Planning and Optimization

E14. Cartesian Abstractions: CEGAR

Malte Helmert and Gabriele Röger

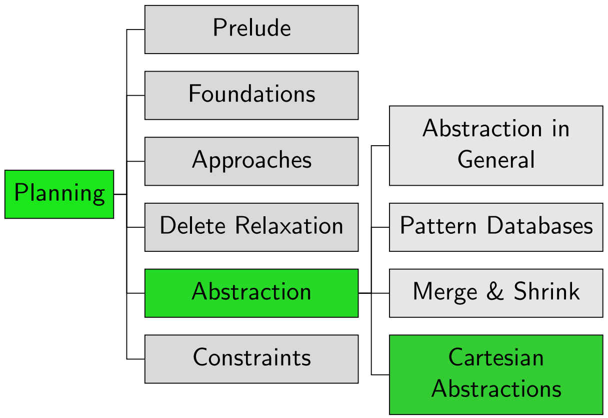
Universität Basel

November 24, 2025

Planning and Optimization

November 24, 2025 — E14. Cartesian Abstractions: CEGAR

Content of the Course



E14.1 CEGAR

Counterexample-Guided Abstraction Refinement

Counterexample-guided abstraction refinement (CEGAR) is an approach to compute a tailored abstraction for a task (or to solve it).

- Start with a very coarse abstraction.
- Iteratively compute an (optimal) abstract solution and check whether it works for the concrete tasks.
  - If yes, the task is solved.
  - If not, refine the abstraction so that the same flaw will not be encountered in future iterations.

CEGAR Algorithm

Generic CEGAR algorithm for planning task Π

$\mathcal{T} :=$ TrivialAbstractTransitionSystem(Π) $\leftarrow$ one abstract state
while not TerminationCondition(): $\leftarrow$ e.g. time/memory limit
    $\tau :=$ FindOptimalTrace($\mathcal{T}$) $\leftarrow$ abstract solution (path in $\mathcal{T}$)
    if τ is “no trace” then **return** Π unsolvable
    $F :=$ FindFlaw($\tau, \Pi, \mathcal{T}$)
    if F is “no flaw” then
        return label sequence of τ as plan for Π
    $\mathcal{T} :=$ Refine($\mathcal{T}, F$)
return $\mathcal{T}$

Open questions:

- What are flaws (and how to find them)? $\rightsquigarrow$ next
- How do we refine the system?

E14.2 Flaws

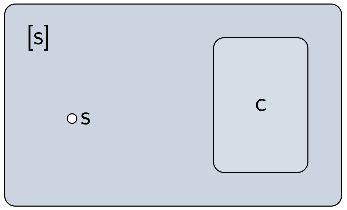
Flaws

A flaw is a reason why (the label sequence of) τ does not solve Π the way it solves the abstract system $\mathcal{T}$ (with abstraction α).

Start from the initial state of Π and iteratively apply the next operator (label) o from τ.

- Precondition flaw: o is not applicable in the current state s.
- Goal flaw: the final state is not a goal state.
- Deviation flaw: the next abstract transition is $a \xrightarrow{o} a'$, the current concrete state is s with $\alpha(s) = a$ but for successor state $s' = s[\![o]\!]$ we have $\alpha(s') \neq a'$ (deviating from the abstract path).

Extracting Flaws



For the refinement, we represent flaws in the form $\langle s, c \rangle$, where

- s is a concrete state,
- $c \subseteq [s]$ is a non-empty Cartesian set,
- the abstract plan relied on "being in c" but $s \notin c$.

$\langle s, c \rangle$ will define the split for the refinement step.

Extracting Different Kinds of Flaws

- Precondition flaw: if o is not applicable in state s, use $\langle s, c \rangle$, where c is the set of concrete states in $[s]$ in which o is applicable.
- Goal flaw: if the final state s is not a goal state, use $\langle s, c \rangle$, where c is the set of concrete goal states in $[s]$.
- Deviation flaw: the next abstract transition is $a \xrightarrow{o} a'$, the current concrete state is s with $\alpha(s) = a$ but for successor state $s' = s[\![o]\!]$ we have $\alpha(s') \neq a'$ (deviating from the abstract path). Use (s, c), where c is the intersection of $[s]$ and $regr(a', o)$.

Easy for Cartesian abstractions, using the results from Ch. E13.

