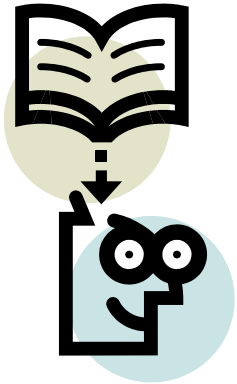
Ames
Research
Center

Translating PDDL into NDDL

Sara Bernardini
London Knowledge Lab
University of London

David E. Smith
NASA Ames Research Center

More than you ever wanted to know



PDDL 1.1

Drive (?trk-truck, ?loc1-location, ?loc2-location)

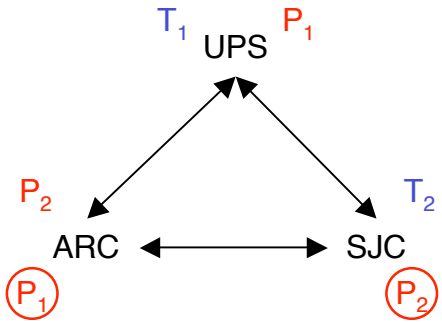
pre: at(?trk, ?loc1)
eff: ¬at(?trk, ?loc1)
at(?trk,?loc2)

Load (?pkg-package,?trk-truck)

pre: at(?trk, ?loc)
at(?pkg, ?loc)
eff: ¬at(?pkg, ?loc)
in(?pkg,?trk)

Unload (?pkg-package,?trk-truck)

pre: at(?trk, ?loc)
in(?pkg, ?trk)
eff: ¬in(?pkg, ?trk)
at(?pkg,?loc)



T_1, T_2 – truck
P_1, P_2 – package
ARC, UPS, SJC – location

Init: at(T_1, UPS)
at(T_2, SJC)
at(P_1, UPS)
at(P_2, ARC)

Goal: at(P_1, ARC)
at(P_2, SJC)

PDDL 2.1

Drive (?trk, ?loc1, ?loc2)

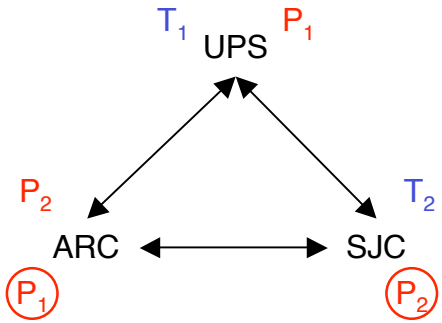
pre: at-start at(?trk, ?loc1)
eff: at-start ¬at(?trk, ?loc1)
et-end at(?trk,?loc2)

Load (?pkg,?trk)

pre: at-start &
over-all at(?trk, ?loc)
at-start at(?pkg, ?loc)
eff: at-start ¬at(?pkg, ?loc)
at-end in(?pkg,?trk)

Unload (?pkg,?trk)

pre: at-start &
over-all at(?trk, ?loc)
at-start in(?pkg, ?trk)
eff: at-start ¬in(?pkg, ?trk)
at-end at(?pkg,?loc)



T_1, T_2 – truck
P_1, P_2 – package
ARC, UPS, SJC – location

Init: at(T_1, UPS)
at(T_2, SJC)
at(P_1, UPS)
at(P_2, ARC)

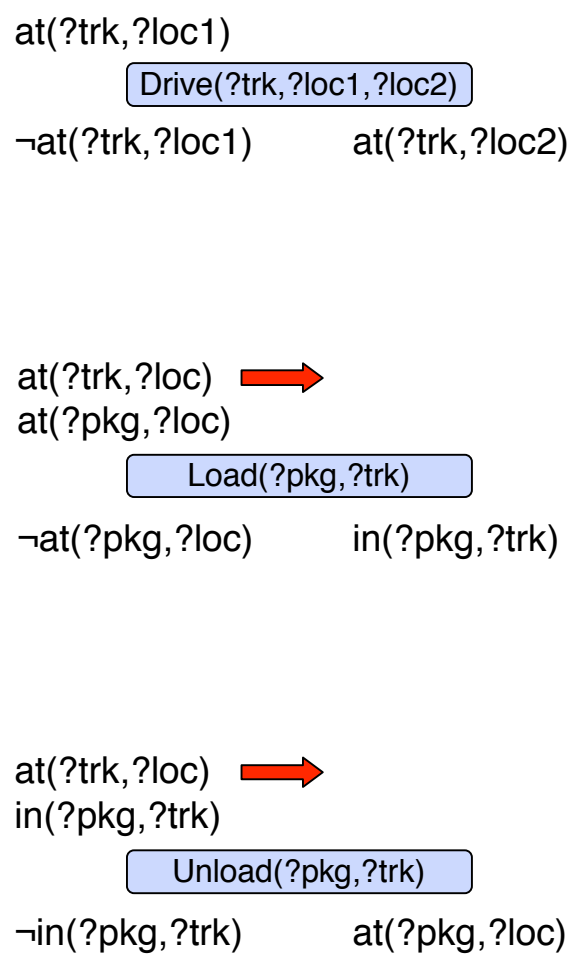
Goal: at(P_1, ARC)
at(P_2, SJC)

PDDL 2.1

```
Drive (?trk, ?loc1, ?loc2)
    pre:    at-start  at(?trk, ?loc1)
    eff:    at-start  ¬at(?trk, ?loc1)
            et-end    at(?trk,?loc2)

Load (?pkg,?trk)
    pre:    at-start
            over-all  at(?trk, ?loc)
            at-start  at(?pkg, ?loc)
    eff:    at-start  ¬at(?pkg, ?loc)
            at-end    in(?pkg,?trk)

Unload (?pkg,?trk)
    pre:    at-start
            over-all  at(?trk, ?loc)
            at-start  in(?pkg, ?trk)
    eff:    at-start  ¬in(?pkg, ?trk)
            at-end    at(?pkg,?loc)
```



Ames Research Center

NDDL

Timelines

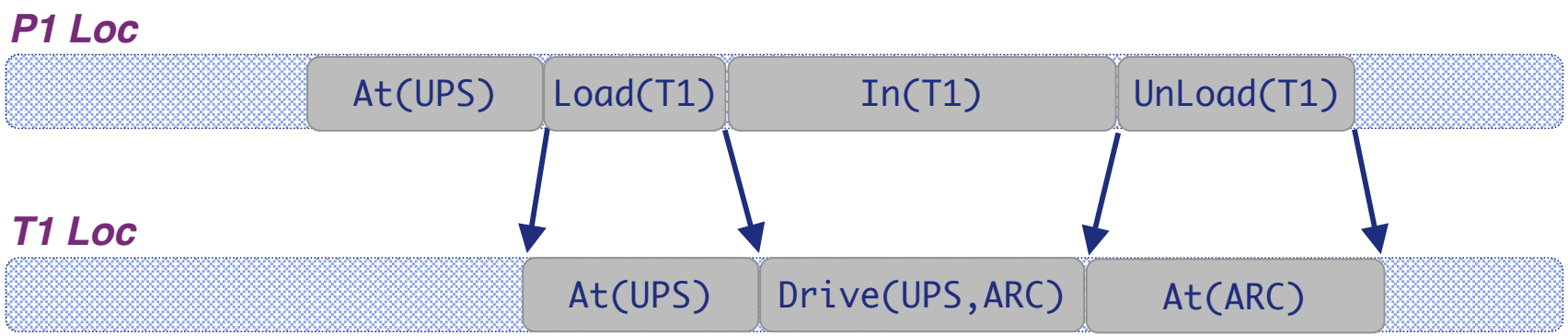
PkgLoc(p):
- Predicates: At(loc l) [0,∞]
  In(truck t) [0,∞]
  Load(truck t) [10,10]
  Unload(truck t) [10,10]

TrkLoc(t):
- Predicates: At(loc l) [0,∞]
  Drive(loc l1, loc l2) [20,20]

Compatabilities

PkgLoc(p)

In(truck t)	met-by Load(truck t) meets Unload(truck t)
Load(truck t)	met-by At(loc l) meets In(truck t) contained-by TrkLoc(truck t).At(loc l)
UnLoad(truck t)	met-by In(truck t) meets At(loc l) contained-by TrkLoc(truck t).At(loc l)



Naive Translation

Drive (?trk, ?loc1, ?loc2)

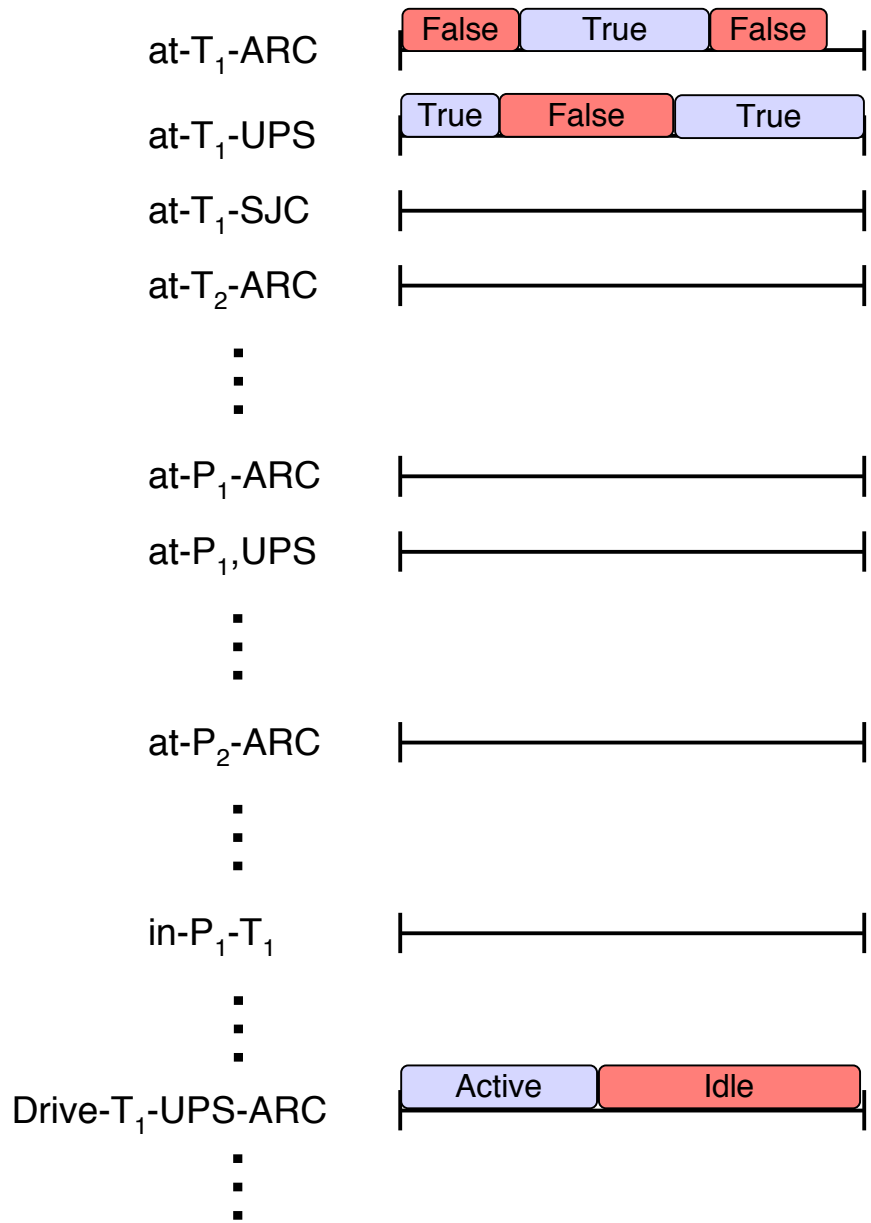
pre:	at-start	at(?trk, ?loc1)
eff:	at-start	¬at(?trk, ?loc1)
	et-end	at(?trk,?loc2)

Load (?pkg,?trk)

pre:	at-start	
	over-all	at(?trk, ?loc)
	at-start	at(?pkg, ?loc)
eff:	at-start	¬at(?pkg, ?loc)
	at-end	in(?pkg,?trk)

Unload (?pkg,?trk)

pre:	at-start	
	over-all	at(?trk, ?loc)
	at-start	in(?pkg, ?trk)
eff:	at-start	¬in(?pkg, ?trk)
	at-end	at(?pkg,?loc)



What We Want

Drive (?trk, ?loc1, ?loc2)

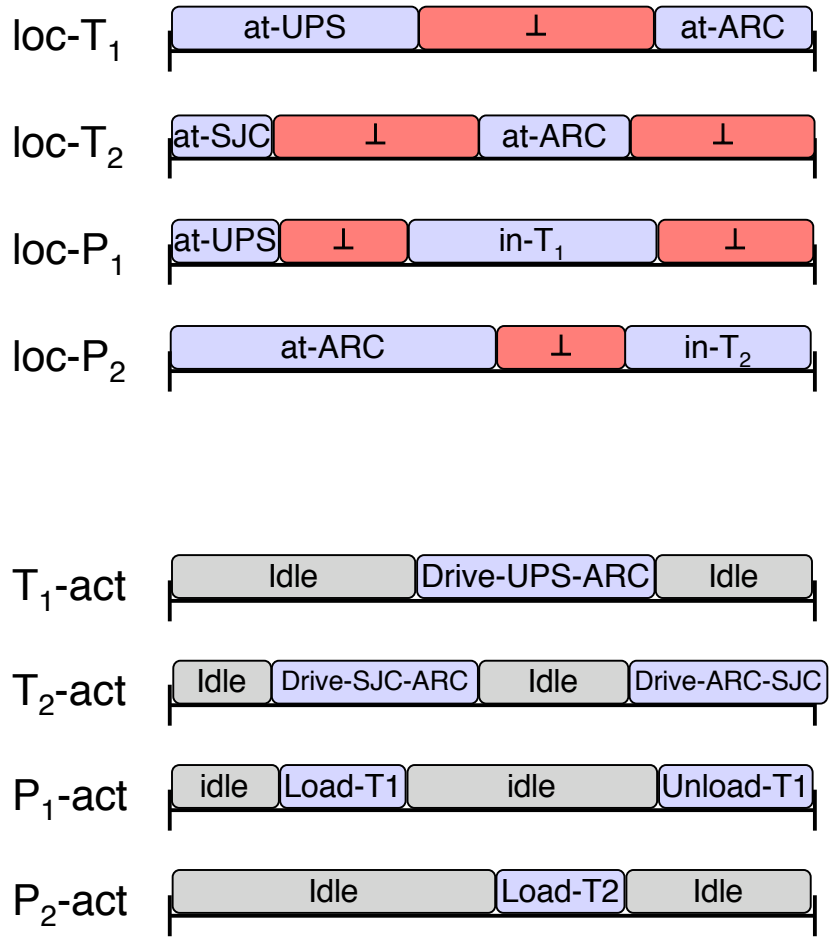
pre: at-start at(?trk, ?loc1)
eff: at-start ¬at(?trk, ?loc1)
et-end at(?trk,?loc2)

Load (?pkg,?trk)

pre: at-start
over-all at(?trk, ?loc)
at-start at(?pkg, ?loc)
eff: at-start ¬at(?pkg, ?loc)
at-end in(?pkg,?trk)

Unload (?pkg,?trk)

pre: at-start
over-all at(?trk, ?loc)
at-start in(?pkg, ?trk)
eff: at-start ¬in(?pkg, ?trk)
at-end at(?pkg,?loc)



History

- Helmert 2006:thesis, 2009:AIJ

  non-numerical & ***non-temporal*** PDDL
  to Multi-valued Planning Task (MPT).

