Controller Synthesis Using Uninterpreted Functions

Overview of a work in progress

Georg Hofferek and Roderick Bloem

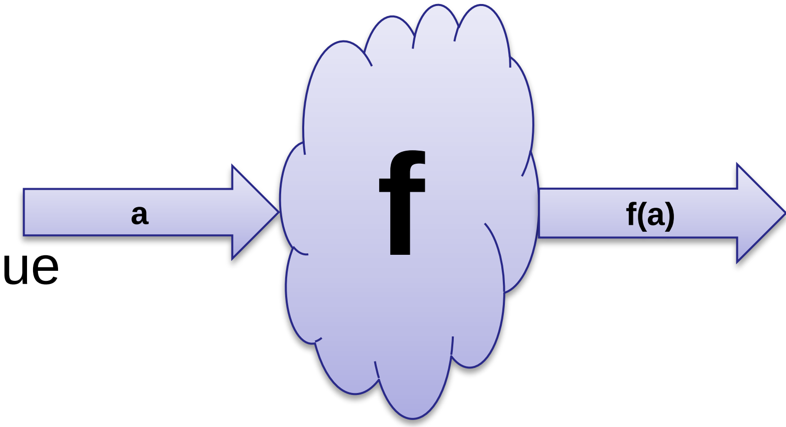
IAIK – Graz University of Technology, Austria
georg.hofferek@iaik.tugraz.at

What is an **Uninterpreted Function**?

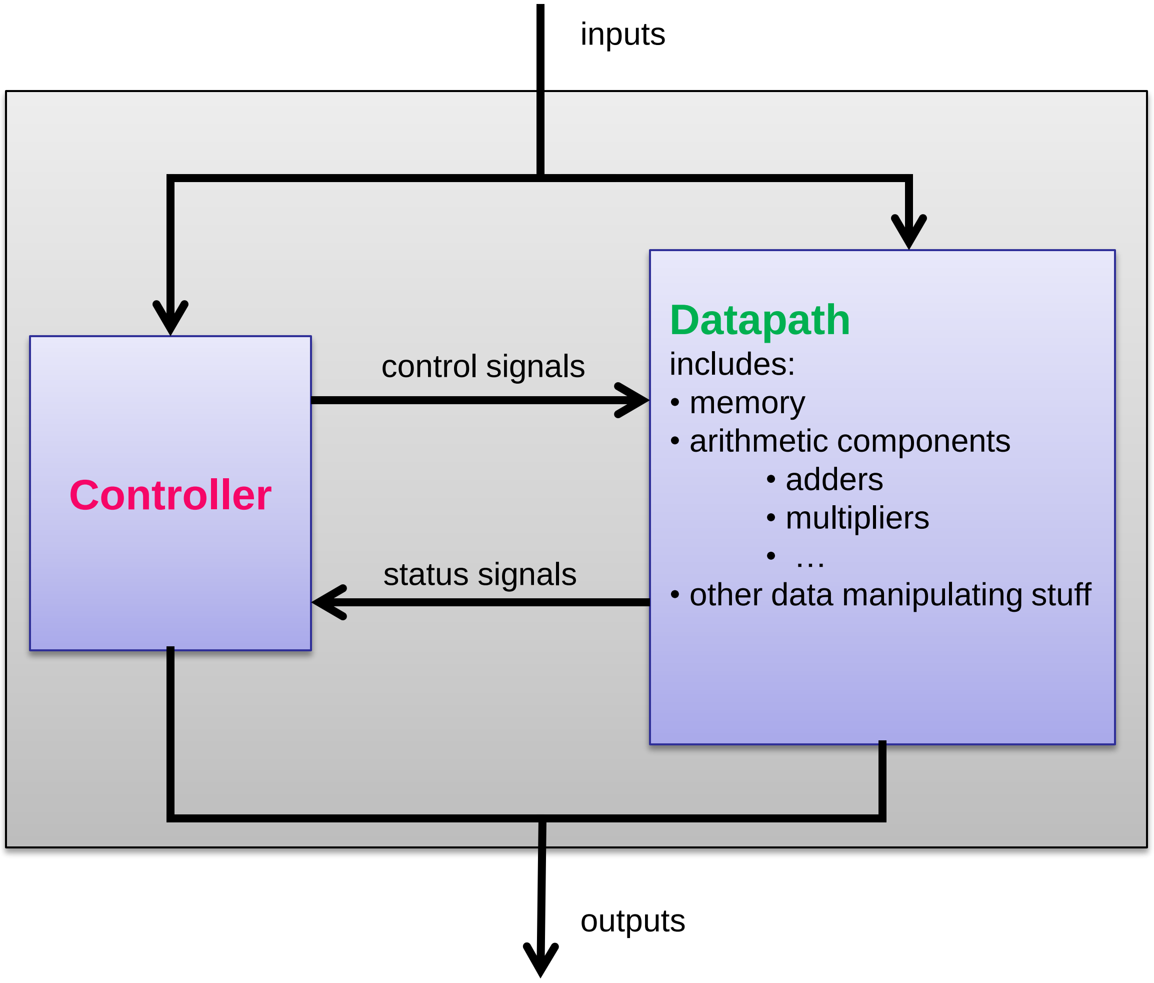
- A **function…** (obviously) ☺
  - Possibly n-ary
  - Mapping input value(s) to output value
- … which is **uninterpreted.**
  - i.e., we do not know/care about its “internals”
- But: **functional consistency**
  - $(a = b) \implies f(a) = f(b)$

