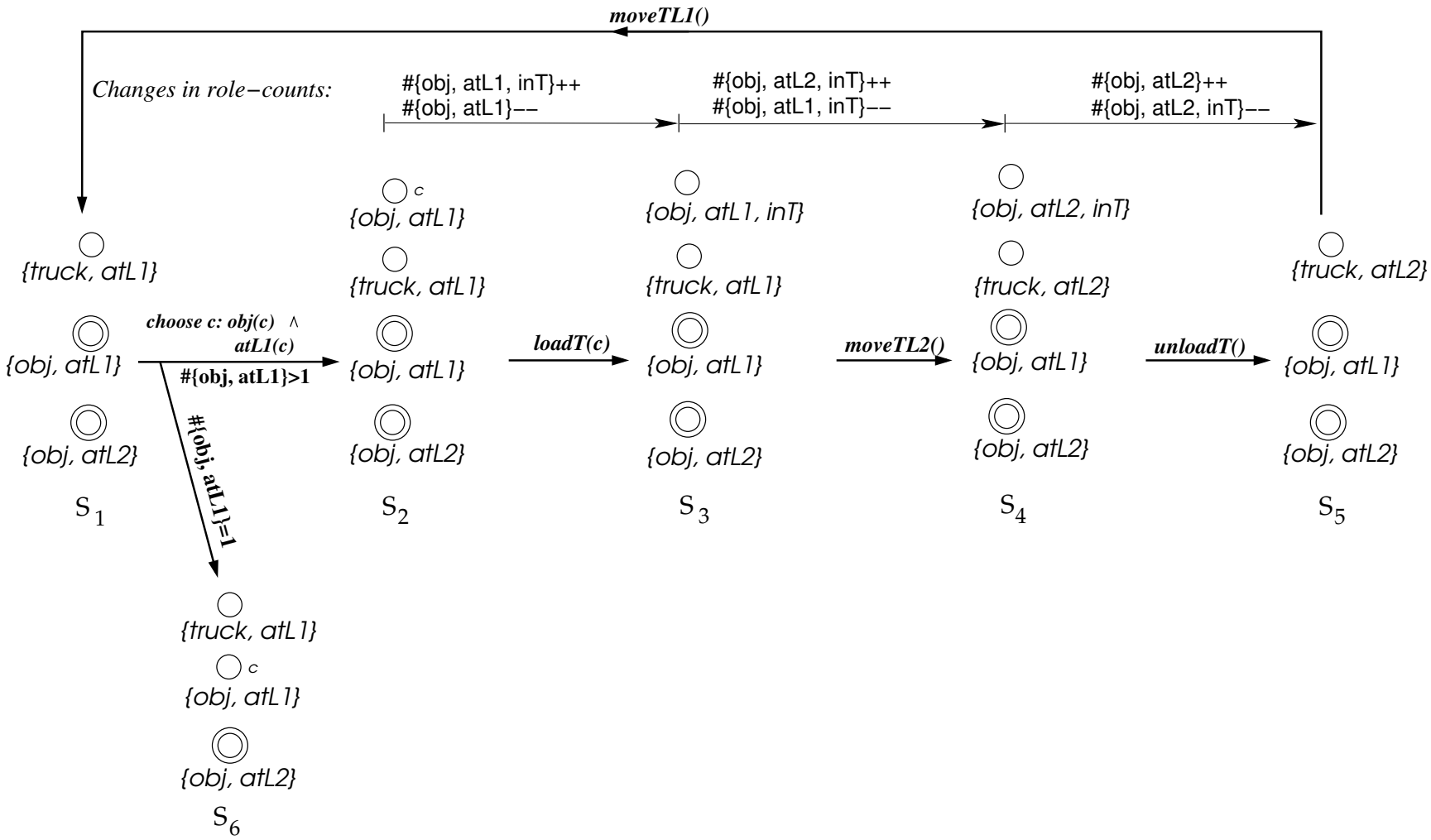
Theorem

We can compute a disjunction of linear constraints on the initial register values ($\bar{R}^0$) for reaching S with $\bar{F}$. The computed conditions are necessary and sufficient if every simple loop with shortcuts is order independent, and sufficient otherwise.