  MPT: *- Multi-valued State Variables*
  - Operators: preconditions--effects

  Similar steps, but:

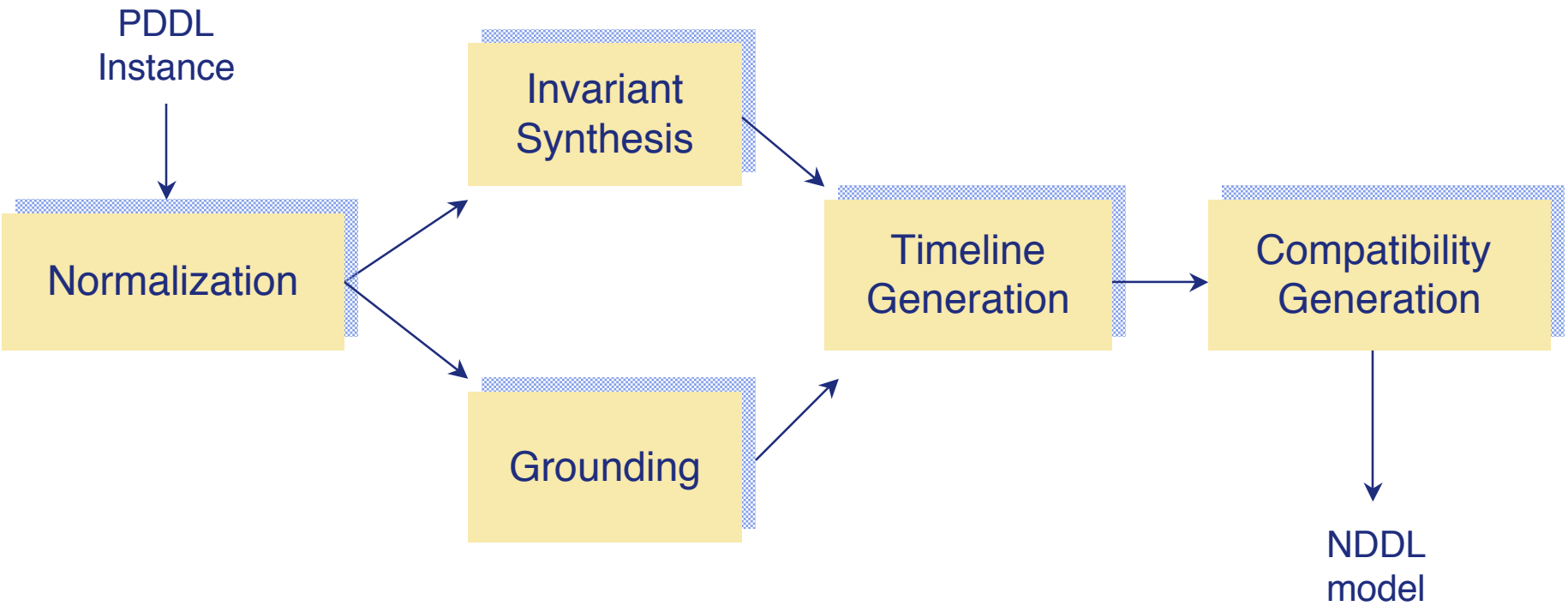
  - Temporal Extension of the MPT formalism
  - New Invariant Synthesis Algorithm
  - Identification of Mutex Operators
  - Complex compatibilities synthesis

- Gerevini and Schubert 2000: Discoplan

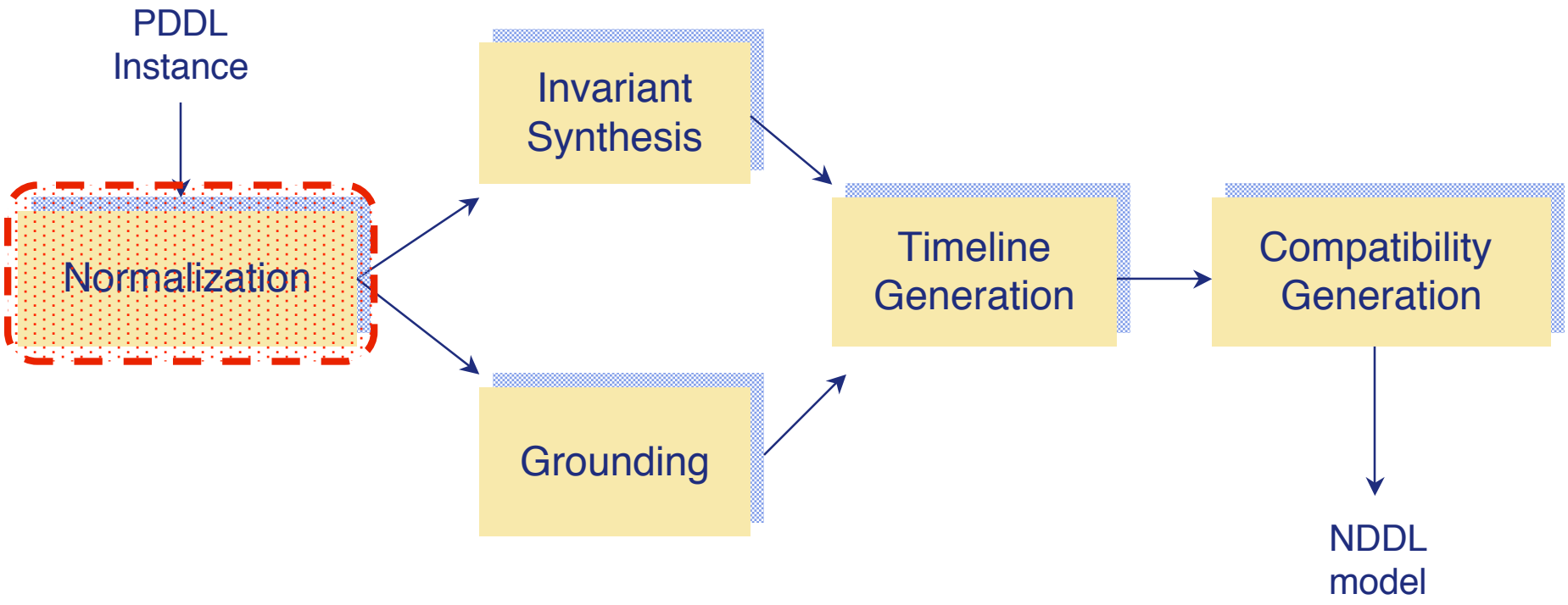
- Fox and Long 1998: TIM

Ames Research Center

Translation Steps



Translation Steps



Normalization

[Helmert-2009]

Normalization: 3 tasks

1. Compiles away ***types***
2. Simplify ***conditions***
   - -goal formula
   - -axiom bodies
   - -operator conditions
   - -conditions of conditional effects
3. Simplify ***effects***

Eliminating Types

[Helmert-2009]

Drive (?trk-truck, ?loc1-location, ?loc2-location)
pre: at(?trk, ?loc1)
eff: ¬at(?trk, ?loc1)
at(?trk,?loc2)

↓

Drive (?trk, ?loc1, ?loc2)
pre: truck(?trk)
location(?loc1), location(?loc2)
at(?trk, ?loc1)
eff: ¬at(?trk, ?loc1)
at(?trk,?loc2)

Simplifying Conditions

(goals, preconds, premises,ce-conds)

[Helmert-2009]

1. Implications

Actions:	A pre: $\Phi \Rightarrow \Psi$	→	A pre: $\neg\Phi \vee \Psi$
Axioms:	$(\Phi \Rightarrow \Psi) \Rightarrow e$	→	$(\neg\Phi \vee \Psi) \Rightarrow e$

2. Universally quantified conditions
3. Disjunctive goals
4. Disjunctive conditions
5. Existential quantifiers

Simplifying Conditions

(goals, preconds, premises,ce-conds)

[Helmert-2009]

1. Implications
2. Universally quantified conditions

$\forall x \Phi$ → $\neg \exists x \neg \Phi$ → $\neg \Psi$ + axiom: $\exists x \neg \Phi \Rightarrow \Psi$

3. Disjunctive goals
4. Disjunctive conditions
5. Existential quantifiers

Simplifying Conditions

(goals, preconds, premises,ce-conds)

[Helmert-2009]

1. Implications
2. Universally quantified conditions
3. Disjunctive goals

$$G_1 \vee G_2 \;\Rightarrow\; \Psi \;+\; \text{axiom: } G_1 \vee G_2 \Rightarrow \Psi$$

4. Disjunctive conditions
5. Existential quantifiers

Simplifying Conditions

(goals, preconds, premises,ce-conds)

[Helmert-2009]

1. Implications
2. Universally quantified conditions
3. Disjunctive goals
4. Disjunctive conditions

Actions:

A pre: $C_1 \vee C_2$ eff: Ψ	→	A_1 pre: C_1 eff: Ψ	+	A_2 pre: C_2 eff: Ψ

Axioms:

$C_1 \vee C_2 \Rightarrow \Psi$	→	$C_1 \Rightarrow \Psi$	+	$C_2 \Rightarrow \Psi$

5. Existential quantifiers

Simplifying Conditions

(goals, preconds, premises,ce-conds)

[Helmert-2009]

1. Implications
2. Universally quantified conditions
3. Disjunctive goals
4. Disjunctive conditions
5. Existential quantifiers

$$\exists x\ \Phi \Rightarrow \Psi \quad \Rightarrow \quad \forall x\ (\Phi \Rightarrow \Psi)$$

Simplifying Effects

[Helmert-2009]

1. Universal quantifiers

$\forall x\ (e_1 \wedge e_2)$ → $\forall x\ e_1 \wedge \forall x\ e_2$

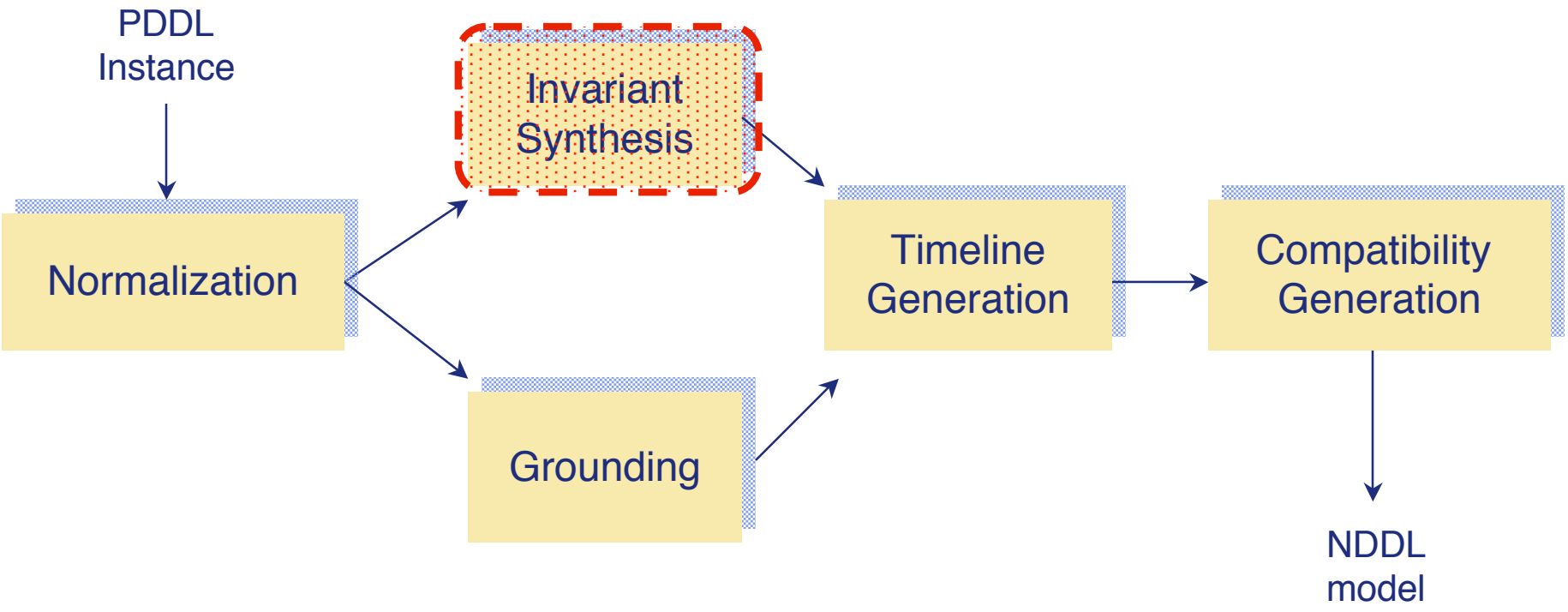
$\Phi \Rightarrow \forall x\ e$ → $\forall x\ (\Phi \Rightarrow e)$

2. Conditional effects

$\Phi \Rightarrow (e_1 \wedge e_2)$ → $\Phi \Rightarrow e_1 \ \wedge\ \Phi \Rightarrow e_2$

$\Phi \Rightarrow (\Psi \Rightarrow e)$ → $(\Phi \wedge \Psi) \Rightarrow e)$

Translation Steps



Invariant Synthesis

- ***Invariant***: property satisfied by all states reachable from initial state

  Interested in ***Mutual Exclusion Invariants***

$$at(T_1,ARC) \otimes at(T_1,UPS) \otimes at(T_1, SJC)$$

$$at(P_1,ARC) \otimes at(P_1,UPS) \otimes at(P_1,SJC) \otimes in(P_1,T_1) \otimes in(P_1,T_2)$$

- We synthesize ***timelines*** by grouping together mutually exclusive atoms identified by ***invariants***

$$loc\text{-}T_1 \in \{at\text{-}ARC,\ at\text{-}UPS,\ at\text{-}SJC\}$$

$$loc\text{-}P_1 \in \{at\text{-}ARC,\ at\text{-}UPS,\ at\text{-}SJC,\ in\text{-}T_1,\ in\text{-}T_2\}$$

- How do we find invariants?

  Guess, check and repair approach

  -Start with invariant candidates

  -Check if candidates are actual invariants

  -If not, refine the candidate and check again

Invariant Candidate

- ***Invariant Candidate***: $C = \langle u, \Phi(u,v) \rangle$

Fixed variables

Subset of propositions in D

$C_1 = \langle \langle ?obj \rangle, \{at(?obj,?loc)\} \rangle$

$C_2 = \langle \langle ?loc \rangle, \{at(?obj,?loc)\} \rangle$

$C_3 = \langle \langle ?obj \rangle, \{at(?obj,?loc), in(?obj,?trk)\} \rangle$

Instance of Candidate

- ***Invariant Candidate***: $C = \langle u, \Phi(u,v) \rangle$

Fixed variables

Subset of propositions in D

$C_1 = \langle \langle ?obj \rangle, \{at(?obj,?loc)\} \rangle$
$C_2 = \langle \langle ?loc \rangle, \{at(?obj,?loc)\} \rangle$
$C_3 = \langle \langle ?obj \rangle, \{at(?obj,?loc), in(?obj,?trk)\} \rangle$

- ***Instance I of candidate C:***

$I_{1\text{-}T1} = \{at(T_1,?loc)\}$
$I_{2\text{-}ARC} = \{at(?obj,ARC)\}$
$I_{3\text{-}T2} = \{at(T_2,?loc), in(T_2,?trk)\}$

Instance of Candidate

- ***Invariant Candidate***: $C = \langle u, \Phi(u,v) \rangle$

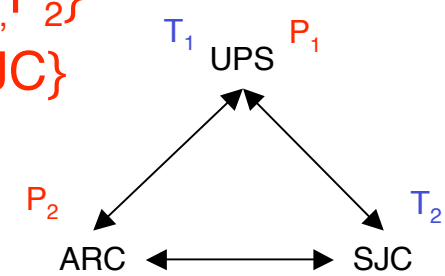
Fixed variables

Subset of propositions in D

$C_1 = \langle \langle ?obj \rangle, \{at(?obj,?loc)\} \rangle$
$C_2 = \langle \langle ?loc \rangle, \{at(?obj,?loc)\} \rangle$
$C_3 = \langle \langle ?obj \rangle, \{at(?obj,?loc), in(?obj,?trk)\} \rangle$

- ***Weight of Instance I in s :*** $W(I,s) \equiv |\{v : \Phi(I,v)\}|$

$I_{1\text{-}T1} = \{at(T_1,?loc)\}$ w=1 {UPS}
$I_{2\text{-}ARC} = \{at(?obj,ARC)\}$ w=2 $\{T_1, P_2\}$
$I_{3\text{-}T2} = \{at(T_2,?loc), in(T_2,?trk)\}$ w=1 {SJC}



Weight of Instance

- ***Invariant Candidate***: $C = \langle u, \Phi(u,v) \rangle$

  Fixed variables — Subset of propositions in D

- ***Weight of instance I in s***: $W(I,s) \equiv |\{v : \Phi(I,v)\}|$

- ***Want to show***: $\forall i \in Inst(C), s \;\; W(i,s) \leq 1$

$$\Phi(I,V_1) \otimes \Phi(I,V_2) \otimes \Phi(I,V_3) \otimes \ldots$$

Checking Candidates

When is a candidate C an actual invariant?