for n-ary function:

$$\left(\bigwedge_{i \in \{1,\ldots,n\}} a_i = b_i \right) \implies f\left(a_1, a_2, \ldots, a_n\right) = f\left(b_1, b_2, \ldots, b_n\right)$$



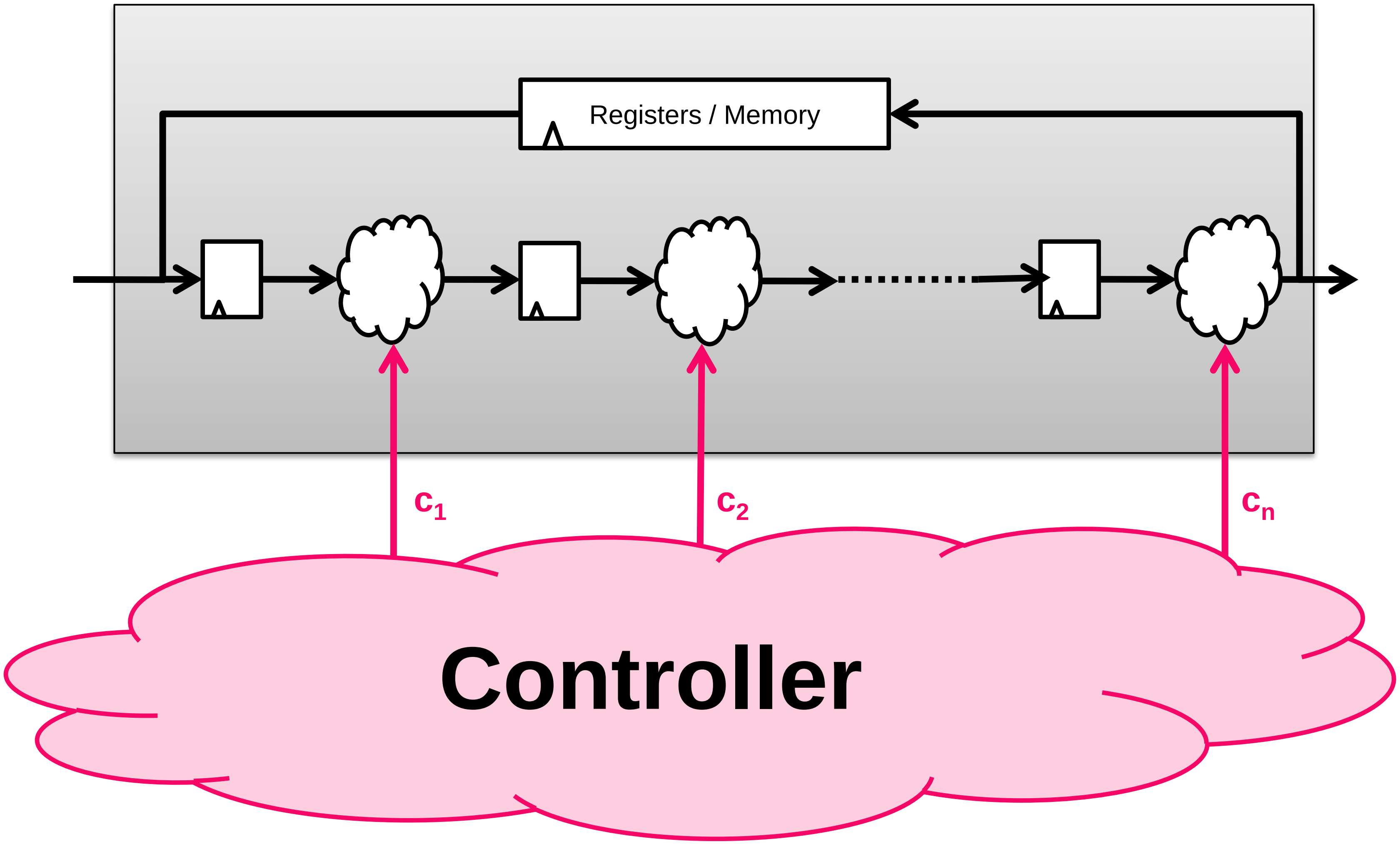
What is a **controller**?



