PROGRAM SYNTHESIS

RUZICA PISKAC
YALE UNIVERSITY

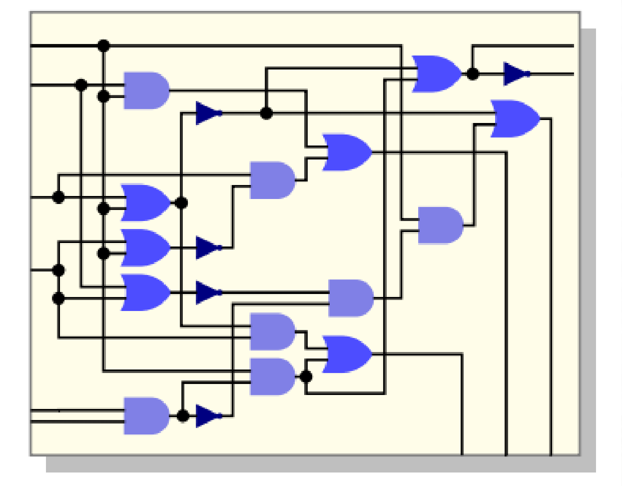
Oregon Programming Languages
Summer School 2023

REACTIVE SYNTHESIS – “HOLY GRAIL” (WELL, ONE OF THEM)

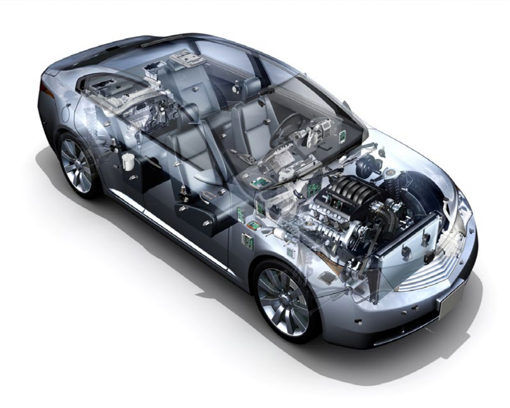
- Autonomous driving
  - Reactive traffic planner decides whether vehicle should stay in the travel lane or perform a passing maneuver, whether it should go or stop, whether it is allowed to reverse, etc.
  - Hierarchical control: reactive traffic planner interacts with mission control (above) and path planner (below).
- Specification consists of
  - Traffic rules (for example “no collision”, “obey speed limits”, ...
  - Goals (for example “eventually the checkpoint should be reached”)
- More in the following survey paper [Murray et al, 2012]

REACTIVE SYNTHESIS

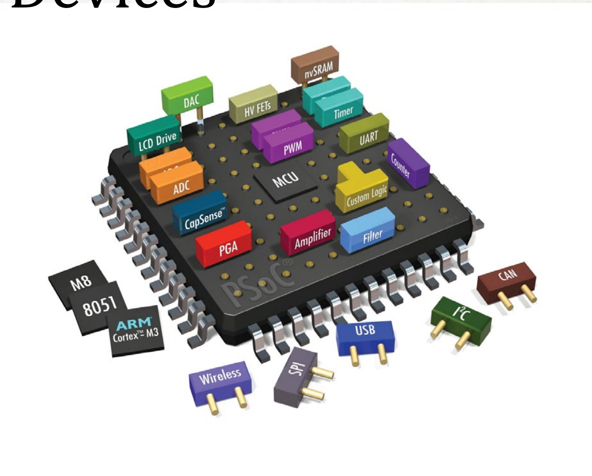
Is a Boolean circuit the right representation for these systems?



Cyber-physical Systems

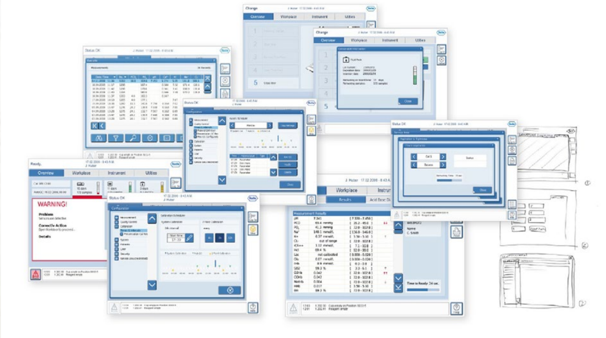


Embedded Devices



Mobile Applications

Graphical User Interfaces



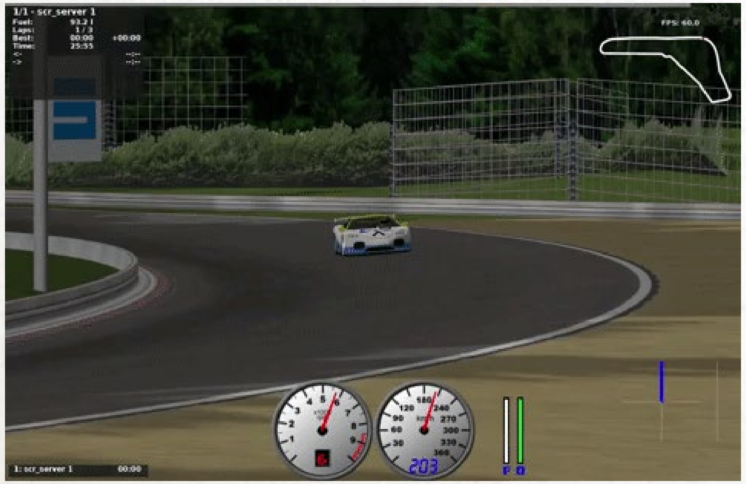
Idea: Temporal Stream Logic
Abstract away from Boolean circuits

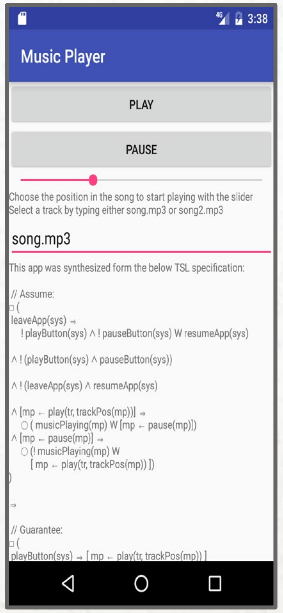
REACTIVE SYNTHESIS – LIMITATIONS

- Applications are still limited to small examples
  - Synthesis from LTL specifications is 2EXPTIME hard
  - Synthesis of distributed systems (where the processes have incomplete information) is in general undecidable
- We tried to synthesize a simple autonomous driving controller [SCAV2017] with current state of the art tools
- The controller only needs to switch between a small number of behaviors, like steering during a bend, or shifting gears on high rpm
- To detect those situations, the controller needs to process 20+ sensors of the car
- This accumulation of sensors values exceeded the capabilities of the tools

SPOILER ALERT

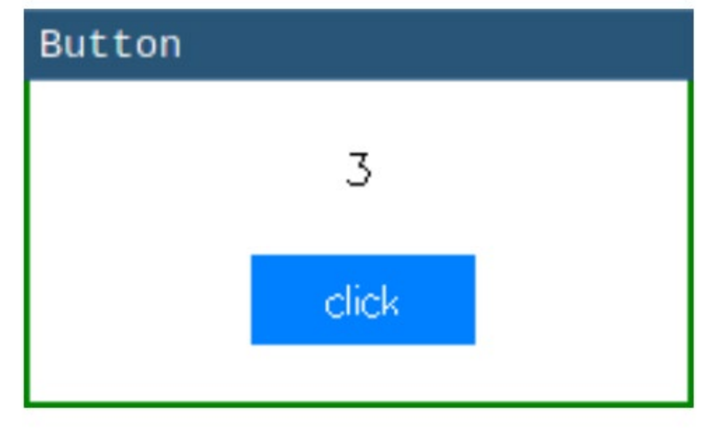
- New logic: TSL (temporal stream logic), defined over streams of data, with user defined/API predicates and function calls
- New synthesis “procedures” – extending the existing work on reactive synthesis to this new logic, outputting executable FRP programs
- New applications: among others we synthesized a controller for a simulator for autonomous vehicles, a music player, ...





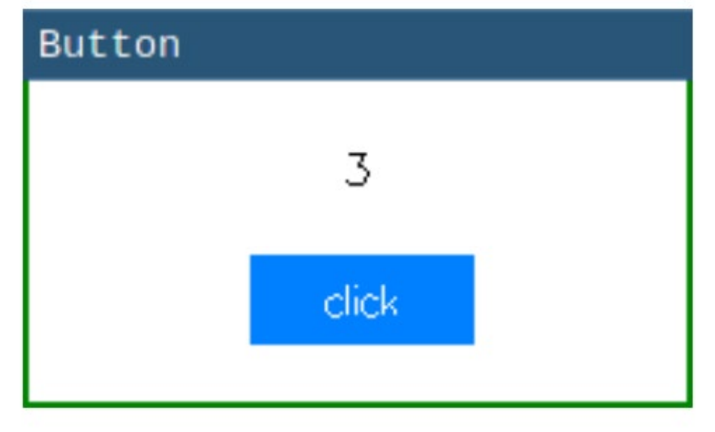
TEMPORAL STREAM LOGIC (TSL)

CAV 2019. Finkbeiner, Klein, Piskac, Santolucito



$$\square\big((\texttt{pressedEvent click} \leftrightarrow [\![\texttt{count} \leftarrowtail \texttt{increment count}]\!]) \wedge [\![\texttt{screen} \leftarrowtail \texttt{display count}]\!]\big)$$

TSL EXAMPLE

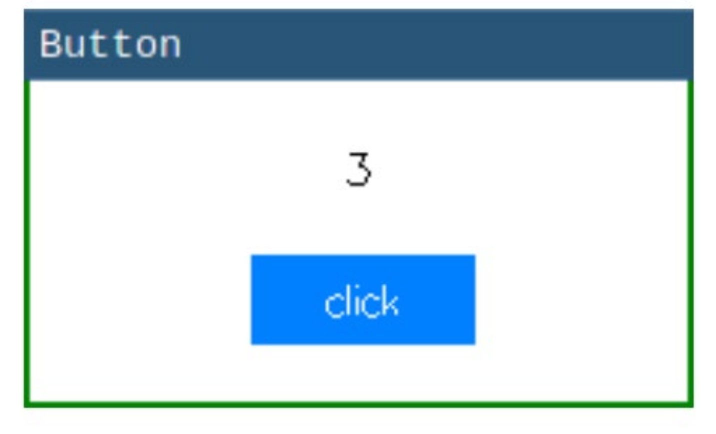



```
yampaButton =
  proc click -> do
    rec
      count' <- hold 0         -< count
      pic    <- arr display -< count'
      count  <-  if pressedEvent click
                 then arr increment -< count'
                 else arr id        -< count'
    returnA -< pic

pressedEvent = ...
increment = ...
display = ...
```

$$\square\Big(\big(\texttt{pressedEvent click} \leftrightarrow [\![\texttt{count} \leftarrowtail \texttt{increment count}]\!]\big) \wedge [\![\texttt{screen} \leftarrowtail \texttt{display count}]\!]\Big)$$

TSL EXAMPLE

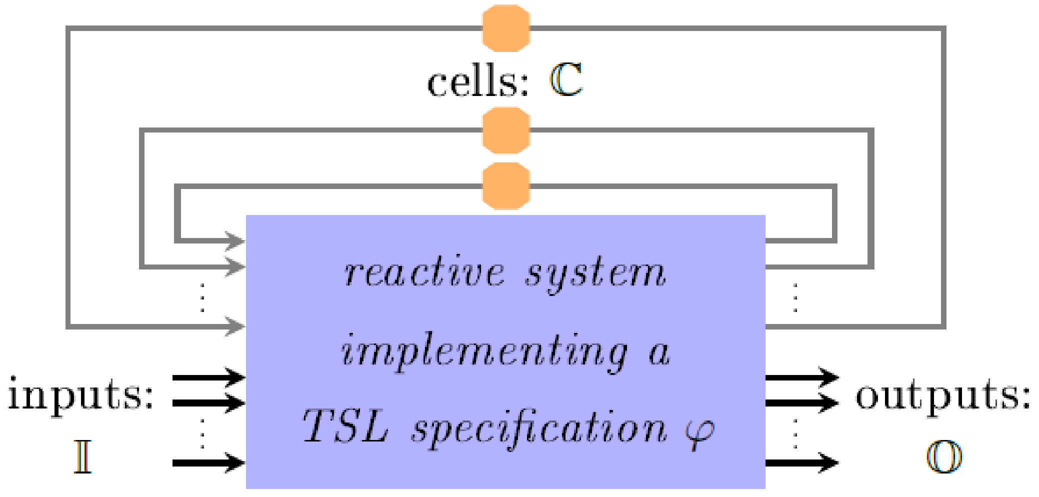
```
yampaButton =
  proc click -> do
    rec
      count'  <- hold 0          -< count
      pic     <- arr display     -< count'
      count   <-  if pressedEvent click
                  then arr increment  -< count'
                  else arr id         -< count'
    returnA -< pic

pressedEvent = ...
increment = ...
display = ...
```



$$\square\Big(\big(\texttt{pressedEvent click} \leftrightarrow [\![\texttt{count} \leftarrowtail \texttt{increment count}]\!]\big) \wedge [\![\texttt{screen} \leftarrowtail \texttt{display count}]\!]\Big)$$

TEMPORAL STREAM LOGIC (TSL)

- All temporal operators are the same as in LTL
- Input variables are not Booleans but *signals*
- Temporal operators are defined on atoms which can either be an update atom, or a predicate applied on function terms



Architecture

Function Term:

$$\tau_F \;:=\; \mathtt{s_i} \;\mid\; \mathtt{f}\ \tau_F^0\ \tau_F^1\ \cdots\ \tau_F^{n-1}$$

Predicate Term:

$$\tau_P \;:=\; \mathtt{p}\ \tau_F^0\ \tau_F^1\ \cdots\ \tau_F^{n-1}$$

Update Term:

$$\tau_U \;:=\; [\![\mathtt{s_o} \leftarrowtail \tau_F]\!]$$

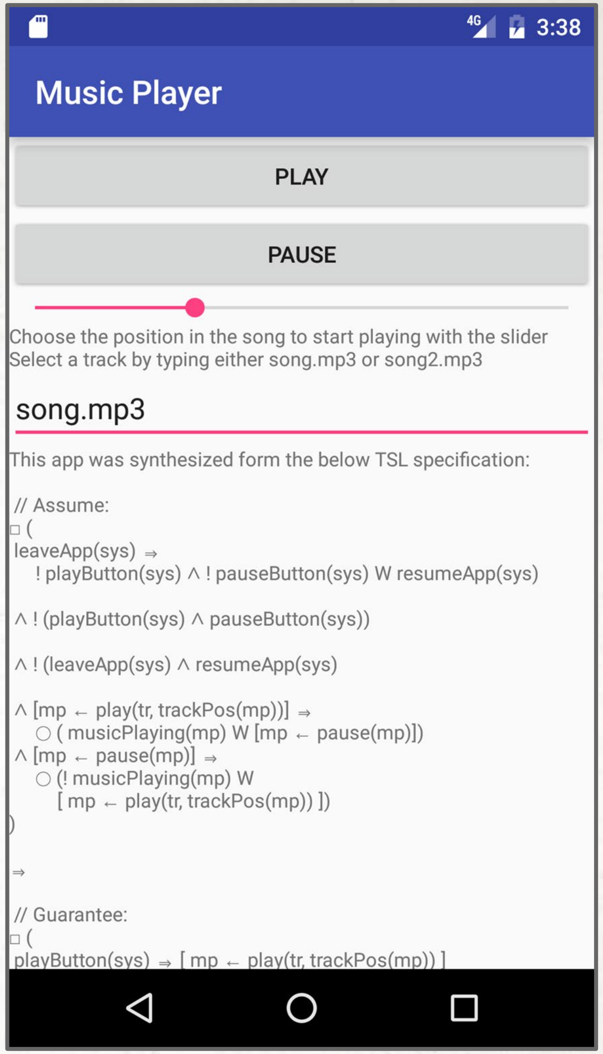
Term Definitions

SYNTHESIZING A MUSIC PLAYER APP

- Android Lifecycle

```
Sys.leaveApp() {
  if (MP.musicPlaying())
    Ctrl.pause();
}

Sys.resumeApp() {
  pos = MP.trackPos();
  Ctrl.play(Tr,pos);
}
```

Finding resume and restart errors in android applications Shan, Z., Azim, T., Neamtiu, I OOPSLA 2016



Available online: GitHub, Google Store

SYNTHESIZING A MUSIC PLAYER APP

- Android Lifecycle

```
Sys.leaveApp() {
  if (MP.musicPlaying())
    Ctrl.pause();
}

Sys.resumeApp() {
  pos = MP.trackPos();
  Ctrl.play(Tr,pos);
}
```

Input "variables" for specification:

- The Android system (`Sys`)
- The Android music player library (`MP`)
- Its control interface (`Ctrl`)
- The currently selected track (`Tr`)

- API functions and routines

SYNTHESIZING A MUSIC PLAYER APP

- Android Lifecycle

```
Sys.leaveApp() {
  if (MP.musicPlaying())
    Ctrl.pause();
}

Sys.resumeApp() {
  pos = MP.trackPos();
  Ctrl.play(Tr,pos);
}
```

```
ALWAYS (leaveApp(Sys) ∧
musicPlaying(MP)
        ⇒ [Ctrl ↢ pause()])

ALWAYS (resumeApp(Sys)
        ⇒ [Ctrl ↢
        play(Tr,trackPos(MP)])
```

SYNTHESIZING A MUSIC PLAYER APP

- Android Lifecycle

```
Sys.leaveApp() {
  if (MP.musicPlaying())
    Ctrl.pause();
  }
```

```
ALWAYS (leaveApp(Sys) ∧
musicPlaying(MP)
       ⇒ [Ctrl ↢ pause()])
```

New task:

On resume app, only play music if the music was already playing when paused.

)

SYNTHESIZING A MUSIC PLAYER APP

- Android Lifecycle

```
bool wasPlaying = false;

Sys.leaveApp() {
  if (MP.musicPlaying()){
    wasPlaying = true;
    Ctrl.pause();}
  else {
    wasPlaying = false;}
}

Sys.resumeApp() {
  if (wasPlaying) {
    pos = MP.trackPos();
    Ctrl.play(Tr,pos);
  }
}
```

ALWAYS (leaveApp(Sys) ∧ musicPlaying(MP)
⇒ [Ctrl ↢ pause()])

ALWAYS (
leaveApp(Sys) ∧ musicPlaying(MP)
⇒
[Ctrl ↢ play(Tr,trackPos(MP)]
AS_SOON_AS
resumeApp(Sys)])

AS_SOON_AS:

$\varphi \; A \; \psi \equiv \neg\psi \; W(\psi \wedge \varphi)$

FUNCTION ABSTRACTION

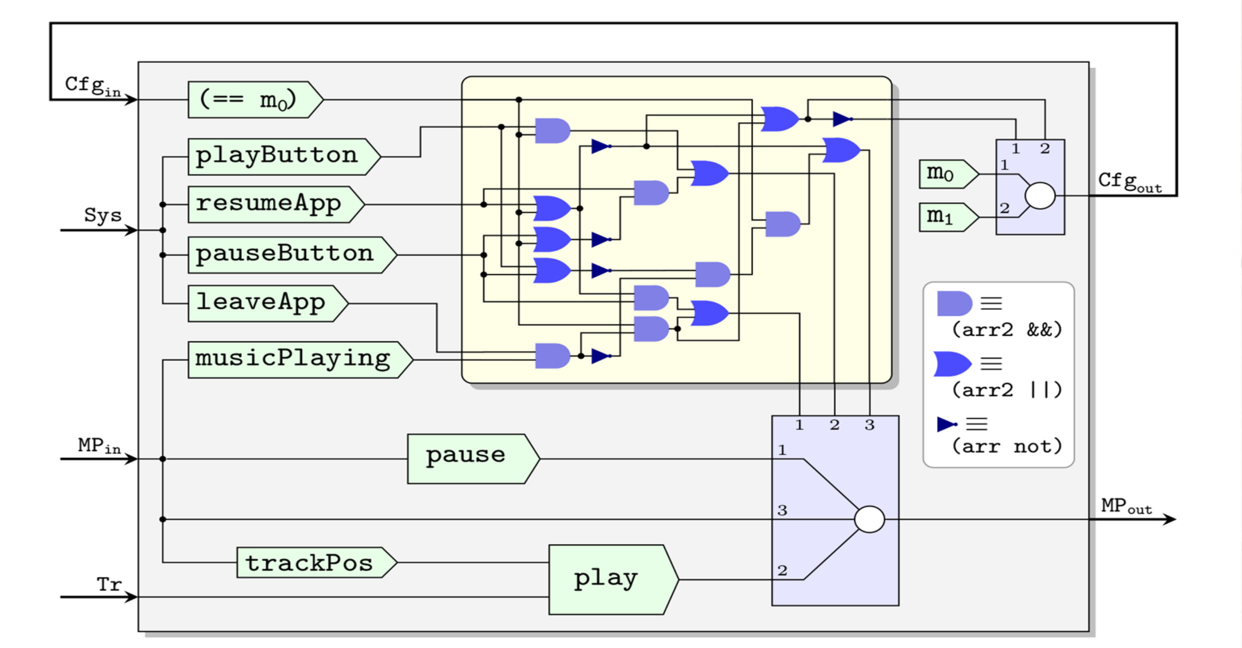
Reactive Synthesis?

```
ALWAYS (leaveApp(Sys) ∧ musicPlaying(MP)
       ⇒ [Ctrl ↢ pause()])

ALWAYS (leaveApp(Sys) ∧ musicPlaying(MP) ⇒
[Ctrl ↢ play(Tr,trackPos(MP)]
           AS_SOON_AS resumeApp(Sys)])
```

