Verifying Temporal Properties of Reactive Systems by Transformation

G.W. Hamilton

School of Computing
Dublin City University
Dublin 9, Ireland
hamilton@computing.dcu.ie

VPT 2015

Outline

Introduction

- We consider the problem of verifying properties of reactive systems
  - continuously react to external events by changing their internal state and producing outputs
- Properties of such systems are usually expressed using a temporal logic
  - safety properties (nothing bad will ever happen)
  - liveness properties (something good will eventually happen)
- One well established technique for this verification is model checking
  - originally developed for finite state systems
  - reactive systems often have an infinite number of states

Background

- Fold/unfold program transformation has been proposed as an automatic approach to this verification problem
  - folding corresponds to the application of a (co)-inductive hypothesis
  - generalization corresponds to abstraction
- Many techniques have been developed for logic programs
  - e.g. Leuschel & Massart, 1999; Roychoudhuri et al., 2000; Fioravanti et al., 2001; Pettorossi et al., 2009; Seki, 2011
- Less techniques have been developed for functional programs
  - notable exception: Lisitsa & Nemytykh, 2007 & 2008
- Most have some limitation such as restriction to finite states, restriction to safety properties or the need for an external constraint solver

Approach

1. Apply distillation to the program defining the reactive system
   - produces a simplified form of program which is easier to analyse
   - removes more intermediate data structures so less generalization is required
2. Define a number of verification rules on the resulting simplified form of program
   - less limitations than those associated with previous techniques
   - always terminates, but may not produce a meaningful result

We apply these techniques to example systems intended to model mutually exclusive access to a shared resource

- safety property: mutual exclusion
- liveness property: non-starvation

Language

Syntax

$e ::= x$	Variable
$\mid c\ e_1 \ldots e_k$	Constructor Application
$\mid \lambda x.e$	λ-Abstraction
$\mid f$	Function Call
$\mid e_0\ e_1$	Application
$\mid \textbf{case}\ e_0\ \textbf{of}\ p_1 \rightarrow e_1 \mid \cdots \mid p_k \rightarrow e_k$	Case Expression
$\mid \textbf{let}\ x = e_0\ \textbf{in}\ e_1$	Let Expression
$\mid e_0\ \textbf{where}\ f_1 = e_1 \ldots f_n = e_n$	Local Function Definitions
$p ::= c\ x_1 \ldots x_k$	Pattern

Language

Semantics

$$((\lambda x.e_0)\ e_1) \overset{\beta}{\rightsquigarrow} (e_0\{x \mapsto e_1\}) \qquad (\textbf{let}\ x = e_0\ \textbf{in}\ e_1) \overset{\beta}{\rightsquigarrow} (e_1\{x \mapsto e_0\})$$

$$\frac{f = e}{f \overset{f}{\rightsquigarrow} e} \qquad \frac{e_0 \overset{r}{\rightsquigarrow} e_0'}{(e_0\ e_1) \overset{r}{\rightsquigarrow} (e_0'\ e_1)}$$

$$\frac{e_0 \overset{r}{\rightsquigarrow} e_0'}{(\textbf{case}\ e_0\ \textbf{of}\ p_1 : e_1 | \ldots p_k : e_k) \overset{r}{\rightsquigarrow} (\textbf{case}\ e_0'\ \textbf{of}\ p_1 : e_1 | \ldots p_k : e_k)}$$

$$\frac{p_i = c\ x_1 \ldots x_n}{(\textbf{case}\ (c\ e_1 \ldots e_n)\ \textbf{of}\ p_1 : e_1' | \ldots | p_k : e_k') \overset{c}{\rightsquigarrow} (e_i\{x_1 \mapsto e_1, \ldots, x_n \mapsto e_n\})}$$

Specifying Reactive Systems

Reactive systems have to react to a series of external events by updating their state. We use a stream datatype:

$$Stream\ a ::= Cons\ a\ Stream$$

For our example mutual exclusion systems, external events belong to the following datatype:

$$Event ::= Request_1 \mid Request_2 \mid Take_1 \mid Take_2 \mid Release_1 \mid Release_2$$

System states belong to the following datatype:

$$SysState ::= State\ ProcState\ ProcState$$

$$ProcState ::= T \mid W \mid U$$

Example 1

```
f es T T
where
f = λes s1 s2.Cons (SysState s1 s2) (case es of
                                      Cons e es → case e of
                                                    Request1 → case s1 of
                                                                 U → f es s1 s2
                                                               | _ → case s2 of
                                                                       U → f es s1 s2
                                                                     | _ → f es W s2
                                                  | Request2 → case s2 of
                                                                 U → f es s1 s2
                                                               | _ → case s1 of
                                                                       U → f es s1 s2
                                                                     | _ → f es s1 W
                                                  | Take1    → case s1 of
                                                                 W → f es U s2
                                                               | _ → f es s1 s2
                                                  | Take2    → case s2 of
                                                                 W → f es s1 U
                                                               | _ → f es s1 s2
                                                  | Release1 → case s1 of
                                                                 U → f es T s2
                                                               | _ → f es s1 s2
                                                  | Release2 → case s2 of
                                                                 U → f es s1 T
                                                               | _ → f es s1 s2)
```

Example 2

```
f es T T
where
f = λes s1 s2.Cons (SysState s1 s2) (case es of
                                      Cons e es → case e of
                                                    Request1 → case s1 of
                                                                 T → f es W s2
                                                               | _ → f es s1 s2
                                                  | Request2 → case s2 of
                                                                 T → f es s1 W
                                                               | _ → f es s1 s2
                                                  | Take1    → case s1 of
                                                                 W → case s2 of
                                                                       T → f es U s2
                                                                     | _ → f es s1 s2
                                                               | _ → f es s1 s2
                                                  | Take2    → case s2 of
                                                                 W → case s1 of
                                                                       T → f es s1 U
                                                                     | _ → f es s1 s2
                                                               | _ → f es s1 s2
                                                  | Release1 → case s1 of
                                                                 U → f es T s2
                                                               | _ → f es s1 s2
                                                  | Release2 → case s2 of
                                                                 U → f es s1 T
                                                               | _ → f es s1 s2)
```

Example 3

```
f es T T Zero Zero
where
f = λes s1 s2 t1 t2.Cons (SysState s1 s2) (case es of
                                  Cons e es → case e of
                                                Request1 → case s1 of
                                                             T → f es W s2 (Succ t2) t2
                                                           | _ → f es s1 s2 t1 t2
                                              | Request2 → case s2 of
                                                             T → f es s1 W t1 (Succ t1)
                                                           | _ → f es s1 s2 t1 t2
                                              | Take1    → case s1 of
                                                             W → case s2 of
                                                                   T → f es U s2 t1 t2
                                                                 | _ → case (t1 < t2) of
                                                                         True  → f es U s2 t1 t2
                                                                       | False → f es s1 s2 t1 t2
                                                           | _ → f es s1 s2 t1 t2
                                              | Take2    → case s2 of
                                                             W → case s1 of
                                                                   T → f es s1 U t1 t2
                                                                 | _ → case (t2 < t1) of
                                                                         True  → f es s1 U t1 t2
                                                                       | False → f es s1 s2 t1 t2
                                                           | _ → f es s1 s2 t1 t2
                                              | Release1 → case s1 of
                                                             U → f es T s2 Zero t2
                                                           | _ → f es s1 s2 t1 t2
                                              | Release2 → case s2 of
                                                             U → f es s1 T t1 Zero
                                                           | _ → f es s1 s2 t1 t2)
```

Linear-time Temporal Logic

Syntax

$$
\begin{array}{rll}
\varphi, \psi ::= & \top & \text{True} \\
\mid & \bot & \text{False} \\
\mid & p & \text{Propositional Variable} \\
\mid & \neg\varphi & \text{Negation} \\
\mid & \varphi \vee \psi & \text{Disjunction} \\
\mid & \varphi \wedge \psi & \text{Conjunction} \\
\mid & \varphi \Rightarrow \psi & \text{Implication} \\
\mid & \Box\varphi & \text{Always} \\
\mid & \Diamond\varphi & \text{Some Time} \\
\mid & \bigcirc\varphi & \text{Next Time}
\end{array}
$$

Linear-time Temporal Logic

Semantics

Models π consist of an infinite number of states $\langle s_0, s_1, \ldots \rangle$ such that each state supplies an assignment to the atomic propositions. For a model π and position i:

$$
\begin{array}{lll}
\pi, i \models \top & & \\
\pi, i \not\models \bot & & \\
\pi, i \models p & \text{iff} & p \in s_i \\
\pi, i \models \neg\varphi & \text{iff} & \pi, i \not\models \varphi \\
\pi, i \models \varphi \vee \psi & \text{iff} & \pi, i \models \varphi \text{ or } \pi, i \models \psi \\
\pi, i \models \varphi \wedge \psi & \text{iff} & \pi, i \models \varphi \text{ and } \pi, i \models \psi \\
\pi, i \models \varphi \Rightarrow \psi & \text{iff} & \pi, i \not\models \varphi \text{ or } \pi, i \models \psi \\
\pi, i \models \Box\varphi & \text{iff} & \forall j \geq i . \pi, j \models \varphi \\
\pi, i \models \Diamond\varphi & \text{iff} & \exists j \geq i . \pi, j \models \varphi \\
\pi, i \models \bigcirc\varphi & \text{iff} & \pi, i + 1 \models \varphi
\end{array}
$$

Temporal Properties

We translate the atomic propositions of temporal formulae into our functional language, using the following datatype for truth values:

$$\mathit{TruthVal} ::= \mathit{True} \mid \mathit{False} \mid \mathit{Undefined}$$

Property 1: Mutual Exclusion

$$\begin{array}{l}\Box(\textbf{case } s \textbf{ of} \\ \quad \mathit{SysState}\ s_1\ s_2 \rightarrow \textbf{case } s_1 \textbf{ of} \\ \qquad\qquad U \rightarrow \textbf{case } s_2 \textbf{ of} \\ \qquad\qquad\qquad\qquad U \rightarrow \mathit{False} \\ \qquad\qquad\qquad\qquad \mid \_ \rightarrow \mathit{True} \\ \qquad\qquad \mid \_ \rightarrow \mathit{True})\end{array}$$

Temporal Properties

Property 2: Non-Starvation (Process 1)

$$\Box((\textbf{case } s \textbf{ of}\ SysState\ s_1\ s_2 \rightarrow \textbf{case } s_1 \textbf{ of}\ W \rightarrow True \mid \_ \rightarrow False) \Rightarrow \Diamond(\textbf{case } s \textbf{ of}\ SysState\ s_1\ s_2 \rightarrow \textbf{case } s_1 \textbf{ of}\ U \rightarrow True \mid \_ \rightarrow False))$$

And similarly for Process 2.

Verification

We first of all transform the reactive systems definitions into the following simplified form using distillation:

Distilled Form

$$
\begin{array}{lcl}
e^{\rho} & ::= & \mathit{Cons}\ e_0^{\rho}\ e_1^{\rho} \\
& | & f\ x_1 \ldots x_n \\
& | & \textbf{case}\ x\ \textbf{of}\ p_1 \rightarrow e_1^{\rho} \mid \cdots \mid p_k \rightarrow e_n^{\rho},\ \text{where}\ x \notin \rho \\
& | & x\ e_1^{\rho} \ldots e_n^{\rho},\ \text{where}\ x \in \rho \\
& | & \textbf{let}\ x = \lambda x_1 \ldots x_n . e_0^{\rho}\ \textbf{in}\ e_1^{(\rho \cup \{x\})} \\
& | & e_0^{\rho}\ \textbf{where}\ f_1 = \lambda x_{1_1} \ldots x_{1_k} . e_1^{\rho} \ldots f_n = \lambda x_{n_1} \ldots x_{n_k} . e_n^{\rho}
\end{array}
$$

During verification, all let variables are given an undefined value, so there will be a finite number of states.

Verification

We define verification rules $\mathcal{P}[\![e]\!]\ \varphi\ \phi\ \rho$ where e is an expression in this simplified form, φ is the temporal formula to be verified, ϕ is a function variable environment and ρ is the set of previously encountered function calls (used for the detection of loops).

Theorem (Soundness)

$\forall e \in \text{Prog}, \varphi \in \text{WFF}: \mathcal{P}[\![e]\!]\ \varphi\ \emptyset\ \emptyset = \mathit{True} \Rightarrow \pi, 0 \vDash \varphi$
where π is a model for e.

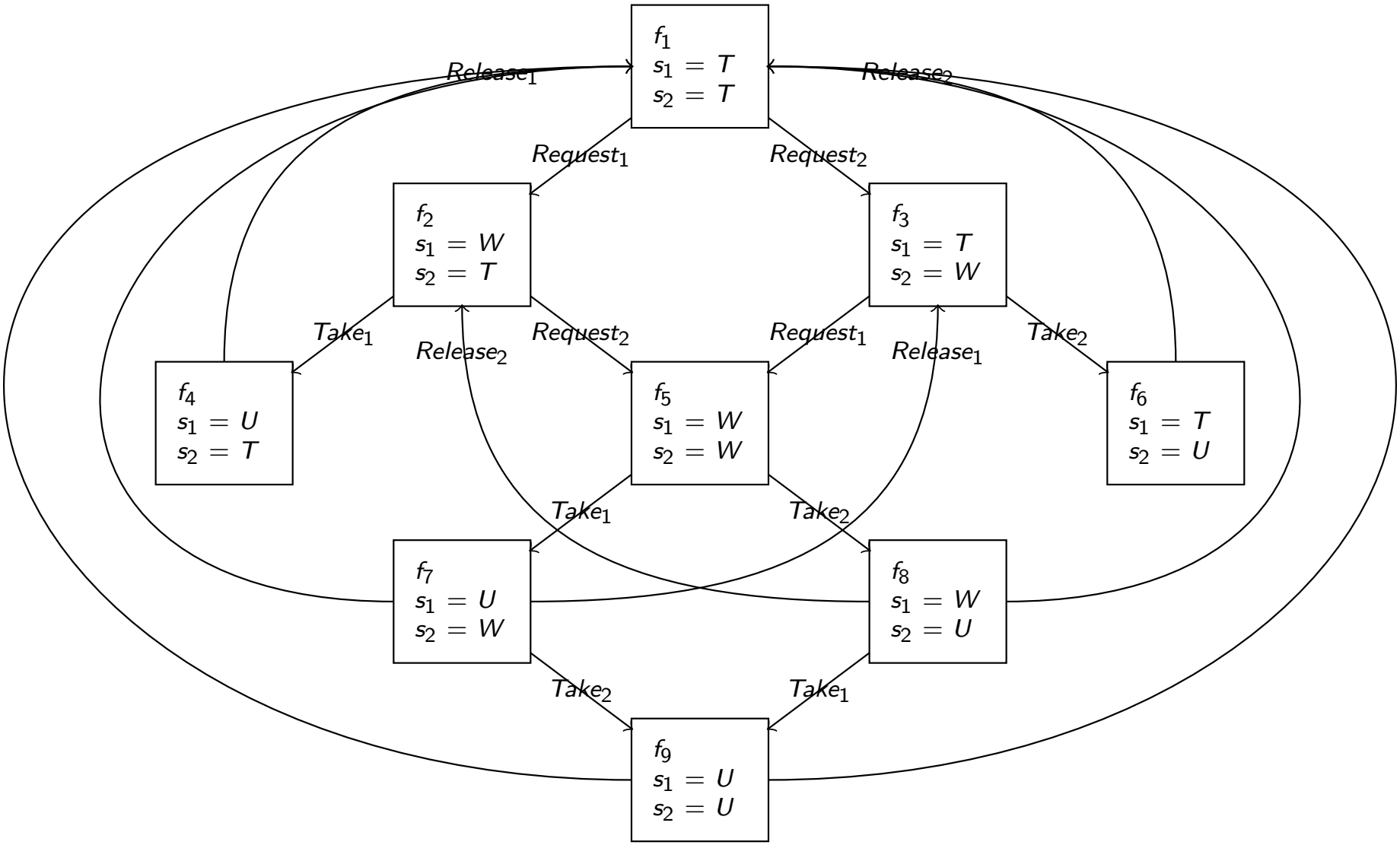
Theorem (Termination)

$\forall e \in \text{Prog}, \varphi \in \text{WFF}: \mathcal{P}[\![e]\!]\ \varphi\ \emptyset\ \emptyset$ always terminates.

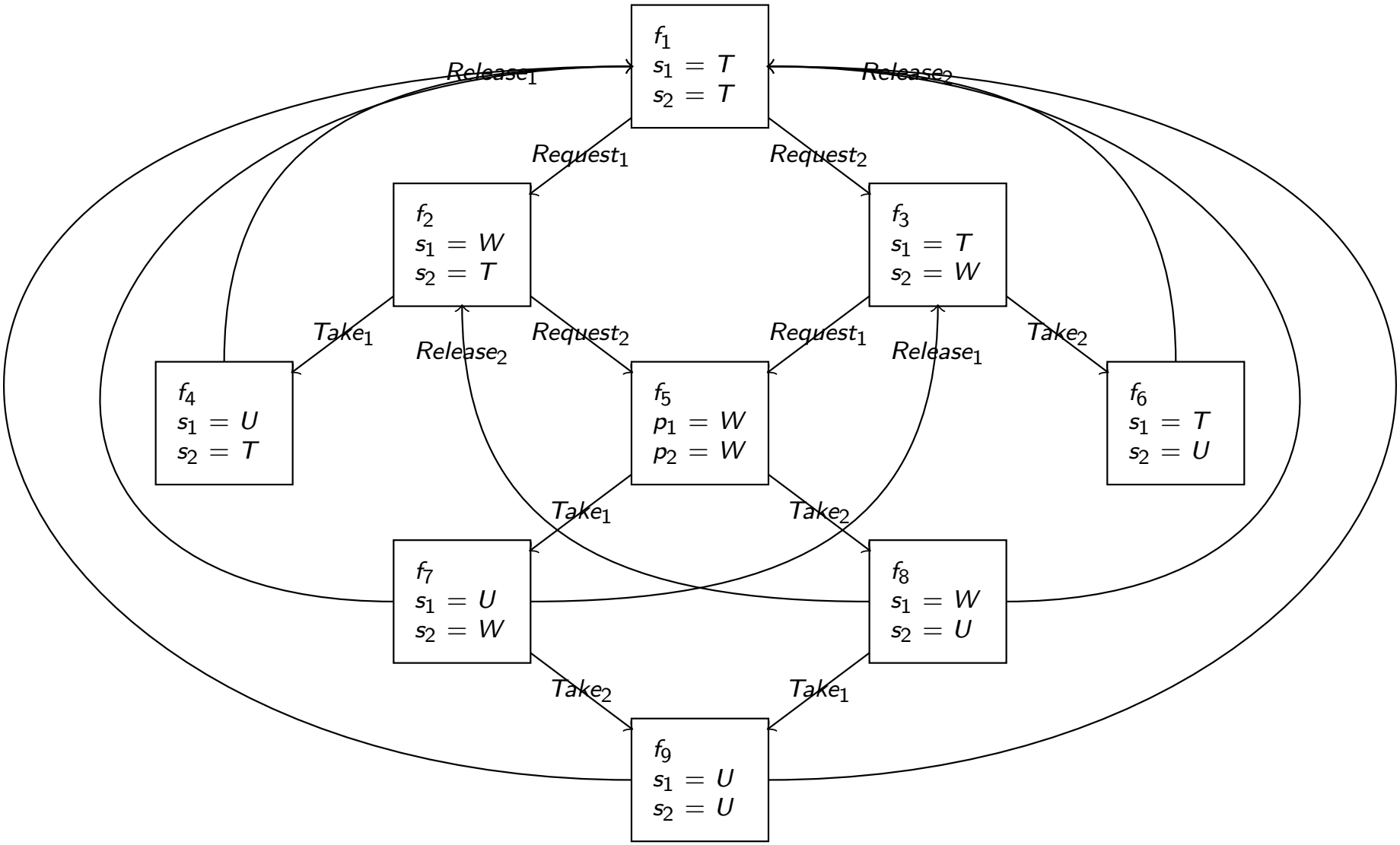
Result of Transforming Example 1

$$
\begin{array}{l}
f_1\ es \\
\textbf{where} \\
f_1 = \lambda es.\mathit{Cons}\ (\mathit{SysState}\ T\ T)\ (\textbf{case}\ es\ \textbf{of} \\
\qquad \mathit{Cons}\ e\ es \rightarrow \textbf{case}\ e\ \textbf{of} \\
\qquad\qquad \mathit{Request}_1 \rightarrow f_2\ es \\
\qquad\qquad \mid \mathit{Request}_2 \rightarrow f_3\ es \\
\qquad\qquad \mid \_ \rightarrow f_1\ es) \\
f_2 = \lambda es.\mathit{Cons}\ (\mathit{SysState}\ W\ T)\ (\textbf{case}\ es\ \textbf{of} \\
\qquad \mathit{Cons}\ e\ es \rightarrow \textbf{case}\ e\ \textbf{of} \\
\qquad\qquad \mathit{Take}_1 \rightarrow f_4\ es \\
\qquad\qquad \mid \mathit{Request}_2 \rightarrow f_5\ es \\
\qquad\qquad \mid \_ \rightarrow f_2\ es) \\
f_3 = \lambda es.\mathit{Cons}\ (\mathit{SysState}\ T\ W)\ (\textbf{case}\ es\ \textbf{of} \\
\qquad \mathit{Cons}\ e\ es \rightarrow \textbf{case}\ e\ \textbf{of} \\
\qquad\qquad \mathit{Request}_1 \rightarrow f_5\ es \\
\qquad\qquad \mid \mathit{Take}_2 \rightarrow f_6\ es \\
\qquad\qquad \mid \_ \rightarrow f_3\ es) \\
f_4 = \lambda es.\mathit{Cons}\ (\mathit{SysState}\ U\ T)\ (\textbf{case}\ es\ \textbf{of} \\
\qquad \mathit{Cons}\ e\ es \rightarrow \textbf{case}\ e\ \textbf{of} \\
\qquad\qquad \mathit{Release}_1 \rightarrow f_1\ es \\
\qquad\qquad \mid \_ \rightarrow f_4\ es) \\
f_5 = \lambda es.\mathit{Cons}\ (\mathit{SysState}\ W\ W)\ (\textbf{case}\ es\ \textbf{of} \\
\qquad \mathit{Cons}\ e\ es \rightarrow \textbf{case}\ e\ \textbf{of} \\
\qquad\qquad \mathit{Take}_1 \rightarrow f_7\ es \\
\qquad\qquad \mid \mathit{Take}_2 \rightarrow f_8\ es \\
\qquad\qquad \mid \_ \rightarrow f_5\ es) \\
f_6 = \lambda es.\mathit{Cons}\ (\mathit{SysState}\ T\ U)\ (\textbf{case}\ es\ \textbf{of} \\
\qquad \mathit{Cons}\ e\ es \rightarrow \textbf{case}\ e\ \textbf{of} \\
\qquad\qquad \mathit{Release}_2 \rightarrow f_1\ es \\
\qquad\qquad \mid \_ \rightarrow f_6\ es)
\end{array}
$$