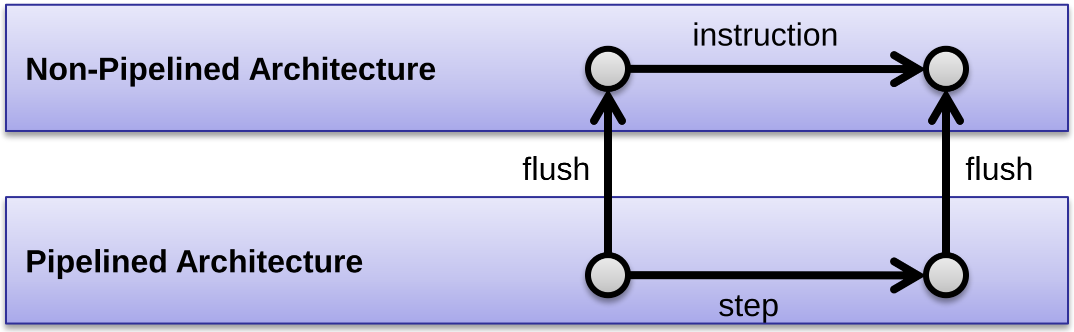
Controller versus **Datapath** are like:

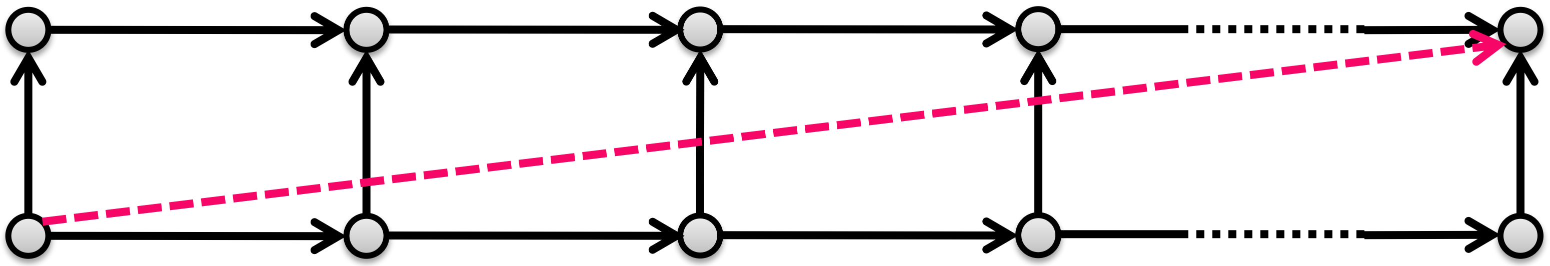
- **Driver** versus **Car**
- **Musician** versus **Piano**
- …