E14.3 Refinement

CEGAR Algorithm

Generic CEGAR algorithm for planning task Π

$\mathcal{T}$:= TrivialAbstractTransitionSystem(Π)
while not TerminationCondition():
    τ := FindOptimalTrace($\mathcal{T}$)
    if τ is "no trace" then **return** Π unsolvable
    F := FindFlaw($\tau, \Pi, \mathcal{T}$)
    if F is "no flaw" then
        return label sequence of τ as plan for Π
    $\mathcal{T}$:= Refine($\mathcal{T}, F$)
return $\mathcal{T}$

Open questions:

- How do we refine the system?

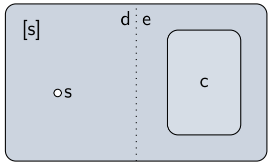
Refinement

Refinement splits abstract state $[s]$ and maintains the transition system induced by the underlying abstraction.

Refine($\langle S', L', c', T', s'_0, S'_\star \rangle$, $\langle s, c \rangle$)

$\langle d, e \rangle := \text{Split}([s], s, c)$
$S'' := S' \setminus \{[s]\} \cup \{d, e\}$
$T'' := \text{RewireTransitions}(T', [s], d, e)$
if $[s] = s'_0$ then $s''_0 := d$ else $s''_0 := s'_0$
if $[s] \in S'_\star$ then $S''_\star := (S''_\star \setminus \{[s]\}) \cup \{e\}$ else $S''_\star := S'_\star$
return $\langle S'', L', c', T'', s''_0, S''_\star \rangle$

Split $[s]$ into d and e.



Refinement

Refinement splits abstract state $[s]$ and maintains the transition system induced by the underlying abstraction.

Refine($\langle S', L', c', T', s'_0, S'_\star \rangle$, $\langle s, c \rangle$)

$\langle d, e \rangle := \text{Split}([s], s, c)$
$S'' := S' \setminus \{[s]\} \cup \{d, e\}$
$T'' := \text{RewireTransitions}(T', [s], d, e)$
if $[s] = s'_0$ then $s''_0 := d$ else $s''_0 := s'_0$
if $[s] \in S'_\star$ then $S''_\star := (S''_\star \setminus \{[s]\}) \cup \{e\}$ else $S''_\star := S'_\star$
return $\langle S'', L', c', T'', s''_0, S''_\star \rangle$

Update incident transitions of $[s]$.

- Check for each incoming and outgoing transition of $[s]$ (including self-loops) whether it needs to be rewired from/to d, from/to e, or both.
- Easy for SAS$^+$ operators and Cartesian abstract states.

Refinement

Refinement splits abstract state $[s]$ and maintains the transition system induced by the underlying abstraction.

Refine($\langle S', L', c', T', s'_0, S'_\star \rangle$, $\langle s, c \rangle$)

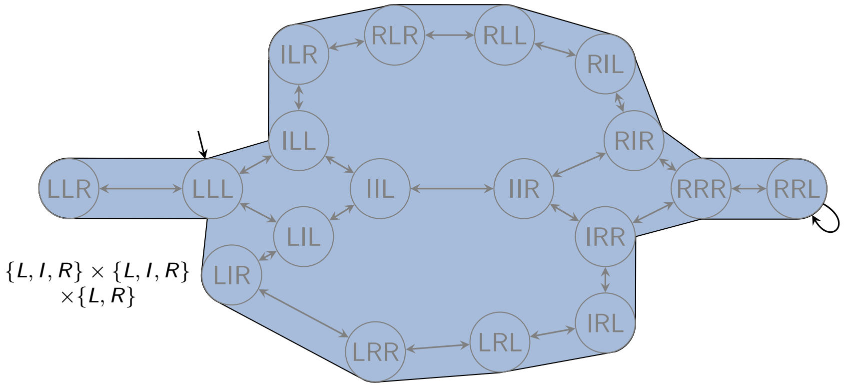
$\langle d, e \rangle := \text{Split}([s], s, c)$
$S'' := S' \setminus \{[s]\} \cup \{d, e\}$
$T'' := \text{RewireTransitions}(T', [s], d, e)$
if $[s] = s'_0$ then $s''_0 := d$ else $s''_0 := s'_0$
if $[s] \in S'_\star$ then $S''_\star := (S''_\star \setminus \{[s]\}) \cup \{e\}$ else $S''_\star := S'_\star$
return $\langle S'', L', c', T'', s''_0, S''_\star \rangle$

Update abstract initial state and goal states.