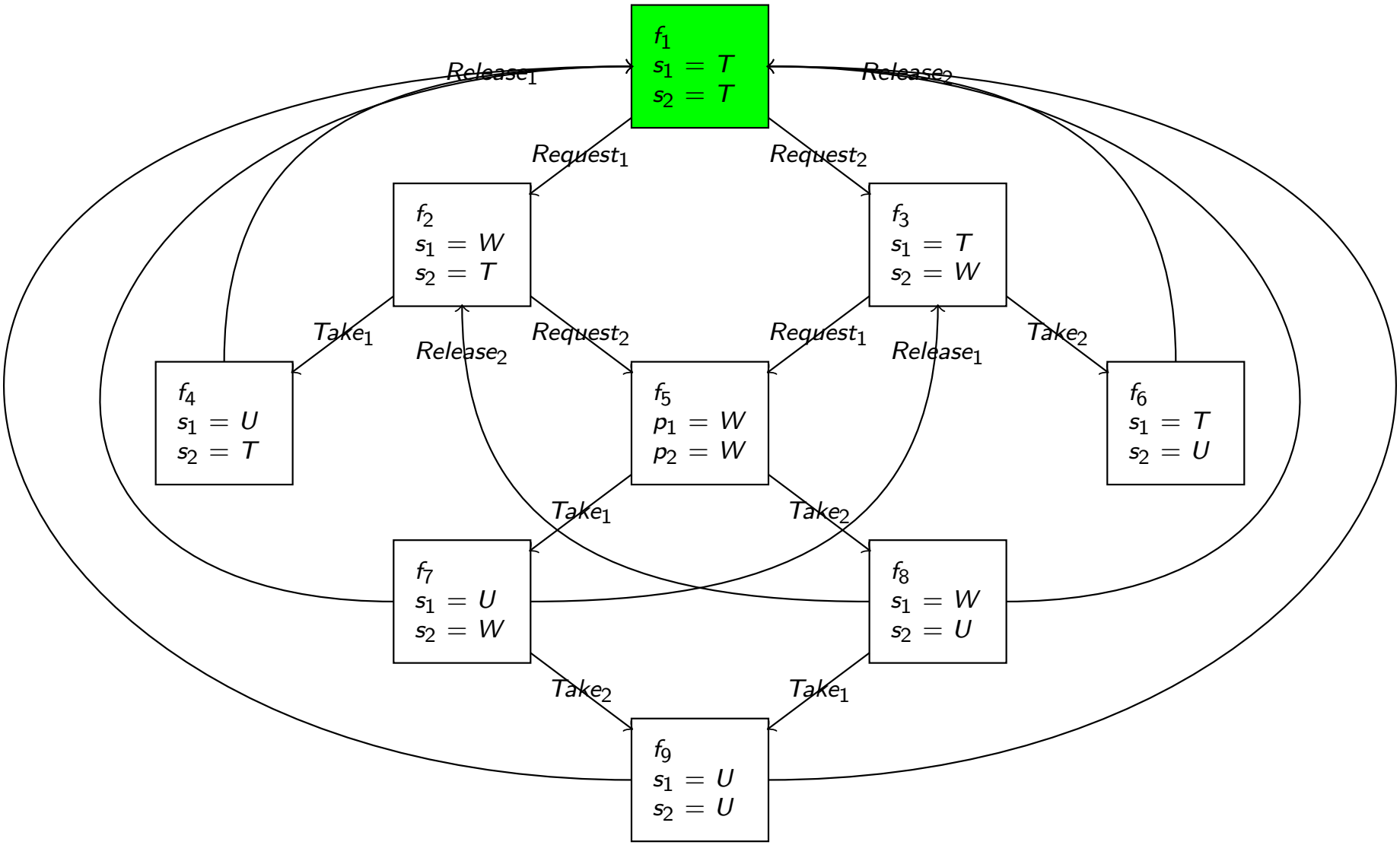
LTS Representation of Transformed Example 1



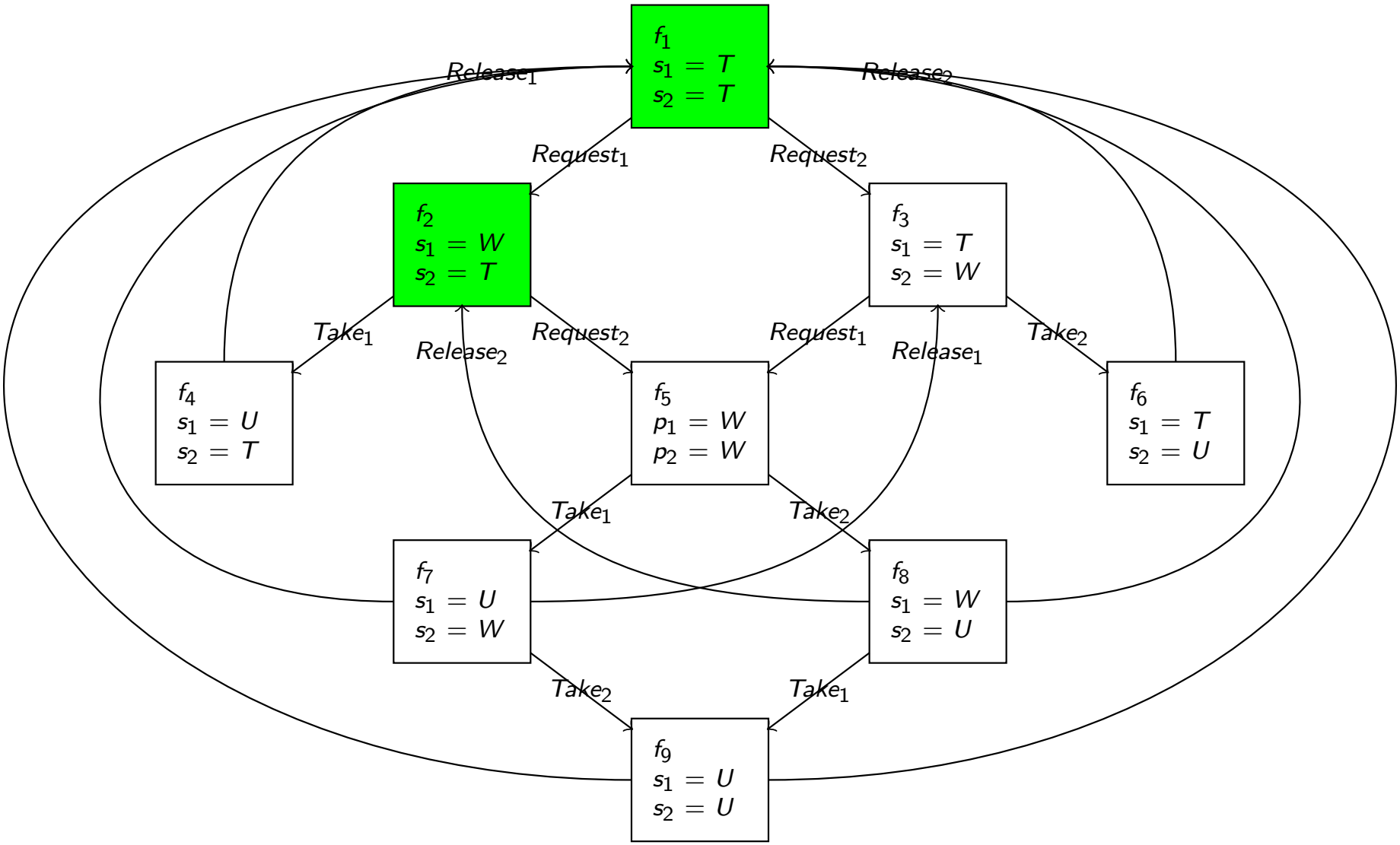
Verification of Property 1 for Example 1



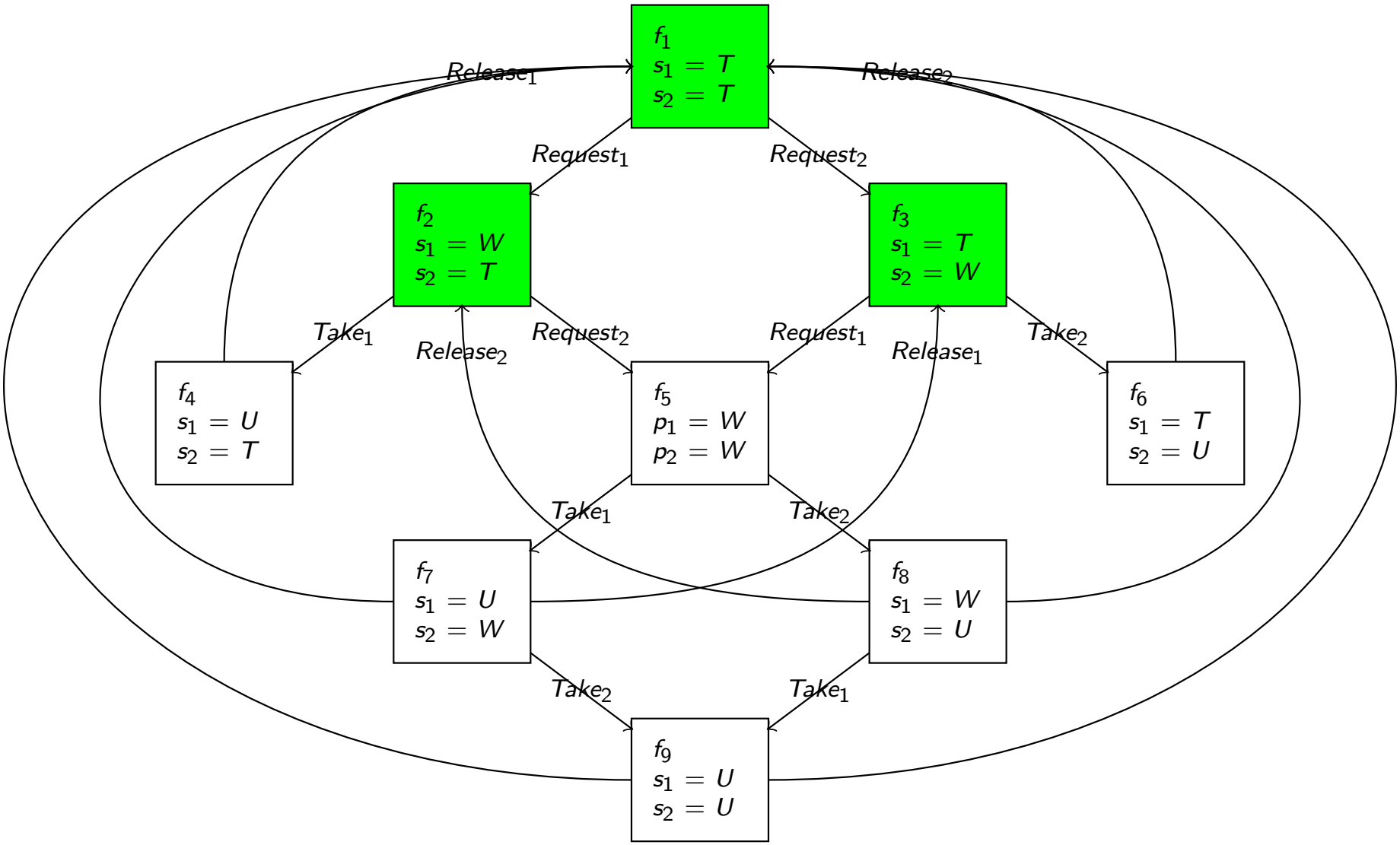
Verification of Property 1 for Example 1



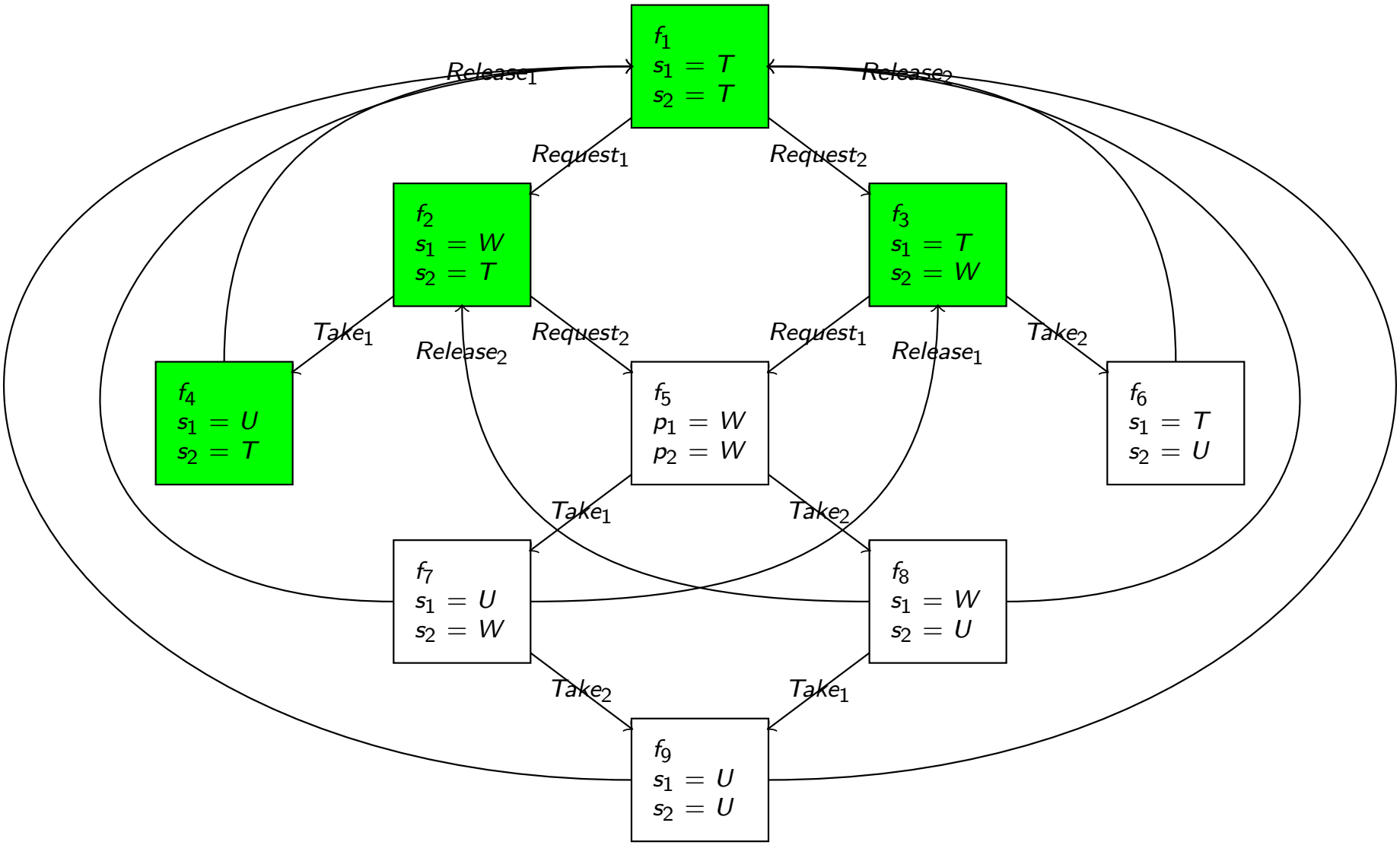
Verification of Property 1 for Example 1



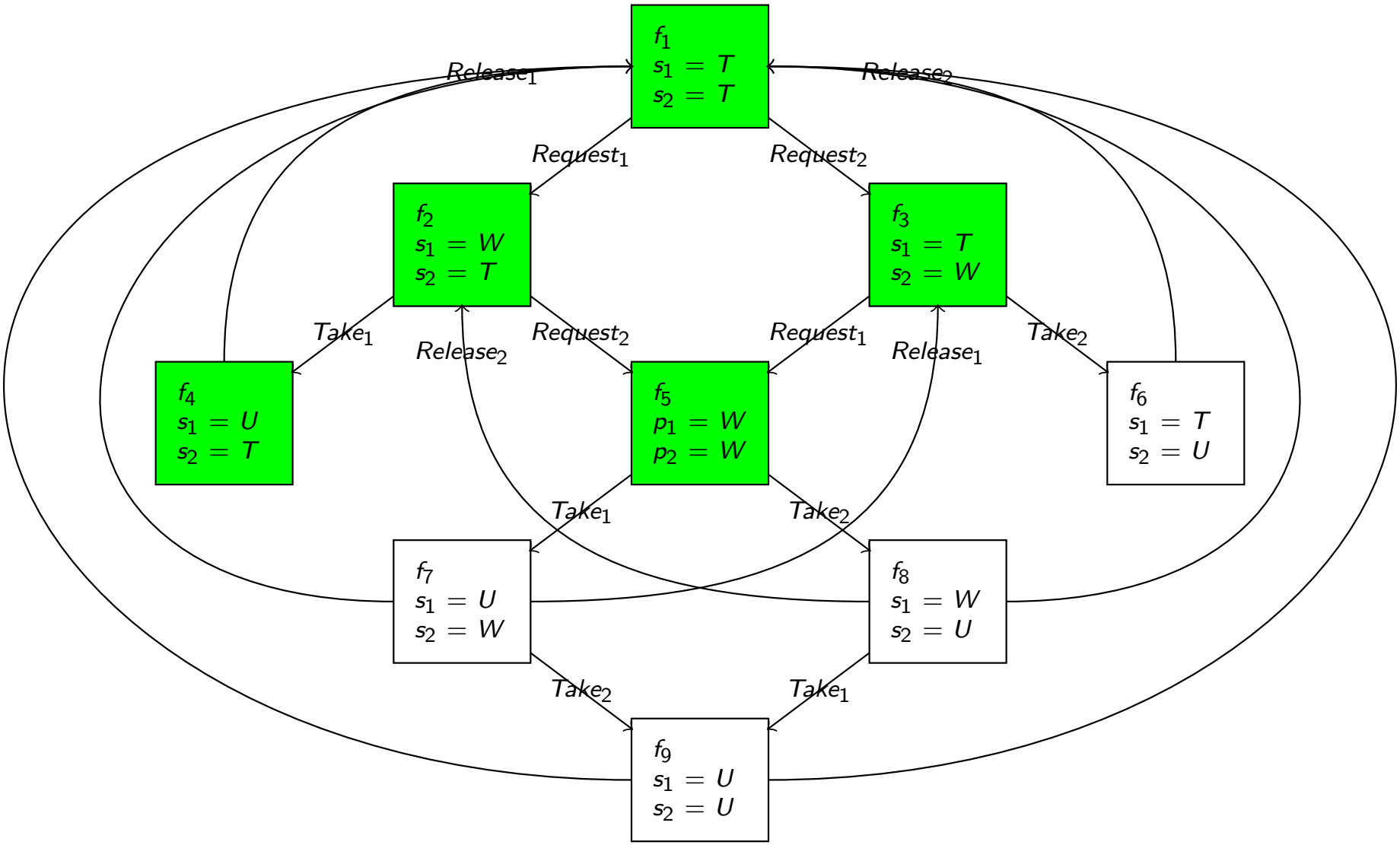
Verification of Property 1 for Example 1



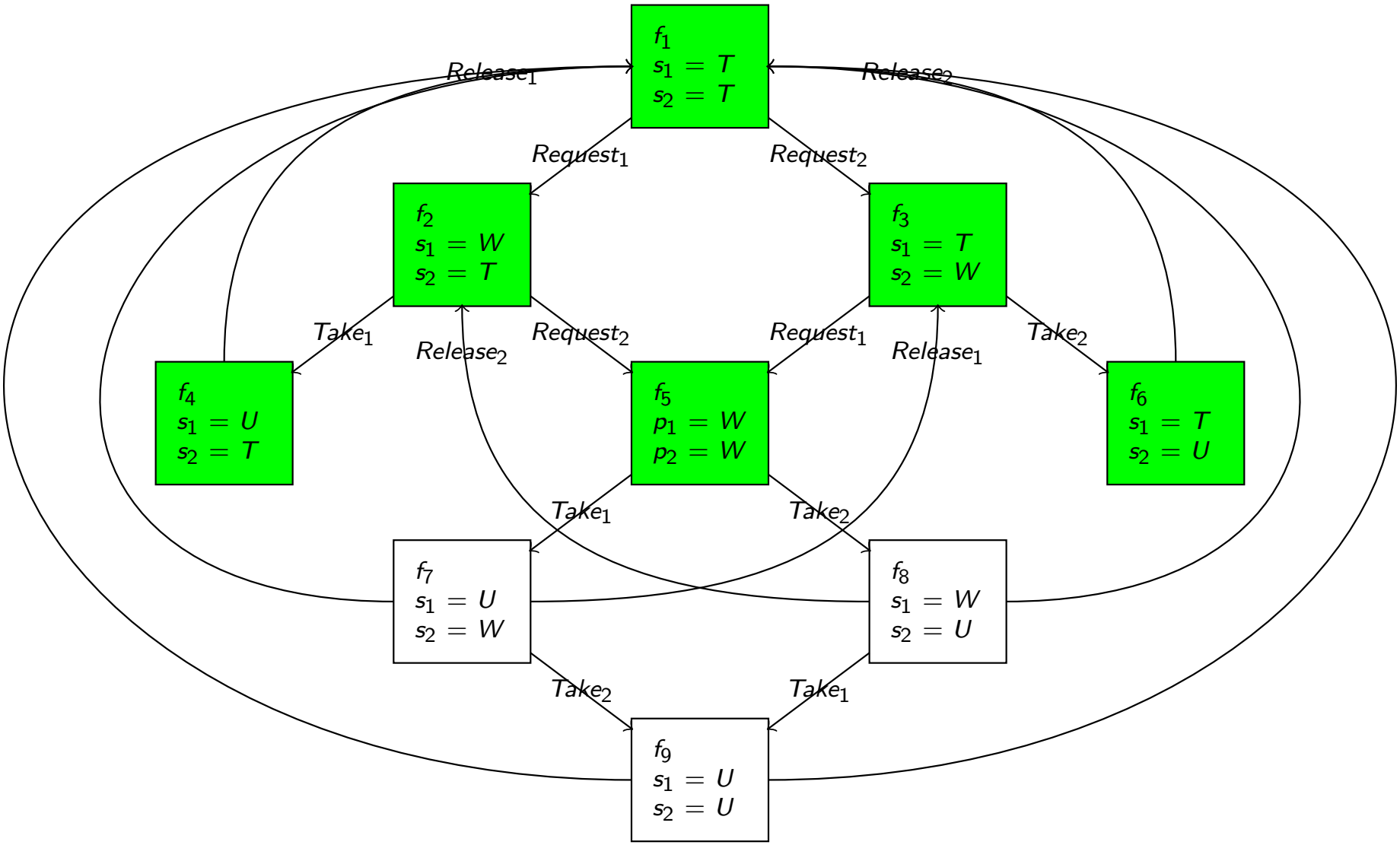
Verification of Property 1 for Example 1