Motivation: **Pipelined Microprocessor**



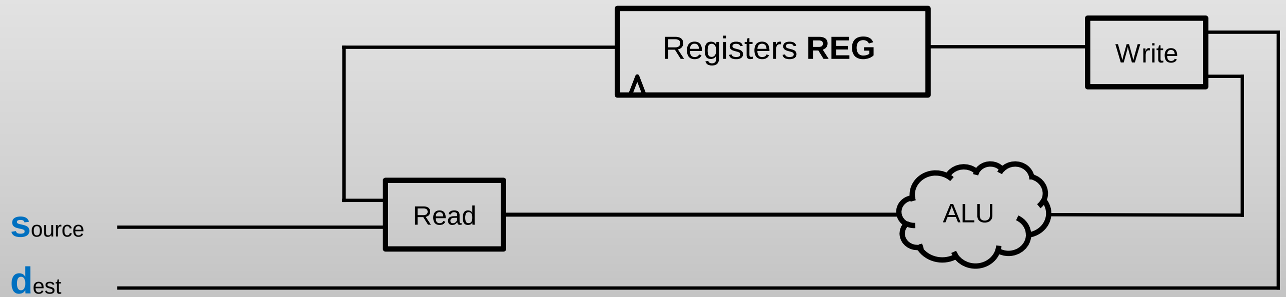
Equivalence: Commutativity



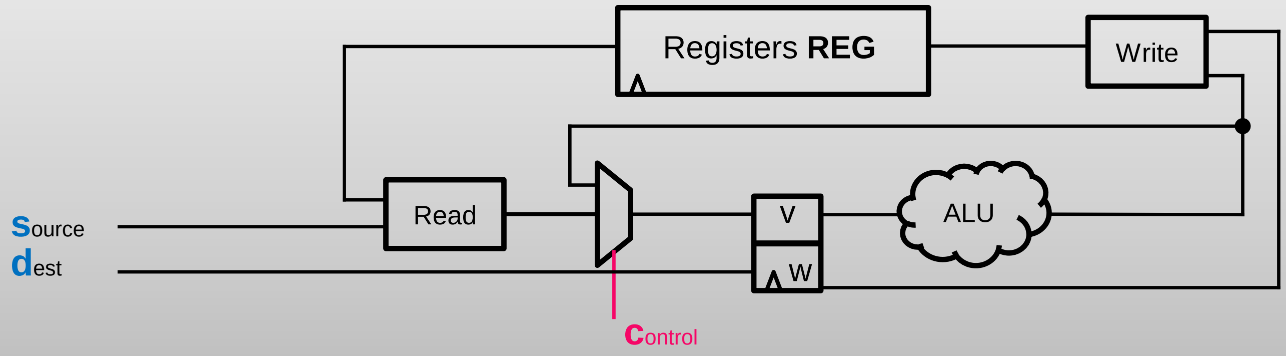


(Very) Simple Example

Non-pipelined Architecture (=reference):



Pipelined Architecture:



Synthesis Approach

- Define equivalence criterion: φ
- Claim: $\forall \bar{R} \,.\, \forall \bar{f} \,.\, \forall \bar{i}, \bar{s} \,.\, \exists \bar{c} \,.\, \exists \bar{R}' \,.\, \exists \bar{s}' \,.\, \varphi$
- **Reads:** *"For all (initial) array contents, for all interpretations of the functions, and for all inputs and initial states, there are control values, and resulting new array contents and next states, such that the equivalence criterion evaluates to true."*
- If the claim is valid, extract $\bar{c} \triangleq \bar{c}(\bar{f}, \bar{i}, \bar{s})$