Sufficient condition:

1. Weight of C in initial state ≤ 1
2. C is not ***threatened*** by any operator op in D

op does not ***threaten*** C if:

op does not increase the weight of any instance of C beyond 1

Type 1 Ops – Inert

$C_1 = \langle \langle ?obj \rangle, \{at(?obj,?loc)\} \rangle$

op does not ***threaten*** C if:

1. op does not affect the weight of C

at(?t,?l)
washrack(?l)

weight=k

wash-truck (?t)

weight=k

Type 2 Ops – Decreasing

$C_1 = \langle \langle ?obj \rangle, \{at(?obj,?loc)\} \rangle$

op does not ***threaten*** C if:

2. op decreases the weight of C by 1

at(?t,?l) weight=k

vaporize-truck(?t,?l)

¬at(?t,?l) weight=k-1

Type 2 Ops – Decreasing

$C_1 = \langle \langle ?obj \rangle, \{at(?obj,?loc)\} \rangle$

op does not ***threaten*** C if:

2. op decreases the weight of C by 1

	at(?t,?l)		weight=k
2s		vaporize-truck(?t,?l)	
	¬at(?t,?l)		weight=k-1
		at(?t,?l)	weight=k
2e		vaporize-truck(?t,?l)	
		¬at(?t,?l)	weight=k-1
	at(?t,?l)		weight=k
2x		vaporize-truck(?t,?l)	
		¬at(?t,?l)	weight=k-1

Type 3 Ops – Increasing

op does not ***threaten*** C if:

3. op increases the weight by 1 at a timepoint
   but its conditions require that the weight be 0 at that timepoint

$\forall ?l : \neg at(?t,?l)$ weight=0

build-truck(?t,?n)

at(?t,?n) weight=1

Type 3 Ops – Increasing

op does not ***threaten*** C if:

3. op increases the weight by 1 at a timepoint
   but its conditions require that the weight be 0 at that timepoint

3s

$\forall ?l : \neg at(?t,?l)$ weight=0

build-truck(?t,?n)

at(?t,?n) weight=1

3e

$\forall ?l : \neg at(?t,?l)$ weight=0

build-truck(?t,?n)

at(?t,?n) weight=1

3x

$\forall ?l : \neg at(?t,?l)$ weight=0

build-truck(?t,?n)

at(?t,?n) weight=1

Type 4 Ops – Balanced

op does not ***threaten*** C if:

4. op is ***balanced*** at all timepoints

	at(?t,?l1)	weight=k
4s	Insta-Drive (?t,?l1,?l2)	
	¬ at(?t,?l1)	weight=k-1
	at(?t,?l2)	weight=k

	at(?t,?l1)	weight=k
4e	Insta-Drive (?t,?l1,?l2)	
	¬ at(?t,?l1)	weight=k-1
	at(?t,?l2)	weight=k

Type 5 Ops – Unbalanced

op does not ***threaten*** C if:

5. op is ***temporarily unbalanced*** and certain ***mutex*** conditions apply

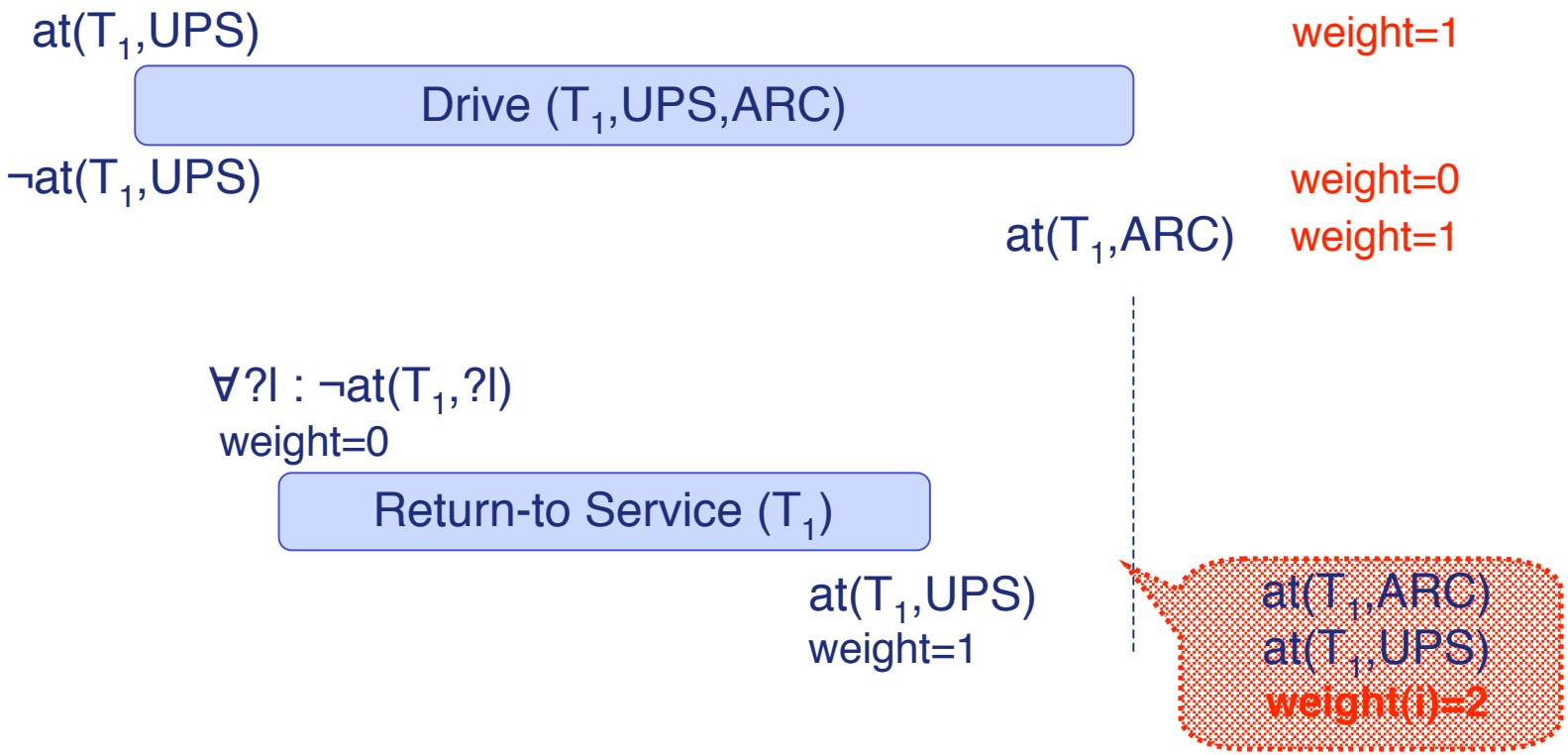
	at(?t,?l1)		weight=1
5s	Drive (?t,?l1,?l2)		
	¬at(?t,?l1)		weight=0
		at(?t,?l1)	weight=1

	at(?t,?l1)		weight=1
5e	Drive (?t,?l1,?l2)		
		¬at(?t,?l1)	weight=0
		at(?t,?l1)	weight=1

Type 5s Ops – Unbalanced

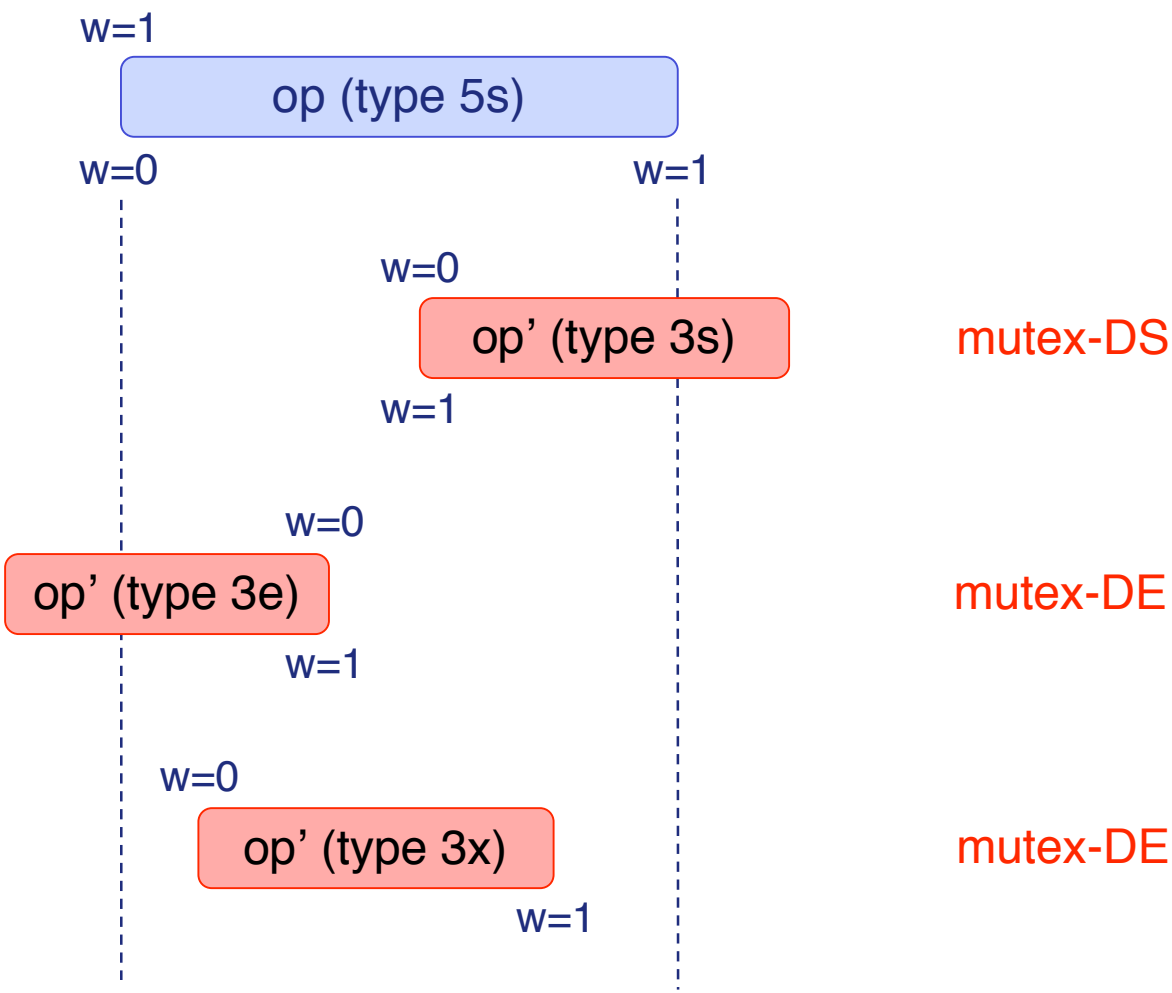
When op is ***temporarily unbalanced,*** we require that all operators that could intervene and increase the weight are ***mutex*** with op

Those operators may disrupt the balance of the candidate



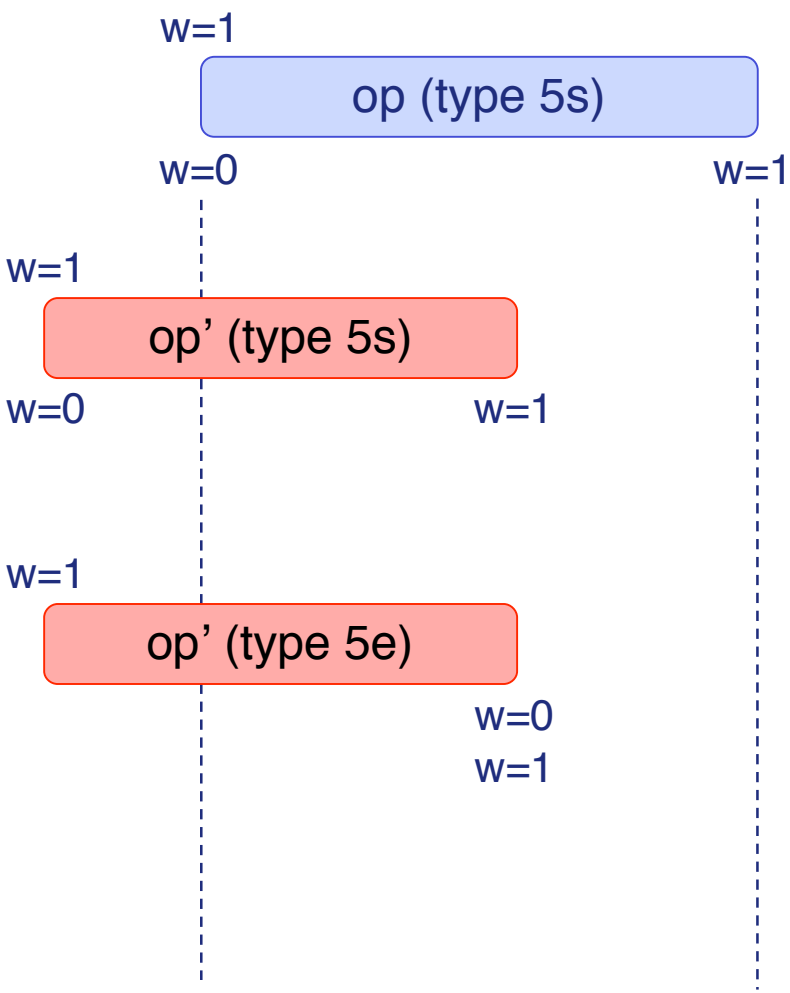
Type 5s – Mutex Requirements

op is a ***non-threatening temporarily unbalanced operator*** if for each op':



Type 5s – Mutex Requirements

op is a ***non-threatening temporarily unbalanced operator*** if for each op':



not possible

mutex-DE

Type 5 Ops – Unbalanced

op does not ***threaten*** C if:

5. op is ***temporarily unbalanced*** and certain ***mutex*** conditions apply

5s

at(?t,?l1)

Drive (?t,?l1,?l2)

¬at(?t,?l1)

at(?t,?l1)

mutex w/type 3, 5e

5e

at(?t,?l1)

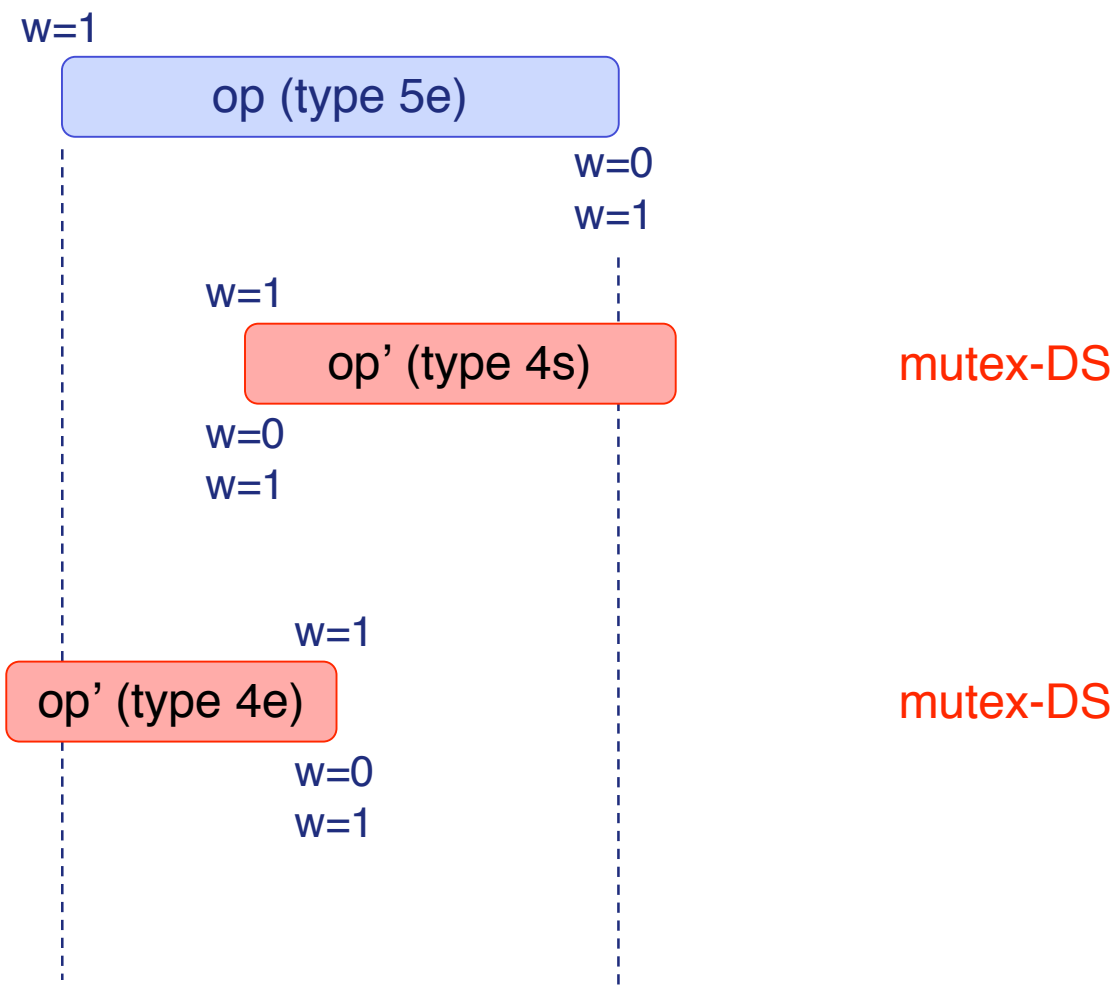
Drive (?t,?l1,?l2)