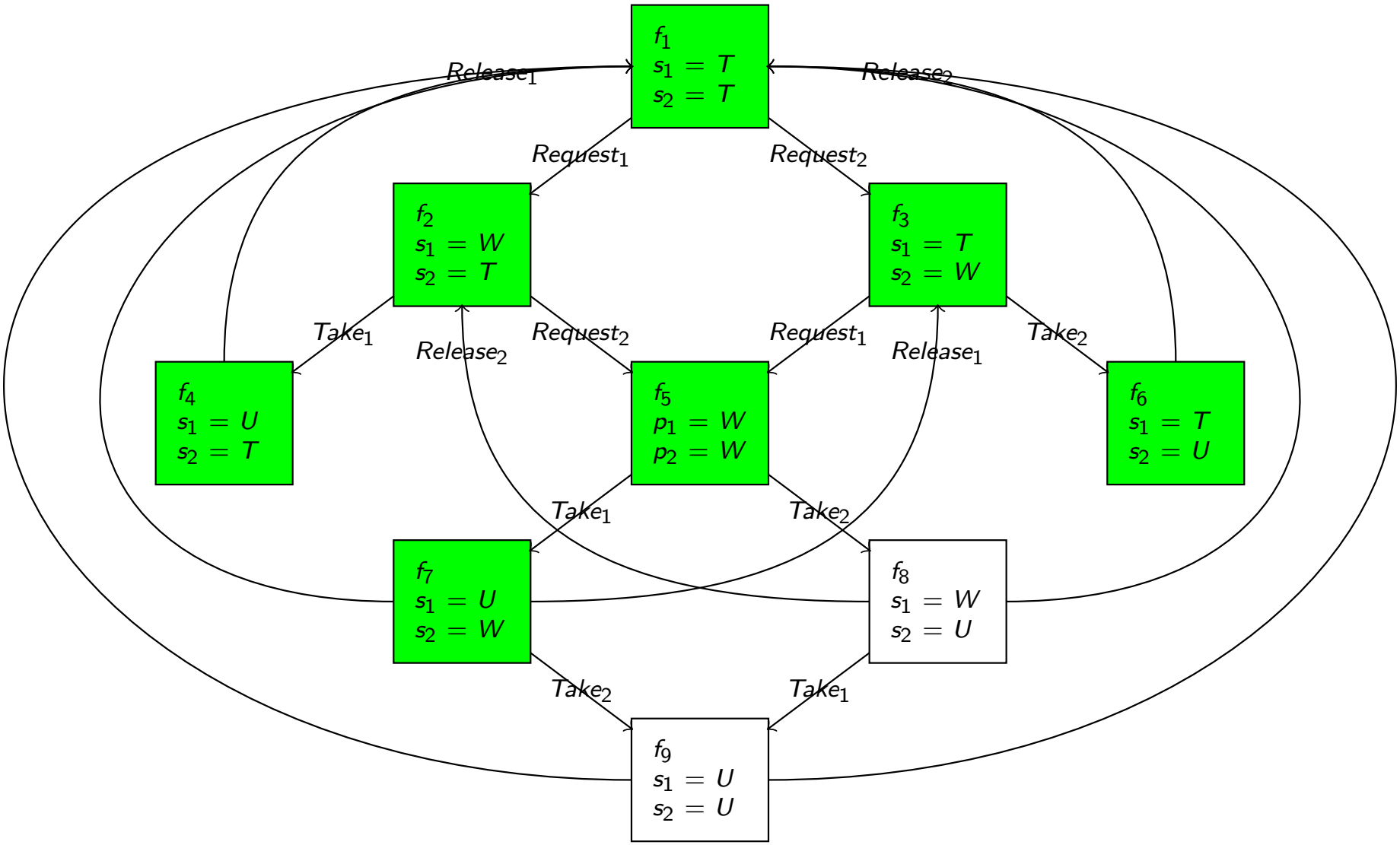
Verification of Property 1 for Example 1



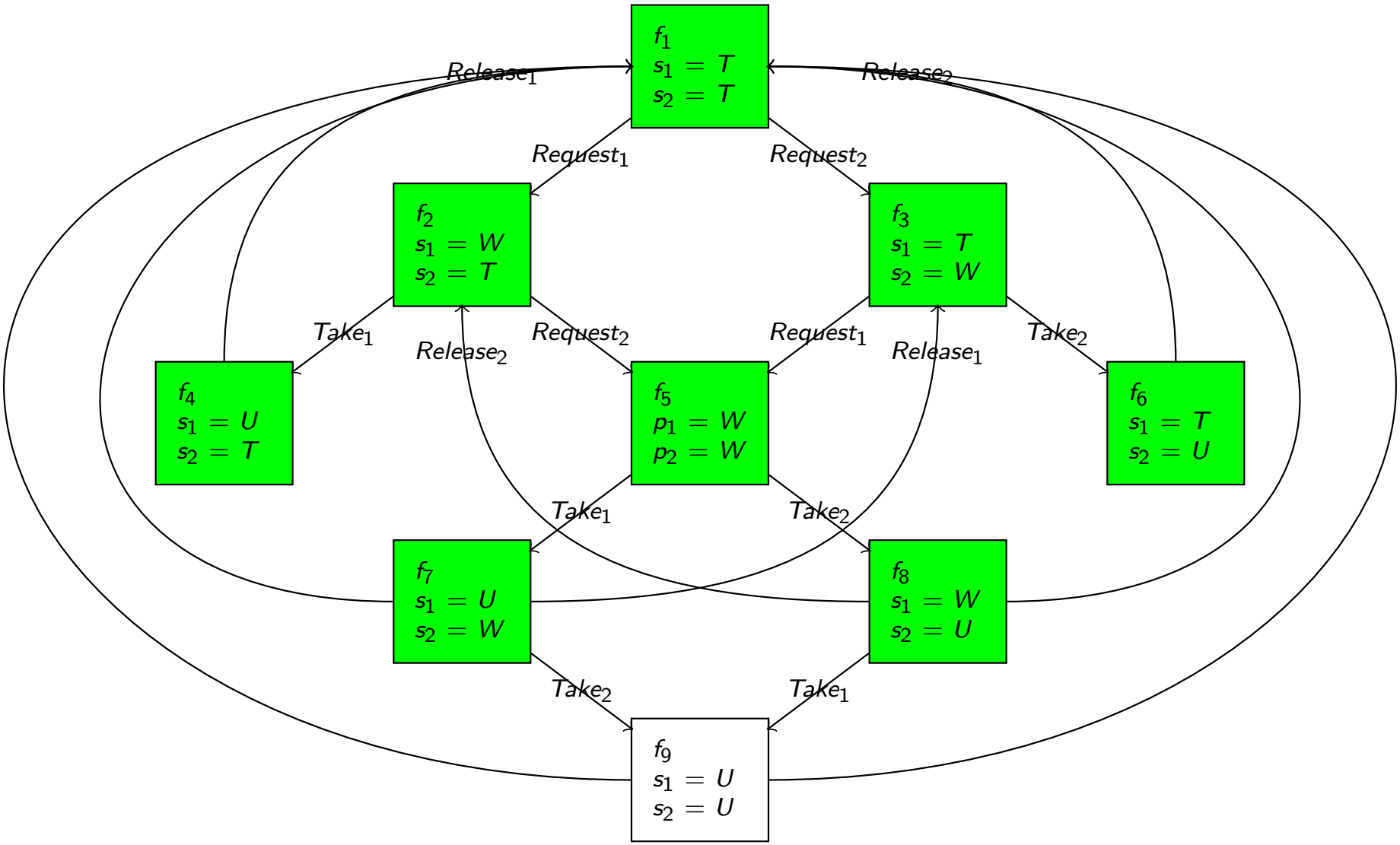
Verification of Property 1 for Example 1



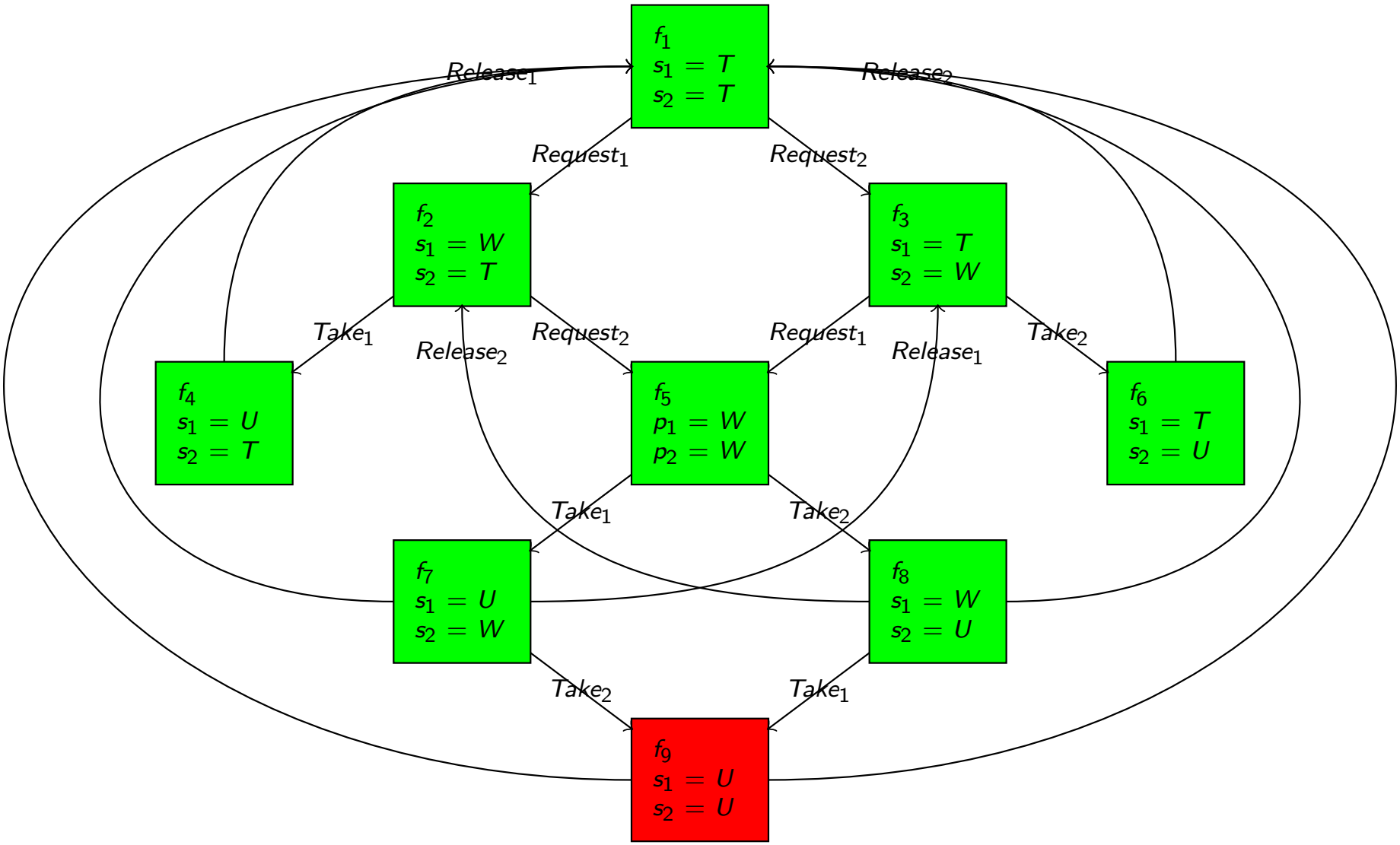
Verification of Property 1 for Example 1



Verification of Property 1 for Example 1



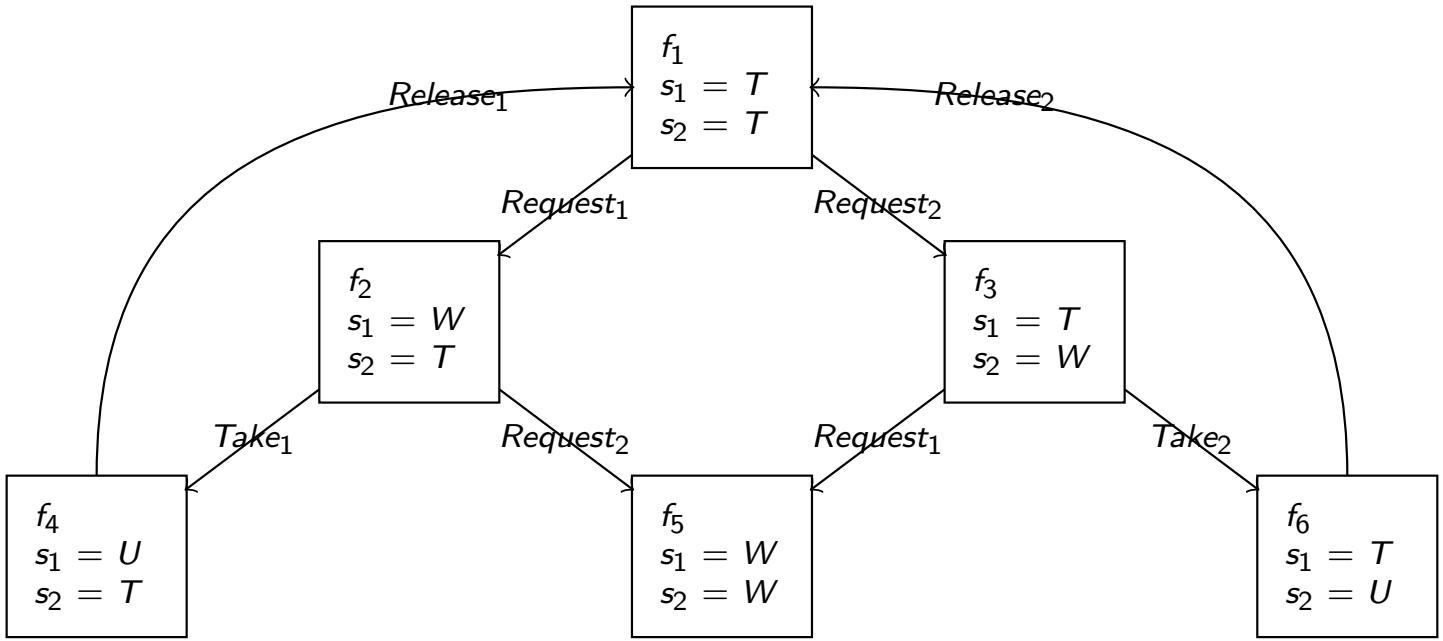
Verification of Property 1 for Example 1



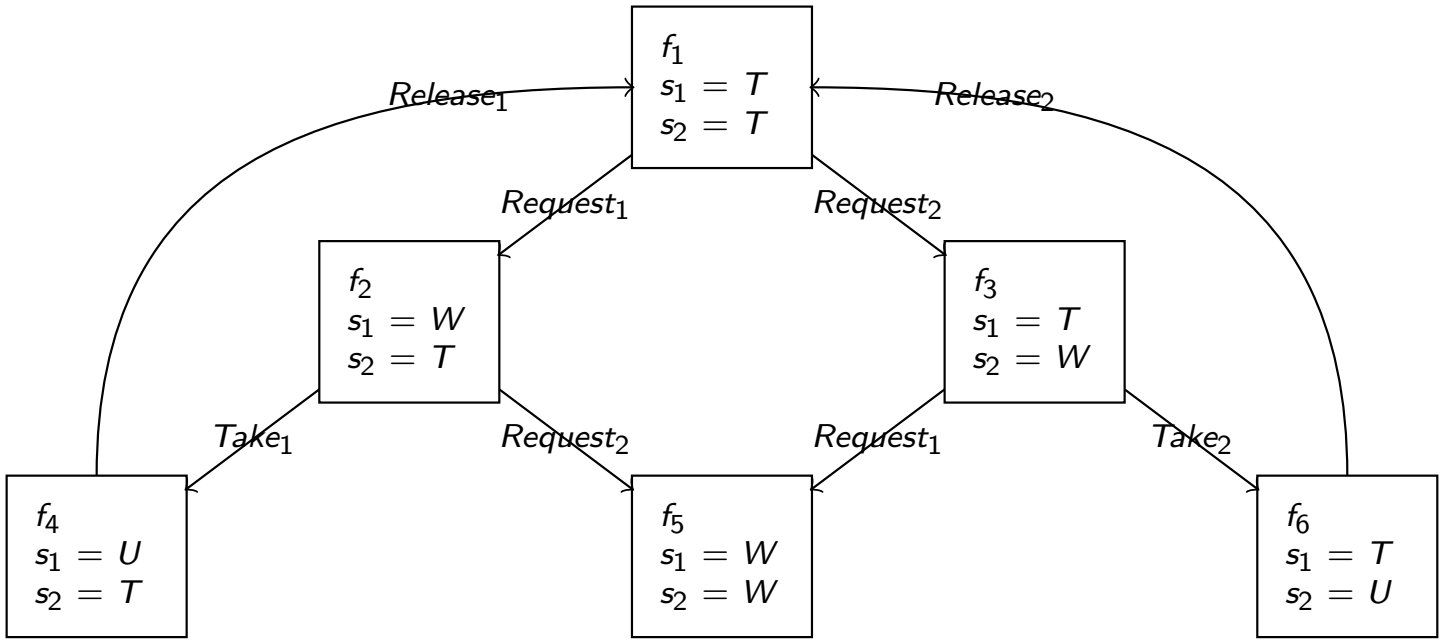
Result of Transforming Example 2

$$
\begin{array}{l}
f_1\ es \\
\textbf{where} \\
f_1 = \lambda es.Cons\ (SysState\ T\ T)\ (\textbf{case}\ es\ \textbf{of} \\
\qquad Cons\ e\ es \rightarrow \textbf{case}\ e\ \textbf{of} \\
\qquad\qquad Request_1 \rightarrow f_2\ es \\
\qquad\qquad \mid Request_2 \rightarrow f_3\ es \\
\qquad\qquad \mid \_ \rightarrow f_1\ es) \\
f_2 = \lambda es.Cons\ (SysState\ W\ T)\ (\textbf{case}\ es\ \textbf{of} \\
\qquad Cons\ e\ es \rightarrow \textbf{case}\ e\ \textbf{of} \\
\qquad\qquad Take_1 \rightarrow f_4\ es \\
\qquad\qquad \mid Request_2 \rightarrow f_5\ es \\
\qquad\qquad \mid \_ \rightarrow f_2\ es) \\
f_3 = \lambda es.Cons\ (SysState\ T\ W)\ (\textbf{case}\ es\ \textbf{of} \\
\qquad Cons\ e\ es \rightarrow \textbf{case}\ e\ \textbf{of} \\
\qquad\qquad Request_1 \rightarrow f_5\ es \\
\qquad\qquad \mid Take_2 \rightarrow f_6\ es \\
\qquad\qquad \mid \_ \rightarrow f_3\ es) \\
f_4 = \lambda es.Cons\ (SysState\ U\ T)\ (\textbf{case}\ es\ \textbf{of} \\
\qquad Cons\ e\ es \rightarrow \textbf{case}\ e\ \textbf{of} \\
\qquad\qquad Release_1 \rightarrow f_1\ es \\
\qquad\qquad \mid \_ \rightarrow f_4\ es) \\
f_5 = \lambda es.Cons\ (SysState\ W\ W)\ (\textbf{case}\ es\ \textbf{of} \\
\qquad Cons\ e\ es \rightarrow \textbf{case}\ e\ \textbf{of} \\
\qquad\qquad \_ \rightarrow f_5\ es) \\
f_6 = \lambda es.Cons\ (SysState\ T\ U)\ (\textbf{case}\ es\ \textbf{of} \\
\qquad Cons\ e\ es \rightarrow \textbf{case}\ e\ \textbf{of} \\
\qquad\qquad Release_2 \rightarrow f_1\ es \\
\qquad\qquad \mid \_ \rightarrow f_6\ es)
\end{array}
$$

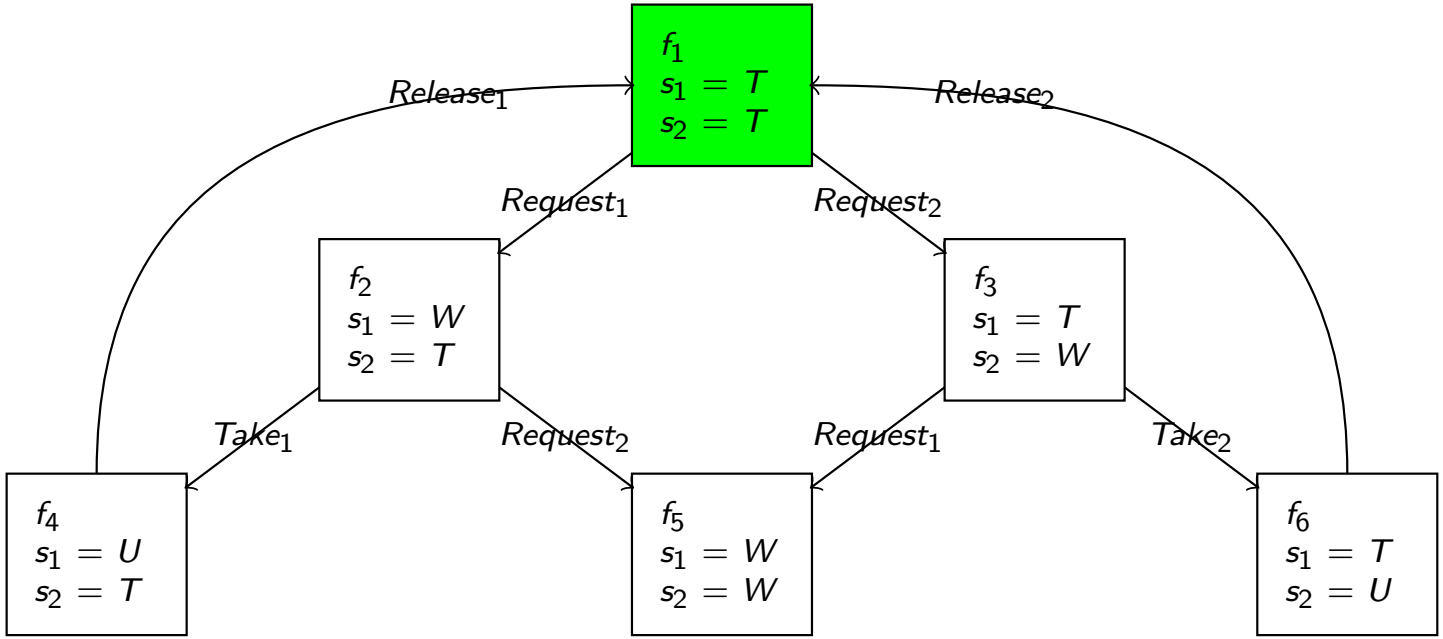
LTS Representation of Transformed Example 2



