Hierarchical CnC Specifications

CnC Workshop, October 14th, 2017

Milind Kulkarni (Purdue University)
Kath Knobe (Rice University)
Zoran Budimlić (Rice University)

CnC today

- Motivated by *separation of concerns*
  - Programmer provides a CnC specification, specifying exactly what data and control dependences exist in a program
  - (Potentially) different programmer provides a tuning specification to impact mapping and scheduling decisions
  - (Definitely) different runtime system responsible for enforcing the dependences at runtime and using the tuning specification to determine how to execute the program
- Key: algorithm designer does not have to worry about anything other than the dependence structure of their code

CnC today

- But not all is perfect!
- One key decision algorithm writer still has to make: *granularity*
  - How to break program up into steps and data/tag collections
  - How coarse/fine-grained computation is going to be *(how much work done per CnC step)*
  - How coarse/fine-grained data is going to be *(how much data in each data item)*
- Decided at the CnC spec level, and not something visible/controllable at either the tuning spec or the runtime level

granularity concerns

- Choosing the right granularity has effects for performance, and there's no "right" answer:
  - Coarser:

    + less scheduling overhead

    - less parallelism, too much data/computation for certain computational resources, less scheduling flexibility,
  - Finer:

    + but maximum flexibility and parallelism

    - more scheduling overhead
- Right granularity extremely dependent on target platform, input, etc.

prior takes on this problem

Rewrite CnC programs (i.e., compiler-based approaches)

+ Perform polyhedral tiling on CnC programs to adjust data/step granularity for regular programs

+ Perform “graph rewrites” that fuse CnC steps and data collections for irregular programs

- Constrained by what the compiler is able to figure out

- No help when the best choice is only known dynamically.

  Depends on input data, runtime state, dynamic hardware environment

prior takes on this problem

Defer to tuning component

+ Hierarchical affinity groups provide scheduler hints to “group” various computation steps together in time and on the platform

+ More dynamic than compiler approaches, but:

- Groups are “hints” only – no guarantees
- Ultimate granularity of execution is still fine-grained: still incurs runtime overhead, does not include coarse-grained optimizations (e.g., restructuring a coarse-grain step for better execution on GPU)

what we propose

- A *hierarchy spec* for CnC programs
  - Captures a range of granularity choices
  - Specific choices leads to a specific grain of execution
- Choices made either at compile time or at runtime
  - Can provide different step implementations for different granularity choices
- Guarantee: same results regardless of granularity choices made

what this talk is

- A discussion of:
  - hierarchy specs for CnC—what are they, how do they work
  - semantics of CnC hierarchy
  - why hierarchy specs might be useful

what this talk isn't

- A discussion of implementation
- Just ideas at the moment, we haven't implemented them!

adding hierarchy to CnC

- Starting point: a program written at a fine level of granularity
- Each step represents a small, atomic computation on a small piece of data

adding hierarchy to CnC

- Can coarsen granularity by “merging” two differently named steps together
- foobar does the work of foo and bar, but is now conceptually a single step (and obeys step semantics)

adding hierarchy to CnC

- Can also coarsen granularity by merging tag instances together
- Conceptually, steps now operate on entire rows *i*, rather than individual elements *i, j*

different valid specs

- But there may be many different choices for coarsening

different valid specs

- There are many different ways that you could coarsen
- Could merge bar and baz together

different valid specs

- There are many different ways that you could coarsen
- Could coarsen indices for collection for baz and foo, but not for foo

different valid specs

- But there are many different ways that you could coarsen
- Could merge foo and bar and coarsen indices for baz
  - Dozens of variations!
  - Note: At this point we pick one and don't see the others

Attribute propagation

- Classic flat CnC: attributes describe the static and dynamic meaning of an application
- Hierarchical CnC: extend these rules
- Issue:
  - step becomes *executed*

    is a primitive at the bottom level
  - higher level step becomes *executed*

    First requires that its high level control is *available.*
  - issue: the high level control might not be statically known. But rules propagate this from the input *available*