Example: Equivalence Criterion

complete – ISA:

$$\begin{array}{lcll}
\varphi_{cI} & = & \left.\left[REG'_{cI} \Leftarrow REG\{w \leftarrow ALU(v)\}\right] \wedge\right\} & \text{complete} \\
 & & \left.\left[REG''_{cI} \Leftarrow REG'_{cI}\{d \leftarrow ALU(REG'_{cI}[s])\}\right]\right\} & \text{ISA}
\end{array}$$

step – complete:

$$\begin{array}{lcll}
\varphi_{sc} & = & \left.\begin{array}{l}\left[REG'_{sc} \Leftarrow REG\{w \leftarrow ALU(v)\}\right] \wedge \\ (w' = d) \wedge \left[v' = (c\,?\,ALU(v) \,:\, REG[s]\right] \wedge\end{array}\right\} & \text{step} \\
 & & \left.\left[REG''_{sc} \Leftarrow REG'_{sc}\{w' \leftarrow v')\}\right]\right\} & \text{complete}
\end{array}$$

Equivalence criterion:

$$\varphi \Leftrightarrow \varphi_{cI} \wedge \varphi_{sc} \wedge \forall i\,.\left(REG''_{cI}[i] = REG''_{sc}[i]\right)$$

Transformations

- Equivalence criterion is a first-order formula, using the theories of
  - Arrays (A)
  - Uninterpreted Functions (U)
  - Equality (E)

- Three reductions/transformations:
  - A-U-E → U-E 🙂 (proof done)
  - U-E → E 🙂 (proof done)
  - E → Propositional Logic 😟 (proof in progress)

A-U-E → U-E

1. Replace Array-Writes with fresh variables and apply **write axiom**
   $A\{i \leftarrow v\} \quad \rightsquigarrow \quad A' : \big(\forall j \neq i \,.\, A'[j] = A[j]\big) \wedge A'[i] = v$
2. Replace existential quantifications with fresh variables
3. Replace universal quantifications with conjunction over index set
4. Replace Array-Reads with uninterpreted functions $A[i] \rightsquigarrow A(i)$

$$\forall \bar{R} \,.\, \forall \bar{f} \,.\, \forall \bar{i}, \bar{s} \,.\, \exists \bar{c} \,.\, \exists \bar{R}' \,.\, \exists \bar{s}' \,.\, \varphi_{A,U,E}$$

$$\Updownarrow$$

$$\forall \bar{f}, \bar{f_R} \,.\, \forall \bar{i}, \bar{s} \,.\, \exists \bar{c} \,.\, \exists \bar{f}_{R'} \,.\, \exists \bar{s}', \lambda \,.\, \varphi_{U,E}$$

Ackermann's Reduction: UIF-E → E

- Replace all function instances with fresh variables $f(x) \rightsquigarrow \hat{f}_x$ and thus obtain $\overline{\varphi_{U,E}}$
- Add functional consistency constraints

$$FC \quad \Leftrightarrow \quad [(x = y) \to (\hat{f}_x = \hat{f}_y)] \wedge \ldots$$

and obtain

$$\varphi_e \Leftrightarrow FC_{f,f_R} \to \left(FC_{f_{R'}} \wedge \overline{\varphi_{U,E}}\right)$$

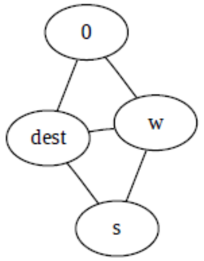
$$\forall \bar{f}, \bar{f_R} \,.\, \forall \bar{i}, \bar{s} \,.\, \exists \bar{c} \,.\, \exists \bar{f_{R'}} \,.\, \exists \bar{s}', \lambda \,.\, \varphi_{U,E}$$