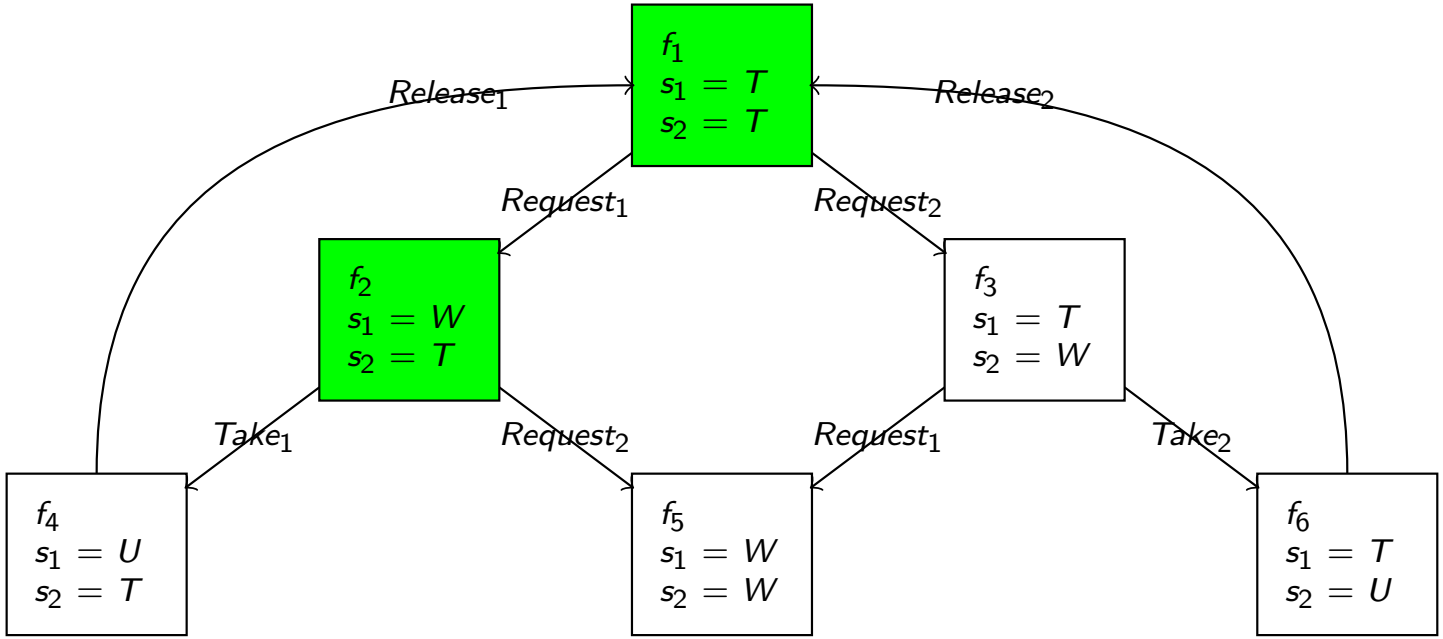
Verification of Property 1 for Example 2



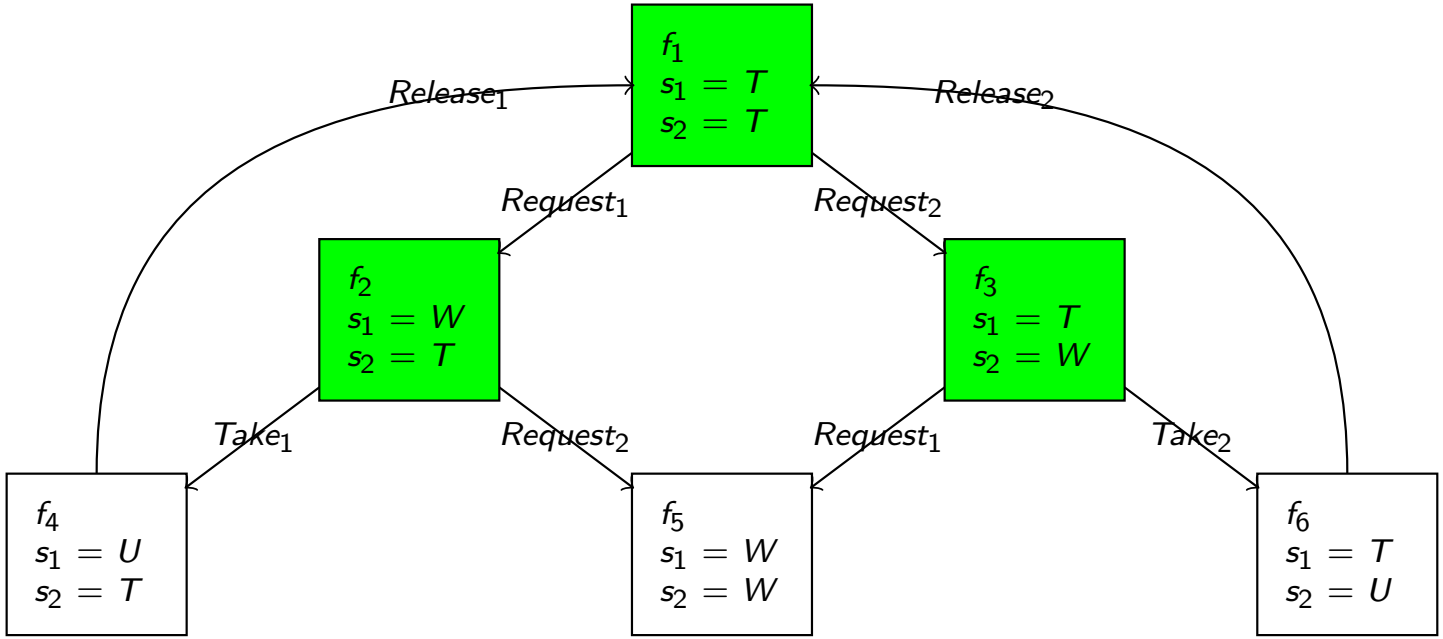
Verification of Property 1 for Example 2



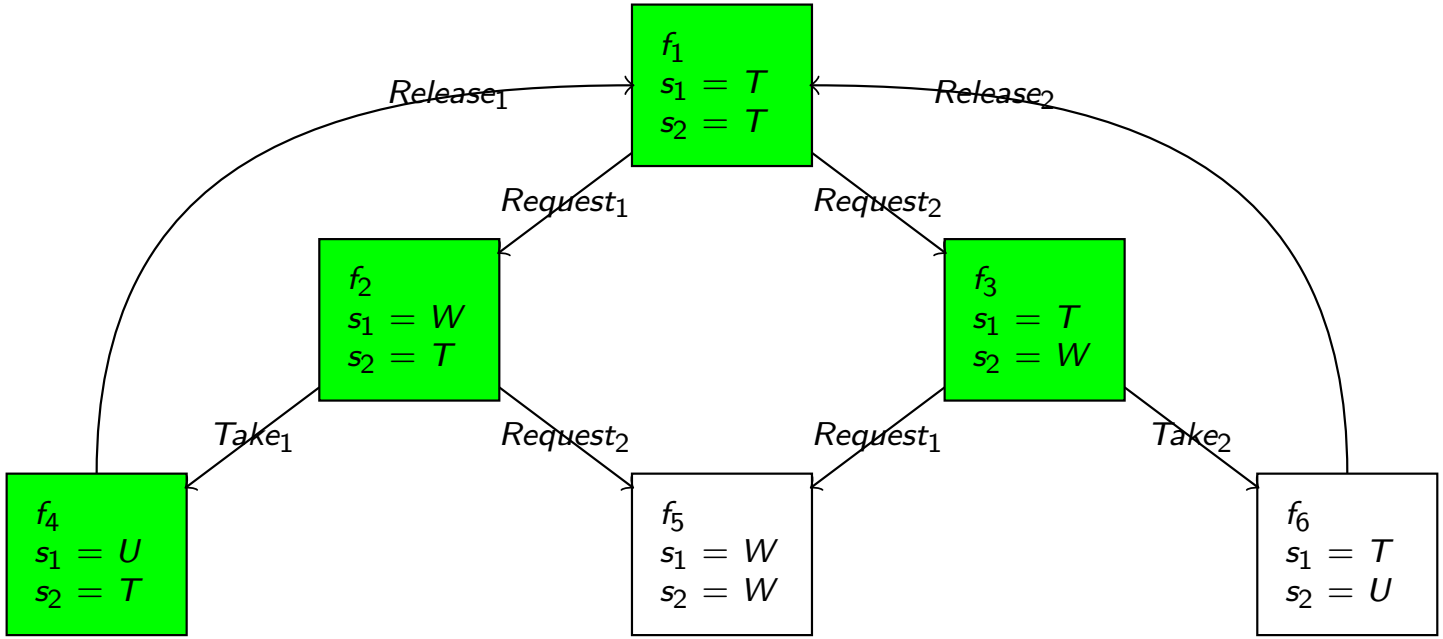
Verification of Property 1 for Example 2



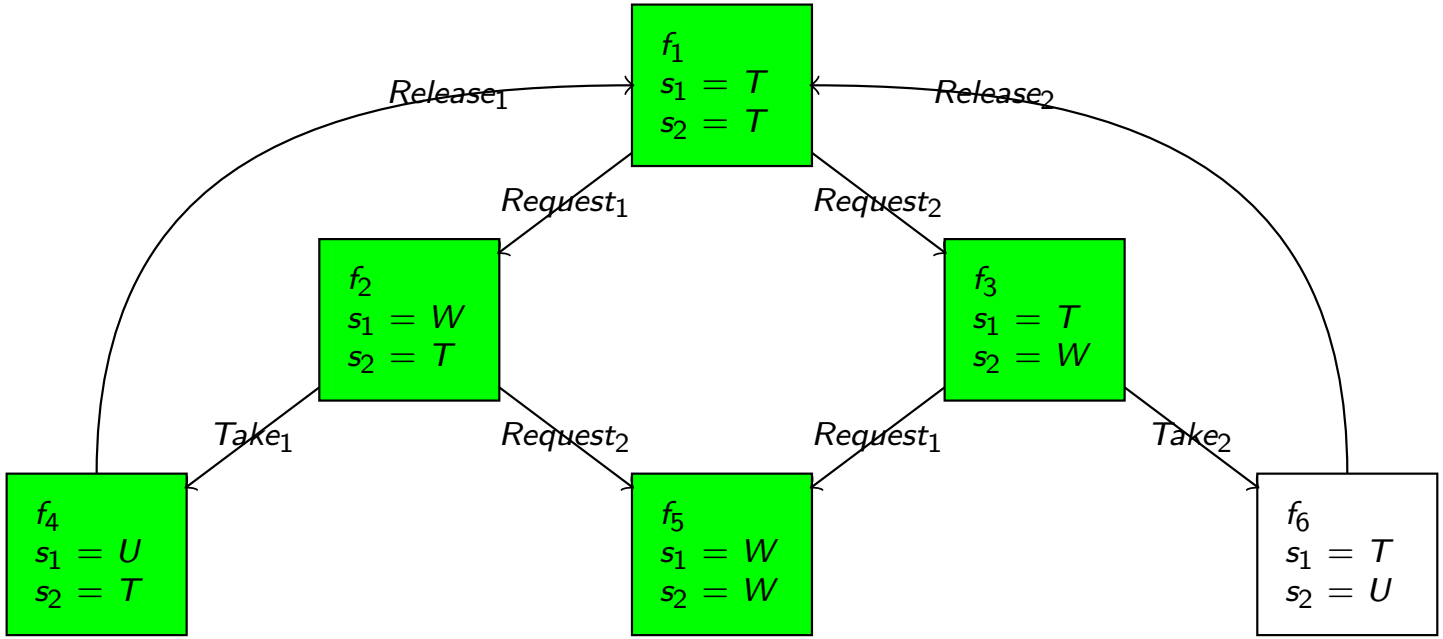
Verification of Property 1 for Example 2



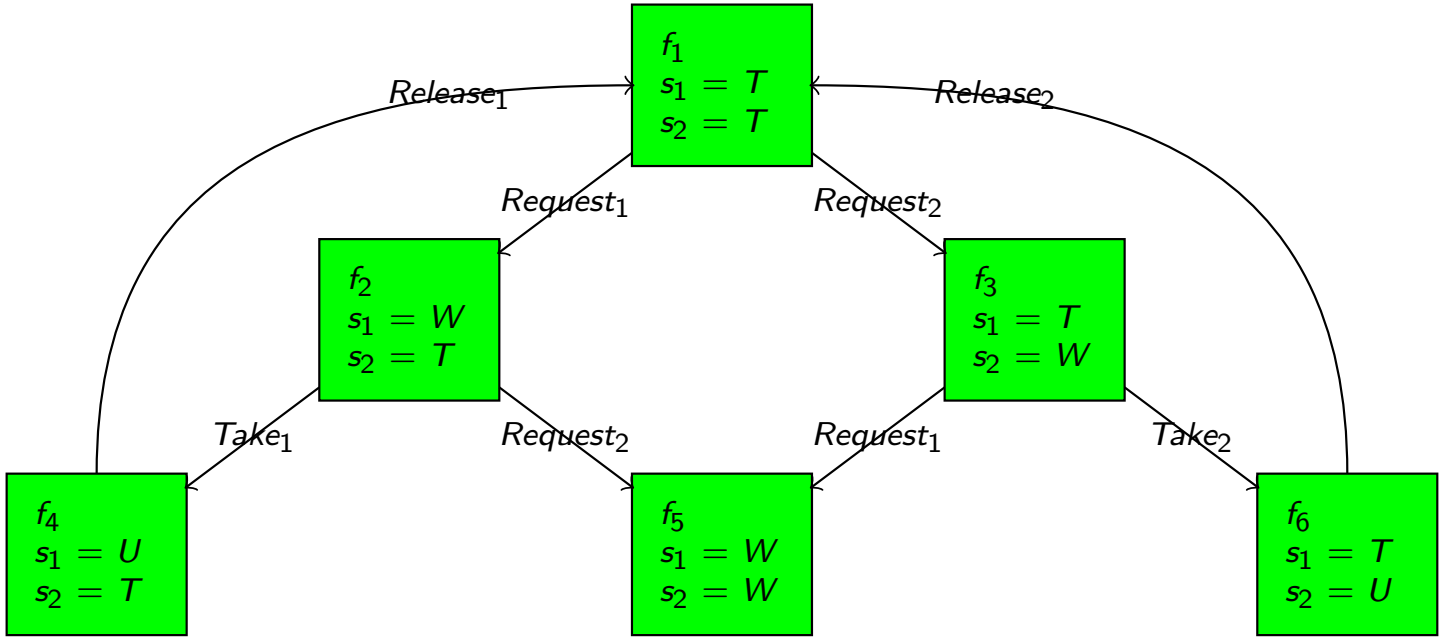
Verification of Property 1 for Example 2



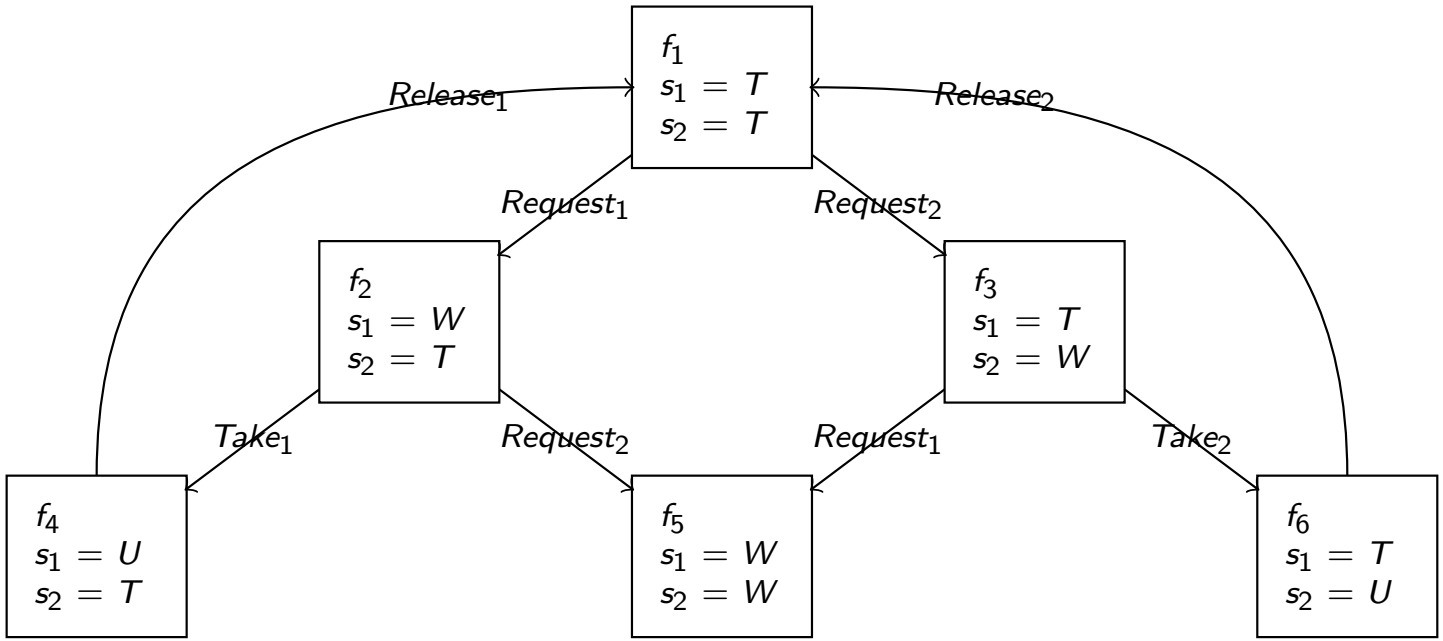
Verification of Property 1 for Example 2



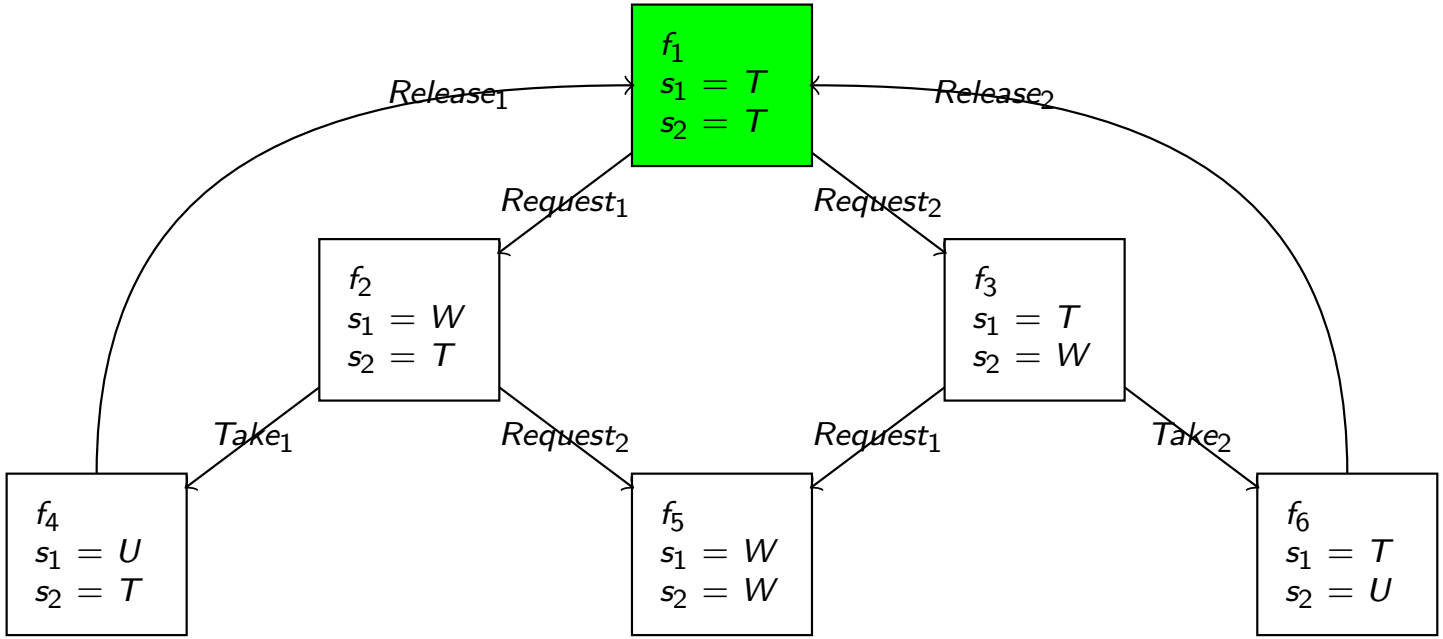
Verification of Property 1 for Example 2