Control

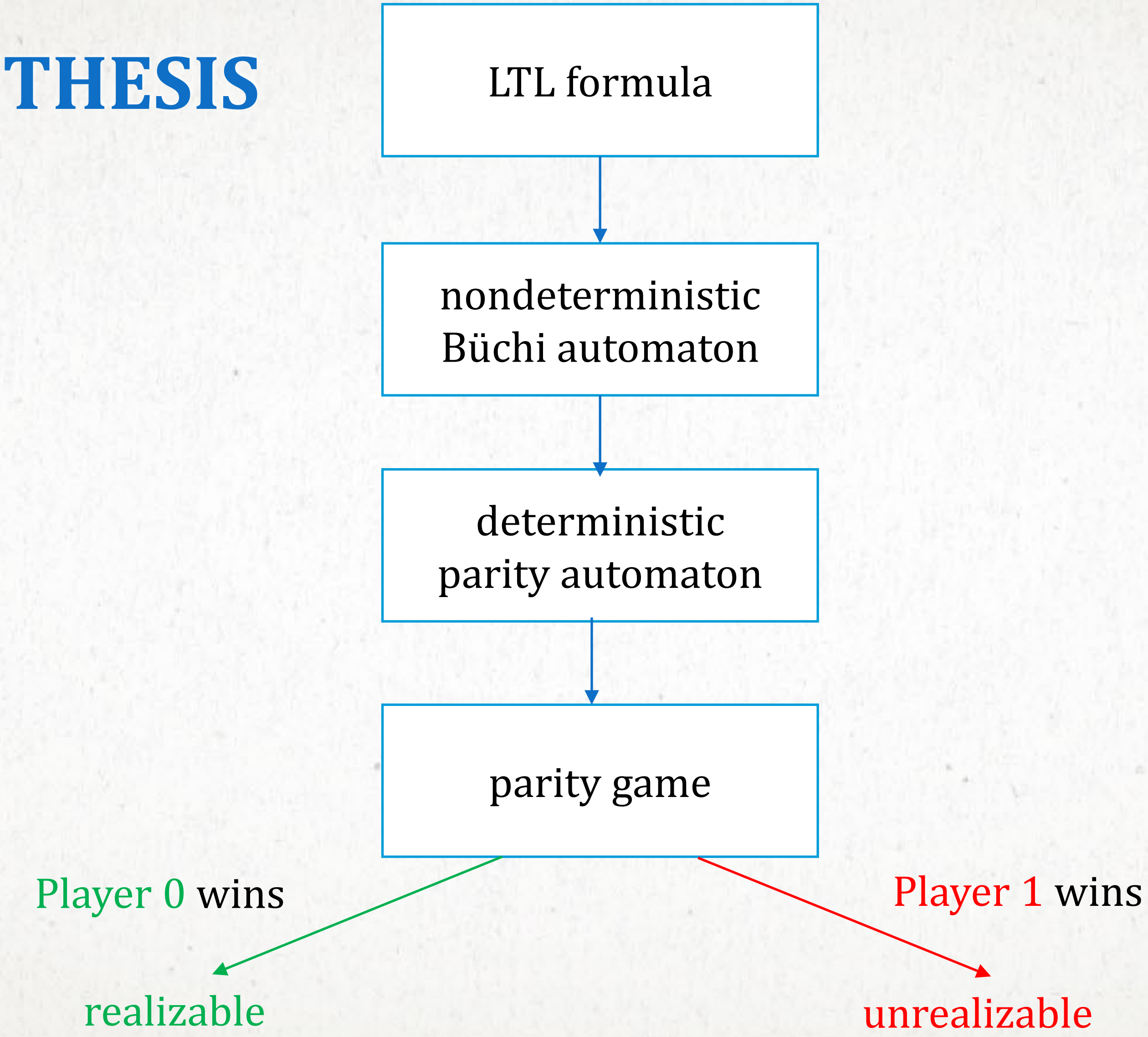
```
leaveApp(Sys){ ... }

musicPlaying(MP) { ... }

play(Tr,trackPos(MP)) { ... }

resumeApp(Sys) { ... }
```

Pure Data Transformations

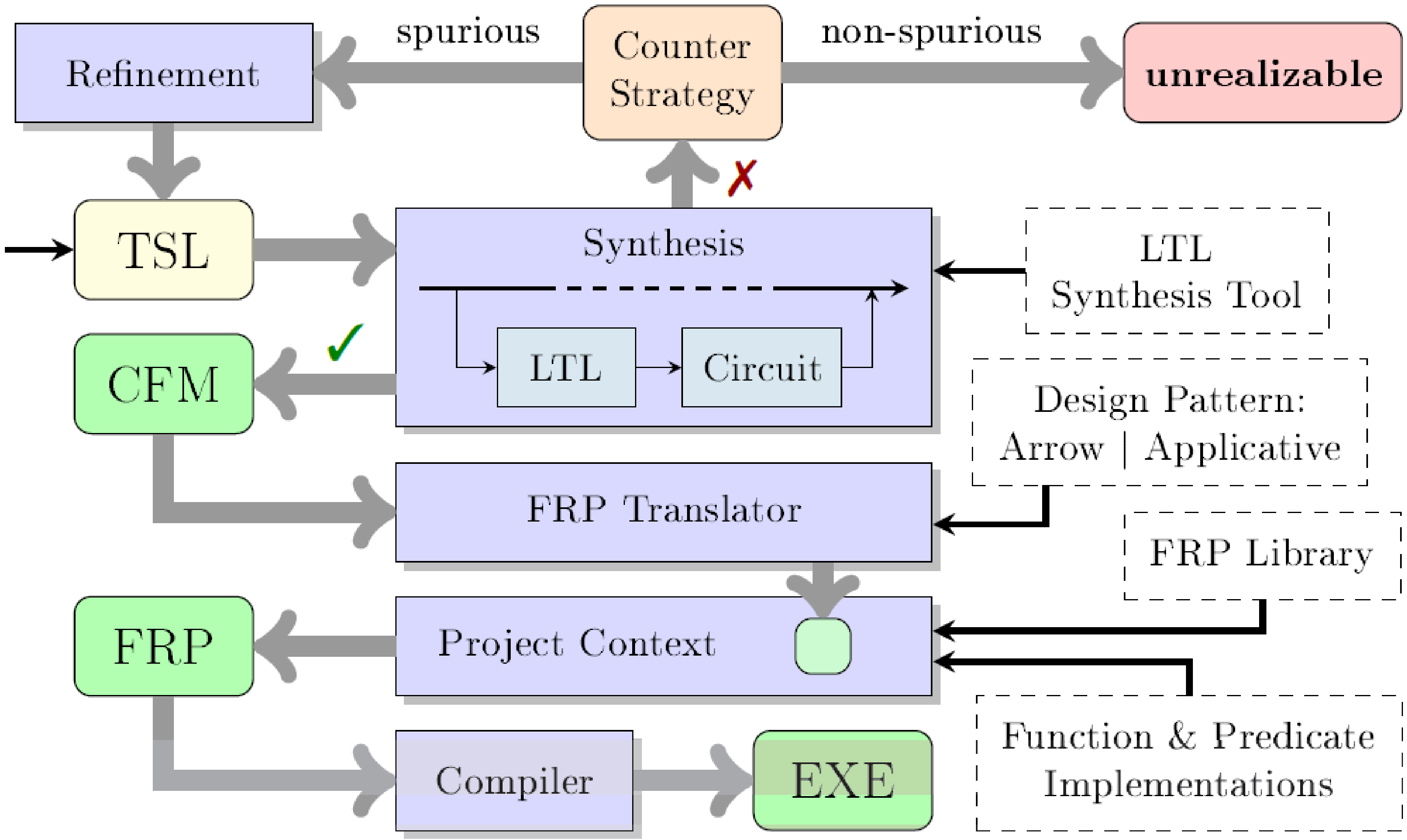


SYNTHESIS FROM TSL SPECIFICATIONS

LTL SYNTHESIS

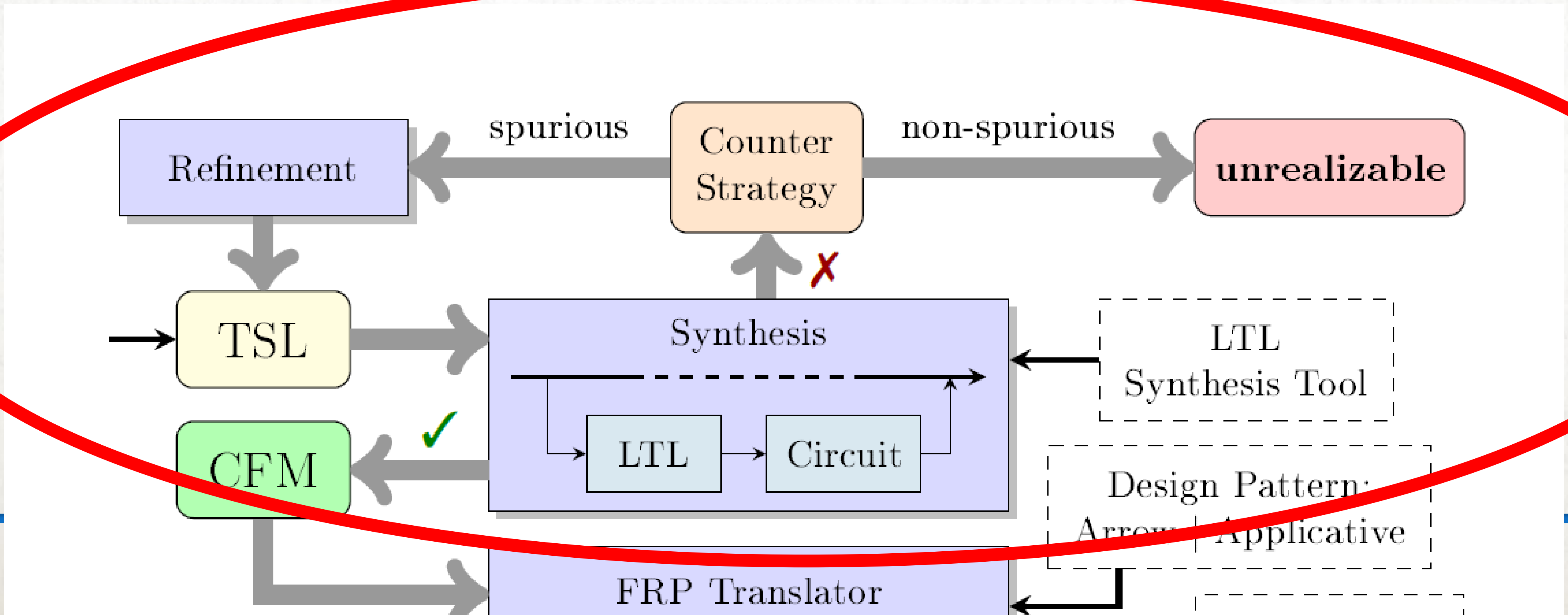


OVERVIEW OF THE SYNTHESIS PROCEDURE



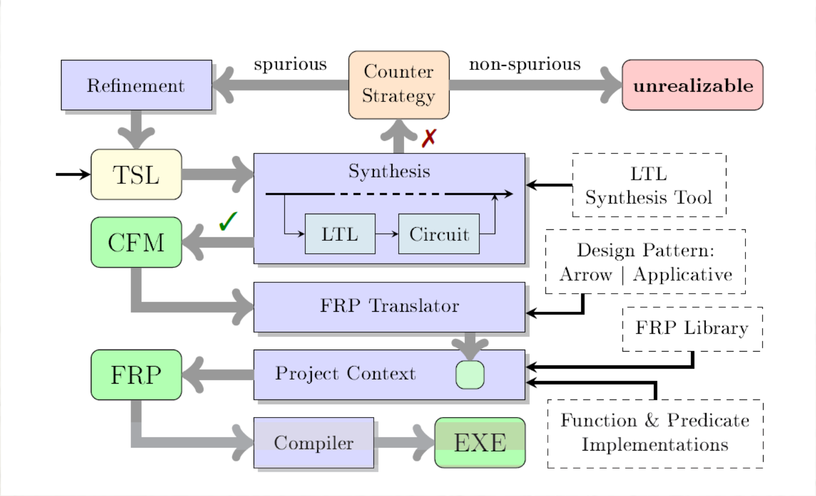
TSL SYNTHESIS PROCEDURE

- **Theorem1:** TSL synthesis problem is undecidable (reducing the Post correspondence problem to a TSL synthesis problem)



TSL SYNTHESIS PROCEDURE

- **Theorem1:** TSL synthesis problem is undecidable (a proof by reducing the Post correspondence problem to a TSL synthesis problem)
- **Theorem2:** If the abstracted TSL formula is realizable (in LTL), then is the original formula also realizable
- An LTL synthesis tool constructs a control flow, which means that this flow holds for any given implementation of predicates and functions



TSL SYNTHESIS PROCEDURE – EXAMPLE 0.1

TSL specification

$$\boldsymbol{F}\, p(x) \Rightarrow \boldsymbol{FG}\, p\,(y)$$

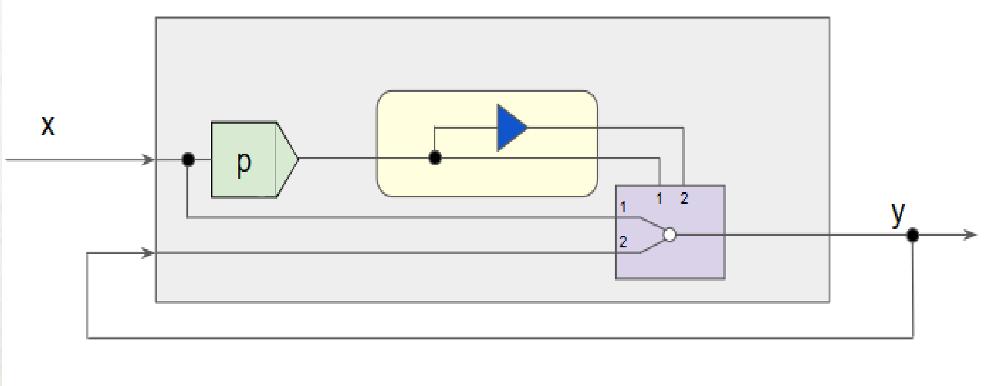
x – input, y – output signals

Syntactic conversion

LTL specification

$$\boldsymbol{F}\, p_x \Rightarrow \boldsymbol{FG}\, p_y$$

p_x, p_y – inputs



This LTL specification is unrealizable: the system simply set p_x to be always true, and p_y – to be always false

TSL SYNTHESIS PROCEDURE – EXAMPLE 1.1

TSL

$$\boldsymbol{F}\, p(x) \Rightarrow \boldsymbol{FG}\, p\,(y) \wedge \boldsymbol{F}\,[y \leftarrowtail y]$$

x – input, *y – output*

Syntactic conversion

LTL specification

$$\boldsymbol{G}\left(\left(y_y \wedge \neg y_x\right) \vee \left(\neg y_y \wedge y_x\right)\right) \wedge$$

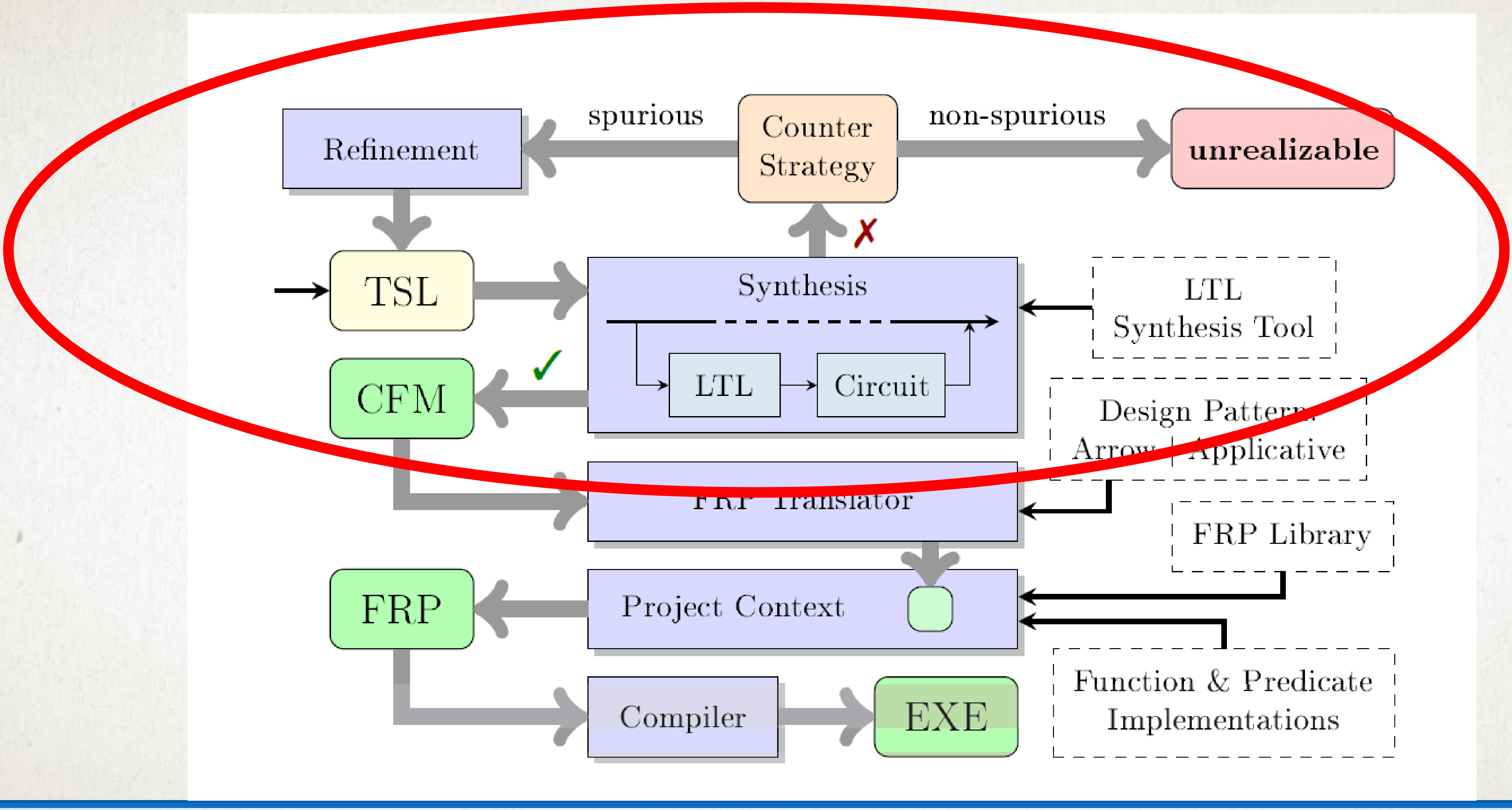
$$\boldsymbol{F}\, p_x \Rightarrow \boldsymbol{FG}\, p_y \wedge \boldsymbol{F}\, y_y$$

p_x, p_y *– inputs*

The top line specifies that *y* can be updated with only one value.

TSL TO LTL ABSTRACTION

- Given a TSL formula, the abstracted LTL formula will be a conjunction of
    - Syntactic conversion from the TSL formula
    - Globally quantified formulas describing the uniqueness of the updates
- This abstraction might need infinitely many terms, if there are functions in the specification
- There are specifications demonstrating that observation
- In practice: lazy instantiation and CEGAR loop



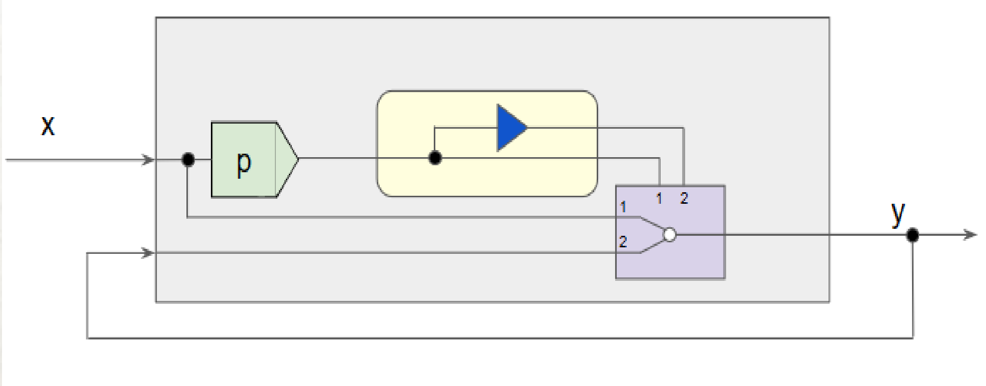
Refinement
spurious
Counter Strategy
non-spurious
unrealizable
TSL
Synthesis
LTL
Circuit
LTL Synthesis Tool
CFM
Design Pattern
Arrow
Applicative
FRP Translator
FRP Library
FRP
Project Context
Function & Predicate Implementations
Compiler
EXE

FROM STRATEGIES TO SPECIFICATION REFINEMENT

TSL specification

$$\boldsymbol{F}\, p(x) \Rightarrow \boldsymbol{FG}\, p\,(y)$$

x – input, y – output signals



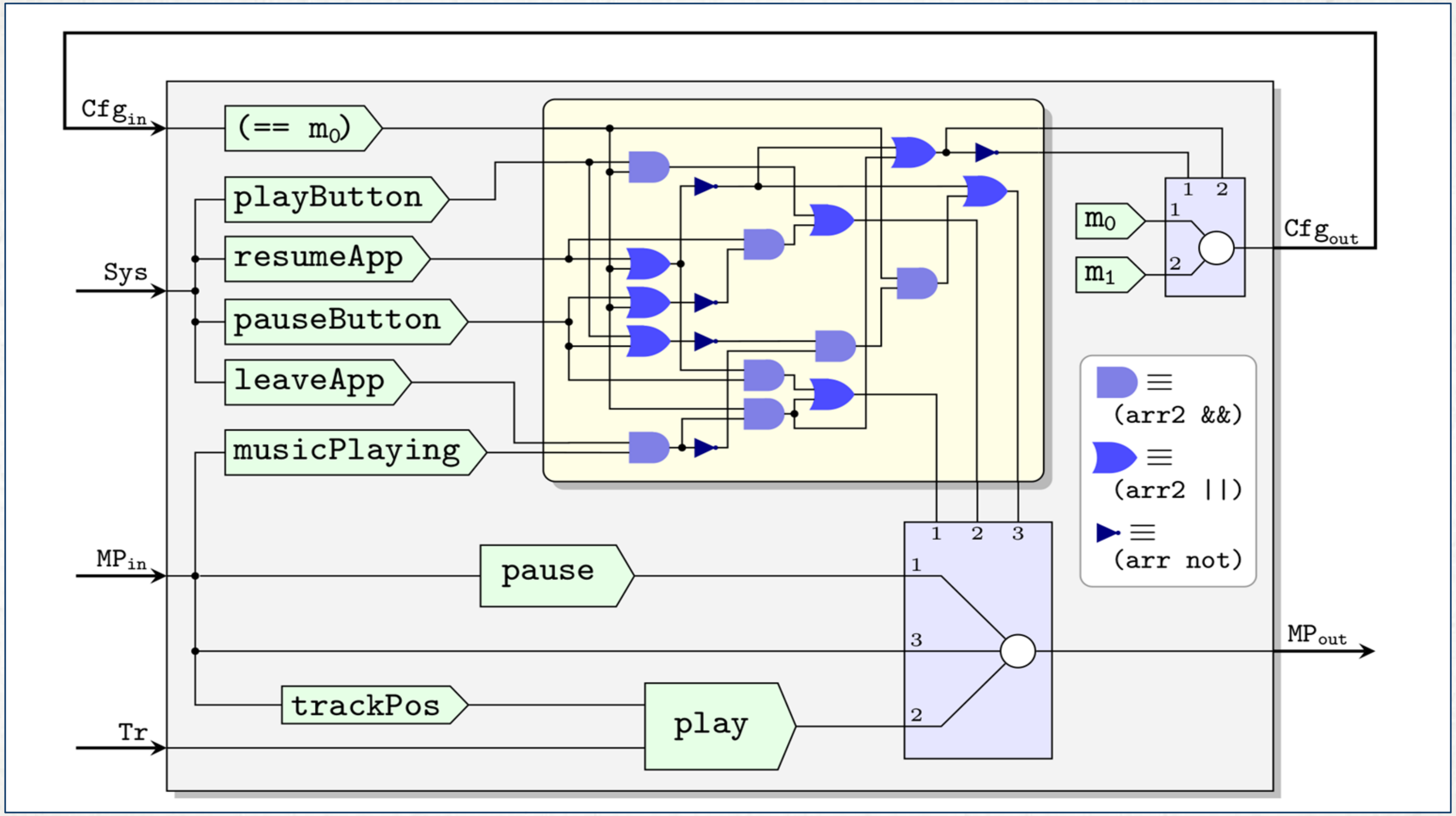
TSL specification refinement

$$\boldsymbol{F}\, p(x) \wedge$$
$$\boldsymbol{G}([y \leftarrow x] \wedge p(x) \Rightarrow \boldsymbol{X} p(y)) \wedge$$
$$\boldsymbol{G}([y \leftarrow x] \wedge \neg p(x) \Rightarrow \boldsymbol{X} \neg p(y)) \wedge$$
$$\boldsymbol{G}([y \leftarrow y] \wedge p(y) \Rightarrow \boldsymbol{X} p(y)) \wedge$$
$$\boldsymbol{G}([y \leftarrow y] \wedge \neg p(y) \Rightarrow \boldsymbol{X} \neg p(y))$$
$$\Rightarrow \boldsymbol{FG}\, p\,(y)$$