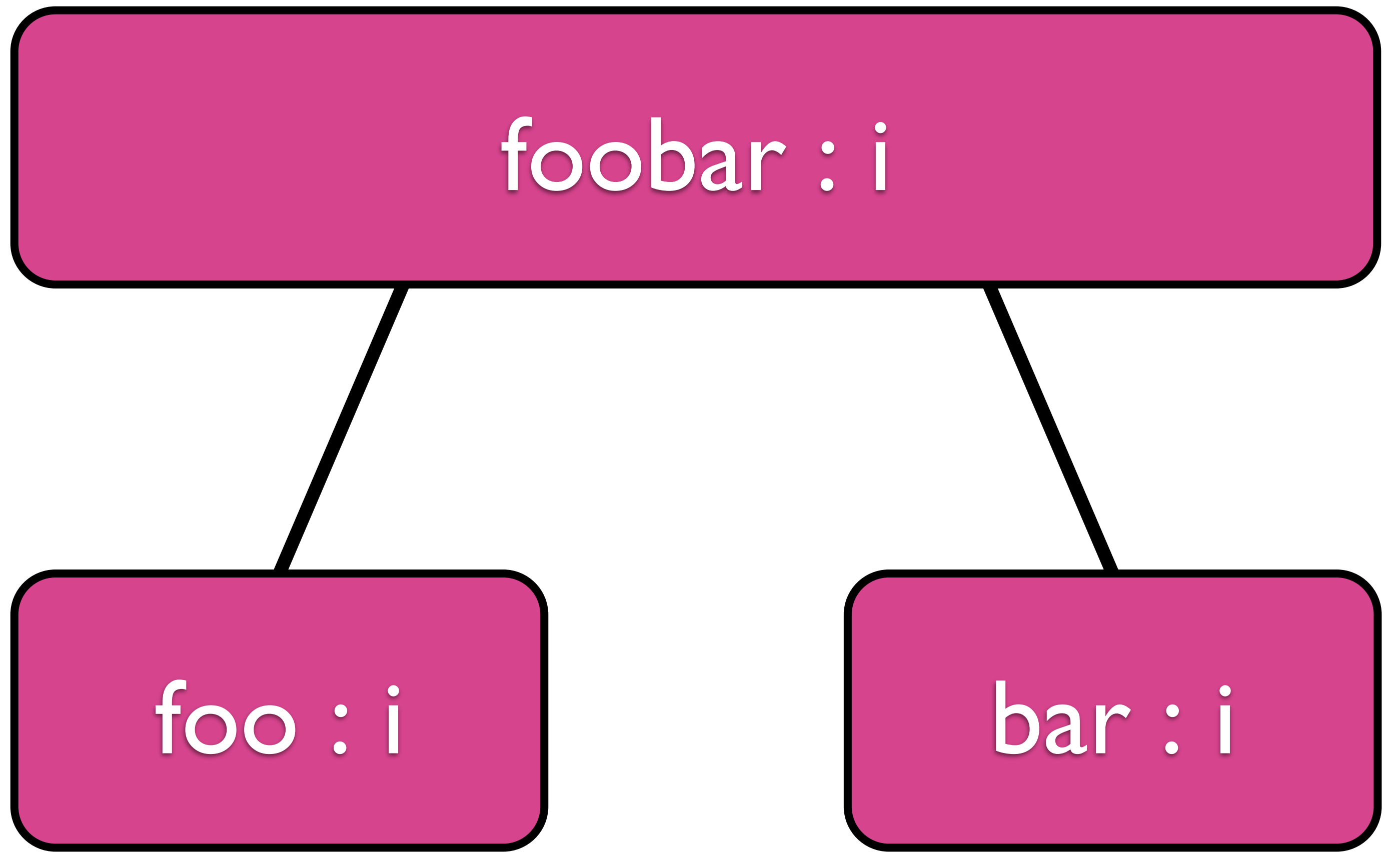
many possible hierarchical specs

- Choice of one hierarchical spec is a decision, not a representation of options
- Constraint hierarchy: a hierarchical CnC spec that captures a range of different valid, but equivalent, CnC specs
- First thought: just specify different valid specs
    - Combinatorial explosion in number of valid specs, so this isn't practical