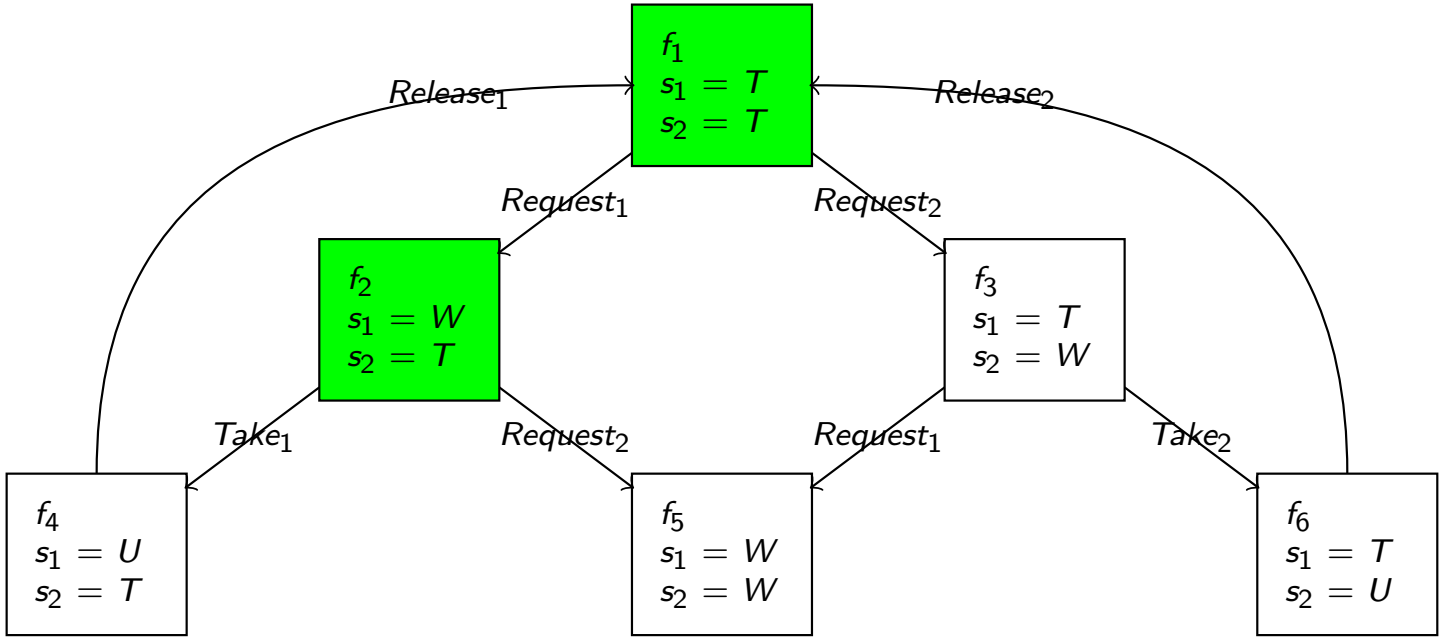
Verification of Property 2 for Example 2



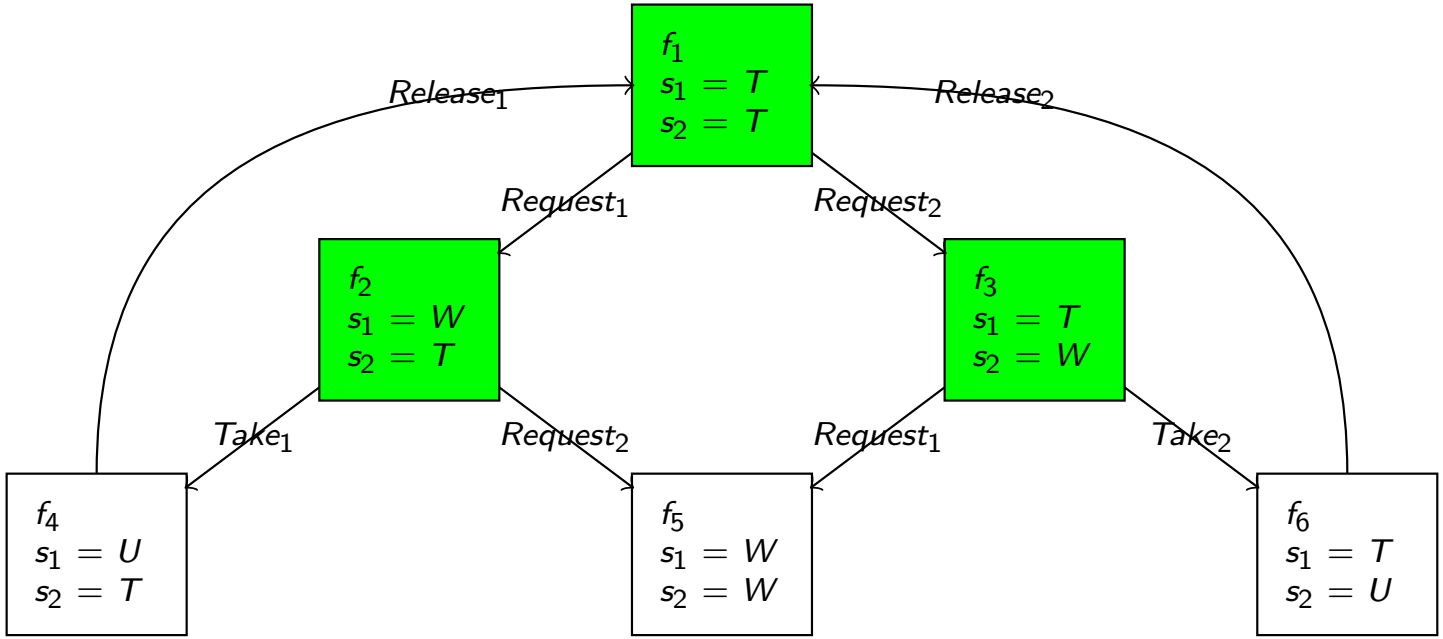
Verification of Property 2 for Example 2



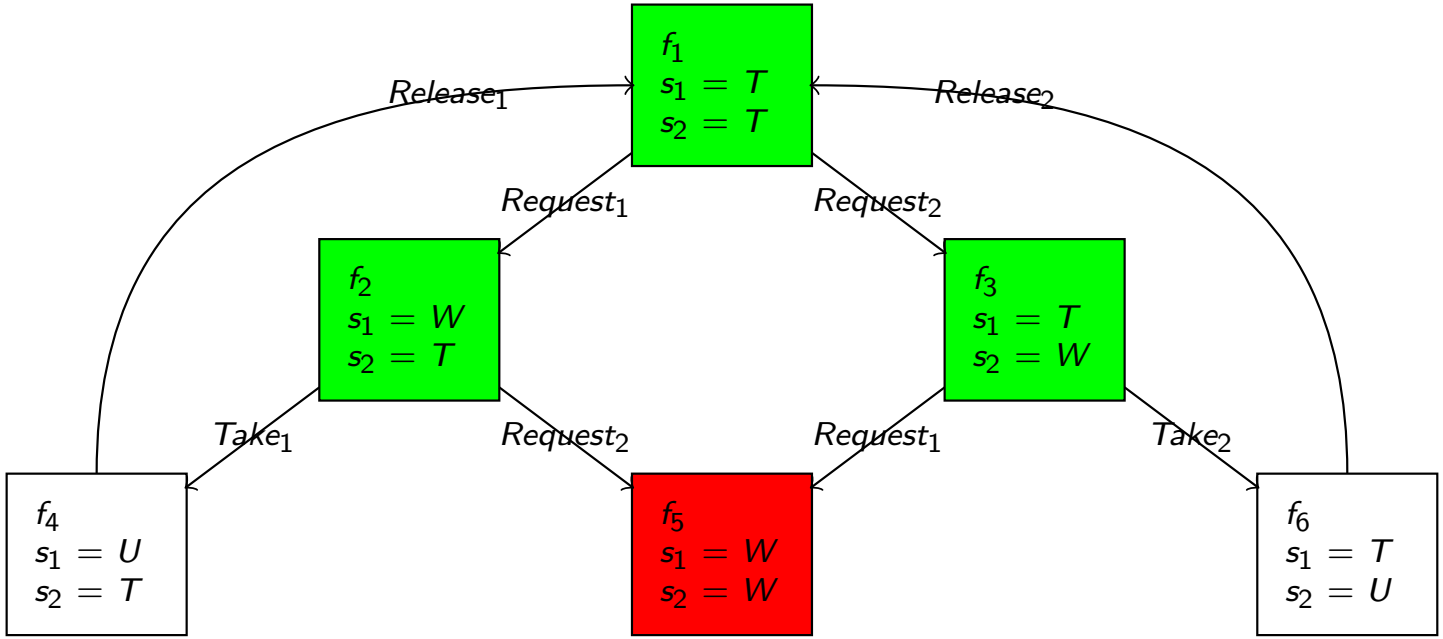
Verification of Property 2 for Example 2



Verification of Property 2 for Example 2



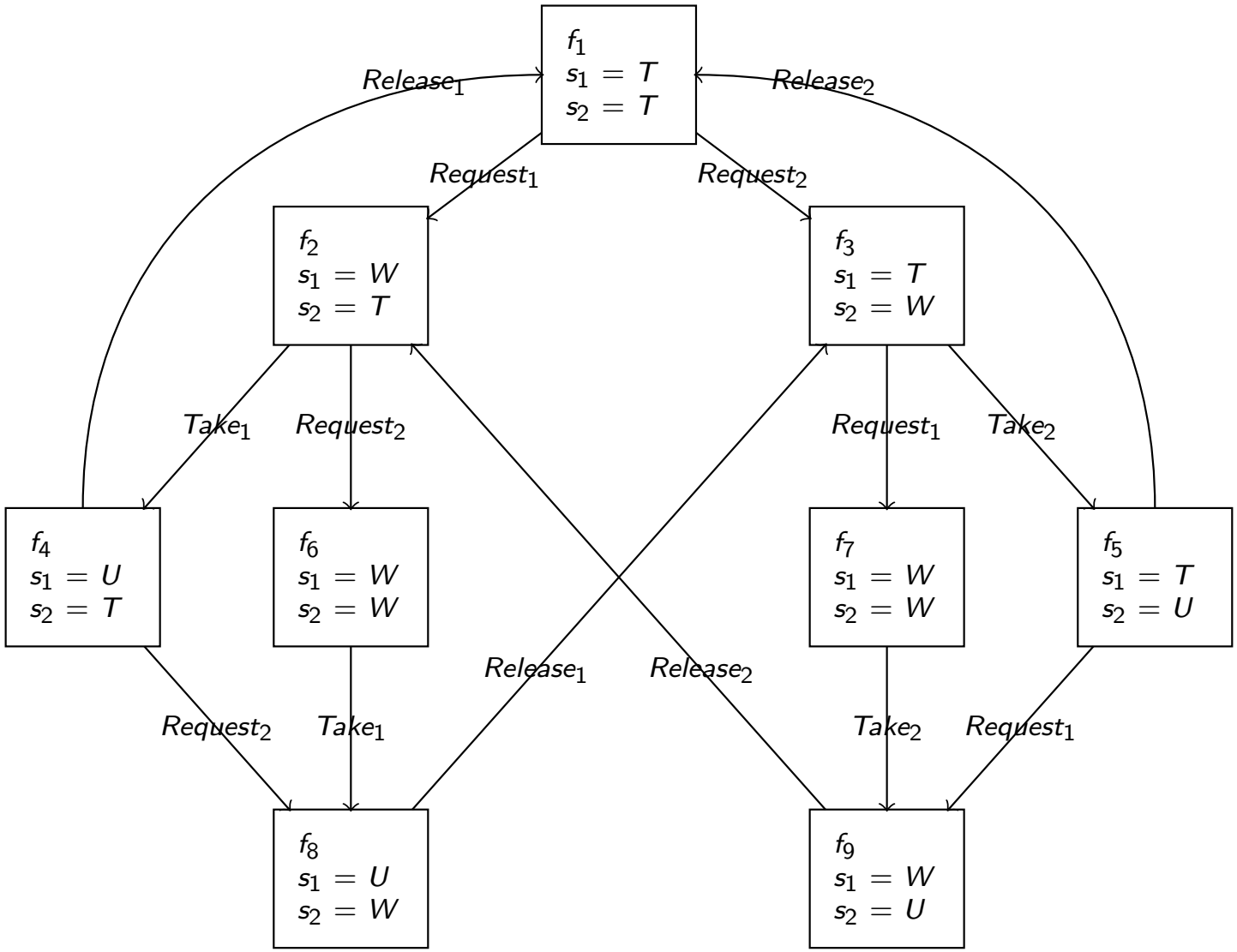
Verification of Property 2 for Example 2



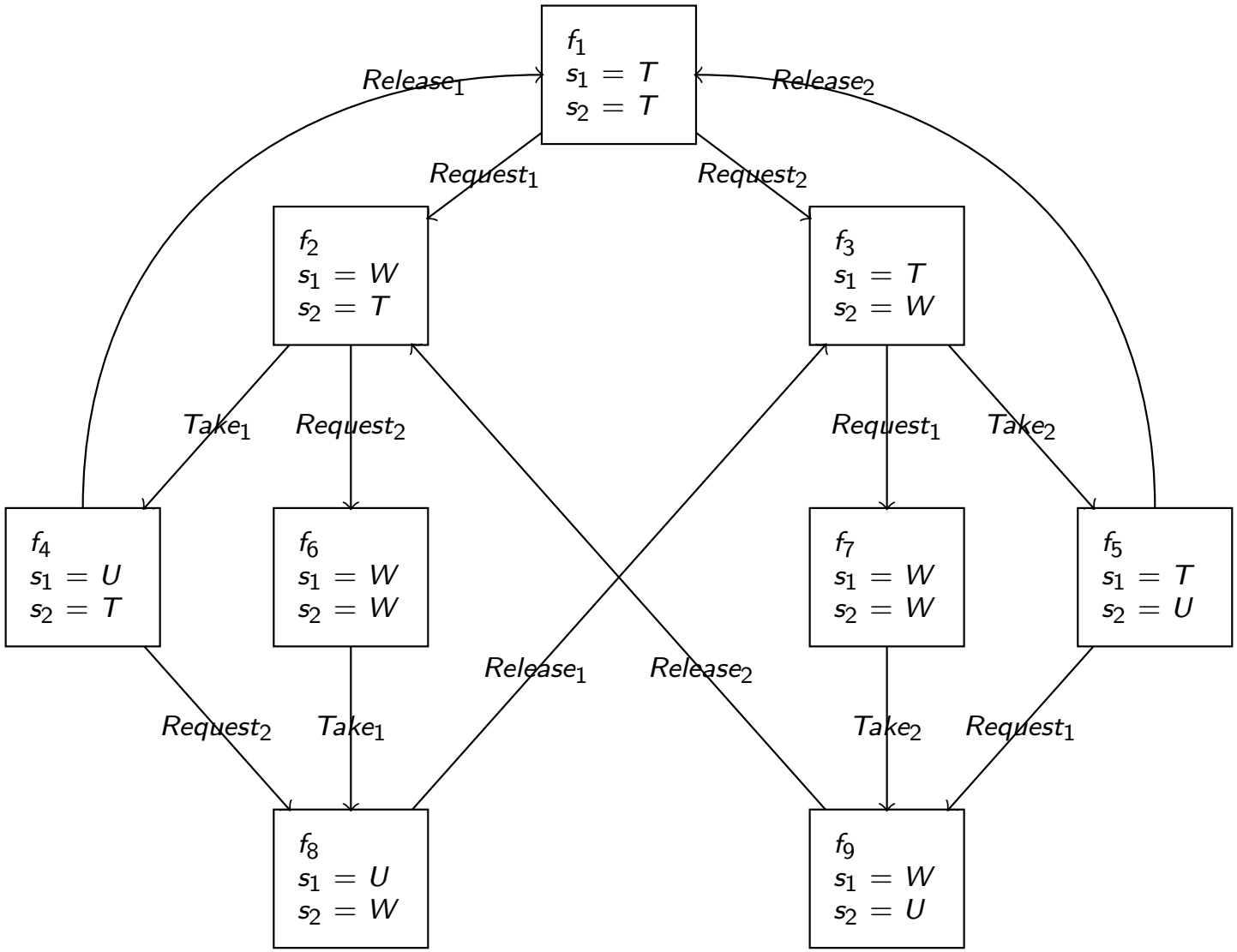
Result of Transforming Example 3

$$
\begin{array}{l}
f_1\ es \\
\textbf{where} \\
f_1 = \lambda es.\mathit{Cons}\ (\mathit{SysState}\ T\ T)\ (\textbf{case}\ es\ \textbf{of} \\
\qquad \mathit{Cons}\ e\ es \rightarrow \textbf{case}\ e\ \textbf{of} \\
\qquad\qquad \mathit{Request}_1 \rightarrow f_2\ es \\
\qquad\qquad \mid \mathit{Request}_2 \rightarrow f_3\ es \\
\qquad\qquad \mid \_ \rightarrow f_1\ es) \\
f_2 = \lambda es.\mathit{Cons}\ (\mathit{SysState}\ W\ T)\ (\textbf{case}\ es\ \textbf{of} \\
\qquad \mathit{Cons}\ e\ es \rightarrow \textbf{case}\ e\ \textbf{of} \\
\qquad\qquad \mathit{Take}_1 \rightarrow f_4\ es \\
\qquad\qquad \mid \mathit{Request}_2 \rightarrow f_6\ es \\
\qquad\qquad \mid \_ \rightarrow f_2\ es) \\
f_3 = \lambda es.\mathit{Cons}\ (\mathit{SysState}\ T\ W)\ (\textbf{case}\ es\ \textbf{of} \\
\qquad \mathit{Cons}\ e\ es \rightarrow \textbf{case}\ e\ \textbf{of} \\
\qquad\qquad \mathit{Take}_2 \rightarrow f_5\ es \\
\qquad\qquad \mid \mathit{Request}_1 \rightarrow f_7\ es \\
\qquad\qquad \mid \_ \rightarrow f_3\ es) \\
f_4 = \lambda es.\mathit{Cons}\ (\mathit{SysState}\ U\ T)\ (\textbf{case}\ es\ \textbf{of} \\
\qquad \mathit{Cons}\ e\ es \rightarrow \textbf{case}\ e\ \textbf{of} \\
\qquad\qquad \mathit{Release}_1 \rightarrow f_1\ es \\
\qquad\qquad \mid \mathit{Request}_2 \rightarrow f_8\ es \\
\qquad\qquad \mid \_ \rightarrow f_4\ es) \\
f_5 = \lambda es.\mathit{Cons}\ (\mathit{SysState}\ T\ U)\ (\textbf{case}\ es\ \textbf{of} \\
\qquad \mathit{Cons}\ e\ es \rightarrow \textbf{case}\ e\ \textbf{of} \\
\qquad\qquad \mathit{Release}_2 \rightarrow f_1\ es \\
\qquad\qquad \mid \mathit{Request}_1 \rightarrow f_9\ es \\
\qquad\qquad \mid \_ \rightarrow f_5\ es) \\
f_6 = \lambda es.\mathit{Cons}\ (\mathit{SysState}\ W\ W)\ (\textbf{case}\ es\ \textbf{of} \\
\qquad \mathit{Cons}\ e\ es \rightarrow \textbf{case}\ e\ \textbf{of} \\
\qquad\qquad \mathit{Take}_1 \rightarrow f_8\ es \\
\qquad\qquad \mid \ldots
\end{array}
$$

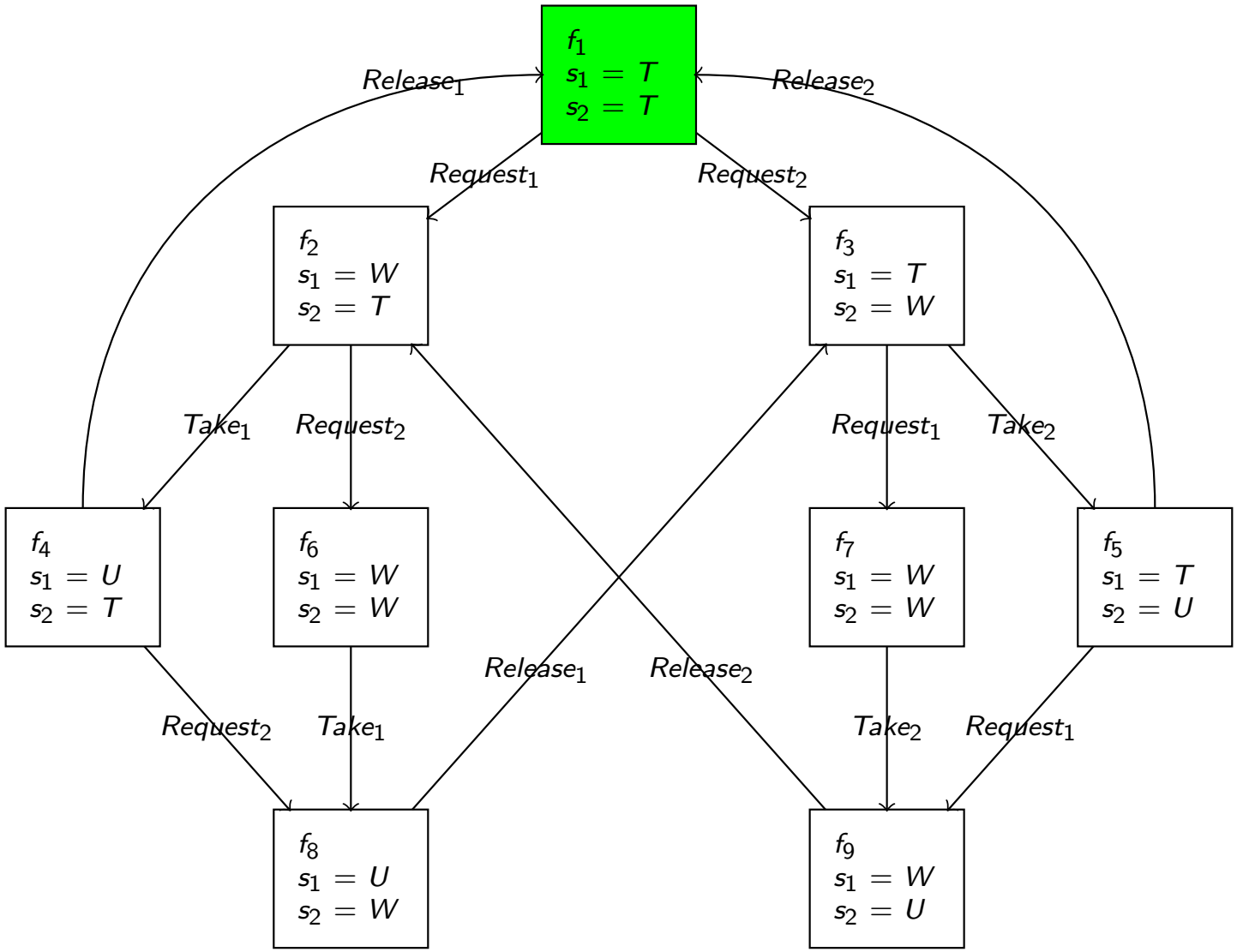
LTS Representation of Transformed Example 3