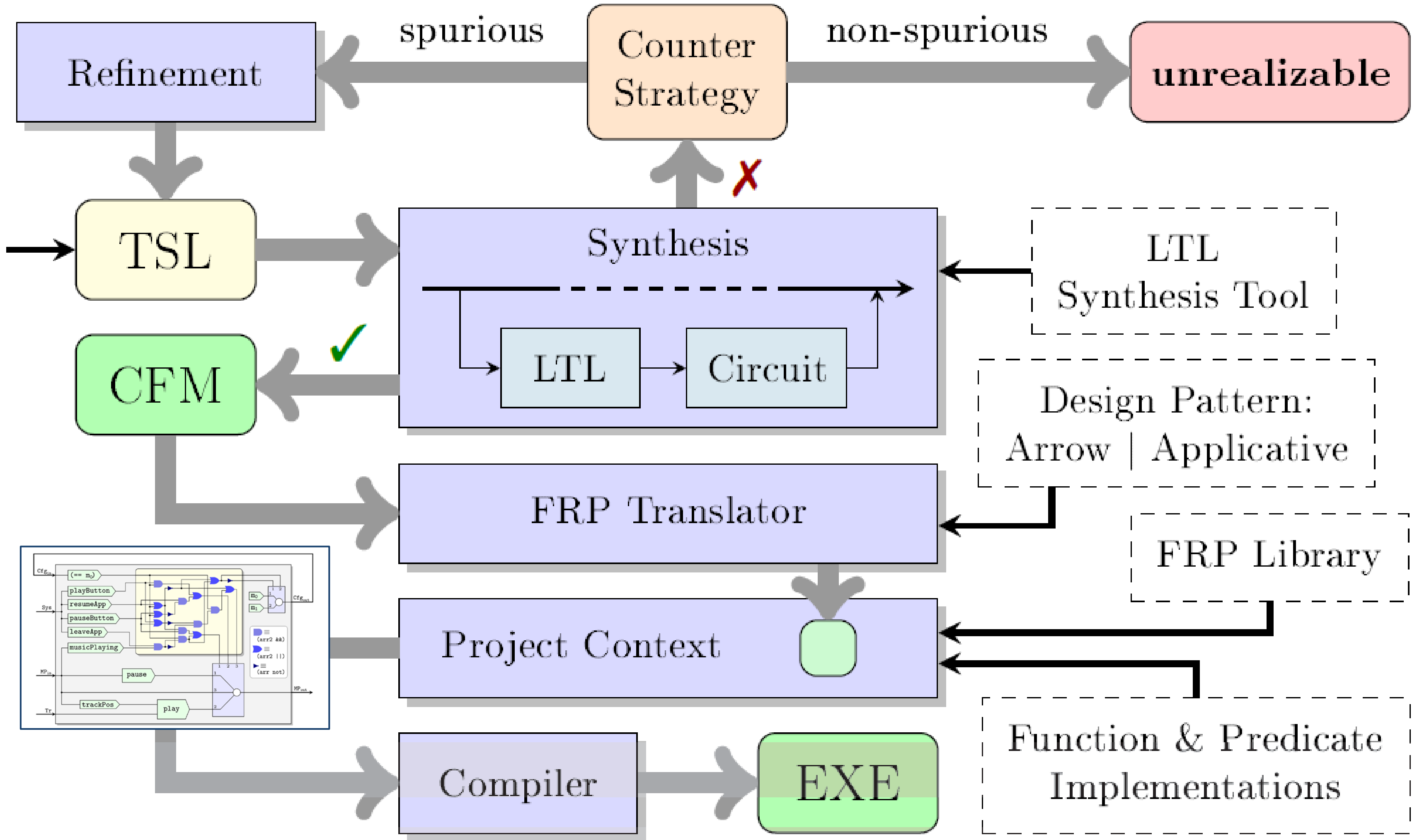
x – input, y – output signals

This new specification is strong enough to be realizable in LTL, when abstracted

MUSIC PLAYER SYNTHESIS



MUSIC PLAYER SYNTHESIS

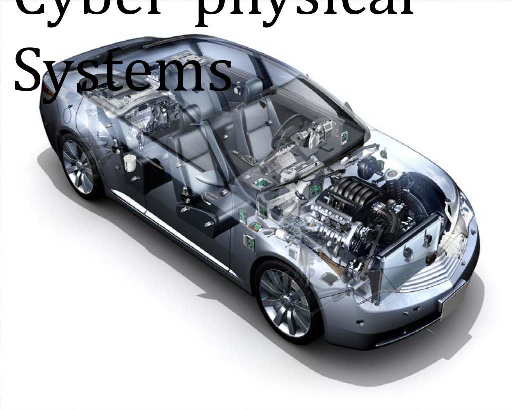


REACTIVE SYSTEMS

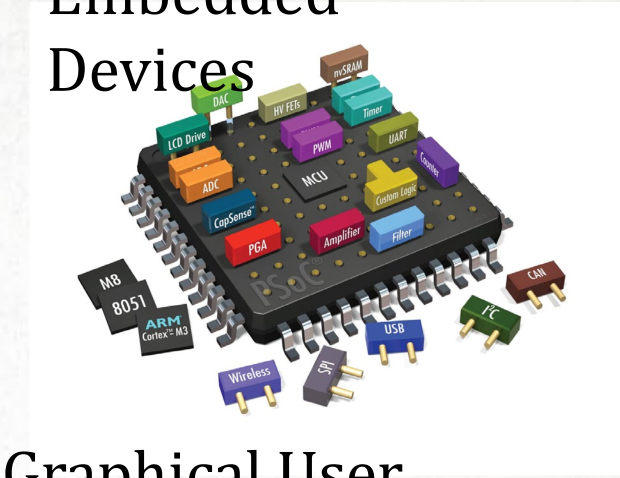
Abstracting from data transformations allows synthesis to scale to new application domains.

We trade theoretical complexity for practical scalability.

Cyber-physical Systems



Embedded Devices



Mobile Applications

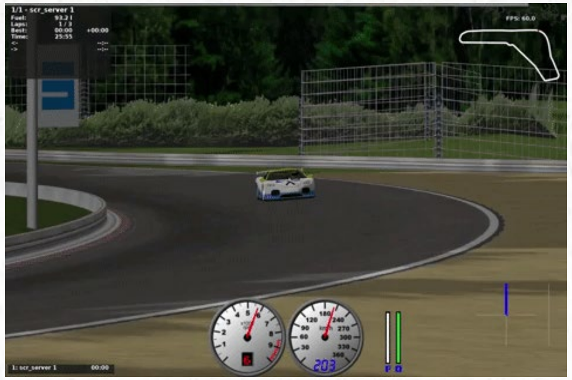
Graphical User Interfaces

Temporal Stream Logic - Synthesis Beyond the Bools. CAV 2019. Finkbeiner, Klein, Piskac, Santolucito
Synthesizing Functional Reactive Programs. Haskell 2019. Finkbeiner, Klein, Piskac, Santolucito

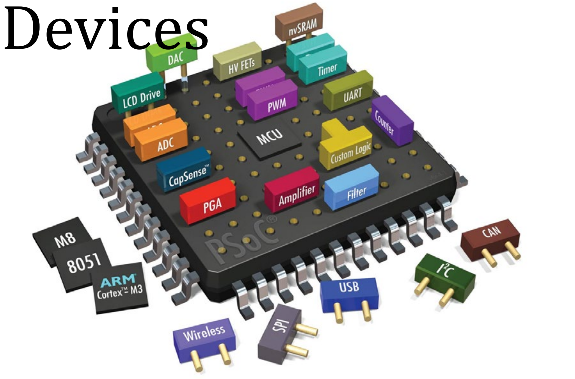
REACTIVE SYSTEMS

Synthesized a self-driving car controller in < 4 seconds

Cyber-physical

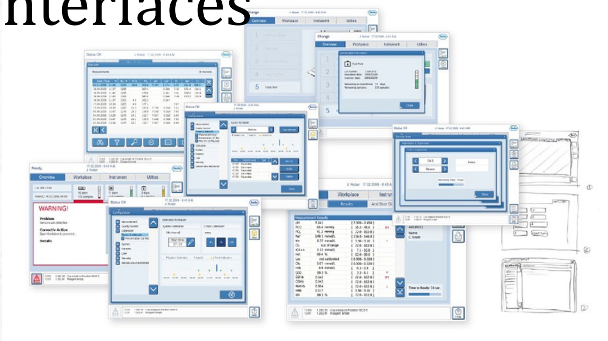


Embedded Devices



Mobile Applications

Graphical User Interfaces

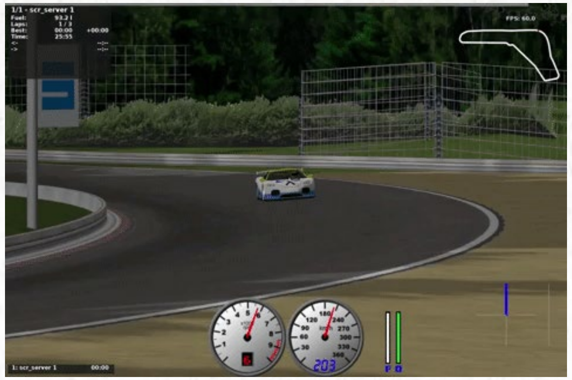


Temporal Stream Logic - Synthesis Beyond the Bools. CAV 2019. Finkbeiner, Klein, Piskac, Santolucito
Synthesizing Functional Reactive Programs. Haskell 2019. Finkbeiner, Klein, Piskac, Santolucito

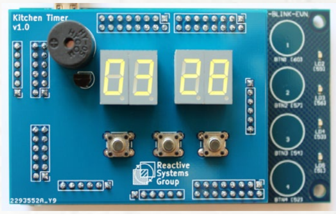
REACTIVE SYSTEMS

Synthesized a self-driving car controller in < 4 seconds

Cyber-physical

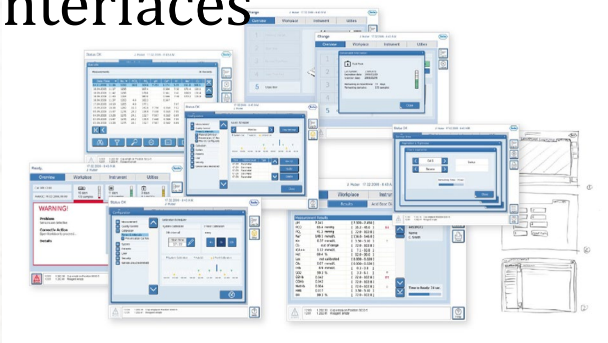


Embedded Devices



Mobile Applications

Graphical User Interfaces

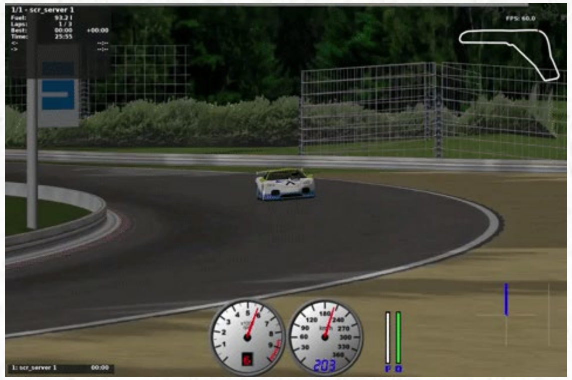


Temporal Stream Logic - Synthesis Beyond the Bools. CAV 2019. Finkbeiner, Klein, Piskac, Santolucito
Synthesizing Functional Reactive Programs. Haskell 2019. Finkbeiner, Klein, Piskac, Santolucito

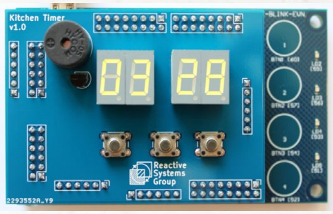
REACTIVE SYSTEMS

Synthesized a self-driving car controller in < 4 seconds

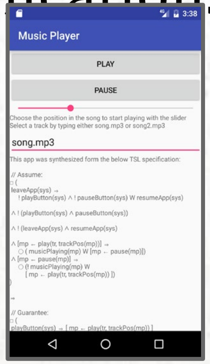
Cyber-physical



Embedded Devices



Mobile Applications



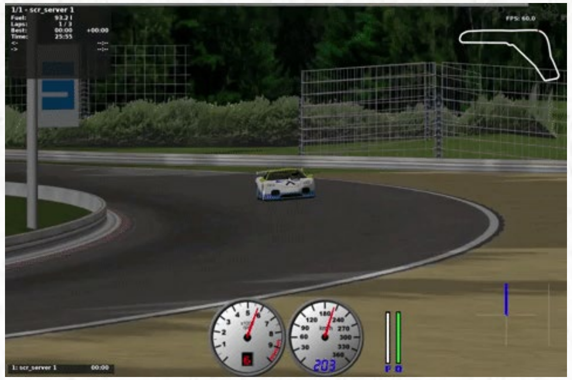
Graphical User Interfaces

Temporal Stream Logic - Synthesis Beyond the Bools. CAV 2019. Finkbeiner, Klein, Piskac, Santolucito
Synthesizing Functional Reactive Programs. Haskell 2019. Finkbeiner, Klein, Piskac, Santolucito

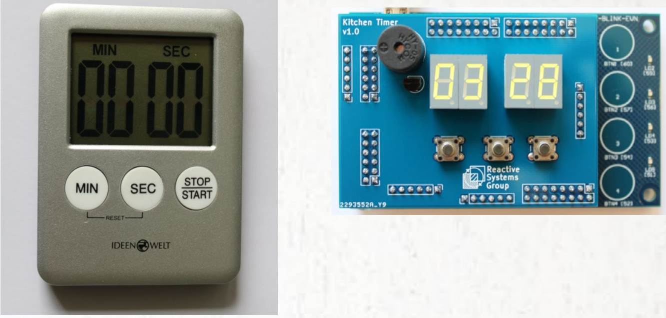
REACTIVE SYSTEMS

Synthesized a self-driving car controller in < 4 seconds

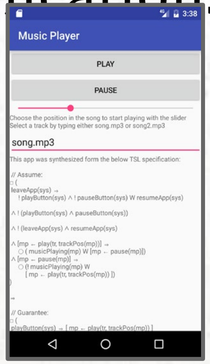
Cyber-physical



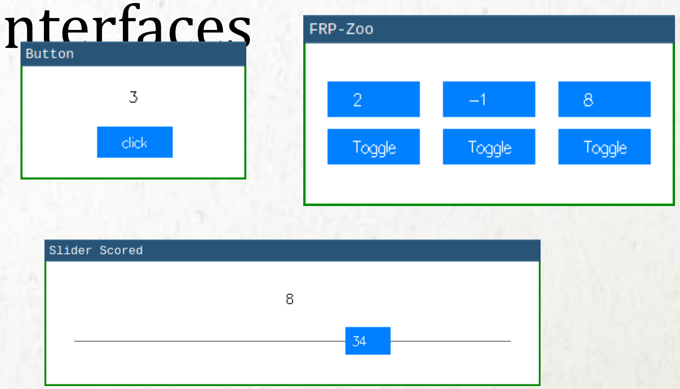
Embedded Devices



Mobile Applications



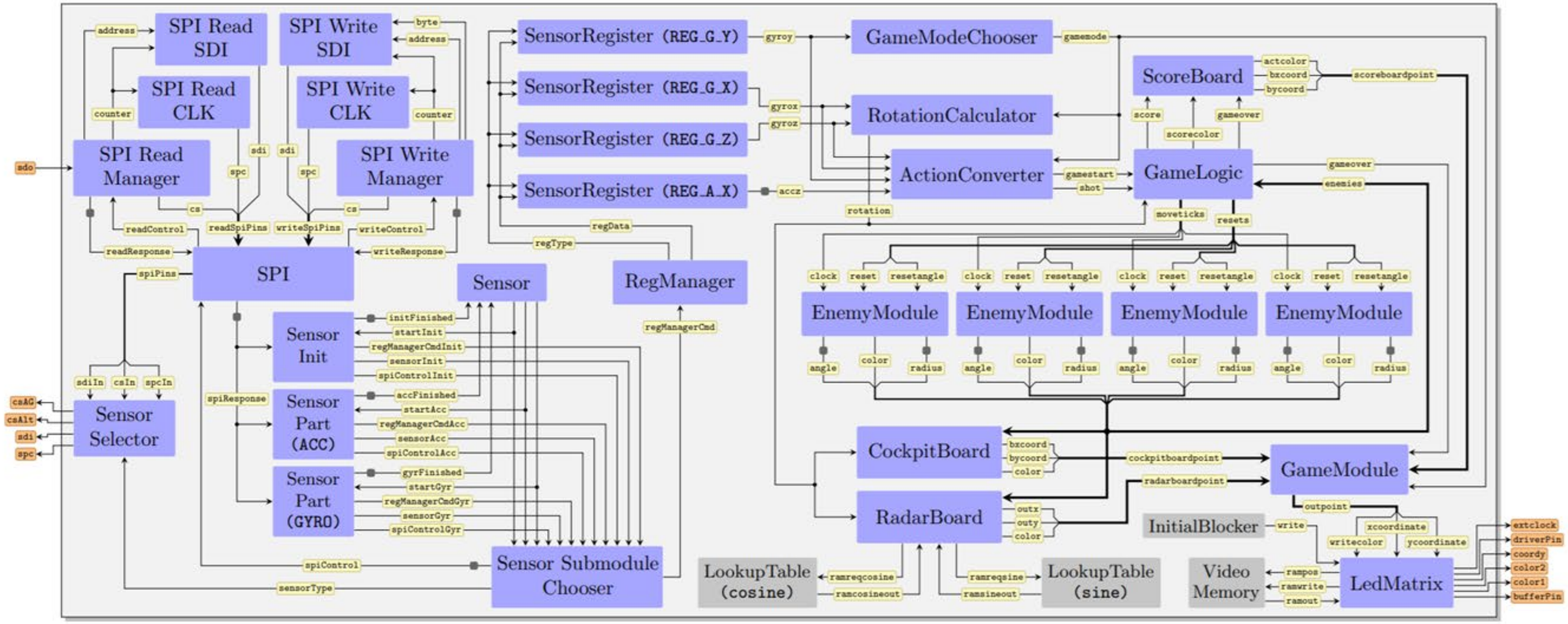
Graphical User Interfaces

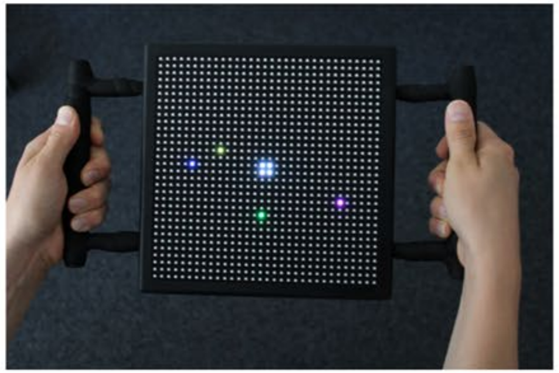


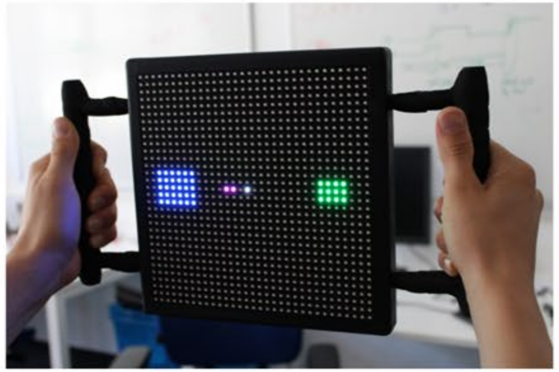
Temporal Stream Logic - Synthesis Beyond the Bools. CAV 2019. Finkbeiner, Klein, Piskac, Santolucito
Synthesizing Functional Reactive Programs. Haskell 2019. Finkbeiner, Klein, Piskac, Santolucito

Synthroids

Syntroids: Synthesizing a Game for FPGAs using Temporal Logic Specifications.
Geier, Heim, Klein, Finkbeiner: FMCAD 2019







https://github.com/reactive-systems/Synthroids

LIVE DEMOS

https://barnard-pl-labs.github.io/dynamicGrammars/frontEnd/dynamicGrammars.html
(Dylan Iskandar, Raven Rothkopf, Leyi Cui)

https://monkeyarya.github.io/moveCube/
(Arya Sinha)

https://barnard-pl-labs.github.io/tsl-api/
(Rhea Kothari, Danielle Cai, Nupur Dave)

https://stately.ai/viz/5fadaf7f-90ff-48cd-b36a-9a45dd5246a8
(Shmuel Berman)
https://github.com/Barnard-PL-Labs/tsltools/blob/master/src/test/res/specs/Heating.tsl

Can reactive synthesis and syntax-guided synthesis be friends?
Choi, Finkbeiner, Piskac, Santolucito: PLDI 2022

NOT ALL FUNCTIONS ARE REALLY UNINTERPRETED

```
always assume {

 (! (room.heating.off <-> room.heating.on)) ;
 ([ room.heating.ctrl <- turnOn() ]
   -> F ([ room.heating.ctrl <- turnOff() ] R room.heating.on)) ;
 ([ room.heating.ctrl <- turnOff() ]
   -> F ([ room.heating.ctrl <- turnOn() ] R room.heating.off));
 ([ room.heating.ctrl <- turnOff() ]
   -> F (! (gt outside.temperature room.temperature)));
}
always guarantee {

 gt outside.temperature room.temperature
   -> F room.heating.off
```

Can reactive synthesis and syntax-guided synthesis be friends?
Choi, Finkbeiner, Piskac, Santolucito: PLDI 2022

BEYOND UNINTERPRETED FUNCTIONS

$$\square\, ([\![y \leftarrow y]\!] \vee [\![y \leftarrow x]\!]) \wedge \diamond\, p\ x \rightarrow \diamond\, p\ y$$

TSL spec

$$\square\, (\texttt{x\_to\_y} \rightarrow (\texttt{p\_x} \leftrightarrow \bigcirc \texttt{p\_y})) \Longrightarrow$$

$$\square\, \neg(\texttt{y\_to\_y} \wedge \texttt{x\_to\_y}) \wedge \square\, (\texttt{y\_to\_y} \vee \texttt{x\_to\_y}) \wedge \diamond\, \texttt{p\_x} \rightarrow \diamond\, \texttt{p\_y}$$

Refined Approximation

BEYOND UNINTERPRETED FUNCTIONS

$$\square\,(⟦y \leftarrowtail y⟧ \vee ⟦y \leftarrowtail x⟧)$$
$$\wedge\ \Diamond\, p\ x \rightarrow \Diamond\, p\ y$$

TSL spec

$$\square\,(\texttt{x\_to\_y} \rightarrow (\texttt{p\_x} \leftrightarrow \bigcirc \texttt{p\_y}))$$
$$\Longrightarrow$$
$$\square\ \neg(\texttt{y\_to\_y} \wedge \texttt{x\_to\_y})$$
$$\wedge\ \square\,(\texttt{y\_to\_y} \vee \texttt{x\_to\_y})$$
$$\wedge\ \Diamond\,\texttt{p\_x} \rightarrow \Diamond\,\texttt{p\_y}$$

Refined Approximation

The refinement is a partial encoding of the semantics of uninterpreted functions.

Can we use the same strategy for other theories?