¬at(?t,?l1)

at(?t,?l1)

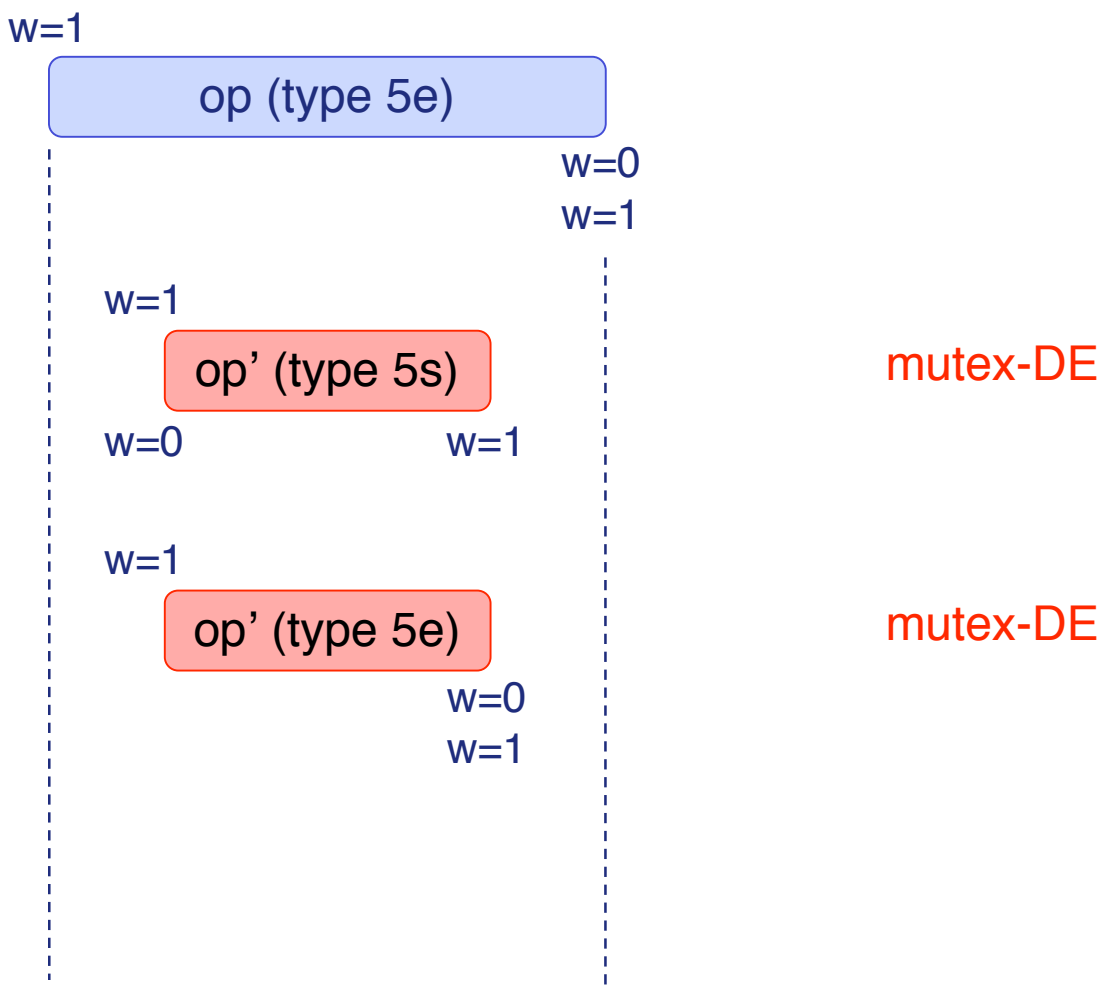
Type 5e – Mutex Requirements

op is a ***non-threatening temporarily unbalanced operator*** if for each op':



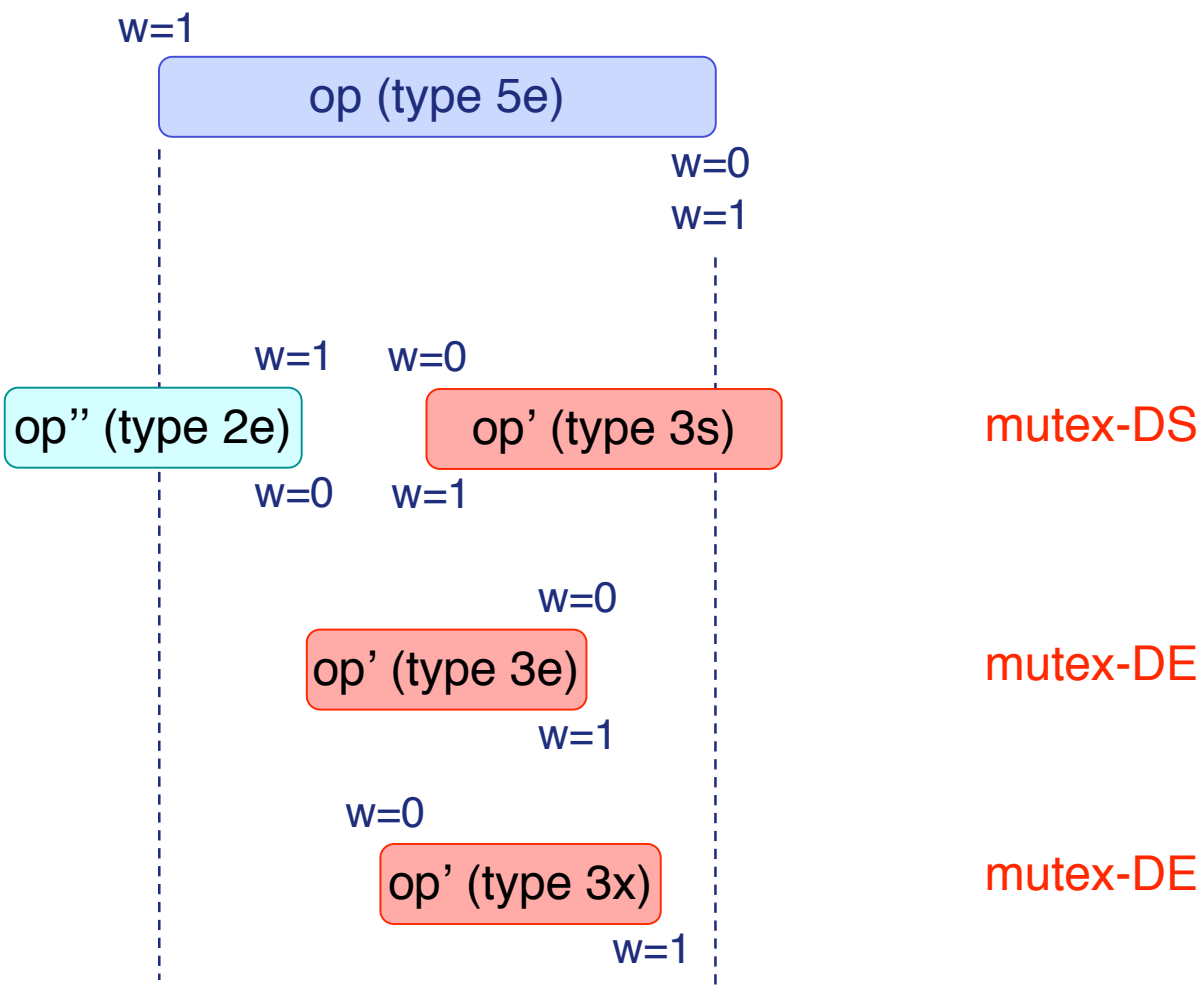
Type 5e – Mutex Requirements

op is a ***non-threatening temporarily unbalanced operator*** if for each op':



Type 5e – Mutex Requirements

op is a ***non-threatening temporarily unbalanced operator*** if for each op':



Type 5 Ops – Unbalanced

op does not ***threaten*** C if:

5. op is ***temporarily unbalanced*** and certain ***mutex*** conditions apply

5s

at(?t,?l1)

Drive (?t,?l1,?l2)

¬at(?t,?l1) at(?t,?l1)

mutex w/type 3, 5e

5e

at(?t,?l1)

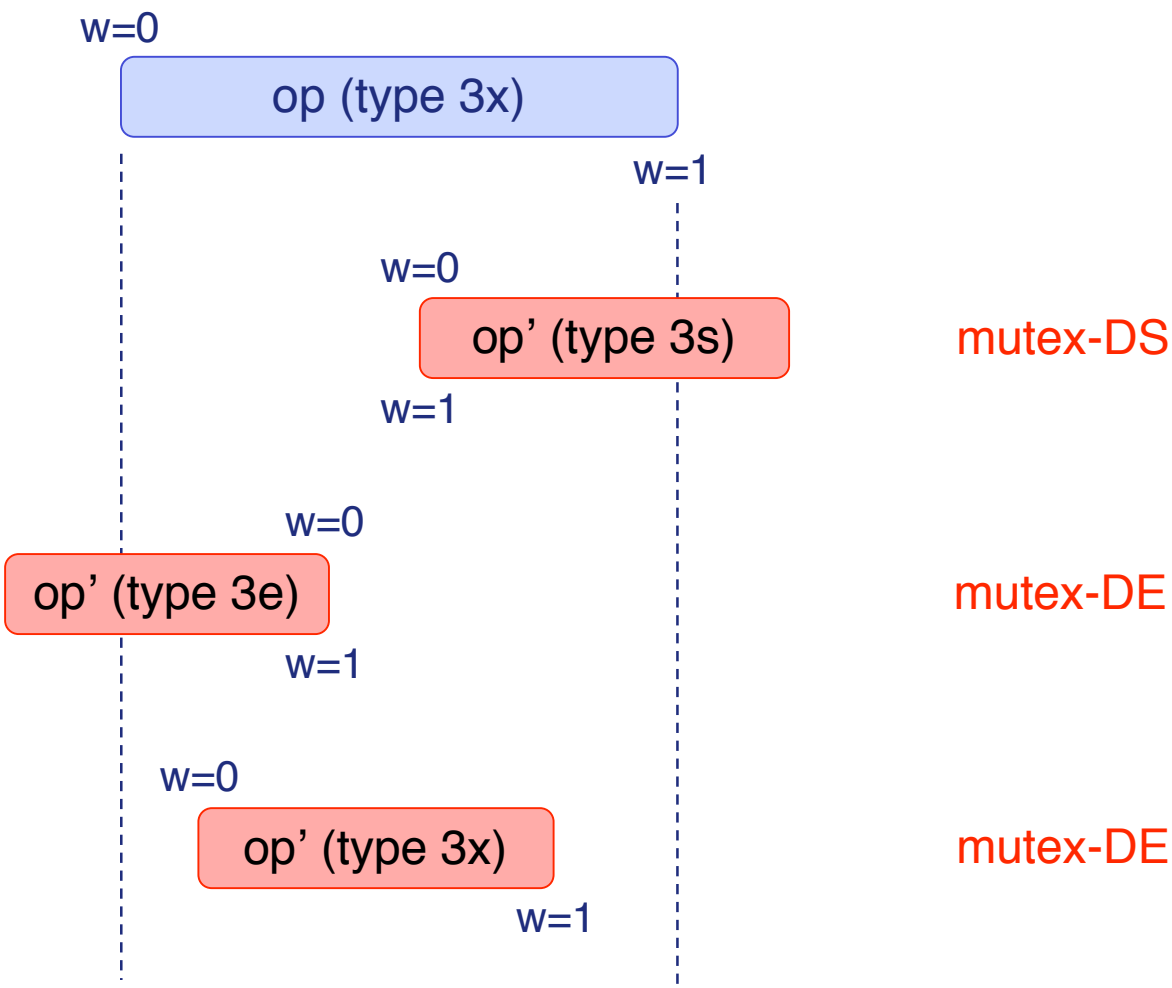
Drive (?t,?l1,?l2)

¬at(?t,?l1)
at(?t,?l1)

mutex w/type 3, 4, 5

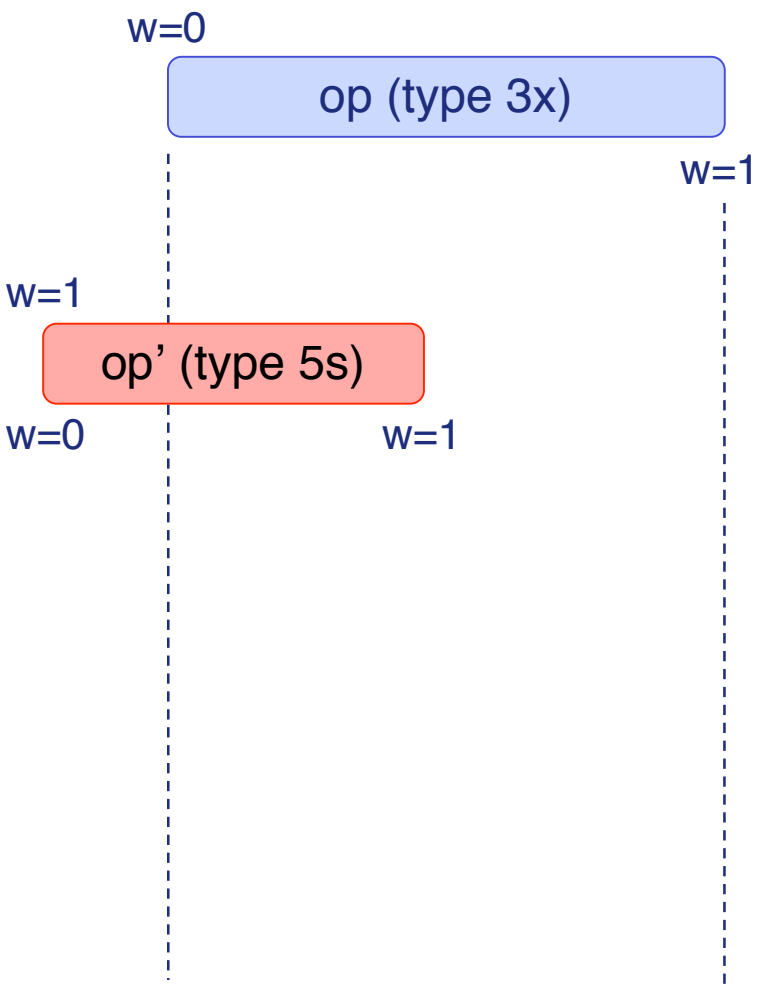
Type 3x – Mutex Requirements

op is a ***non-threatening temporarily unbalanced operator*** if for each op':



Type 3x – Mutex Requirements

op is a ***non-threatening temporarily unbalanced operator*** if for each op':



mutex-DE

Type 3 Ops – Increasing

op does not ***threaten*** C if:

3. op increases the weight by 1 at a timepoint
   but its conditions require that the weight be 0 at that timepoint

3s

$\forall ?l : \neg at(?t,?l)$ weight=0

build-truck(?t,?n)

at(?t,?n) weight=1

3e

$\forall ?l : \neg at(?t,?l)$ weight=0

build-truck(?t,?n)

at(?t,?n) weight=1

3x

$\forall ?l : \neg at(?t,?l)$ weight=0

build-truck(?t,?n)

at(?t,?n) weight=1

Checking Candidates recap

op does not ***threaten*** C if:

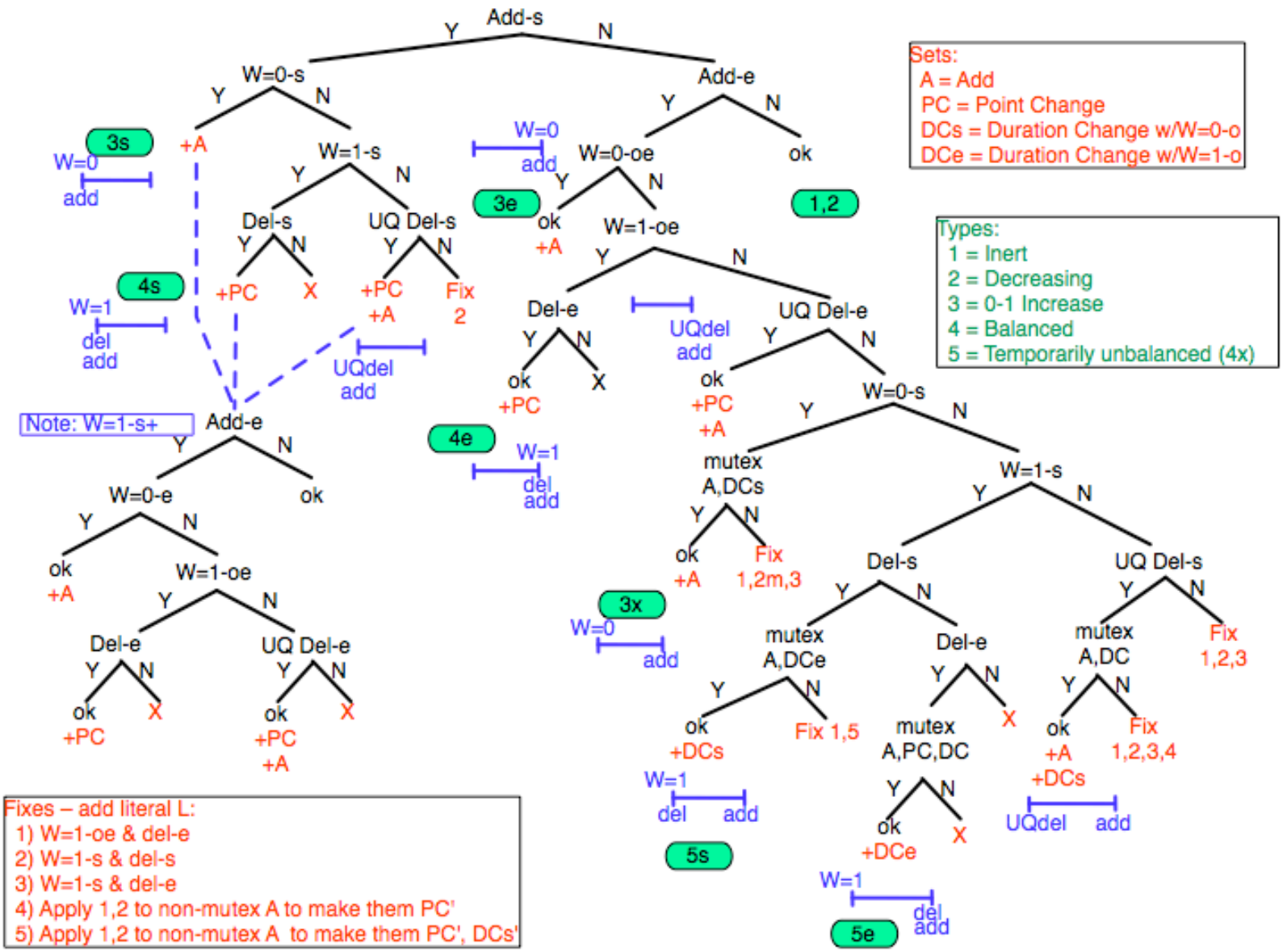
1. Inert – op does not affect the weight
2. Decreasing – op decreases the weight by 1
3. Increasing – op increases the weight by 1
   but conditions require that the weight is 0 prior

   3x – mutex with all type 3, type 5s
4. Balanced – op decreases then increases weight by 1
5. Temporarily unbalanced – spread out over time (4x)

   5s – mutex with all type 3, type 5e

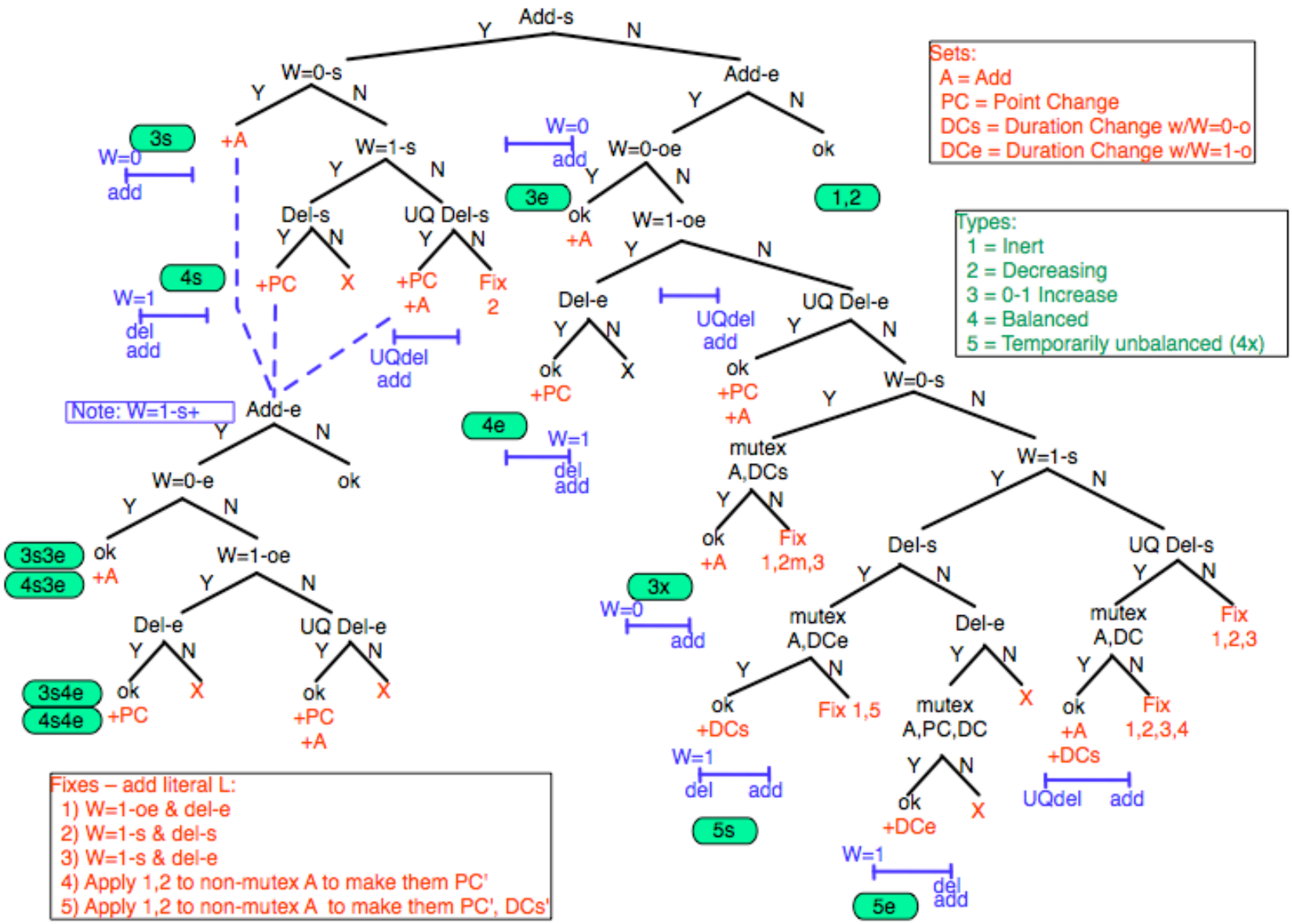
   5e – mutex with all type 3, type 4, type 5

More Difficult Cases



Ames Research Center

More Difficult Cases



Finding Invariants

Guess, check and repair approach

-Start with invariant candidates

$C_{at1} = \langle \langle ?obj \rangle, \{at(?obj,?loc)\} \rangle$ w=1

$C_{at2} = \langle \langle ?loc \rangle, \{at(?obj,?loc)\} \rangle$ w=2 X

$C_{in1} = \langle \langle ?obj \rangle, \{in(?obj,?trk)\} \rangle$ w=0

$C_{in2} = \langle \langle ?trk \rangle, \{in(?obj,?trk)\} \rangle$ w=0

-Check if candidates are actual invariants

initial conditions

Drive, Load, Unload

-If not, refine the candidate and check again

add additional predicates

Example

Candidate: $C_{at1} = \langle \langle ?obj \rangle, \{at(?obj,?loc)\} \rangle$

initially: w=1

Drive op: type 5s

mutex w/type 3?

at(?trk,?loc1)

Drive(?trk,?loc1,?loc2)

¬at(?trk,?loc1) at(?trk,?loc2)

Load op: type 2s

at(?trk,?loc) →
at(?pkg,?loc)

Load(?pkg,?trk)

¬at(?pkg,?loc) in(?pkg,?trk)

UnLoad op: unprotected add!

at(?trk,?loc) →
in(?pkg,?trk)

Unload(?pkg,?trk)

¬in(?pkg,?trk) at(?pkg,?loc)

Example

Candidate: $C_{at1} = \langle \langle ?obj \rangle, \{at(?obj,?loc)\} \rangle$

initially: w=1

Drive op: type 5s

mutex w/type 3?

at(?trk,?loc1)

Drive(?trk,?loc1,?loc2)

¬at(?trk,?loc1) at(?trk,?loc2)

Load op: type 2s

at(?trk,?loc) →
at(?pkg,?loc)

Load(?pkg,?trk)

¬at(?pkg,?loc) in(?pkg,?trk)

UnLoad op: unprotected add!

at(?trk,?loc) →
in(?pkg,?trk)

Unload(?pkg,?trk)

¬in(?pkg,?trk) at(?pkg,?loc)

Refine candidate: add in(?pkg,?trk)

Example

Candidate: $C_{at\text{-}in1} = \langle \langle ?obj \rangle, \{at(?obj,?loc), in(?obj, ?trk)\} \rangle$

initially: w=1

Drive op: type 5s

mutex w/type 3?

at(?trk,?loc1)

Drive(?trk,?loc1,?loc2)

¬at(?trk,?loc1) at(?trk,?loc2)

Load op: type 5s

at(?trk,?loc) →
at(?pkg,?loc)

Load(?pkg,?trk)

¬at(?pkg,?loc) in(?pkg,?trk)

UnLoad op: type 5s

at(?trk,?loc) →
in(?pkg,?trk)

Unload(?pkg,?trk)

¬in(?pkg,?trk) at(?pkg,?loc)

Finding Invariants

$C_{at1} = \langle \langle ?obj \rangle, \{at(?obj,?loc)\} \rangle$ UnLoad $\Rightarrow$ in(?obj,?trk)

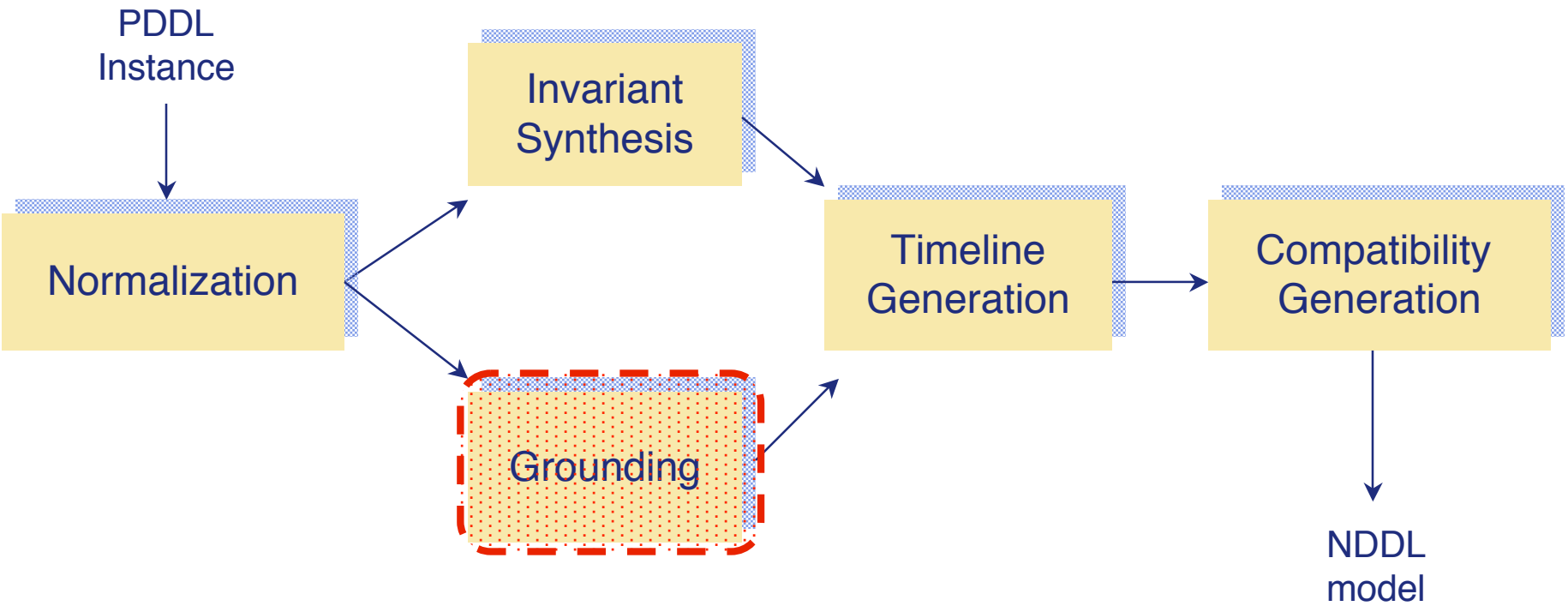
$C_{at2} = \langle \langle ?loc \rangle, \{at(?obj,?loc)\} \rangle$ X (init)

$C_{in1} = \langle \langle ?obj \rangle, \{in(?obj,?trk)\} \rangle$ Load $\Rightarrow$ at(?obj,?trk)

$C_{in2} = \langle \langle ?trk \rangle, \{in(?obj,?trk)\} \rangle$ X (Load)

$C_{at1\text{-}in1} = \langle \langle ?obj \rangle, \{at(?obj,?loc), in(?obj, ?trk)\} \rangle$

Translation Steps



Grounding

[Helmert-2009]

- ***Grounded*** PDDL domain:

  all the literals occurring in the goal formula and operators are ground literals

- How to calculate a grounded PDDL domain?