The way we defined the flaws, e can never be the abstract initial state and d never be an abstract goal state.

E14.4 Example

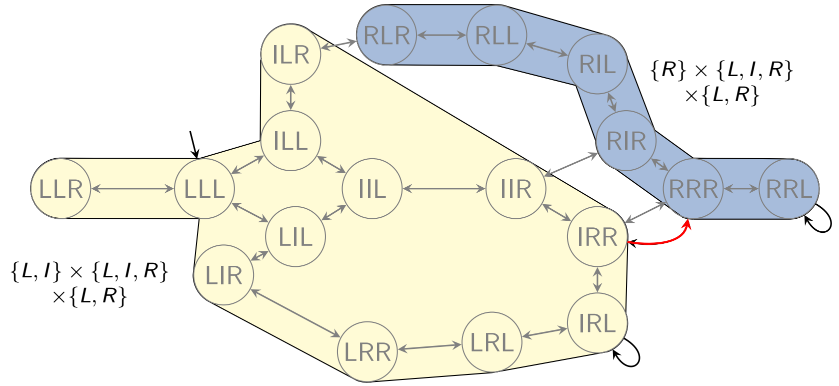
Example: Two Packages, One Truck



Abstract plan $\langle\rangle$ ends in state LLL, which is not a goal.
Refine $\{L, I, R\} \times \{L, I, R\} \times \{L, R\}$ with split $(LLL, \{R\} \times \{R\} \times \{L, R\})$.

$\rightsquigarrow$ split on first or second variable;
$\rightsquigarrow$ $\{L, I\} \times \{L, I, R\} \times \{L, R\}$ and $\{R\} \times \{L, I, R\} \times \{L, R\}$

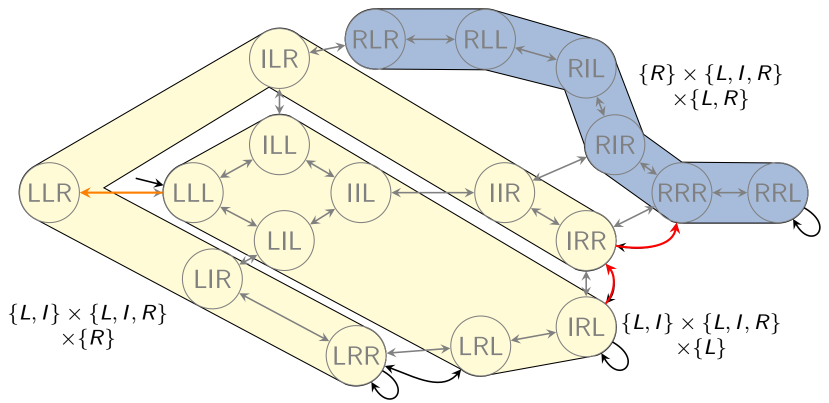
Example: Two Packages, One Truck



Abstract plan $\langle \text{drop}_{A,R} \rangle$; first action inapplicable in LLL.
Refine $\{L, I\} \times \{L, I, R\} \times \{L, R\}$ with split $(LLL, \{I\} \times \{L, I, R\} \times \{R\})$.

$\rightsquigarrow$ split on first or third variable;
$\rightsquigarrow$ $\{L, I\} \times \{L, I, R\} \times \{L\}$ and $\{L, I\} \times \{L, I, R\} \times \{R\}$

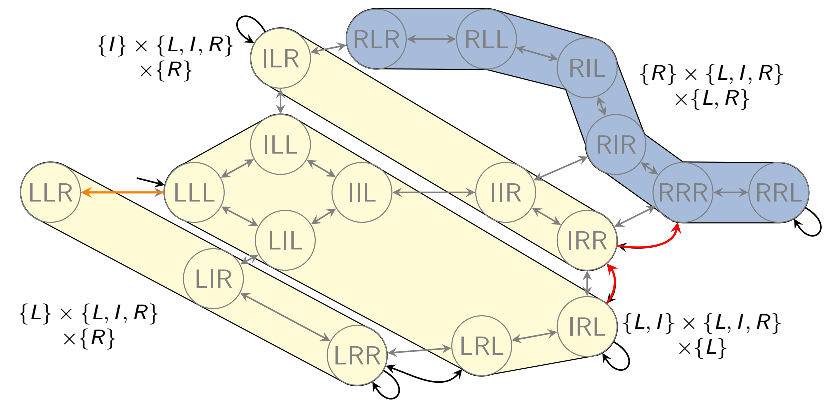
Example: Two Packages, One Truck



Abstract plan $\langle \text{move}_{L,R}, \text{drop}_{A,R} \rangle$; second action inapplicable in LLR.
Refine $\{L, I\} \times \{L, I, R\} \times \{R\}$ with split $(LLR, \{I\} \times \{L, I, R\} \times \{R\})$.