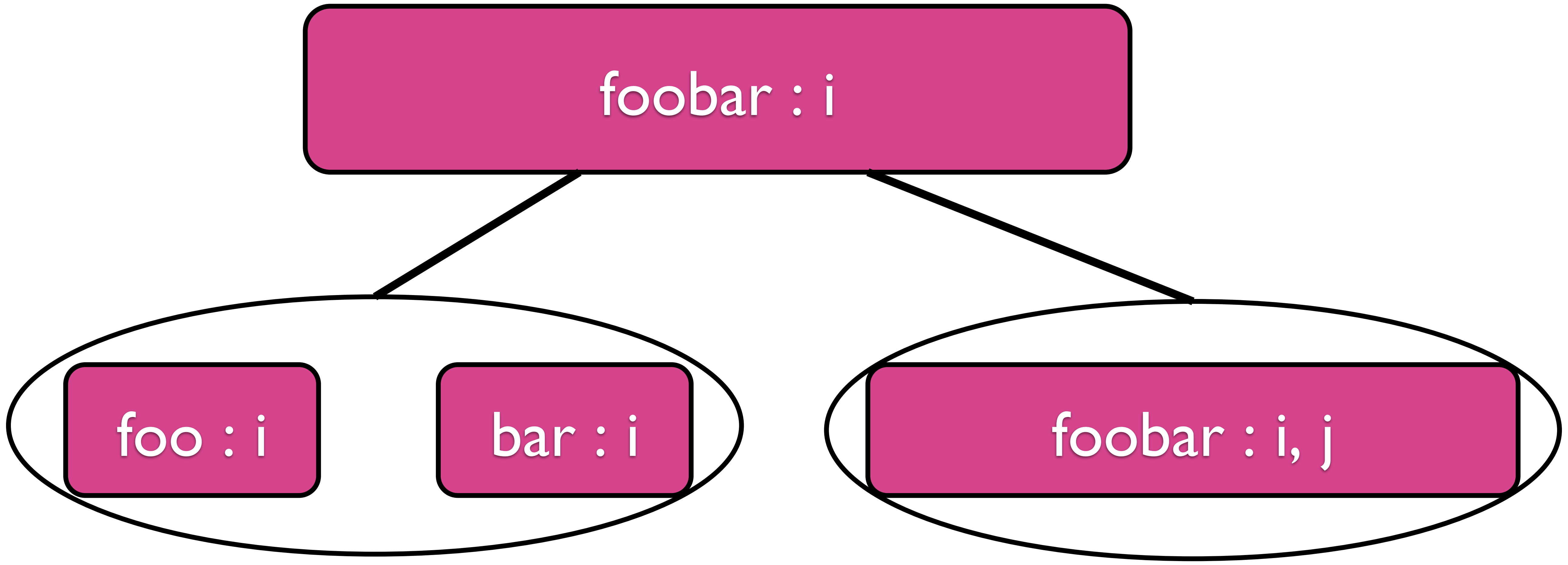
hierarchy specs

Key insight: two different key "operations" that govern different granularity decisions:

"and" decomposition shows how a coarse-grain step can be broken down into multiple fine-grain steps



"or" decomposition offers a choice of multiple decomposition options (basic configuration: "or" choice of multiple "and" decompositions)



DAG

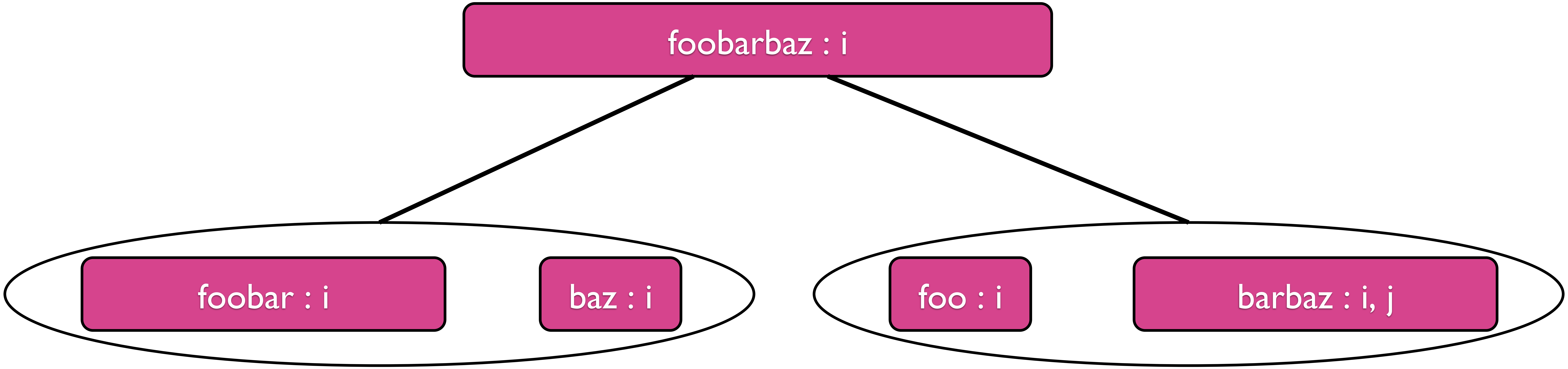
- This process may expand the tree by duplicating lower level subtrees under multiple branches of the higher level tree.
- Create a DAG
  - Has no impact on the meaning
  - Significantly shrinks the spec
  - Makes it more understandable