Can reactive synthesis and syntax-guided synthesis be friends?
Choi, Finkbeiner, Piskac, Santolucito: PLDI 2022

SYNTAX-GUIDED SYNTHESIS (SYGUS)

Semantic Constraint

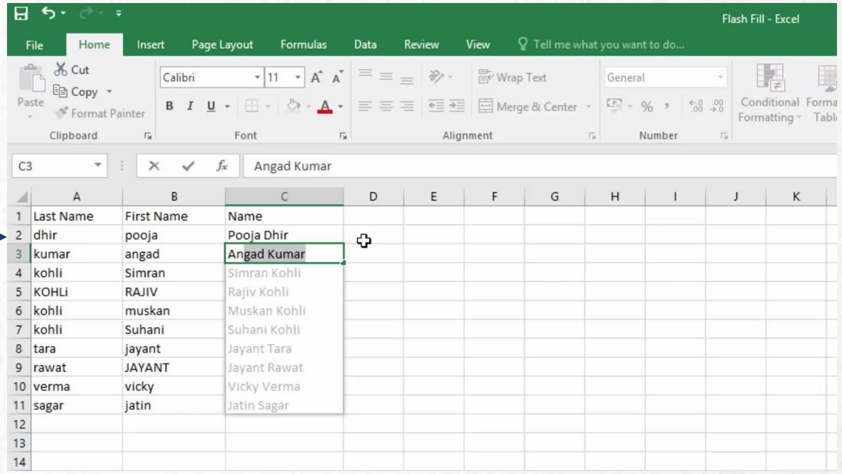
Input v_1	*Output*
Dr. Eran Yahav	*Yahav, E.*
Prof. Kathleen S. Fisher	*Fisher, K.*
Bill Gates, Sr.	*Gates, B.*
George Ciprian Necula	*Necula, G.*
Ken McMillan, II	*McMillan, K.*

Syntactic Constraint

$$
\begin{aligned}
\text{String expr } P &:= \texttt{Switch}((\mathrm{b}_1, \mathrm{e}_1), \cdot\cdot, (\mathrm{b}_n, \mathrm{e}_n)) \\
\text{Bool b} &:= \mathrm{d}_1 \vee \cdot\cdot \vee \mathrm{d}_n \\
\text{Conjunct d} &:= \pi_1 \wedge \cdot\cdot \wedge \pi_n \\
\text{Predicate } \pi &:= \texttt{Match}(v_i, \mathrm{r}, k) \mid \neg\, \texttt{Match}(v_i, \mathrm{r}, k) \\
\text{Trace expr e} &:= \texttt{Concatenate}(\mathrm{f}_1, \cdot\cdot, \mathrm{f}_n) \\
\text{Atomic expr f} &:= \texttt{SubStr}(v_i, \mathrm{p}_1, \mathrm{p}_2) \\
&\mid \texttt{ConstStr}(s) \\
&\mid \texttt{Loop}(\lambda w : \mathrm{e}) \\
\text{Position p} &:= \texttt{CPos}(k) \mid \texttt{Pos}(\mathrm{r}_1, \mathrm{r}_2, \mathrm{c}) \\
\text{Integer expr c} &:= k \mid k_1 w + k_2 \\
\text{Regular Expression r} &:= \texttt{TokenSeq}(\mathrm{T}_1, \cdot\cdot, \mathrm{T}_m) \\
\text{Token T} &:= C+ \mid [\neg C]+ \\
&\mid \text{SpecialToken}
\end{aligned}
$$

Syntax-Guided Synthesis

Synthesized Program



Great for data transformation problems!

REACTIVE SYNTHESIS

Temporal Logic Specification

Guarantee 3. *When a length-four locked burst starts, no other accesses s HREADY is high, so the current burst ends at the fourth occurrence of I true initially separately from the case in which it is not).*

$$\square((\mathrm{HMASTLOCK} \wedge \mathrm{HBURST} = \mathrm{BURST4} \wedge \mathrm{START} \wedge \mathrm{HREADY}) \rightarrow \bigcirc(\neg\mathrm{START}\ \mathcal{W}\ [3](\neg\mathrm{START} \wedge \mathrm{HREADY}))),$$

$$\square((\mathrm{HMASTLOCK} \wedge \mathrm{HBURST} = \mathrm{BURST4} \wedge \mathrm{START} \wedge \neg\mathrm{HREADY}) \rightarrow \bigcirc(\neg\mathrm{START}\ \mathcal{W}\ [4](\neg\mathrm{START} \wedge \mathrm{HREADY}))).$$

Guarantee 6. *If we do not start an access in the next time step, the bus*

For each master i,

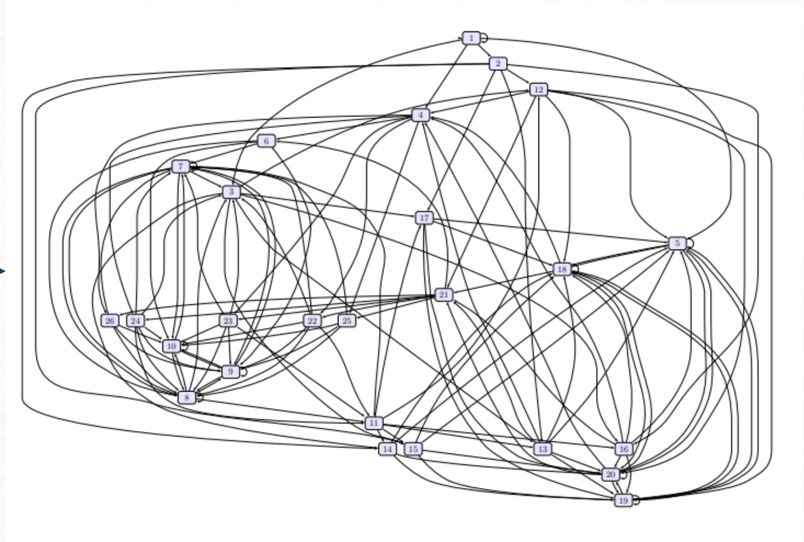
$$\square(\bigcirc(\neg\mathrm{START}) \rightarrow ((\mathrm{HMASTER} = i \leftrightarrow \bigcirc(\mathrm{HMASTER} = i)) \wedge (\mathrm{HMASTLOCK} \leftrightarrow \bigcirc(\mathrm{HMASTLOCK})))).$$

Assumption 4. *We assume that all input signals are low initially.*

$$\bigwedge_i (\neg\mathrm{HBUSREQ[i]} \wedge \neg\mathrm{HLOCK[i]}) \wedge \neg\mathrm{HREADY}.$$

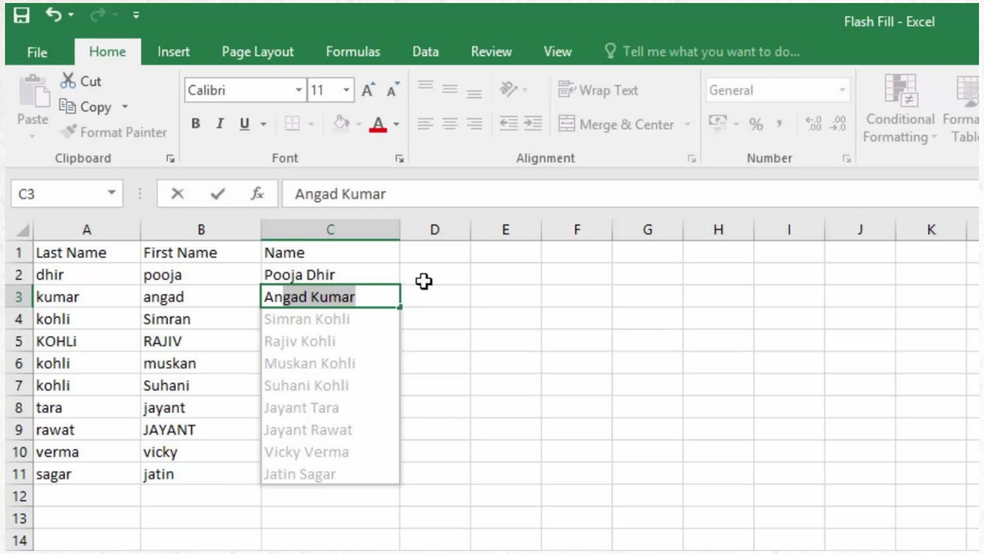
→ **Reactive Synthesis** →

Synthesized Model



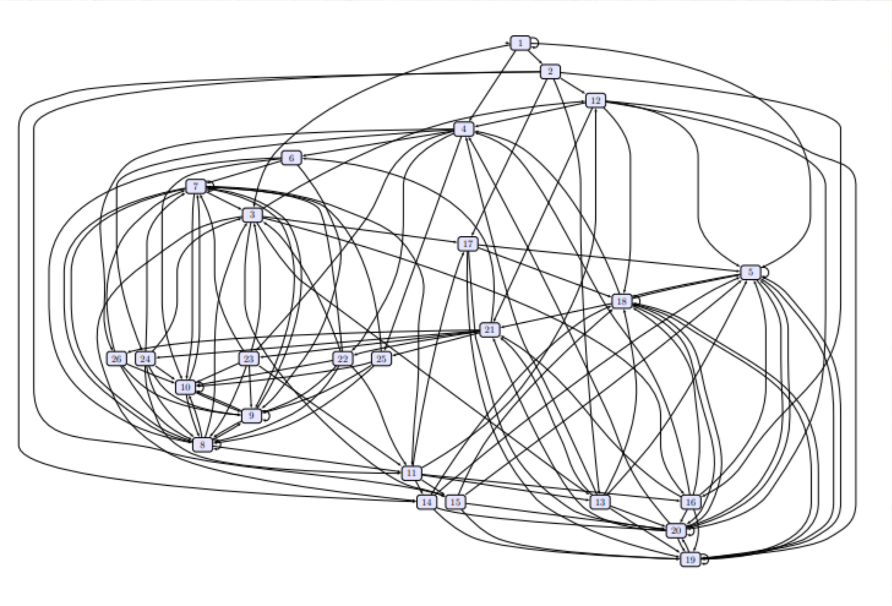
Great for control-flow problems!

SYNTAX-GUIDED SYNTHESIS (SYGUS)



Good for data transformation problems

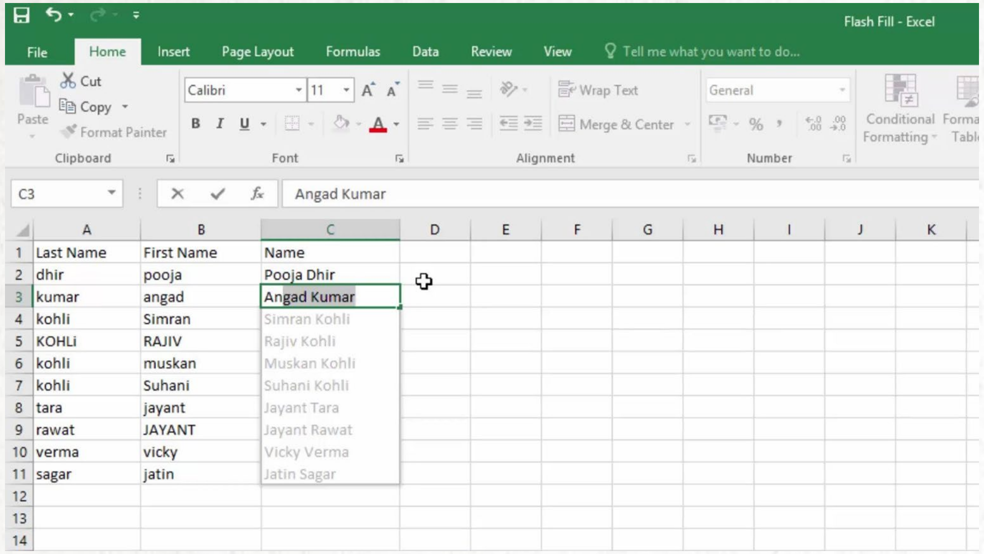
REACTIVE SYNTHESIS



Good for control-flow problems

But there’s a catch…

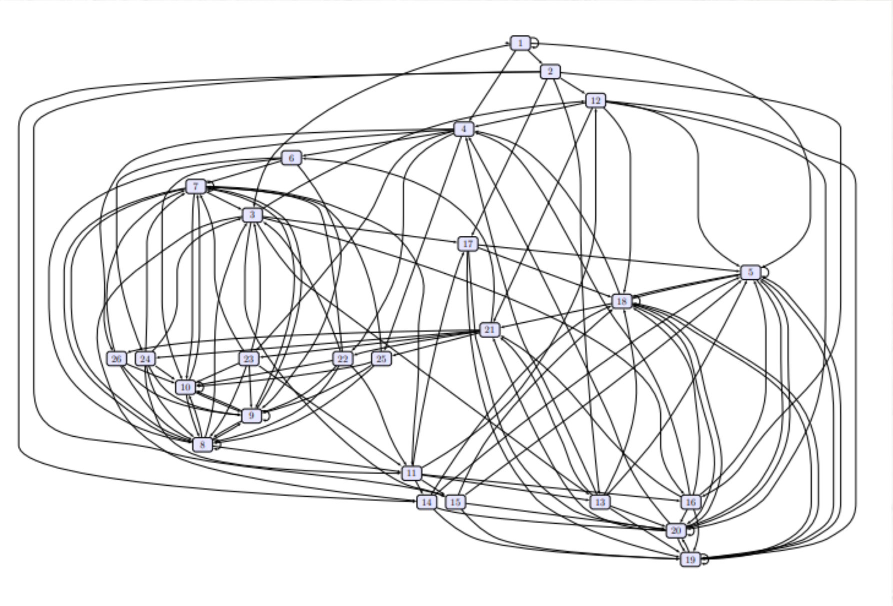
SYNTAX-GUIDED SYNTHESIS (SYGUS)



Good for data transformation problems

Not designed for control flow

REACTIVE SYNTHESIS



Good for control-flow problems

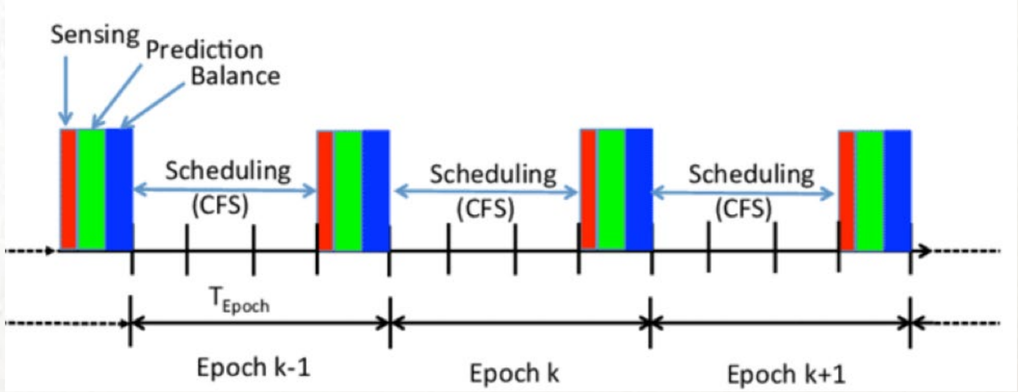
Not designed for data transformations

But even trivial programs have both data and control.

WHAT DOES IT MEAN TO HAVE BOTH DATA AND CONTROL?

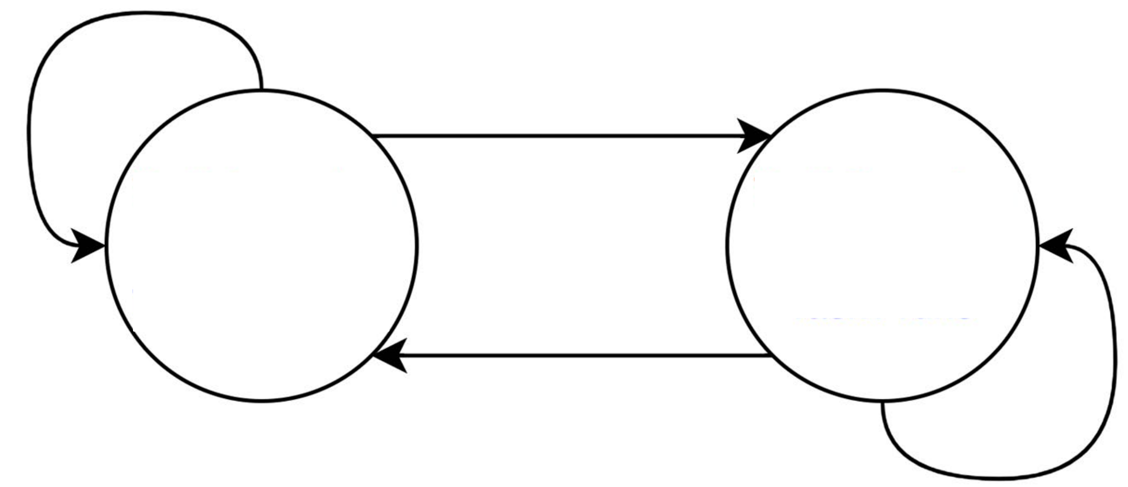
Linux Completely Fair Scheduler

- Runs the task
  "that has run for the least amount of time"
- "Time" is weighted
  - 1 µs of prioritized task → 0.25 µs
  - 1 µs of a low-priority task → 5 µs
- Control:
  Enqueuing and dequeing tasks
- Data:
  Calculate how long each process has run



EVEN VAST SIMPLIFICATIONS CAN STILL HAVE DATA AND CONTROL

- Scheduler with two tasks
- Task 1 must run at least twice



- States:
  - Run task 1
  - Run task 2

- Data transformations:
  - Count number of executions

- Can we synthesize this?

WE HAVE A LANGUAGE TO SPECIFY IT...

- Scheduler with two tasks
- Task 1 must run at least twice

- States:
  - Run task 1
  - Run task 2

- Data transformations:
  - Count number of executions

Temporal Stream Logic Modulo Theories (TSL-MT)

$$
\begin{aligned}
\Box(& [\mathtt{task} \leftarrow\!\prec \mathtt{task1}] \lor [\mathtt{task} \leftarrow\!\prec \mathtt{task2}] \\
\land & [\mathtt{task} \leftarrow\!\prec \mathtt{task1}] \leftrightarrow [\mathtt{taskTime1} \leftarrow\!\prec \mathtt{add\ taskTime1}\ 1] \\
\land & [\mathtt{task} \leftarrow\!\prec \mathtt{task2}] \leftrightarrow [\mathtt{taskTime2} \leftarrow\!\prec \mathtt{add\ taskTime2}\ 1] \\
\land & \ \mathtt{eq\ taskTime1}\ 0 \rightarrow \Diamond(\mathtt{eq\ taskTime1}\ 2)\ )
\end{aligned}
$$

WE HAVE A LANGUAGE TO SPECIFY IT...

- Scheduler with two tasks
- Task 1 must run at least twice

- States:
  - Run task 1
  - Run task 2

- Data transformations:
  - Count number of executions

Temporal Stream Logic Modulo Theories (TSL-MT)

$$\begin{aligned} &\Box(\,[\mathtt{task} \leftarrowtail \mathtt{task1}] \;\vee\; [\mathtt{task} \leftarrowtail \mathtt{task2}] \\ &\wedge\; [\mathtt{task} \leftarrowtail \mathtt{task1}] \;\leftrightarrow\; [\mathtt{taskTime1} \leftarrowtail \mathtt{add\ taskTime1\ 1}] \\ &\wedge\; [\mathtt{task} \leftarrowtail \mathtt{task2}] \;\leftrightarrow\; [\mathtt{taskTime2} \leftarrowtail \mathtt{add\ taskTime2\ 1}] \\ &\wedge\; \mathtt{eq\ taskTime1\ 0} \;\rightarrow\; \Diamond(\mathtt{eq\ taskTime1\ 2})\,) \end{aligned}$$

Control

$$\begin{aligned} &\Box(\,[\mathtt{task} \leftarrowtail \mathtt{task1}] \;\vee\; [\mathtt{task} \leftarrowtail \mathtt{task2}] \\ &\wedge\; [\mathtt{task} \leftarrowtail \mathtt{task1}] \;\leftrightarrow\; [\mathtt{taskTime1} \leftarrowtail \mathtt{add\ taskTime1\ 1}] \\ &\wedge\; [\mathtt{task} \leftarrowtail \mathtt{task2}] \;\leftrightarrow\; [\mathtt{taskTime2} \leftarrowtail \mathtt{add\ taskTime2\ 1}] \\ &\wedge\; \mathtt{eq\ taskTime1\ 0} \;\rightarrow\; \Diamond(\mathtt{eq\ taskTime1\ 2})\,) \end{aligned}$$

FIRST, ADD THEORIES TO TSL TO GET TSL-MT...

- Scheduler with two tasks
- Task 1 must run at least twice

- States:
  - Run task 1
  - Run task 2

- Data transformations:
  - Count number of executions

Temporal Stream Logic Modulo Theories (TSL-MT)

$$
\begin{aligned}
\square(\ & [\mathtt{task} \leftarrowtail \mathtt{task1}] \vee [\mathtt{task} \leftarrowtail \mathtt{task2}] \\
\wedge\ & [\mathtt{task} \leftarrowtail \mathtt{task1}] \leftrightarrow [\mathtt{taskTime1} \leftarrowtail \mathtt{add\ taskTime1}\ 1] \\
\wedge\ & [\mathtt{task} \leftarrowtail \mathtt{task2}] \leftrightarrow [\mathtt{taskTime2} \leftarrowtail \mathtt{add\ taskTime2}\ 1] \\
\wedge\ & \mathtt{eq\ taskTime1}\ 0 \rightarrow \Diamond(\mathtt{eq\ taskTime1}\ 2)\ )
\end{aligned}
$$

Control

$$
\begin{aligned}
\square(\ & [\mathtt{task} \leftarrowtail \mathtt{task1}] \vee [\mathtt{task} \leftarrowtail \mathtt{task2}] \\
\wedge\ & [\mathtt{task} \leftarrowtail \mathtt{task1}] \leftrightarrow [\mathtt{taskTime1} \leftarrowtail \mathtt{add\ taskTime1}\ 1] \\
\wedge\ & [\mathtt{task} \leftarrowtail \mathtt{task2}] \leftrightarrow [\mathtt{taskTime2} \leftarrowtail \mathtt{add\ taskTime2}\ 1] \\
\wedge\ & \mathtt{eq\ taskTime1}\ 0 \rightarrow \Diamond(\mathtt{eq\ taskTime1}\ 2)\ )
\end{aligned}
$$

Data

$$
\begin{aligned}
\square(\ & [\mathtt{task} \leftarrowtail \mathtt{task1}] \vee [\mathtt{task} \leftarrowtail \mathtt{task2}] \\
\wedge\ & [\mathtt{task} \leftarrowtail \mathtt{task1}] \leftrightarrow [\mathtt{taskTime1} \leftarrowtail \mathtt{add\ taskTime1}\ 1] \\
\wedge\ & [\mathtt{task} \leftarrowtail \mathtt{task2}] \leftrightarrow [\mathtt{taskTime2} \leftarrowtail \mathtt{add\ taskTime2}\ 1] \\
\wedge\ & \mathtt{eq\ taskTime1}\ 0 \rightarrow \Diamond(\mathtt{eq\ taskTime1}\ 2)\ )
\end{aligned}
$$

THEN, WE NEED A SYNTHESIS PROCEDURE FOR TSL-MT...

- Reactive (TSL) Synthesis can synthesize "control"
- All functions are uninterpreted!
- SyGuS can synthesize "data"
- But it can't generate state machines!