  - compute an ***over-approximation*** of the atoms that can ever be true
  - instantiate the goal formula and the operators on those atoms

- Over-approximation of reachable atoms:

  by using a ***relaxed version*** of the domain

  - delete effects of operators are ignored
  - negative literals in conditions assumed true

Relaxed PlanGraph

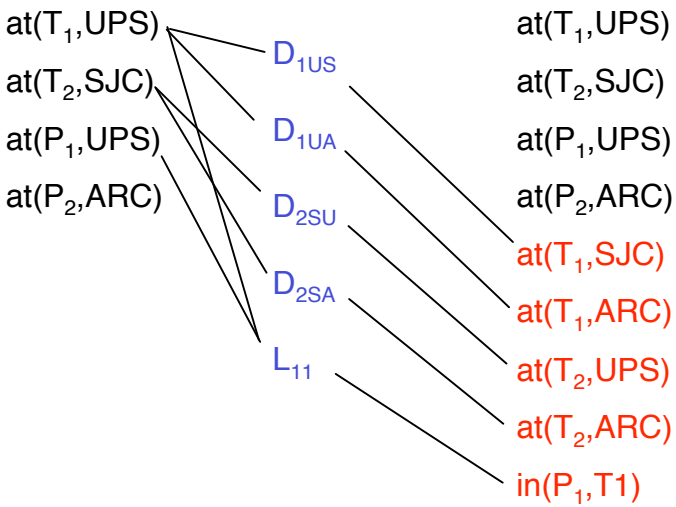
at(T_1,UPS)

at(T_2,SJC)

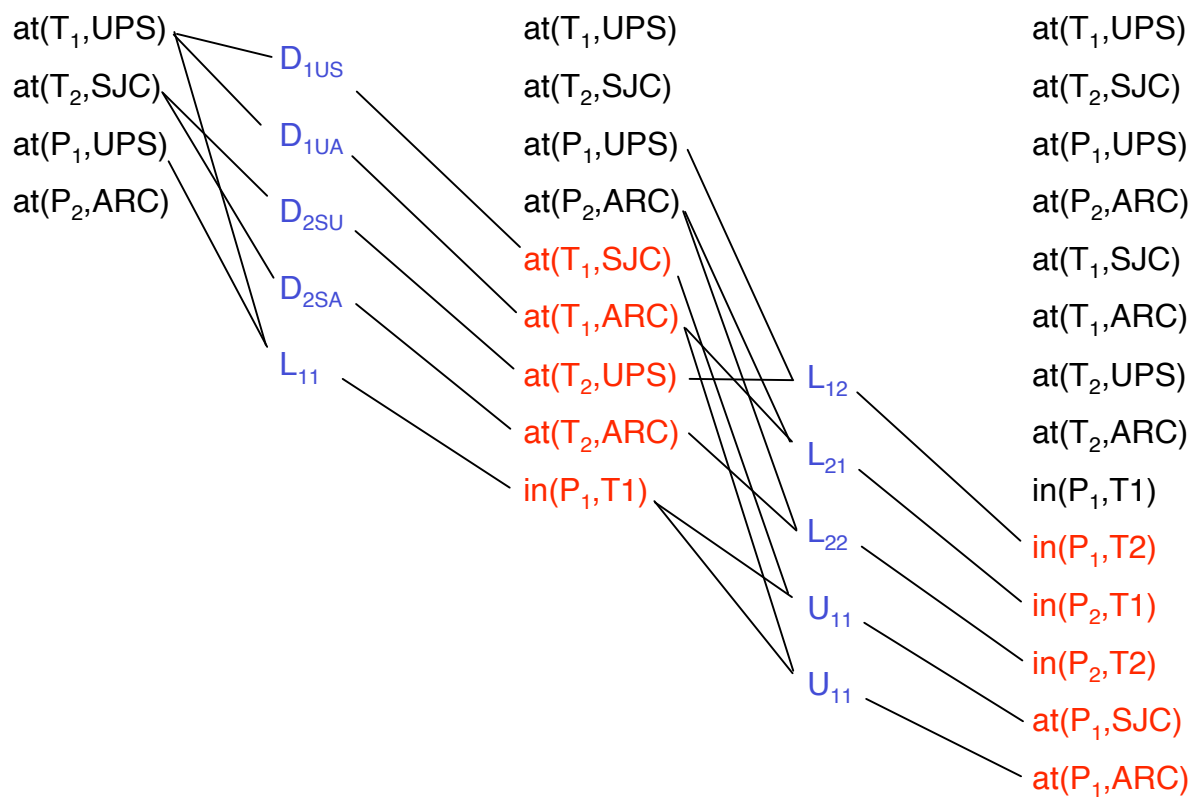
at(P_1,UPS)

at(P_2,ARC)

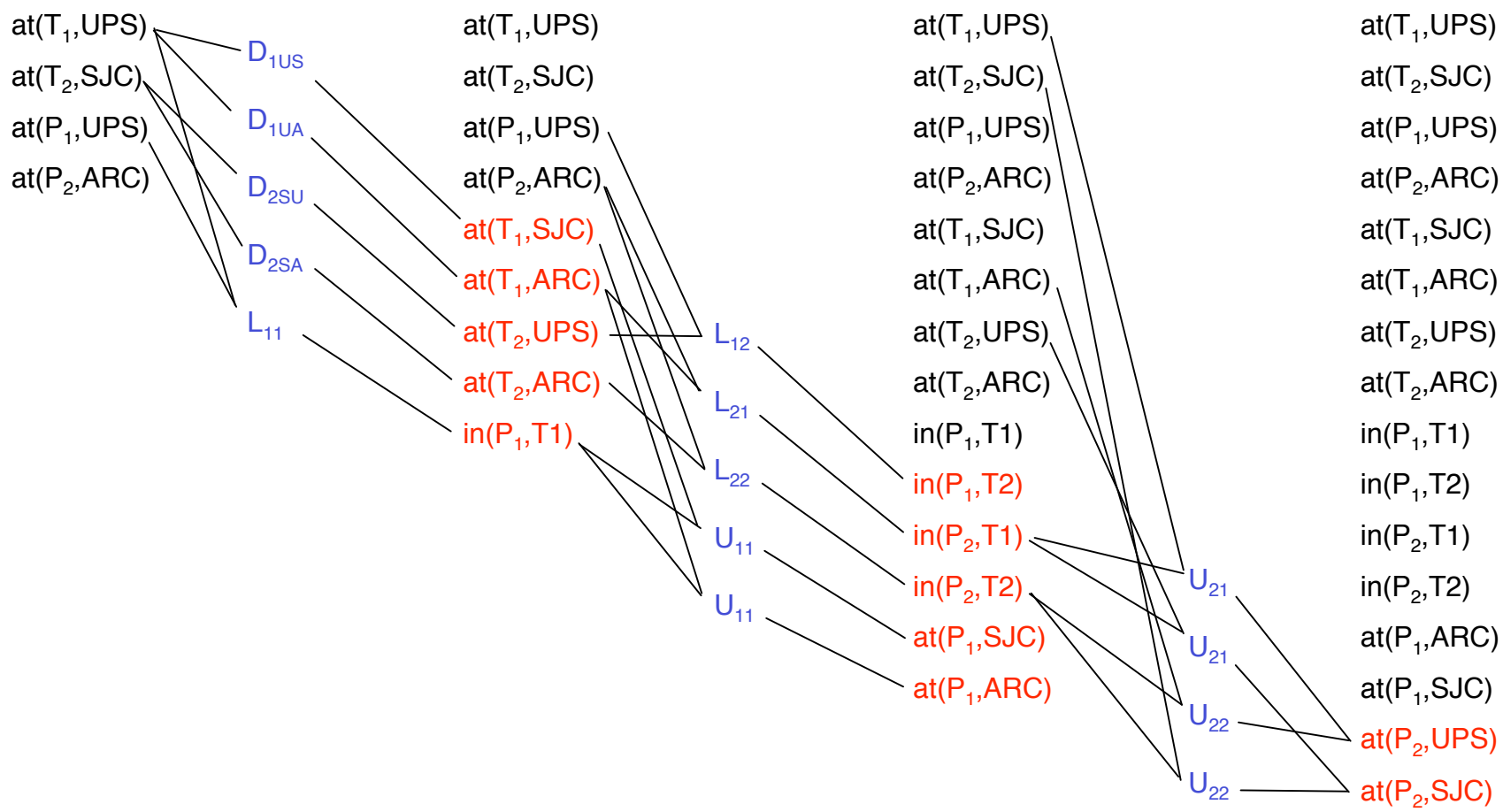
Relaxed PlanGraph



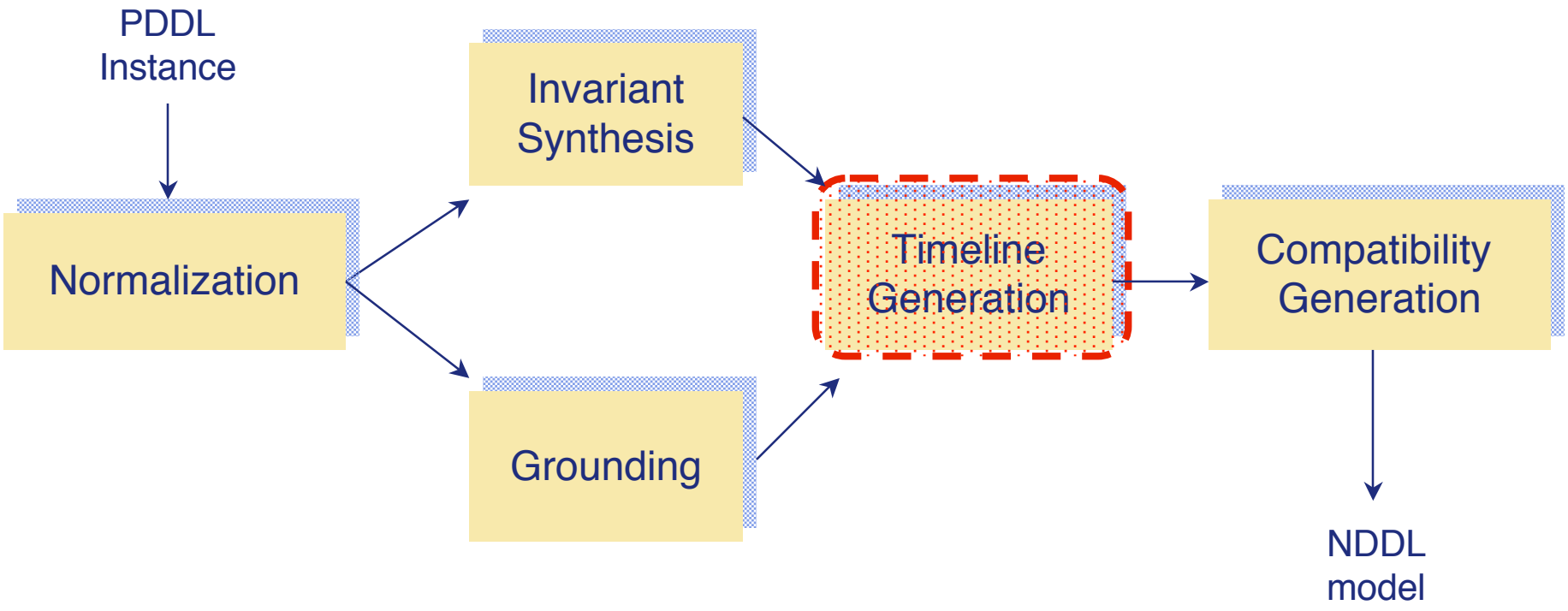
Relaxed PlanGraph



Relaxed PlanGraph



Translation Steps



Timeline Generation

- We synthesize a set of timelines from each invariant:

Invariant: $C_{at\text{-}in1} = \langle \langle ?obj \rangle, \{at(?obj,?loc), in(?obj, ?trk)\} \rangle$

Instances: i_1:?obj $\rightarrow T_1$
i_2:?obj $\rightarrow T_2$
i_3:?obj $\rightarrow P_1$
i_4:?obj $\rightarrow P_2$

$\Downarrow$

Four ***timelines***: $\Gamma_{at\text{-}in\text{-}T1}$, $\Gamma_{at\text{-}in\text{-}T2}$, $\Gamma_{at\text{-}in\text{-}P1}$, $\Gamma_{at\text{-}in\text{-}P2}$

- ***Activities*** for $\Gamma_{at\text{-}in\text{-}P1}$?

Reachable ground atoms with ?obj = P_1:

Activities[$\Gamma_{at\text{-}in\text{-}P1}$] = $\{at(P_1,ARC), at(P_1,UPS), at(P_1,SJC), in(P_1,T_1), in(P_1,T_2), \perp\}$

Proposition Timelines

$\Gamma_{\text{at-in-T1}}$: at(T_1,ARC), at(T_1,UPS), at(T_1,SJC), $\perp$

$\Gamma_{\text{at-in-T2}}$: at(T_2,ARC), at(T_2,UPS), at(T_2,SJC), $\perp$

$\Gamma_{\text{at-in-P1}}$: at(P_1,ARC), at(P_1,UPS), at(P_1,SJC), in(P_1,T_1), in(P_1,T_2), $\perp$

$\Gamma_{\text{at-in-P2}}$: at(P_2,ARC), at(P_2,UPS), at(P_2,SJC), in(P_2,T_1), in(P_2,T_2), $\perp$

Op Timelines

For each ground operator op in D $\Rightarrow$ one timeline T_{op}

Activities for T_{op}?

Two activities: Exe (executing)
Idle (not executing)

$\Gamma_{Drive(T1,UPS,ARC)}$: Exe, Idle

$\Gamma_{Drive(T1,UPS,SJC)}$: Exe, Idle

…

$\Gamma_{Drive(T2,UPS,SJC)}$: Exe, Idle

…

$\Gamma_{Load(P1,T1)}$: Exe, Idle

$\Gamma_{Load(P1,T2)}$: Exe, Idle

…

$\Gamma_{UnLoad(P1,T1)}$: Exe, Idle

…

Better Op Timelines

Cliques of mutex ops can go on the same timeline

Drive(T_1,UPS, ARC) ⊗ Drive(T_1,UPS, SJC) ⊗ …

Drive(T_2,UPS, ARC) ⊗ Drive(T_2,UPS, SJC) ⊗ …

$\Gamma_{Drive(T1)}$: Drive(T_1,UPS, ARC), Drive(T_1,UPS, SJC), …, Idle

$\Gamma_{Drive(T2)}$: Drive(T_2,UPS, ARC), Drive(T_2,UPS, SJC), …, Idle

Load(P_1,T_1) ⊗ UnLoad(P_1,T_1) ⊗ Load(P_1,T_2) ⊗ Load(P_1,T_2)

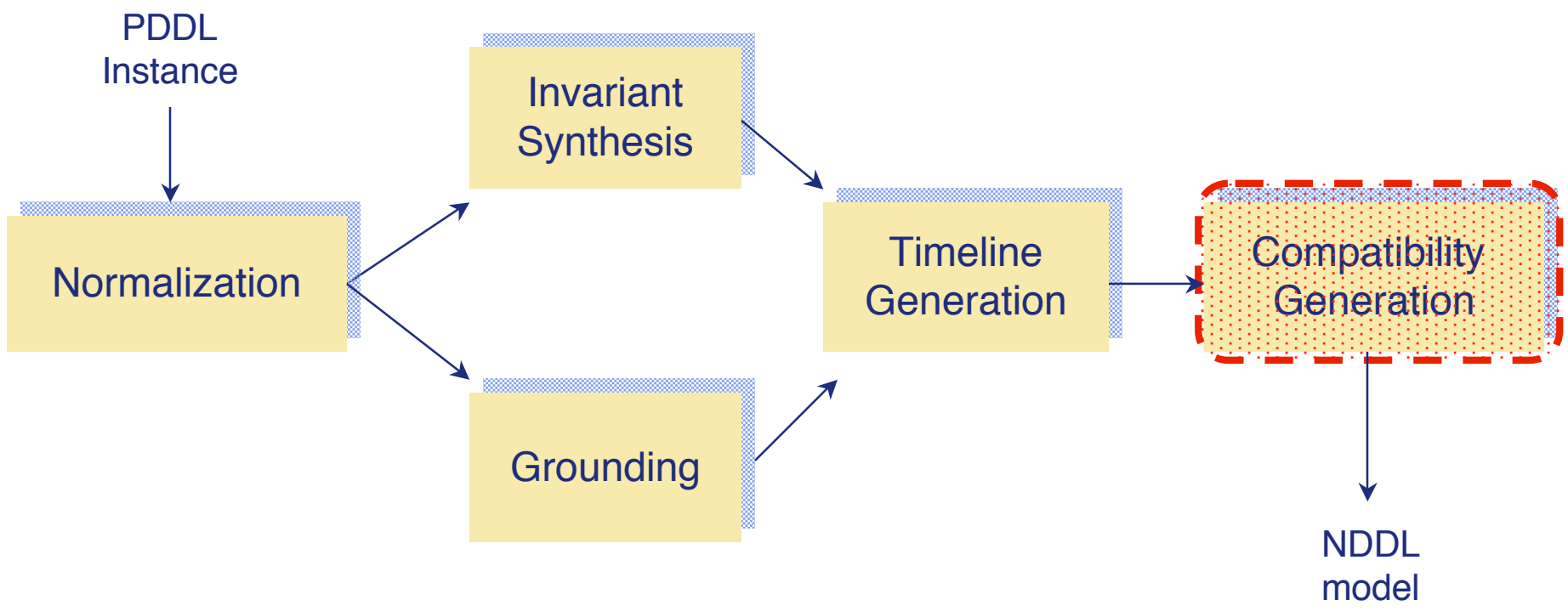
Load(P_2,T_1) ⊗ UnLoad(P_2,T_1) ⊗ Load(P_2,T_2) ⊗ UnLoad(P_2,T_2)

$\Gamma_{Load\text{-}UnLoad(P1)}$: Load($P_1$,$T_1$), UnLoad($P_1$,$T_1$), Load($P_1$,$T_2$), Load($P_1$,$T_2$), Idle

$\Gamma_{Load\text{-}UnLoad(P2)}$: Load($P_2$,$T_1$), UnLoad($P_2$,$T_1$), Load($P_2$,$T_2$), Load($P_2$,$T_2$), Idle

but Load(P_1,T_1) can occur in parallel with Load(P_2,T_1)

Translation Steps



Op Compatibility Synthesis

PDDL: op $\Rightarrow$ NDDL: T_{op}

How to generate the compatibilities for the activities exe and not_exe of the timeline T_{op}?