generating a CnC spec

- Can generate a flat CnC spec from a hierarchy spec by making a “cut” in the hierarchy spec DAG
- Must satisfy “or” and “and” hierarchy constraints as discussed next

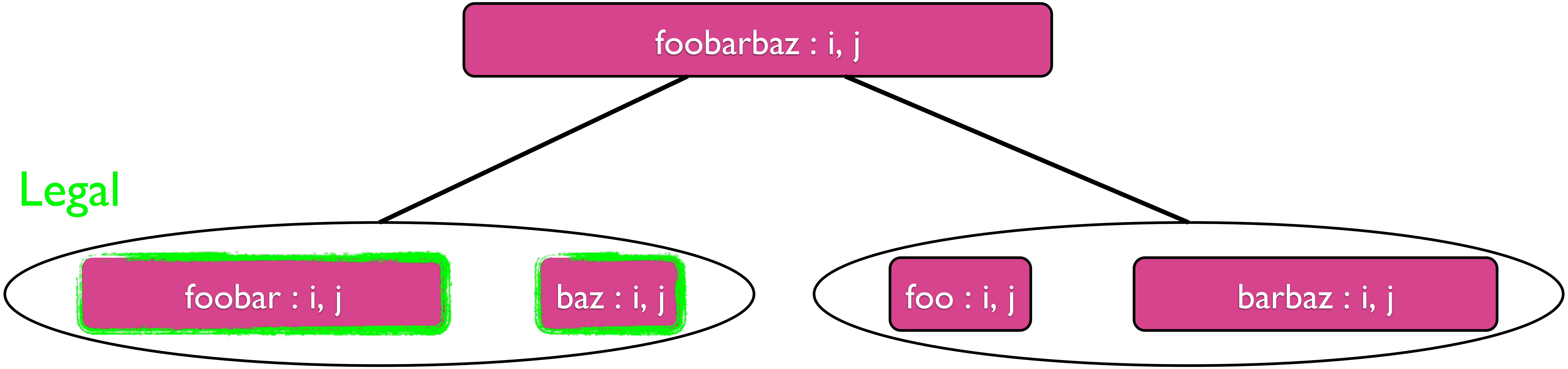
satisfying “or” constraints

- For any given “or” constraint, choose the whole computation from one branch



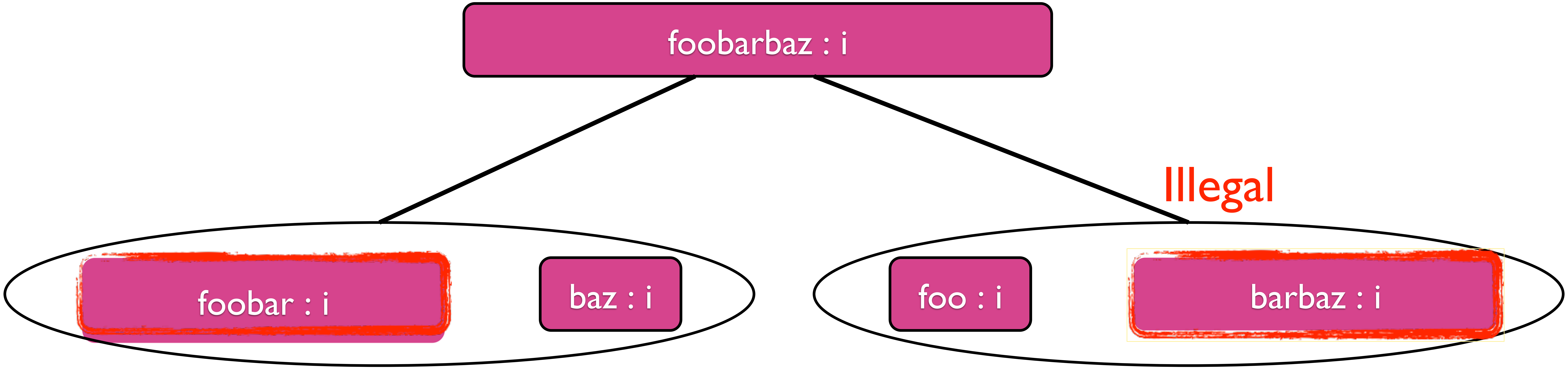
satisfying “or” constraints