Temporal Stream Logic Modulo Theories (TSL-MT)

$$
\begin{aligned}
\square(\ & [\mathtt{task} \leftarrowtail \mathtt{task1}] \ \vee\ [\mathtt{task} \leftarrowtail \mathtt{task2}] \\
\wedge\ & [\mathtt{task} \leftarrowtail \mathtt{task1}] \ \leftrightarrow\ [\mathtt{taskTime1} \leftarrowtail \mathtt{add\ taskTime1}\ 1] \\
\wedge\ & [\mathtt{task} \leftarrowtail \mathtt{task2}] \ \leftrightarrow\ [\mathtt{taskTime2} \leftarrowtail \mathtt{add\ taskTime2}\ 1] \\
\wedge\ & \mathtt{eq\ taskTime1}\ 0 \ \rightarrow\ \Diamond(\mathtt{eq\ taskTime1}\ 2)\ )
\end{aligned}
$$

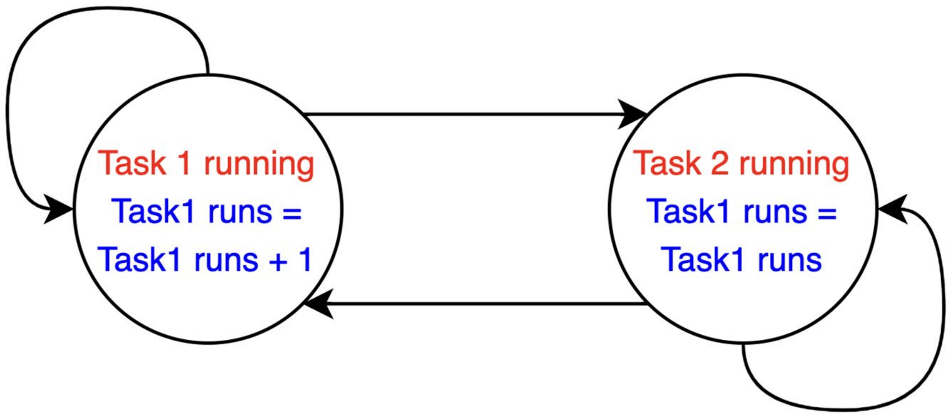
Control

$$
\begin{aligned}
\square(& [\mathtt{task} \leftarrowtail \mathtt{task1}] \ \vee\ [\mathtt{task} \leftarrowtail \mathtt{task2}] \\
\wedge\ & [\mathtt{task} \leftarrowtail \mathtt{task1}] \ \leftrightarrow\ [\mathtt{taskTime1} \leftarrowtail \mathtt{add\ taskTime1}\ 1] \\
\wedge\ & [\mathtt{task} \leftarrowtail \mathtt{task2}] \ \leftrightarrow\ [\mathtt{taskTime2} \leftarrowtail \mathtt{add\ taskTime2}\ 1] \\
\wedge\ & \mathtt{eq\ taskTime1}\ 0 \ \rightarrow \Diamond(\mathtt{eq\ taskTime1}\ 2)\ )
\end{aligned}
$$

Data

$$
\begin{aligned}
\square(& [\mathtt{task} \leftarrowtail \mathtt{task1}] \ \vee\ [\mathtt{task} \leftarrowtail \mathtt{task2}] \\
\wedge\ & [\mathtt{task} \leftarrowtail \mathtt{task1}] \ \leftrightarrow\ [\mathtt{taskTime1} \leftarrowtail \mathtt{add\ taskTime1}\ 1] \\
\wedge\ & [\mathtt{task} \leftarrowtail \mathtt{task2}] \ \leftrightarrow\ [\mathtt{taskTime2} \leftarrowtail \mathtt{add\ taskTime2}\ 1] \\
\wedge\ & \mathtt{eq\ taskTime1}\ 0 \ \rightarrow \Diamond(\mathtt{eq\ taskTime1}\ 2)\ )
\end{aligned}
$$

HOW TO SYNTHESIZE?



Temporal Stream Logic Modulo Theories (TSL-MT)

$$\begin{aligned}&\square(\ [\mathtt{task} \leftarrowtail \mathtt{task1}] \ \vee\ [\mathtt{task} \leftarrowtail \mathtt{task2}]\\ &\wedge\ [\mathtt{task} \leftarrowtail \mathtt{task1}] \leftrightarrow [\mathtt{taskTime1} \leftarrowtail \mathtt{add\ taskTime1}\ 1]\\ &\wedge\ [\mathtt{task} \leftarrowtail \mathtt{task2}] \leftrightarrow [\mathtt{taskTime2} \leftarrowtail \mathtt{add\ taskTime2}\ 1]\\ &\wedge\ \mathtt{eq\ taskTime1}\ 0 \rightarrow \Diamond(\mathtt{eq\ taskTime1}\ 2)\ )\end{aligned}$$

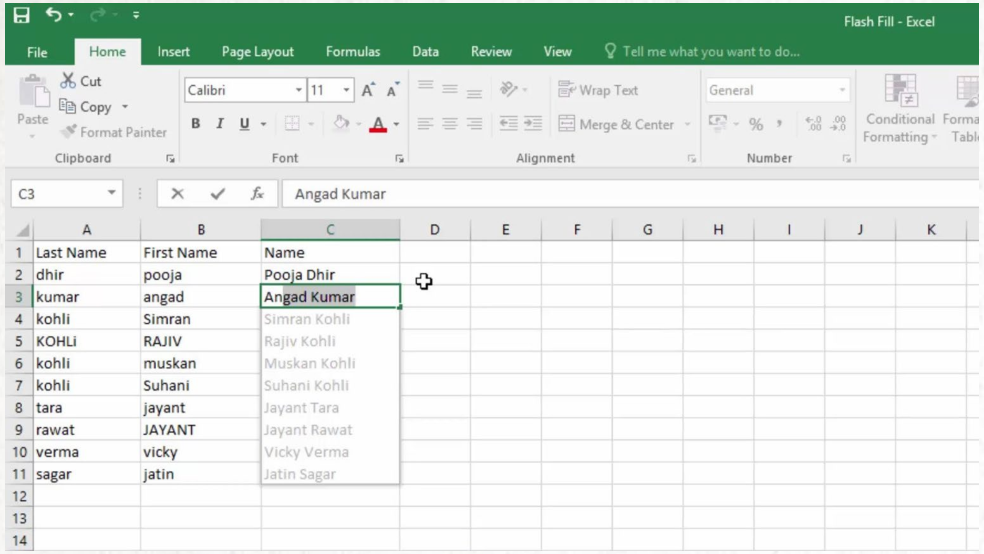
Control

$$\begin{aligned}&\square(\ [\mathtt{task} \leftarrowtail \mathtt{task1}] \ \vee\ [\mathtt{task} \leftarrowtail \mathtt{task2}]\\ &\wedge\ [\mathtt{task} \leftarrowtail \mathtt{task1}] \leftrightarrow [\mathtt{taskTime1} \leftarrowtail \mathtt{add\ taskTime1}\ 1]\\ &\wedge\ [\mathtt{task} \leftarrowtail \mathtt{task2}] \leftrightarrow [\mathtt{taskTime2} \leftarrowtail \mathtt{add\ taskTime2}\ 1]\\ &\wedge\ \mathtt{eq\ taskTime1}\ 0 \rightarrow \Diamond(\mathtt{eq\ taskTime1}\ 2)\ )\end{aligned}$$

Data

$$\begin{aligned}&\square(\ [\mathtt{task} \leftarrowtail \mathtt{task1}] \ \vee\ [\mathtt{task} \leftarrowtail \mathtt{task2}]\\ &\wedge\ [\mathtt{task} \leftarrowtail \mathtt{task1}] \leftrightarrow [\mathtt{taskTime1} \leftarrowtail \mathtt{add\ taskTime1}\ 1]\\ &\wedge\ [\mathtt{task} \leftarrowtail \mathtt{task2}] \leftrightarrow [\mathtt{taskTime2} \leftarrowtail \mathtt{add\ taskTime2}\ 1]\\ &\wedge\ \mathtt{eq\ taskTime1}\ 0 \rightarrow \Diamond(\mathtt{eq\ taskTime1}\ 2)\ )\end{aligned}$$

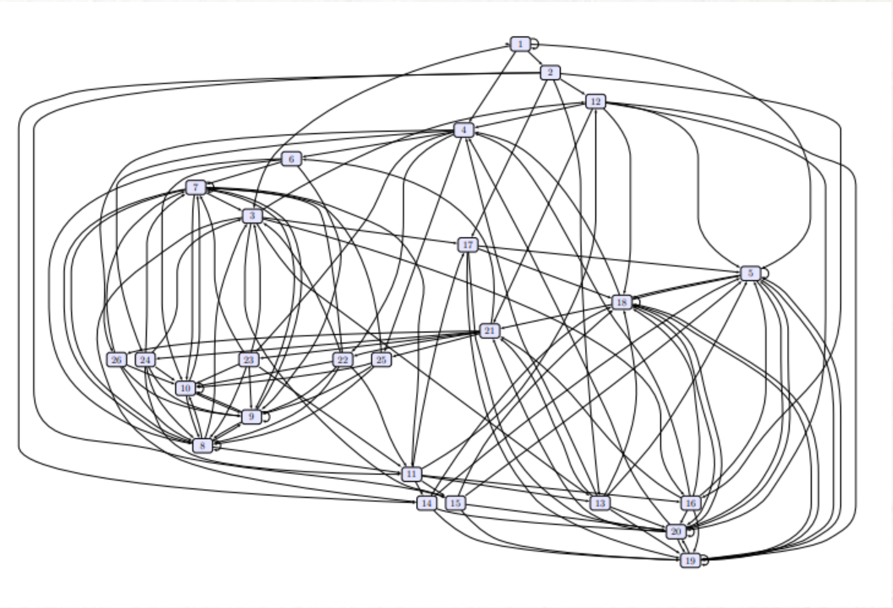
SYNTAX-GUIDED SYNTHESIS (SYGUS)



Good for data transformation problems

Does not have control flow

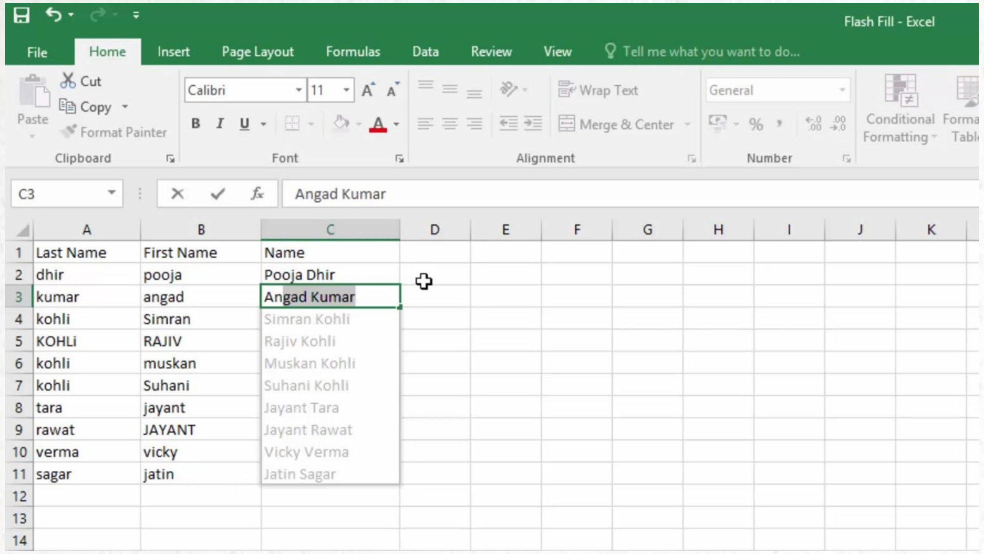
REACTIVE SYNTHESIS



Good for control-flow problems

Does not have data transformations

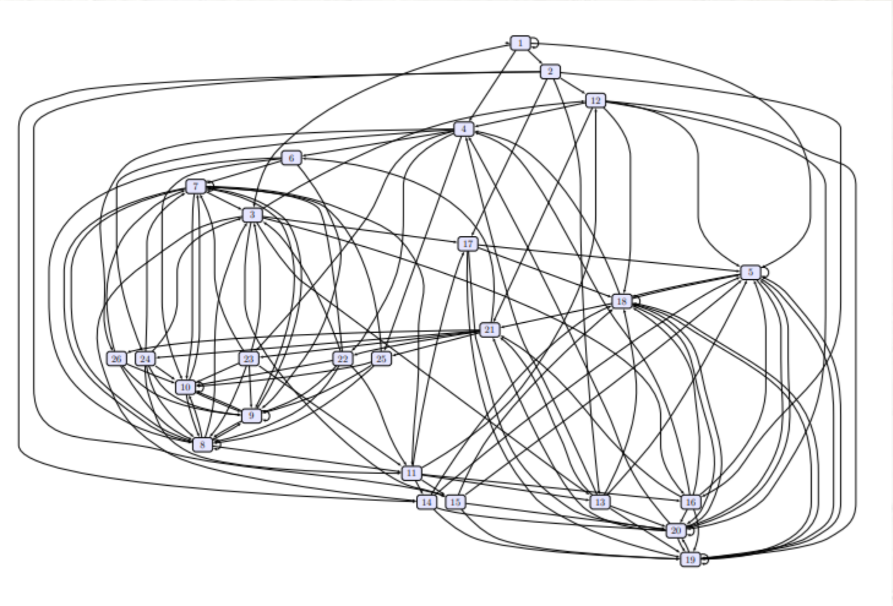
SYNTAX-GUIDED SYNTHESIS (SYGUS)



Good for data transformation problems

Does not have control flow

REACTIVE SYNTHESIS



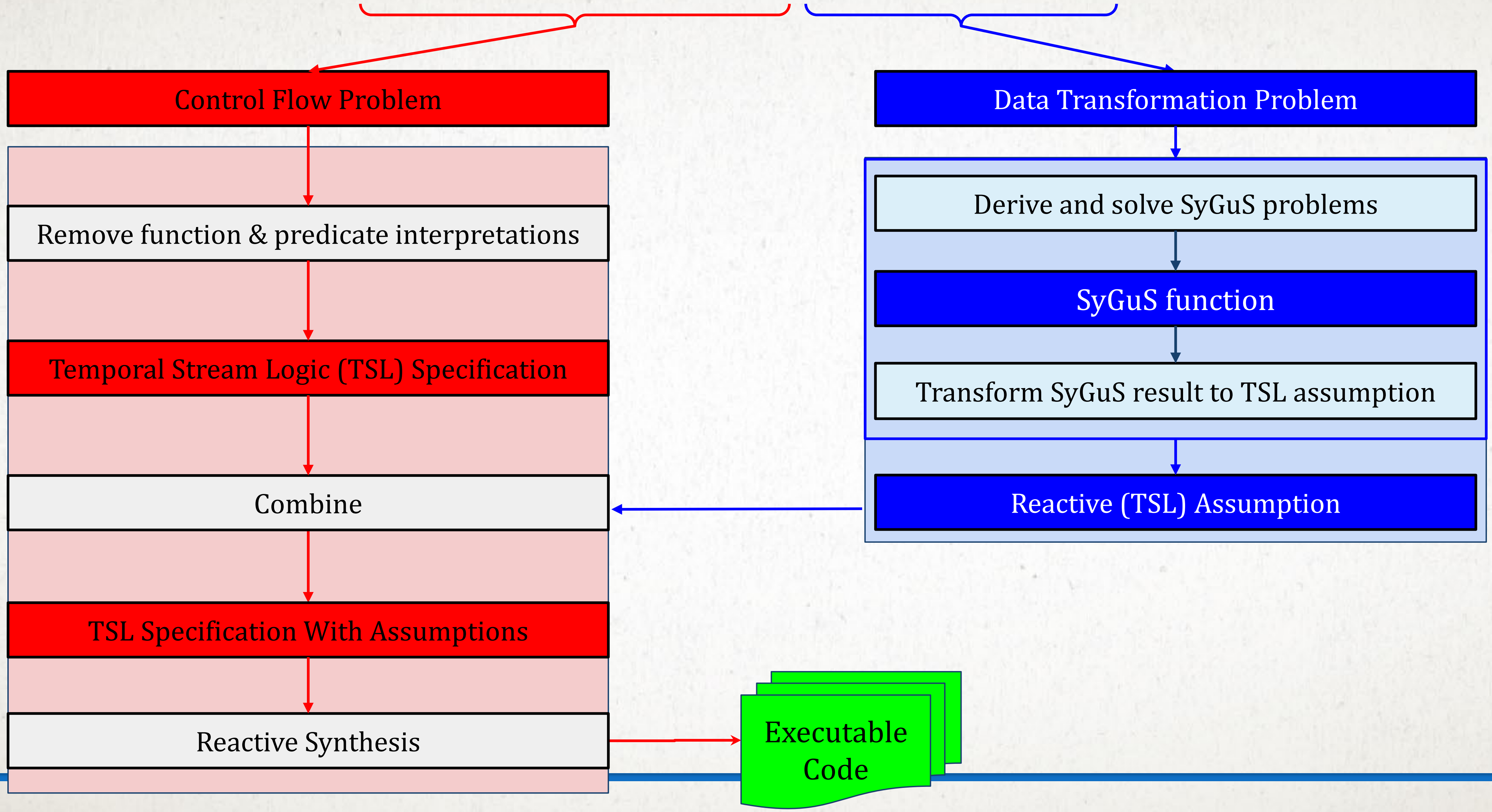
Good for control-flow problems

Does not have data transformations

REACTIVE SYNTHESIS AND SYNTAX-GUIDED SYNTHESIS CAN BE FRIENDS!

- SyGuS can "teach" Reactive Synthesis how to solve data transformation problems
- Reactive Synthesis can then handle control flow problems
- **Use both to solve problems they excel at, then communicate!**
- Can synthesize a simple linux scheduler…
- But also C code for the Linux Completely Fair Scheduler!

Synthesizing a … Temporal Stream Logic Modulo Theories Specification



SYNTHESIZING THE TWO-TASK SCHEDULER

- Scheduler with two tasks
- Task 1 must run at least twice

Original Specification

$$\begin{aligned}
\Box(\ & [\mathtt{task} \leftarrowtail \mathtt{task1}] \ \vee \ [\mathtt{task} \leftarrowtail \mathtt{task2}] \\
\wedge\ & [\mathtt{task} \leftarrowtail \mathtt{task1}] \ \leftrightarrow \ [\mathtt{taskTime1} \leftarrowtail \mathtt{add\ taskTime1\ 1}] \\
\wedge\ & [\mathtt{task} \leftarrowtail \mathtt{task2}] \ \leftrightarrow \ [\mathtt{taskTime2} \leftarrowtail \mathtt{add\ taskTime2\ 1}] \\
\wedge\ & \mathtt{eq\ taskTime1\ 0} \ \rightarrow \ \Diamond(\mathtt{eq\ taskTime1\ 2})\ )
\end{aligned}$$

Control

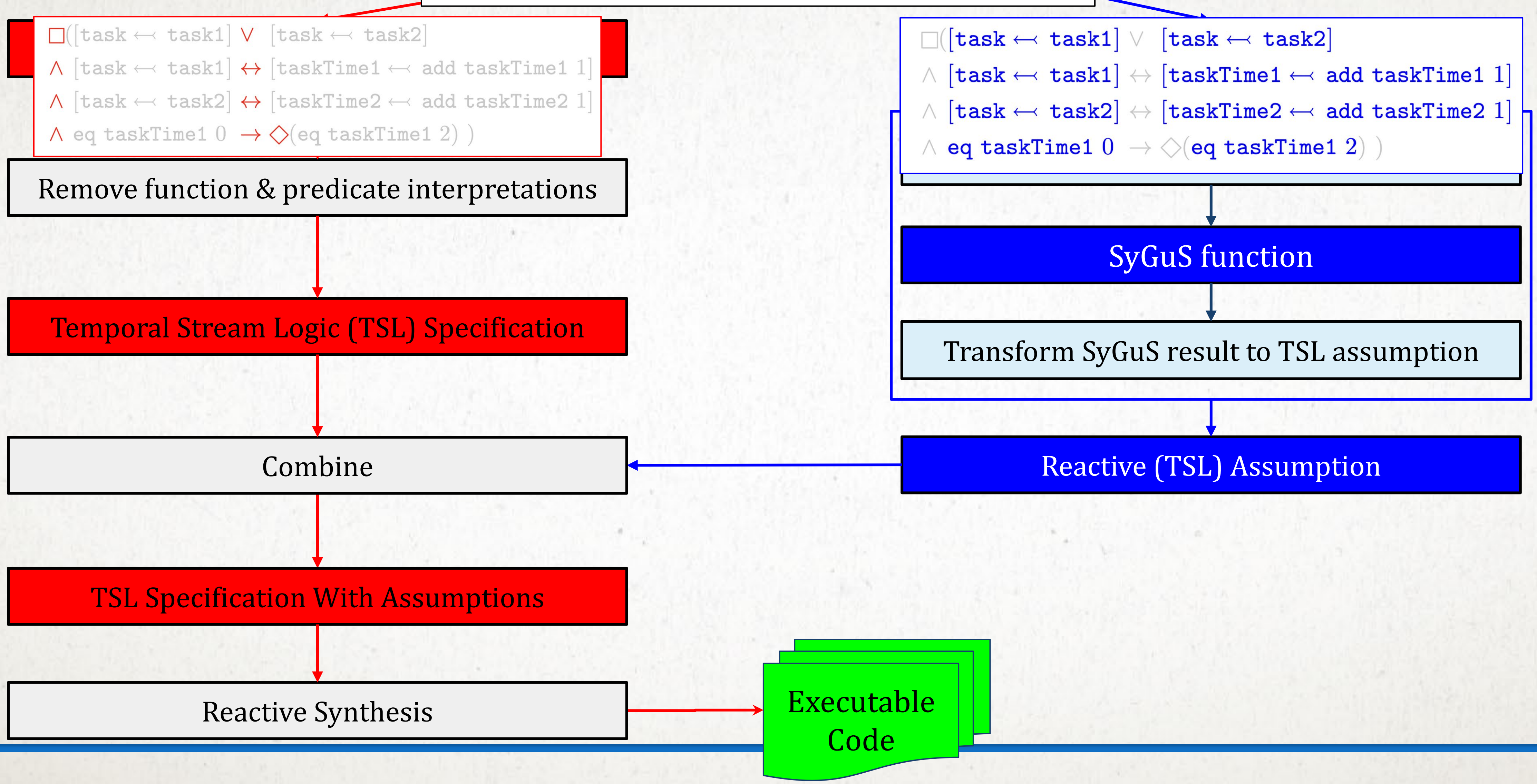
$$\begin{aligned}
\Box(\ & [\mathtt{task} \leftarrowtail \mathtt{task1}] \ \vee \ [\mathtt{task} \leftarrowtail \mathtt{task2}] \\
\wedge\ & [\mathtt{task} \leftarrowtail \mathtt{task1}] \ \leftrightarrow \ [\mathtt{taskTime1} \leftarrowtail \mathtt{add\ taskTime1\ 1}] \\
\wedge\ & [\mathtt{task} \leftarrowtail \mathtt{task2}] \ \leftrightarrow \ [\mathtt{taskTime2} \leftarrowtail \mathtt{add\ taskTime2\ 1}] \\
\wedge\ & \mathtt{eq\ taskTime1\ 0} \ \rightarrow \ \Diamond(\mathtt{eq\ taskTime1\ 2})\ )
\end{aligned}$$

Data

$$\begin{aligned}
\Box(\ & [\mathtt{task} \leftarrowtail \mathtt{task1}] \ \vee \ [\mathtt{task} \leftarrowtail \mathtt{task2}] \\
\wedge\ & [\mathtt{task} \leftarrowtail \mathtt{task1}] \ \leftrightarrow \ [\mathtt{taskTime1} \leftarrowtail \mathtt{add\ taskTime1\ 1}] \\
\wedge\ & [\mathtt{task} \leftarrowtail \mathtt{task2}] \ \leftrightarrow \ [\mathtt{taskTime2} \leftarrowtail \mathtt{add\ taskTime2\ 1}] \\
\wedge\ & \mathtt{eq\ taskTime1\ 0} \ \rightarrow \ \Diamond(\mathtt{eq\ taskTime1\ 2})\ )
\end{aligned}$$