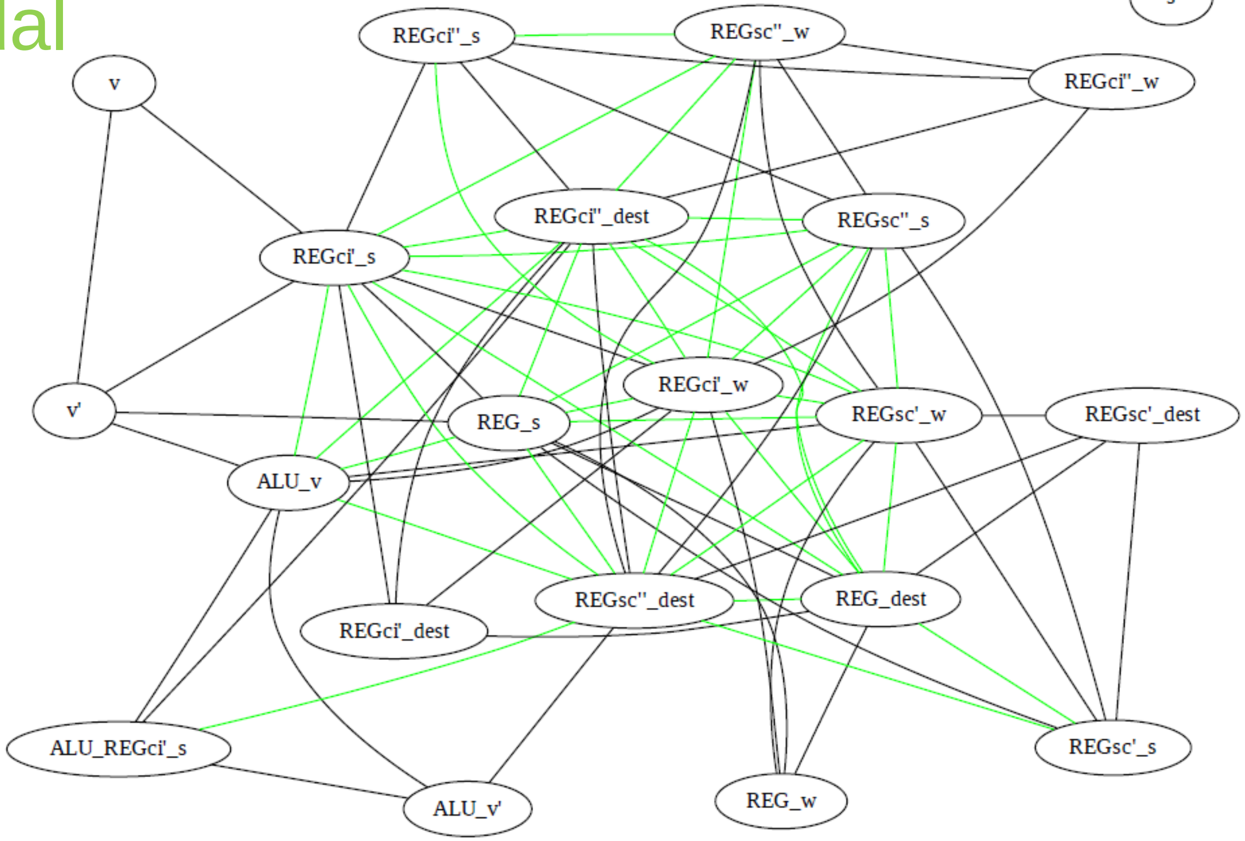
$$\Updownarrow$$

$$\forall \widehat{f^u}, \bar{f}^u_R, \bar{i}, \bar{s} \,.\, \exists \bar{c} \,.\, \exists \widehat{f^{u,e}_{R'}}, \bar{s}', \lambda \,.\, \forall \widehat{f^e}, \widehat{f^e_R} \,.\, \varphi_e$$

E → Prop. Logic (Graph-based)