- For any given “or” constraint, choose the whole computation from one branch



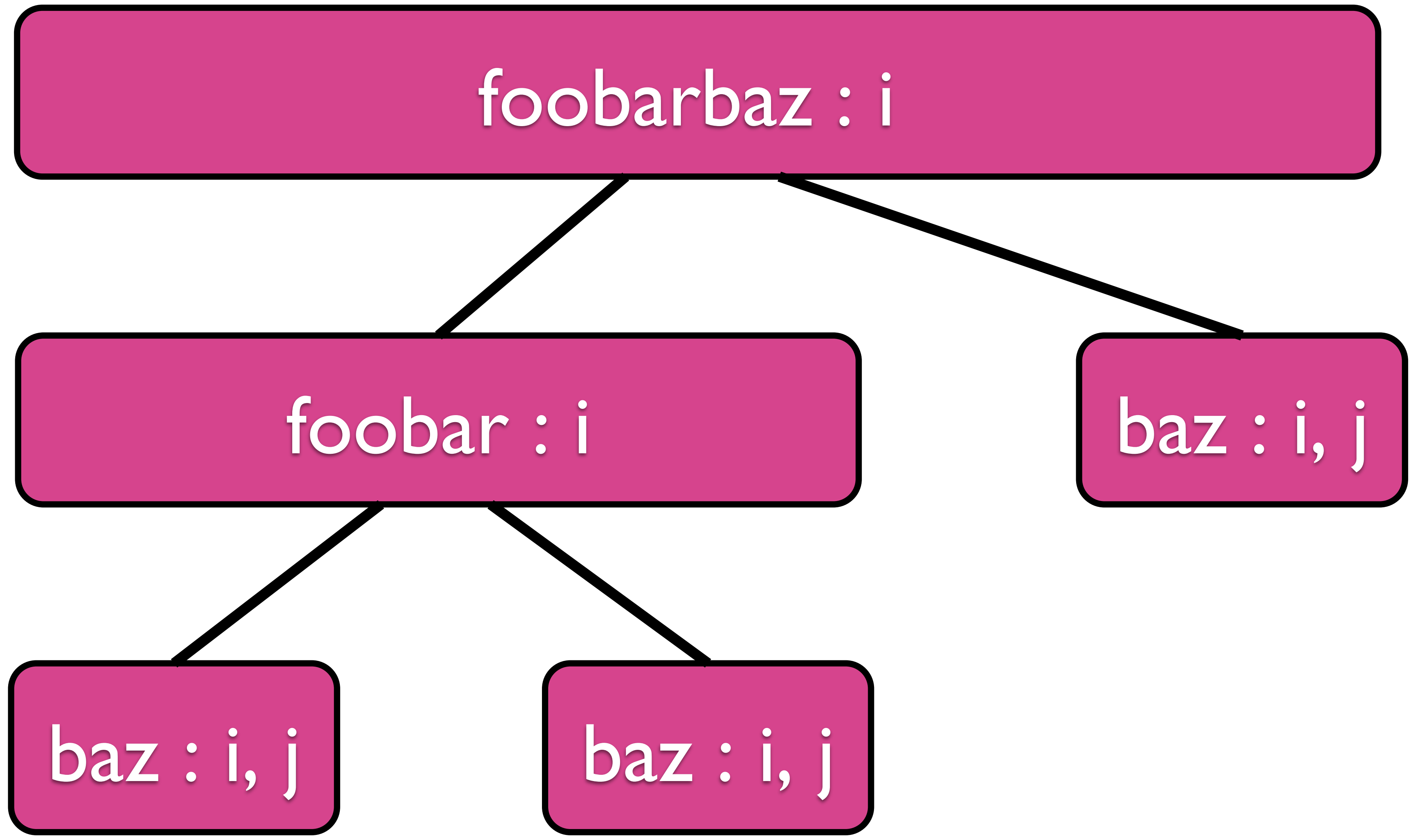
satisfying "or" constraints

- For any given "or" constraint, choose the whole computation from one branch



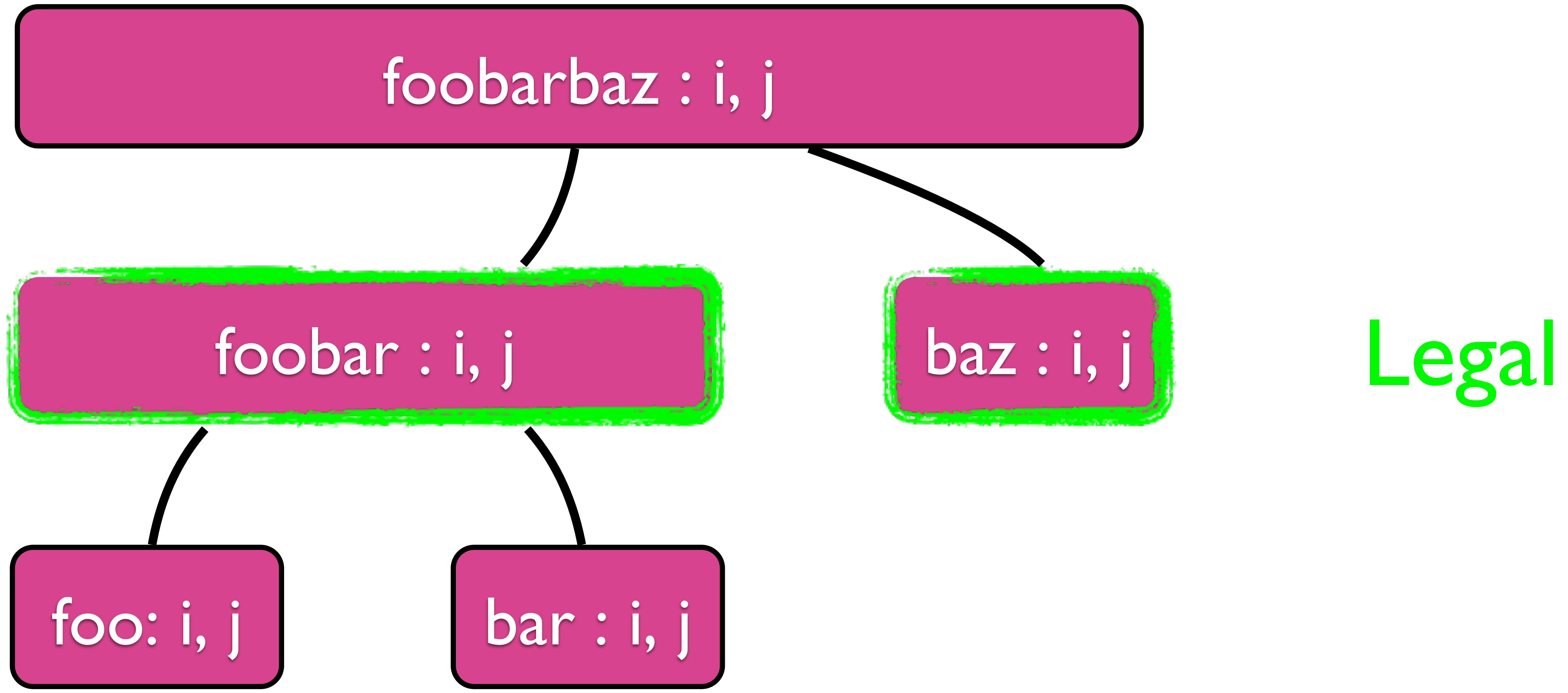
satisfying "and" constraints

- Satisfy "and" constraints by making a cut in the tree through the "and" nodes