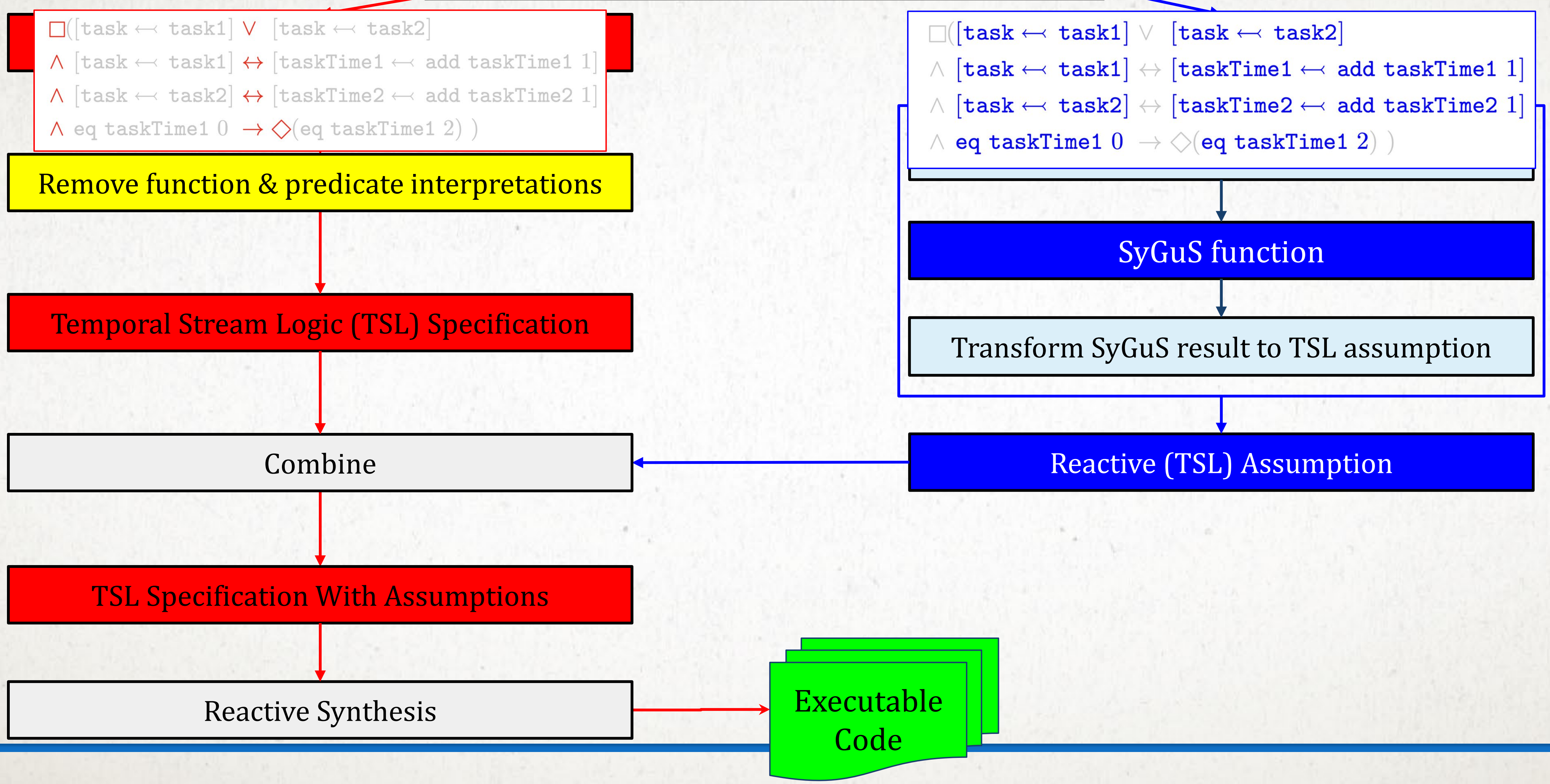
Synthesizing a ... Temp... Specification

$$\Box(\,[\texttt{task} \leftarrow \texttt{task1}] \lor [\texttt{task} \leftarrow \texttt{task2}]$$
$$\land\ [\texttt{task} \leftarrow \texttt{task1}] \leftrightarrow [\texttt{taskTime1} \leftarrow \texttt{add taskTime1 } 1]$$
$$\land\ [\texttt{task} \leftarrow \texttt{task2}] \leftrightarrow [\texttt{taskTime2} \leftarrow \texttt{add taskTime2 } 1]$$
$$\land\ \texttt{eq taskTime1 } 0 \rightarrow \Diamond(\texttt{eq taskTime1 } 2)\,)$$



Synthesizing a ... Temp... Specification

$$
\begin{aligned}
&\Box(\, [\mathtt{task} \leftarrowtail \mathtt{task1}] \;\vee\; [\mathtt{task} \leftarrowtail \mathtt{task2}] \\
&\wedge\; [\mathtt{task} \leftarrowtail \mathtt{task1}] \;\leftrightarrow\; [\mathtt{taskTime1} \leftarrowtail \mathtt{add\ taskTime1\ 1}] \\
&\wedge\; [\mathtt{task} \leftarrowtail \mathtt{task2}] \;\leftrightarrow\; [\mathtt{taskTime2} \leftarrowtail \mathtt{add\ taskTime2\ 1}] \\
&\wedge\; \mathtt{eq\ taskTime1\ 0} \;\rightarrow\; \Diamond(\mathtt{eq\ taskTime1\ 2})\ )
\end{aligned}
$$



REMOVING FUNCTION&PREDICATE INTERPRETATIONS FROM TSL-MT

- Temporal Stream Logic Modulo Theories to…
  Temporal Stream Logic

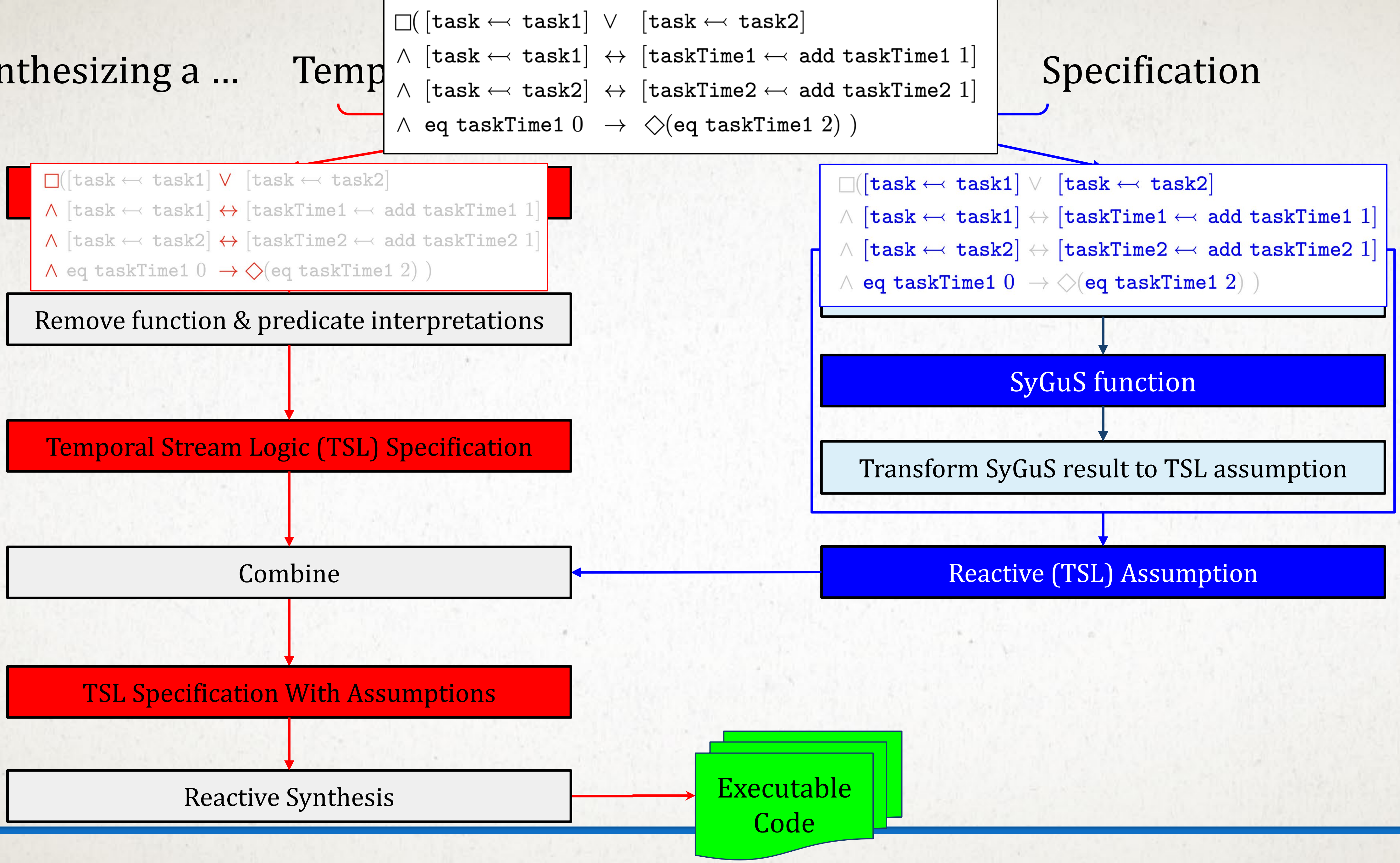
$$
\begin{aligned}
\square(\ &[\texttt{task} \leftarrowtail \texttt{task1}] \ \vee\ [\texttt{task} \leftarrowtail \texttt{task2}]\\
\wedge\ &[\texttt{task} \leftarrowtail \texttt{task1}] \leftrightarrow [\texttt{taskTime1} \leftarrowtail \texttt{add taskTime1 } 1]\\
\wedge\ &[\texttt{task} \leftarrowtail \texttt{task2}] \leftrightarrow [\texttt{taskTime2} \leftarrowtail \texttt{add taskTime2 } 1]\\
\wedge\ &\texttt{eq taskTime1 } 0 \ \rightarrow\ \diamond(\texttt{eq taskTime1 } 2)\ )
\end{aligned}
$$

→

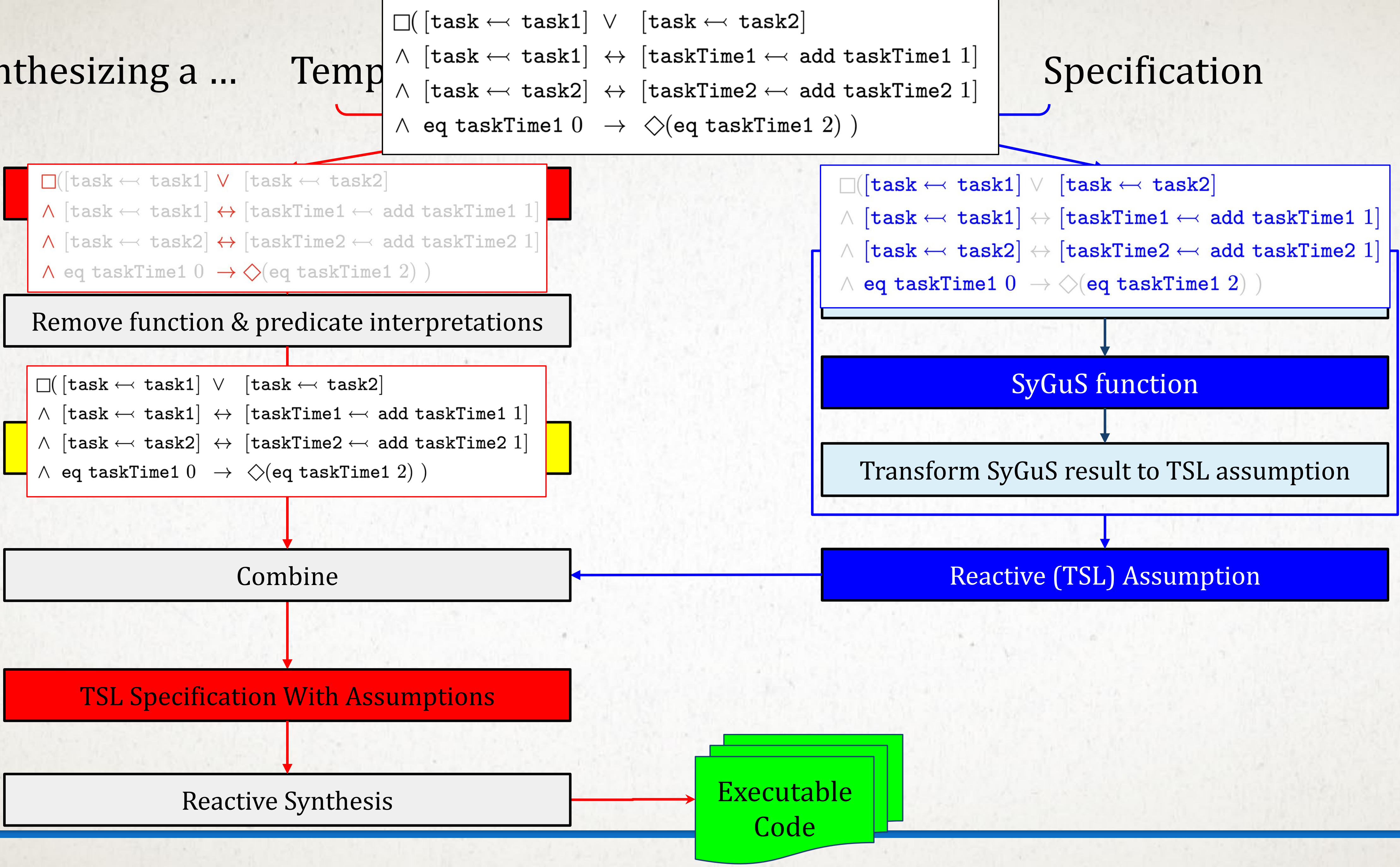
$$
\begin{aligned}
\square(\ &[\texttt{task} \leftarrowtail \texttt{task1}] \ \vee\ [\texttt{task} \leftarrowtail \texttt{task2}]\\
\wedge\ &[\texttt{task} \leftarrowtail \texttt{task1}] \leftrightarrow [\texttt{taskTime1} \leftarrowtail \texttt{add taskTime1 } 1]\\
\wedge\ &[\texttt{task} \leftarrowtail \texttt{task2}] \leftrightarrow [\texttt{taskTime2} \leftarrowtail \texttt{add taskTime2 } 1]\\
\wedge\ &\texttt{eq taskTime1 } 0 \ \rightarrow\ \diamond(\texttt{eq taskTime1 } 2)\ )
\end{aligned}
$$

- Removes interpretations of `eq` and `add`: make it a pure control flow problem
- But it now doesn't know that `0 + 1 + 1 = 2`

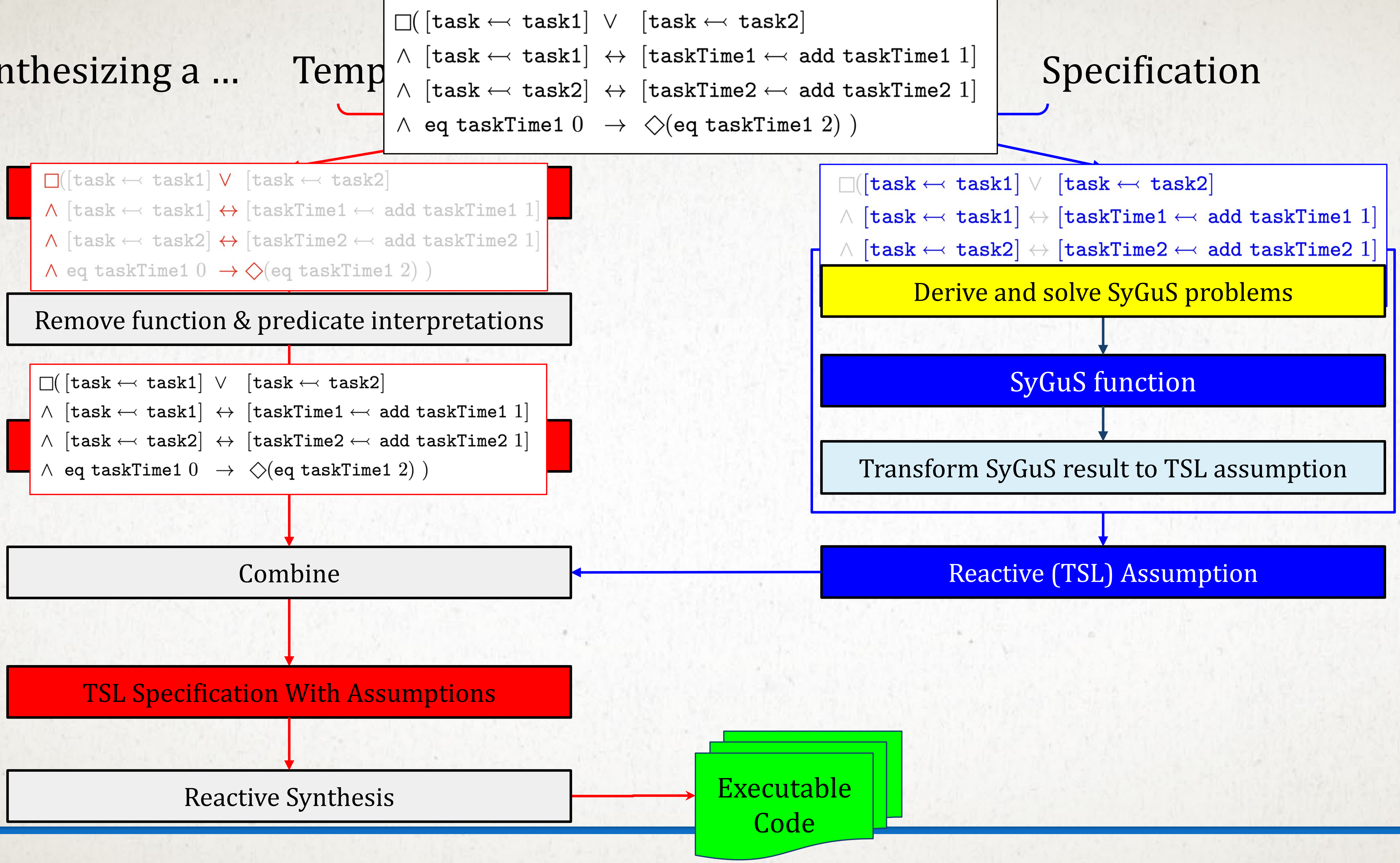
Synthesizing a ... Temp Specification



Synthesizing a ... Temp ... Specification



Synthesizing a … Temp… Specification



SYNTAX-GUIDED SYNTHESIS (SYGUS)

Semantic Constraint

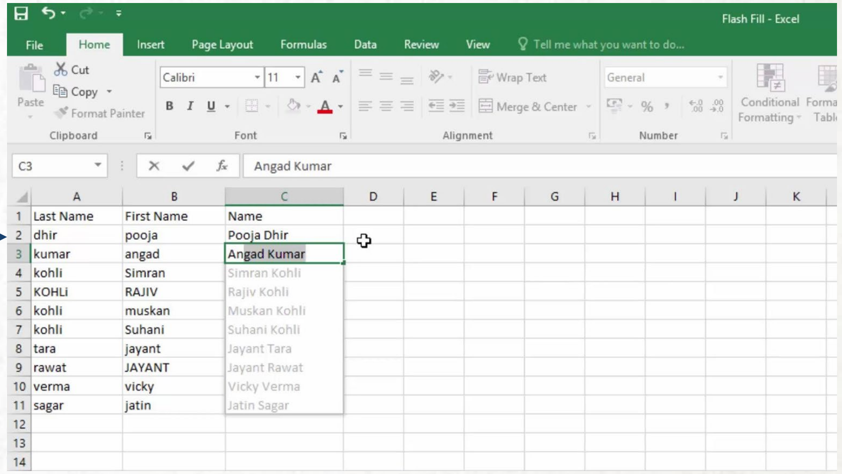
Input v_1	*Output*
Dr. Eran Yahav	*Yahav, E.*
Prof. Kathleen S. Fisher	*Fisher, K.*
Bill Gates, Sr.	*Gates, B.*
George Ciprian Necula	*Necula, G.*
Ken McMillan, II	*McMillan, K.*

Syntactic Constraint

$$
\begin{array}{rcl}
\text{String expr } P & := & \texttt{Switch}((b_1, e_1), \cdot\cdot, (b_n, e_n)) \\
\text{Bool b} & := & d_1 \vee \cdot\cdot \vee d_n \\
\text{Conjunct d} & := & \pi_1 \wedge \cdot\cdot \wedge \pi_n \\
\text{Predicate } \pi & := & \texttt{Match}(v_i, r, k) \mid \neg\, \texttt{Match}(v_i, r, k) \\
\text{Trace expr e} & := & \texttt{Concatenate}(f_1, \cdot\cdot, f_n) \\
\text{Atomic expr f} & := & \texttt{SubStr}(v_i, p_1, p_2) \\
 & \mid & \texttt{ConstStr}(s) \\
 & \mid & \texttt{Loop}(\lambda w : e) \\
\text{Position p} & := & \texttt{CPos}(k) \mid \texttt{Pos}(r_1, r_2, c) \\
\text{Integer expr c} & := & k \mid k_1 w + k_2 \\
\text{Regular Expression r} & := & \texttt{TokenSeq}(T_1, \cdot\cdot, T_m) \\
\text{Token T} & := & C+ \quad \mid \quad [\neg C]+ \\
 & \mid & \text{SpecialToken}
\end{array}
$$

Syntax-Guided Synthesis

Synthesized Program



DERIVE AND SOLVE SYGUS PROBLEMS

Original specification

$\Box($ [task $\leftarrow$ task1] $\vee$ [task $\leftarrow$ task2]
$\wedge$ [task $\leftarrow$ task1] $\leftrightarrow$ [taskTime1 $\leftarrow$ add taskTime1 1]
$\wedge$ [task $\leftarrow$ task2] $\leftrightarrow$ [taskTime2 $\leftarrow$ add taskTime2 1]
$\wedge$ eq taskTime1 0 $\rightarrow$ $\Diamond$(eq taskTime1 2) $)$