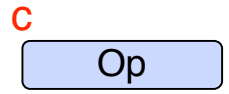
Compatibilities synthesized from the conditions Cond(op) and effects Eff(op)

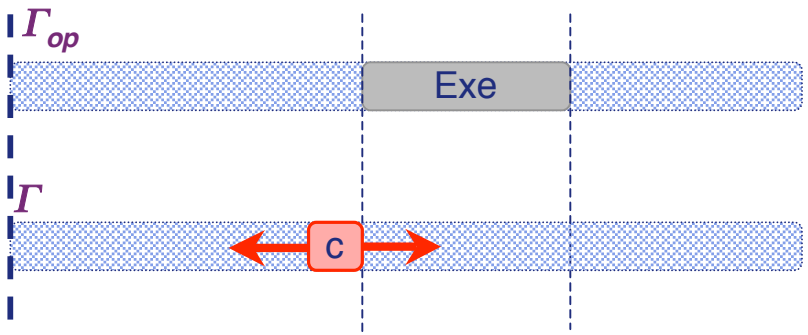
- ***Conditions***:
  - Positive at_start/over_all/at_end
  - Protected Negative at start/over_all/at_end
  - Unprotected Negative at start/over_all/at_end

- ***Effects***:
  - Add Effect at_start/at_end
  - Protected Delete Effect at_start/at_end
  - Unprotected Delete Effect at_start/at_end

Op Compatibility Synthesis

Positive Condition at_start:

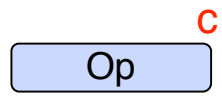


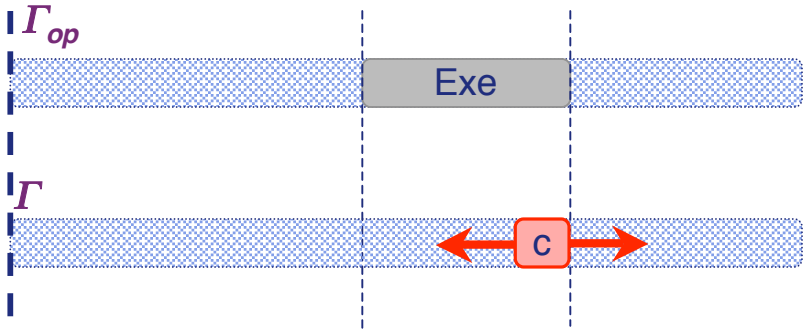


Γ.c.start < exe.start ≤ Γ.c.end

Γ_{op}:: exe{
starts_before_end Γ.c;
precedes (Γ.c.start,start);}

Positive Condition at_end:



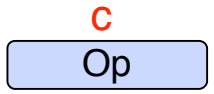


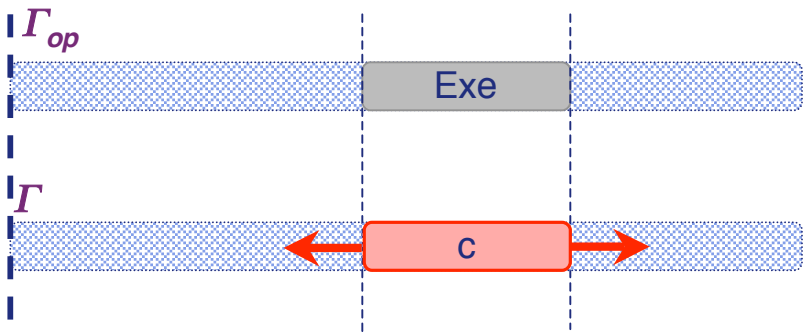
Γ.c.start < exe.end ≤ Γ.c.end

Γ_{op}:: exe{
ends_before_end Γ.c;
precedes (Γ.c.start,end);}

Op Compatibility Synthesis

Positive Condition over_all:





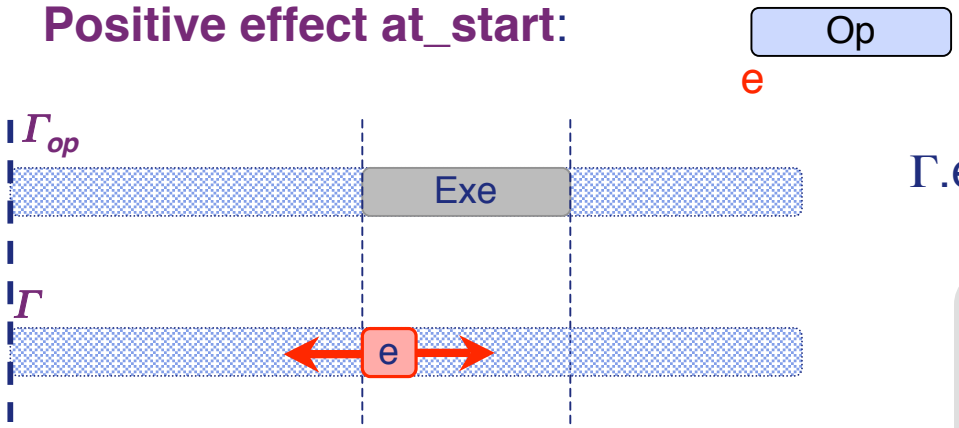
Γ.c.start ≤ exe.start
& exec.end ≤ Γ.c.end

Í

Γ_{op}:: exe{
starts_before_end Γ.c;
ends_before_end Γ.c;}

Op Compatibility Synthesis

Positive effect at_start:

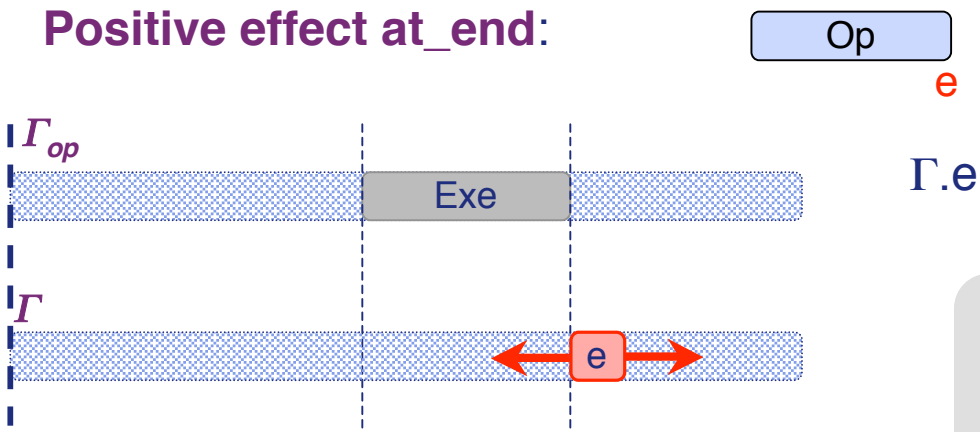


Γ.e.start ≤ exe.start < Γ.e.end

Í

Γ_{op}:: exe{
starts_after Γ.e;
precedes (start, Γ.e.end);}

Positive effect at_end:



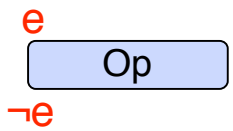
Γ.e.start ≤ exe.start < Γ.e.end

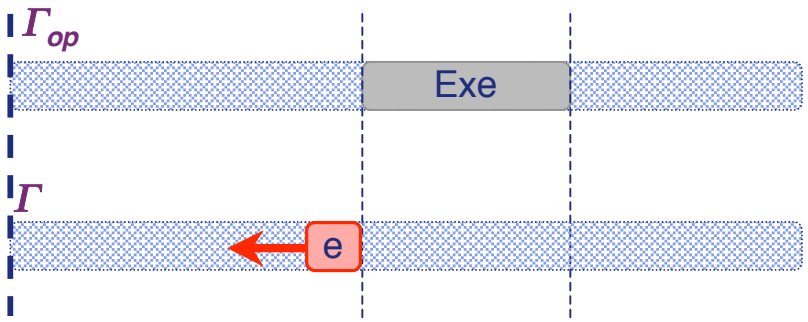
Í

Γ_{op}:: exe{
ends_before Γ.e;
precedes (end, Γ.e.end);}

Op Compatibility Synthesis

Condition and Negative effect at_start:





Γ.e.start < exe.start = Γ.e.end = Γ.$\perp$.start

Í

Γ_{op}:: exe{ met_by(Γ.e);
starts(Γ.$\perp$);}

Op Compatibility Synthesis

at(T_1,UPS)

Drive(T_1,UPS,ARC)

¬at(T_1,UPS) at(T_1,ARC)

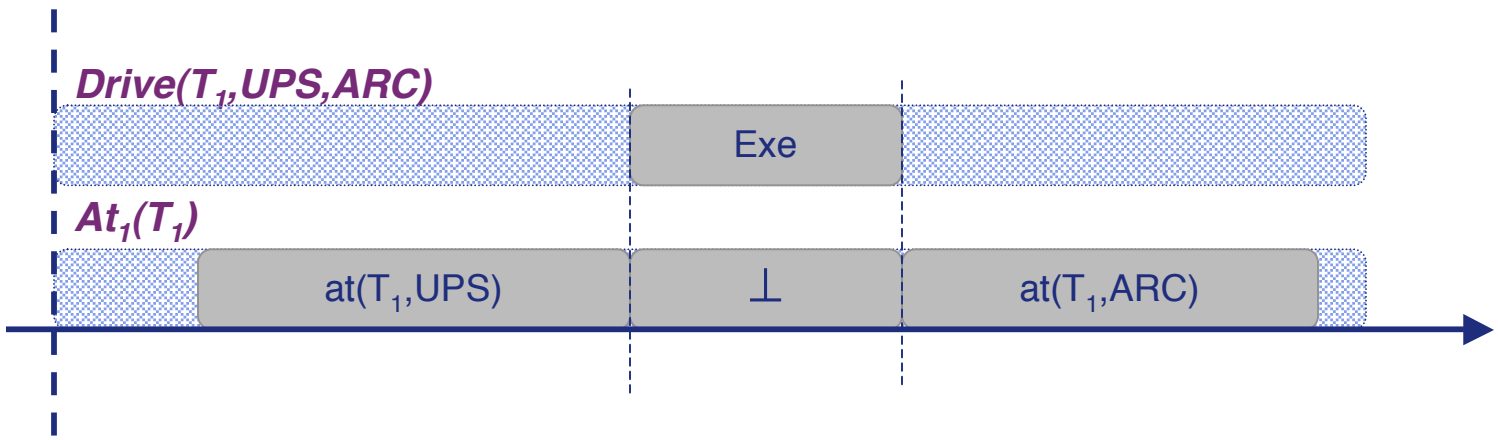
PDDL

```
:durative-action Drive(T1,UPS,ARC)
  :condition   at_start  at(T1,UPS)
  :effect (and
            at_start  ¬at(T1,UPS)
            at_end    at(T1,ARC)
```

Ë

NDDL

```
Drive(T1,UPS,ARC):: Exe{
        met_by(At1(T1).at(T1,UPS));
        equals(At1(T1).⊥);
        meets(At1(T1).at(T1,ARC));}
```



Explanatory Frame Axioms

Positive PDDL proposition p => NDDL activity p

- How to generate the compatibilities for the activity p?

They are synthesized on the basis of ***frame axioms*** of p

- Two disjunctions:

1. activities that cause the ***start*** of p
2. activities that cause the ***end*** of p

- We look for PDDL operators that contain p or its negation in their effects.

Compatibilities for Propositions

Γ

at(P1,UPS)

What activities:

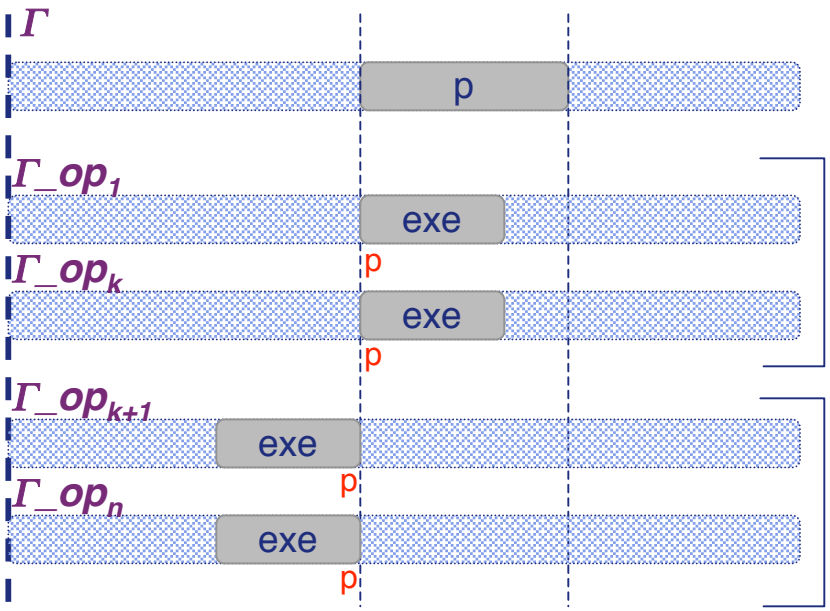
1. cause the ***start*** of p ?
2. cause the ***end*** of p ?

Explanatory frame axioms

1. Find all ops with effects p or ¬p
2. Form disjunction

Frame Axioms

Which operators cause the activity ***p*** to **start**?



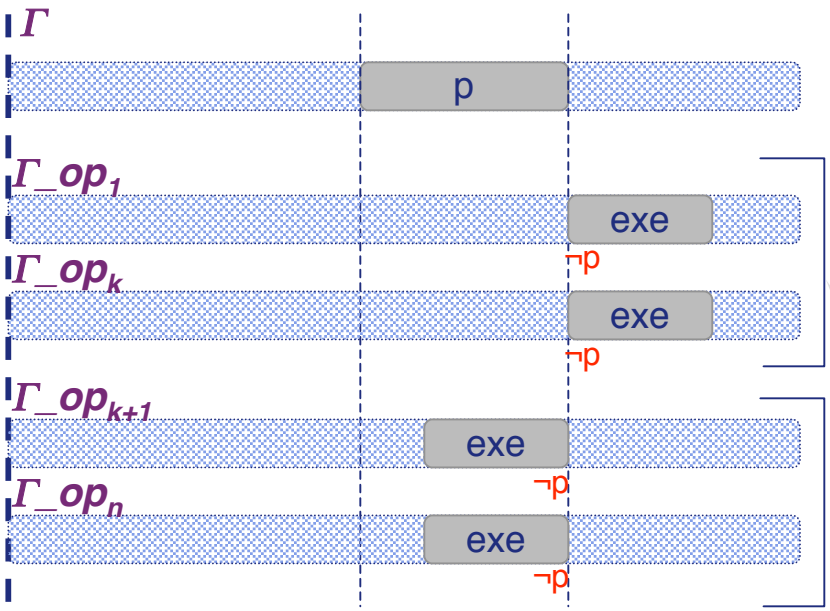

```
T:: p{
  if (guard=c1)
            starts(op1.exe);
  ...
  if (guard=ck)
            starts(opk.exe);
  if (guard=ck+1)
            met_by(opk+1.exe);
  ...
  if (guard=cn)
            met_by(opn.exe);}
```

operators whose effects **at_end** contain p

operators whose effects ***at_start*** contain p

Frame Axioms

Which operators cause the activity ***p*** to **end**?