Translating Plans To Abacus Programs

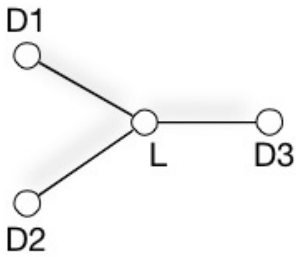


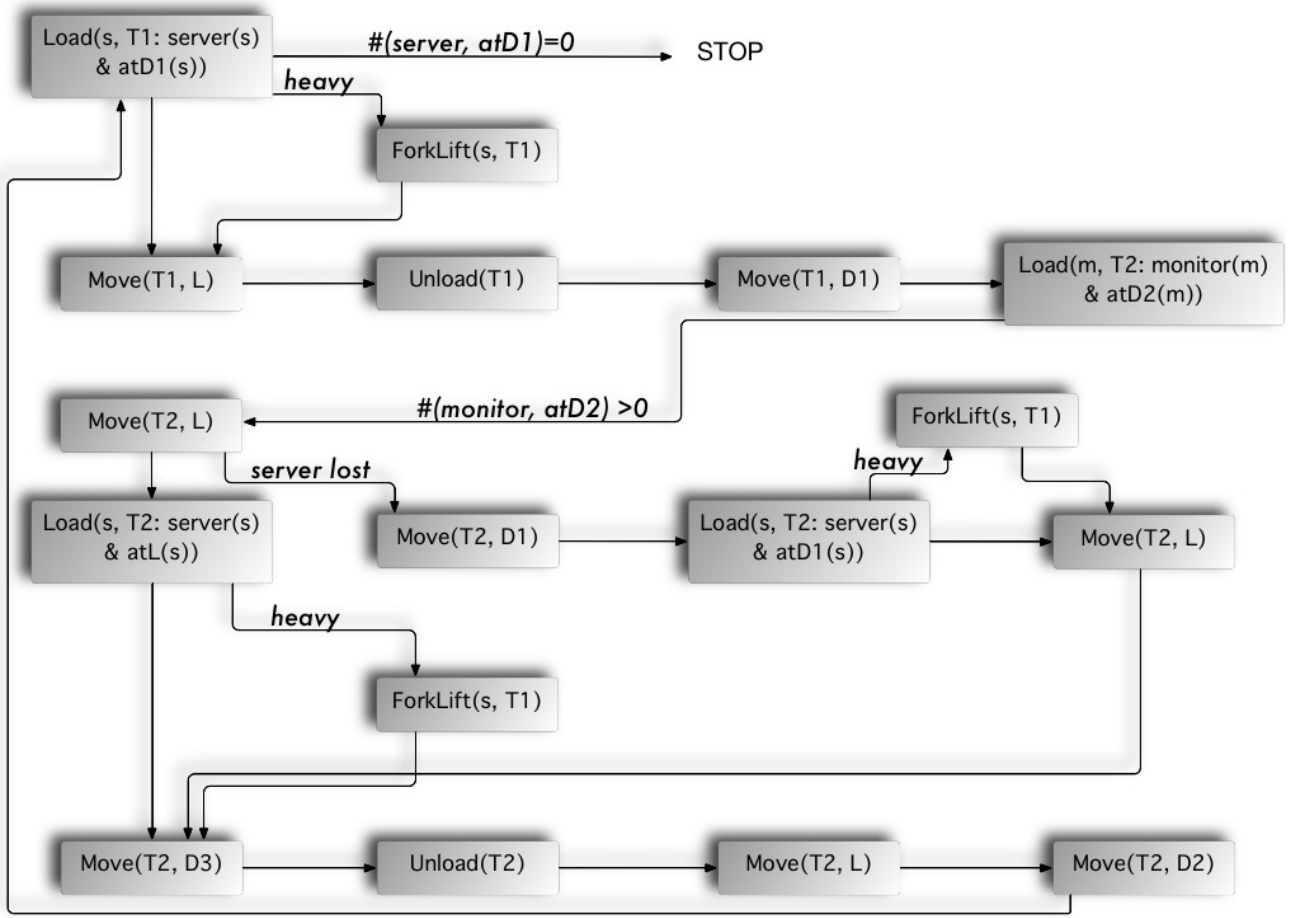
Translating Plans To Abacus Programs



Works for all unary, and a class of binary domains

Empirical Results





$$s_3^f = m_3^f = \sum_{i=0}^{7} k_i;$$
$$s_1^f = s_1^0 - \sum_{i=0}^{7} k_i - k_0 - k_5 - k_6 - k_7 = 0$$

Empirical Results

Time Taken to Compute Preconditions

Problem	Time (s)	Problem	Time(s)
Accumulator	0.01	Prize-A(7)	0.02
Corner-A	0.00	Recycling	0.02
Diagonal	0.01	Striped Tower	0.02
Hall-A	0.01	Transport	0.01
Prize-A(5)	0.01	Transport (conditional)	0.06

Conclusion

- An approach for computing summarized effects of loops of actions
- Use during construction and for precondition evaluation
- Future work:
  - Greater translation to abacus programs (counts of properties)
  - Further categorization of tractable classes
  - Expression of order dependent constraints
  - Use of symbolic precondition evaluation for non-numeric branches.

Complete Preconditions

Preconditions for a Simple Loop with Shortcuts

- $I = \sum_{i=0}^{m} k_i$; $\bar{F} = \bar{R}^0 + \sum_{i=0}^{m} k_i \Delta^{loop_i}$
- $R_j \in \mathcal{R}^- : \quad \bigvee_{x=0_j,\ldots,m_j} \{k_{i<x} = 0; k_x \neq 0;$
  $R_j^0 + \sum_{i \geq x} k_i \Delta^{\text{loop}_i} + \delta_x - \Delta^{\text{loop}_x} \geq 0\}$
- $R_j \in \mathcal{R}^+ : \quad \bigvee_{x=0_j,\ldots,m_j} \{k_{i<x} = 0; k_x \neq 0; R_j^0 + \delta_x \geq 0\}$

References I

Bonet, B., Palacios, H., and Geffner, H. (2009).
Automatic derivation of memoryless policies and finite-state controllers using classical planners.
In *Proc. of the 19th International Conf. on Artificial Intelligence Planning and Scheduling*.

Fikes, R., Hart, P., and Nilsson, N. (1972).
Learning and Executing Generalized Robot Plans.
Technical report, AI Center, SRI International.

Levesque, H. J. (2005).
Planning with loops.
In *Proc. of IJCAI*, pages 509–515.

References II

Winner, E. and Veloso, M. M. (2007).
LoopDISTILL: Learning domain-specific planners from example plans.
In *Workshop on AI Planning and Learning, ICAPS*.