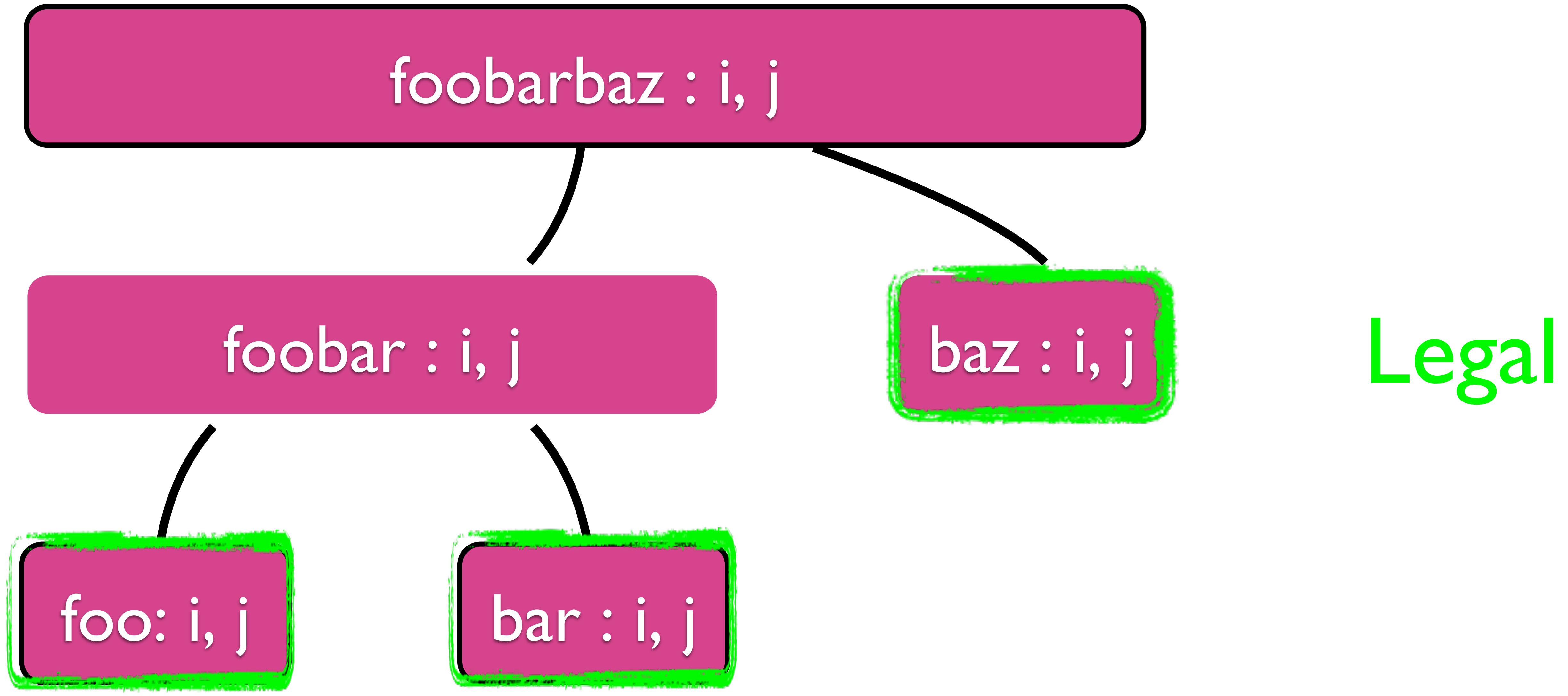
satisfying “and” constraints

- Satisfy “and” constraints by making a cut in the tree through the “and” nodes



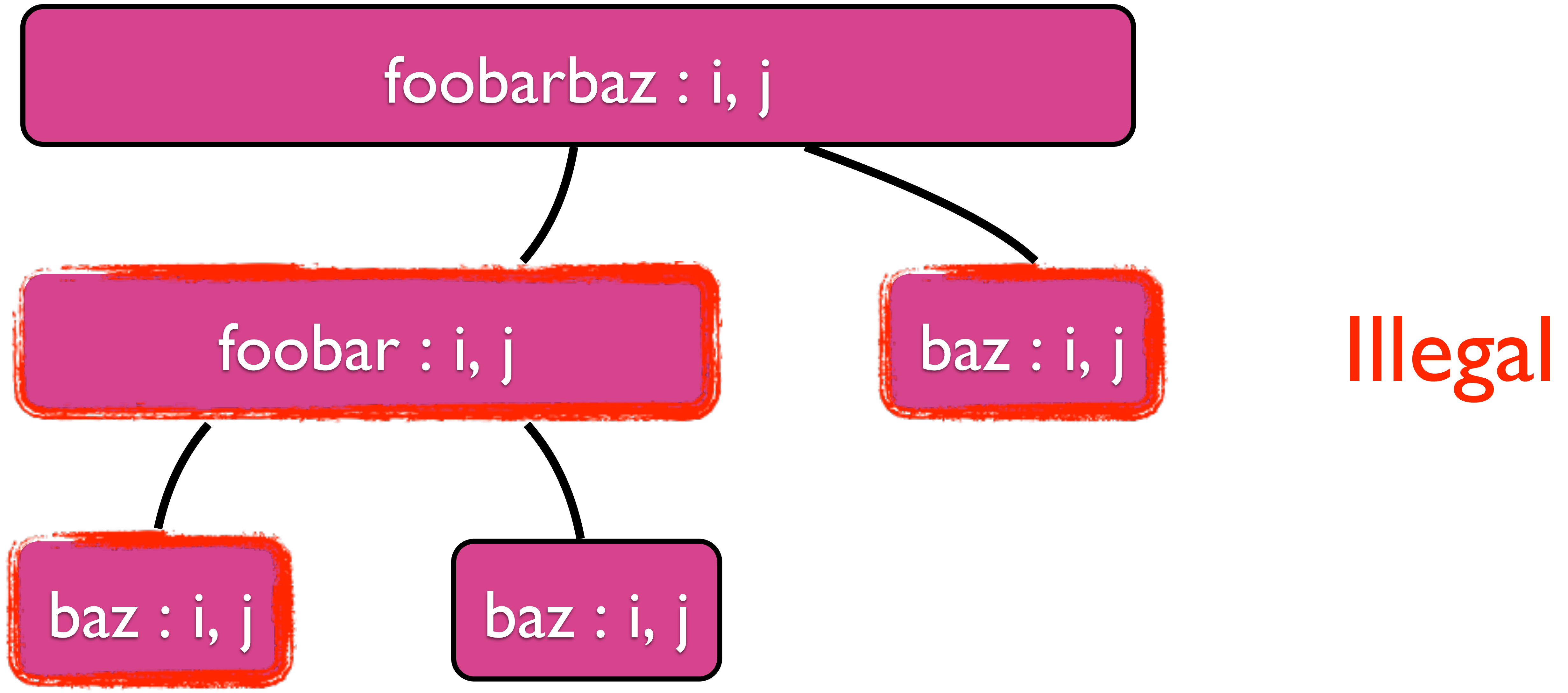
satisfying “and” constraints

- Satisfy “and” constraints by making a cut in the tree through the “and” nodes



satisfying “and” constraints

- Satisfy “and” constraints by making a cut in the tree through the “and” nodes



CnC semantics

- When a specific hierarchy is chosen from the constraints on the hierarchy, we have a single unconstrained hierarchical CnC program
- This can occur at compile time or runtime or during a single unified unordered process.
- The rules for attribute propagation on a given hierarchy apply

conclusions

- A hierarchical CnC spec specifies an application at a variety of equivalent grains

  - An appropriate grain might be choosen statically (by the tuning expert or a compiler) or dynamically (by the runtime)

- The constraints on hierarchy specifies all legal hierarchies

  - An appropriate hierarchically might be choosen statically (by the tuning expert or a compiler) or dynamically (by the runtime)

Future work

- Work out more details
- Implement
- Assess
- Repeat