$\rightsquigarrow$ split on first variable;
$\rightsquigarrow$ $\{L\} \times \{L, I, R\} \times \{R\}$ and $\{I\} \times \{L, I, R\} \times \{R\}$

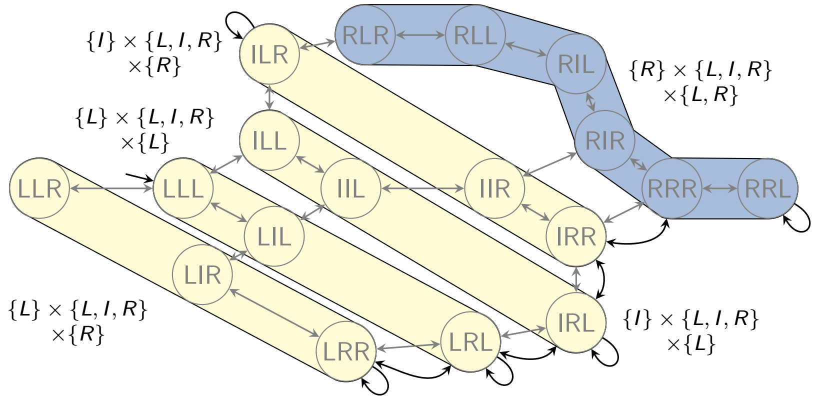
Example: Two Packages, One Truck



Abstract plan $\langle \text{move}_{L,R}, \text{drop}_{A,R} \rangle$; deviation flaw at first transition.
Refine $\{L, I\} \times \{L, I, R\} \times \{L\}$ with split $(LLL, \{I\} \times \{L, I, R\} \times \{L\})$.

$\rightsquigarrow$ split on first variable;
$\rightsquigarrow$ $\{L\} \times \{L, I, R\} \times \{L\}$ and $\{I\} \times \{L, I, R\} \times \{L\}$

Example: Two Packages, One Truck



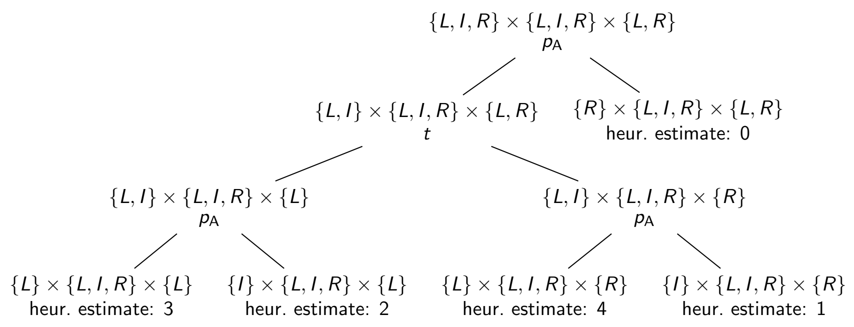
...

E14.5 Heuristic Representation

Representation

- In every iteration, we split one abstract state based on one variable.
- Represent abstraction as binary tree of abstract states.
  - Root: Single state of trivial abstraction
  - Leaves: Abstract states of final abstraction
- With each inner node, we store the variable on which the state was split.

Representation: Running Example



E14.6 Summary

Summary

Counterexample-guided abstraction refinement (CEGAR):

- Iteratively improve a coarse abstraction:
  - Find an optimal abstract solution.
  - Try it in the concrete transition system.
  - If it fails, extract a flaw and refine the abstraction.
- Flaws: unsatisfied precondition, unsatisfied goal, deviation.
- Refinement: split abstract state based on flaw to avoid repeating it.
- Can be efficiently implemented for Cartesian abstractions.
- Can stop at any time. The resulting heuristic is safe, goal-aware, admissible and consistent.