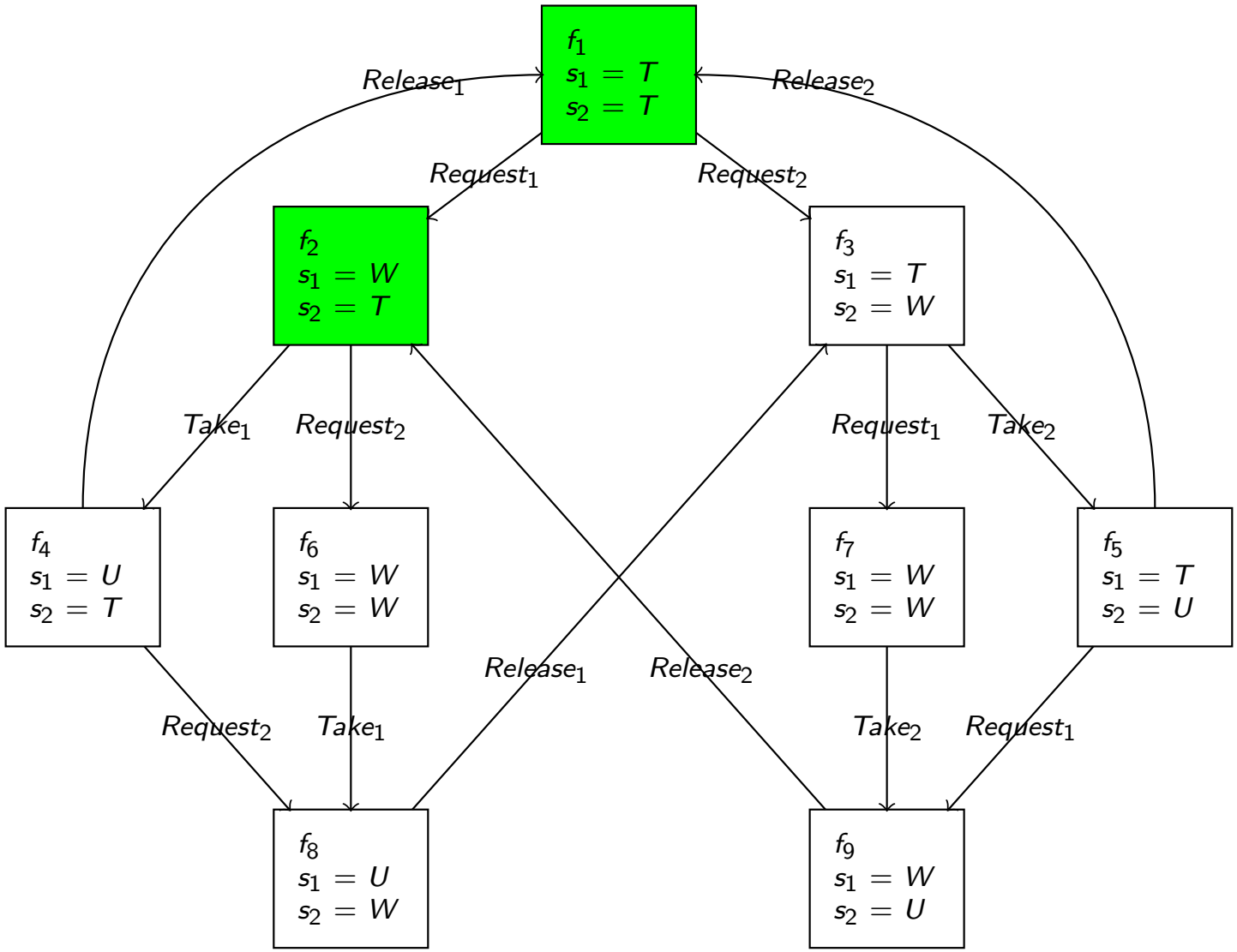
Verification of Property 1 for Example 3



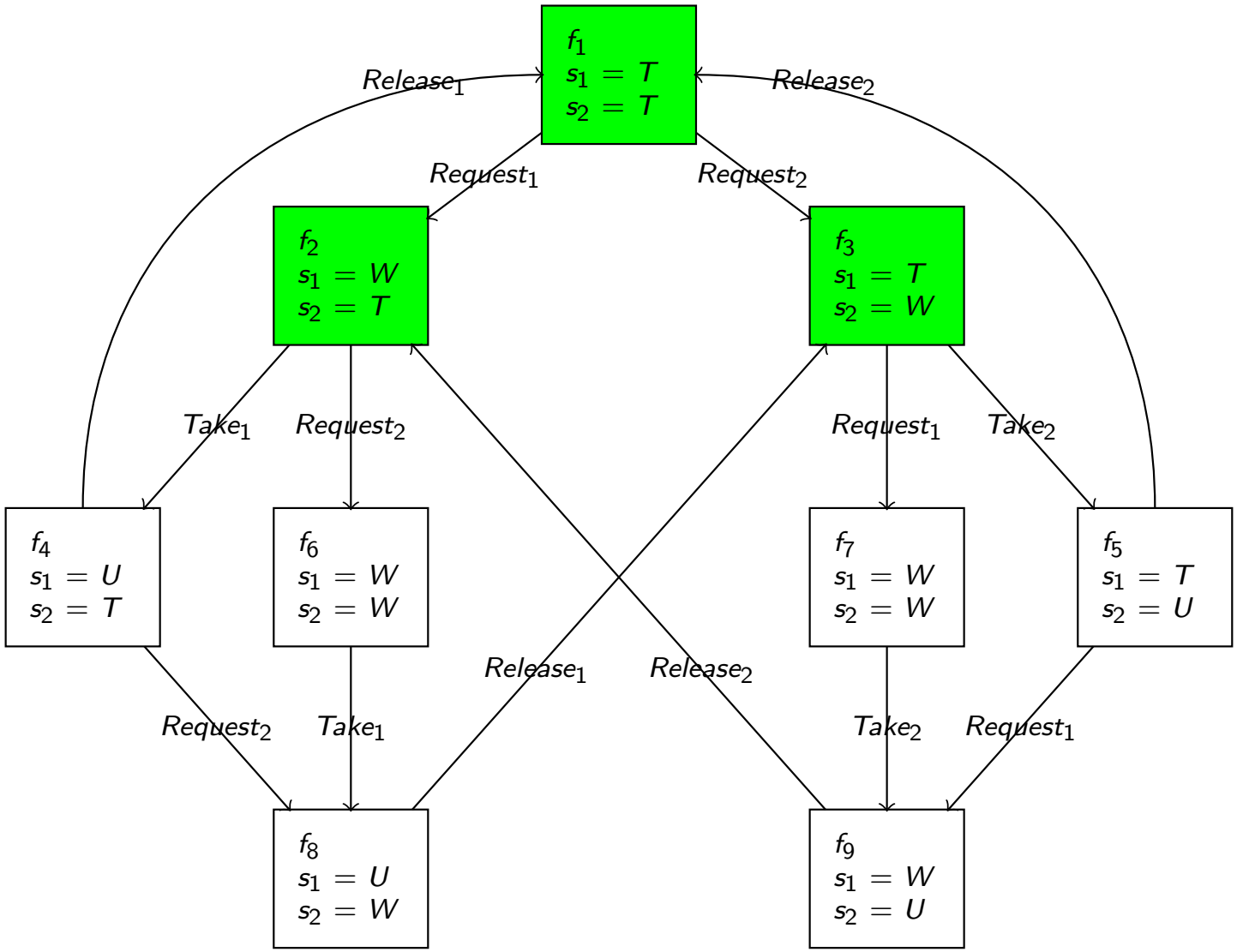
Verification of Property 1 for Example 3



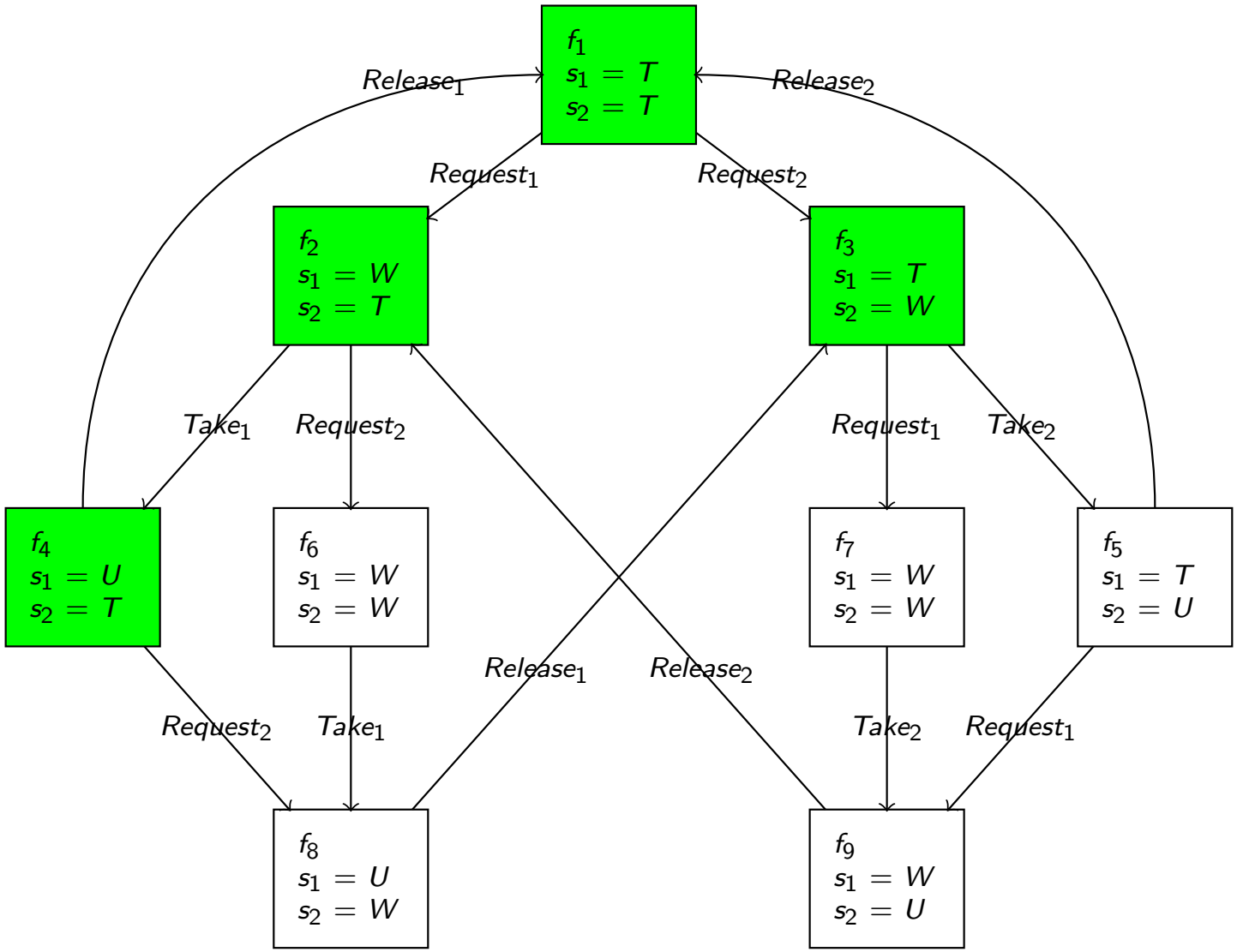
Verification of Property 1 for Example 3



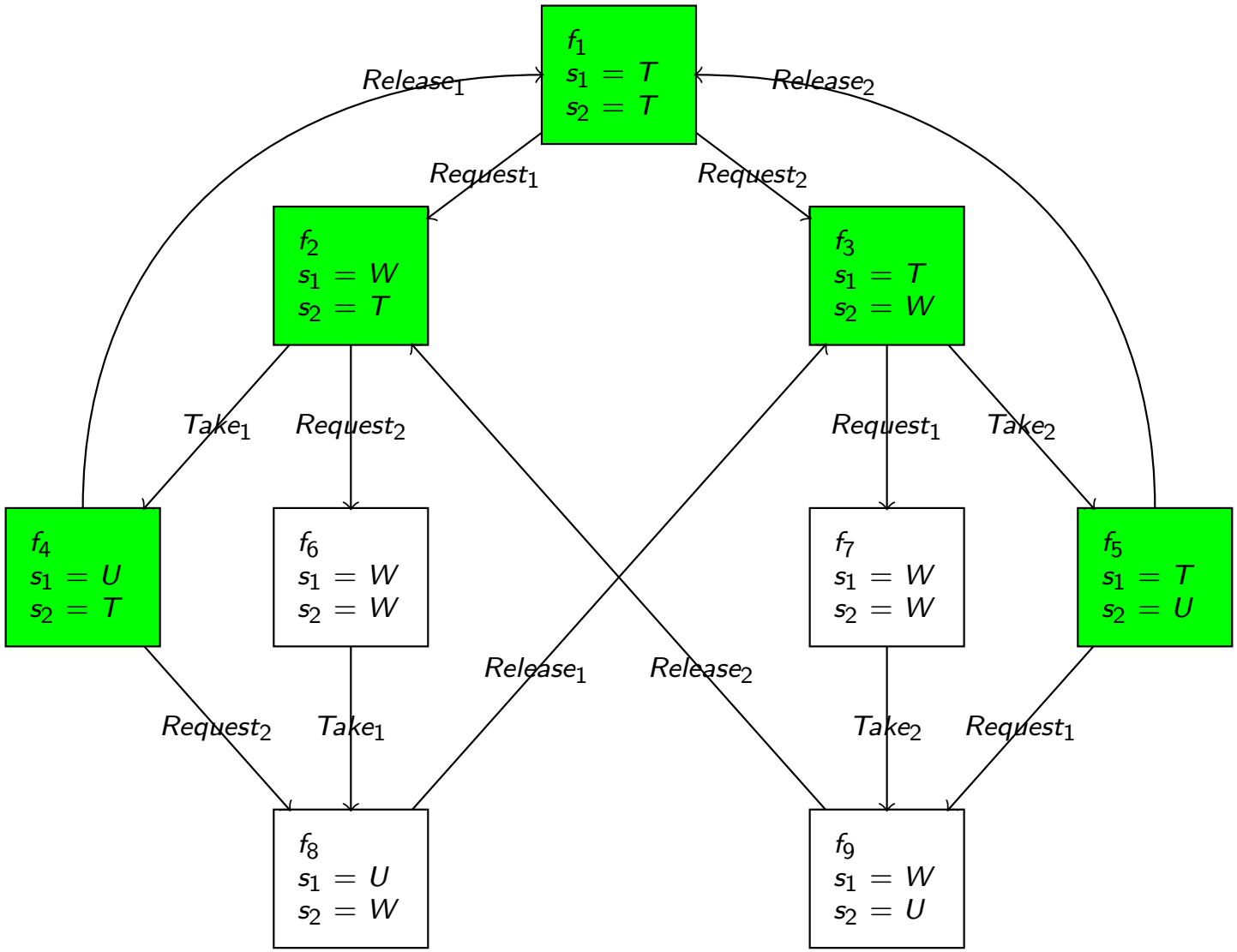
Verification of Property 1 for Example 3



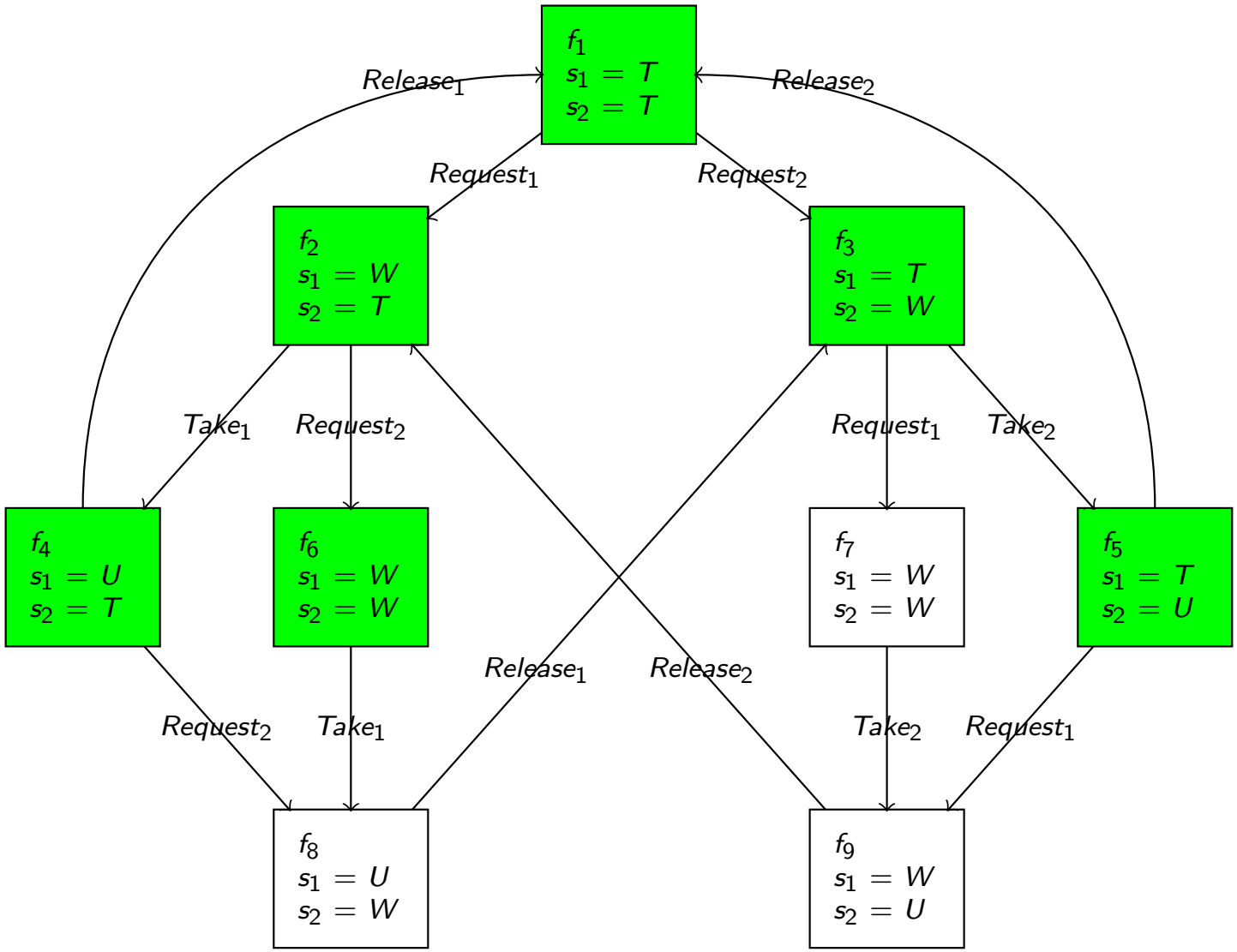
Verification of Property 1 for Example 3



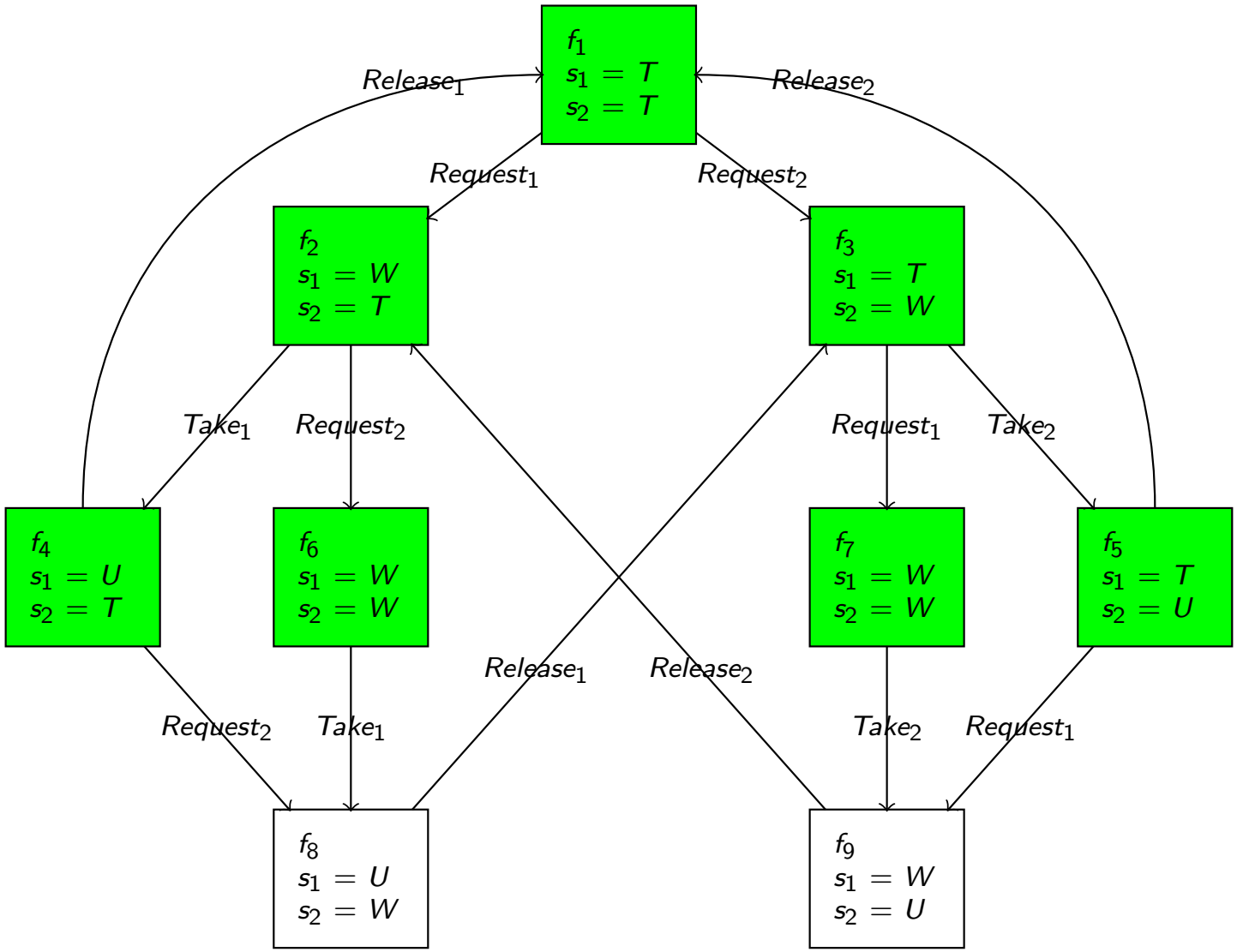
Verification of Property 1 for Example 3