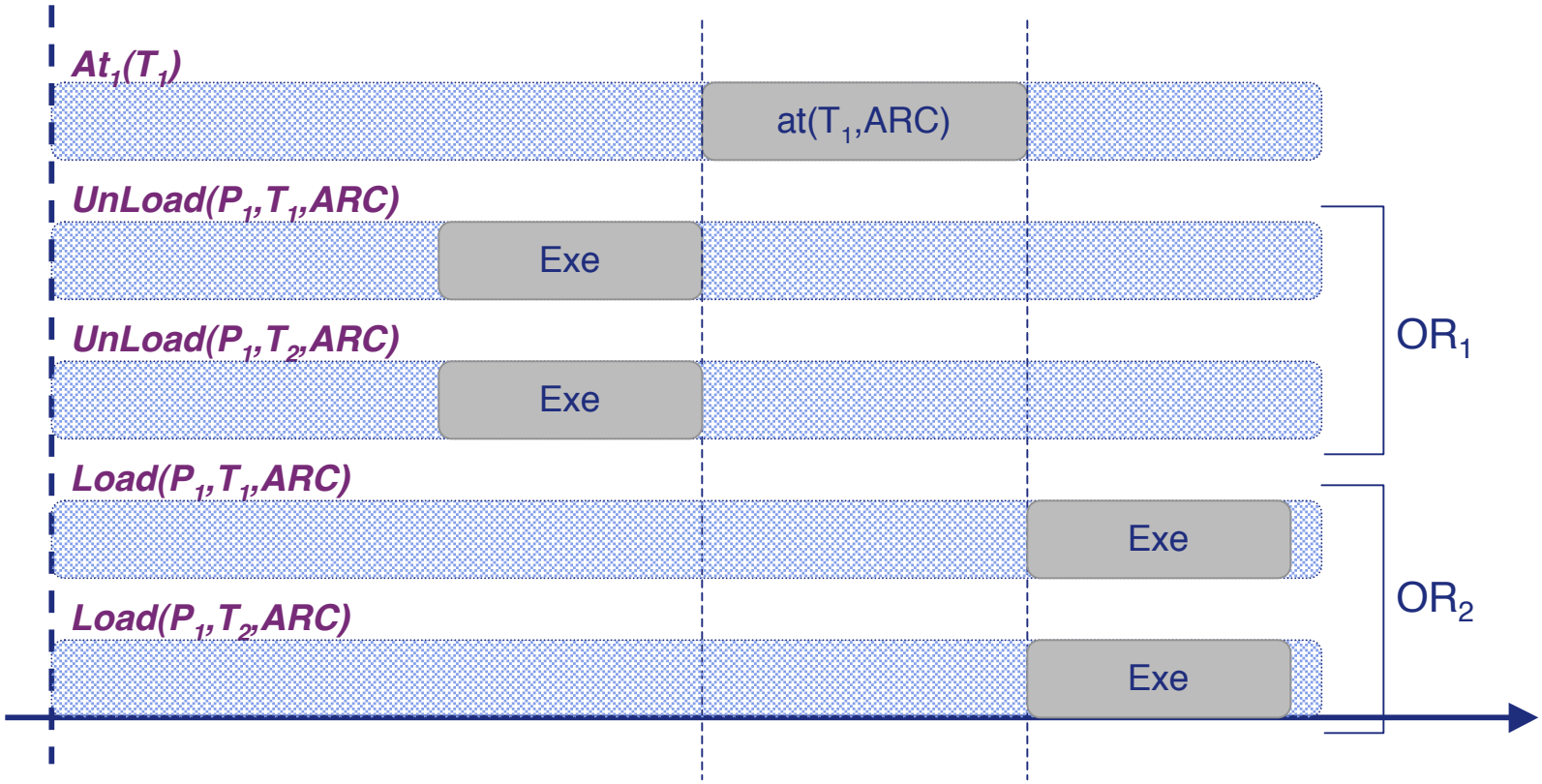



```
T:: p{
  if (guard=c1)
            meets(op1.exe);
  ...
  if (guard=ck)
            meets(opk.exe);
  if (guard=ck+1)
            ends(opk+1.exe);
  ...
  if (guard=cn)
            ends(opn.exe);}
```

operators whose effects **at_end** contain ¬p

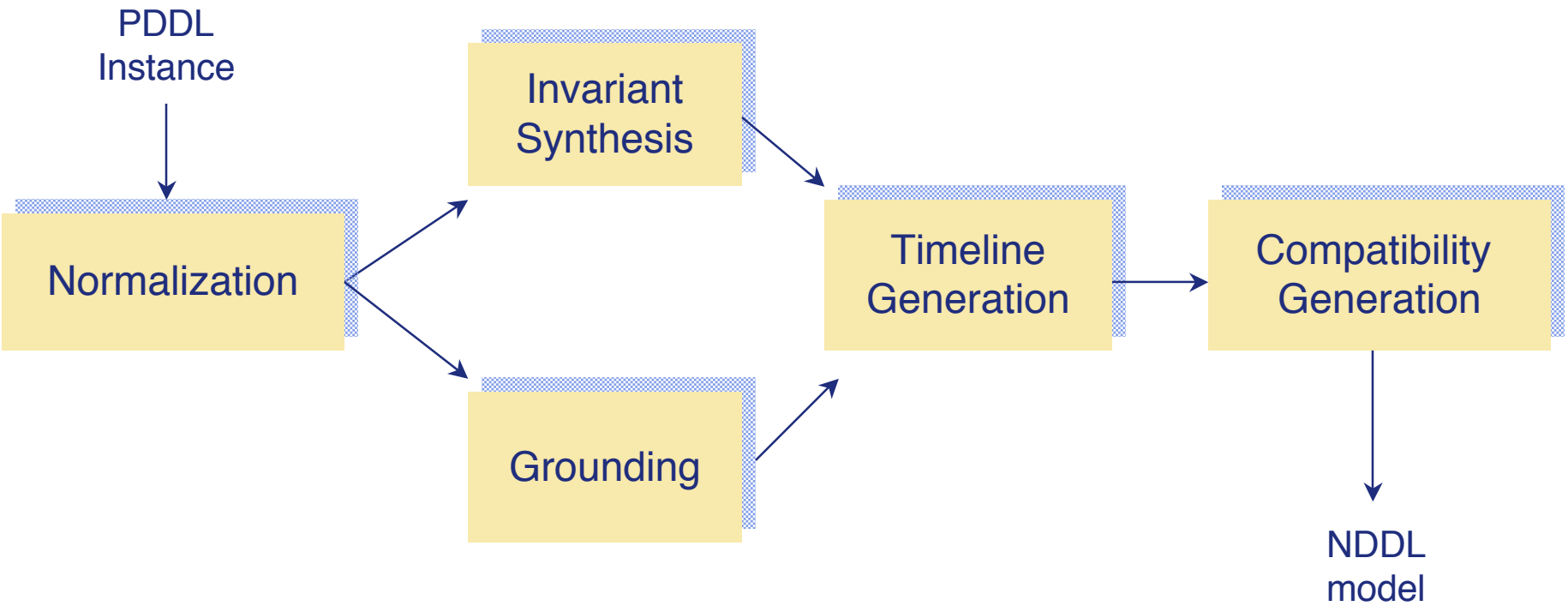
operators whose effects ***at_start*** contain ¬p

Frame Axioms



Ames Research Center

Translation Steps



Performance

[Helmert-2009]

Non-temporal problems from IPC 1-5: 1612

Runtime(s)	Problems
< 1	1054
1–10	452
10-30	67
30-60	10
60-120	14
> 120	15

Ames Research Center

Performance

[Helmert-2009]

Non-temporal IPC 1-5 problems: 1612

Runtime(s)	Problems
< 1	1054
1–10	452
10-30	67
30-60	10
60-120	14
> 120	15

Domain	Problem	Runtime(s)
Satellite	35	130
PSR-Large	47	146
Satellite	31	148
PSR-Large	42	154
PSR-Large	43	182
PSR-Large	45	182
Satellite	36	189
PSR-Large	40	198
PSR-Large	49	306
Satellite	32	372
PSR-Large	44	377
PSR-Large	46	570
PSR-Large	48	715
Satellite	33	806
PSR-Large	50	1191

989,250 Ops

544,209 Vars

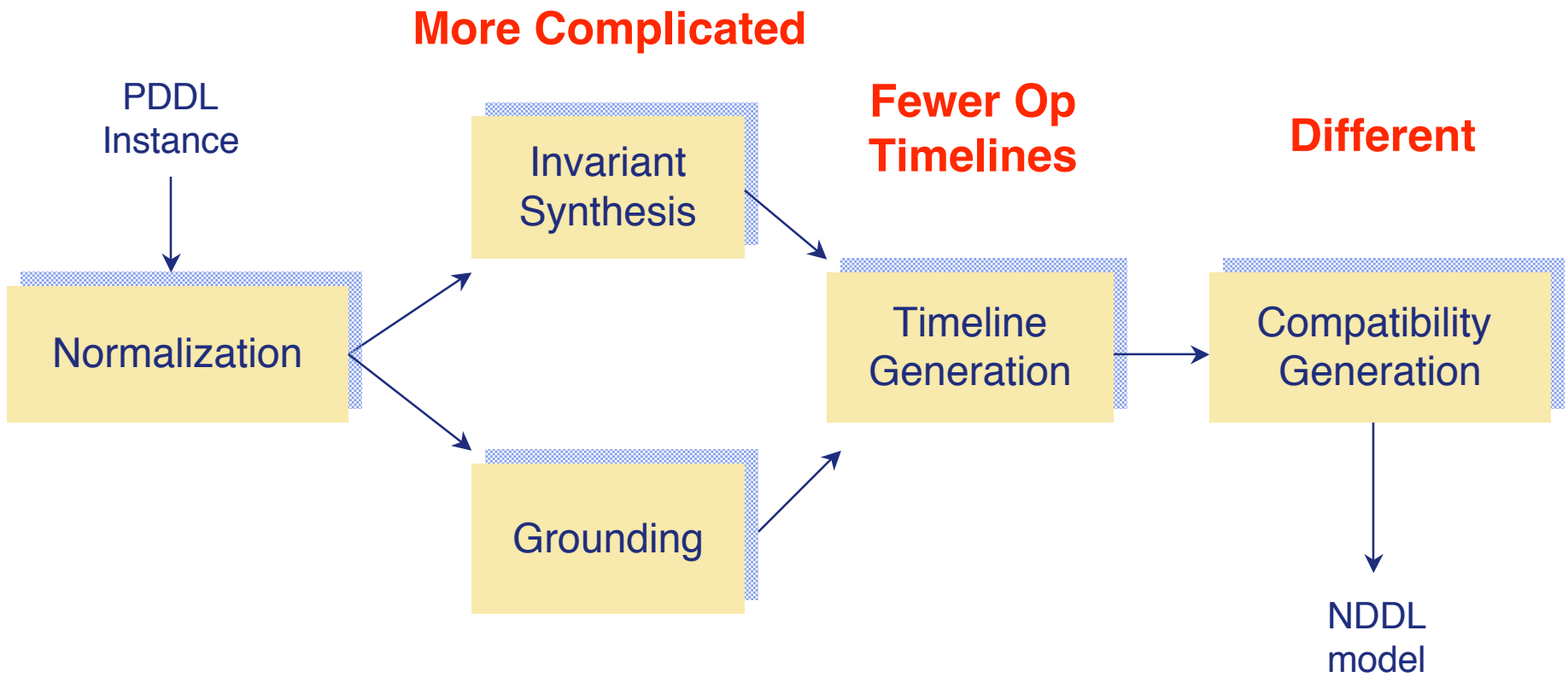
Ames Research Center

Breakdown (runtime > 1s)

[Helmert-2009]

Domain	Tasks	Parsing	Normalize	Invariants	Grounding	Generation
Airport	30/50	8-18%	1-4%	2-8%	48-64%	20-27%
Assembly	4/30	5-8%	< 1%	< 1%	80-84%	10-13%
Depot	7/22	4-6%	< 1%	< 1%	65-70%	26-30%
Driverlog	5/20	4-6%	< 1%	< 1%	69-73%	22-26%
Freecell	76/80	3-5%	< 1%	< 1%	67-71%	24-28%
Grid	4/5	4%	< 1%	< 1%	69-72%	24-27%
Logistics	21/63	3-6%	< 1%	< 1%	68-75%	21-30%
Miconic-full	95/150	1-5%	0-2%	< 1%	84-92%	7-10%
MPrime	23/35	3-6%	< 1%	< 1%	70-75%	22-27%
Mystery	14/30	3-6%	< 1%	< 1%	70-74%	22-27%
Openstacks	7/30	4-6%	< 1%	< 1%	63-65%	30-33%
OptTelegraph	41/48	4-7%	< 1%	< 1%	60-68%	25-35%
Pipes-NoTank	22/50	3-6%	< 1%	< 1%	71-75%	20-24%
Pipes-Tank	43/50	2-5%	< 1%	< 1%	71-76%	19-26%
PSR-Large	43/50	5-7%	< 1%	< 1%	80-87%	7-16%
PSR-Medium	27/50	5-8%	< 1%	< 1%	81-87%	8-14%
PSR-Small	2/50	29-34%	4%	1-11%	39-56%	11-13%
Rovers	22/40	2-7%	< 1%	< 1%	77-95%	4-17%
Satellite	18/36	1-6%	< 1%	< 1%	66-75%	20-34%
Storage	12/30	5-6%	< 1%	< 1%	64-67%	27-32%
TPP	15/30	3-5%	< 1%	< 1%	73-76%	20-24%
Trucks	20/30	5-9%	< 1%	< 1%	62-69%	25-33%
ZenoTravel	7/20	4%	< 1%	< 1%	73-76%	21-24%

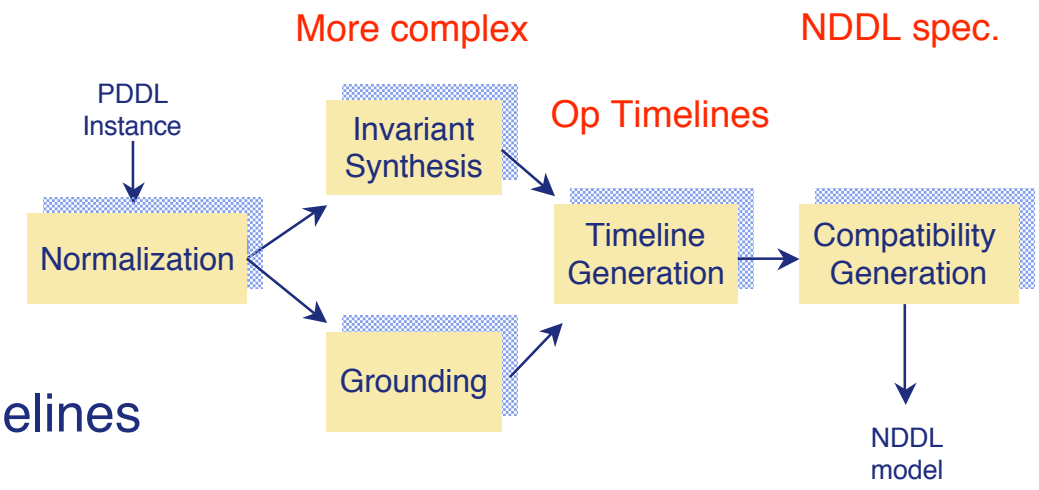
Differences for Temporal



Same Computational Complexity

Conclusions

1. Translation from PDDL2.1 into NDDL
   - ***Invariant Synthesis***
   - ***Timeline Generation***
   - ***Compatibility Generation***



2. Issues:
   - compacting operator timelines
   - numeric quantities
   - optimization criteria
   - timed initial literals (PDDL 2.2)
   - preferences (PDDL 3.0)

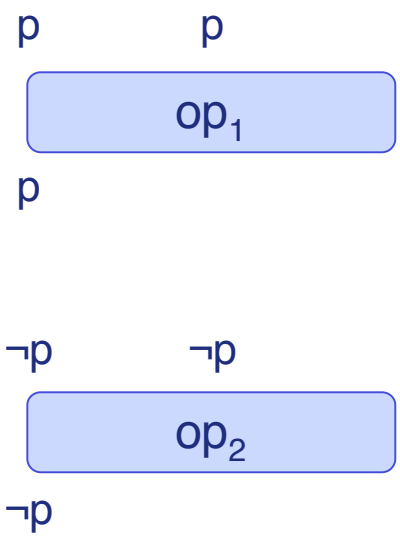
Extras

Mutex for Durative Actions

When are two durative PDDL operators ***mutex***?

Nine types of mutex:

1. **Start-Start**:
   op1 and op2 cannot start at the same time



Mutex for Durative Actions

Nine types of mutex:

1. **Start-Start**:
   op1 and op2 cannot start at the same time
2. **End-End:**
   op1 and op2 cannot end at the same time
3. **Start-End:**
   op1 cannot start at the same time that op2 ends
4. **Dur-Start:**
   op2 cannot start during op1
5. **Dur-End:**
   op2 cannot end during op1
6. **Dur-Dur:**
   op1 and op2 cannot overlap

Dual cases: End-Start (dual to 3), Start-Dur (dual to 4), and End-Dur (dual to 5)