- Build the non-polar equality graph
- Make it chordal





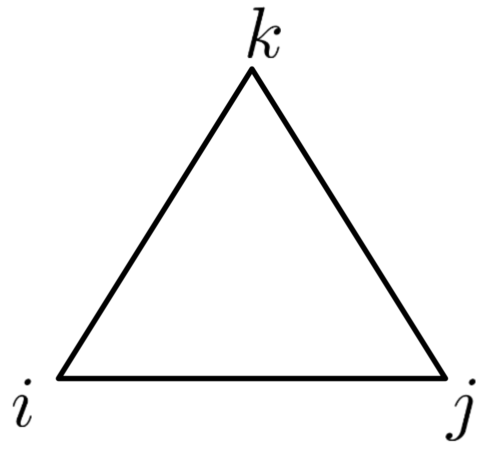
E → Prop. Logic (continued)

- Replace equalities with fresh Boolean variables

$$(x = y) \quad \rightsquigarrow \quad e_{x,y}$$

- For each triangle (i, j, k) in the equality graph, add the following conjunct to $\mathcal{B}_{trans}$

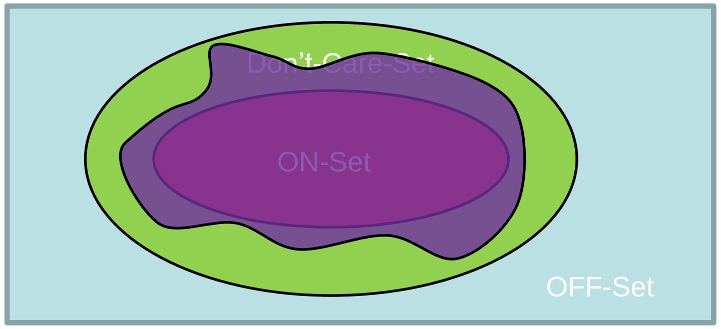
$$(e_{i,j} \wedge e_{j,k} \rightarrow e_{i,k}) \wedge$$
$$(e_{i,j} \wedge e_{i,k} \rightarrow e_{j,k}) \wedge$$
$$(e_{i,k} \wedge e_{j,k} \rightarrow e_{i,j})$$



- Open point:
  - Respect quantifier structure

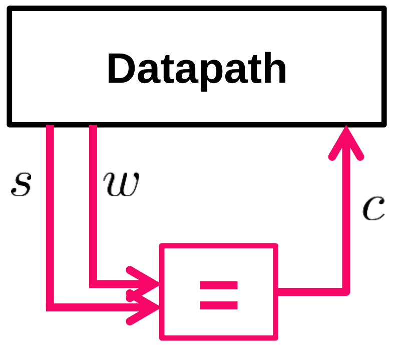
Extract Function for Control Logic

- We started from: $\forall \bar{R} \,.\, \forall \bar{f} \,.\, \forall \bar{i}, \bar{s} \,.\, \exists \bar{c} \,.\, \exists \bar{R'} \,.\, \exists \bar{s'} \,.\, \varphi$
- Apply transformations, obtain φ_{prop}
- Existentially quantify "next states" $\bar{s'}, \bar{R'}$
  - i.e., quantify all variables which "come from" one of the next state variables. E.g. $e_{REG''_s, REG'_s}$
- Expand existential quantification of $\bar{c}$
  - Example: $\exists c \,.\, \varphi_{prop} \Leftrightarrow (c \wedge \varphi_{prop}) \vee (\neg c \wedge \varphi_{prop})$
- Find cofactors of c
  - Positive Cofactor: ON-Set + DC-Set
  - Negative Cofactor: OFF-Set + DC-Set
- Find function in this interval



Results / Summary

- We started from
  - a **datapath** of the target system
  - a **reference implementation**
  - an **equivalence criterion**
- We obtained
  - Boolean function(s) for the control logic
  - in terms of
    - (dis-)equalities between inputs and states
    - Example: $c \Leftrightarrow (s = w) \Leftrightarrow e_{s,w}$



Open Points / Current Problems

- Proof(s) for Transformations
  - unfinished
- Practical issues
  - Runtime complexity?
  - Efficiency:
    - BDDs
    - SMT Solvers
- “Implementation”
  - Only hardcoded for simple pipeline example
  - Based on BDD operations
  - Not even (completely) finished