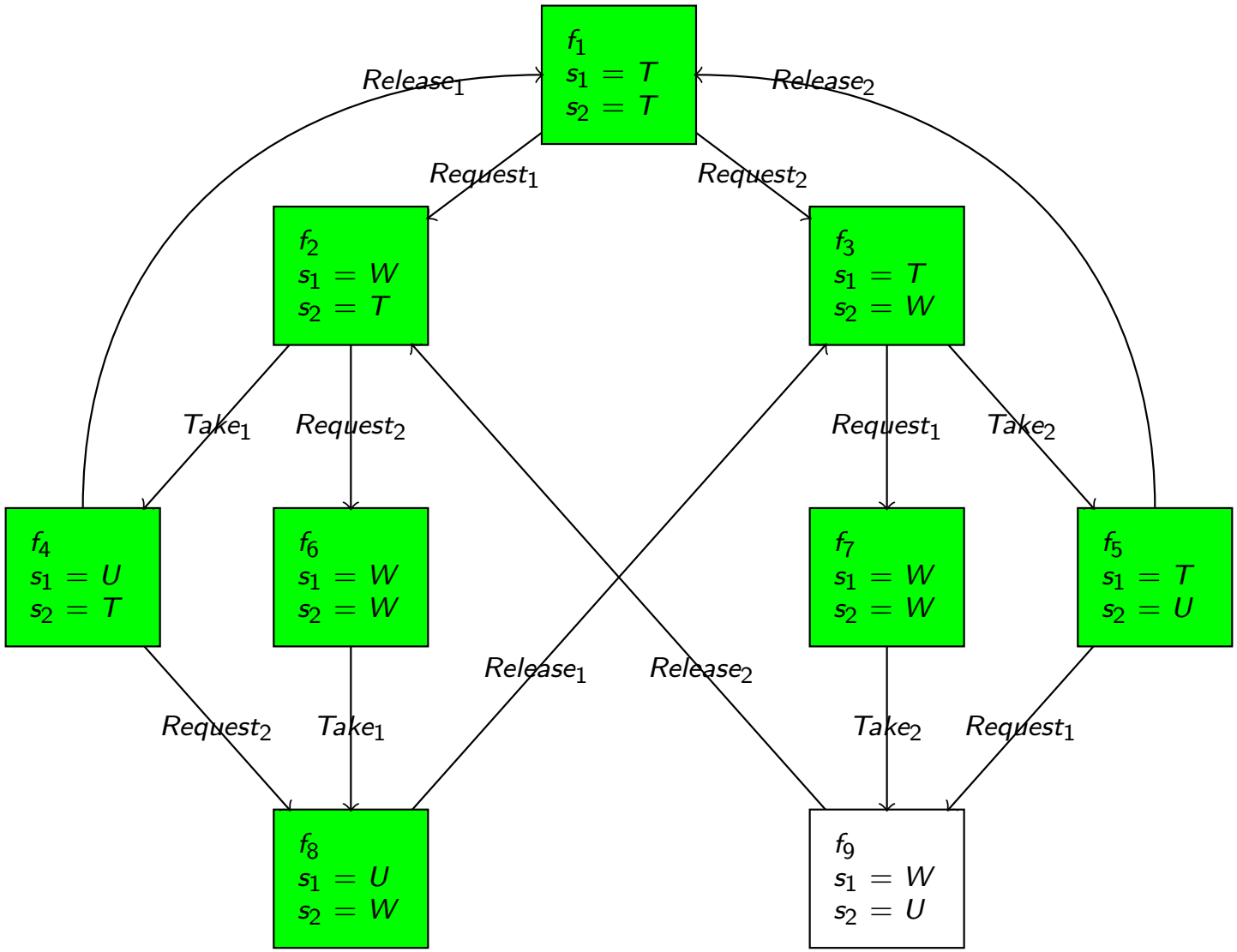
Verification of Property 1 for Example 3



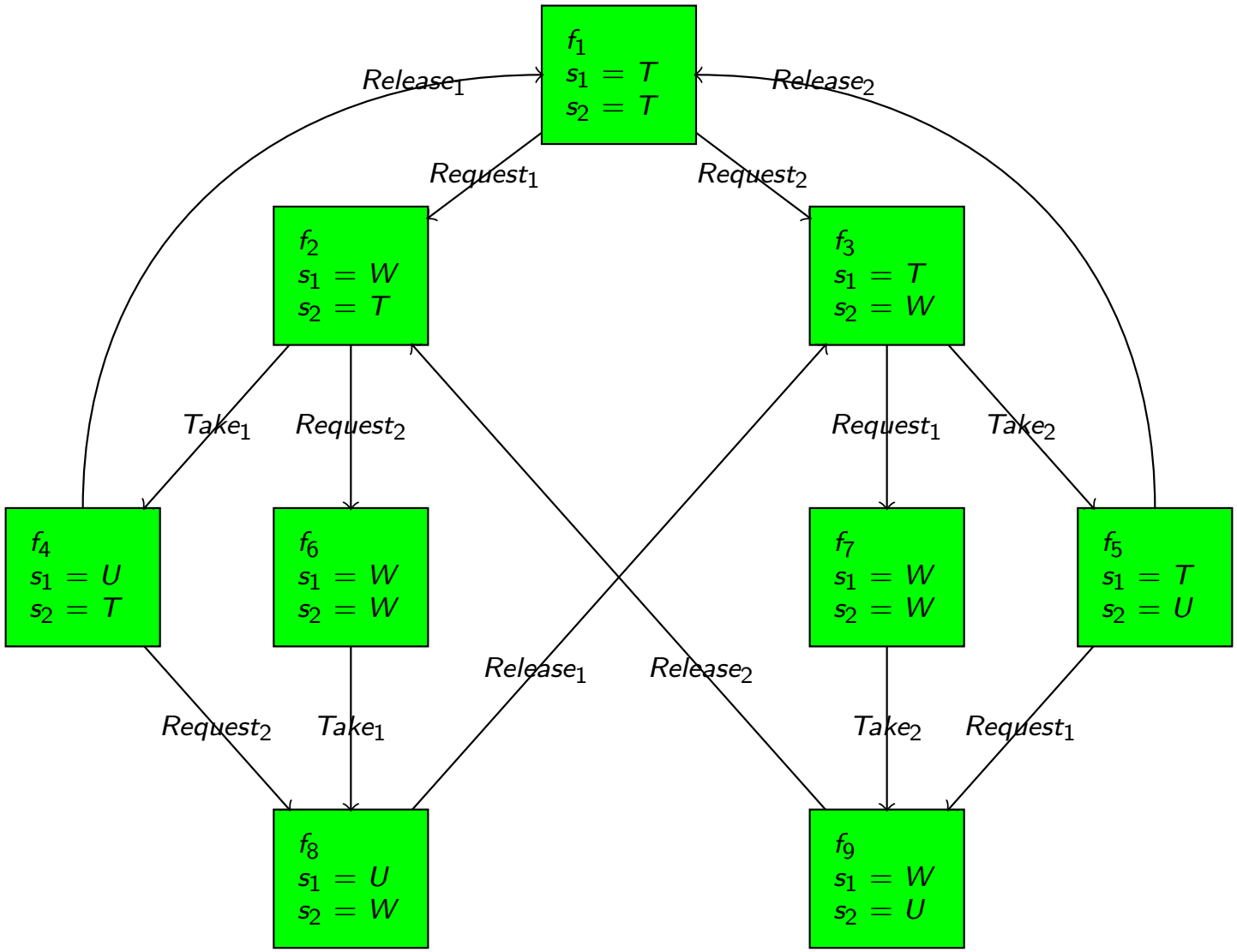
Verification of Property 1 for Example 3



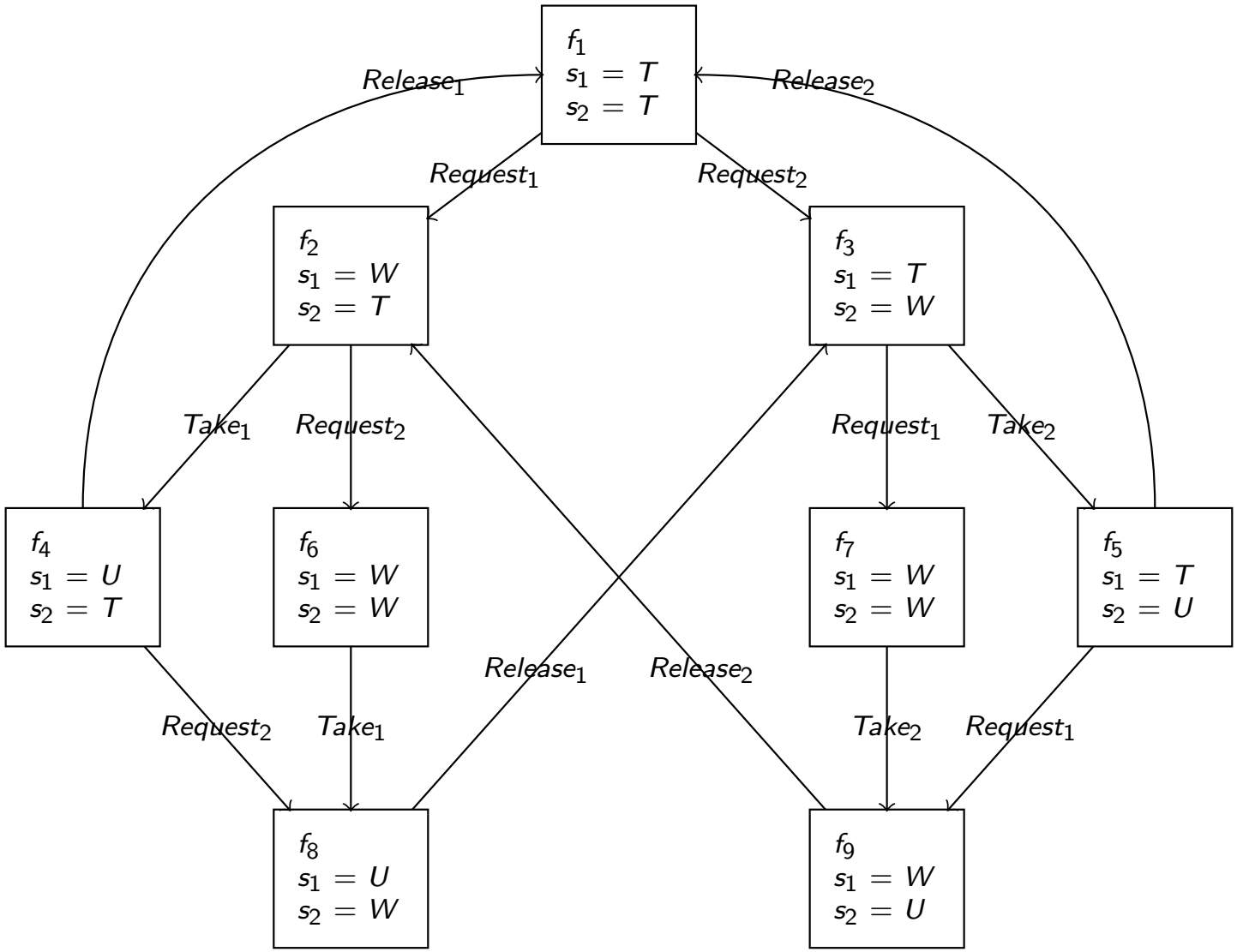
Verification of Property 1 for Example 3



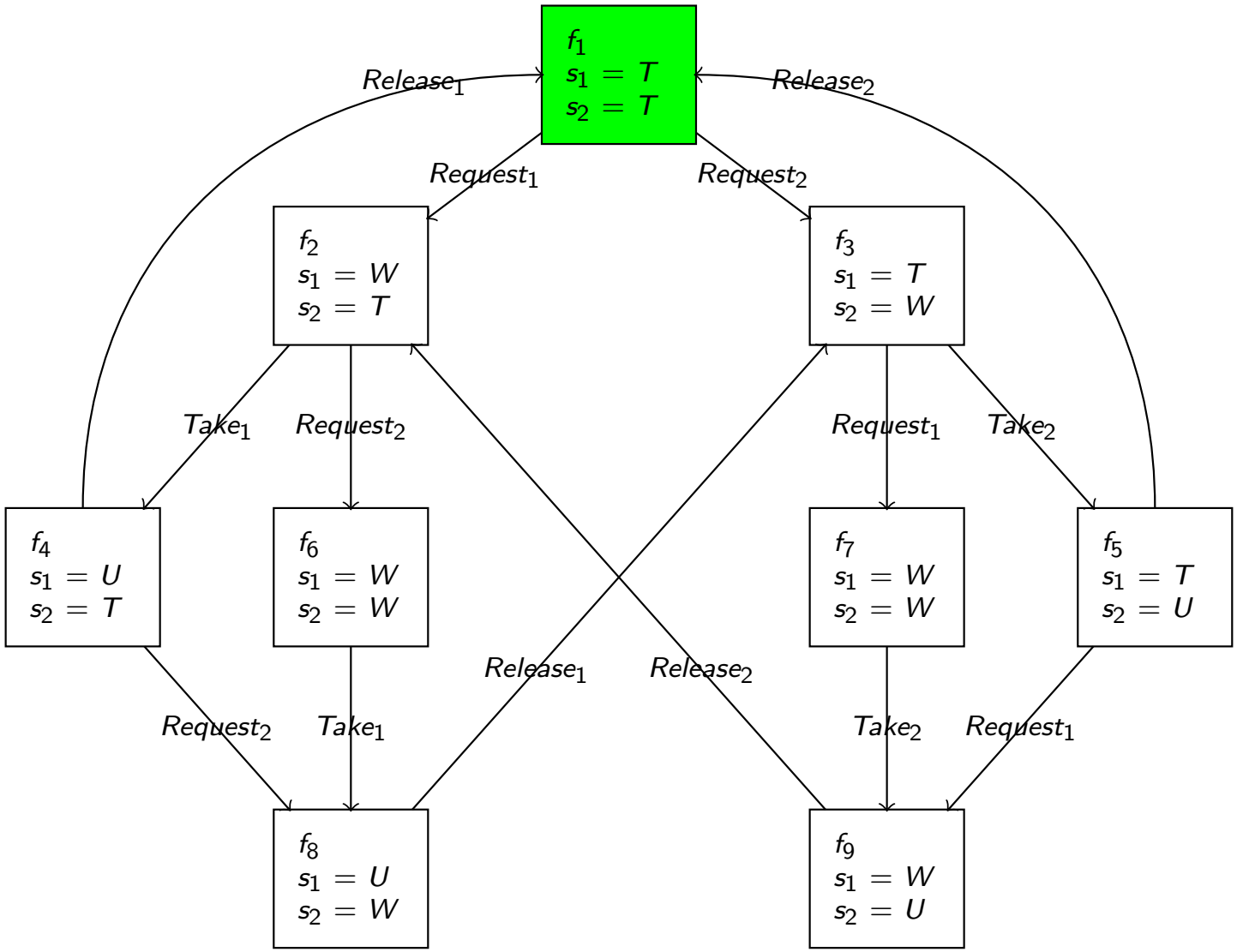
Verification of Property 1 for Example 3



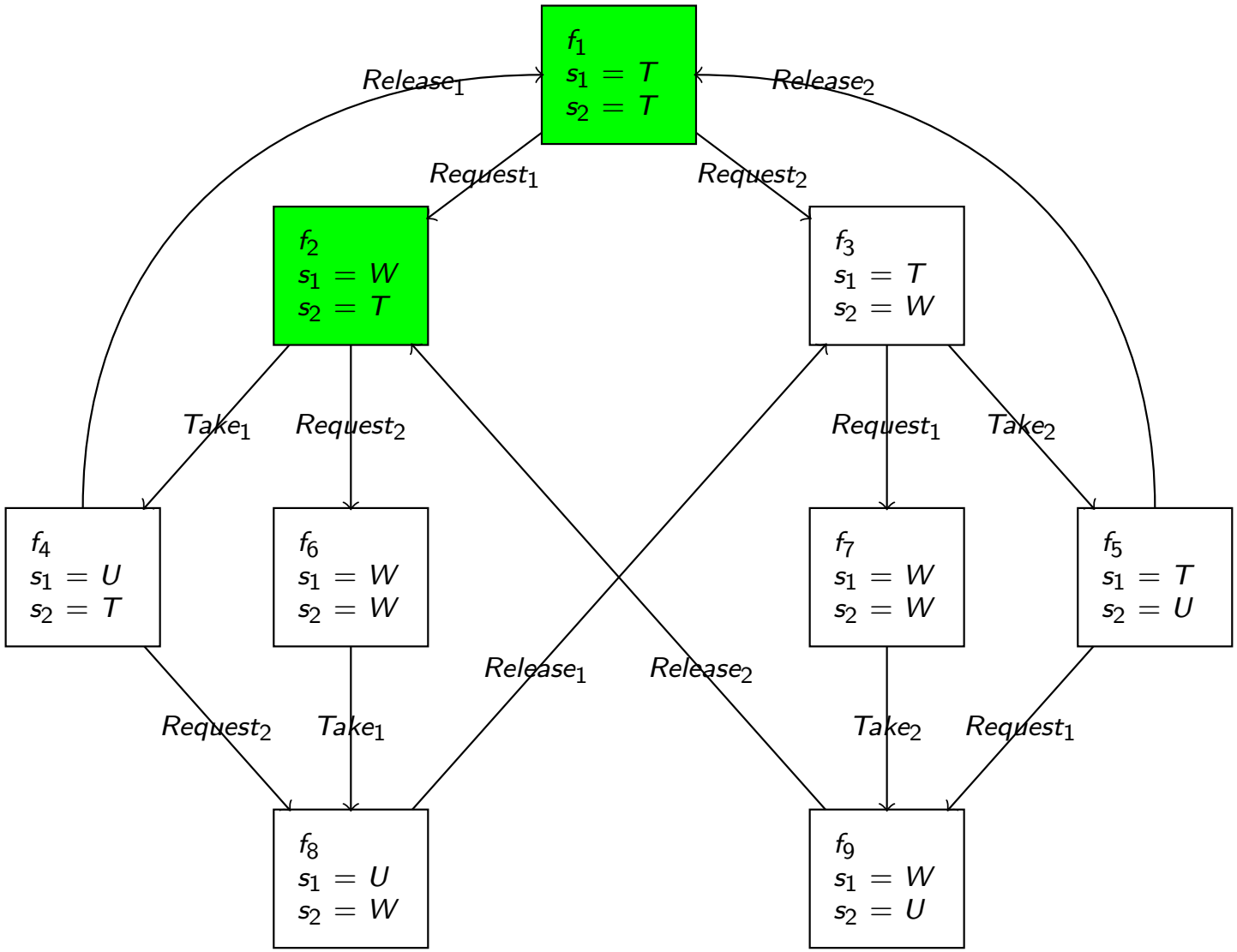
Verification of Property 2 for Example 3



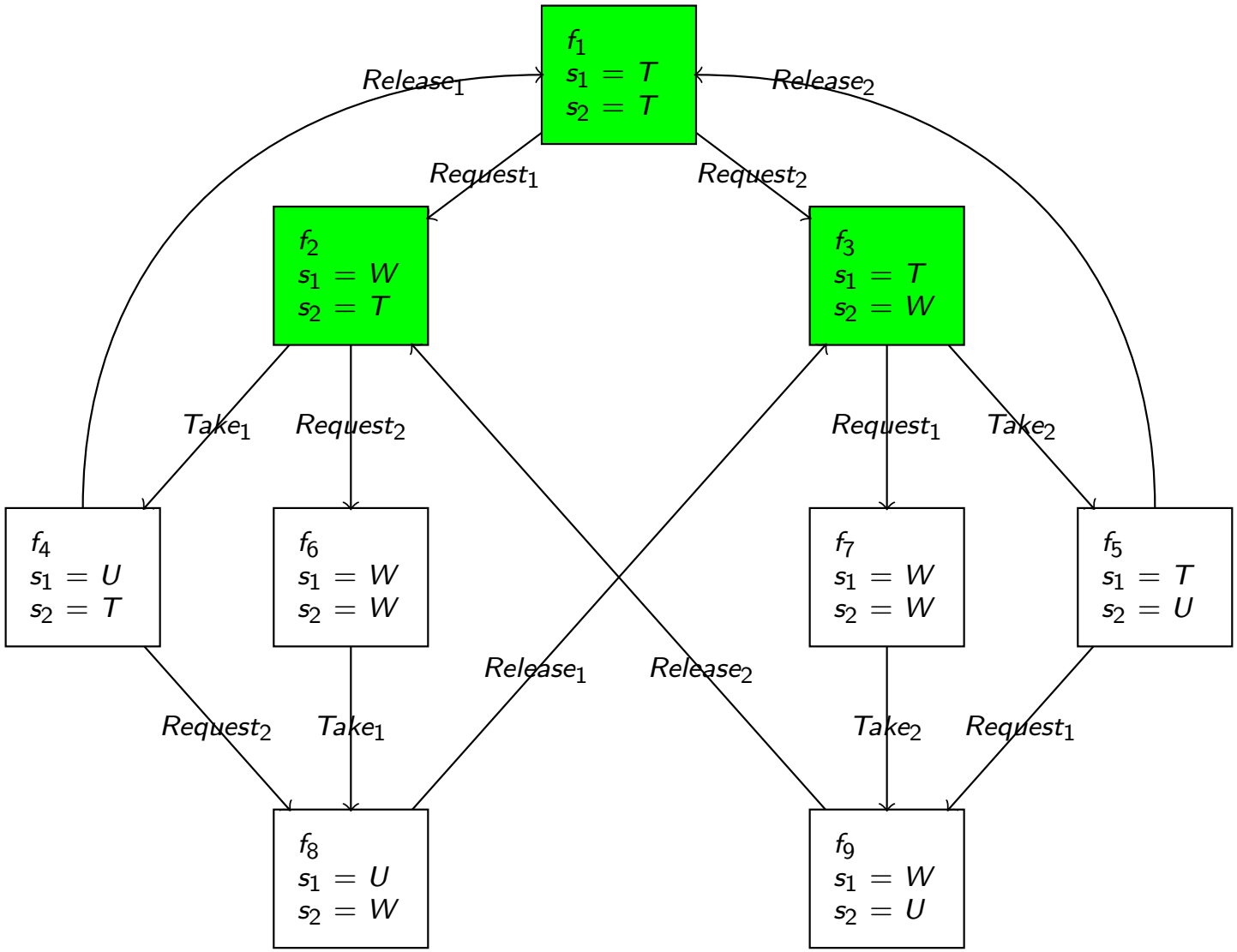
Verification of Property 2 for Example 3