Semantic Constraint

Pre-condition	Program	Post-condition
eq taskTime1 0	$\mathcal{S}$	eq taskTime1 2

Syntactic Constraint

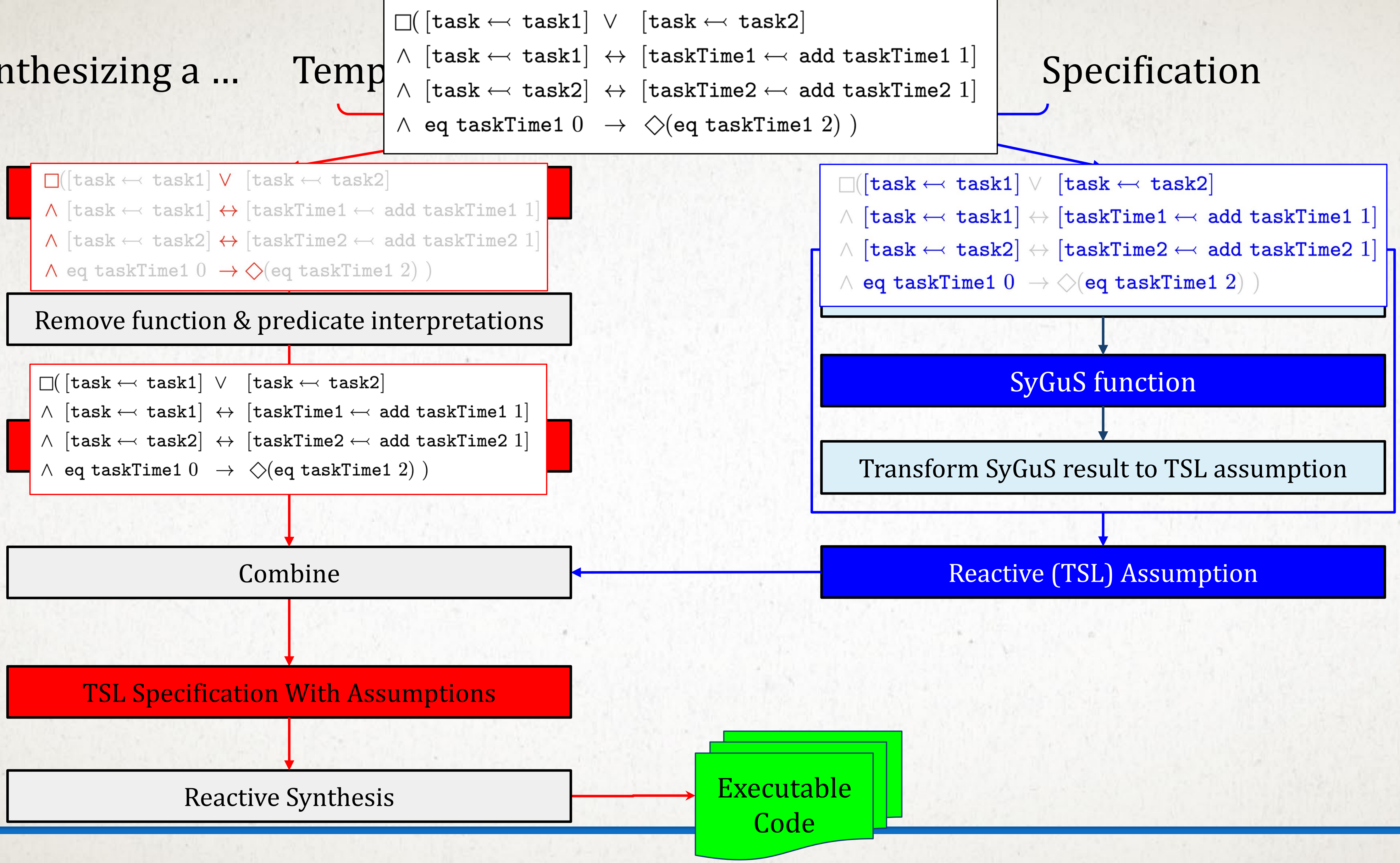
$\mathcal{S}$ ::= add $\mathcal{S}$ 1 | taskTime1

Syntax-Guided Synthesis

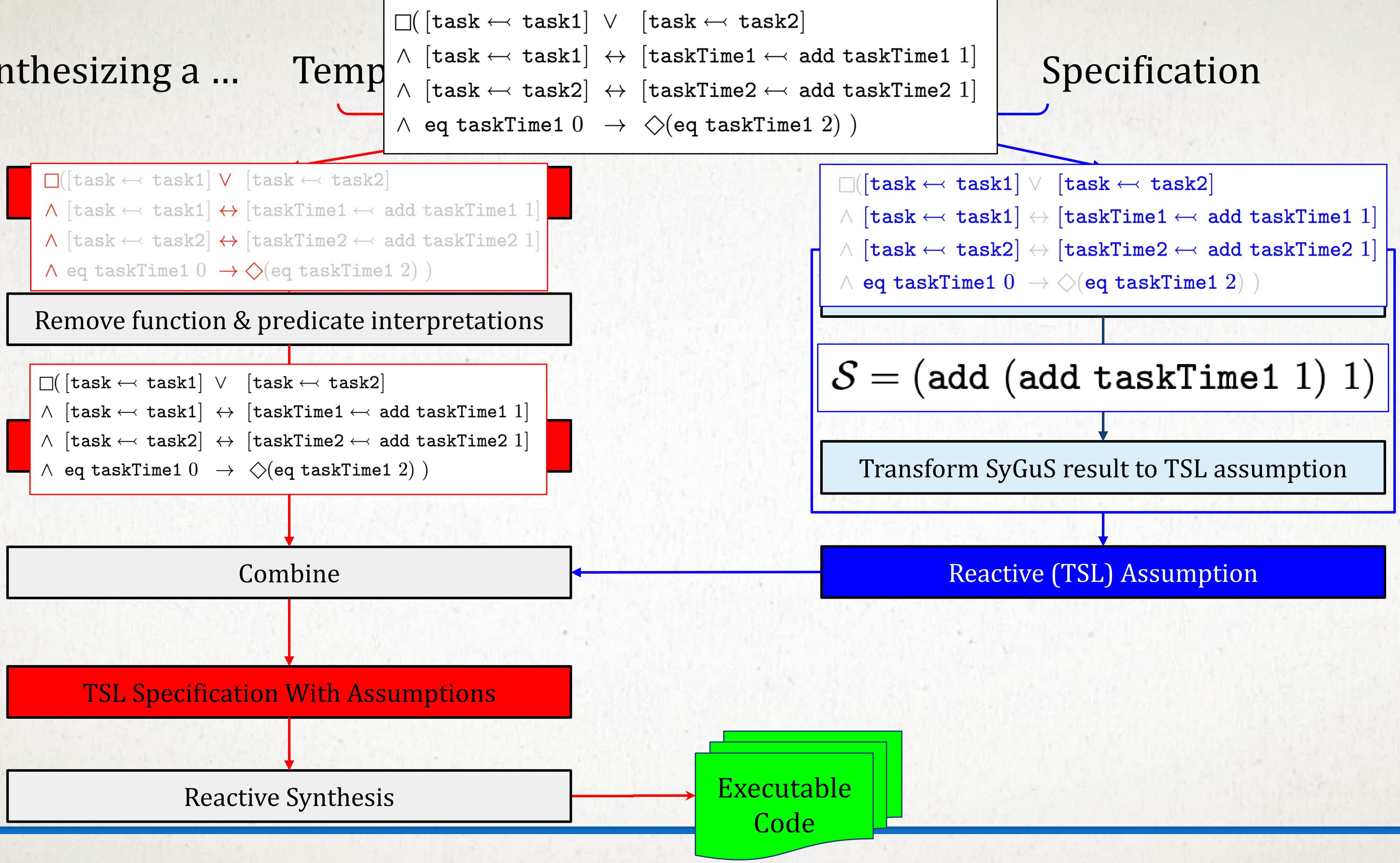
SyGuS-synthesized function

$\mathcal{S}$ = (add (add taskTime1 1) 1)

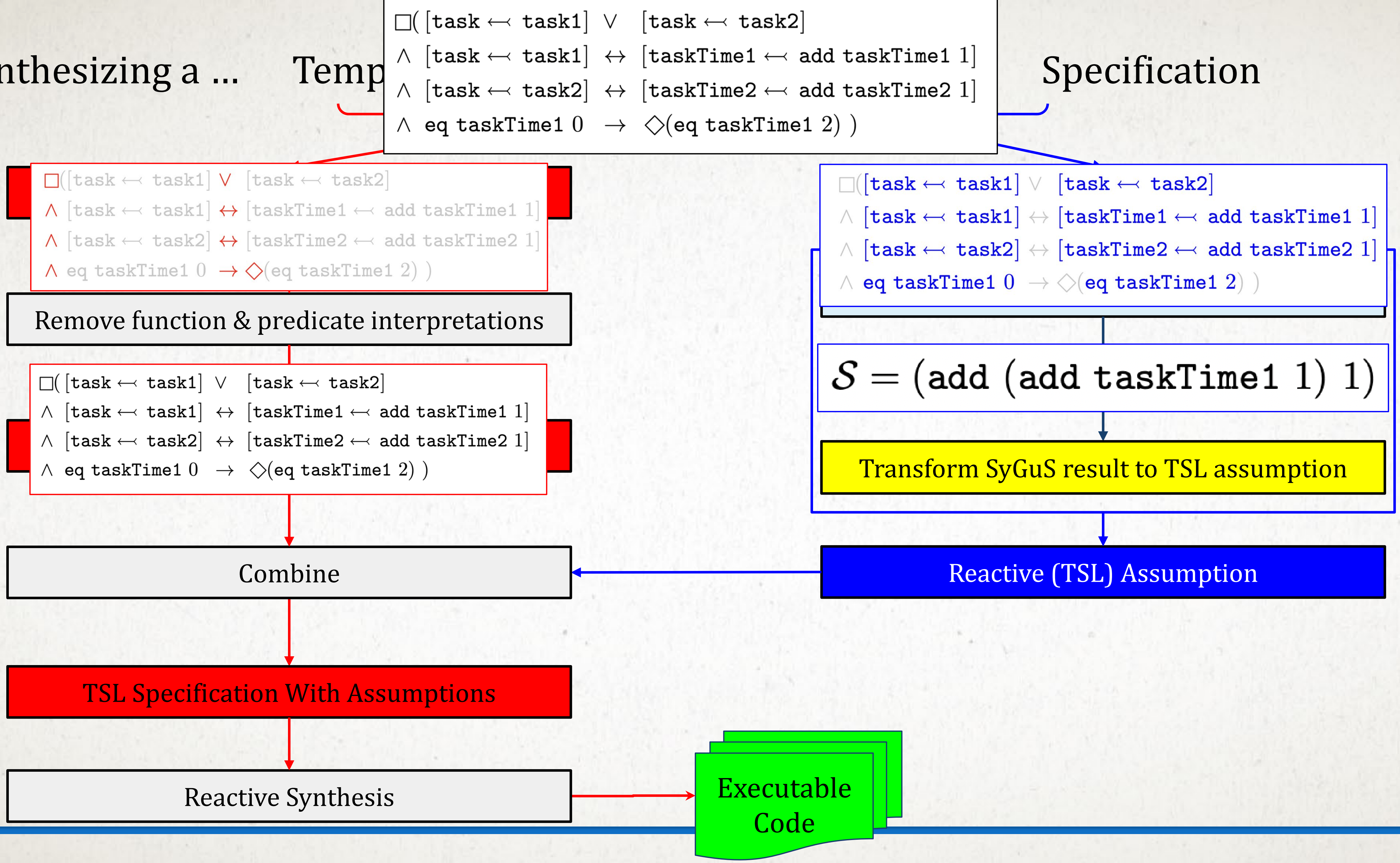
Synthesizing a … Temp Specification



Synthesizing a … Temp Specification



Synthesizing a ... Temp Specification



HOW TO COMMUNICATE SYGUS RESULT TO REACTIVE SYNTHESIS?

Original specification

$$\square(\,[\mathtt{task} \leftarrowtail \mathtt{task1}] \lor [\mathtt{task} \leftarrowtail \mathtt{task2}]$$
$$\land\ [\mathtt{task} \leftarrowtail \mathtt{task1}] \leftrightarrow [\mathtt{taskTime1} \leftarrowtail \mathtt{add\ taskTime1}\ 1]$$
$$\land\ [\mathtt{task} \leftarrowtail \mathtt{task2}] \leftrightarrow [\mathtt{taskTime2} \leftarrowtail \mathtt{add\ taskTime2}\ 1]$$
$$\land\ \mathtt{eq\ taskTime1}\ 0 \rightarrow \Diamond(\mathtt{eq\ taskTime1}\ 2)\,)$$

SyGuS-synthesized function

$$\mathcal{S} = (\mathtt{add}\ (\mathtt{add\ taskTime1}\ 1)\ 1)$$

Solution: Transform each “level” of the AST into a timestep of computation

TRANSFORMING SYGUS RESULT TO TEMPORAL STREAM LOGIC (TSL)

Original specification

$$\Box(\,[\texttt{task} \leftarrowtail \texttt{task1}] \vee [\texttt{task} \leftarrowtail \texttt{task2}]$$
$$\wedge\ [\texttt{task} \leftarrowtail \texttt{task1}] \leftrightarrow [\texttt{taskTime1} \leftarrowtail \texttt{add taskTime1 1}]$$
$$\wedge\ [\texttt{task} \leftarrowtail \texttt{task2}] \leftrightarrow [\texttt{taskTime2} \leftarrowtail \texttt{add taskTime2 1}]$$
$$\wedge\ \texttt{eq taskTime1 0} \rightarrow \Diamond(\texttt{eq taskTime1 2})\,)$$

SyGuS-synthesized function

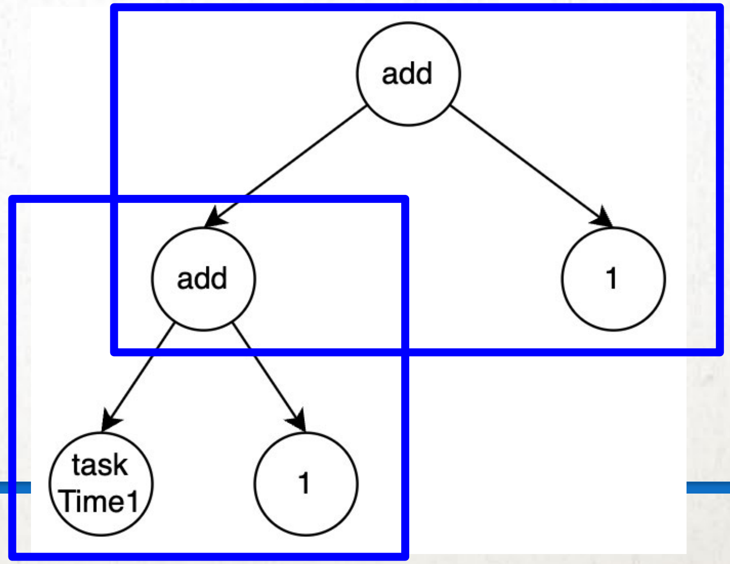
$$\mathcal{S} = (\texttt{add}\ (\texttt{add taskTime1 1})\ 1)$$

SyGuS Result as TSL Assumption

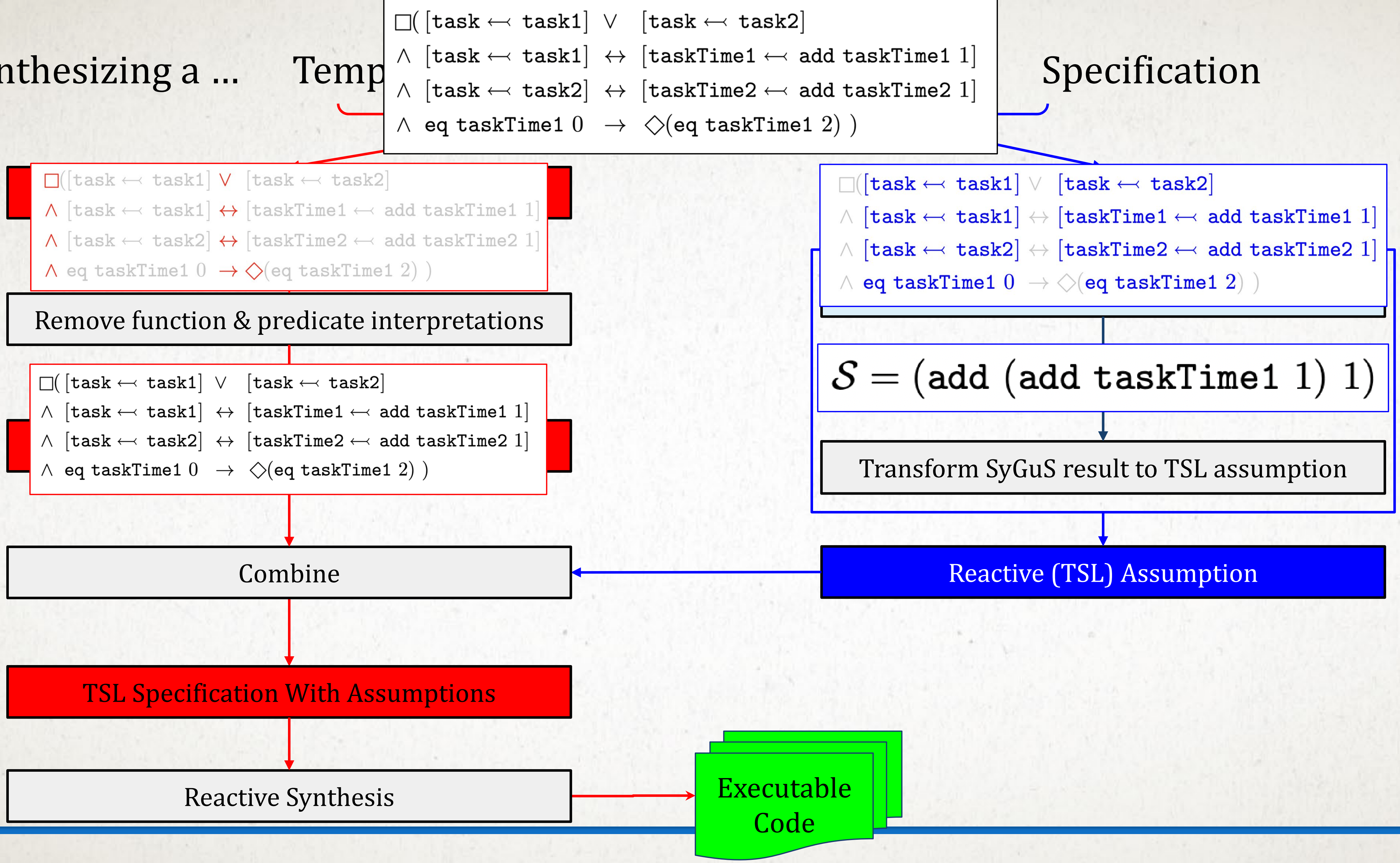
Pre-condition	Program	Post-condition
eq taskTime1 0	$\mathcal{S}$	eq taskTime1 2

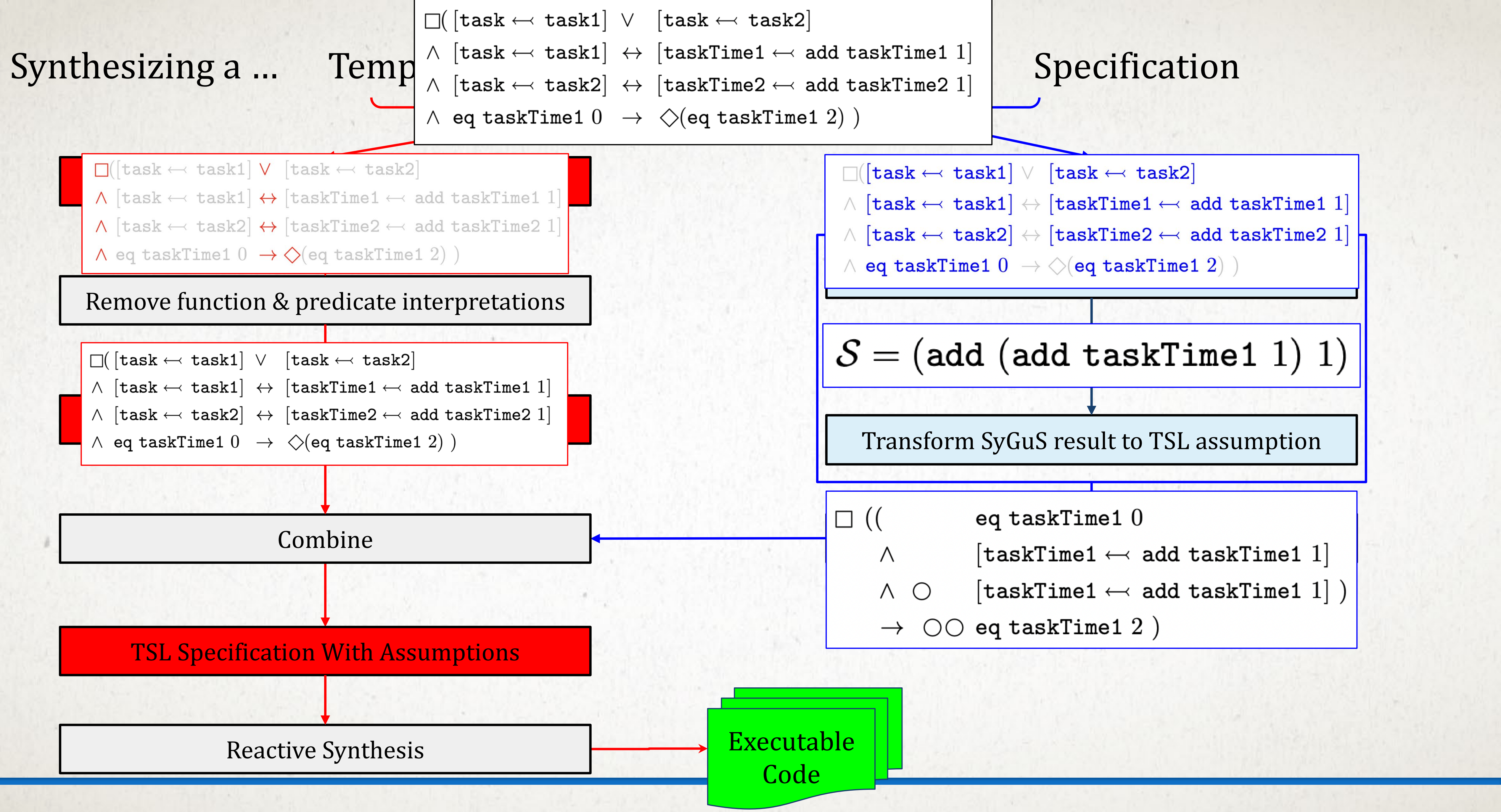
$$\Box\ ((\quad \text{Pre-condition} \quad \mathcal{S} \quad \rightarrow \quad \text{Post-condition}\ )$$