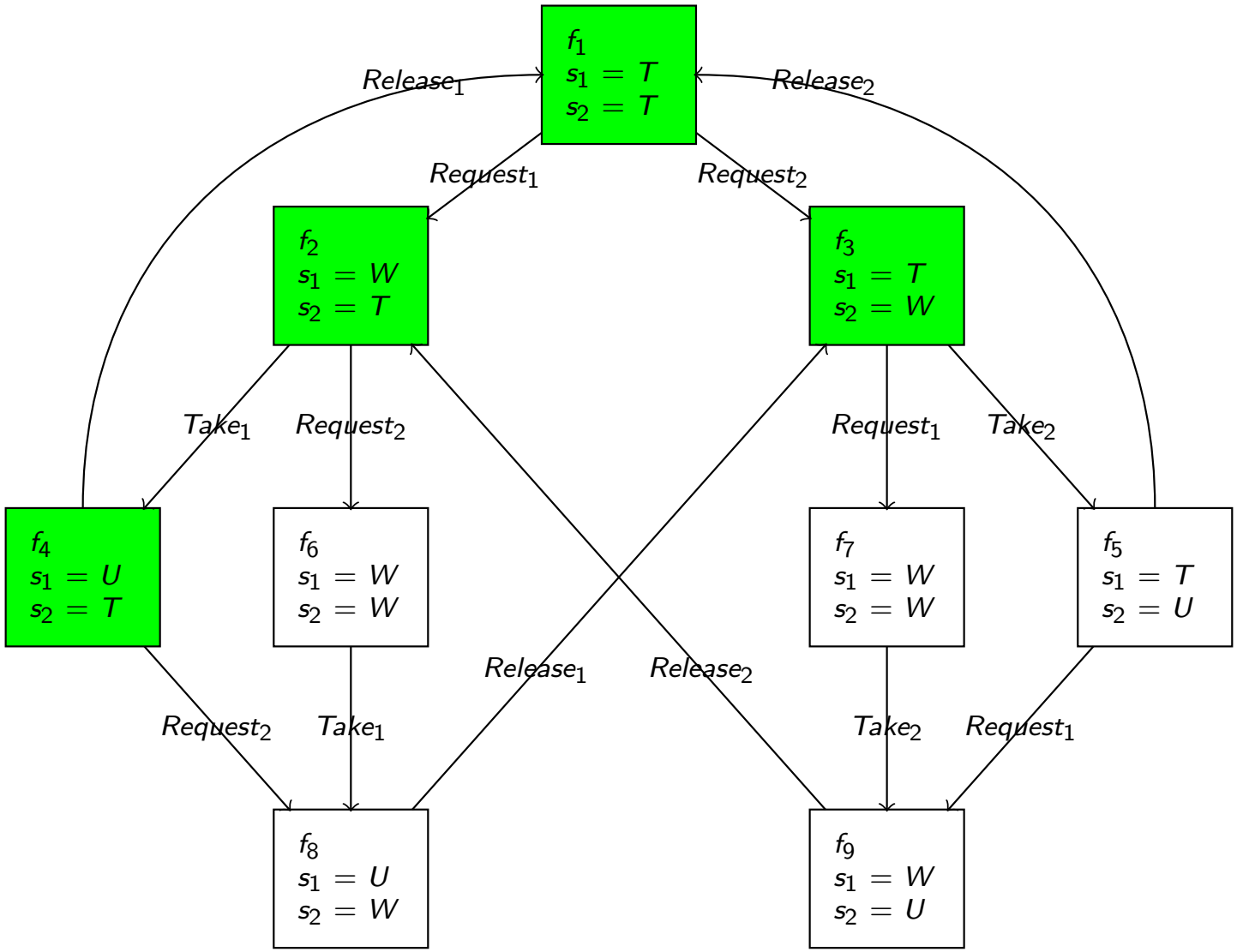
Verification of Property 2 for Example 3



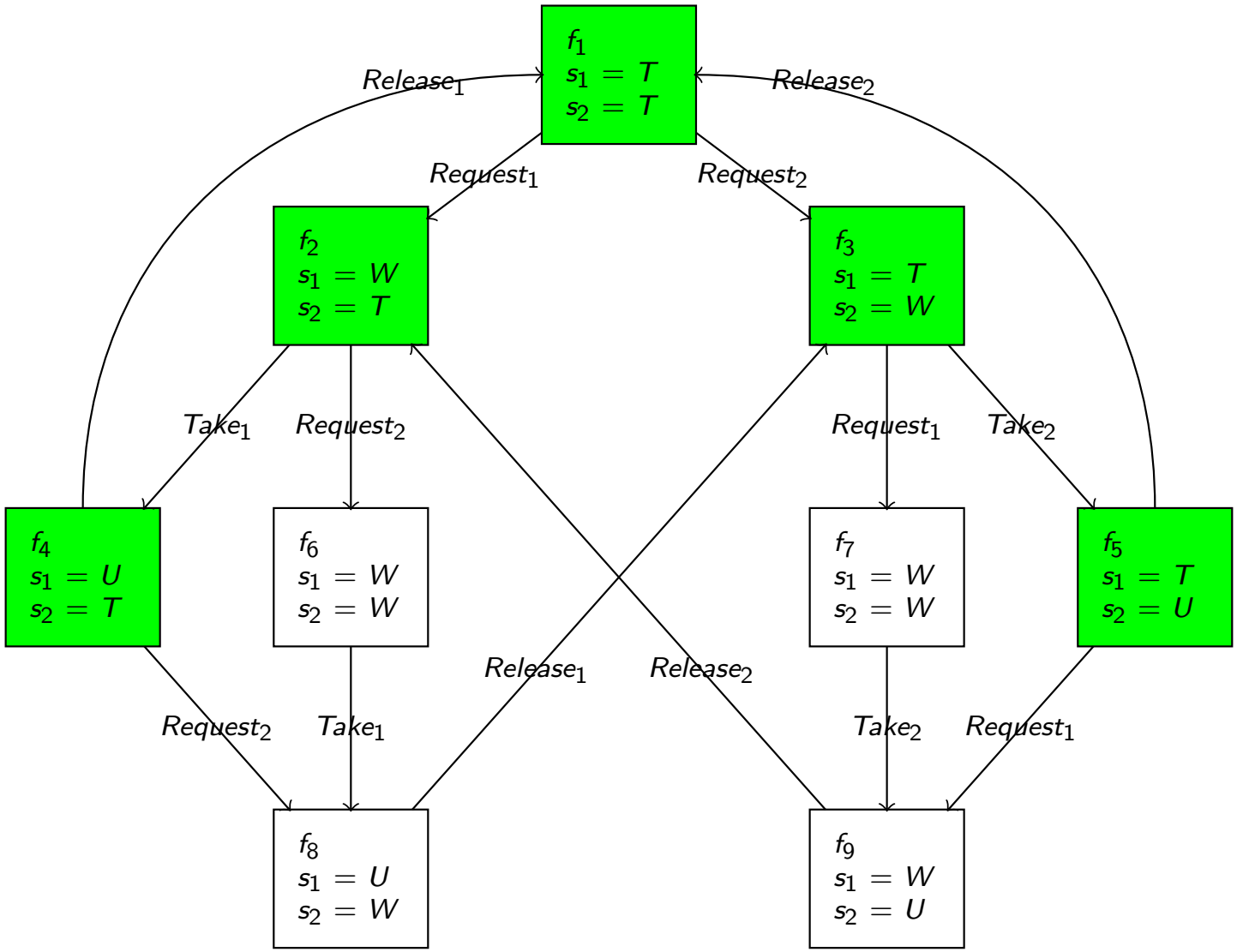
Verification of Property 2 for Example 3



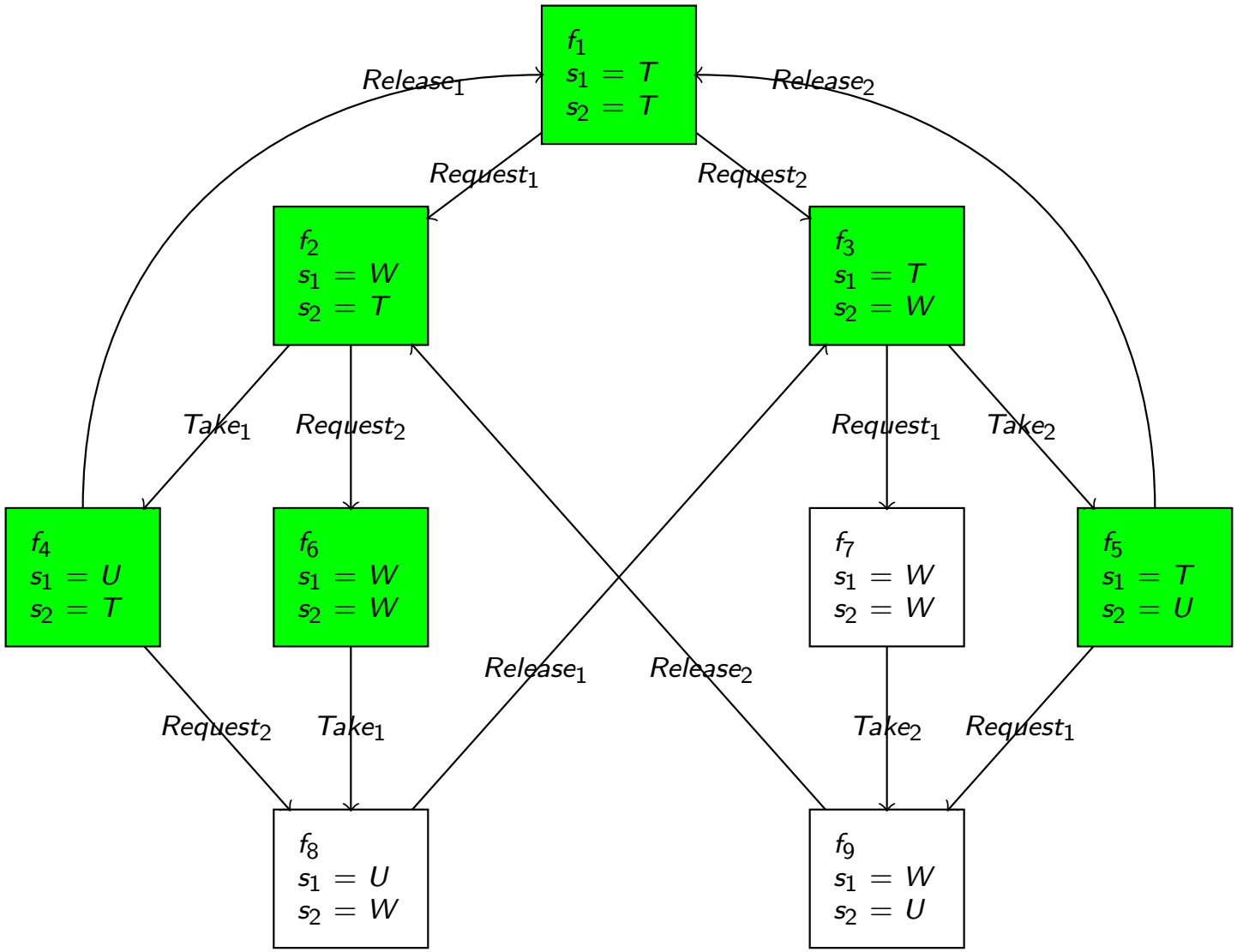
Verification of Property 2 for Example 3



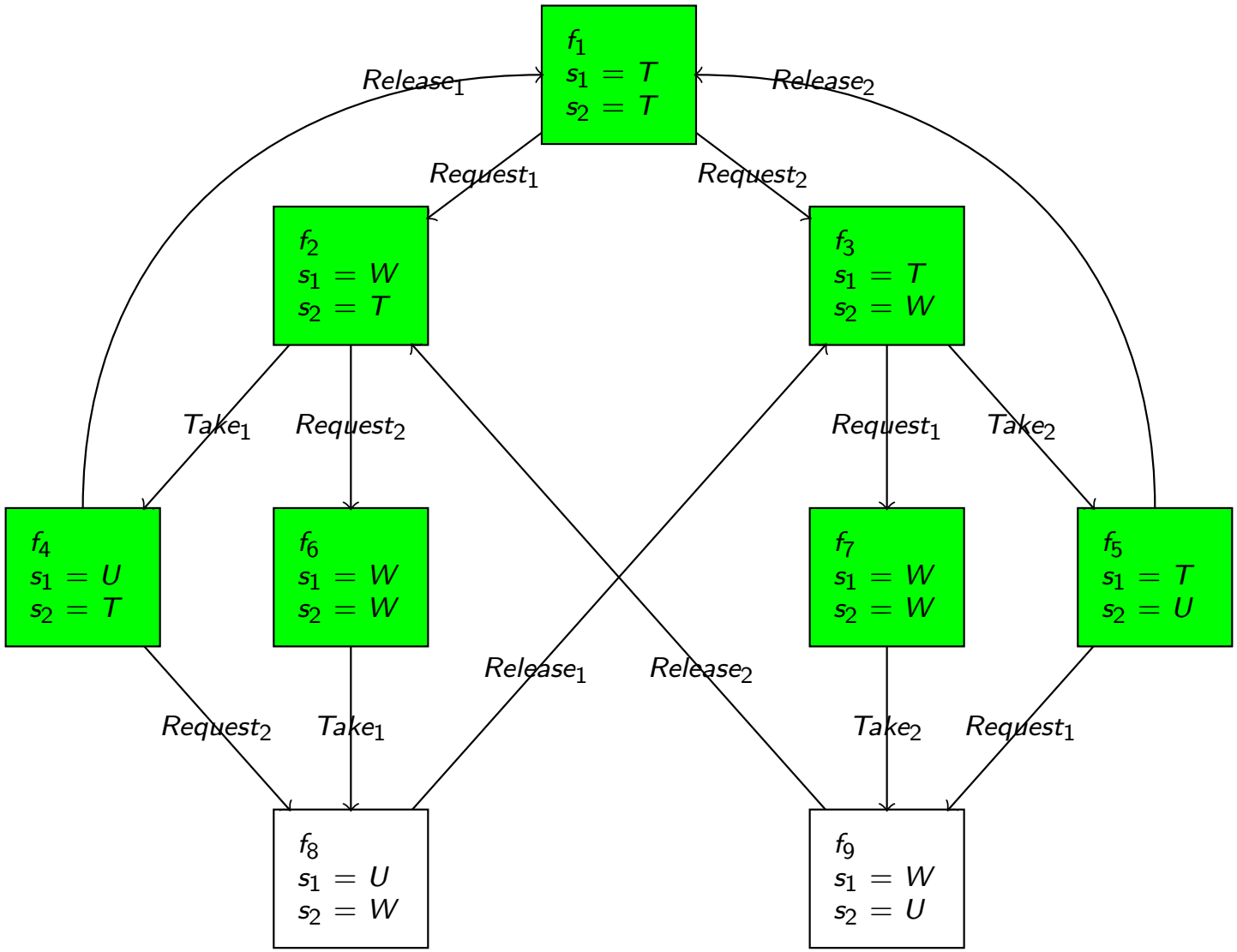
Verification of Property 2 for Example 3



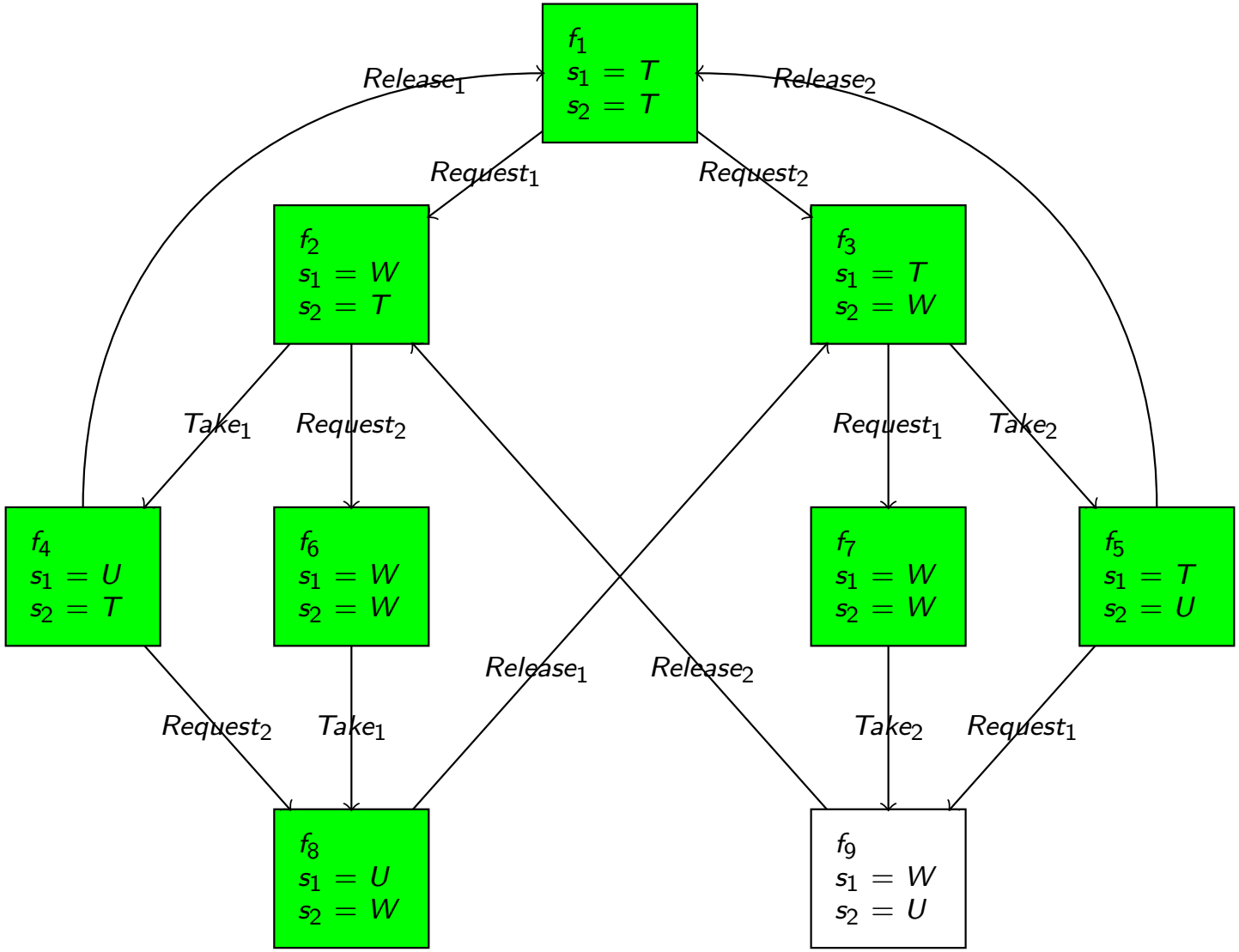
Verification of Property 2 for Example 3



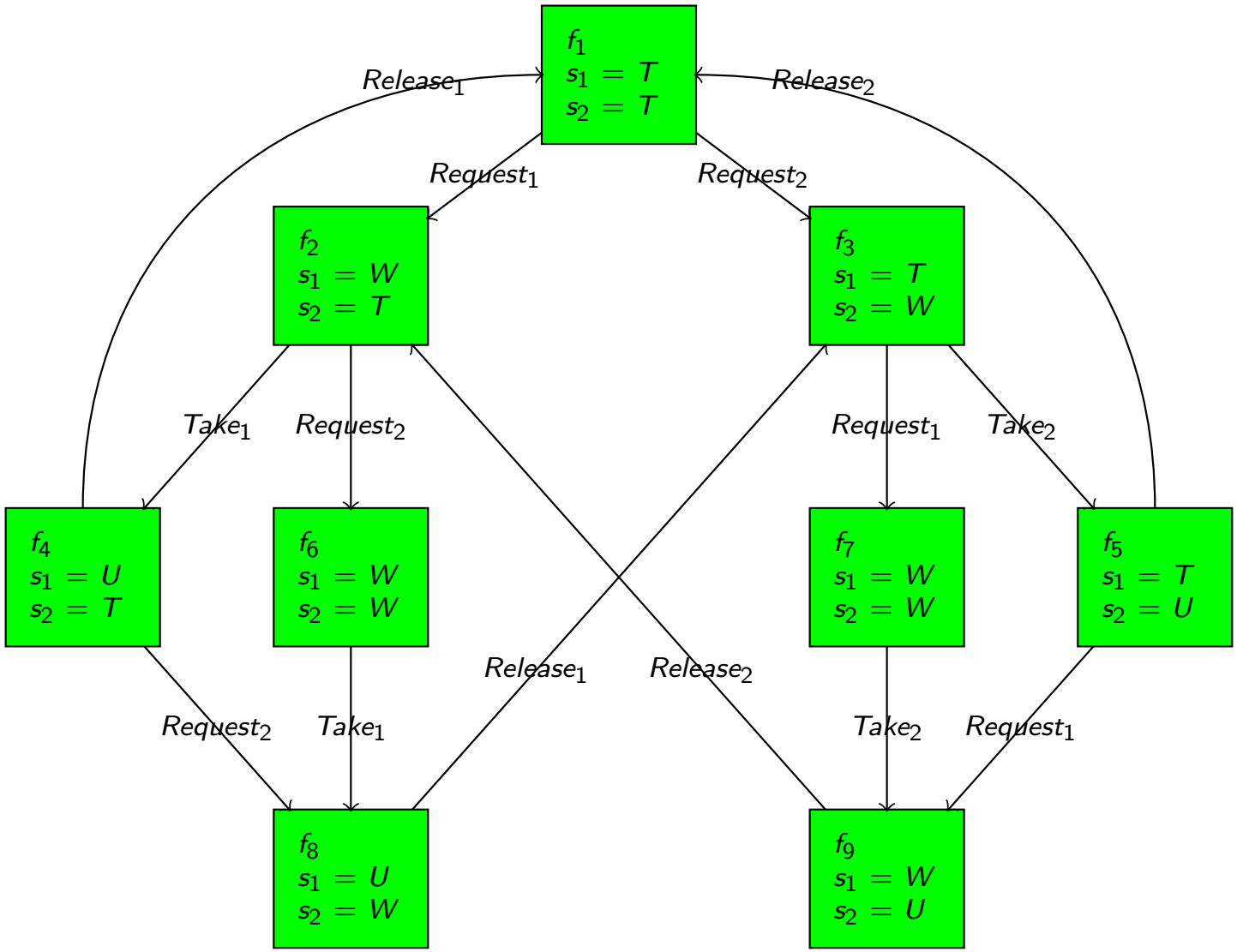
Verification of Property 2 for Example 3



Verification of Property 2 for Example 3



Verification of Property 2 for Example 3



Conclusions

- We have shown how distillation can be used to verify both safety and liveness properties of reactive systems.
- Our technique gives a finite state approximation of the original system in which all intermediate data is given an undefined value.
- Standard finite state model checking techniques can then be applied.
- Since more intermediate data structures are removed using distillation than other comparable techniques such as positive supercompilation and partial evaluation, more accurate results are obtained.

Related Work

- Verification of temporal properties using logic programs:
  - Leuschel & Massart, 1999
  - Roychoudhuri et al., 2000
  - Fioravanti et al., 2001
  - Pettorossi et al., 2009
  - Seki, 2011
- Verification of temporal properties using functional programs:
  - Supercompilation: Lisitsa & Nemytykh, 2007 & 2008
  - Higher Order Recursion Schemes (HORS): Kobayashi, 2009; Lester et al., 2010