AST



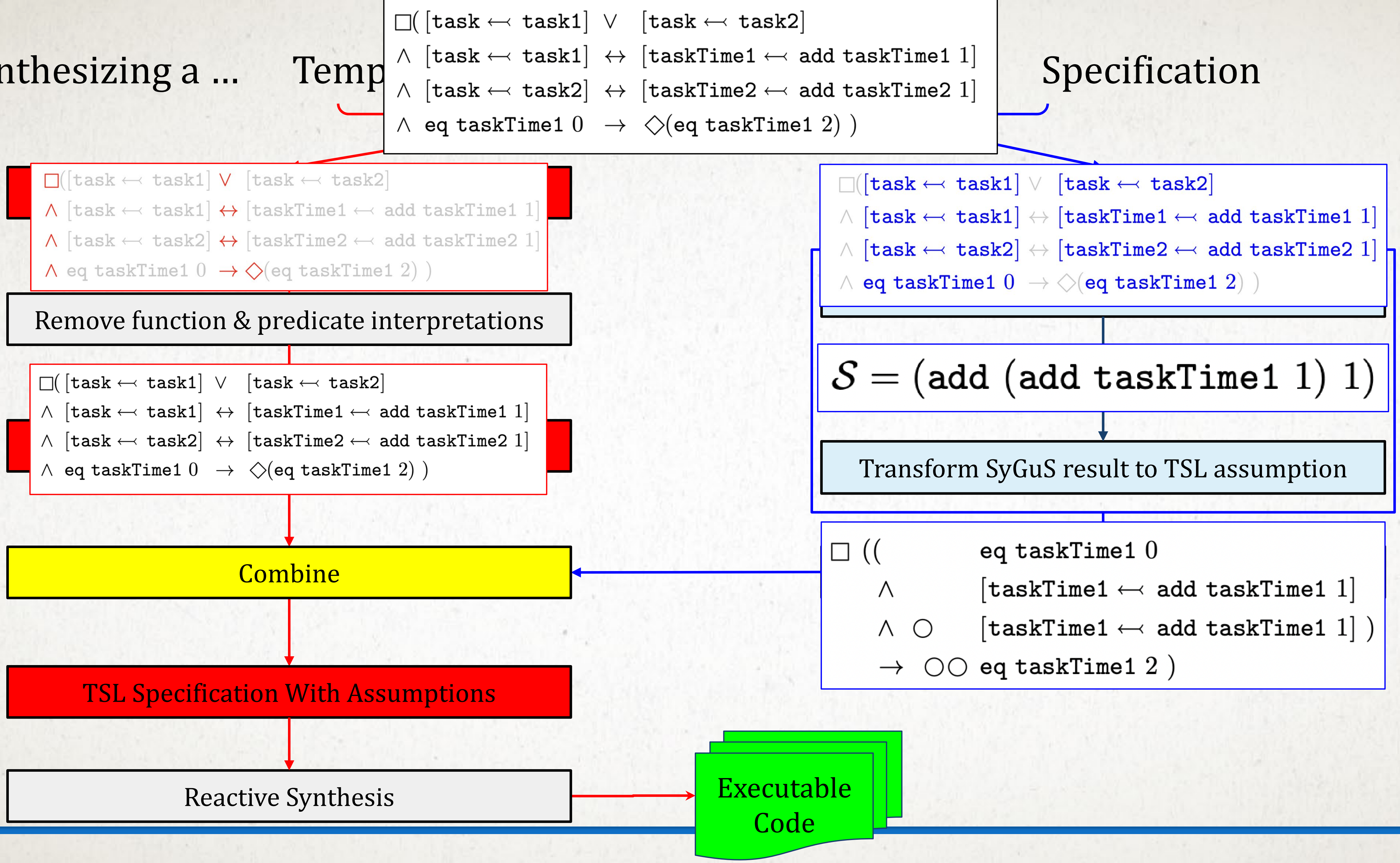
Synthesizing a … Temp Specification





Synthesizing a ... Temp Specification
□([task ↢ task1] ∨ [task ↢ task2]
∧ [task ↢ task1] ↔ [taskTime1 ↢ add taskTime1 1]
∧ [task ↢ task2] ↔ [taskTime2 ↢ add taskTime2 1]
∧ eq taskTime1 0 → ◇(eq taskTime1 2))
□([task ↢ task1] ∨ [task ↢ task2]
∧ [task ↢ task1] ↔ [taskTime1 ↢ add taskTime1 1]
∧ [task ↢ task2] ↔ [taskTime2 ↢ add taskTime2 1]
∧ eq taskTime1 0 → ◇(eq taskTime1 2))
Remove function & predicate interpretations
□([task ↢ task1] ∨ [task ↢ task2]
∧ [task ↢ task1] ↔ [taskTime1 ↢ add taskTime1 1]
∧ [task ↢ task2] ↔ [taskTime2 ↢ add taskTime2 1]
∧ eq taskTime1 0 → ◇(eq taskTime1 2))
Combine
TSL Specification With Assumptions
Reactive Synthesis
Executable Code
□([task ↢ task1] ∨ [task ↢ task2]
∧ [task ↢ task1] ↔ [taskTime1 ↢ add taskTime1 1]
∧ [task ↢ task2] ↔ [taskTime2 ↢ add taskTime2 1]
∧ eq taskTime1 0 → ◇(eq taskTime1 2))
S = (add (add taskTime1 1) 1)
Transform SyGuS result to TSL assumption
□ ((eq taskTime1 0
∧ [taskTime1 ↢ add taskTime1 1]
∧ ○ [taskTime1 ↢ add taskTime1 1])
→ ○○ eq taskTime1 2)

Synthesizing a ... Temp Specification



COMBINE CONTROL SPECIFICATION WITH THE DATA ASSUMPTION

Temporal Stream Logic (TSL) Specification

$$
\begin{aligned}
&\square(\,[\mathtt{task} \leftarrowtail \mathtt{task1}] \vee [\mathtt{task} \leftarrowtail \mathtt{task2}] \\
&\wedge [\mathtt{task} \leftarrowtail \mathtt{task1}] \leftrightarrow [\mathtt{taskTime1} \leftarrowtail \mathtt{add\ taskTime1\ 1}] \\
&\wedge [\mathtt{task} \leftarrowtail \mathtt{task2}] \leftrightarrow [\mathtt{taskTime2} \leftarrowtail \mathtt{add\ taskTime2\ 1}] \\
&\wedge \mathtt{eq\ taskTime1\ 0} \rightarrow \lozenge(\mathtt{eq\ taskTime1\ 2})\,)
\end{aligned}
$$

SyGuS Result as TSL Assumption

$$
\begin{aligned}
\square\,((\quad & \mathtt{eq\ taskTime1\ 0} \\
\wedge \quad & [\mathtt{taskTime1} \leftarrowtail \mathtt{add\ taskTime1\ 1}] \\
\wedge \bigcirc \quad & [\mathtt{taskTime1} \leftarrowtail \mathtt{add\ taskTime1\ 1}]\,) \\
\rightarrow \bigcirc\bigcirc \quad & \mathtt{eq\ taskTime1\ 2}\,)
\end{aligned}
$$

TSL specification with assumptions: Teaching reactive synthesis that 0+1+1=2!

Assumption

$$
\begin{aligned}
(\square\,((\quad & \mathtt{eq\ taskTime1\ 0} \\
\wedge \quad & [\mathtt{taskTime1} \leftarrowtail \mathtt{add\ taskTime1\ 1}] \\
\wedge \bigcirc \quad & [\mathtt{taskTime1} \leftarrowtail \mathtt{add\ taskTime1\ 1}]\,) \\
\rightarrow \bigcirc\bigcirc \quad & \mathtt{eq\ taskTime1\ 2}\,))\ \rightarrow
\end{aligned}
$$

Guarantee

$$
\begin{aligned}
&\square(\,[\mathtt{task} \leftarrowtail \mathtt{task1}] \vee [\mathtt{task} \leftarrowtail \mathtt{task2}] \\
&\wedge [\mathtt{task} \leftarrowtail \mathtt{task1}] \leftrightarrow [\mathtt{taskTime1} \leftarrowtail \mathtt{add\ taskTime1\ 1}] \\
&\wedge [\mathtt{task} \leftarrowtail \mathtt{task2}] \leftrightarrow [\mathtt{taskTime2} \leftarrowtail \mathtt{add\ taskTime2\ 1}] \\
&\wedge \mathtt{eq\ taskTime1\ 0} \rightarrow \lozenge(\mathtt{eq\ taskTime1\ 2})\,)
\end{aligned}
$$

RESULT CAN NOW BE SYNTHESIZED!

- From our original TSL-MT specification, we obtained the TSL specification with assumptions
- We know how to synthesize TSL! (CAV '19, Haskell '19)

TSL specification with assumptions: Teaching reactive synthesis that 0+1+1=2!

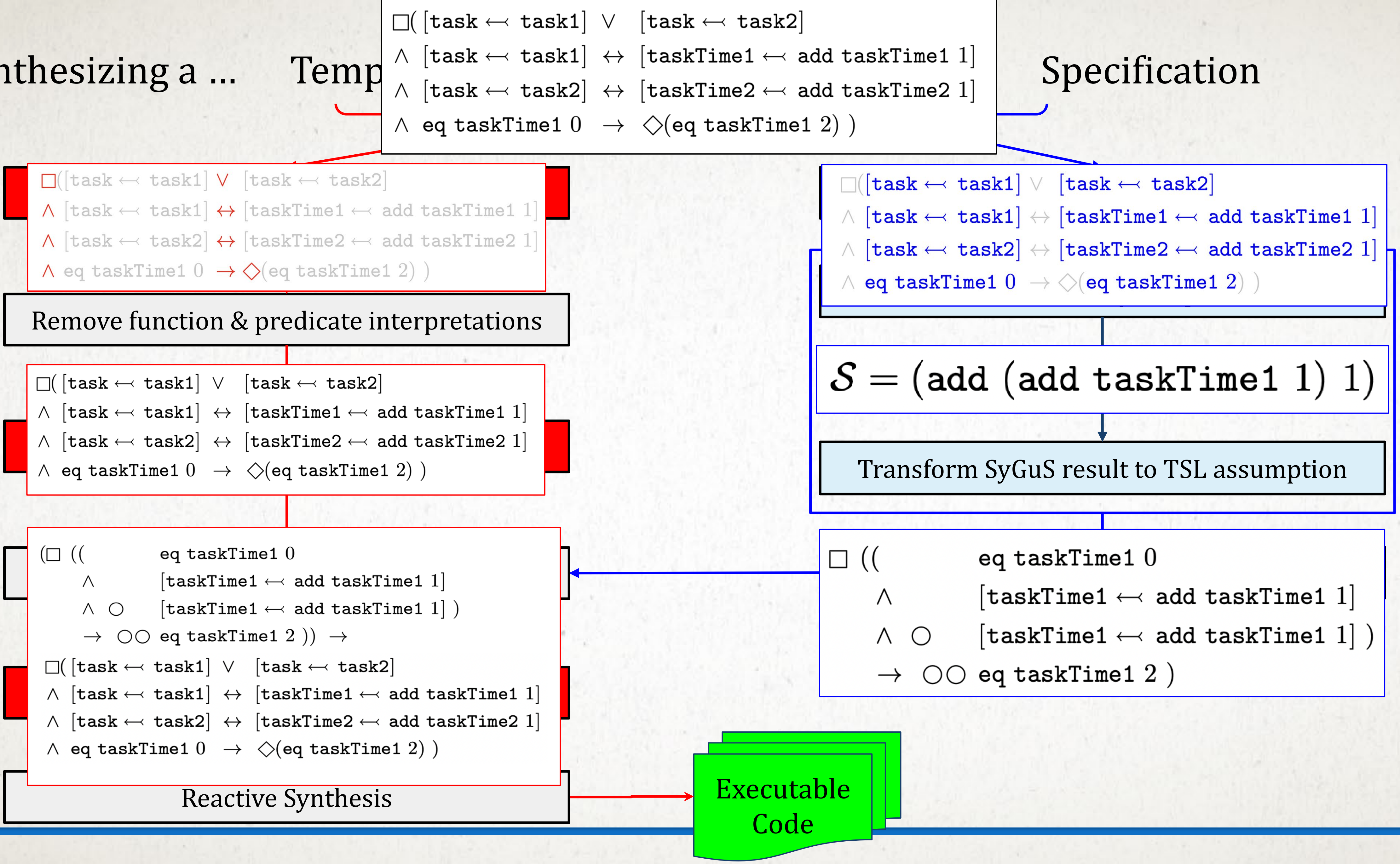
$$
\begin{aligned}
(\square\ ((\quad & \mathtt{eq\ taskTime1}\ 0 \\
\wedge \quad & [\mathtt{taskTime1} \leftarrowtail \mathtt{add\ taskTime1}\ 1] \\
\wedge\ \bigcirc \quad & [\mathtt{taskTime1} \leftarrowtail \mathtt{add\ taskTime1}\ 1]\ ) \\
\rightarrow\ \bigcirc\bigcirc\ & \mathtt{eq\ taskTime1}\ 2\ ))\ \rightarrow
\end{aligned}
$$

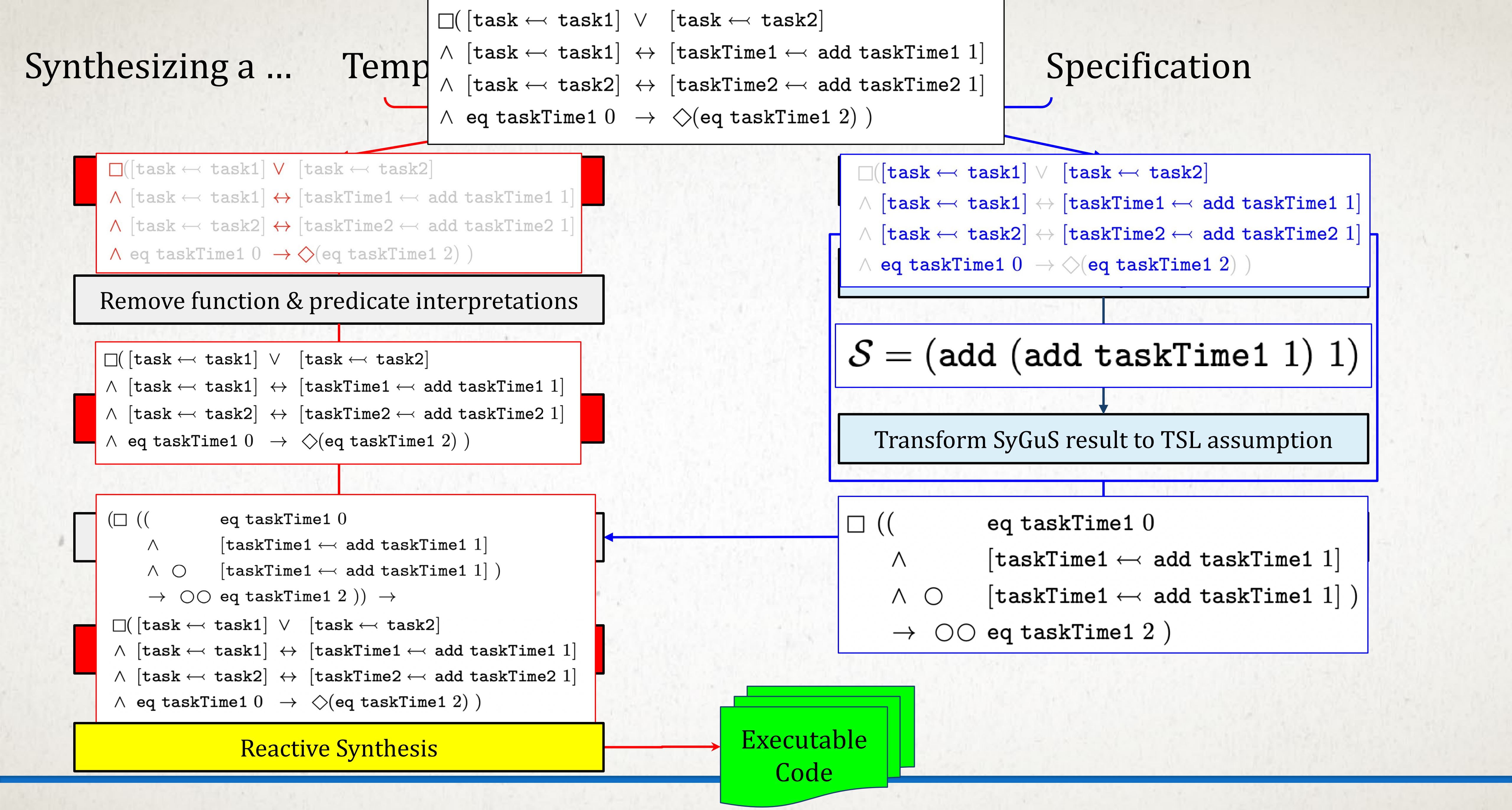
Assumption

$$
\begin{aligned}
\square(\ & [\mathtt{task} \leftarrowtail \mathtt{task1}]\ \vee\ [\mathtt{task} \leftarrowtail \mathtt{task2}] \\
\wedge\ & [\mathtt{task} \leftarrowtail \mathtt{task1}]\ \leftrightarrow\ [\mathtt{taskTime1} \leftarrowtail \mathtt{add\ taskTime1}\ 1] \\
\wedge\ & [\mathtt{task} \leftarrowtail \mathtt{task2}]\ \leftrightarrow\ [\mathtt{taskTime2} \leftarrowtail \mathtt{add\ taskTime2}\ 1] \\
\wedge\ & \mathtt{eq\ taskTime1}\ 0\ \rightarrow\ \diamond(\mathtt{eq\ taskTime1}\ 2)\ )
\end{aligned}
$$

Guarantee

Synthesizing a … Temp… Specification




Synthesizing a ...
Temp
Specification
□([task ↢ task1] ∨ [task ↢ task2]
∧ [task ↢ task1] ↔ [taskTime1 ↢ add taskTime1 1]
∧ [task ↢ task2] ↔ [taskTime2 ↢ add taskTime2 1]
∧ eq taskTime1 0 → ◇(eq taskTime1 2))
□([task ↢ task1] ∨ [task ↢ task2]
∧ [task ↢ task1] ↔ [taskTime1 ↢ add taskTime1 1]
∧ [task ↢ task2] ↔ [taskTime2 ↢ add taskTime2 1]
∧ eq taskTime1 0 → ◇(eq taskTime1 2))
Remove function & predicate interpretations
□([task ↢ task1] ∨ [task ↢ task2]
∧ [task ↢ task1] ↔ [taskTime1 ↢ add taskTime1 1]
∧ [task ↢ task2] ↔ [taskTime2 ↢ add taskTime2 1]
∧ eq taskTime1 0 → ◇(eq taskTime1 2))
(□ ((eq taskTime1 0
∧ [taskTime1 ↢ add taskTime1 1]
∧ ○ [taskTime1 ↢ add taskTime1 1])
→ ○○ eq taskTime1 2)) →
□([task ↢ task1] ∨ [task ↢ task2]
∧ [task ↢ task1] ↔ [taskTime1 ↢ add taskTime1 1]
∧ [task ↢ task2] ↔ [taskTime2 ↢ add taskTime2 1]
∧ eq taskTime1 0 → ◇(eq taskTime1 2))
Reactive Synthesis
Executable Code
□([task ↢ task1] ∨ [task ↢ task2]
∧ [task ↢ task1] ↔ [taskTime1 ↢ add taskTime1 1]
∧ [task ↢ task2] ↔ [taskTime2 ↢ add taskTime2 1]
∧ eq taskTime1 0 → ◇(eq taskTime1 2))
S = (add (add taskTime1 1) 1)
Transform SyGuS result to TSL assumption
□ ((eq taskTime1 0
∧ [taskTime1 ↢ add taskTime1 1]
∧ ○ [taskTime1 ↢ add taskTime1 1])
→ ○○ eq taskTime1 2)

REACTIVE SYNTHESIS

Temporal Logic Specification

Guarantee 3. *When a length-four locked burst starts, no other accesses s HREADY is high, so the current burst ends at the fourth occurrence of I true initially separately from the case in which it is not).*

$$\square\big((\text{HMASTLOCK} \wedge \text{HBURST} = \text{BURST4} \wedge \text{START} \wedge \text{HREADY}) \rightarrow$$
$$\bigcirc(\neg\text{START}\ \mathcal{W}\ [3](\neg\text{START} \wedge \text{HREADY}))\big),$$
$$\square\big((\text{HMASTLOCK} \wedge \text{HBURST} = \text{BURST4} \wedge \text{START} \wedge \neg\text{HREADY}) \rightarrow$$
$$\bigcirc(\neg\text{START}\ \mathcal{W}\ [4](\neg\text{START} \wedge \text{HREADY}))\big).$$

Guarantee 6. *If we do not start an access in the next time step, the bus*

For each master i,

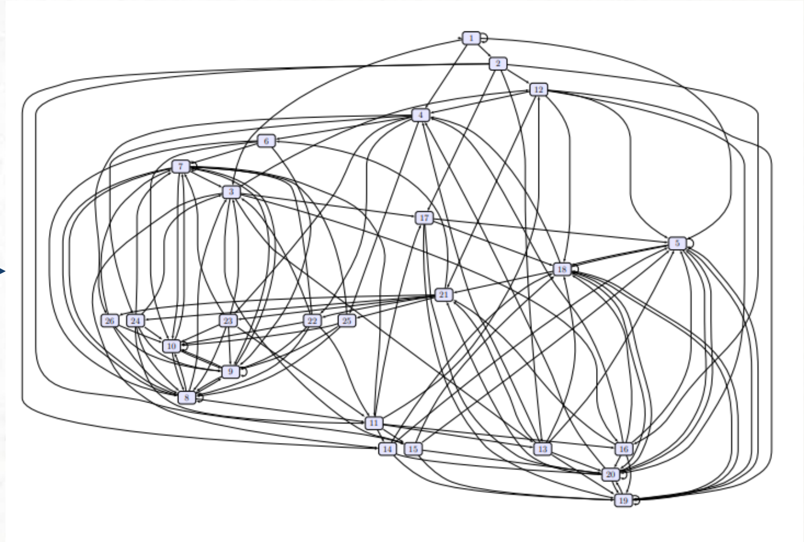
$$\square\big(\bigcirc(\neg\text{START}) \rightarrow \big((\text{HMASTER} = i \leftrightarrow \bigcirc(\text{HMASTER} = i)) \wedge (\text{HMASTLOCK} \leftrightarrow \bigcirc(\text{HMASTLOCK}))\big)\big).$$

Assumption 4. *We assume that all input signals are low initially.*

$$\bigwedge_i \big(\neg\text{HBUSREQ[i]} \wedge \neg\text{HLOCK[i]}\big) \wedge \neg\text{HREADY}.$$

Reactive Synthesis

Synthesized Model



EVALUATION OF TEMOS (FOR TSL-MT)

Benchmark (φ)	$\|\varphi\|$	$\|\mathbb{P}\|$	$\|\mathbb{F}\|$	$\|\psi\|$	ψ Generation (s)	TSL Synthesis (s)	Sum (s)	Synthesized LoC
Music Synthesizer								
Vibrato	10	2	2	21	0.431	0.914	1.345	206
Modulation	33	4	4	41	2.012	3.983	5.995	1352
Intertwined	58	4	4	41	2.157	3.178	5.335	1366
Multi-effect	27	6	6	45	3.145	81.470	84.615	1463
Pong								
Single-Player	27	1	1	5	0.043	0.571	0.614	169
Two-Player	49	2	2	12	0.181	0.625	0.806	195
Bouncing	27	3	2	25	0.418	0.808	1.226	169
Automatic	27	5	2	54	0.541	0.988	1.529	214
Escalator								
Simple	29	1	2	2	0.011	0.434	0.445	166
Counting	57	2	2	8	0.100	0.592	0.692	241
Bidirectional	57	5	11	9	0.340	2.291	2.631	279
Smart	65	8	2	34	3.034	0.935	3.969	179
CPU Scheduler								
Round Robin	21	2	4	16	0.149	0.740	0.889	252
Load Balancer	39	3	4	12	0.531	2.128	2.659	208
Preemptive	54	4	4	12	0.548	0.765	1.313	356
CFS	81	8	5	12	0.533	2.443	2.976	2825

SOME USEFUL LINKS

- Rajeev Alur's tutorial on SyGuS (additional material: real world applications):
https://simons.berkeley.edu/talks/syntax-guided-program-synthesis

- Roderick Bloom's tutorial on reactive synthesis (additional material: shield synthesis):
https://www.newton.ac.uk/seminar/36472/

- Bernd Finkbeiner's tutorial on reactive synthesis (additional material: bounded synthesis, synthesis of distributed systems):
https://simons.berkeley.edu/talks/reactive-synthesis

- Priyanka Golia's talk on functional synthesis:
https://priyanka-golia.github.io/files/slides/qbf_workshop.pdf

- Simons program on synthesis:
https://simons.berkeley.edu/workshops/synthesis-models-systems/schedule#simons-tabs

CONCLUSIONS

- Software synthesis is an exciting idea that started as an interesting theoretical question (“can we derive the program automatically?”) but today is a part of software development used by millions of users
- Various types of software synthesis:
  - Reactive synthesis
  - Deductive synthesis / functional synthesis
  - Syntax-guided synthesis
- Which synthesis type to choose (and what is your specification) depends on the application and goal
- Various applications: network, cyber-psychical systems, AI correctness.
- Synthesis today: connecting many different fields of research